2

OBJECTVISION™

REFERENCE

GUIDE

ObjectVision™

Reference Guide

BORLAND INTERNATIONAL, INC. 1800 GREEN HILLS ROAD
P.O. BOX 660001, SCOTTS VALLEY, CA 95067-0001

PRINTED IN THE USA.
10 9 8 7 6 5 4 3 2 1

R1

C O N T E N T S

Part 5 Appendixes and Glossary

T A B L E S

F I G U R E S

I N T R O D U C T I O N

This book is the second in a set of two manuals:

- *Getting Started* tells you how to install and begin using ObjectVision.
- This book, the *Reference Manual,* is your major source for information about ObjectVision. It contains detailed information about each feature of ObjectVision.

You should complete the ObjectVision tutorial in the *Getting Started* manual before you begin reading this manual. If you are an ObjectVision 1.0 user, you might want to read Appendix A, "Summary of 2.0 changes."

What's in this manual

This manual is divided into six parts:

- **Introduction** identifies parts of the ObjectVision window and explains how to get online help.

Part 1: ObjectVision basics

- **Chapter 1, "Using ObjectVision applications,"** explains how to begin entering data into ObjectVision applications and teaches beginners the ObjectVision terminology.

Part 2: Designing applications

- **Chapter 2, "Design preparation,"** helps you define your application's objectives and prepare for designing your forms.
- **Chapter 3, "Form design,"** introduces the Form Tool and the elements of a form with emphasis on creating your application's forms.
- **Chapter 4, "Tree basics,"** introduces the tree concept and explains the difference between value and event trees.
- **Chapter 5, "Creating trees,"** teaches you how to write trees and gives several examples of both types of trees.

- **Chapter 6, "Writing expressions,"** teaches you how to write expressions.

Part 3: Functions

- **Chapter 7, "@Function basics,"** outlines @functions and their uses in ObjectVision.
- **Chapter 8, "@Function descriptions,"** lists all the ObjectVision @functions with formats and examples for each.
- **Chapter 9, "Menu customization,"** shows how to customize your applications and their menus.

Part 4: Linking

- **Chapter 10, "Linking basics,"** explains the basics of linking and outlines the techniques for linking to an existing database.
- **Chapter 11, "Linking options,"** describes the linking options Locate, Filter, and VFields with several examples of each option.
- **Chapter 12, "Linking to Paradox tables,"** takes you through an example of linking to a Paradox table.
- **Chapter 13, "Linking to dBASE tables,"** guides you through linking to dBASE tables.
- **Chapter 14, "Linking to Btrieve tables,"** gives an example of a link to Btrieve.
- **Chapter 15, "Linking to ASCII files,"** summarizes linking to ASCII and walks you through an example.
- **Chapter 16, "Linking through DDE,"** outlines linking between ObjectVision and a DDE-compatible program.
- **Chapter 17, "Tips for designing database tables,"** gives you some helpful tips for designing your database tables while keeping ObjectVision in mind.

Part 5: Creating @functions

- **Chapter 18, "Using @REGISTER with DLLs,"** gives some examples of registering functions and using them in your ObjectVision applications.

Part 6: Appendixes and Glossary

- **Appendix A, "Summary of 2.0 changes,"** briefly describes the changes in version 2.0. Users of ObjectVision 1.0 should read this section first.
- **Appendix B, "Error messages,"** lists the error messages you might see and suggests ways to correct them.
- **Appendix C, "Application limitations,"** lists ObjectVision limitations that you should be aware of while you are designing applications.
- **Appendix D, "The ANSI character set,"** contains tables of ANSI characters that you can use in ObjectVision.

- **Appendix E, "Keyboard operations,"** lists the keyboard commands and shortcuts.
- **Appendix F, "Using the Runtime disk,"** describes ObjectVision Runtime and gives you tips on copying your applications.

Typography conventions

This manual uses two icons as follows:

This icon appears next to all new @functions listed in Chapter 8.

This icon appears next to event functions.

All other typefaces are listed in *Getting Started*.

What does ObjectVision do?

Since most users are familiar with paper forms, the transition to ObjectVision forms is an easy one. ObjectVision uses forms to gather, display, calculate, edit, and print values. Several forms can be stacked in an ObjectVision application just as paper forms might be stacked or stapled together.

Using trees

ObjectVision uses graphics to display calculations and actions called **trees**. The graphic trees are very similar to flow charts because they evaluate a condition, then, based on the result, perform the next branch or conclusion.

A field can have a value tree and an event tree at the same time.

ObjectVision can calculate a value for a field based on that field's *value tree*. A value tree is a set of branches, conditions, and conclusions that determine the value to give to the field. When a condition is met, the conclusion is evaluated, giving the field its new value.

ObjectVision has another type of tree called an *event tree*. Event trees are also a set of branches, conditions, and conclusions; the difference is that in event trees, the first condition is an event like *Click* or *Select*. Other branches and conditions in event trees work just like value trees.

Once an event occurs to the object or field, the event condition is met, and the actions in the conclusion are performed. Event trees

can assign values to other fields, open or close other forms or applications, or change the ObjectVision menus.

Guided completion

A very important feature is *guided completion*. ObjectVision will automatically guide you through a form, selecting only the fields that need values. ObjectVision will continue selecting fields until the form is complete. You can also customize completion by using event trees. Once a field is completed, you can program its event tree to select another field.

You can interrupt guided completion by selecting other fields with the mouse or by using *Tab* or *Shift + Tab*. You can edit any unprotected field at any time by interrupting guided completion. You can resume guided completion by selecting File | Resume. Guided completion becomes inactive once the form is complete.

Saving applications and data

ObjectVision forms are usually saved separately from the values they display. You can save a form with a single set of values in it, but saving each set of values this way uses a lot of disk space.

To optimize disk space and be able to share values with other applications, values are stored in database tables or ASCII files. Values can also be in a file created by a DDE-supported Windows application. Values from these sources are delivered to ObjectVision by *links*.

Linking to data sources

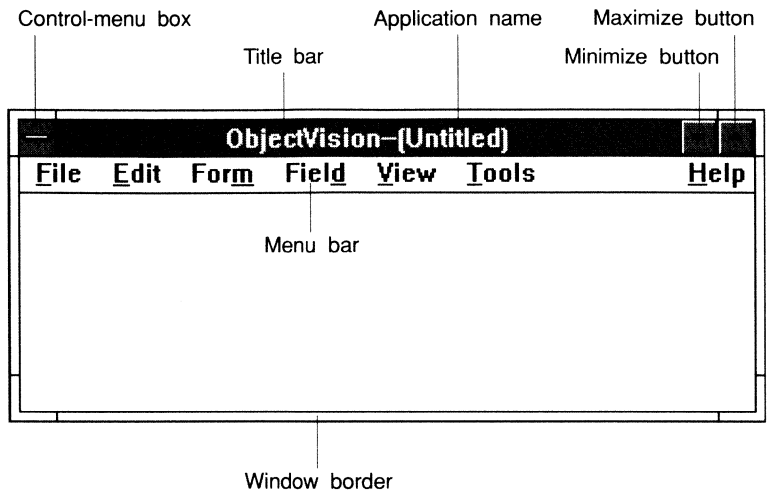
ObjectVision can link to a comma-delimited ASCII file or to Paradox, Btrieve, and dBASE-compatible database tables. ObjectVision can link to an existing database table, or it can create any supported database table.

ObjectVision can use DDE links to share information with other Windows applications. With DDE links, two or more ObjectVision applications can share information.

The ObjectVision window

The following figure illustrates the ObjectVision application window. The paragraphs following the figure describe each part of the ObjectVision window.

ObjectVision application window



Control-menu box The Control-menu box, in the upper left corner of the window, lets you move and size the window, close the window, or switch to another application.

Title bar The title bar shows the name of the application, ObjectVision, and the name of your open file. The title bar displays `(Untitled)` after the application name when no file is open or the file you're creating hasn't been saved.

Menu bar The menu bar contains ObjectVision's menu names. When you click a menu name, a list of that menu's commands displays. A command is an action ObjectVision performs, such as Open or Exit.

Maximize, Minimize, and Restore buttons The Maximize and Minimize buttons are in the upper right corner of the application window. The Maximize button enlarges the window to fill the entire screen, and the Minimize button reduces the window to an icon.

If the window is maximized, the Restore button replaces the Maximize button. Restore simply restores the window to its previous size and position.

Window border

You can change the size and shape of most windows on your desktop. Using a mouse, you can drag either a side or a corner of the window to change its size. For more help on sizing windows, consult your Windows manuals.

Getting help

There are three types of help: general, context-sensitive, and object-specific.

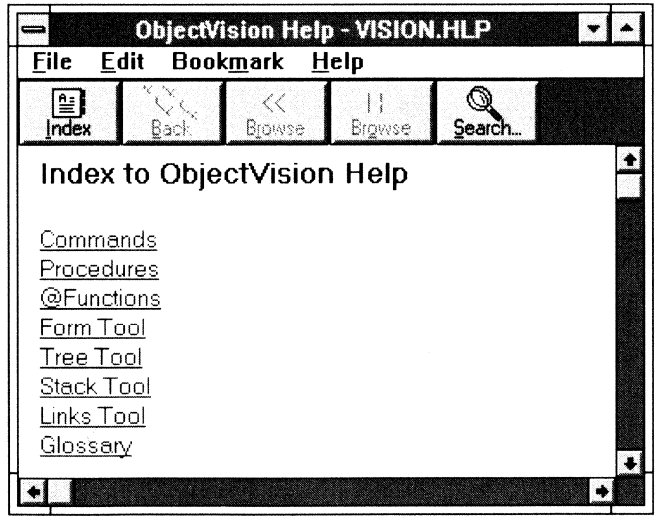
General help

You can get general ObjectVision help by choosing Help from the right corner of the menu bar. A brief list of topics appears. If you don't know the specific topic, select Index.

After you select Help | Index, you can choose a topic from the list by positioning the pointer on the topic (the pointer changes from an arrow to a hand), and then pressing the left mouse button.

Any underlined words you see while you're reading a topic's information indicate that there's more information available for that word. To read the extra information, click the underlined word.

ObjectVision Windows Help Index



In the help figure there are five buttons. From left to right they are

- Index, which returns you to the index of topics.
- Back, which returns you to the topic you previously viewed.
- The two Browse buttons, which move you forward and backward in a series of related topics.
- Search, which searches the help files for specific information you need.

If you need more information with the Windows Help system, select Help from the menu. To exit from ObjectVision Windows Help, double-click the control menu box.

Context-sensitive help

If you need help with an ObjectVision feature you are trying to use, press *F1*. ObjectVision searches the general help topics and gives you help *sensitive* (related) to that ObjectVision feature.

Pressing *F1* is a faster way of getting the help information you need, but you can always get the same information by selecting Help from the ObjectVision menu.

For example, if you want help on Properties (a menu command) you can click on it then press *F1*. The help file will return help for the first item under Properties. To see the help screens for any following item, click Browse forward.

If you need specific help for Properties | Form | Event Tree, click Properties | Form then use the arrow keys to highlight Event Tree. After it's highlighted, press *F1*.

Object-specific help

When you design a form, you can create custom help messages. Messages can be attached as properties to fields, columns, and buttons. For more information on creating help for your applications, see Chapter 3, "Form design."

You can't select a button with the mouse because the mouse causes the Click event for that button, which performs the button's event tree conclusion.

If you are using an application created by someone else, you may find that a message unique to the application appears when you press *F1*. This object help typically relates to the object you have selected. To select a button object for its help you must use *Tab*.

PART 1

ObjectVision basics

C H A P T E R

1

Using ObjectVision applications

This chapter explains how to use ObjectVision applications and helps you understand ObjectVision terms.

Terms to know

For a longer list of terms, see the Glossary.

Before you proceed you should have an understanding of the following terms.

- **Application** An application is an .OVD file you create using ObjectVision.
- **Stack** A stack is a set of forms. You can think of a stack as you would think of a set of stapled forms. An application has one stack that holds one or more forms.
- **Form** A form is what ObjectVision uses to hold objects. A form interface is used because most people are familiar with paper forms. ObjectVision forms can be thought of as electronic versions of paper forms that do much more than paper forms.
- **Object** An object is something you place on a form. Objects can be fields or tables containing values, graphics, text, or shapes that affect the appearance of the form.
- **Button** A button is an object that can be clicked, selected, or unselected. Usually an event tree is attached to

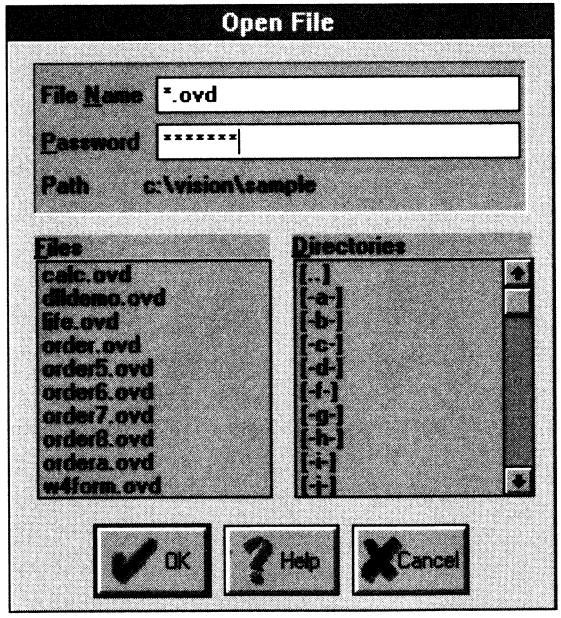
a button so that when it's clicked, an action is performed.

- **Field** — A field is where values are displayed. There are single fields that contain only one value at a time, and there are column fields that contain many values displayed as a column of related values.
- **Table** — A table is where columns of related values are displayed at the same time.

Opening an application

To use ObjectVision applications, first start ObjectVision, then choose File | Open. The File | Open dialog box that lists ObjectVision applications appears as shown in Figure 1.1.

Figure 1.1
File | Open dialog box



To select a file to open, use one of these methods:

- Use the mouse and scroll bars to highlight an application in the Files box, and then either double-click the file name, press *Enter*, or click OK.
- Type in the directory and the name of the ObjectVision application in the File Name box, then press *Enter*.

- Click inside the Files box, type the first letter of the application name until that application is highlighted, then press *Enter*.
- Use the *Tab* and arrow keys to highlight an application in the Files box, then press *Enter*.
- Press the *PgDn*, *PgUp*, and arrow keys to highlight an application in the Files box, then press *Enter*.

Finding an application

The File | Open dialog box lists files in the current directory that have the .OVD extension. Files with the .OVD extension are assumed to be ObjectVision application files.

If the file you want is in a different directory or drive or has a different file extension, you can find it by typing the drive, directory, or extension in the File Name box. For example, if someone gave you an application called CUSTOMER.BLD on disk, you would put the disk in the drive and type `a:\customer.bld` in the File Name box, then press *Enter* or click OK.

*Wildcard characters such as ? and * are also acceptable.*

Once you open an application, the directory it's in becomes the *working directory*. ObjectVision uses the working directory as the default location for graphics, database tables, and any saves you do of that application.

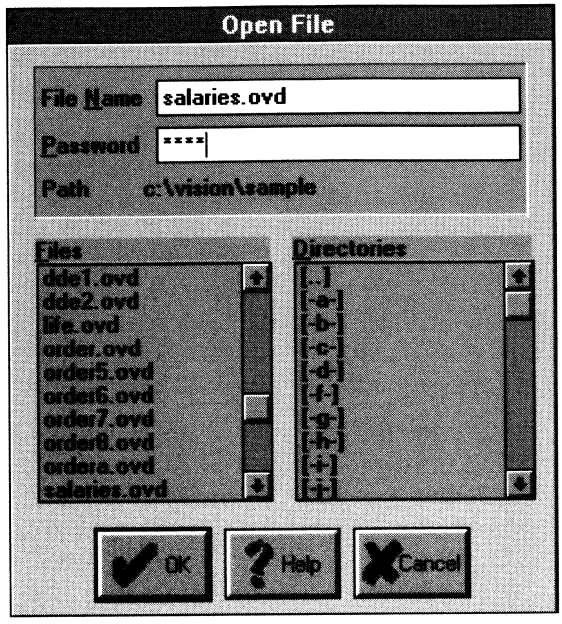
Password protection

If you open an application that has been saved with a password but you don't know the password, you won't be able to edit the design of the application because the entire Tools menu is removed from the menu bar.

Without the password you *can* use the application just as you would use a run-time version. You can enter values or retrieve values—you just can't make changes to the design of the application.

If you do know the password and want to make changes to the application's design, select File | Open, type or highlight the application name, then type in the password before you click OK. In Figure 1.2 the password is only four characters long. Notice that the characters you type are not displayed. Instead you see a set of four asterisks.

Figure 1.2
Opening with a password



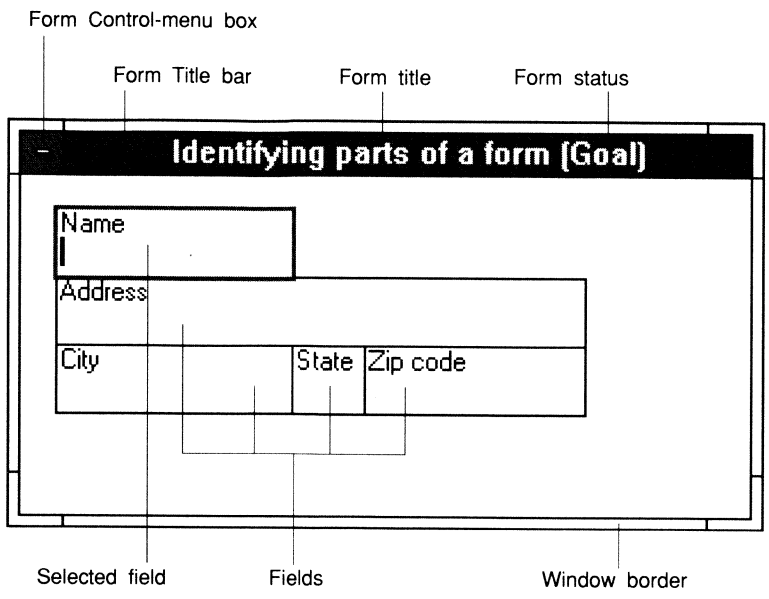
Don't confuse ObjectVision passwords with passwords for database tables.

Passwords can be any set of keyboard characters up to 16 characters. Once you're finished designing your application, you can save it with a password by typing in a password in the File | Save As box. If you don't want a password, type only a file name, then press OK or *Enter*.

What makes a form

The form title bar displays the form title and its status. Figure 1.3 illustrates the parts of a typical form.

Figure 1.3
Parts of a typical form



The form status appears enclosed in parentheses following the form title and indicates one of the following values:

- *Goal*: The top form in a stack is automatically selected as the Goal form. When another form is selected, that form temporarily becomes the Goal.
- *Complete*: When all fields on the form have a value, the form is complete. You may still change the value of any unprotected field.
- *Prompt*: When a field on the Goal form has a tree that needs a value from a field, guided completion will go to that field for the value. If that field is on another form, that form becomes active and displays the prompt status. If that field is not on any form, a Scratchpad form automatically appears requesting the value. In both cases, the prompt form disappears after the required values are typed.

When you display more than one form on the workspace, only one form can be the *active form* that receives input. The title bar of the active form is always highlighted with a different color or intensity.

You can select another form in the stack by clicking any visible part of the form or by selecting the form by using Form | Select.

If the form is smaller than the application window, you can move the form to a new location by dragging the title bar or by choosing the Move command from the Control menu.

If the form is taller or wider than the application window, the form will appear with scroll bars. If a form has scroll bars, you won't be able to move the form.

Fields

In Figure 1.3, the selected field is Name.

Once you open an application, a field requiring a value becomes the *selected field* and is highlighted. You either select a value (when the field is a selection method) or you type information. See the "Guided completion" section later in this chapter.

A *calculated* field, when selected, displays a heavy outline with a dashed line inside. A calculated field gets its value from a linked database, a DDE link, an ASCII file, or an event or value tree. A calculated field can be protected so its value cannot be overwritten.

Control-menu box

The Control-menu box to the left of the form title is similar to that of the ObjectVision title bar, but has its own shortcut. To open a form Control menu, either click the Control-menu box or press *Alt+-* (hyphen).

The application window's Control-menu box is a long bar (*Alt+Spacebar*) and a form window's Control-menu box is a short bar (*Alt+-* (hyphen)).

You can double-click a form Control-menu to close a form. Closing a form doesn't remove it from the application. It only closes it from the display. You can reopen a form by using Form | Select, then choosing a form from the list.

Form border

The form border is usually a thin line surrounding the form. You can position a form anywhere inside the ObjectVision border.

A form border with scroll bars displays only around forms larger than the application window. In this case, you won't see the whole form at one time. Use the scroll bars to reposition what you see and to fill out other areas of the larger forms.

Scratchpad forms

If you create a field but don't place it on a form, it will appear on a Scratchpad form only if the field's value is needed for a tree or if you select Field | Find.

A Scratchpad form is simply a temporary form that prompts you for a field's value. After you type a value and press *Enter*, the Scratchpad form closes.

Guided completion

Generally, guided completion selects fields from top to bottom and from left to right.

Guided completion is a unique ObjectVision feature that makes it easy and efficient for you to fill in forms. ObjectVision automatically determines the field order and guides you to each field requiring a value. When you open a form, ObjectVision selects the first field. Once you type a value and press *Enter*, ObjectVision selects the next field for you.

Calculated fields and fields linked to external sources that have received values already are omitted from guided completion, so those fields will not be selected for you.

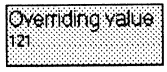
When you design a form, you can alter guided completion by using @functions like @ASSIGN and @FIELDFIND. If you're using a form that someone else has designed, don't be surprised if you're prompted to a field on another form. For more information on customizing completion see page 31.

You can interrupt the field sequence at any time by selecting any form or field manually. You can interrupt guided completion by using any of these four methods:

1. Clicking a field with the mouse.
2. Pressing the *Tab* key to move forward on the form or *Shift + Tab* to move backwards.
3. Choosing Field | Find from the menu bar.
4. Choosing Form | Select.

If you interrupt the field sequence, perhaps to make a correction to a previously entered value, you can return to guided completion by choosing File | Resume.

Overriding values



An *override* occurs when you change the information in a calculated field. ObjectVision displays a dot pattern to indicate that a field contains an *overriding* value.

If you want an overridden field to be returned to its calculated value, select the field, then choose Field | Calculate. The calculated value will reappear.

Protected fields

Protection is a field property. See Chapter 2 for more information on properties.

Some fields may be protected. This means that you can't override the value in the field or view the field's tree, depending on how the application designer protected the field.

A field protected from a value override has a dashed line inside a heavy solid line when the field is selected. A field protected from viewing its tree doesn't change its appearance.

If the field is protected from overrides, you won't get a cursor in the field when you select it. You can override values in protected fields by using @DDEPOKE. If you don't want to have values changed through DDE, check Ignore Remote Requests. See Chapter 16, "Linking through DDE."

Selecting fields

If you want to stop using guided completion so you can choose which fields to fill in, you can click any field or select it by pressing *Tab*. This is useful if you want to see the result of entering different values for a calculated field.

For example, if a field called Discount has a value tree that needs a value from a field called Quantity, you could type in a value for Quantity, press *Enter* or *Tab*, then see what displays in Discount. You could select Quantity again, retype a value, press *Enter* or *Tab*, then see how the value in Discount changes.

If you are working with large forms or large stacks of forms, you can use Field | Find to select another field. The Find dialog box alphabetically lists all field names in the current application. After you select a field, the form containing that field appears, and that field is selected.

You can also use Field | Find to locate fields on Scratchpad forms. If more than one form contains the selected field, the form closest to the top of the stack appears. If the field you select is not contained on any form, the Scratchpad form displays the field.

If you're using the Form Tool, you can search for a field by using Form | Find. This works the same way as Field | Find when you're in form completion mode.

Selecting forms

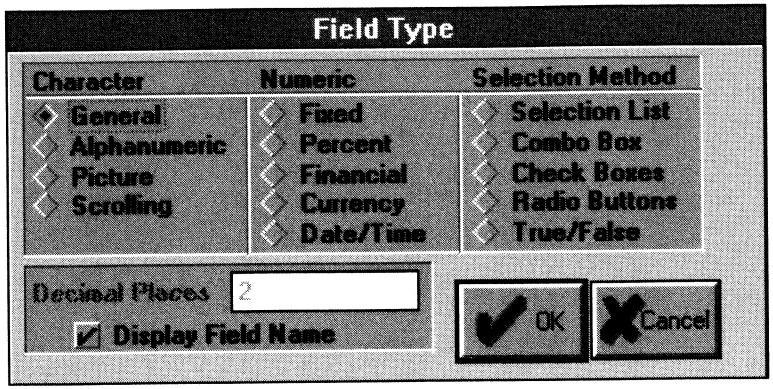
To select another form, choose Form | Select. The Form | Select dialog box lists form names for the current application according to their order in the application's stack of forms. Once a form is selected, it appears in front of any other visible forms and moves to the top of the form stack.

Understanding field types

There are several different types of ObjectVision fields. The type of values you enter (numbers or letters) or select (with selection methods) depends on the field type. This list briefly describes each type. A more detailed explanation follows.

- **Character:** Can be any keyboard character, but they may be restricted to only letters.
- **Numeric:** Any format of numbers such as percentages, currency, or dates. Scientific notation is not supported. There can be restrictions on decimal places or date formats such as Day/Month/Year.
- **Selection Method:** You can select a value from a list. There are several ways to display the list.

Figure 1.4
Field types



Character types

There are four character types: General, Alphanumeric, Picture, and Scrolling.

- **General** is the default value for field types. This type lets you type any keyboard character. After a value is entered into a field of this type, ObjectVision converts it to the appropriate type. For example, if you type .345 in a General field and press *Enter* or *Tab*, ObjectVision converts it and places a zero in front (0.345).
- **Alphanumeric** accepts any keyboard character. Unlike the General field type, it does not try to convert the value.

A picture for a social security number might look like ###-##-####.

- **Picture** accepts only specific characters. The following symbols represent what you can type when you define a picture field:

#	numbers only
?	letters only (lower- and uppercase)
&	letters only (converts to uppercase)
@	any character
!	any character (converts to uppercase)

Any other character, such as hyphens in a social security number, are taken literally. You do not have to retype these characters.

When typing in picture fields, you won't see the above characters—instead you'll see an underscore for each character. Once you begin typing, the characters you type will replace the underscores. You also have to complete the picture string; otherwise you'll see the message "Field value is incomplete."

- **Scrolling** holds more information than what the physical field on the form will allow. To read any information that can't be

displayed at one time, use the scroll bars attached to the right of the field. This field will accept any keyboard character.

When you create a scrolling field, be sure to check the limits of the database you connect it to.

Numeric types

All of the numeric types require numbers for values, but they can be displayed in different formats.

- **Fixed** displays the number you type with a fixed number of decimal places. If you enter more decimal places than necessary, the number will be rounded. For example, if you've set the number of decimal places to 5, and then type 1.259839, the number displayed will be 1.25984.
- **Percent** displays the number you type as a percentage. It can have up to fifteen decimal places, but the default is 2. For example, if you type 0.12, it will be displayed as 12.00%.
- **Financial** separates thousands with the whole number separator (the commas in 3,234,578) and shows negative numbers in parentheses.
- **Currency** displays the number you type as currency with a currency symbol before the number and the appropriate currency notation ($14.95). The currency settings can be changed by using the International icon in the Windows Control Panel. Negative numbers are displayed in parentheses.
- **Date/time** assumes the number you type is a date or time. The formats will display differently depending on how you've set up the Windows Control Panel on your computer. However, ObjectVision will convert the displayed format to the format that was selected when the form was first designed, if they are different.

Selection Method types

There are five types of selection methods. Basically, a selection method gives you a choice of one or more values.

- **Selection List** contains a list of values that automatically appears when the field is selected, and you *must* select one of the available values. You cannot type a new value.

- **ComboBox** contains a list of values. To select one, click the down arrow to the right side of the field, then highlight one of the values. You can also type in a new value.
- **Check boxes** display a list of values with boxes next to them. To select a value, click its box. Only one value can be selected at a time.
- **Radio Buttons** work like check boxes except circles are displayed instead of boxes. Only one value can be selected at a time.
- **True/false** looks like a check box, but its only choices are True (checked) or False (unchecked).

Displaying field names

The Field Type dialog box in Figure 1.4 contains a check box for displaying a field's name. The default is on (checked). If you uncheck Display Field Name, the field's name will be hidden.

Clearing forms

When you use an ObjectVision application, you may need to clear a form or a form set (remove all field values) so that you can enter new values into a blank form. Two ObjectVision commands clear forms:

- Edit | Clear All clears all the values on every form in the application. This command also removes data from all fields on Scratchpad forms.
- Form | Clear clears only the values on the active form.

Calculated values whose trees use constant values are not cleared (such as a fields with @NOW as its value tree). DDE-linked values are also not cleared.

Clearing forms with external links

A Clear button is often included on forms that link to external data, so that only values associated with the link can be cleared from the form by pressing the Clear button.

For more information on links, see Chapter 10, "Linking basics."

Entering and editing values

Once you open an application and ObjectVision selects a field, you simply select or type the value and press *Enter* or *Tab*. After pressing *Enter*, guided completion selects the next field, which can be on the active form or on a different form in the application.

If you press *Tab* instead of *Enter*, the next field on the active form is selected whether it requires a value or not. The next field is determined only by the physical layout of the fields, going from left to right and from top to bottom. See also page 31 for a figure that demonstrates field selection.

Editing field values

You can use the keyboard or the mouse to edit values you have typed or entered while filling in your forms. The following paragraphs describe text editing methods.

Positioning the pointer

With a mouse, you can click anywhere in a field. When you move the pointer into a field, the pointer changes to an I-beam. To select the field value, click and drag the I-beam across the characters. If you double-click on a character in a word, the entire word is selected. If you double-click on a symbol or a number, all the characters between the two surrounding blank spaces are selected (for example, if you double-click the 3 in the string "August 31 is the last...", then 31 is selected as the word).

You can also press *Home* or *End* to position the text pointer in a field. Pressing *Home* moves the pointer to the left of the first character in the field. Pressing *End* moves the pointer to the right of the last character in the field.

Removing text

You don't need to press *Del* to remove *selected* text. The next characters you type automatically replace the selected text.

You can also press *Backspace* to erase characters to the left of the pointer. Pressing *Del* removes characters to the right of the pointer.

If you type a new value, but do not press *Enter*, you can press *Esc* to remove the new value and restore the field to its previous value. If you type a new value and then press *Enter* or *Tab*, you can choose Edit | Undo or press *Alt+Backspace* to restore the previous value.

To remove all values from a form, choose Form | Clear. Choose Edit | Clear All to remove all values from an *entire* application. Most forms include a Clear button that clears only the linked values.

Cutting, copying, and pasting values

Besides typing values into selected fields, you can paste values you have cut or copied into those fields. Values you cut or copy to the Clipboard remain there until you change it, clear it, or exit Windows. The *Clipboard* is a temporary storage area for information you want to paste into fields or into other applications.

For example, you can copy text from a word processor and paste it in an ObjectVision conclusion, text object, or help text box.

Remember, the Clipboard holds only one piece of information at a time. Each time you cut or copy another piece of information to the Clipboard, the previous item is permanently removed.

The paragraphs that follow describe the ObjectVision Edit | Copy, Edit | Cut, and Edit | Paste commands.

Shift Del
is a shortcut for Edit | Cut.

Edit | Cut deletes the selected text from its field and transfers it to the Clipboard. You can choose this command to remove text and paste it into another field or application.

Ctrl Ins
is a shortcut for Edit | Copy command.

Edit | Copy transfers a copy of the selected text to the Clipboard without deleting the selected text. You can choose this command to copy text for pasting into another field or another application. If you accidentally copy unwanted text, you can remove the copied text from the Clipboard and restore its previous contents by choosing Edit | Undo.

Shift Ins
is a shortcut Edit | Paste.

Edit | Paste inserts a copy of the Clipboard contents at the pointer location. You can choose this command to paste text from another field or another application. You can remove the text you inserted by choosing Edit | Undo.

When you paste data from the Clipboard, the contents of the Clipboard remain unchanged so you can paste the same information into as many places as you want.

Using table objects

When you're using a table object, you can position on values in several ways. You can use the table's scroll bar if it has one, the keyboard function keys, buttons that were included with the application, or the *Tab* and *Enter* keys.

Using table scroll bars

The scroll box in the scroll bar on a table will always appear in the middle. This happens because the table object doesn't know how many values are in the linked table (the database table size can always change). If the ObjectVision table object isn't linked to a database table, the scroll bars won't work.

If you use buttons or function keys, the row pointer moves up and down the table. With the scroll bars, the row pointer stays and the table values move.

If you use the arrows of the scroll bars, you'll scroll through the linked database table one record at a time, just as if you pressed a Next or a Previous button. The row pointer doesn't change positions when you use the scroll bars. For example, if a table has three rows and the row pointer is next to the second row when you click the up scroll arrow, the value in row one moves down to row two (the current row), the value in row two moves to row three, and a new value appears in row one. The values move in the table object while the row pointer keeps its position.

Clicking in the areas between the scroll box and the arrows causes a page up or a page down. The row pointer doesn't change its position.

Using function keys

Instead of using scroll bars to page through records on a linked database table, you can use function keys.

Table 1.1
Function key descriptions

Function key	Action
F3	View previous record (if there is one)
F4	View next record (if there is one)
F5	View the previous records (number of rows – 1)
F6	View the next records (number of rows –1)
F7	View first record
F8	Clear all records from the table object

Using button with tables

Sometimes buttons are included in an application. You can use these buttons to position in a linked database table. When you create a link, ObjectVision will ask if you want to create automatic buttons (See Chapter 10, "Linking basics," for more information.) You can create the following automatic buttons:

- TOP moves to the first record in the database table.
- BOTTOM moves to the last record in the database table.
- PREVIOUS moves to the previous record in the database table.
- NEXT moves to the next record in the database table.
- PAGEUP goes to the previous page in the table (a page is defined as the number of columns in the ObjectVision table minus one).
- PAGEDN goes to the next page in the table (a page is defined as the number of columns in the ObjectVision table minus one).

Printing forms

You can print forms with the current information contained in an application. To print only the active form, choose File | Print Form.

To print all the forms in the application, choose File | Print All. Each form in the application prints on a separate page.

You can also use File | Print Link to print the form multiple times, once for each value in the link. The number of forms printed can be limited by a restricted range. (See Chapter 11, "Linking options," for more information on links and restricted ranges.)

If a form you want to print contains an ObjectVision table connected to a database table, the form will be printed multiple times until every value of the database table has been printed.

For example, if you create a table that displays 5 values at a time, then connect it to a database table with 100 records, 20 forms will be printed.

All of these menu commands have corresponding @functions.

- @PRINT(*"FormName"*) for File | Print Form
- @PRINTALL for File | PrintAll
- @PRINTLINK(*"LinkName"*) for File | Print Link

You can use these functions to print forms automatically. See Chapter 8, "@Function descriptions."

Fonts

ObjectVision uses Windows font conventions for both screen display and printing. For best printing results, make sure the appropriate printer fonts are installed for Windows. Because of differences among printers, the exact size and appearance of printed forms might vary depending on the printer you use. For the closest font match onscreen, select View | Printer.

If the font you select doesn't print correctly, it's because Windows can't match the font you selected, but it will print the closest match. If you're using a font manager other than Windows, check that it's properly installed.

To see a list of printer fonts (fonts your printer will accept), choose Properties | Object | Label Font or Value Font from the Form Tool.

When you design a form, you should keep in mind what fonts your users will have for printing.

Printer setup

ObjectVision prints to the currently selected printer in Windows. The printer you print to must be active.

To select a printer, choose File | Printer Setup. A list of the available printers will appear. Select a printer, then click OK.

To set up a printer, click the Setup button in the Printer Setup dialog box. ObjectVision brings up a dialog box unique to your printer driver. Use the buttons and menus to set up your printer, or consult your Windows manual for information on printer setup.

Viewing value trees

When you're using an application and you want to see the value tree of a field, select the field, then choose Field | Show Tree. You can also select the field, then double-click in the field, and the tree will display.

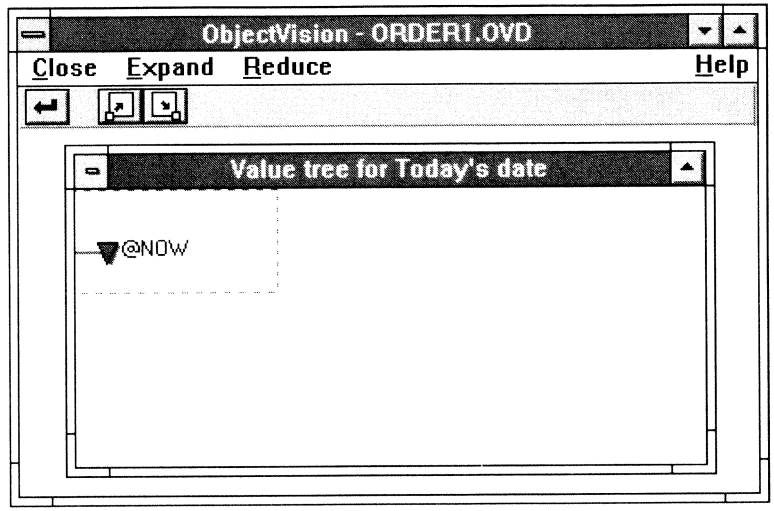
Note that you can only view value trees, not event trees.

If the field doesn't have a value tree, Show Tree will be dimmed and unselectable. It will also be unselectable if the field has a value tree that's protected.

The tree object bar

Once you've selected Field | Show Tree, a window displays with an Object bar and a value tree.

Figure 1.5
A value tree



The three buttons on the Object bar correspond to commands on the menu bar. From left to right these buttons are

- **Close**, which closes the window and returns you to the form.
- **Expand**, which magnifies the tree.
- **Reduce**, which shrinks the tree so that more will fit in the window.

If the tree you're viewing is large, you can also use the scroll bars or arrow keys to move the tree and view other nodes of it.

C H A P T E R

2

Design preparation

This chapter helps you define your application's objectives and prepare to design your forms. The following chapter shows you how to design your application. Chapters 4 and 5 discuss trees. Chapters 6 through 9 describe expressions and @functions. Chapters 10 through 16 discuss linking.

Defining your objectives

Before you begin designing your application, you should define its objectives. You should also develop a theme or central idea so that your users will remember how to use your application.

For instance, you could design an application that lets your users get customer profiles from one database table and save customer orders to another database table. In this example you could protect the customer profile so users could not alter the information, but they could still enter the orders customers make.

The following questions may help you to define your application objectives:

- What do my users need? Do they need to look at or update information in a database table? Do they need to enter values into a table?
- What path do I want my users to follow?

- How much control over values am I giving my users? Do they need full control over values to be able to change and save those values in a database table?
- Will I need to link to an existing database table, or will I need to create my own database tables, or both?
- How many forms will I need for this application?

Defining form objects

The Object bar and its use is described in detail in the next chapter.

Once you know what you want your application to do, list the objects you'll need to put on your forms.

The ObjectVision objects are described below with a graphic of the buttons from the Object bar to the left.

Field is a space for values. A field can get its value from user input, a link, or a tree. If an error occurs while giving a field its value, it will be assigned ERR or NA. If the field's value is longer than the field itself, a string of # symbols appears instead of the value—although the value is still there.

Button is an object that can be clicked, selected, or unselected. It is most useful when used with an event tree whose condition is Click—when the button is clicked, the event produces an action.

Table is a set of columns that act as one object. Columns are like fields except they *display multiple values* while still only having *one current value* (the current value is used with trees or links).

Text is an object where you can write instructions, or you can use this space for titles. A user cannot change this text without using the Form Tool.

Filled rectangle is used to add flare to your forms. A filled rectangle is good for creating shadows behind other objects. It can also add texture to a form.

Rounded rectangle is simply a variation of a filled rectangle.

Line is used to draw simple lines on a form.

Graphic is used to import bitmaps and metafiles from the Clipboard or to place (on the form) .OVG files you've already created.

Outlining the user path

It's important to outline the path you want your users to take. If you want them to type a name first, then the field *Name* should be your first field, and so on. After you have listed all of the objects your application needs, separate those that the user will fill in or modify from those that can be calculated or retrieved from a database table.

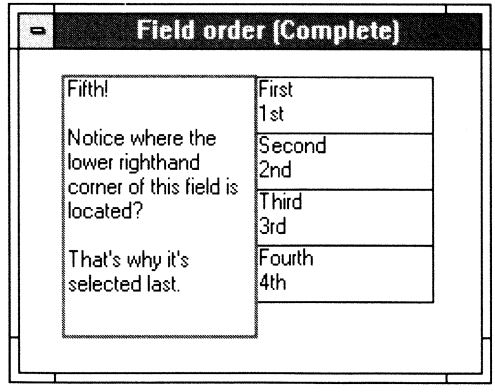
Guided completion

Use guided completion to your advantage. Guided completion selects only fields that require values from the user. If guided completion comes to a calculated field, it'll prompt for any values that field needs to complete its tree.

Fields reading values through links are ignored *if* they have already received values from the link. You might still want to organize the fields for aesthetic reasons.

Fields for guided completion are selected from top to bottom, left to right, and are based on the position of the bottom right corner of the field. Figure 2.1 illustrates how a long field in the upper left is read last because its bottom right corner is further down than the others.

Figure 2.1
Field order



Changing guided completion

If you want to change the order of guided completion, you can use @functions with event trees to select fields or open forms. The following @functions can be used to customize guided completion for your applications:

- @EVENT
- @FIELDSELECT
- @FORMSELECT
- @FORMCLOSE
- @RESUME

Writing help for your users

Keep your help specific to the object; you don't want to confuse your users.

There are two ways you can aid users of your application. You can add text objects to your forms that explain what you need the user to do. You can also add help to certain objects.

Objects that accept help screens are fields, columns, and buttons. For these objects, select Help from the object's properties. A dialog box appears where you can type any instructions for your users.

Setting up ObjectVision for design

The following sections describe the various things you should consider when you are designing your forms.

Setting object defaults

You can set each object's properties so that every object you place on a form has the set values. These defaults are saved so that every time you start ObjectVision the defaults are ready.

For instance, if your company's colors are blue and white you could set the field properties so that each field's background color is blue and the field name is white.

Properties from Value tree to Help cannot be set as defaults.

To set the defaults for an object, right-click on the object's button on the Object bar. A list of that object's properties appears. Select the property you want to set. Figure 2.2 shows each object and its properties. The following section explains each property.

Figure 2.2
Objects and properties

Property \ Object		Field	Column	Cell	Button	Table	Text	Rectangle	Rounded rectangle	Line	Graphic
Field type		✓	✓								
Alignment		✓	✓				✓				
Label font		✓	✓			✓	✓				
Value font		✓		✓							
Color	Label	✓	✓			✓	✓				
	Value	✓		✓							
	Back-ground	✓	✓	✓		✓	✓	✓	✓		
	Border	✓				✓	✓	✓	✓	✓	✓
Borders		✓					✓	✓	✓		✓
Line width		✓				✓	✓	✓	✓	✓	✓
Fill pattern								✓	✓		
Protection		✓	✓								
Value tree		✓	✓								
Event tree		✓	✓		✓		✓				✓
Field		✓			✓						
Name/Text		✓	✓		✓	✓	✓				
Help		✓	✓		✓						
ScrollBar						✓					

Field Type This property lets you set the default field type so that every field you create has *that* field type.

Alignment This property aligns the values or text. You can choose to align the values in the object to the left, right, center, or justified. Justified aligns text to both the left and the right.

Label font This property selects the font and size of the object's label.

Value font This property selects the font and size of the object's value—what the user types in.

Color Color selects the colors for the object. *Label* refers to the label, or name, of the object. *Value* is what the user types. *Background* is the color for the interior of the object. *Border* is the color for the border.

Borders You can place a border around the entire object by checking Outline. If you only want an underline, or a border on one side, you can check only that side. Your choices are Left, Right, Top, and Bottom.

Line width This property selects the width of the border lines. If you select a line width but don't select a border, this property will have no effect.

Fill Pattern You can choose one of 15 patterns to add texture or shading to objects. If you choose solid black, it will cover any color you add to the object.

If the fill pattern is hashed (made of patterns of lines) the fill pattern will take the color you give the object, leaving the background white. All other fill patterns will be shades of gray on a colored background.

Protection You can choose to protect a field in two ways. You can prohibit users from changing a field's value, or you can prohibit them from viewing the field's value tree.

Value tree This is a special property type explained in Chapter 4, "Tree basics."

Event tree This is another special property type explained in Chapter 4, "Tree basics."

Field This property allows you to add a new field. You can copy a field multiple times (so you'll get that field's properties copied as well), then you select a copied field, select Field from its property list, and then type the name you want to give it. The copied field then takes the new field name and you have two fields with the same properties, but different names.

Name/Text This property lets you edit or change the name of the object. You can edit a field name or edit the text in a text object.

Help You use this property to add help to an object. Users of your application can then select the object, press *F1*, and see your help message.

ScrollBar This is a property for tables only. When it is on, the scroll bar is displayed; when it is off, the scroll bar is not displayed. The ObjectVision default is off.

View

You can select View from the menu to change the way you view ObjectVision forms. You can also select Rulers and Grids—two options that help you position objects on your forms.

Printer and Screen

View | Printer changes the screen display so that it represents what will be printed. If you have Rulers on and you create an object that is one inch by one inch, that will be the size of the object when the form is printed.

If you select Screen, ObjectVision will display screen characters, which may be smaller, instead of print characters.

Setting Ruler

When you select View | Ruler, you can choose where you want to place rulers on your forms. They can go along the top border of your form, along the left side, or both. The rulers can be set to inches, centimeters, or characters. Characters display one average character width along the top, and one average character height along the left.

Rulers don't appear in the application window.

Once you set rulers on, they will appear on every *form* that you use in the Form Tool. Rulers don't appear in form completion mode.

Using Grid

While using the Form Tool, you can choose View | Grid to select an underlying grid—Coarse, Medium, or Fine. Coarse uses a one-character by one-line grid pattern, Medium a one-half-character by one-half line, and Fine a one-fourth-character by one-fourth line grid.

ObjectVision constrains all form objects to a grid. When you create an object and attempt to place it on a form, the grid will only allow that object to be placed on a grid line and not on any space in between.

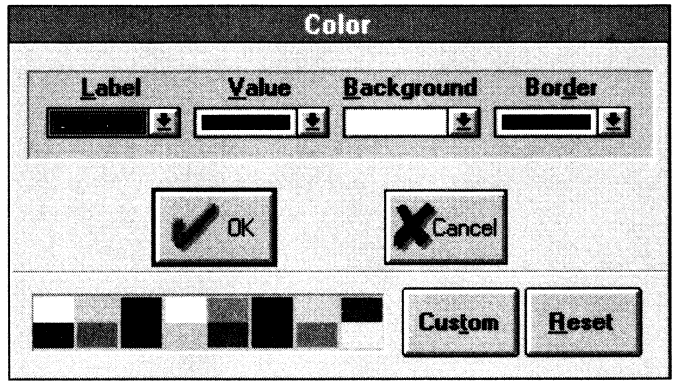
Color

You can add colors to objects as an object property. There are 16 colors available, but you can customize these colors and use them throughout your application.

Selecting colors for an object

You select colors from a list. Each color in the list corresponds to a color in the custom selection. If you haven't customized any colors, the 16 colors displayed are the default colors.

Figure 2.3
Color dialog box



Each object will accept certain color types. Fields will accept all four color types, whereas lines only accept a border color.

1. To give an object color, right-click the object then choose Color from the list of properties. You'll see the Color dialog box in Figure 2.2.
2. There are combo-box colors for Label, Value, Background, and Border. To select a color, click the arrow next to the color type you want.
3. You'll see a list of 16 colors. Select one of the colors. It will appear in the box under the color type heading.

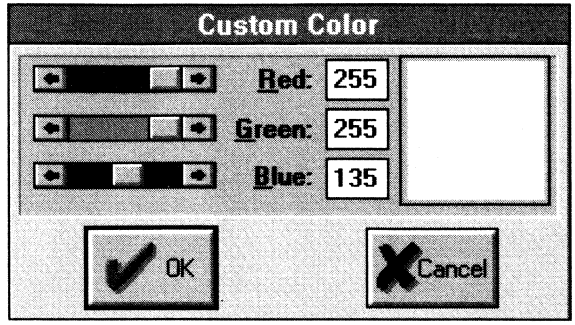
If you decide to customize a color, all objects that use that color will change from the old color to the customized color.

Customizing color

Once a color is customized, it becomes a global setting, so every time you choose Color, your custom colors appear. You might want to customize colors before you begin placing objects on your forms, although you can customize colors at any time.

The color palette is a global setting. You can customize the colors for the *entire* application by using the Custom Color dialog box for any object.

Figure 2.4
Custom Color dialog box



To customize a color, follow these instructions:

1. Click the color you want to customize. Note that you can't customize the transparent color (appears as white) or the black color.
2. Click Custom. A dialog box appears with the RGB (red, green, and blue) color settings for that color and the color displayed in a box.
3. You can edit the color by sliding the scroll box of the red, green, and blue colors until you see the color you want. You can also type the RGB numbers in their corresponding boxes. Once you type the number, the color automatically displays.

You can reset any color to its original color by either clicking cancel in the Custom Color dialog box or by selecting the color in the Color dialog box and clicking Reset.

C H A P T E R

3

Form design

This chapter introduces the Form Tool. The emphasis of this chapter is on creating your own forms.

Placing objects on a form

When you use the Form Tool, you can place objects on a form, create new forms, or use the Stack and Links Tools (you can also select the Stack and Links Tools when you first start ObjectVision, but for the Links Tool you have to have an open application).

1. To begin designing forms, start ObjectVision, then choose Tools | Form.
2. A box asking for the name of the form appears. Type the name of the form you want to create—you can always change the label later by inspecting the form's properties or by using the Stack Tool.
3. Begin placing objects on the form using either the menu or the buttons on the Object bar.

Using the Object bar

The Object bar has a button for each object you can place on a form. Each of the object buttons is described on page 30.

1. To place an object on a form, click its button on the Object bar.
2. If the object needs a label, you will see a dialog box asking for the label. ObjectVision automatically creates a default name. If

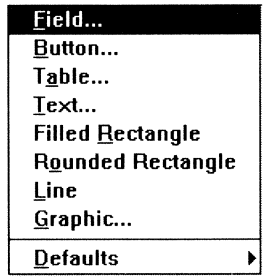
you want to use the default name, click OK or press *Enter*; otherwise, type in a new name, then press *Enter*.

3. The pointer will change from an arrow to an icon that represents the object you're placing on the form. ObjectVision has default sizes for each of the objects. If you want the default size, click the left mouse button once. If you want to size the object as you place it on the form, click the left mouse button and hold it while you drag the object to the size you want.

Using the menu

You can select each object by using the menu. When you select the Objects menu, you'll see the list shown in Figure 3.1.

Figure 3.1
Objects menu



Select the object you want to place on your form, then follow step 3 (in the previous section) for placing the object.

Sizing the object

After you place the object on the form, you'll see that the object has square handles on the corners. You can use these handles to resize the object.

When you place the pointer on a handle, the pointer changes from an arrow to a copy of the handle. Click and drag the handle until the object is the size and shape you want. Sizing tables is discussed later in this chapter.

Editing object properties

Every object has a set of properties. You can set default values for each property of an object. Setting defaults and object properties is described on page 32.

Once you've placed an object on a form, you can edit its properties. The quickest way to edit them is to select the object by right-clicking it. A list of that object's properties appears.

You can also see the object's properties by selecting the object, then selecting Properties | Object. A list of that object's properties appears. You can then select a single property, such as Color, then edit that property.

Using the Repeat command

When you select Properties | Object from the menu, you'll see the Repeat property. Although this isn't itself a property, it allows you to repeat the last setting of properties.

For example, if you create Field1 and give it a background color, then create Field2 and choose Properties | Object | Repeat, Field2 will have the same background color.

Note that this only works for one property at a time. You can't create a field and give it several properties, then repeat those same properties. If you want to repeat several properties at the same time, you can use object defaults (as described on page 32), or Edit | Copy and Paste.

Table objects

Table objects differ slightly from other objects. Tables *display* multiple values in columns, but each column can only have one value at a time—in the *current* cell.

The term *current row* is used to describe the current values in a row of the table. Each column in the table has a current cell that is in the current row. In Figure 3.2, the highlighted row is the current row. Notice the triangle to the left of highlighted row. This triangle, called the *row pointer*, displays next to the current row when you're completing a form (not designing it).

Note that when the row pointer changes position in form completion mode, the value in each column changes and each column in the table receives the Change event (for event trees on the columns). See Chapter 4, "Tree basics," for more information on event trees.

Figure 3.2
Table object current values

Customer Info Table		
Id	Name	Address
1386	Aberdeen	45 Utah Street
1388	Svenvald	Gouvernment House
1784	McDougal	4950 Pullman Ave NE
2177	Bonnefemme	128 University Drive
2579	Chavez	Cypress Drive

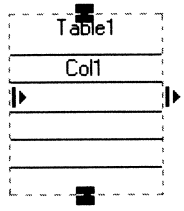
A table is made of one or more columns. Each column is made of one or more cells. To select a table, click the table label. To select a column in a table, click the column label. To select a cell, click inside the cell.

Placing a table object on a form

You can place a table object on a form just as you would place any form object (see the explanation on page 39). A table, however, is sized differently.

When you first place a table on a form, you'll see the table's label and a single column. If you use the ObjectVision default size (by clicking the mouse once), the table is one column in width by one cell in length. You can also drag a table to the size that you want. When you size the table, the length determines the number of cells, and the width determines the width of *one* column as shown in Figure 3.3. If you want more columns or cells, see the sections on adding columns and cells later in this chapter.

Figure 3.3
A table and a column



To move the entire table, drag the table label, or drag a cell in the table. To select the entire table, click the table label bar. If the table label isn't visible, click a cell in the table twice slowly. In Figure 3.3 the table is selected.

Editing table objects

Once you've placed a table object on a form, you might want to edit it.

Cut, copy, and paste

Unlike a field, a table (with unique names and properties) can exist only once in an application. There can be only one Table1, one Table2, and so on. There cannot be *two* tables with the same name (for example, you can't have two tables named Table1). You can cut, copy, and paste a table, but the results of these edits are different from cutting, copying, and pasting fields.

Pasting a cut table

The first time you paste a table that has been cut from the application, you'll get the exact table back. The table will have the same label names, links, and properties. If you try to copy the cut table *twice*, only the first table is exactly the same. Every other time you paste the table, you'll get a copy of the table with new label names, which you can change, but *no* trees or links.

Pasting a copied table

When you copy a table to the Clipboard, then try to paste it, you'll get a table with the same properties (color, size, fonts, field types), but the pasted table will have different label names, and *no* trees or links.

For example, if you have Table1 with Col1, Col2, and Col3 and you copy the entire table then paste it somewhere in that same application, the labels will change to ObjectVision defaults of Table2 with Col4, Col5, and Col6.

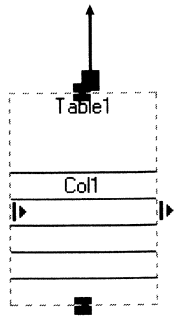
You can always rename the table and column label; you just can't rename the table and columns to exactly match the labels of another table.

Sizing the table label

To size the table label, place the pointer over the top square handle. Drag the handle up or down until the label reaches the size you want.

You can have a table with no label or you can create a table with a large label header as shown in Figure 3.4.

Figure 3.4
Sizing table labels



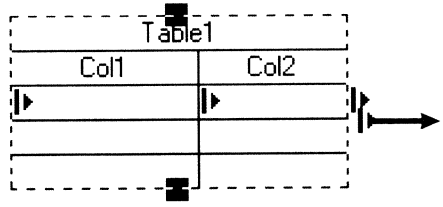
If you remove the table label by downsizing it, you can restore it by clicking twice slowly in a cell in the table (*not* a quick double-click, which opens the column value tree), then dragging the upper handle until the label is the size you want.

Adding columns to the table

To add a column to a table, select the table. The table appears with square handles on the top and bottom and triangle handles between columns and on both sides of the table.

Drag a triangle handle to add a column in that spot. ObjectVision gives the column a default name such as Col1. You can rename the column by right-clicking the column label, selecting Name/Text from the list of properties, then editing the name.

Figure 3.5
Adding a column



Moving columns in the table

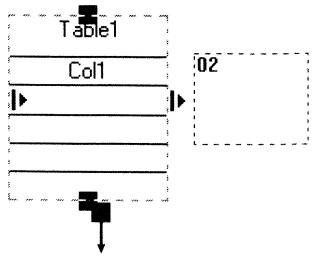
You can move a column by dragging it. For example, if you have Col1, Col2, and Col3, but you want to place Col2 in front of Col1, click the label of Col2 and drag it in front of Col1. To reposition any column, you only have to place part of it in the new location.

Adding or deleting rows in the table

To add rows to a table, select the table. All columns in a table must have the same number of rows.

Drag the bottom handle. When you move the handle up or down, you'll see the number of rows in the table. When you have the number of rows you want, release the mouse button.

Figure 3.6
Adding rows



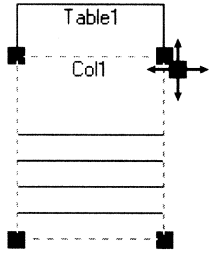
Sizing the column label

To size the column label, select the column instead of the table. To select the column, click the column label.

The column will be surrounded by a dotted line and handles will appear on the corners. Drag any handle up or down until the column label reaches the size you want. You can right-click in the column label to change any property of the column.

You can have a column with no label or with a large label as shown in Figure 3.7.

Figure 3.7
Sizing column labels



You can only remove the column label if you first remove the table label. You can restore the column label by clicking a cell in the table (twice slowly) then dragging the upper handle.

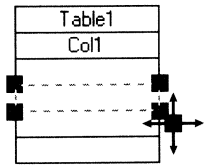
Sizing column cells

To size the cells inside a column, select any cell by clicking inside its box. All cells in a column are the same width and have the same fonts and colors. All cells in a table are the same height.

When the cell is selected, it will be surrounded by a dotted line and handles will appear on the corners. You can drag any handle up or down, left or right until the cell reaches the size you want. Notice that when you change the cell's size, you change the size of the table.

You can right-click in a cell to change the Value font and background color of all cells in the column.

Figure 3.8
Sizing cells



Editing cells

If you want to size all the cells in a column, select any cell in that column. Drag a handle until the cell is the size you want.

All cells in a *column* have the same width. When you change the width of a cell, you change the width of the entire column.

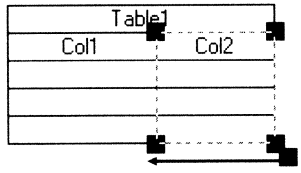
All cells in a *table* have the same height. When you change the height of one cell, the heights of all cells change.

Editing columns

You can't use the cut, copy, and paste commands with columns or cells in a column. Instead, you edit these parts of the table graphically.

To delete an entire column, select the column, or any cell in that column, then drag a handle towards the opposite side of the column until the two handles overlap.

Figure 3.9
Deleting columns



After you release the mouse button, ObjectVision asks if you want to delete the column. This is the last chance to back out because Edit | Undo doesn't work in this case.

If the column you just deleted was attached to another field, perhaps in a tree, the column isn't really deleted. It becomes a field on a Scratchpad form. When the value for that column is needed, ObjectVision goes to the Scratchpad and prompts the user for its value.

If you want to ensure that the column you deleted isn't on a Scratchpad form, use Field | Find from the main menu. If ObjectVision lists the column, you know it is used in a tree.

Adding scroll bars to a table

You can add scroll bars to a table object by selecting that property from the table's property list.

Scroll bars appear when you select the property. In the Form Tool, the scroll bar appears as a shaded area to the right of the table. You can see the actual scroll bars in form completion mode.

Scroll bars only work when the table object is linked to a database table. If the table isn't linked, the scroll bars appear during form completion mode, but they aren't active.

The scroll box in the scroll bar on a table will always appear in the middle. This happens because the table object doesn't know how many values are in the linked table (the database table size can always change).

When you use @functions or function keys, the row pointer moves up and down the table. With the scroll bars, the row pointer stays and the table values move.

If you use the arrows of the scroll bars, you'll scroll through the linked database table one record at a time. The row pointer doesn't change positions when you use the scroll bars. For example, if a table has three rows and the row pointer is next to the second row when you click the up scroll arrow, the value in row one moves down to row two (the current row), the value in row two moves to row three, and a new value appears in row one. The values move in the table object while the row pointer keeps its position.

Clicking in the areas between the scroll box and the arrows causes a page up or a page down. The row pointer doesn't change its position.

Saving an application with a table

If you save an application with values displayed, the values in a table may not be saved.

Displayed table values only get saved with an application if they are *not* linked to a database table.

If the ObjectVision table *is* linked to a database table, the values of the table are stored in the database and not in the ObjectVision application.

Editing forms

You can edit forms in several ways, but you must be using the Form Tool. To select and edit parts of a form, use Form | Select. This command selects an existing form so you can edit it.

You can then use Form | Find to locate a field. ObjectVision lists all the fields in the application. Once you select a field, ObjectVision selects the form it's on and then selects the field.

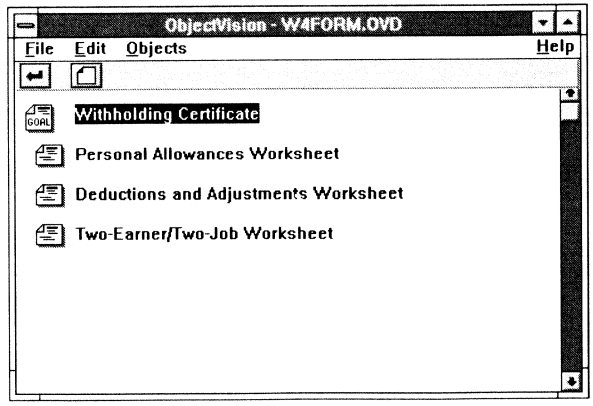
To create a new form, use Form | New. This command selects a new form. You'll be prompted for the form name when you start ObjectVision and select Tools | Form.

The Stack Tool

You can select Tools | Stack when you first start ObjectVision, or you can select the Stack Tool while you use the Form Tool. When you're using the Form Tool, there are two ways to select the Stack Tool: You can choose Tools | Stack from the menu, or you can click stack on the objects bar.

You can use the Stack Tool to reorganize forms in your application. If you open an application, then go to the Stack Tool, you'll see all the forms in that application in the order they appear in the application.

Figure 3.10
The Stack Tool



Rearranging forms

1. To rearrange existing forms in an application, open the Stack Tool.
2. Select a form by clicking it or by highlighting it with the arrow keys.
3. Use the Edit commands to cut, copy, and paste a form. When you paste a form, ObjectVision first allows you to edit the form name, then after you press *Enter*, ObjectVision pastes the form below the selected form unless you check Insert Before. You can also edit the form name before you paste it.

Creating a new form

1. You can create a new form in the Stack Tool by choosing Objects | Form.
2. ObjectVision will display the Form Name dialog box, where you type the name of the new form.
3. If you want the form to be inserted after the selected form, click OK or press *Enter*.
4. If you want the form to be inserted before the highlighted form in the Stack Tool, check Insert Before, then click OK or press *Enter*.

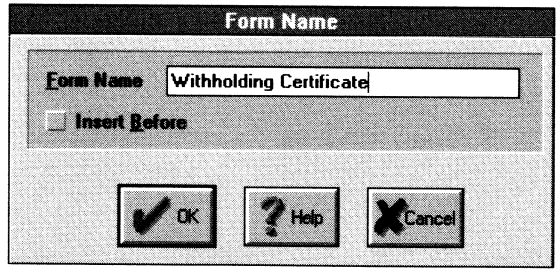


Figure 3.11
Typing the form name

Design tips

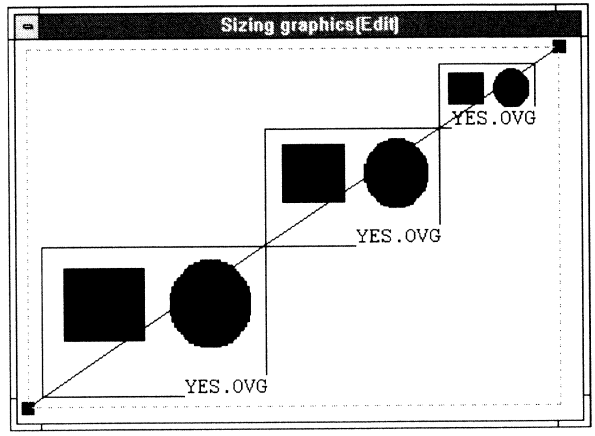
When you're designing forms, you might want to use one or two of these tips to enhance your forms.

Sizing graphics

You can increase or decrease the size of a graphic without distorting the image.

1. First, open the Form Tool and place the graphic you want to size on the form.
2. Create a line object and drag the line from a top corner of the graphic through the diagonal corner and extend it beyond the object boundaries.
3. Select the graphic object and drag the bottom corner handle along the diagonal line until one of the graphic's sides is the size you want.

Figure 3.12
Sizing a graphic proportionally

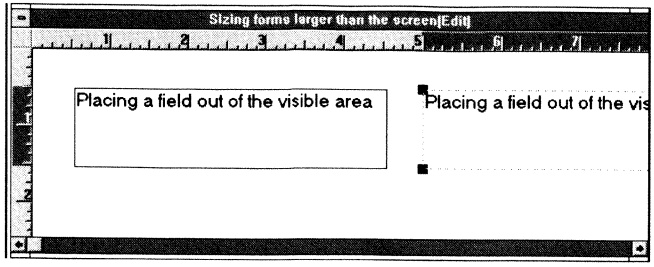


Sizing a form

You can use the rulers to judge the size of your forms. You can extend a form beyond the constraints of the computer screen by following this method:

1. Create any form object and extend its size.
2. Position the object so it extends beyond the form window.
3. Use the scroll bars to reposition the screen to the area you're working on.

Figure 3.13
Sizing with an object



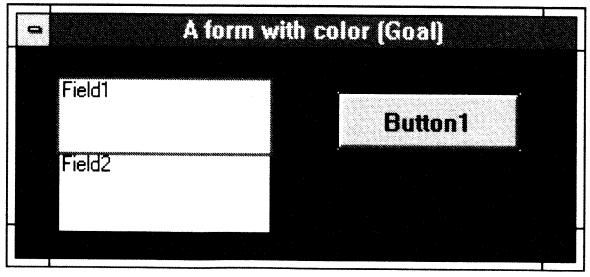
Adding color to an entire form

The background color of ObjectVision forms depends on the color settings in the Windows Control Panel. If you want to ensure that your users see the colors you want, create your own background color.

1. Create a filled rectangle that covers the entire form.
2. Right-click on the filled rectangle object. Choose Color from its list of properties.
3. Select a color. All filled and rounded rectangles are transparent. That is, any fields or tables show through the color of the rectangle. You can create the filled rectangle either first or last since all fields will show up through the color.

   Fields with a transparent (default) background color will appear to have the same background color as the filled or rounded rectangle. To avoid this, change the default background of your fields.

Figure 3.14
A form with colored background

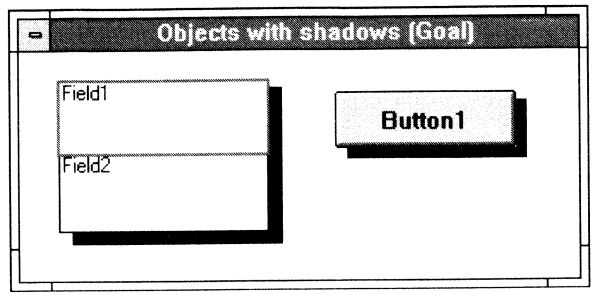


Creating shadows

You can create shadows to emphasize any form object.

1. In the Form Tool, choose Objects | Filled rectangle or click its button on the Object bar.
2. Position and size the object under or around an object.
3. You can attach a color or a fill pattern to the rectangle to give various effects.

Figure 3.15
Adding shadow to objects



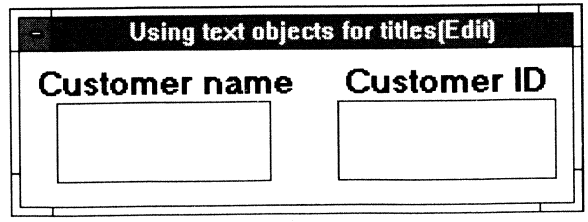
Using text fields

You can use text fields to give instructions to users or to create a larger or different field label.

1. First, create a field and uncheck Display Field Name in the Field Type dialog box. This removes the field name.
2. Create a text object with the field name or information as the text.
3. Position the text object either around or in the field.

If you place the text object *inside* the field, be sure to test it in form completion mode. A text object is hidden when a field is selected if the text object is located in the data entry area.

Figure 3.16
Adding text objects



C H A P T E R

4

Tree basics

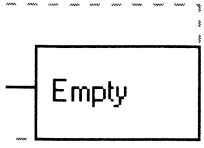
This chapter introduces trees and explains the difference between value trees and event trees. The next chapter explains how to create trees.

Understanding trees

Trees are a set of instructions attached to an object such as a field, a form, or a stack. There are two types of trees—value trees and event trees. Fields and columns are the only objects that can have both types of trees attached at the same time because they are the only objects that can have value trees.

- Value trees calculate a value for the object they are attached to.
- Event trees perform an action after an event occurs to their object.

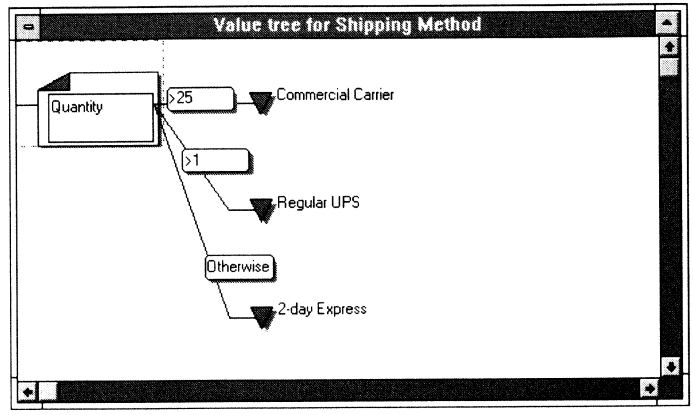
Trees are made of nodes. There are branch nodes—restricted and unrestricted—and conclusion nodes.



An empty node (shown at left) is a placeholder. It indicates the absence of a tree or the lack of a conclusion after a branch. While creating trees you can leave empty nodes in, but if ObjectVision encounters an empty node while evaluating a tree, an error value results.

Every node has a condition except the root branch (the first branch). ObjectVision reads trees from top to bottom and left to right. Figure 4.1 shows how nodes work together.

Figure 4.1
A simple value tree



ObjectVision reads the tree in Figure 4.1 in much the same way you'd read a flow chart: Is Quantity (branch) greater than 25 (condition)? If the answer is yes, then Shipping Method is Commercial carrier. If the answer is no, then evaluate the next condition.

ObjectVision works through a tree until it reaches a conclusion. If none of the conditions is true, ObjectVision will give the field an NA value (for value trees) or it won't perform an action (for event trees). When no other condition is true, ObjectVision checks for an Otherwise condition.

Otherwise is the last condition to be evaluated. When all other conditions are false, ObjectVision uses the Otherwise condition to obtain a value. If you always use Otherwise in value trees, your fields will always get values and not NA. You might get ERR from an @function that can't make its calculation.

Before you proceed with the rest of this chapter, you should be familiar with these three terms:

- **Branch** Branches evaluate fields. Once a field is evaluated it is compared with the first condition. In the above example, Quantity is the branch field. There are also unrestricted branches that don't depend on field values. A more detailed explanation follows.

- **Condition** — A condition is either true or false when compared with the branch value. In the above example, the first condition is >25. The tree evaluates if Quantity is greater than 25. If it is, then the following branch or conclusion is used.
- **Conclusion** — A conclusion is the end node that supplies a value or expression to the tree's field or object. In the above example, the possible values (conclusions) for the field Shipping Method are Commercial Carrier, Regular UPS, or 2-day Express, depending on which condition is true.

ObjectVision evaluates a tree by moving from top to bottom and left to right. If a condition is true, ObjectVision follows the instructions in the conclusion. If a condition is false, ObjectVision moves down the tree and evaluates the next condition.

Once a condition is evaluated as true, ObjectVision ignores any following conditions (even if they're also true). ObjectVision can't move up a tree, only down.

Value trees

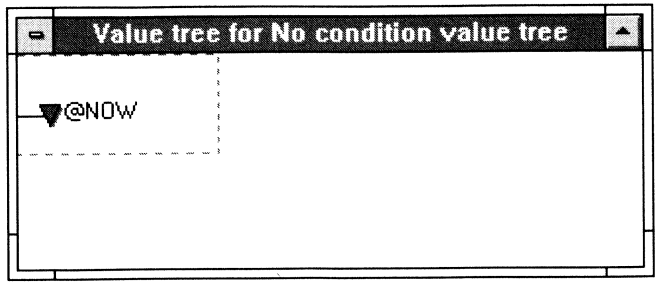
Value trees are the procedures ObjectVision uses to calculate a field's value. Value trees affect only the value of the field they are attached to. They can check values in other fields, but they cannot change them.

Value trees only calculate a value for their field after any conditions (if there are any) are met. Only one condition can be selected at a time. The last line in a value tree conclusion is the line that determines the field's value.

Value trees don't need conditions. For example, a value tree can contain only a conclusion expression that supplies a value to a field. Event trees, on the other hand, must have conditions.

Figure 4.2 shows how a field can have the current date or time when the application is opened (provided your computer's clock is properly set). In this example, you'd have to set the field type of No condition value tree to a date/time type, or you'd see only the date/time decimal number.

Figure 4.2
No condition value tree

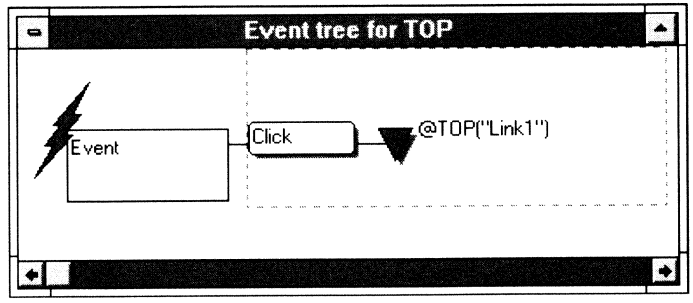


Value trees are evaluated when ObjectVision is first opened, when values a tree needs are supplied, and when links deliver values that the tree uses. This happens automatically, much like formulas in spreadsheets. You can, however, cause a recalculation by using Field | Calculate while in form completion mode.

Event trees

Event trees cause actions. Actions can assign values to other fields, select new forms, even change menu items. You can assign event trees to a form, a stack, or an object on a form.

Figure 4.3
An event tree



The following actions can be done with event trees but not with value trees:

- Select a new form when a button is clicked
- Change the menu when the application is opened (an event tree on the stack)
- Assign a value to another field
- Print a form when it's complete
- Save and close an application when all forms are complete
- Create new events (user-defined events) that can be used in any event tree condition

Event conditions

Figure 4.4 shows which objects can have event trees and what events each object can receive.

Figure 4.4
Objects and their events

Object \ Event	Click	Open	Close	Select	Unselect	Change	@EVENT	Ctrl+A through Ctrl+Z
Stack		✓	✓				✓	✓
Form		✓	✓	✓	✓	✓	✓	
Field or column				✓	✓	✓	✓	
Button	✓			✓	✓		✓	
Table								
Text	✓							
Rectangle								
Rounded rectangle								
Line								
Graphic	✓							

Notice that rectangles and lines can't have events. This is because these objects don't have names and can't be referenced in a tree. Tables can't have events, but each column in the table can.

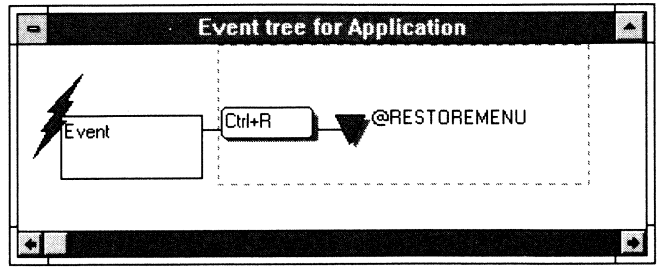
If you really want to place an event on a line or a rectangle you can create an illusion. Create an empty text object (with one border for a line), then attach the event tree.

Event trees are evaluated whenever an event occurs to the object. ObjectVision looks for a corresponding event condition in the object's event tree. You can also force an event by sending an event to an object using @EVENT.

Using Ctrl+A through Ctrl+Z

The control key plus A through Z are ObjectVision events to the stack. You can use these events to cause actions in the stack. For example, you could use *Ctrl+R* as a condition in a stack event tree with the conclusion @RESTOREMENU.

Figure 4.5
Using *Ctrl+R* to restore menus



When you use an application that has control keys, you hold down *Ctrl* while you press the letter key. This action sends the control event to the stack, which then performs the action (in the above example in Figure 4.5, the action is restoring the menus).

User-defined events

You can also define any event using @EVENT. You can use @EVENT to send an ObjectVision event to an item. For example, to send a click event to a button called Button1, you'd use @EVENT("Button1","Click").

To create your own event, you type the event you want instead of "Click". The item you send the event to should have that event in its event tree as a condition. For example, you could send an event called "menu" to the stack by using @EVENT("Stack", "menu").

Branches

There are two types of branches—restricted and unrestricted. A restricted branch is simply a branch that is associated with a field. An unrestricted branch is not limited to values in a field.

Restricted branches

If a restricted branch evaluates a field that gets its value from a tree, the branch symbol will have a small tree in it. That tree must be evaluated first since the branch can't be determined until its field has a value.

Figure 4.6
A branch with a tree

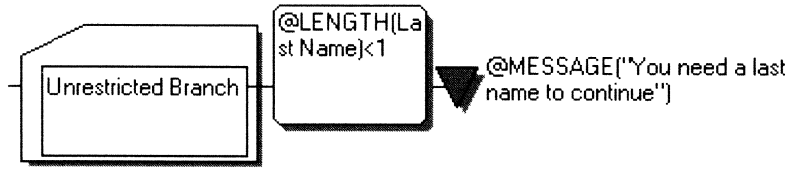
A restricted branch is useful if your tree is based on values in other fields, for example in the simple comparison *is Credit Report equal to Good?*. But if you want to do more than simple comparisons, unrestricted branches work best.

Unrestricted branch

Unrestricted branches allow you to evaluate fields in conditions to conclusions rather than in branches. In Figure 4.1, only the value in Quantity is evaluated in the root branch. Unrestricted branches aren't limited to one field and its value.

With unrestricted branches you can use @functions. The example below shows how this is useful in an event tree.

Figure 4.7
An unrestricted branch

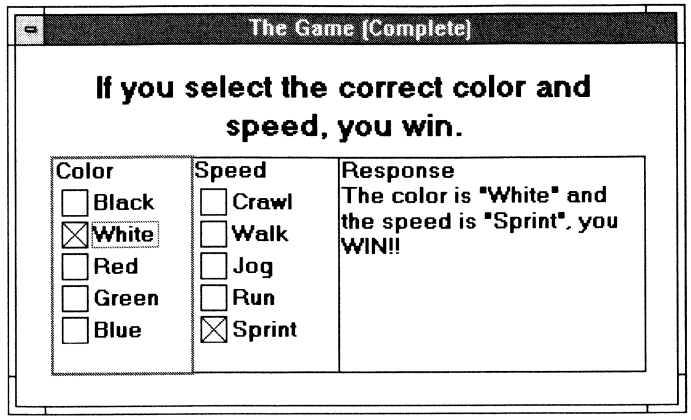


The example in Figure 4.7 could only be done with a normal branch if you had a field with the value @LENGTH(Last Name). The next example, however, *does* have fields that branches could be attached to. But if you tried to create the tree without the unrestricted branch, the tree would be pages long.

Suppose you have two fields and you need a third field to display a message based on the values in *both* previous fields. How could you do this without an unrestricted branch? It'd be fairly difficult unless you created a "dummy" field.

For example, Figure 4.8 has two fields, Color and Speed. The object of the game is to guess the correct color and speed.

Figure 4.8
The game

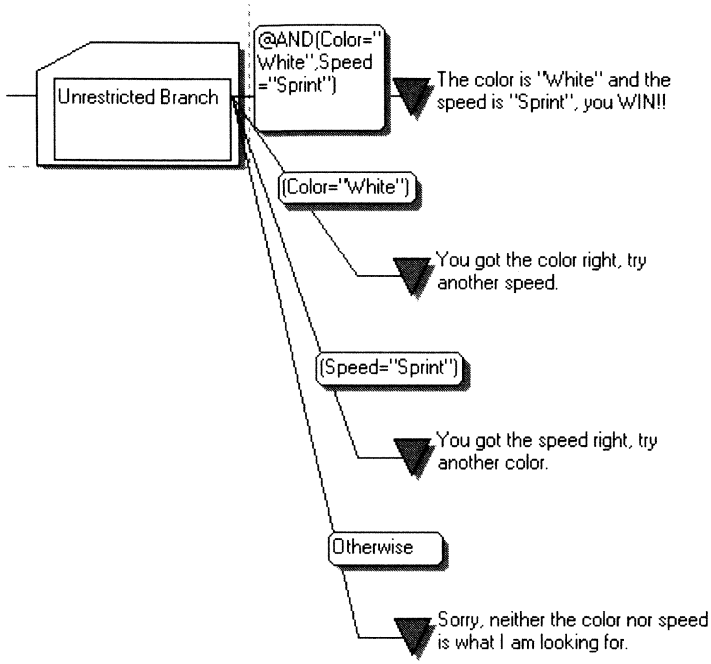


To play the game you need to know how you guessed. Four messages should appear in Response:

- one to tell you both of your guesses are wrong,
- one to tell you Color is right, but Speed is wrong,
- one to tell you Speed is right, but Color is wrong,
- and one to tell you that you guessed both right.

Figure 4.9 shows the value tree for Response. Notice that the unrestricted branch itself is not associated with any one field, but each condition is associated with both fields, one field, or neither field.

Figure 4.9
The value tree for Response



Conditions

Conditions that follow restricted branches always refer to the value of the branch's field. Conditions following unrestricted branches may refer to several different fields, or to expressions that aren't associated with particular fields.

If none of the branch conditions evaluates to true, ObjectVision can't calculate a value for the field and returns the error value *NA*.

Conditions can be blank or they can be expressions. These expressions can include @functions and operators. To learn to write expressions, read Chapter 6, "Writing expressions."

The Otherwise condition

You can use the reserved word Otherwise in conditions when the conclusion you need happens only when all other conditions are false. For an example, see Figure 4.9.

Conclusions

Conclusions can be as simple as returning a value to a field (such as sending 2-day Express to Shipping Method in Figure 4.1), or they can be complex expressions that use values from many fields.

Event vs. value conclusions

The only difference between event tree conclusions and value tree conclusions is that event conclusions *perform actions* while value conclusions *return a value* to the field. ObjectVision doesn't allow you to use conclusions in value trees that might cause actions. For instance, an event conclusion could override a value of another field and send a new value to it.

Event conclusions can change menus, open or close forms, create links, locate values in a database table, or simply display a message.

Both types of trees can have multilined conclusions. For value trees, the last line determines the field's value.

Printing trees

To print a tree you must be editing it—you can't print trees without using the Form Tool. From the tree menu choose File | Print Tree or Print All. Print Tree only prints the current tree you're viewing. Print All prints all trees of the type you're editing (value or event) in the application. To print a form or stack event tree, you must use File | Print Tree.

Large trees

Large trees are printed on several pieces of paper. You can reduce or expand the trees using the sizing buttons to fit more or less of a tree on the printed page.

ObjectVision prints a tree with the same number of nodes horizontally per page as are displayed on the screen.

C H A P T E R

5

Creating trees

This chapter teaches you how to create trees and gives several examples of both value and event trees. You should read the previous chapter on tree basics before you read this chapter.

Creating a new tree

To create a new tree (or to modify an existing value tree) you must first be in the Form Tool. Once there, select the object you'll attach the tree to. There are two ways to select objects: with the mouse or with the menus.

When you right-click an object, you're inspecting its properties. The list of properties appears near the object. From this list you can select Value Tree or Event Tree depending on the object you select.

Selecting with the mouse

To select a field, right-click anywhere inside the field. You can do the same for text, graphics, and buttons. For columns you'll need to select the column by clicking in the column title, and then right-click anywhere in the column.

To select a form, right-click anywhere in the form title bar. You can also right-click any space on the form that is not used by an object.

To select the stack, right-click anywhere outside the form but in the ObjectVision window. You can also right-click on the application title bar.

This following list displays after you right-click a field:

Figure 5.1
Properties for a field

Selecting with the menus

To select an object with the menu you must first select the object with the left mouse button or the *Tab* key, then choose Properties | Object. It's much easier and faster to right-click on the object.

To create a tree for a form, the form must be active. Use Form | Select. Next, choose Properties | Form | Event Tree from the menu.

Since there is only one stack per application, you don't need to select it before you can use Properties | Stack | Event Tree.

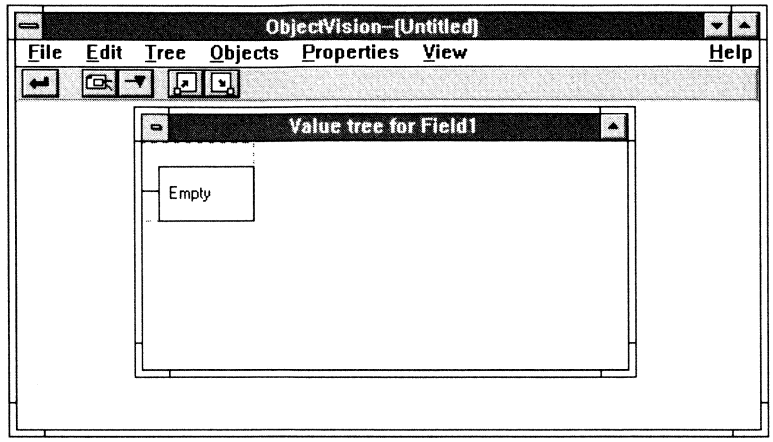
Selection shortcuts

- To create or edit a value tree you can double-click anywhere in the field and immediately go to that field's value tree.
- You can double-click a button, graphic, or text object to view its event tree.
- When you're editing a value tree, you can choose Tree | Select from the menu to select any other value tree in the application.
- While editing value and event trees, you can right-click in a branch to change it. You can also right-click in the condition or conclusion to edit it.
- To quickly edit a branch, condition, or conclusion double-click on it.

The tree object bar

Once you've selected either Value Tree or Event Tree, a window displays with a different Object bar and an empty tree.

Figure 5.2
A new tree



The five buttons on the Object bar correspond to commands on the menu bar. From left to right these buttons are

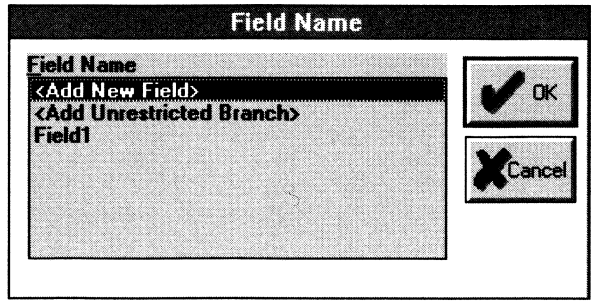
- **Close Tool**, which closes the window and returns you to the Form Tool. The menu equivalent is File | Close Tool.
- **Branch**, which inserts a branch at the selected location. The menu equivalent is Object | Branch.
- **Conclusion**, which inserts a condition and its conclusion at the selected location. The menu equivalent is Object | Conclusion.
- **Expand**, which magnifies the tree. The menu equivalent is View | Expand.
- **Reduce**, which shrinks the tree so that more will fit in the window. The menu equivalent is View | Reduce.

When you are viewing a tree that you've already written, a thin dotted line frames part of the tree. This box encloses either a branch or a conclusion. This is the selected area of the tree.

Adding a branch

To create or add a branch to a value tree, click the branch button or choose Objects | Branch. To add a branch to an event tree, you'll see the Event Condition dialog box before you see the branch types dialog box.

Figure 5.3
Branch types



If you want to create a restricted branch, select one of the fields from the list. You can either double-click the field name or highlight it and click OK.

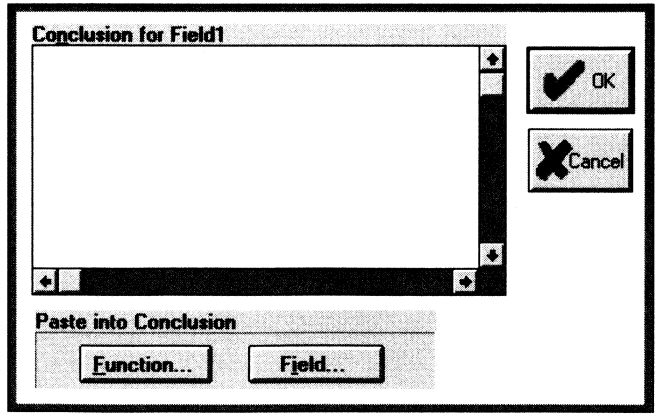
You can also create a new field. If you create a field here you should remember to place it on the form later. If you don't place it on a form, it'll appear on a Scratchpad form. To create a new field, select <Add New Field>. When a box appears, type the name of the field you want to create, then press *Enter* or click OK.

If you want an unrestricted branch, select <Add Unrestricted Branch>.

Adding conclusions

Once you click the conclusion button or choose Objects | Conclusion, the Conclusion dialog box appears (unless the conclusion requires a condition). Figure 5.4 shows the Conclusion dialog box for a value tree.

Figure 5.4
Value tree Conclusion dialog box



1. Type a conclusion expression. To write more than one line, use *Ctrl+Enter* instead of *Enter* because *Enter* alone is the same as clicking OK.
2. You can use @functions in your expressions by clicking Paste Function or by typing the function yourself.
3. If you're using fields in your expression, you can use Paste Field. Using this button eliminates typos. The list is also handy when you have a lot of field names and can't remember all of them.
4. When you've finished writing the conclusion, press *Enter* or click OK.
5. If you decide not to keep the conclusion you've written, click Cancel.

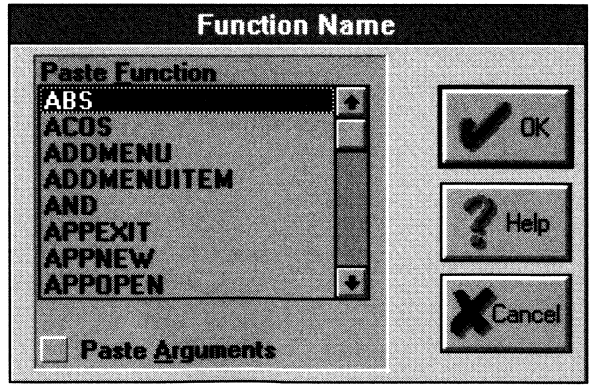
Using Paste Function

The Paste Function button works the same for conditions and conclusions in both value and event trees. The only difference is that in event tree conclusions you can use both value and event functions, while in any condition or in value trees you can only use value functions. The Paste Function box will list only the @functions that you can use.

If you type an event function anywhere except in an event tree conclusion you'll get an error. This is a good reason to use the Paste Function button.

When you click Paste Function a list of the available functions appears. There is a check box beneath the list for pasting arguments. If you click the box, the argument types for the function appear when you paste the function in your expression.

Figure 5.5
The Paste Function dialog box



To paste a function, either double-click on the function or highlight it using the mouse and scroll bars, then click OK.

Using Paste Field

You can paste field names into your tree expressions by using the Paste Field button. This button works like the Paste Function button. When you click the button you'll see a list of all the fields in your application.

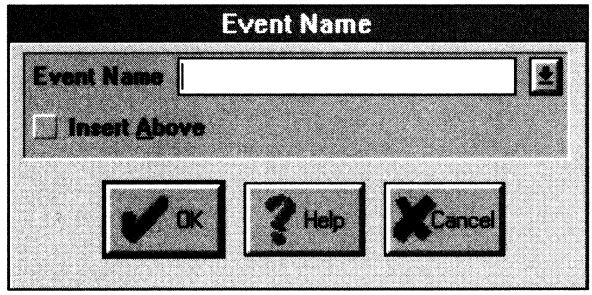
This button is practical when you have a lot of fields and can't remember the exact field names.

Adding conditions

If you add a branch or conclusion after the root node, you'll be prompted for a condition first.

Figure 5.6 shows the Condition dialog box for an event tree.

Figure 5.6
The Event tree Condition dialog box



You can select an event by clicking the down arrow to the right of the Event Name box, or you can type in the event you want to use.

Event trees must begin with an event condition. From then on, both value and event trees work the same.

To add a condition follow these steps. Note that the Condition dialog box resembles the Conclusion dialog box shown in Figure 5.4.

1. Type a condition expression or an event for event conditions.

2. If you check Insert Above, the condition and its corresponding branch or conclusion will be inserted above the selected area in the tree. Otherwise, the new condition is placed below the selected area in the tree.

3. You can use @functions in your conditions just as you would for conclusions.
4. If you're using fields in your expression, you can use Paste Field.
5. When you've finished writing the condition, press *Enter* or click OK.
6. If you don't want to continue writing the condition, click Cancel. This cancels only the current condition.

To learn how to write expressions see Chapter 6, "Writing expressions."

Editing trees

Scrolling and viewing value trees

You can view more or less of the tree by using the Expand or Reduce buttons. ObjectVision adds scroll bars to the tree window whenever a tree is larger than the window. You can scroll the window vertically or horizontally to view other portions of the tree by using the mouse or by pressing ↑ or ↓ or ← or →.

Edit menus

You can edit trees that you've written or copied. The following paragraphs describe each edit command. Their keyboard shortcuts are listed as well.

Undo
Alt Backspace

Edit | Undo reverses the previous edit—one you may have made in error. Undo only undoes one previous edit. You can't use Undo to reverse any other edits.

Cut
Shift Del

Edit | Cut deletes the highlighted branch or conclusion and everything following it. A copy of the cut material is sent to the Clipboard, so you can paste it back until you copy something else to the Clipboard.

Copy Ctrl Ins

Edit | Copy copies the highlighted branch or conclusion and everything following it to the Clipboard. Copy works like Cut except is doesn't delete the highlighted part.

Paste Shift Ins

Edit | Paste places a copy of the tree from the Clipboard to the current tree. Use Paste to insert a portion of a tree that was previously copied or cut to the Clipboard.

Clear Del

Edit | Clear deletes the highlighted branch or conclusion without changing the Clipboard contents.

Changing branches

You can change both restricted and unrestricted branches. It's easy to change the field of a restricted branch. Changing an unrestricted branch to a restricted branch (and vice-versa) is more complicated.

Changing restricted branches

There are three ways to change the field reference of a restricted branch.

- First, double-click on the branch that you want to edit.

  Note that choosing Name from the list changes the name of the field in the branch. It also changes the name of the field everywhere else in the application.
- Second, highlight the branch then choose Properties | Name to change the name, or Properties | Field to get a list of available fields. ObjectVision displays an alphabetical list of field names. You can choose to create a new field or change the restricted branch to an unrestricted branch.
- Third, highlight the branch, right-click in the area, then choose Field for a list of the available fields.

After changing the branch field, you should make sure that any conditions that follow the branch are still valid.

Changing an unrestricted branch

Right-click on the unrestricted branch, then choose Field. You can also choose Properties | Field.

When the Field Name dialog box appears, either select a field name or choose <Add New Field>.

Remember that once you change an unrestricted branch to a restricted branch you'll most likely have to alter all of the conditions that follow that branch.

Changing conclusions

You can change a conclusion expression in three ways. Select the tree, and then either

- highlight the conclusion/conclusion area, right-click in the highlighted area, then choose Conclusion, or
- highlight the conclusion/conclusion area, then choose Properties | Conclusion, or
- double-click the conclusion that you want to edit.

ObjectVision displays the Conclusion dialog box that contains the current conclusion expression. Once you change the conclusion expression and choose OK, the tree displays the change.

Changing conditions

You can change the condition expression in the same three ways as a conclusion. See the "Changing conclusions" section for directions.

The Condition dialog box appears displaying the current condition. You can either retype a condition or edit the existing condition. Once you change the condition, choose OK or press *Enter*.

Copying trees

You can copy trees or portions of trees from other fields and other ObjectVision applications by using the Windows Clipboard. This is a quick way of duplicating a long tree, and is faster and easier than writing the new tree entirely from scratch.

Once you copy a tree you can edit it to fit into your application. When you copy trees keep in mind that the Clipboard can hold only one cut or copied tree portion at a time.

To copy an entire value tree, highlight the root branch, then select Edit | Copy. To copy an event tree, highlight the first node after the root event (the event condition). You'll need to change the event every time you copy an event tree. Note that you can only copy one event (and its branches) at a time.

To copy sections of a tree, highlight the beginning node and copy it. The rest of the tree is copied. For more detailed information, see the sections below.

Tree | Select vs. Tree | Find

Tree | Select shows the tree of the field you select—even if it's blank. Tree | Find finds the first tree in the application that uses the value of the field you select.

When you're viewing a tree and want to see that type of tree (value or event) of another field, use Tree | Select. Double-click the field you want from the list, then that tree will appear. For example, if you're viewing a value tree, Tree | Select will only show other value trees, not event trees.

When you're viewing a tree and want to see a tree that contains a specific field, you can use Tree | Find. When you select a field, ObjectVision displays a tree that contains the field. If the field appears in multiple trees or multiple times in the same tree, you can repeatedly use Tree | Find until the additional trees are displayed.

Application to application

To copy a tree from one application and bring it to another, you can either open both applications or you can open one, cut or copy to the Clipboard, then open the second application and paste. To open two applications at the same time, open ObjectVision twice. Windows makes it *appear* as if two ObjectVisions are running simultaneously.

Open one application in each window of ObjectVision. Go to the tree that you want to copy. Highlight the branch that contains what you need. If you're copying the entire tree highlight the root branch.

Highlighting trees

To select part of a tree, you only need to highlight the beginning of the part. For instance, to select an entire value tree you only need to highlight the root branch because ObjectVision copies everything down the tree starting with the selected area.

You can only highlight a branch or a condition/conclusion. When you click part of the tree, it becomes framed in a thin dotted line—the highlight.

After you've highlighted the part of the tree you want, choose Edit | Copy from the menu. ObjectVision places a copy of the tree in the Clipboard.

Next, go to the other application and select the object you want to attach the tree to. When you're viewing that object's empty tree, select Edit | Paste. ObjectVision places an exact copy of the first tree in the current object's tree. You can edit the tree to fit into your application.

ObjectVision doesn't copy the condition. After you paste the tree, you'll need to add the condition. Note that if you paste at the tree's root, you don't need to edit it since roots don't have conditions.

Object to object

You can't copy information between tree types. If you copy a value tree, it can only be pasted into another value tree.

To copy a tree from one object to another, highlight the tree as described in the previous section. Copy it using Edit | Copy. Select the other tree by either using Tree | Select for value trees, or by clicking Close Tool, right-clicking on the object, then selecting Value Tree or Event Tree from its properties.

Once you're viewing the object's tree you can paste the copied tree by using Edit | Paste.

If you want to paste part of a tree in the middle of another tree, copy the part, then highlight the branch or conclusion where you want to paste the part. ObjectVision pastes the part directly below the highlighted branch or conclusion unless you check Insert Above in the Condition dialog box.

C H A P T E R

6

Writing expressions

This chapter teaches you how to write expressions and explains operators and their order of precedence.

Expression basics

Expressions make calculations based on values you enter into the expression, or on values contained in a field or in a database table. The syntax of ObjectVision expressions is compatible with Quattro Pro's formulas. If you know how to write Quattro Pro formulas, you already know how to write ObjectVision expressions.

Expression locations

Expressions are located throughout ObjectVision in:

- value and event tree conclusions
- value and event tree conditions
- linking options: VFields and filters
- commands sent through DDE *to* ObjectVision

Expression syntax

Expressions can be up to 4,096 characters long. You can include spaces between operators and values, but ObjectVision deletes them automatically.

Table 6.1
How an expression is evaluated

The expression is treated as a	if the first expression begins with
Label (string)	' "" ^
Formula	. + - (@ # $ 0 1 2 3 4 5 6 7 8 9
Reserved word	Yes No True False
String	<none of the above>

A label is a constant string value.

A formula is one or more values separated by operators. Values can be field names that represent values, @functions that compute values, or constants such as numbers or strings.

A reserved word is a word that has a specific meaning to ObjectVision. The case doesn't matter—Yes is treated the same as YES and yes.

Condition expressions

Condition expressions for trees are slightly different. They have more reserved words and may begin with a comparison operator (such as < or >).

Table 6.2
How a condition is evaluated

The condition is treated as a	if the condition begins with
Label (string)	' "" ^
Modified formula	= < > <= >= <>
Formula	. + - (@ # $ 0 1 2 3 4 5 6 7 8 9
Reserved word	Yes No True False
Reserved word for conditions	Otherwise
Reserved word for event conditions	CTRL+A through CTRL+Z Open Click Change Select Unselect Close
String	<none of the above>

In a condition, the initial comparison operator is *always* applied last regardless of operator precedence rules.

Potential values for a field

If you write an expression that doesn't use operators, @functions, or parentheses, the expression is treated as a constant. It can be a numeric constant such as 3.1415 or a string constant such as red.

Whenever you have a *constant* (not an formula) in a condition or a conclusion expression, that constant becomes a possible value for the field that it refers to.

For example, if you have a branch on Field1, then conditions of 1, 2, 3, and 4, those four constants are possible values for Field1.

If you format Field1 as a selection list, ObjectVision will automatically create the list with the above constants as possible values for Field1. This works for all Selection Method field types.

Operator precedence

Formulas use operators, or mathematical symbols, to express a relationship between two or more values. Any blanks before or after the operator are ignored.

The result of a formula depends on the order in which ObjectVision performs the arithmetic operations. ObjectVision has assigned a *precedence* to the operators and evaluates the operators in their order of precedence.

Table 6.3 lists the ObjectVision operators in their order of precedence. The precedence of operators is from top to bottom and left to right for each line with multiple operators.

Table 6.3
Expression operators

Order of precedence	Operator
First	%
Second	^
Third	– (negative) + (positive)
Fourth	* /
Fifth	+ –
Sixth	< > <= >= = <>
Seventh	&

% (percent) calculates the percentage of a number. For example, 98% is 0.98. ObjectVision will retain the % sign for easy reading in trees.

^ (exponential) takes a number to the power of another number. For example, 2^3 is read as two to the power of three which equals 8, and 3^2 is read as three to the power of two which is 9.

– (negative) changes the sign of the value. For example, if Field1 has a value of 10, –Field1 is –10, and if Field1 has a value of –10, then –Field1 is 10.

+ (positive) can be used to force a formula. If you want to begin a formula with a field name, you can write +Field1-Field2. You may want to avoid any confusion by only using the parentheses to force a formula, such as (Field1-Field2).

*** (multiplication)** multiplies two numbers. If Field1 has a value of 10, (Field1 * –Field1) is –100.

/ (division) divides two numbers. If Field1 has a value of 10, (Field1 / –Field1) is –1.

+ (addition) adds two numbers. For example, if Field1 has a value of 10, (Field1 + Field1) is 20. The addition symbol in this expression cannot be confused with the positive sign (+) because the expression would then be meaningless.

– (subtraction) subtracts one number from another.

= (equal) is used in logical expressions that are evaluated as Yes or No. (Field1=10) is read *is Field1 equal to 10?*

<> (not equal) compares two values. For example, the expression (Field1<>10) is evaluated as Yes if Field1 does not equal 10, but is evaluated as No if Field1 does equal 10.

< (less than) compares two values. For example, 0 < 10 results in a Yes value.

> (greater than) compares two values. For example, 0 > 10 results in a No value.

<= (less than or equal) compares two values. For example, Field1 <= 10 results in a Yes value if Field1 is either less than or equal to 10.

>= (greater than or equal) compares two values. For example, Field1 >= 10 results in a Yes value if Field1 is either greater than or equal to 10.

& (string concatenation) concatenates ("adds") two strings together. For example, ("Address"&"Field1") will have the value AddressField1.

Using parentheses

To ensure that ObjectVision evaluates an expression in the order you want, use parentheses to enclose the portion you want calculated first. You can nest parentheses inside other parentheses, but don't take nesting beyond eight levels. ObjectVision calculates the innermost set of parentheses first. For example,

4 * 2 + 3 = 11
4 * (2 + 3) = 20
(4 * 2) + (3 + 5) * 4 = 40
((4 * 2) + (3 + 5)) * 4 = 64

If you don't use parentheses, ObjectVision performs the calculations according to their rules of precedence. Without the parentheses, the last equation, 4 * 2 + 3 + 5 * 4, would be evaluated

4 * 2 = 8, and
5 * 4 = 20, so
8 + 3 + 20 = 31!

Field names in expressions

You can include field names in an expression to refer to the field's *current value*. When a field name is in an expression, the expression is evaluated *only* if the field has a value.

When you enter a field name in an expression, you don't have to match the case of the field name.

If you refer to a field name that doesn't exist, ObjectVision automatically creates a new field by that name. Unless you later add the field to a form using the Form Tool, ObjectVision will prompt for this field using the Scratchpad form.

To avoid inadvertently creating new fields, you can use the Paste Field button instead of typing the field name.

If you prefer typing field names, you should get used to enclosing field names within single quotation marks. Single quotation marks are not needed for all field names, but since they are required for some names (see the following section), you might want to get into this habit and avoid errors.

Note, however, that if you use the Paste Field button in the Condition and Conclusion dialog boxes to include a field name in an expression, quotation marks are automatically added if they're needed. (These dialog boxes are discussed starting on page 68.)

Pasting the name eliminates typographical errors and makes it easy for you to include a field name in a formula.

Single quotation marks

Single quotation marks are required in four situations:

- Your field name starts with any character other than a letter, such as *'2nd Choice'* and *'%Gain'*.
- The last character of your field name is a blank, such as *'Other . . . '*.
- Your field name contains characters other than letters, numbers, spaces, periods, or underscores. For example, *'Self-Employed'*, *'City/State'*, and *'US Citizen?'*.
- Your field name is one of the reserved keywords *Yes*, *No*, *TRUE*, or *FALSE*.

In addition, you must use *two* single quotation marks when a single quote is part of a field name. You must then enclose the entire field name in single quotation marks as well because it contains a character that is not a letter, number, space, period, or underscore. For example, to enter the field name Spouse's Occupation in an expression, type `'Spouse''s Occupation'`.

Data conversion

While evaluating an expression, ObjectVision attempts to perform any necessary data conversions. For operations and @functions that require numbers, text values are converted into numbers when possible.

Data types

ObjectVision fields have one of four data types:

Don't use symbols for separating thousands.

- **Numeric:** Numeric values are numbers such as *1048*, *–85*, or *39.95*. ObjectVision does not support an exponential format for numbers in expressions.
- **String:** String values are a sequence of characters containing between 0 and 4096 characters. In a formula, you must double-quote a string. To represent a double quote within a string, use two double quotes instead of one.
- **Logical:** Logical values are boolean types: either Yes or No.

- **Error:** ObjectVision returns ERR or NA when an error occurs or when you use @ERR or @NA.

Conversion samples

Table 6.4 shows the complete set of data conversion rules. The Error data type is not included in the table because any error value in an expression causes the result of the expression to be that error value.

Table 6.4
Automatic data conversion in expressions

	Numeric result	String result	Logical result
Numeric data	No conversion required.	Converts the number to a string form of the number in general format.	Converts 0 to No, all other numbers to Yes.
String data	Converts the string to a number, or ERR. The empty string "" is converted to 0.	No conversion required.	Converts the string to a logical value, or ERR.
Logical data	Converts Yes to 1; converts No to 0.	Converts Yes to the string "Yes"; converts No to the string "No".	No conversion required.

An ERR error results when ObjectVision is unable to convert an argument to the necessary type.

For example, the result of the expression +"23"+3 is 26. The text string "23" is converted to the number 23 for the addition operation.

Similarly, the result of the expression (Yes+Yes) is 2. The logical value Yes is converted to 1 for the addition operation. No is converted to 0.

The result of the expression +"42"&3 is "423". The numeric value 3 is converted to a "3" string for the concatenation operation.

If you want to format a number in a special way, see @FORMAT in Chapter 8, "@Function descriptions."

PART 2

@Functions

C H A P T E R

7

@Function basics

This chapter describes the types, argument syntax, and locations of @functions. All ObjectVision @functions are listed and described with examples in the following chapter.

What are @functions?

ObjectVision provides many built-in functions—called *@functions* since they are always preceded by an at-sign—that perform calculations or procedures. @Functions don't return a value until all arguments have a value, although some @functions don't require arguments.

All @functions are located in expressions. There are three places for expressions—value trees, event trees, and link options. Because not all @functions can go in all three locations, the @functions are divided into two groups.

- *Value functions* perform a calculation and return a value.
- *Event functions* perform an action such as registering a custom function, selecting a form, or assigning a value to a field.

Value functions

These functions take a list of arguments and perform a calculation based on the argument values. For example, the function @SUM adds all the values in its argument list and returns the *sum* of

those values. If you wrote @SUM(1,2,3,4), the calculation is 1 + 2 + 3 + 4, and the returned value is 10. You can substitute field names for numbers and get the sum of several field values.

Value functions can be included in expressions as follows:

@SUM(1,2,3,4) – @SUM(Field1,Field2)

If the value for Field1 is 3 and the value for Field2 is 4, the returned value of the expression would be 3 because

1 + 2 + 3 + 4 = 10

and

3 + 4 = 7

so

10 – 7 = 3.

Value functions can appear in all three locations for functions. A brief list of all functions begins on page 89.

Event functions

Event functions are slightly more complex than value functions—they return values as well as perform actions. An event function *cannot* be used in value trees, but all value functions can be used in event trees.

For more information regarding event trees and ObjectVision events see page 57.

Event functions can only be used in event tree conclusions. They can't be used in conditions, value tree conclusions, or link expressions. However, DDE execute commands are treated as event conclusions so event functions can be used in them. For more information on DDE see Chapter 16, "Linking through DDE."

Event functions perform their actions *after* an event occurs. For example, if you want to assign the value 10 to Field2 every time Field1 has been selected, you would create an event tree for Field1 where the event condition is *Select*. The conclusion to the event would be @ASSIGN(Field2,10).

Using @functions

When you use @functions in expressions, you must pay special attention to function syntax and arguments.

Function syntax

All ObjectVision functions have the same basic syntax:

@FUNCTION(*Argument 1, Argument 2, ...*)

A function's syntax must meet the following rules:

- You must always include the leading @ when typing a function name.
- You can type the function names in either uppercase or lowercase letters, or a combination of the two.
- You must enclose the required arguments in parentheses.
- When there are multiple arguments, you must separate them with the separator character you specify in the Windows Control Panel. In the U.S., the default list-separator character is the comma.
- If the function syntax specifies a certain order for arguments, you must enter them in that order.
- You cannot leave spaces between the @ sign and the function name. You *can* leave spaces between arguments, parentheses, and the function name, but ObjectVision deletes them after you have defined the expression.
- You can nest functions inside other functions. For example, @ INT(@NOW).
- Some functions, such as @NOW, don't take arguments and don't require parentheses.

Arguments in functions

The information required by the function is called the *argument*. Most functions need at least one argument. The type of information required depends on the specific function. There are three general types of arguments:

When you use arguments, you must separate them with commas.

- numeric values
- string values
- logical values

Each function argument can be a constant value, a field name, an expression, or another function. ObjectVision will convert arguments to the type required by the function. See Chapter 6, "Writing expressions," to learn about writing expressions.

Enclose constant strings within double quotation marks.

When you enter a constant string value as an argument, you must enclose it within double quotation marks. Constant string values include phrases, sentences, numbers (as strings), or even file names, field names, and form names.

Check Paste Arguments when you paste functions. Paste Arguments gives you help for using quotes with arguments.

When the field name contains a comma or an apostrophe and you want the field's *value*, you must enclose the field name in single quotation marks and precede the apostrophe with another apostrophe. For example, if a field name is Spouse's Occupation, the field name would be written as 'Spouses"s Occupation' in the function argument.

Using parentheses in arguments

Certain @functions perform their actions or calculations on fields or columns. The @function can use the value of the field you type in for the argument, *or* the @function can *reference* another field or column whose name is the value in the field you type in for the argument.

For instance, @ASSIGN sends a value to a field. The syntax is @ASSIGN(*FieldName, Value*). If you want to send the value 234 to Field1, you'd write @ASSIGN(Field1, 234).

If, however, you want to use Field1 as a field whose value specifies another field (perhaps by a selection list of field names), you'd type @ASSIGN((Field1), 234). The parentheses tell ObjectVision to send the value to the field whose name is referenced in Field1.

Function types

All @functions are categorized as either value functions or event functions. You'll notice that you can't use event functions in value trees, but you can use value functions in event trees. Value functions don't have restrictions. They can be used anywhere expressions can be used.

Event functions do have restrictions—they can only be used in event tree conclusions. An easy way to know if you can use a function is to always use the Paste Function command from the Condition or Conclusion dialog box because ObjectVision will only display the functions you can use.

All ObjectVision functions have a category and a type. The categories are either Value or Event. In each category there are several types.

Table 7.1
Value and Event function types

Value function types	Event function types
Mathematical	Linking
Date/time	Menu customization
String	Menu equivalents
Logical	Miscellaneous
Financial	
Miscellaneous	

Table 7.2: Mathematical functions

Mathematical functions	Returns
@ABS(*X*)	Absolute value of *X*.
@ACOS(*X*)	Arc cosine of *X*.
@ASIN(*X*)	Arc sine of *X*.
@ATAN(*X*)	Arc tangent of *X* (2 quadrant).
@ATAN2(*X*,*Y*)	Arc tangent of *Y*/*X* (4 quadrant).
@AVG(*List*)	Average of *List*.
@COS(*X*)	Cosine of angle *X*.
@COLUMNAVG(*ColumnName*)	Average of all displayed values in *ColumnName*.
@COLUMNCOUNT(*ColumnName*)	Number of items displayed in *ColumnName*.
@COLUMNMAX(*ColumnName*)	Maximum value displayed in *ColumnName*.
@COLUMNMIN(*ColumnName*)	Minimum value displayed in *ColumnName*.
@COLUMNSUM(*ColumnName*)	Sum of all displayed values in *ColumnName*.
@DEGREES(*X*)	Number of degrees in *X* radians.
@EXP(*X*)	*e* raised to the *X*th power.
@INT(*X*)	Integer portion of *X*.
@LINKAVG("*LinkName*", "*DataField*")	Returns average of all values in current range for *DataField*.
@LINKCOUNT("*LinkName*")	Returns the number of records in the current range.
@LINKMIN("*LinkName*", "*DataField*")	Returns the minimum value of *DataField* in the current range.
@LINKMAX("*LinkName*", "*DataField*")	Returns the maximum value of *DataField* in the current range.
@LINKSUM("*LinkName*", "*DataField*")	Returns the sum of all values in *DataField*.
@LN(*X*)	Log base *e* of *X*.
@LOG(*X*)	Log base 10 of *X*.
@MAX(*List*)	Maximum value in *List*.
@MIN(*List*)	Minimum value in *List*.
@MOD(*X*,*Y*)	Remainder of *X*/*Y*.

Table 7.2: Mathematical functions (continued)

@PI	The value π (3.141592653589793238).
@RADIANS(*X*)	Number of radians in *X* degrees.
@ROUND(*X,Num*)	*X* rounded to the number of digits specified with *Num* (up to 15).
@SIN(*X*)	Sine of angle *X*.
@SQRT(*X*)	Positive square root of *X*.
@SUM(*List*)	Sum of values in *List*.
@TAN(*X*)	Tangent of angle *X*.

Table 7.3: Date and time functions

Date/time functions	**Returns**
@DATE(*Yr, Mo, Day*)	Date serial number
@DATEVALUE("*DateString*")	Date serial number
@DAY(*DateTimeNumber*)	Day of the month (1-31)
@HOUR(*DateTimeNumber*)	Hour of the day (0-23)
@MINUTE(*DateTimeNumber*)	Minute of the hour (0-59)
@MONTH(*DateTimeNumber*)	Month (1-12)
@NOW	Current date/time serial number
@SECOND(*DateTimeNumber*)	A second (0-59)
@TIME(*Hr, Min, Sec*)	Time serial number
@TIMEVALUE("*TimeString*")	Time serial number
@TODAY	Current date serial number
@WEEKDAY(*DateTimeNumber*)	Day of the week (1-7)
@YEAR(*DateTimeNumber*)	Year (1800-2099)

Table 7.4: String functions

String functions	**Returns**
@CHAR(*Code*)	Character for ANSI decimal number *Code*. (See Appendix D for the ANSI character set.)
@CODE("*String*")	ANSI decimal code for the first character in *String*.
@EXACT("*String1*", "*String2*")	Yes if *String1* and *String2* are identical; otherwise, No.
@FIND("*Substring*", "*String*", *StartNum*)	Character position of the first *Substring* found in *String*.
@LEFT("*String*",*Num*)	First *Num* characters in *String*.
@LENGTH("*String*")	Number of characters in *String*.
@LOWER("*String*")	Lowercase letters of *String*.
@MID("*String*", *StartNum, Num*)	*Num* characters of *String*, beginning with the *StartNum* character position.
@PROPER("*String*")	Text in *String* with the first letter in each word capitalized.
@REPEAT("*String*", *Num*)	*String*, repeated *Num* times.
@REPLACE("*String*", *StartNumber, Num*, "*NewString*")	Removes *Num* characters from *String*, beginning with *StartNumber*; inserts *NewString* in its place.
@RIGHT("*String*", *Num*)	Last *Num* characters in *String*.

Table 7.4: String functions (continued)

@TRIM("*String*")	*String* without leading, trailing, or consecutive spaces.
@UPPER("*String*")	*String* in uppercase characters

Table 7.5: Logical functions

Logical functions	Returns
@AND(*LogicalList*)	Yes if all arguments in *LogicalList* are true; otherwise, No.
@IF(*Cond, TrueExpr, FalseExpr*)	*TrueExpr* if *Cond* is Yes; *FalseExpr* if *Cond* is No.
@NOT(*Logical*)	Yes if *Logical* is false; otherwise, No.
@OR(*LogicalList*)	Yes if any argument in *LogicalList* is true; otherwise, No.

Table 7.6: Financial functions

Financial functions	Returns
@CTERM(*Rate, Fv, Pv*)	Number of compounding periods (kept for compatibility; see @NPER).
@DDB(*Cost, Salvage, Life, Period*)	Double-declining depreciation allowance.
@FV(*Pmt, Rate, Nper*)	Future value of an annuity (kept for compatibility; see @FVAL).
@FVAL(*Rate, Nper, Pmt, Pv, Type*)	Future value of an annuity (an improved version of @FV).
@IPAYMT(*Rate, Per, Nper, Pv, Fv, Type*)	Interest portion of a payment amount for a loan.
@IRATE(*Nper, Pmt, Pv, Fv, Type*)	Periodic interest rate (an improved version of @ RATE; see page 94).
@IRR(*Guess, ColumnName*)	Internal rate of return.
@NPER(*Rate, Pmt, Pv, Fv, Type*)	Number of periods (an improved version of @CTERM and @ TERM; see page 94).
@NPV(*Rate, ColumnName, Type*)	Present value of future cash flow.
@PAYMT(*Rate, Nper, Pv, Fv, Type*)	Payment amount for a loan (an improved version of @PMT; see page 94).
@PMT(*Pv, Rate, Nper*)	Payment amount for a loan (kept for compatibility; see @PAYMT).
@PPAYMT(*Rate, Per, Nper, Pv, Fv, Type*)	Principal portion of a payment amount for a loan.
@PV(*Pmt, Rate, Nper*)	Present value of an annuity (kept for compatibility; see @PVAL).
@PVAL(*Rate, Nper, Pmt, Fv, Type*)	The present value of an annuity (an improved version of @PV; see page 94).
@RATE(*Fv, Pv, Nper*)	Periodic interest rate (kept for compatibility; see @IRATE).
@SLN(*Cost, Salvage, Life*)	Straight-line depreciation allowance.
@SYD(*Cost, Salvage, Life, Period*)	Sum-of-the-years'-digits' depreciation allowance of an asset.
@TERM(*Pmt, Rate, Fv*)	Number of payment periods of an investment (kept for compatibility; see @NPER).

Table 7.7: Value miscellaneous functions

Miscellaneous functions	Returns
@BLANK	A blank value to a field.
@CHOOSE(*Number, List*)	The value in *List* in the position of *Number*.
@CURRENTFILE	Returns the name of the current file.
@CURRENTPATH	Returns the name of the current path.
@ERR	Returns the value ERR (error).
@FORMAT(*Num, Format, Places*)	Formats the number so it can be saved or sent to another application in that format.
@ISBLANK(*FieldName*)	Returns Yes if *FieldName* is blank; otherwise, No.
@ISCOMPLETE("*FormName*")	Returns Yes if *FormName* is complete; otherwise, No.
@NA	Returns the value NA (not available).
@SELECTEDFIELD	Returns the name of the currently selected field.
@SELECTEDFORM	Returns the name of the currently selected form.
@TYPE(*Value*)	Returns a number (1, 2, 4, or 16) indicating the data type of *Value*.
@VERSION	Returns the version number of ObjectVision.

Table 7.8: Linking functions

Linking functions	Results
@ASCIIOPEN("*LinkName*", "*FileName*", "*OVFields*", *Option*)	Opens the *LinkName* link between an ASCII file and the ObjectVision *OVFields*.
@BOTTOM("*LinkName*")	Moves to the last record in the open database.
@BTRVOPEN("*LinkName*", "*DictionaryPath*", "*TableName*", "*DataFields*", "*OVReadFields*", "*OVWriteFields*", *IndexNo, Option*)	Opens the *LinkName* link between the Btrieve table *TableName DataFields* and the ObjectVision fields.
@CLEAR("*LinkName*")	Clears the associated link fields in preparation for writing a new record.
@CLOSE("*LinkName*")	Closes and disconnects the associated link fields and dissolves the link.
@DBOPEN("*LinkName*", "*FileName*", "*DataFields*", "*OVReadFields*", "*OVWriteFields*", "*IndexFile*", *Option*)	Opens an dBASE link between the dBASE data file's *DataFields* and ObjectVision's fields.
@DDEEXECUTE("*LinkName*", "[*Commands*]")	Sends commands to a linked DDE application.
@DDEOPEN("*LinkName*", "*Application*", "*Document*", "*RemoteNames*", "*OV Fields*"	Opens a DDE link between *Application Document RemoteNames* and *OVFields*.
@DDEPOKE("*LinkName*", "*RemoteName*", *Value*)	Sends values to a linked DDE application.
@DELETE("*LinkName*")	Deletes current database record.
@FILTERACTIVATE("*LinkName*", "*FilterName*",)	Activates the filter to *LinkName*.
@FILTERDEACTIVATE("*LinkName*")	Deactivates the filter to *LinkName*.
@INSERT("*LinkName*")	Inserts the current ObjectVision values into the data file at the current record.
@LINKVALUE("*DataField*")	Returns the current value of *DataField*.

Table 7.8: Linking functions (continued)

@LOCATE("*LinkName*", "*InexactFlag*", *IndexValues*)	Returns the record in *LinkName* that matches *IndexValues*.
@NEXT("*LinkName*")	Moves to the next data file record in the open link.
@PAGEDN("*LinkName*")	Moves down the database table by one page.
@PAGEUP("*LinkName*")	Moves up the database table by one page.
@PREVIOUS("*LinkName*")	Moves to the previous record in the open database.
@PRINTLINK("*LinkName*")	Prints the current form once for *each* value in the current range of *LinkName*.
@PXOPEN("*LinkName*", "*TableName*", "*DataFields*", "*OVReadFields*", "*OVWriteFields*", "*SecIndexField*", *Option*)	Opens the *LinkName* link between the Paradox *Filename DataFields* and the ObjectVision fields.
@STORE("*LinkName*")	Appends a new record or updates an existing record in the open database.
@TOP("*LinkName*")	Moves to the first record in the open database or ASCII file.
@UPDATE("*LinkName*")	Updates a record in the open database with the current ObjectVision values.

Table 7.9: Menu customization functions

Menu customization function	**Results**
@ADDMENU("*MenuName*", *Position*)	Adds a command (*MenuName*) to the application's menu at *Position*.
@ADDMENUITEM("*MenuName*", "*MenuItemName*", *Position*)	Adds items (*MenuItemName*) to the menu command (*MenuName*).
@CHECKMENUITEM("*MenuName*", "*MenuItemName*")	Places a check mark next to the item.
@DELETEMENU("*MenuName*")	Deletes the menu command *MenuName*.
@DELETEMENUITEM("*MenuName*", "*MenuItemName*")	Deletes the item from the menu command.
@RESTOREMENU	Erases all menu changes and returns to normal ObjectVision menus.
@UNCHECKMENUITEM("*MenuName*", "*MenuItemName*")	Removes the check mark from the item.

Table 7.10: Menu equivalents functions

Menu equivalents function	Results
@APPEXIT	Exits the application without saving.
@APPNEW	Creates a new application.
@APPOPEN("*AppName*")	Opens the application *AppName*.
@FORMSELECT("*FormName*")	Makes *FormName* the selected form.
@FORMCLOSE("*FormName*")	Closes *FormName*.
@FIELDFIND(*FieldName*)	Finds *FieldName* in the current application.
@FIELDCALCULATE(*FieldName*)	Calculates the value of *FieldName*.
@FIELDCLEAR(*FieldName*)	Clears any values in the field.
@PRINTFORM("*FormName*")	Prints *FormName*.
@PRINTALL	Prints all the forms in the application.
@RESUME	Resumes guided completion.
@SAVE	Saves the current application.
@SAVEAS("*FileName*")	Saves the current application as *FileName*.

Table 7.11: Event miscellaneous functions

Event miscellaneous function	Results
@ASSIGN(*FieldName, Value*)	Gives *FieldName* the *Value*.
@EVENT("*Item*", "*Event*")	Sends any event to an *Item*.
@EXEC("*CommandLine*", *ShowOption*)	Opens another Windows application.
@MESSAGE("*String*")	Puts the *String* message in a pop-up dialog box.
@REGISTER("*OVAlias*", "*ArgTypes*", "*ArgHelp*", "*LibName*", "*FuncName*", *Type*)	Registers a new @function.
@SETTITLE("*Title*")	Changes the window title from ObjectVision to *Title*.

Using financial functions

This section describes the relationships among the financial functions.

Table 7.12
How the two forms of financial functions are related

Old style	New style
@CTERM(*Rate, Fv, Pv*)	@NPER(*Rate*, 0, –*Pv, Fv*, 0)
@FV(*Pmt, Rate, Nper*)	@FVAL(*Rate, Nper*, –*Pmt*, 0, 0)
@PMT(*Pv, Rate, Nper*)	@PAYMT(*Rate, Nper*, –*Pv*, 0, 0)
@PV(*Pmt, Rate, Nper*)	@PVAL(*Rate, Nper*, –*Pmt*, 0, 0)
@RATE(*Fv, Pv, Nper*)	@IRATE(*Nper*, 0, –*Pv, Fv*, 0)
@TERM(*Pmt, Rate, Fv*)	@NPER(*Rate*, –*Pmt*, 0, *Fv*, 0)

The arguments for these financial functions are as follows:

Be sure to express Rate and Nper in the same type of time period—for example, if one is months, the other can't be years.

Rate = Interest rate.
Nper = Number of periods (should be an integer greater than 0).
Pv = Present value.
Pmt = Payment.
Fv = Future value.
Type = 0 if payments are at the end of each period (an *ordinary annuity*), 1 if at the beginning (an *annuity due*).

These functions represent a transaction that takes place over *Nper* time periods, which is months (in this description) for simplicity's sake. At the beginning of the first month, you receive *Pv* dollars. Each month, you get *Pmt* dollars. At the end of the last month, you get *Fv* dollars. Depending on *Type*, the payment of *Pmt* dollars is either at the beginning of each month or at the end. One or two of these numbers, *Pv*, *Pmt*, and *Fv*, must be negative, meaning that you will actually pay money rather than receive it.

This formula shows how the financial variables are related; ObjectVision uses a more complicated formula to solve for any one of them.

Under these conditions, the transaction can be interpreted as a loan, with interest compounded monthly. The five variables are related by the formula

$$Pv(1+Rate)^{Nper}+Pmt\left(\frac{(1+RateType)\,(1+Rate)^{Nper-1}}{Rate}\right)+Fv=0$$

and any one variable can be determined from the other four.

Note In all these functions, as well as @NPV and @IRR, amounts (for arguments and function results) with positive signs represent money received, and amounts with negative signs represent money paid. In the old style functions, @PV, @PMT, @FV, @ RATE, @TERM, and @CTERM, the amounts are usually all positive regardless of which way the money changes hands.

Only integer values are allowed for *Nper*. If you borrow money from a bank for 15.2 months with interest paid monthly, giving *Nper* a value of 15.2 in the financial functions will only be a rough indicator of what the bank will tell you to pay. In order to compute the figures the way the bank would, you have to consider two transactions, one for 15 months and one for 0.2 months.

C H A P T E R

8

@Function descriptions

This chapter lists ObjectVision @functions in alphabetical order. Format, use, and examples are given for each function.

Note that all examples in this chapter use the comma as the list separator. You can change this separator in the International section of the Windows Control Panel.

ABS

Format @ABS(*X*)

X = a numeric value.

Use @ABS returns the absolute (positive) value of *X*.

Examples @ABS(Actual Weight – Ideal Weight) = 7, where Actual Weight – Ideal Weight is –7 or 7

@ABS(-100) = 100

@ABS(100) = 100

@ABS(0) = 0

ACOS

2.0

Format @ACOS(*X*)

X = a numeric value between –1 and 1.

Use @ACOS returns the arc cosine of *X*. The result is the angle (in radians) whose cosine is *X*. To convert radians to degrees, use the @DEGREES function (page 119).

Examples @ACOS(1) = 0
@ACOS(0.5) = 1.047198
@DEGREES(@ACOS(0.5)) = 60
@ACOS(X) = ERR (means that X is greater than 1 or less than –1)

ADDMENU

2.0

Format @ADDMENU("*MenuName*", *Position*)

MenuName = the name you want for the menu command. The ObjectVision defaults are File, Edit, Form, Field, View, and Tools.

Position = where you want the command to go. The values begin at 0 for the far left position. To place new menus at the end of the list, use –1.

Use @ADDMENU creates a new menu command. To assign a keyboard shortcut to a menu command, use the & symbol. For instance, the F in File on the menu bar is underlined. Only one letter can be used as a shortcut key.

After you have added the menu command, use @ADDMENUITEM to place items under the command heading.

Examples @ADDMENU("&Quit", 0) = places Quit as a menu command in the far left position.

ADDMENUITEM

2.0

Format @ADDMENUITEM("*MenuName*", "*MenuItemName*", *Position*)

MenuName = the menu command under which the item will go.

MenuItemName = the name of the item you want placed under *MenuName*. If this is blank (""), a separating line will be placed in the list of items.

Position = where you want the item to be placed. The values begin at 0 for the top position. To place new menu items at the bottom of the list, use –1.

Use @ADDMENUITEM adds new items under a menu command. When you select the menu item during form completion mode, you cause a user-defined event of the same name (including punctuation and & symbols) to be sent to the stack. See Chapter 9, "Menu customization."

You don't need to include the & symbol if *MenuName* contains a shortcut key.

Examples @ADDMENUITEM("Quit", "Exit to DOS", 0) = places *Exit to DOS* as the top item under the menu command Quit.

AND

Format @AND(*LogicalList*)

LogicalList = one or more logical values or expressions that evaluate to a logical value, up to a maximum of 14.

Use @AND returns the logical value Yes if all of the arguments evaluate as true; otherwise, it returns the logical value No.

Examples @AND(USCitizen, State Resident) = Yes if both USCitizen and State Resident are Yes; otherwise, it returns No.

@AND(Date<@DATEVALUE("01/01/92"), Policy=Good) = Yes if both arguments are true; otherwise, it returns No.

APPEXIT

Format @APPEXIT

Use This function exits the current application without saving any changes. If you are saving all data to tables by using the Enter button, you can use this function to exit the application quickly. You can also use this with other @functions.

For instance, you could set up your application so that when all forms are complete, they are printed (using @PRINTALL), and then the application is closed (using @APPEXIT).

Examples @APPEXIT = exits ObjectVision.

APPNEW

2.0

Format @APPNEW

Use This function works like File | New from the main menu—it starts a new application. When the new application is opened, the current application (if any) is closed. If you have not saved the current application, ObjectVision will *not* save it before opening the new application.

Examples @APPNEW = closes the current application and creates a new untitled one.

APPOPEN

2.0

Format @APPOPEN("*AppName*")

AppName = the name of the application you want to open. If it is not in the current directory, you must type the entire path name.

Use This function opens an application and closes the current one. You do not need to type the .OVD extension. You can use this function in the same manner as @APPEXIT. Once all forms are complete, you can have ObjectVision automatically exit the current application and open a new one.

Examples @APPOPEN("C:\VISION\APPS\ORDER") = opens the Order application in the directory C:\VISION\APPS.

ASCIIOPEN

Format @ASCIIOPEN("*LinkName*", "*FileName*", "*OVFields*", *Option*)

"*LinkName*" = the name you specify for the link.
"*FileName*" = the name of an ASCII file in the current directory. When the ASCII file is not in the current directory, you can use its complete path name. Data values within the ASCII file must be delimited by commas. Values that

contain commas or double quotes must be surrounded by double quotes. Other values can optionally be surrounded by double quotes.

"OVFields" = a comma-delimited list of ObjectVision field names.
Option = 0=READ, 1=WRITE, or 2=APPEND.

Use

This function is provided for compatibility with ObjectVision version 1.0. You should use the Links Tool for creating ASCII links.

- **READ:** The file is opened for reading, and the first line of values is imported into the *OVFields*.
- **WRITE:** The file is opened for writing at the beginning of the file. Any data already in the file is overwritten. If the file does not exist, it is created.
- **APPEND:** The file is opened for writing to the end of the file. Any data already in the file remains as is. If the file does not exist, it is created.

@ASCIIOPEN opens the ASCII file *FileName* and links it with the ObjectVision fields you specify in *OVFields*. It returns Yes if successful; otherwise, No.

Examples @ASCIIOPEN("CURREAD", "CURSTOCK.ASC","High,Low,Buy,Sell", 0)

This function opens the CURREAD link to the ASCII file CURSTOCK.ASC in the current directory and reads in values for the ObjectVision fields High, Low, Buy, and Sell.

ASIN

2.0

Format @ASIN(*X*)

X = a numeric value between –1 and 1

Use @ASIN calculates the arc sine of *X*. The result is the angle (in radians) whose sine is *X*. To convert radians to degrees, use the @DEGREES function (page 119).

Examples
@ASIN(1) = 1.570796
@ASIN(0.25) = 0.25268
@DEGREES(@ASIN(0.5)) = 30
@ASIN(X) = ERR (means X is greater than 1 or less than –1)

ASSIGN

2.0

Format @ASSIGN(*FieldName, Value*)

FieldName = the name of the field receiving the assigned value.
Value = the value to give to *FieldName*.

Use This function assigns a value to *FieldName*. The value can be specified in the function, or it can be retrieved from another field.

If you use @ASSIGN to give a value to a field that has a value tree, it may appear to a user that the value tree isn't working because @ASSIGN gives it another value (see the second example below).

Examples @ASSIGN(Years of Service, ((@NOW – Date Hired)/365.25))

The above example assigns the value of the expression (@NOW – Date Hired/365.25) to the field Years of Service.

@ASSIGN(Field1, Field1+1) = increments the value in Field1 by one.

ATAN

2.0

Format @ATAN(*X*)

X = a numeric value.

Use @ATAN calculates the arc tangent of *X*. The result is the angle (in radians) whose tangent is *X*. To convert radians to degrees, use the @DEGREES function (page 119).

Examples @ATAN(0.5) = 0.463648
@ATAN(1) = 0.785398
@DEGREES(@ATAN(1)) = 45

ATAN2

2.0

Format @ATAN2(*X,Y*)

X = a numeric value.
Y = a numeric value.

Use @ATAN2 calculates the arc tangent of the angle represented by the point with (x,y) coordinates *X* and *Y*. The result is the angle (in radians) whose

tangent is Y/X. The result is between $-\pi$ and π, with the quadrant chosen appropriately.

If either X or Y is 0, the result is ERR.

Caution The order of arguments is the opposite of the ATAN2 function in FORTRAN and other programming languages.

To convert radians to degrees, use the @DEGREES function (page 119).

Examples @ATAN2(1,2) = 1.107149
@DEGREES(@ATAN2(1,1)) = 45

AVG

2.0

Format @AVG(*List*)

List = between 1 and 14 numeric values.

Use @AVG calculates the average of all values in *List*. Multiple values must be separated by commas. If any of the referenced values contain ERR, the resulting value is ERR.

Examples @AVG(Field1, Field2, Field3) = the same as adding all three fields together, then dividing by three.

@AVG(10, 12, 8, 9, 1) = 8

@AVG(1.104, 2.346, 34.58, 23.54, 15.465) = 15.407

BLANK

Format @BLANK

Use @BLANK, as a value tree conclusion, assigns a blank (null) value to a field, which lets you clear a field after an invalid value has been entered.

Examples The following example checks if a field value is too short. If it is, then the field becomes blank, and the user must type another value.

Condition: @LENGTH(Field1)<2

Conclusion: @BLANK

BOTTOM

Format @BOTTOM("*LinkName*")

"*LinkName*" = a link name to a database table.

Use *If the link is Auto Insert or Auto Update, an insert or update is attempted before the @BOTTOM is completed.*

@BOTTOM repositions the "*LinkName*" database link to the last record in the database table. The link then delivers values to ObjectVision through read connections. If the link is a primary link to a table object, the table object currency is the last row containing values. "*LinkName*" must be the name of an open database link. Typically, you'd use this function with a button when you create a link to a database table.

Examples @BOTTOM("ORDER")

BTRVOPEN

Format @BTRVOPEN("*LinkName*", "*Dictionary*", "*TableName*", "*DataFields*", "*OVReadFields*", "*OVWriteFields*", *IndexNo, Option*)

"*LinkName*" = the name you specify for the link and use to reference the link in all subsequent functions.

"*Dictionary*" = the path name of the XQL files (FIELD.DDF, FILE.DDF, and INDEX.DDF) that contain the field definitions for *TableName*.

"*TableName*" = the name of a valid Btrieve table name contained in Dictionary. When the file is not in the current directory, you can use its complete path name.

"*DataFields*" = contains the names of the database fields, delimited by commas, to be connected to your ObjectVision fields.

"*OVReadFields*" = contains the names of your ObjectVision fields, delimited by commas, to hold imported values from *FileName DataFields*. There must be a one-to-one correspondence between the *DataFields* and your ObjectVision fields.

"*OVWriteFields*" = contains the names of your ObjectVision fields, delimited by commas, to write their values to *FileName DataFields*. There must be a one-to-one correspondence between the file *DataFields* and your ObjectVision fields.

IndexNo = specifies which predefined indexing identifier to use (an existing index from 0 through 23) as defined in the dictionary.

Option = specifies how closely you want to match the index value. Use 0 for an exact match, 1 for an inexact match.

Use

This function is provided for compatibility with ObjectVision version 1.0. You should use the Links Tool for creating Btrieve links.

@BTRVOPEN opens the Btrieve file identified by the XQL table *TableName* and links it with the ObjectVision fields. @BTRVOPEN returns Yes if successful; otherwise, No.

Btrieve links require a dictionary that defines field names, locations, sizes, and types for fields in the Btrieve file. The dictionary also defines indexes for the data table.

ObjectVision uses the same dictionary structure Novell XQL uses. If you are linking to a preexisting Btrieve file, use XQL or other Novell products to create a dictionary if one does not exist.

Examples

@BTRVOPEN("Patients", "C:\Btrieve", "Patients", "ID, First Name", "ID, First", "ID, First", "0")

CHAR

Format

@CHAR(*Code*)

Code = a numeric value from 1 to 255, inclusive.

Use

CHAR returns the onscreen character corresponding to the given code. Refer to Appendix D, "The ANSI character set," for a complete list of characters and their corresponding codes.

Examples

@CHAR(33) = !
@CHAR(34) = "
@CHAR(35) = #
@CHAR(36) = $
@CHAR(75) = K

CHECKMENUITEM

2.0

Format @CHECKMENUITEM("*MenuName*", "*MenuItemName*")

"*MenuName*" = the name of the menu command on the main menu.
"*MenuItemName*" = the name of the item under the menu command.

Use This function places a check mark next to *MenuItemName* in the list of items under the command in the main menu. You would use this to show a user that the menu item has been selected. To remove the check, use @UNCHECKMENUITEM.

Examples @CHECKMENUITEM("File", "New") = places a check mark next to New under the File command.

CHOOSE

2.0

Format @CHOOSE(*Number, List*)

Number = the position of the item in *List*.
List = a list of items counted from left to right.

Use This function holds a list of items from which one item will be chosen based on *Number*. *Number* is typically a variable, perhaps from the value in a field. If there isn't an item in *List* that corresponds to *Number*, then ERR is returned.

Examples @CHOOSE(Field1, "Johnson", "Smith", "Kirby", "Murray", "Larson") = "Johnson" if the value in Field1 is 0, "Smith" if Field1 is 1, "Kirby" if 2, and so on.

@CHOOSE(@WEEKDAY(@NOW)-1,"Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday") = gives the day of the current week.

CLEAR

Format @CLEAR("*LinkName*")]

"*LinkName*" = the name of the link to the ASCII file or the Paradox, dBASE, or Btrieve database table.

Use This function clears the ASCII, Paradox, dBASE, or Btrieve link fields associated with the "*LinkName*" link. You'd use this function to prepare for writing a new record or for "what if" types of calculations.

Because the link position is undefined after you use @CLEAR, an error message appears if @PREVIOUS, @NEXT, @PAGEUP, @PAGEDN, or @UPDATE is executed afterward. To define a position on the database table, use @TOP, @BOTTOM, @LOCATE or an Auto Locate option on the link.

"*LinkName*" must be the name of a link previously opened with Tools | Links.

You might want to place a button field in your application forms to make it easy for the user to clear the form of linked values.

@CLEAR returns Yes if successful; otherwise, No.

Examples @CLEAR("ORDER"), as used in the *Credit* sample application. This function clears the fields in the ORDER.DB database through the open database link ORDER.

CLEARALL

2.0

Format @CLEARALL

Use This function works like Edit | ClearAll—it clears all the values in an application. Calculated fields that use constant values are not cleared (such as a field with @NOW as its value tree).

CLOSE

Format @CLOSE("*LinkName*")

"*LinkName*" = an ASCII, Paradox, dBASE, Btrieve, or DDE data file link name.

Use @CLOSE clears the read connected fields, clears the ObjectVision values, and dissolves the "*LinkName*" link.

"*LinkName*" must be the name of a link previously opened with Tools | Links. Any fields containing values unrelated to links retain their current values.

@CLOSE returns Yes if successful; otherwise, No.

Examples @CLOSE("CURSTOCK.ASC"), closes the ASCII link CURSTOCK.ASC and disconnects the link from the associated ObjectVision fields.

@CLOSE("ORDER"), as used in the *Credit* sample application, disconnects the ORDER.DB database from the associated ObjectVision fields using the database link ORDER.

CODE

Format @CODE("*String*")

"*String*" = a string value.

Use @CODE returns the ANSI code of the first character in *String*. This is the opposite of the @CHAR function, which returns the character corresponding to the given code. If *String* is the empty string, ERR is returned.

Examples
@CODE("!") = 33
@CODE("#") = 35
@CODE("$") = 36
@CODE("?") = 63
@CODE(Hello) = 72 (When Hello is a field name with a value = Handicapped)
@CODE("Sam") = 83 (code for *S*)
@CODE(Marital Status) = 87, (*W*, when the field value = Widowed)

COLUMNAVG

2.0

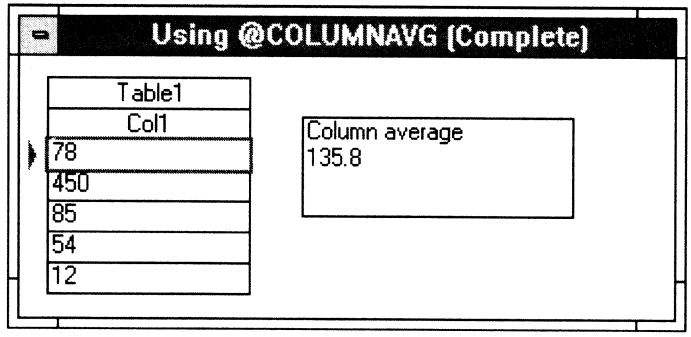
Format @COLUMNAVG(*ColumnName*)

ColumnName = the name of the column to average. It must be a numeric type.

Use This function calculates the average of all the values visible in the column. This function is different from most spreadsheets. If a column has four cells but only three values (one cell has no value), the sum will be divided by three. However, if you press *Enter* in the empty cell, which gives the cell the null value, @COLUMNAVG divides by four.

Examples @COLUMNAVG(Col1) = 135.8

Figure 8.1
Average of Col1



COLUMNCOUNT

2.0

Format @COLUMNCOUNT(*ColumnName*)

ColumnName = the name of the column to count.

Use This function counts the number of visible values in the column. A blank cell isn't counted. Note that a field can appear blank, but contain the null value. A cell can get a null value if you press *Enter* in the cell without typing a value.

Examples @COLUMNCOUNT(Col1) = 5 (See Figure 8.1.)

COLUMNMAX

2.0

Format @COLUMNMAX(*ColumnName*)

ColumnName = the name of the column from which this function gets the maximum value. It must be a numeric type.

Use This function returns the maximum visible value in the column.

Examples @COLUMNMAX(Col1) = 450 (See Figure 8.1.)

COLUMNMIN

2.0

Format @COLUMNMIN(*ColumnName*)

ColumnName = the name of the column from which this function gets the minimum value. It must be a numeric type.

Use This function returns the minimum visible value in the column.

Examples @COLUMNMAX(Col1) = 12 (See Figure 8.1.)

The above example gets its minimum value from the fields in Col1 in the example for @COLUMNAVG.

COLUMNSUM

2.0

Format @COLUMNSUM(*ColumnName*)

ColumnName = the name of the column containing the values to be added. The column must be a numeric type.

Use This function adds all visible values in the column.

Examples @COLUMNSUM(Col1) = 679 (See Figure 8.1.)

COS

2.0

Format @COS(*X*)

X = a numeric value.

Use @COS returns the cosine of the angle *X*. *X* must be given in radians, not degrees. To convert degrees to radians, use the @RADIANS function (page 159).

Examples
@COS(@RADIANS(60)) = 0.5
@COS(@RADIANS(75)) = 0.258819
@COS(@RADIANS(45)) = 0.707107
@COS(@PI/3) = 0.5

CTERM

2.0

Format @CTERM(*Rate, Fv, Pv*)

Rate = a numeric value representing the fixed interest rate per compounding period.
Fv = a numeric value representing the value the investment will reach at some point.
Pv = a numeric value representing the present value of the investment.

Use @CTERM calculates the number of time periods required for an investment of *Pv* to reach a value of *Fv*, while earning interest of *Rate* per compounding period.

An equivalent for this formula using @NPER (see page 147) is

@NPER(*Rate*, 0, –*Pv*, *Fv*, 0)

@CTERM assumes that the investment is an ordinary annuity. @NPER, which is calculated differently than but is related to @CTERM, lets you use an argument, *Type*, to indicate whether the investment is an ordinary annuity or an annuity due.

Examples Assuming that your savings account has an annual interest rate of 7% , how long would it take a $3000 deposit to reach $5000? The answer is

@CTERM(7% ,5000,3000) = 8 years

Note that if the *Rate* figure is cast in terms of years, the result is in years as well. If the interest were compounded monthly, you'd enter `7% /12` to divide the answer by 12 and get a monthly figure.

You can also use the more sophisticated @NPER to solve this problem:

@NPER(7% ,0,-3000,5000,0) = 8

Other examples:

@CTERM(0.07,5000,3000) = 8
@CTERM(0.10,5000,3000) = 6
@CTERM(0.12,5000,3000) = 5
@CTERM(0.12,10000,7000) = 4

CURRENTFILE

2.0

Format @CURRENTFILE

Use This function returns the name and extension of the current open file. If the file hasn't been saved, this function returns the null string.

Examples @CURRENTFILE = *ORDER.OVD* if you are currently working with the *Order* application that came with ObjectVision.

CURRENTPATH

2.0

Format @CURRENTPATH

Use This function returns the path of the currently active ObjectVision application. If the file hasn't been saved, this function returns the null string.

Examples @CURRENTPATH = C:\VISION\ if you are currently working with an application that was installed with the ObjectVision default directory.

DATE

Format @DATE(*Yr,Mo,Day*)

Yr = a numeric value from 1800 to 2099 (or from 0 to 199 for Quattro Pro compatibility).
Mo = a numeric value from 1 to 12.
Day = a numeric value from 1 to 31.

Use @DATE returns the serial number of the date specified with year, month, and day arguments. This serial number can range from –36522 to 73050, and represents dates from January 1, 1800 to December 31, 2099. Serial number 0 represents December 30, 1899.

The fractional portion of a date serial number is used by the time functions. The *Mo* and *Day* values can be outside of the above range to represent displacements from the current month or day.

Examples @DATE(1987,1,1) = 31778
@DATE(1987,9,13) = 32033
@DATE(1900,1,1) = 2
@DATE(1991,6,19 – Lead Time) = 33394, (representing June 5, 1991 where the value of the Lead Time field is 14).

DATEVALUE

Format @DATEVALUE("*DateString*")

"*DateString*" = a string value in any valid date format, enclosed by quotes.

Use @DATEVALUE returns a serial date value that corresponds to the value in *DateString*. This serial number can range from –36522 to 73050, and represents the number of days from December 30, 1899 up to the date referenced in the expression. The earliest date available is January 1, 1800.

The / (slash) is interchangeable with the – (hyphen) when you enter the *DateString*. Trailing or leading spaces are ignored. There are five valid formats for *DateString*, assuming U.S. defaults have been set in the Windows Control Panel:

- DD-MMM-YY ("04-Jul-87")
- DD-MMM ("04-Jul") (assumes the current year)

- MMM-YY ("Jul-87") (assumes the first of the month)
- MM/DD/YY ("11/9/95") (the short date format in the Windows Control Panel)
- MMMM D, YYYY ("January 1, 1992") (the long date format in the Windows Control Panel)

If the value in *DateString* is in an incorrect format or represents an impossible date, an ERR value is returned. The same date formats entered in a field can be used for the *DateString* argument.

Examples

@DATEVALUE("04-May-87") = 31901
@DATEVALUE("04-May") = 31901
@DATEVALUE("JUL-86") = 31594
@DATEVALUE("07/04/87") = 31962
@DATEVALUE("July 1, 1986") = 31594

DAY

Format @DAY(*DateTimeNumber*)

DateTimeNumber = a number in the range –36522 (January 1, 1800) to 73050 (December 31, 2099).

Use @DAY returns the day portion of the *DateTimeNumber*. The result of this function is an integer from 1 to 31. Decimal portions of *DateTimeNumber* represent the time but are ignored for the day computation.

Examples @DAY(31779) = 2 from the converted date 1/2/87.

@DAY(32134) = 23 from the converted date 12/23/87.

@DAY(73051) = ERR because the number you entered was larger than 73050.

@DAY(Date) = 1, when the Date field value is the date/time serial number for 1/Feb/1990.

DBOPEN

Format @DBOPEN("*LinkName*", "*FileName*", "*DataFields*", "*OVReadFields*", "*OVWriteFields*", "*IndexFile*", *Option*)

"*LinkName*"	=	name you specify for the dBASE-compatible database link.
"*FileName*"	=	name of the dBASE-compatible database file (.DBF). If this file isn't in your path, then give the path name of the file as well.
"*DataFields*"	=	names of the database fields, delimited by commas, to be connected to your ObjectVision fields.
"*OVReadFields*"	=	names of your ObjectVision fields, delimited by commas, to hold imported values from *FileName DataFields*. There must be a one-to-one correspondence between the *DataFields* and your ObjectVision fields.
"*OVWriteFields*"	=	names of your ObjectVision fields, delimited by commas, to write their values to *FileName DataFields*. There must be a one-to-one correspondence between the database and your ObjectVision fields.
"*IndexFile*"	=	name of a dBASE-compatible index file (.NDX). The .NDX file must be in the same directory as the .DBF database file unless you specify the complete path name. ObjectVision uses the index file, if present, to order and locate the database records.
Option	=	Use 0 for an exact match, 1 for an inexact match. Inexact indicates that you want to locate the record that has the same beginning characters as the index value.

Use @DBOPEN opens a named, bidirectional database link between *FileName* (a dBASE file) and your application.

@DBOPEN returns Yes if successful; otherwise, No.

This function is provided for compatibility with ObjectVision version 1.0. You should use the Links Tool for creating dBASE links.

Examples @DBOPEN("DBORDER","DBORDER.DBF", "Term,Auth","Credit Terms,Credit Authorization","Credit Terms,Credit Authorization","DBORDER.NDX", 0)

This example opens a write link, named DBORDER, to the DBORDER.DBF database. The database fields are Term and Auth, and the ObjectVision fields are Credit Terms and Credit Authorization.

DDB

2.0

Format @DDB(*Cost,Salvage,Life,Period*)

Cost	=	a numeric value representing the amount paid for an asset.
Salvage	=	a numeric value representing the worth of an asset at the end of its useful life.
Life	=	a numeric value representing the expected useful life of an asset.
Period	=	a numeric value representing the time period for which you want to determine the depreciation expense.

The following must be true:

$Life \geq Period \geq 1$
Life and *Period* must be integers
$Cost \geq Salvage \geq 0$

Use @DDB determines accelerated depreciation values for an asset, given the initial cost, life expectancy, end value, and depreciation period. It calculates depreciation using the double-declining balance method.

Depreciation value (*DDB*) and book value (*BV*) are calculated by:

$BV := Cost$
$DDB := 2BV/Life$
$BV := BV - DDB$

DDB is adjusted, if necessary, to ensure that $DDB \geq 0$, and the subsequent calculation of *BV* has $BV \geq Salvage$.

Examples Suppose you just bought a new $4000 computer. Your dealer says you can sell it back to the store for $350 after eight years, but no one would want to buy it after that. In other words, the *Salvage* value of that computer is $350 and its *Life* is 8. To calculate what the depreciation allowance of this

computer (according to the double-declining balance method) will be by the second year, enter this formula:

@DDB(4000,350,8,2)

The result is $750.

The following examples show depreciation values for the first five years of a $15,000 investment with a salvage value of $3000 and a life of 10 years:

@DDB(15000,3000,10,1) = $3,000
@DDB(15000,3000,10,2) = $2,400
@DDB(15000,3000,10,3) = $1,920
@DDB(15000,3000,10,4) = $1,536
@DDB(15000,3000,10,5) = $1,228.80

DDEEXECUTE

2.0

Format @DDEEXECUTE("*LinkName*", "*[Commands]*")

"*LinkName*" = the name of the open DDE link.
"*[Commands]*" = commands the DDE application will recognize and execute.

Use This function sends commands to the linked DDE application. That application will then execute the commands it is given. For the syntax of the command string, you'll need to refer to that application's documentation.

If you're sending commands to another ObjectVision application, the syntax rules are the same as the rules for event tree conclusions except that each line must be enclosed in [ ].

Examples @DDEEXECUTE("LinktoOV", ""[@DDEOPEN(""1to2"", ""VISION"", ""TWO.OVD"", ""data2"", ""data1"")]")

The example above sends the DDE open command to a linked ObjectVision application.

@DDEEXECUTE("LinktoOV","[@ASSIGN(Field1, ""new value"")] [@FORMSELECT(""Form 1"")]")

The example above sends a command to a linked ObjectVision application that causes the application to assign the value "new value" to Field1, then selects the form "Form 1." Note that two double quotes surround "new

value" because it is a string embedded in the another string—the *Command* string.

DDEOPEN

Format @DDEOPEN("*LinkName*", "*Application*", "*Document*", "*RemoteNames*", "*OVFields*")

"*LinkName*"	=	name you specify for the DDE link.
"*Application*"	=	name of a Windows application without the three-character extension. If the application isn't in the path, you must type the entire path name.
"*Document*"	=	name of a document file to be referenced in the *Application*. If the file isn't in the current directory, you must type the entire path name.
"*RemoteNames*"	=	names of the data fields in the *Document*, delimited by commas. There must be a one-to-one correspondence between the DDE *RemoteNames* and your ObjectVision *OVFields*.
"*OVFields*"	=	names of the ObjectVision fields, delimited by commas, that you want to link to the *RemoteNames*. There must be a one-to-one correspondence between the DDE *RemoteNames* and the ObjectVision fields.

Use @DDEOPEN opens a named, hot DDE link between a Windows *Application* and your application. It returns Yes if successful; otherwise, No.

ObjectVision automatically displays new values for the connected fields whenever data values change in the linked document.

Examples @DDEOPEN("INTLTAX","Excel", "EXCISE.XLS","Country,Curtax", "Country of Origin,Current Tax")

This example opens a DDE link named INTLTAX to the Windows application Excel and the document file EXCISE.XLS. The Excel data fields are Country and Curtax, and the ObjectVision fields are Country of Origin and Current Tax.

DDEPOKE

2.0

Format @DDEPOKE("*LinkName*", "*RemoteName*", *Value*)

"*LinkName*"	=	the name you specified for the DDE link.
"*RemoteName*"	=	the name of the cell or place in the remote application where you want to put *Value*.
Value	=	the value or string you want to send to the linked DDE application.

Use This function sends a value to a linked DDE application. You can send information from one ObjectVision application to another. Since the link must be active before you can send information, you don't need to specify what the Windows application is—it's specified with the opened link.

Examples @DDEPOKE("LinktoOV", "Field1", "The link works.")
@DDEPOKE("LinktoSpreadsheet", "A1", "The link works.")

DEGREES

2.0

Format @DEGREES(*X*)

X = a numeric value representing radians.

Use @DEGREES converts the given number of radians to degrees, using the following formula:

$$\frac{180}{\pi} X$$

Examples @DEGREES(0.5) = 28.64789
@DEGREES(@PI/2) = 90
@DEGREES(Field1) = 0.974028 If the value in Field1 is 0.017.

DELETE

Format @DELETE("*LinkName*")

"*LinkName*" = an open link to a database file.

Use @DELETE deletes the record at the current location in the database file using the "*LinkName*" link. ObjectVision is unable to display a record that has been deleted (or marked for deletion, in the case of dBASE databases).

"*LinkName*" must be the name of an opened database link.

@DELETE returns Yes if successful; otherwise, No.

Examples @DELETE("ORDER") This function deletes the current record in the database table connected with the ORDER link.

DELETEMENU

2.0

Format @DELETEMENU("*MenuName*")

"*MenuName*" = the name of the menu command you want to delete.

Use @DELETEMENU deletes a menu command. You can use this to delete the standard ObjectVision menus, then use @ADDMENU to create your own menus.

You don't need to delete all menu items before you can delete the menu name. If you delete the menu name, all menu items below it are deleted as well.

See @RESTOREMENU for information on restoring the ObjectVision default menus.

Examples @DELETEMENU("File") = deletes File from the main menu.

DELETEMENUITEM

2.0

Format @DELETEMENUITEM("*MenuName*", "*MenuItemName*")

"*MenuName*" = the menu command the item is under.
"*MenuItemName*" = the name of the item you want delete.

Use @DELETEMENUITEM deletes items under a menu command.

Examples @DELETEMENUITEM("File", "New") = deletes New under File in the main menu. You cannot create new files now while using this application.

ERR

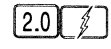
Format @ERR

Use @ERR returns the error value ERR.

The ERR value resulting from this function is used most often with the @IF function, to bring attention to error conditions.

Examples @ERR = ERR
@IF(Field1>Field2,0,@ERR) = 0 (if Field1>Field2) or ERR (if Field1<=Field2)

EVENT



Format @EVENT("*Item*", "*Event*")

"*Item*" = an object, form, or stack to receive the event.
"*Event*" = the event you want to send to *Item*.

Use This function sends a user-defined event to an *Item*. It also can send any ObjectVision event to an object (such as Click, or Select). The *Item* should have an event tree with *Event* as a condition; otherwise, the @EVENT function won't start an action. If *Item* is left blank, the event will be sent to the stack because ObjectVision looks for *Item* first as a field, then as a form. If *Item* isn't found, the event is sent to the stack.

Examples @EVENT("stack", "open")

The above example sends the Open event to the application stack.

@EVENT("Field1","Ohyeah")

The above example sends the user-defined event Ohyeah to Field1. Field1 should have Ohyeah as an event tree condition so that when this event is sent, an action is performed.

EXACT

Format @EXACT("*String1*", "*String2*")

"*String1*" = a valid string value.
"*String2*" = a valid string value.

Use @EXACT compares the values of *String1* and *String2*. If the values are *exactly* identical, including capitalization and diacritical marks (such as ~), it returns Yes. If there are any differences, it returns No.

To compare strings or cell contents without regard to capitalization or diacritical marks, use the = operator. For example, *+Field1=Field2* returns *Yes* if the contents of the fields are the same but are capitalized differently.

Examples @EXACT("client","Client") = No
@EXACT("client","client") = Yes
@EXACT(29,"29") = Yes
@EXACT(Field1,"yes") = Yes (if Field1 contains the string value *yes*)
@EXACT(client,client) = Yes

EXEC

Format @EXEC("*CommandLine*", *ShowOption*)

"*CommandLine*" = the program you want to execute. If the program isn't in your DOS path, you'll need to type the entire path to the program.
ShowOption = how you want the program window to be displayed:

0 = the program's normal size
1 = minimized—only the icon displays
2 = maximized—the program uses the entire screen

Use You use this function to open other Windows applications.

Examples @EXEC("NOTEPAD", 0) opens the Notepad.
@EXEC("CLOCK", 1) opens the Clock, but minimizes it.

EXP

2.0

Format @EXP(*X*)

X = a numeric value ≤43.

Use @EXP returns the mathematical constant *e*, raised to the *X*th power. This function is the inverse of a natural logarithm, @LN (page 141).

Examples @EXP(3.4) = 29.9641000474
@EXP(1) = 2.718281828459 (the actual value of *e*)
@SQRT(@EXP(2)) = 2.71828183
@LN(@EXP(2.5)) = 2.5

FIELDCALCULATE

2.0

Format @FIELDCALCULATE(*FieldName*)

FieldName = the name of the field to calculate.

Use This function works like Field | Calculate from the main menu except instead of calculating the selected field, it calculates the field you specify in *FieldName*.

If the field you want to calculate needs a value from another field, you'll be prompted for that field's value first.

If you want to use the value in a field, you must enclose the field name in parentheses to get the value in that field.

Examples @FIELDCALCULATE(Total Cost) = calculates the value for Total Cost.

@FIELDCALCULATE((Calculate Field)) = calculates the field whose name is currently the value of the field Calculate Field.

FIELDCLEAR

2.0

Format @FIELDCLEAR(*FieldName*)

FieldName = the name of the field to clear.

Use This function clears the field you specify in *FieldName*. The value of *FieldName* can also come from another field.

If you want to use the value in a field, you must enclose the field name in parentheses to get the value in that field.

Examples @FIELDCLEAR(Name) = clears the value for the field Name.

@FIELDCLEAR((Field to Clear)) = clears any field depending on which field name is currently the value in the field Field to Clear.

FIELDFIND

2.0

Format @FIELDFIND(*FieldName*)

FieldName = the name of the field to find.

Use This function works like Field | Find from the main menu. If you want to use the value in a field, you must enclose the field name in parentheses to get the value in that field.

Examples @FIELDFIND(Total Cost) = finds the field named Total Cost.

@FIELDFIND((Field to Find)) = finds any field depending on which field name is currently the value of the field Field to Find.

FILTERACTIVATE

2.0

Format @FILTERACTIVATE("*LinkName*", "*FilterName*")

"*LinkName*" = the name of the link to the table on which you want to activate a filter.
"*FilterName*" = the name of the filter to activate.

Use This function can be used to activate a filter instead of activating the filters using the Links Tool. By using this function and @FILTERDEACTIVATE with

event trees you can switch the filter on and off during form completion mode.

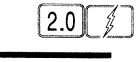
Examples @FILTERACTIVATE("Customer", "Filter 1")

This example uses the Customer link to a database of customer profiles and filters for records that match the customer name entered in the field Company. If you typed Borland in the field Company, only records for Borland will be passed through the filter.

When you want to filter customers based on another filter expression, you could activate another filter by using this function.

When a filter is activated, an @TOP is automatically performed if the filter option Auto is selected.

FILTERDEACTIVATE



Format @FILTERDEACTIVATE("*LinkName*")

"*LinkName*" = the name of the link to the database table on which you want to deactivate a filter.

Use This function deactivates a filter. Since only one filter can be active on a link, the FilterName is not needed.

Examples @FILTERDEACTIVATE("Customer")

This example deactivates the filter on the "Customer" link.

FIND

Format @FIND("*Substring*", "*String*", *StartNumber*)

"*Substring*" = a valid string value, representing the value to search for.

"*String*" = a valid string value, representing the value to search through.

StartNumber = a numeric value ≥ 0, representing the character position at which to begin searching.

Use @FIND searches through the given *String* for the value given as *Substring*. If it finds *Substring*, it returns the character position at which the first occurrence was found. *StartNumber* begins the search at that number of characters into the string. 0 = the first character in the string, 1 = the second, and so on.

@FIND is case-sensitive and is also sensitive to diacritical marks used in non-English languages. You can overcome the case-sensitivity of this function by using @UPPER to force one or more of the strings into all caps; for example,

@FIND(@UPPER(FieldA),@UPPER(FieldB),0)

forces both the substring in FieldA and the string in FieldB to uppercase, then searches for the substring.

@FIND is most often used in conjunction with two other string functions: @REPLACE (to perform search-and-replace operations on strings) and @MID (to access substrings).

If @FIND fails to find any occurrences of *Substring*, or if the *StartNumber* given is less than 0 or greater than the length of *String*, the result is ERR.

Examples
@FIND("i","find",0) = 1
@FIND("nd","find",2) = 2
@FIND("F","find",0) = ERR
@FIND("f","find",3) = ERR
@FIND("d","find",4) = ERR

FORMAT

2.0

Format @FORMAT(*Num*, *Format*, *Places*)

Num = the number to format.
Format = a numeric value (0-5) that designates how you want to format the number. The numbers correspond with field types as shown below:

0 = General
1 = Fixed
2 = Percent
3 = Financial
4 = Currency
5 = Date/time

Places = the number of decimal places for *Num*, or in the case of dates and times, how you want the date/time number to display:

0 = 8/1/91 (Windows short format)
1 = August 1, 1991 (Windows long format)
2 = 1-Aug-91
3 = 1-Aug
4 = Aug 91
5 = 3:15 pm
6 = 3:15:45 pm
7 = 15:15
8 = 15:15:45
9 = 8/1/91 15:15 (Windows short format)

Use This function is useful for sending formatted values to other applications.

Examples @FORMAT(0.00078,2,3) = 0.078%
@FORMAT(49.95,4,2) = $49.95
@FORMAT(32000,5,2) = 4-Oct-08

FORMCLEAR

Format @FORMCLEAR("*FormName*")

"*FormName*" = the name of the form you want to clear.

Use This function works like Form | Clear from the main menu except instead of clearing the current form, it clears the form that you specify in *FormName*.

If you want to use the value in a field as the *FormName*, you must enclose the field name in parentheses to get the value in that field.

Examples @FORMCLEAR("Sales Order") = clears the Sales Order form.

@FORMCLEAR(Form to Clear) = clears the form named in the field Form to Clear.

@FORMCLEAR(@SELECTEDFORM) = clears the currently selected form.

FORMCLOSE

2.0

Format @FORMCLOSE("*FormName*")

"*FormName*" = the name of the form you want to close. It must be enclosed in double quotes.

Use This function closes a form. It works the same as double-clicking the control menu box for the form. You can use this function to automatically close a form once it is complete.

Examples @FORMCLOSE((Form to Close)) = closes the form in the field Form to Close.
@FORMCLOSE("Form 1") = closes Form 1.

FORMSELECT

2.0

Format @FORMSELECT("*FormName*")

"*FormName*" = the name of the form you want to select.

Use This function selects a form. It works identically to Form | Select from the main menu.

Examples @FORMSELECT(Form to Select) = selects the form in the field Form to Select.
@FORMSELECT("W-4 Form") = selects the W-4 Form.

FV

2.0

Format @FV(*Pmt, Rate, Nper*)

Pmt = a numeric value representing the amount of the periodic payment.
Rate = a numeric value representing the periodic interest rate.
Nper = number of periods, which should be a positive integer.

Use @FV returns the future value of an investment where *Pmt* is invested for *Nper* periods at the rate of *Rate* per period.

An equivalent for this formula using @FVAL (see page 129) is

@FVAL(*Rate, Nper, –Pmt*, 0, 0)

@FV assumes that the investment is an ordinary annuity. @FVAL, which is calculated differently than but is related to @FV, lets you use an argument,

Type, to indicate whether the investment is an ordinary annuity (0) or an annuity due (1).

Examples Assume you want to set aside $500 at the end of each year in a savings account that earns 15% annually. To determine what the account will be worth at the end of six years, enter this formula:

@FV(500,15% ,6)

Your yearly payment of $500 will be worth $4,376.87 in six years.

You could also use the more powerful function @FVAL in this way:

@FVAL(15% ,6,-500,0,0)

Note that in this improved version of @FV, you have to be precise about whether a payment is out of your pocket (a negative number) or paid to you (a positive number).

Other examples:

@FV(200,.12,5) = $1,270.57
@FV(500,0.9,4) = $6,684.50
@FV(800,0.9,3) = $5,208.00

FVAL

2.0

Format @FVAL(*Rate, Nper, Pmt, Pv, Type*)

Rate = a numeric value representing the periodic interest rate.
Nper = number of periods, which should be an integer > 0.
Pmt = a numeric value representing the amount of the periodic payment.
Pv = present value.
Type = 0 if payments are at the end of each period, 1 if they are at the beginning.

Use @FVAL is an updated version of the function @FV (see the FV entry).

Be sure to enter a negative number for money that's out of your pocket and a positive number for money that's coming to you.

Examples Putting money into an account before it earns its interest for that year means you have an annuity due to you, which increases the future value calculations.

Suppose you want to set aside $500 at the *start* of each year in a savings account that earns 15% annually. To determine what the account will be

worth at the end of six years, starting at a present value of zero, enter this formula:

@FVAL(15% ,6,-500,0,1)

Your yearly payment of $500 will be worth $5,033.40 in six years, or $656.53 more than if you deposited the money at the last day of the year as in the previous @FV example.

If the account already had $340 in it before your yearly deposits of $500, calculate the future value after six years with this formula:

@FVAL(15% ,6,-500,-340,1) = $5,819.84

Note that in this improved version of @FV, you must be more precise about whether a payment is out of your pocket (a negative number) or paid to you (a positive number).

HOUR

Format @HOUR(*DateTimeNumber*)

DateTimeNumber = a numeric value from –35522 to 73050.99999, representing a date/time serial number.

Use @HOUR returns the hour portion of *DateTimeNumber*. *DateTimeNumber* must be a valid date/time serial number. Because only the decimal portion of a serial number evaluates to a time, the integer portion of the number is disregarded. The result is a number from 0 (12:00 a.m.) to 23 (11:00 p.m.).

To return standard hours (1–12) instead of military hours (1–24), use the @MOD function (page 145) with a parameter of 12.

(See also @TIME on page 172 and @TIMEVALUE on page 172.)

Examples
@HOUR(.25) = 6
@HOUR(.5) = 12
@HOUR(.75) = 18
@HOUR(@TIMEVALUE("10:08am")) = 10
@MOD(@HOUR(@TIMEVALUE("9:31:52 PM")),12) = 9
@HOUR(Time of Occurrence) = 11, when the Time of Occurrence field value is 11:30 a.m.

IF

Format @IF(*Cond, TrueExpr, FalseExpr*)

Cond = a logical expression representing the condition to be tested.
TrueExpr = a value to return if *Cond* is true.
FalseExpr = a value to return if *Cond* if false.

Use @IF evaluates the logical condition given as *Cond*. If the condition is found to be true, @IF returns the value given as *TrueExpr*. If the condition is false, @IF returns the value given as *FalseExpr*.

The expression entered as *Cond* can be any logical expression that can be evaluated as true or false; for example, *Order<0* or *Quantity*Price=53*.

You can use compound conditions by connecting expressions with @AND or @OR. If you use @AND, all conditions given must be met for the compound condition to evaluate to true. If you use @OR, the expression is true if any of the conditions is met. For example, @AND(*Quantity<50, Quantity>5*) means that the value in Quantity must be between 5 and 50 to evaluate true.

The expressions given as TrueExpr and FalseExpr can be any valid expression.

You can also use the @NOT operator to negate a condition. For example, @NOT(*Age>65*) evaluates true if the value of field Age is *not* greater than 65.

@IF functions can be nested, or used within one another. In other words, *TrueExpr* can contain yet another test to further validate *Cond*. There's no limit on the number of @IF expressions that you can nest, as long as the entire expression doesn't exceed 4096 characters.

Examples @IF(8=7,4,5) = 5
@IF(Order<100,"Yes","No") = Yes if Order < 100; otherwise, No.

@IF(Amount Requested>Amount Allowed, "Ms. Andrews", "") = Ms. Andrews if the value of the Amount Requested field is greater than value of the Amount Allowed field; otherwise, the result is ""(the empty string).

INSERT

Format @INSERT("*LinkName*")

"*LinkName*" = an open link name.

Use @INSERT writes connected ObjectVision field values to a new record for an ASCII, Paradox, dBASE, or Btrieve link. "*LinkName*" must be the name of an active link.

All values written to an ASCII file are double-quoted and comma-delimited. Each double quote in a value will be represented with two double quotes. An end-of-line character is written after the last value when inserting records using ASCII links.

If the link has a write filter, the filter must evaluate to true for the insert to occur. The insert fails if any database constraints are violated by the insertion. For example, if the record violates an index within a Paradox table a message appears stating that the insert will fail.

Examples @INSERT("BUYRECS") inserts the write-connected ObjectVision field values into data fields of the table linked with the BUYRECS link.

INT

Format @INT(*X*)

X = a numeric value.

Use @INT drops any fractional portion of *X*, returning only the integer value. See @ROUND (page 164) for a function that rounds *X* to the nearest integer.

Examples @INT(499.99) = 499
@INT(0.1245) = 0
@INT(Lowest Score) = 6, if the Lowest Score field value is 6.9
@INT(Credit Rating) = –76, if the Credit Rating field value is –76.9

IPAYMT

2.0

Format @IPAYMT(*Rate, Per, Nper, Pv, Fv, Type*)

Rate = a integer greater than 0 representing the fixed periodic interest rate.
Per = a positive integer that identifies the payment period (where *Nper* is the number of periods).
Nper = a positive integer, representing the number of periods of the loan.
Pv = a numeric value representing the amount borrowed (the principal).
Fv = a numeric value representing the future value of the investment.
Type = an argument indicating whether the cash flows occur at the beginning (1) or end (0) of the period.

Use The @IPAYMT and @PPAYMT functions tell what portion of a loan payment is interest and what portion is principal, respectively. Given *Rate, Nper, Pv, Fv,* and *Type,* the monthly payment (if we use a month as the example time period) is given by @PAYMT(*Rate, Nper, Pv, Fv, Type*)—see page 151 for more information on @PAYMT. For each month in the transaction period,

@PAYMT(*Rate, Nper, Pv, Fv, Type*)
= @IPAYMT(*Rate, Per, Nper, Pv, Fv, Type*)
+ @PPAYMT(*Rate, Per, Nper, Pv, Fv, Type*)

where *Per* identifies the *Per*th month, *Per* = 1, ..., *Nper*. @IPAYMT is calculated by computing the simple interest on the outstanding principal from the previous month. @PPAYMT then gives the principal portion of the payment for the current month, and is computed by subtracting @IPAYMT from @PAYMT. The calculation starts by using *Pv* as the outstanding principal at the beginning of the first month.

@IPAYMT(*Rate, Per, Nper, Pv, Fv, Type*)
= *Rate* * @FVAL(*Rate, Per* + (*Type* – 1),
@PAYMT(*Rate, Nper, Pv, Fv, Type*), *Pv, Type*)

Examples If you are two years into a 30-year, 10% mortgage on a $100,000 loan and your interest payment is tax-deductible, then

@IPAYMT(.1/12,2*12,30*12,100000,0,0)

returns your current month's deduction, or –824.03.

Remember, a negative result simply means the money is out of your pocket and not coming to you.

IRATE

2.0

Format @IRATE(*Nper*, *Pmt*, *Pv*, *Fv*, *Type*)

- *Nper* = a positive integer representing the length of the investment in terms of the number of compounding periods.
- *Pmt* = a numeric value representing the amount of the periodic payment.
- *Pv* = a numeric value representing the current value of an investment.
- *Fv* = a numeric value representing the future value of an investment.
- *Type* = a numeric value that indicates whether the cash flows occur at the beginning (1) or end (0) of the period.

Be sure to enter a negative number for money that's out of your pocket and a positive number for money that's coming in to you.

@IRATE is an updated version of the function @RATE (page 159).

@IRATE requires that the initial cash flow (*Pv* + *Type* * *Pmt*) and the last cash flow (*Fv* + (1-*Type*) * *Pmt*) have opposite signs. Otherwise, @IRATE returns ERR because the transaction is not simple and there may not be a meaningful rate.

Examples Assume you're negotiating to buy a $15,000 new car. The salesperson says you can have the car for $500 a month for the next five years. To calculate the monthly percentage rate:

@IRATE(5*12,-500,15000,0,0) = 0.02632

Another example: Assume that you plan to deposit $2000 a year into a savings account that currently contains only $2.38. What interest rate must the account earn to generate $15,000 at the end of five years? Here's how to calculate this:

@IRATE(5,-2000,-2.38,15000,0) = 0.2038

IRR

2.0

Format @IRR(*Guess*, *ColumnName*)

Guess = a numeric value that estimates the internal rate of return on an investment

ColumnName = a column that contains cash flow amounts for the investment

@IRR determines the internal rate of return on an investment. It references a *ColumnName* in your table that contains cash flow information and uses the supplied internal rate of return estimate to calculate the results.

Before you use this function, set up a cash flow table, showing expected cash flow amounts over a period of time. ObjectVision assumes that the amounts are received at regular intervals. Negative amounts are interpreted as cash outflows, and positive amounts as inflows. The first amount must be a negative number, to reflect the initial investment. The following amounts can all be the same for each time period, or they can be different (including a mixture of negatives, positives, or zeros).

The "Multiple rates of return" section discusses how to use @IRR to find the rate of return for a transaction with two or more roots.

Simple transactions

@IRR always returns ERR or a rate of return. Some cash flows have no rate of return, and some have several. If it can be determined that the cash flow has a unique rate of return, then the *Guess* is ignored and @IRR gives that unique value.

ObjectVision can determine unique rates of return for *simple transactions* or cash flow. A simple cash flow has two sets of values: a series of nonpositive values (negative values and zero) that is your cash outflow, and a series of nonnegative values (positive values and zero) that is your cash inflow. A simple cash flow must contain both a negative (cash outflow) value and a positive (cash inflow) value.

Multiple rates of return

In unusual cases, @IRR may have as many as *N-1* roots, where *N* is the number of cells in ColumnName. Consider a *ColumnName* that has the values (–10, +150, –145). The two roots are 3.86% and 1296% , obtainable

from guesses of 0 and 13, respectively. Both of these values are meaningful, if interpreted properly.

Maybe you invested $10 in an oil well, and it struck oil paying you $150, but then it ran into legal difficulties and required you to pay back $145. You had a net loss of $5, but your rate of return is quite large, as you got the use of a relatively large amount of money for a small investment. Or, maybe the real purpose of the transaction was to get a $150 loan from the bank. The bank required you to pay a $10 application fee ahead of time. After you get the loan, you pay back $145. Because you only paid back $155 on a $150 loan, the interest rate is fairly low. The difference in these two interpretations is that in one you are the lender, and in the other you're the borrower.

If you find a transaction with two roots, there is a mechanical way to determine which is the lender rate and which is the borrower rate. Pick a positive term in the *ColumnName*, and increase it by a small amount. If the rate increases, it is a lender rate, and if the rate decreases, it is a borrower rate.

If *ColumnName* has the values (-10,+150,-145),
@IRR(0,*ColumnName*) = 3.86%

If *ColumnName* has the values (-10,+150,-145),
@IRR(10,*ColumnName*) = 1296%

If *ColumnName* has the values (-10,+150.1,-145),
@IRR(0,*ColumnName*) = 3.78%

If *ColumnName* has the values (-10,+150.1,-145),
@IRR(10,*ColumnName*) = 1297%

Thus, because 3.78% < 3.86% and 1297% > 1296% , it follows from the above rule that 3.86% is a borrower rate and 1296% is a lender rate.

Most uses of @IRR are for analyzing an investment in which the first cash flow is negative, and the rate is a lender rate.

Some transactions have no rate of return at all. @IRR(*Guess*,*ColumnName*), with *ColumnName* having the values (-1,+1,-1), returns ERR regardless of the *Guess*. There is no rate of return that is meaningful for this cash flow.

If there are more than two roots, the above analysis can still be used to determine if a particular root is a lender rate or a borrower rate. In some cases, it might still be possible to assign meaning to a root, but it is much more likely that the transaction should really be interpreted as several transactions, with a rate of return for each. For example, the cash flow (-1,+6,-11,+6) has three roots, 0%, 100%, 200%. It is difficult to interpret

such a transaction in terms of interest rates, and the roots are sensitive to small fluctuations.

Examples @IRR(0, Col1) = 16.85%
@IRR(0, Col2) = –20.60%
@IRR(0, Col3) = 4.70%

ISBLANK

2.0

Format @ISBLANK(*FieldName*)

FieldName = the name of the field to check.

Use This function checks if a field has no value. If it has no value, the function returns a Yes; otherwise it returns No.

A field might contain a null string ("") and appear to be blank. If you type a carriage return in a field, ObjectVision will assign the field a null string. Also, if you delete a field value using either *Del* or *Backspace*, the field will be left with a null string. To get a field to be blank, use Form | Clear or @FIELDCLEAR (Field1).

Examples @ISBLANK(Company Slogan) = Yes, if the field is blank; otherwise No.

ISCOMPLETE

2.0

Format @ISCOMPLETE("*FormName*")

"*FormName*" = the name of the form you want to check.

Use This function checks if a form's status is *Complete*. If so, the function returns Yes; otherwise it returns No.

Once a form is complete, you can use another @function to print or close the form. You can also exit the application once all forms are complete. This gives you control over how your forms will be used.

Examples @ISCOMPLETE("Sales Order") = Yes, if the form is complete; otherwise No.

LEFT

Format @LEFT("*String*", *Num*)

"*String*" = a string value.
Num = a numeric value ≥ 0.

Use @LEFT returns the leftmost *Num* characters of *String*. It lets you extract a specified number of characters from the left side of a string.

If *Num* is longer than the length of *String*, all of *String* is returned.

Examples @LEFT("Jennifer",5) = Jenni
@LEFT("Jennifer",15) = Jennifer
@LEFT("155",1) = 1
@LEFT(" Jennifer",6) = J (including five leading spaces)
@LEFT(Job Grade,5) = Super, where the Job Grade field value is Supervisor

LENGTH

Format @LENGTH("*String*")

"*String*" = a string value.

Use @LENGTH returns the number of characters in *String*, including spaces. You can combine strings with the string concatenation operator, an ampersand (&). When *String* is a text string, it must be enclosed by double quotes.

Examples @LENGTH("Hello, world.") = 13

@LENGTH(" Jennifer") = 9 (including preceding space)

@LENGTH("Greetings "&"earthling") = 19 (including space after Greetings)

@LENGTH(FieldA&FieldB) = total number of characters in FieldA and FieldB

@LENGTH(Medical Insurance Provider) = 3, when the Medical Insurance Provider field value is HMO

LINKAVG

2.0

Format @LINKAVG("*LinkName*", "*DataField*")

"*LinkName*" = the name of the open link.
"*DataField*" = the name of the database field. The field should contain numeric values.

Use This function calculates the average of all the available records in the linked database table. The *DataField* does not need to be connected to an ObjectVision field. Any filters or restricted ranges effect the number of records averaged.

Examples @LINKAVG("LinktoBTRV", "Days") = 135 if the average of Days is 135.

LINKCOUNT

2.0

Format @LINKCOUNT("*LinkName*")

"*LinkName*" = the name of the open link.

Use This function counts the number of values in the range of the linked database table. Any filters or restricted ranges effect the number of records counted.

Examples @LINKCOUNT("LinktoPdox") = 1256 if there are 1,256 records available to the link.

LINKMAX

2.0

Format @LINKMAX("*LinkName*", "*DataField*")

"*LinkName*" = the name of the open link.
"*DataField*" = the name of the database field. The field should contain numeric values.

Use This function returns the maximum value in the records of the linked database table. The *DataField* does not need to be connected to an ObjectVision field. Any filters or restricted ranges effect the result of this function.

Examples @LINKMAX("CUSTOMERS", "Credit") = the maximum value in *Credit*.

LINKMIN

2.0

Format @LINKMIN("*LinkName*", "*DataField*")

"*LinkName*" = the name of the open link.
"*DataField*" = the name of the database field. The field should contain numeric values.

Use This function returns the minimum value in the records of the linked database table. The *DataField* does not need to be connected to an ObjectVision field. Any filters or restricted ranges effect the result of this function.

Examples @LINKMIN("Items", "Price") = the minimum price.

LINKSUM

2.0

Format @LINKSUM("*LinkName*", "*DataField*")

"*LinkName*" = the name of the open link.
"*DataField*" = the name of the database field. The field should contain numeric values.

Use This function returns the sum of the records in the linked database table. The *DataField* does not need to be connected to an ObjectVision field. Any filters or restricted ranges effect the result of this function.

Examples @LINKSUM("Items", "Price") = sums all values in the Price data field from the Items link.

LINKVALUE

2.0

Format @LINKVALUE("*LinkName*", "*DataField*")

"*LinkName*" = the name of the open link.
"*DataField*" = the name of the database field. The field should contain numeric values.

Use This function returns the value of the data field in the current record of the link. This function is useful only in VField and Filter expressions.

Examples (@LINKVALUE("Customers", "Zip")>90000) If the field Zip isn't read-connected, then @LINKVALUE must be used to retrieve the current value of Zip in this filter expression. If Zip were read-connected, you could replace @LINKVALUE ("Customers", "Zip") with the OVRead-connected field.

LN

2.0

Format @LN(*X*)

X = a numeric value > 0.

@LN returns the natural logarithm of *X*. A natural logarithm uses the mathematical constant *e* as a base. @LN produces the inverse of @EXP.

Examples @LN(3) = 1.098612289
@LN(@EXP(10)) = 10
@LN(16)/@LN(2) = 4
@LN(-4) = ERR (-4 is less than 0)

LOCATE

2.0

Format @LOCATE("*LinkName*", *InexactFlag*, *IndexValues*)

"*LinkName*" = the name of the link to the database table.
InexactFlag = specifies how closely you want to match the index value. Use 0 for an exact match, 1 for an inexact match.
IndexValues = the index values (ObjectVision fields) to use in the search.

Use This function locates and positions to a record in the linked database table. Once the position is moved on the database table, the link sends the appropriate values to the ObjectVision application through the read connections.

To use this function with your database tables, your tables must be indexed and you can only locate on an indexed field. For more information on locates, see Chapter 10, "Linking basics."

Examples @LOCATE("Link1", 0, "Murphy") = locates Murphy on the database table. Murphy is assumed to be a value under an indexed field called Name. Because *InexactFlag* is 0, only records exactly matching "Murphy" will be located.

LOG

Format @LOG(*X*)

X = a numeric value > 0

Use @LOG returns the base 10 logarithm of *X*.

Examples @LOG(1000) = 3
@LOG(10^23.8) = 23.8
@LOG(16)/@LOG(2) = 4 (log to base 2 of 16)

LOWER

Format @LOWER("*String*")

"*String*" = a string value.

Use @LOWER returns *String* in lowercase characters. Numbers and symbols within a string are unaffected. See also @UPPER and @PROPER.

Examples @LOWER("UPPER") = upper
@LOWER("Hello, world.") = hello, world.
@LOWER("145 Bancroft Lane") = 145 bancroft lane
@LOWER(4839) = 4839
@LOWER(@LEFT("Johnson",1)) = j
@LOWER((First Name)) = joanne, when the First Name field value is JoAnne

MAX

Format @MAX(*List*)

List = one or more numeric values up to a maximum of 14.

Use @MAX returns the largest numeric or date value in *List*. Text values are converted to numbers, if possible. If ObjectVision is unable to convert any of the arguments to a number, or if any of the referenced fields contain ERR, the resulting value is ERR. Logical values are converted to 1 (Yes) or 0 (No).

See also @MIN, @AVG, and @SUM.

Examples @MAX(Risk Category, Age Group) = 8, where the Risk Category field value is 8 and the Age Group field value is 5.

MESSAGE

Format @MESSAGE("*String*")

"*String*" = a string value.

Use @MESSAGE displays *String* in a message box with an OK button.

You typically use this function to inform the user that the value just entered is invalid, such as an out-of-range value.

Examples @MESSAGE("You must enter a value with at least 5 digits ")

MID

Format @MID("*String*", *StartNumber*, *Num*)

"*String*"	=	a string value.
StartNumber	=	a numeric value ≥ 0.
Num	=	a numeric value ≥ 0.

Use @MID extracts the first *Num* characters of *String* starting at character number *StartNumber*. It is similar to the @LEFT function, which extracts *Num* characters of *String* beginning with the first character. The difference

is that you can specify the character number to start with. Note that the first character of a string is character 0.

If *StartNumber* is greater than the length of *String*, or if *Num* is 0, the result is an empty string ("").

Examples @MID("Abraham Lincoln",8,7) = Lincoln
@MID("George Washington",7,4) = Wash
@MID("Theodore Roosevelt",19,5) = ""
@MID(ZIP Code,0,3) = 787, when the ZIP Code field value is 78751
@MID(President,@FIND("Roosevelt",President,0), @LENGTH("Roosevelt"))
= Roosevelt (if the President field value is Franklin Roosevelt)

MIN

Format @MIN(*List*)

List = one or more numeric values up to a maximum of 14.

Use @MIN returns the smallest numeric value in *List*. Text values are converted to numbers, if possible, and logical values are converted to 1 (Yes) or 0 (No).

@MIN returns ERR if ObjectVision is unable to convert any argument to a number. If a field has a blank value (""), it is assumed to be 0.

Examples @MIN(Risk Category,Age Group) = 5, when the Risk Category field value is 8 and the Age Group field value is 5.

MINUTE

Format @MINUTE(*DateTimeNumber*)

DateTimeNumber = a numeric value from –36522 to 73050, representing a date/time serial number.

Use @MINUTE returns the minute portion of *DateTimeNumber*. *DateTimeNumber* must be a valid date/time serial number. If *DateTimeNumber* is given as text, it is converted to a serial number. Because only the decimal portion of a serial number evaluates to a time, the integer portion of the number is disregarded. The result is a number from 0 to 59.

To extract the minute portion of a string that is in time format (instead of serial format), use @TIMEVALUE within the @MINUTE function to translate

the time into a serial number. You can also use @TIME to enter a time value instead of a serial number.

Examples @MINUTE(Time of Occurrence) = 30, when the Time of Occurrence field value is 11:30 a.m.

MOD

Format @MOD(*X*,*Y*)

X = a numeric value.
Y = a numeric value ≠ 0.

Use @MOD divides the *X* value by *Y* and returns the modulus, or remainder value. The result has the same sign as *Y*. Because you cannot divide a number by zero, ERR results if the value of *Y* is zero.

Examples @MOD(3,1) = 0 (3 divided by 1 leaves no remainder)
@MOD(5,2) = 1 (5 divided by 2 leaves a remainder of 1)
@MOD(3,1.1) = 0.8
@MOD(4,0) = ERR
@MOD(Field Kits,Field Kit Orders) = 50, when the Field Kits field value is 500 and the Field Kit Orders field value is 450

MONTH

Format @MONTH(*DateTimeNumber*)

DateTimeNumber = a numeric value from –36522 to 73050, representing a date serial number.

Use @MONTH returns the month portion of *DateTimeNumber*. *DateTimeNumber* must be a valid date/time serial number. Only the integer portion is used. The result is an integer value from 1 (January) to 12 (December).

To extract the month portion of a string that is in date format (instead of serial format), use @DATEVALUE within the @MONTH function to translate the date into a serial number (see page 113). You can also use @DATE to enter a date value instead of a serial number (see page 113).

Examples @MONTH(69858) = 4
@MONTH(58494) = 2
@MONTH(.37373) = 12
@MONTH(@DATEVALUE("3/5/88")) = 3

@MONTH(@DATE(1988,3,5)) = 3
@MOD(@MONTH(@DATEVALUE("3/5/88")),12) = 3
@MONTH(Date) = 2, when the Date field value is 1/Feb/1991

NA

Format @NA

Use @NA returns the special value NA (Not Available). Expressions that depend on a value entered as @NA return the value NA, unless there is an error, in which case they return ERR.

@NA is used to indicate values not included as a possible conclusion in a value tree. @NA ensures that inaccurate data is not displayed when ObjectVision evaluates an unexpected value.

Examples @NA = NA
@IF(Orders<0,@NA,Orders) = NA, when the Orders field is less than 0; otherwise, the value of Orders.

NEXT

Format @NEXT("*LinkName*")

"*LinkName*" = an open ASCII, Paradox, dBASE, or Btrieve link name.

Use @NEXT positions to the next record (or line for ASCII) of the database table. The link then reads values from the read connections. The link must be positioned on a current record for this function to work.

If the link is Auto Insert or Auto Update, an insert or update is attempted before the @NEXT is completed.

Examples @NEXT("ORDER") = positions to the next record on the linked database table.

NOT

Format @NOT(*Logical*)

Logical = a value of Yes or No.

Use @NOT inverts the value of *Logical* and returns Yes if *Logical* is No; otherwise, No if *Logical* is Yes.

Examples If you have a True/False field called Smoker (checked=the person smokes, unchecked=the person doesn't smoke), then you could create the following example on an insurance form.

@NOT(Smoker) = Yes, when the Smoker field value is No

The above example asks, "Is this person NOT a smoker?". If the value of Smoker is No (they don't smoke), then the answer to the question is Yes.

NOW

Format @NOW

Use @NOW returns the serial number of the current date and time in the user's computer clock. The value generated by @NOW is updated to the current date and time each time an expression recalculates a field value.

This serial number can range from –36522 to 73050, and represents the number of days from December 30, 1899 up to the current date. The earliest date available is January 1, 1800 and the latest date available is December 31, 2099.

The integer part of a date/time serial number evaluates to the date; the decimal portion evaluates to the time. To extract just the date portion, use @INT(@NOW). To extract just the time portion, use @MOD(@NOW,1).

Examples @NOW = 31905.572338 (5/8/87, 1:45 PM)
@INT(@NOW) = 31905 (5/8/87)
@MOD(@NOW,1) = 0.572338 (1:45 PM)
@INT(@MOD(@NOW,7)) = 6 (the number of the day of the week)

NPER

2.0

Format @NPER(*Rate, Pmt, Pv, Fv, Type*)

Rate = a numeric value representing the fixed interest rate per compounding period.
Pmt = a numeric value representing the amount of the periodic payment.

Pv = a numeric value representing the present value of the investment.

Fv = a numeric value representing the value that the investment will reach at some point.

Type = a numeric value (1 or 0) that indicates whether the cash flows occur at the beginning or the end of the period respectively.

Use

Be sure to enter a negative number for money that's out of your pocket and a positive number for money that's coming in to you.

@NPER is an updated version of the functions @CTERM and @TERM, which compute the number of payment periods required for an investment of *Fv* using two different sets of arguments (see pages 111 and 171, respectively).

This function returns a positive integer. If the result is less than zero, ERR is returned.

Examples

Assume you have an IRA account that earns 11.5% interest paid annually at the end of the year, and you deposit $2000 into the account at the end of each year. The present account balance is $633. To determine how many payment periods it will take to reach $50,000, use the @NPER function:

@NPER(11.5% ,-2000,-633,50000,0) = 13 payment periods

NPV

2.0

Format

@NPV(*Rate, ColumnName, Type*)

Rate = a numeric value representing a fixed periodic interest rate.

ColumnName = a column containing expected cash flow information.

Type = an argument indicating whether the cash flows occur at the beginning (1) or end (0) of the period.

Use

@NPV calculates the current value of a set of estimated cash flow values (*ColumnName*), discounted at the given interest rate (*Rate*). It helps you determine how much an investment is currently worth, based on expected earnings, although its accuracy depends on the accuracy of the cash flow table.

The formula for @NPV(*Rate,ColumnName,Type*)—if *ColumnName* consists of $v_1, \ldots, v_n$—is given by

If *Type* = 0:

$$\frac{v_1}{(1 + Rate)} + \ldots + \frac{v_n}{(1 + Rate)^n}$$

If *Type* = 1:

$$v_1 + \frac{v_2}{(1 + Rate)} + \ldots + \frac{v_n}{(1 + Rate)^{n-1}}$$

For example, suppose you are considering investing $5000, and you expect a return of $2000 in each of the next four years. Put the values –5000,+2000,+2000,+2000,+2000 in *ColumnName*. The net present value, using a discount rate of 10% , is @NPV(.1, *ColumnName*, 1) = 1340.

The cash flow table you reference should show expected income and debits over a period of time. When *Type* is 0, ObjectVision assumes that the amounts are received at the end of regular intervals; if *Type* is 1, it assumes that amounts are received at the beginning of regular intervals.

ObjectVision also assumes that the length of this interval is the same as the period on which interest is compounded. In other words, if monthly cash flow is estimated, *Rate* needs to show monthly interest. To convert annual interest to monthly interest, simply divide by 12.

Examples @NPV(1.25% , Col1, 0) = $62,683.77
@NPV(15% /12, Col3, 1) = $3,507
@NPV(15% /12, Col2, 0) = $31,216
–2000+@NPV(15% /12, Col1, 0) = $29,215.77 (assumes an initial cash outflow of $2,000)

OR

Format @OR(*LogicalList*)

LogicalList = one or more logical values or expressions that evaluate to a logical value, a value of Yes or No, up to a maximum of 14 values.

Use @OR returns Yes if any argument in *LogicalList* is Yes; otherwise, No.

Examples @OR(Self-Employed, Retired) = Yes, when either the Self-Employed field value or the Retired field value is Yes.

@OR(Self-Employed, Retired) = No, when *both* the Self-Employed field value and the Retired field value are No.

PAGEDN

2.0

Format @PAGEDN("*LinkName*")

"*LinkName*" = an open link to a database table.

Use This function displays records from the linked database table. A page is defined by the number of records you have displayed in a column minus one. If you display five records in a column, a page for the link to that column is four records. When you use @PAGEDN you will see the next four records in the database table with the last record from the previous page displayed in the first row. The link must be positioned on a current record for this function to work.

If the link is Auto Insert or Auto Update, an insert or update is attempted before the @PGDOWN is completed.

Examples @PAGEDN("CustNum")

In Figure 8.2, the table on the left shows what you'd see before using @PAGEDN, and the table on the right shows what you see after using @PAGEDN below shows the records from a database table.

Figure 8.2
Using a PAGEDN button

Before @PAGEDN	
No	Customer Name
1	Johnson
2	McMurphy
3	Smith

After @PAGEDN	
No.	Customer Name.
3	Smith
4	Larsen
5	Henderson

PAGEUP

2.0

Format @PAGEUP("*LinkName*")

"*LinkName*" = an open link to a database table.

Use This function works like @PAGEDN except it displays the previous set of records. The link must be positioned on a current record for this function to work. Both of these functions work great as buttons. Once the button is clicked, the previous set of records gets displayed in the column. The example below shows how this can be done.

If the link is Auto Insert or Auto Update, an insert or update is attempted before the @PAGEUP is completed.

Examples @PAGEUP("CustNum") = shows the previous records in the range.

PAYMT

2.0

Format @PAYMT(*Rate, Nper, Pv, Fv, Type*)

Rate = a numeric value representing interest rate.
Nper = an integer > 0, representing the number of periods of the loan.
Pv = a numeric value representing amount borrowed (the principal).
Fv = a numeric value representing the value that the investment will reach at some point.
Type = a numeric value that indicates whether the cash flows occur at the beginning (1) or the end (0) of the period.

Use

Be sure to enter a negative number for money that's out-of-pocket and a positive number for money that's coming in.

The functions @IPAYMT (page 133) and @PPAYMT (page 153) give the parts of the payment that are interest and principal, respectively.

This function has more versatility than @PMT in that it can calculate annuity due and can take into consideration the future value of the principal.

Examples Assume you want to take out a 30-year $175,000 mortgage with a 17.5% annual interest rate with 12 payments a year, and you'd like to see the difference in your monthly payments if you paid at the start or at the end of the month. All you have to do is enter these two functions:

@PAYMT(17.5% /12,12*30,175000,0,0) = –2566.07
@PAYMT(17.5% /12,12*30,175000,0,1) = –2529.19

If, on the other hand, your mortgage has a so-called balloon payment that leaves you with a hefty amount of unpaid principal at the end of the mortgage, you can still calculate the payment. Just insert the balloon payment amount (say, $80,000) as the *future value* component:

@PAYMT(17.5% /12,12*30,175000,-80000,0) = –2559.68

PI

2.0

Format @PI

@PI returns the value of π (3.141592653589793238), the ratio of a circle's circumference to its diameter.

Use To figure the area of a circle, given the radius in Field1, use the following expression:

```
@PI*Field1^2
```

Examples @PI*13 = 40.84 (circumference of circle with a diameter of 13)
@PI*(7.5)^2 = 176.7142 (area of circle with a radius of 7.5)
@PI*Field1 = the circumference of a circle whose diameter is the value in Field1

PMT

2.0

Format @PMT(*Pv*, *Rate*, *Nper*)

Pv = a numeric value representing the amount borrowed (the principal).
Rate = a numeric value representing the interest rate.
Nper = an integer > 0, representing the number of periods of the loan.

Use @PMT calculates the fully amortized payment of borrowing *Pv* dollars at *Rate* percent per period over *Nper* periods. It assumes that interest is paid at the end of each period.

@PAYMT is a more useful function you can use to perform this calculation. See @PAYMT on page 151 for details.

An equivalent for this formula using @PAYMT is

@PAYMT(*Rate*, *Nper*, – *Pv*, 0, 0)

You can enter the value for *Rate* as a decimal or a percent; for example, .095 or 9.5%. The amount you specify for *Rate* must correlate with the unit used for *Nper*.

In other words, if payments are made and interest calculated annually, the amount entered for *Nper* must represent years. If monthly, *Nper* must represent the number of months the loan covers.

To calculate monthly payments using an annual interest rate, divide the interest rate by 12.

@PMT assumes that the investment is an ordinary annuity. @PAYMT, @IPAYMT, and @PPAYMT, which are calculated differently than but are related to @PMT, let you use an argument, *Type*, to indicate whether the investment is an ordinary annuity or an annuity due.

Examples To calculate a monthly payment (paid at the last day of the month) for a three-year loan of $10,000 at an annual 15% interest rate, enter

@PMT(10000,15% /12,3*12) = $346.65

You can also use the more sophisticated @PAYMT function to figure this payment:

@PAYMT(15% /12,3*12,-10000,0,0) = $-346.65

The negative result means the payment is out of your pocket.

Other examples:

@PMT(1000,0.12,5) = $277.41
@PMT(500,0.16,12) = $96.21
@PMT(5000,16% /12,12) = $453.65
@PMT(12000,0.11,15) = $1,668.78
@PMT(10000,15% /12,36) calculates a monthly payment for a three-year loan of $10,000 at an annual 15% interest rate.

PPAYMT

2.0

Format @PPAYMT(*Rate, Per, Nper, Pv, Fv, Type*)

Rate = a numeric value representing interest rate.
Per = a positive integer representing the number of periods into the loan for which the principal is desired.
Nper = an integer > 0, representing the number of periods of the loan.
Pv = a numeric value representing amount borrowed (the principal).
Fv = a numeric value representing the value the investment will reach at some point.
Type = a numeric value that indicates whether the cash flows occur at the beginning (1) or end (0) of the period.

Use The functions @IPAYMT (page 133) and @PPAYMT give the parts of the payment that are interest and principal, respectively.

See the entry for @IPAYMT for an explanation of how @IPAYMT and @PPAYMT interrelate and how they both relate to @PAYMT.

Examples Assume you're two years into a 30-year, 10% mortgage on a $100,000 loan. To determine what portion of this month's payment is principal, enter

@PPAYMT(.1/12,2*12,30*12,100000, 0, 0) = $-53.54 (the result is negative to indicate the money is out of your pocket)

Another example:

@PPAYMT (.15/4,24,40,10000,0,1) = $-250.84 quarterly payments for $10,000 loan at 15% annual percentage rate adjusted to a quarterly basis over a 10-year term

PREVIOUS

Format @PREVIOUS("*LinkName*")

"*LinkName*" = the name of the open Paradox, dBASE, or Btrieve database link.

Use @PREVIOUS positions to the previous record in the database table. After the reposition, the link reads values from the read connections of the database table. The link must be positioned on a record for this function to work.

If the link is Auto Insert or Auto Update, an insert or update is attempted before the @PREVIOUS is completed.

Examples @PREVIOUS("ORDER")

PRINTALL

Format @PRINTALL

Use This function prints all forms in the application with the current information (with links, it prints the current record). See also @PRINTLINK.

You can program your application to print all the forms once they are all complete. See the example below.

Examples @PRINTALL = prints all forms with the current information.

You can also use this function with a button. When you are working with your application and you want to print what is currently on the form, you would only need to click the button and all forms would be printed.

PRINTFORM

2.0

Format @PRINTFORM("*FormName*")

"*FormName*" = the name of the form you want to print.

Use This function prints one form in the application. You can program your application to print a form once it is selected. You can also list all forms in the application and print only the form that is selected from the list. See the example below.

Examples @PRINTFORM(Select a Form to Print)
@PRINTFORM(@SELECTEDFORM) = prints the selected form.

This function is used in an event tree for the button Print Form Now. When the button is clicked, the form checked in the field Select a Form to Print is printed.

PRINTLINK

2.0

Format @PRINTLINK("*LinkName*")

"*LinkName*" = the name of the open link that contains the values you want to print.

Use This function prints the selected form *once* for each record in the linked database table. If there are 56 records in the table, 56 forms will be printed. This is a fast way of printing only the records you need. You can combine this function with a filter or a restricted range.

Examples @PRINTLINK("CustNum") = will print all the available records linked to the Customers database table. You could combine this with a filter on Customer Number so you'd only print forms belonging to a specific customer number.

PROPER

2.0

Format @PROPER("*String*")

"*String*" = a string value.

@PROPER converts the first letter of every word in *String* to uppercase, and the rest of the characters to lowercase. A word is defined as an unbroken string of alphabetic characters. Any blank spaces, punctuation symbols, or numbers mark the end of a word.

Examples
@PROPER("GEORGE washINGTON") = George Washington
@PROPER("FIRST QUARTER") = First Quarter
@PROPER("JOHN J. SMITH") = John J. Smith
@PROPER("1979's results") = 1979'S Results
@PROPER (Name) = John J. Smith (where field Name contains JOHN J. SMITH)

PV

2.0

Format @PV(*Pmt, Rate, Nper*)

Pmt = a numeric value representing the amount of the periodic payment.
Rate = a numeric value representing periodic interest rate.
Nper = an integer > 0 representing the number of payments to be made.

@PV calculates the present value of an investment where *Pmt* is received for *Nper* periods and is discounted at the rate of *Rate* per period.

An equivalent formula using @PVAL is

@PVAL(*Rate, Nper, –Pmt*, 0, 0)

@PV assumes that the investment is an ordinary annuity. @PVAL, which is calculated differently than but is related to @PV, lets you use an optional argument, *Type*, to indicate whether the investment is an ordinary annuity (0) or an annuity due (1).

Examples For example, suppose your day-care center wants a new van that costs $12,000. The dealership presents two offers: Pay the whole amount up front, or pay $350 per month for the next five years at 7% interest. The present value of the loan is

@PV(350,7% /12,5*12) = $17,675.70

The value of this loan is over $5000 more than the value of the full initial payment.

You can also use @PVAL (described next). The car loan example could be rewritten as

@PVAL(7% /12,5*12,-350,0,0) = $17,675.70

Other examples:

@PV(277,0.12,5) = $998.52
@PV(600,0.17,10) = $2,795.16
@PV(100,0.11,12) = $649.24

PVAL

2.0

Format @PVAL(*Rate, Nper, Pmt, Fv, Type*)

Rate = a numeric value representing periodic interest rate.
Nper = number of periods, which should be an integer > 0.
Pmt = a numeric value representing the amount of the periodic payment.
Fv = future value.
Type = 0 if payments are at the end of each period, 1 if they are at the beginning.

Use @PVAL is an updated version of the function @PV (see the preceding section).

Be sure to enter a negative number for money that's out of your pocket and a positive number for money that's coming in to you.

Examples Your grandfather remembers you in his will and leaves you $24,000 in cash over the next 12 years ($2000 a year) *or* you can have all his government bonds, which mature in 15 years to a worth of $30,000. (Whichever you don't choose goes to charity.) To determine which is worth more, compute the present value of the $24,000. Assume you can invest the money as you accumulate it in a 10% money market account.

@PVAL(10% ,12,2000,0,0) = –13,627.38

Now compare this figure with the present value of the $30,000, which you won't receive for 15 years:

@PVAL(10% ,15,0,30000,0) = –7,181.76

These results tell you that the $24,000 spread over 12 years is a sounder investment (it's worth spending over $13,000 on it, in fact) than the other choice, which is presently worth only $7000.

PXOPEN

Format @PXOPEN("*LinkName*", "*TableName*", "*DataFields*", "*OVReadFields*", "*OVWriteFields*", "*SecIndexField*", *Option*)

"*LinkName*"	=	name you specify for the Paradox database file link.
"*TableName*"	=	name of the Paradox database table (.DB).
"*DataFields*"	=	names of the database fields, delimited by commas, to be connected to your ObjectVision fields.
"*OVReadFields*"	=	names of your ObjectVision fields, delimited by commas, to hold imported values from *TableName DataFields*. There must be a one-to-one correspondence between the data fields and your ObjectVision fields.
"*OVWriteFields*"	=	names of your ObjectVision fields, delimited by commas, to write their values to *TableName DataFields*. There must be a one-to-one correspondence between the data fields and your ObjectVision fields.
"*SecIndexField*"	=	name of the *TableName* field to be used as the index for database access. This field, if specified, is used instead of the database's primary index to order and locate records. If blank (""), the primary index is used. This index must be created by Paradox.
Option	=	specifies how closely you want to match the index value. Use 0 for an exact match, 1 for an inexact match.

Use @PXOPEN opens a named, bidirectional database link between *FileName* and your ObjectVision fields.

@PXOPEN returns Yes if successful; otherwise, No.

This function is available for compatibility with ObjectVision version 1.0. You should use the Links Tool for creating Paradox links.

Examples @PXOPEN("INVEN","INVENTOR", "Item,Quantity", "Warehouse,City", "Warehouse,City","", 0)

This function opens the INVEN write link to the Paradox INVENTOR.DB data file (in the current directory). The database fields are Item and Quantity, and the ObjectVision fields are Warehouse and City.

RADIANS

2.0

Format @RADIANS(*X*)

X = a numeric value representing degrees

@RADIANS converts the given number of degrees to radians, using the following formula:

$$\frac{\pi}{180} X$$

One degree is equal to approximately 0.017 radians.

Examples
@RADIANS (1) = 0.017453
@RADIANS (57) = 0.994838
@RADIANS (@DEGREES (3.5)) = 3.5

RATE

2.0

Format @RATE(*Fv*, *Pv*, *Nper*)

Fv = a numeric value representing the future value of an investment.
Pv = a numeric value representing the current value of an investment.
Nper = a positive integer representing the length of the investment in terms of the number of compounding periods.

Use @RATE calculates the interest rate required in order for an investment of *Pv* to be worth *Fv* within *Nper* compounding periods. If *Nper* represents years, an annual interest rate results; if *Nper* represents months, a monthly interest rate results, and so on.

An equivalent for this formula using @IRATE (see page 134) is

@IRATE(*Nper*,0, –*Pv*, *Fv*, 0)

@RATE assumes that the investment is an ordinary annuity. @IRATE, which is calculated differently than but is related to @RATE, lets you use an

argument, *Type*, to indicate whether the investment is an ordinary annuity or an annuity due.

Examples To determine what yearly interest rate would double an initial investment of $2000 at the end of 10 years, use

@RATE(4000,2000,10) = 7.18%

Other examples:

@RATE(10000,7000,6*12) = 0.50% (monthly)
@RATE(1200,1000,3) = 6.27% (yearly)
@RATE(500,100,25) = 6.65% (yearly)

REGISTER

2.0

Format @REGISTER("*OVAlias*", "*ArgTypes*", "*ArgHelp*", "*LibName*", "*FuncName*", *Type*)

"*OVAlias*" = the new function name to be used in ObjectVision.
"*ArgTypes*" = Code Type.

		Passed	Returned
A	Boolean	integer	integer
B	IEEE float	double	double far*
C	C string	char far*	char far*
F	C string buffer	char far*	char far*
H	Unsigned integer	unsigned int	unsigned int
I	Signed integer	signed int	signed int
J	Unsigned long	unsigned long	unsigned long
o	GetCurrentRow	FARPROC	
p	NextCol	FARPROC	
q	GetLastRow	FARPROC	
r	IsColumn	FARPROC	
s	SetCurrentRow	FARPROC	
t	SetLinkRow	FARPROC	
u	ResetCalculated	FARPROC	
v	ResetOverride	FARPROC	
w	Put	FARPROC	
x	Get	FARPROC	
y	hMainWindow	HANDLE	
z	hInst	HANDLE	

"*ArgHelp*" = the text that appears when you use Paste Arguments in the ObjectVision function dialog box.

"LibName" = the name of the DLL that contains the function you want to register. If the DLL is not in the current path, you will need to type its complete path name.

"FuncName" = the name of the function as it appears in the DLL.
Type = either 0 or 1 for value or event functions respectively.

Use This function registers custom @functions for use in ObjectVision applications. This function is described in great detail later.

Examples @REGISTER("@CHOICE","FACCF","Condition, TrueString, FalseString","CHOICE.DLL","ChooseString", 0)

This example registers the @CHOICE function from the DLL called CHOICE.DLL. For more information on @REGISTER see Chapter 18, "Using @REGISTER with DLLs."

REPEAT

Format @REPEAT("*String*", *Num*)

"String" = a string value.
Num = a numeric value ≥ 0.

Use @REPEAT returns *Num* copies of *String* as one continuous string. You can specify exactly how many times you want the string to be repeated. The @REPEAT function displays a fixed number of copies of *String*.

If *Num* equals 0, @REPEAT returns "". @REPEAT returns ERR (error) if *Num* is less than 0.

When you specify a text string with @REPEAT, it must be enclosed by double quotes.

Examples
@REPEAT("-",20) = ----------
@REPEAT("good day!",3) = good day!good day!good day!
@REPEAT(City,5) = the City field value repeated 5 times
@REPEAT("-",@LENGTH(Jane Hopper)) = ----------

REPLACE

Format @REPLACE("*String*", *StartNum*, *Num*, "*NewString*")

"*String*"	=	a string value, representing the text to operate on.
StartNum	=	a numeric value ≥ 0, representing the character position to begin with.
Num	=	a numeric value ≥ 0, representing the number of characters to delete.
"*NewString*"	=	a string value, representing the characters to insert at position *Num*.

Use @REPLACE lets you replace characters in a text string with a new string. It searches through the given *String* until it reaches the given character position *StartNum*. Then it removes *Num* number of characters from the string, replacing them with *NewString*.

Note that for @REPLACE, as for other string functions, the first character of *String* is counted as character 0.

To replace one string with another, specify 0 as *StartNum*. For *Num*, enter a number equal to or greater than the number of characters in *String*.

To insert one string into another string, specify 0 as *Num*.

To add one string to the end of another, specify as *StartNum* a number equal to the number of characters in *String*.

To delete part or all of a string, specify "" as *NewString*.

Examples @REPLACE("McDonald Corp.",2,6,"Douglas") = McDouglas Corp.

@REPLACE("Leslie J. Cooper",7,3,"") = Leslie Cooper

@REPLACE("Sales Salaries",6,0, "Reps' ") = Sales Reps' Salaries (There must be a space between Reps and the final quotation mark.)

@REPLACE("355 Howard",9,0," St.") = 355 Howard St. (There must be a space between " and *St.*)

@REPLACE(Distribution Area,30,100," USA only") = Limited USA only, when the Distribution Area field value is Limited.

RESTOREMENU

2.0

Format @RESTOREMENU

Use This function restores the menus to the default ObjectVision menus. You would use this function if you changed the menus, then wanted to restore the entire menu selection.

If you only want to change one or two menu commands or items, use @DELETEMENU or @DELETEMENUITEM.

Examples @RESTOREMENU = restores all menus to the original ObjectVision menus.

RESUME

2.0

Format @RESUME

Use This function resumes guided completion. This function is best used after you've used another function that breaks guided completion, such as @FIELDFIND.

Examples @RESUME = resumes guided completion.

RIGHT

Format @RIGHT("*String*", *Num*)

"*String*" = a string value.
Num = a numeric value ≥ 0.

Use @RIGHT returns the last *Num* characters of *String*. It lets you extract a specified number of characters from the right side of a string or label.

If *Num* is 0, the result is "", or an empty string. If *Num* is equal to or greater than the number of characters in *String*, the entire string is returned.

Examples @RIGHT("Jennifer Meyer",5) = Meyer
@RIGHT("Jennifer Meyer",25) = Jennifer Meyer
@RIGHT("Jennifer ",6) = fer (including 3 trailing spaces)
@RIGHT("155",1) = 5
@RIGHT(123,1) = 3

@RIGHT(High School Complete,2) = 10, when the High School Complete field value is Grade 10

ROUND

Format @ROUND(*X*,*Num*)

X = a numeric value.
Num = a numeric value from –15 to 15.

Use @ROUND adjusts the precision of *X* to *Num* decimal points. *Num* specifies the power of 10 to which *X* is rounded. If *Num* is positive, *X* is rounded *Num* digits to the *right* of the decimal point. If *Num* is negative, *X* is rounded *Num*+1 digits to the *left* of the decimal point. For example, if *Num* is –3, *X* is rounded to the nearest thousand.

If *Num* is 0, *X* is rounded to an integer. If *Num* is not an integer, it is rounded off to the nearest integer.

Examples @ROUND(12345.54321,0) = 12346
@ROUND(12345.54321,2) = 12345.54
@ROUND(12345.54321,-2) = 12300
@ROUND(Average Weight,1) = 2143.9, when the Average Weight field value is 2143.877

SAVE

2.0

Format @SAVE

Use This function works like File | Save from the main menu. The application is saved under its original name. The previous version of the application is overwritten.

If you have not given your application a name and you use this function, the Save dialog box will appear. You can then give the file a name, click OK, and the application is saved.

SAVEAS

2.0

Format @SAVEAS("*FileName*")

"*FileName*" = the name under which to save the file.

Use This function works like File | Save As from the main menu. FileName can get its value from a field, so if you create a field called Application Name and a button called Save, you could type an application name in the field, then click the button and the application would be saved under a new name.

This is a handy way to save applications if you aren't linked to database tables. You can store one set of values in each field and then save the entire application with that information. Beware, though, that this type of saving uses more disk space than creating a form and saving sets of records in a database table, because the form itself is duplicated with each save.

Examples @SAVEAS("MyOrder") = saves the current application as MYORDER.OVD

You could create a field called App Name where you type the name you want to call the saved file. If you want to save multiple versions of an application, you could create a conclusion on a button such as the one below. For this example, you'd also create a Scratchpad field called App Number whose value determines the number of the version.

@SAVEAS(App Name & App Number)
@ASSIGN(App Number, (App Number+1))

If App Name has the value Try and App Number begins with a value of 1, this conclusion will produce files TRY1.OVD, TRY2.OVD, TRY3.OVD, and so on.

SECOND

Format @SECOND(*DateTimeNumber*)

DateTimeNumber = a numeric value from –36522 to 73050.99999 representing a date/time serial number.

Use @SECOND returns the second portion of *DateTimeNumber*. *DateTimeNumber* must be a valid date/serial number. Because only the decimal portion of a

serial number evaluates to a time, the integer portion of the number is disregarded. The result is a number from 0 to 59.

To extract the second portion of a string that is in time format (instead of serial format), use @TIMEVALUE within the @SECOND function to translate the time into a serial number (see page 172). You can also use @TIME to enter a time value instead of a serial number (see page 172).

Examples @SECOND(.3655445) = 23
@SECOND(.2543222) = 13
@SECOND(35) = 0
@SECOND(@NOW) = 49, if the current date and time on the computer clock running ObjectVision is March 17, 1990, 2:15:49 p.m.
@SECOND(@TIME(3,15,22)) = 22
@SECOND(@TIMEVALUE("10:08:45 am")) = 45
@SECOND(@TIMEVALUE("10:08 am")) = 0

SELECTEDFIELD

2.0

Format @SELECTEDFIELD

Use This function returns the name of the currently selected field.

Examples @SELECTEDFIELD = Sales Representative if this field is selected.

SELECTEDFORM

2.0

Format @SELECTEDFORM

Use This function returns the name of the currently selected form.

Examples @SELECTEDFORM = Current Accounts if this is the selected form.

@FORMCLOSE(@SELECTEDFORM) = closes the current form.

SETTITLE

2.0

Format @SETTITLE("*Title*")

"*Title*" = the new title. This replaces the default ObjectVision title.

Use This function changes the title of the program. You can use this function to add the final touch to customizing your applications. You can change the menus and the title to reflect your company's style.

Examples @SETTITLE("ACME WIDGETS") = removes ObjectVision and replaces it with ACME WIDGETS

@SETTITLE("") = the program window now has no title.

@SETTITLE(@CURRENTPATH&@CURRENTFILE) = changes the name of the title to the full path and application name.

SIN

2.0

Format @SIN(*X*)

X = a numeric value.

@SIN returns the sine of the angle *X*. *X* must be given in radians, not degrees. To convert degrees to radians, use the @RADIANS function (page 159).

Examples @SIN(@RADIANS(30)) = 0.5
@SIN(@PI/6) = 0.5

SLN

2.0

Format @SLN(*Cost,Salvage,Life*)

Cost	=	a numeric value representing the amount paid for an asset.
Salvage	=	a numeric value representing the value of an asset at the end of its useful life.
Life	=	a numeric value representing the number of years of useful life for the asset.

@SLN calculates the straight-line depreciation allowance for an asset over one period of its life, using the following formula:

$$\frac{Cost - Salvage}{Life}$$

To compute accelerated depreciation (allowing higher depreciation values in the first years of the asset's life), use the @SYD function (page 169). To calculate depreciation using the double-declining balance method, use the @DDB function (page 116).

Examples Assume you just bought a new $4000 computer. Your dealer says you can sell it back to the store for $350 after eight years, but no one would want to buy it after that. In other words, the *Salvage* value of that computer is $350 and its *Life* is 8. To determine the depreciation allowance of the computer for each year of its life, enter this formula:

@SLN(4000,350,8) = 456.25

Other examples:

@SLN(15000,3000,10) = $1,200
@SLN(5000,500,5) = $900
@SLN(1800,0,3) = $600

SQRT

2.0

Format @SQRT(*X*)

X = a numeric value ≥ 0.

@SQRT returns the square root of *X*. If *X* is a negative value, the result of @SQRT is ERR.

Examples
@SQRT(9) = 3
@SQRT(2) = 1.414213562
@SQRT(144) = 12
@SQRT(@SQRT(16)) = 2
@SQRT(-4) = ERR

STORE

Format @STORE("*LinkName*")

"*LinkName*" = an open Paradox, dBASE, or Btrieve database link.

Use @STORE uses the current ObjectVision write-connected values either to insert a new record into the database table (@Insert), or, if a record is being viewed, to update the current ObjectVision record (@Update). See @INSERT and @UPDATE.

Examples @STORE("INVEN") = stores values to the database table linked with "INVEN".

SUM

Format @SUM(*List*)

List = one or more numeric values, up to a maximum of 14.

Use @SUM returns the total of all numeric values in *List*. *List* can be any combination of field value references, logical values, and numeric values. Text values are converted to numbers, if possible. Logical values evaluate as 1 (Yes) or 0 (No).

When more than one argument is used, arguments must be separated by commas. Any value @SUM is unable to convert to a number causes ERR (error) to be returned.

Examples @SUM(78,125) = 203
@SUM(203,417,94) = 714
@SUM(Food Expense,Lodging Expense,Travel Expense,Miscellaneous) = 714, when the Food Expense field value is 78, Lodging Expense is 125, Travel Expense is 417, and Miscellaneous is 94.

SYD

2.0

Format @SYD(*Cost*, *Salvage*, *Life*, *Period*)

Cost	=	a numeric value representing the initial cost of an asset.
Salvage	=	a numeric value representing the value of the asset at the end of its life expectancy.
Life	=	a numeric value representing the length of the asset's life expectancy.
Period	=	a numeric value representing the period for which you want to calculate depreciation.

The following must be true:

$Cost \geq Salvage \geq 0$
$Life \geq Period \geq 1$

@SYD calculates depreciation amounts for an asset using an accelerated depreciation method. This allows higher depreciation in the earlier years

of the asset's life. @SYD uses the following formula to compute depreciation:

$$\frac{(Cost - Salvage)\ (Life - Period + 1)}{Life\ (Life + 1)/2}$$

Examples Assume you just bought a new $4000 computer. Your dealer says that you can sell it back to the store for $350 after eight years, but that no one would want to buy it after that. In other words, the *Salvage* value of that computer is $350 and its *Life* is 8. To see what the depreciation allowance of this computer will be by the second year (using this method of depreciation), enter this formula:

@SYD(4000,350,8,2) = 709.72

The following examples show depreciation values for the first five years of an asset's life. These can be compared to those calculated with the @DDB function, which distributes more of the depreciation in the first year of life.

@SYD(12000,1000,5,1) = $3,667
@SYD(12000,1000,5,2) = $2,933
@SYD(12000,1000,5,3) = $2,200
@SYD(12000,1000,5,4) = $1,467
@SYD(12000,1000,5,5) = $733

@DDB(12000,1000,5,1) = $4,800
@DDB(12000,1000,5,2) = $2,880
@DDB(12000,1000,5,3) = $1,728
@DDB(12000,1000,5,4) = $1,036.80
@DDB(12000,1000,5,5) = $555.20

TAN

2.0

Format @TAN(*X*)

X = a numeric value.

@TAN returns the tangent of the angle *X*. *X* must be given in radians, not degrees. To convert degrees to radians, use the @RADIANS function (page 159).

Examples @TAN(4) = 1.157821
@TAN(@PI/4) = 1
@TAN(@RADIANS(45)) = 1

TERM

2.0

Format @TERM(*Pmt*, *Rate*, *Fv*)

Pmt = a numeric value representing the amount of the periodic payment.
Rate = a numeric value representing a fixed, periodic interest rate accrued by the investment.
Fv = a numeric value representing the future value of the investment.

@TERM computes the number of payment periods required in order to accumulate an investment of *Fv*, making regular payments of *Pmt* and accruing interest at the rate of *Rate*.

An equivalent for this formula using @NPER (see page 147) is

@NPER(*Rate*, – *Pmt*, 0, *Fv*, 0)

@TERM assumes that the investment is an ordinary annuity. @NPER, which is calculated differently than but is related to @TERM, lets you use an optional argument, *Type*, to indicate whether the investment is an ordinary annuity or an annuity due.

Examples To determine how long it will take to accrue $50,000 by depositing $2000 at the end of each year into a savings account that earns 11% annually, enter this formula:

@TERM(2000,11% ,50000) = 13

You can also use the more sophisticated @NPER function to calculate this example:

@NPER(11% ,-2000,0,50000,0) = 13

Other examples:

@TERM(300,6% ,5000) = 12 years
@TERM(500,7% ,1000) = 2 years
@TERM(500,.07,1000) = 2 years
@TERM(1000,10% ,50000) = 19 years
@TERM(100,5% ,1000) = 9 years

TIME

Format @TIME(*Hr,Min,Sec*)

Hr = a number from 0 to 23, representing Hour.
Min = a number from 0 to 59, representing Minute.
Sec = a number from 0 to 59, representing Second.

Use @TIME returns the date/time serial number represented by *Hr:Min:Sec*. Each of these arguments must be within the valid ranges specified above. Any fractional portions are truncated.

Examples @TIME(3,0,0) = 0.125 (3:00 am)
@TIME(3,30,15) = 0.14600694444 (3:30:15 am)
@TIME(18,15,59) = 0.76109953704 (6:15:59 pm)

TIMEVALUE

Format @TIMEVALUE("*TimeString*")

"*TimeString*" = a string value in any valid time format, enclosed by quotes.

Use @TIMEVALUE returns a serial time value in the range from 0 to 0.9999, calculated as the fractional portion of a 24-hour day that corresponds to the value in *TimeString*. TimeString must be a text value containing a valid time from 0:00:00 to 23:59:59.

If the value in *TimeString* is not in the correct format, or is not enclosed in quotes, an ERR value is returned.

You can enter the time string in any of the ObjectVision time display formats; the "AM" or "PM" designation is optional. There are two valid formats for *TimeString*:

- HH:MM:SS AM/PM (03:45:30 PM)
- HH:MM AM/PM (03:45 PM)

The actual time separator character used (recognized) is the one you specify in the Windows Control Panel.

Examples @TIMEVALUE("03:30:15 AM") = 0.1460069444
@TIMEVALUE("03:00") = 0.125
@TIMEVALUE("03:45 PM") = 0.65625
@TIMEVALUE("18:15:59") = 0.76109953704
@TIMEVALUE("3.45") = ERR

@TIMEVALUE("14:00 PM") = ERR

TODAY

2.0

Format @TODAY

Use @TODAY returns the numeric value of the system's date. It is equal to the expression @INT(@NOW).

TOP

Format @TOP("*LinkName*")

"*LinkName*" = an open ASCII, Paradox, dBASE, or Btrieve link name.

Use @TOP repositions the "*LinkName*" database link to the first record in the ASCII, Paradox, dBASE, or Btrieve file based on link indexing order. "*LinkName*" must be the name of a link previously opened with Tools | Link. The link must be positioned on a current record for this function to work.

If the link is Auto Insert or Auto Update, an insert or update is attempted before the @TOP is completed.

@TOP returns Yes if successful; otherwise, No.

Examples @TOP("ORDER"), repositions the ORDER link to the first record in the ORDER database.

TRIM

2.0

Format @TRIM("*String*")

"*String*" = a string value.

@TRIM removes any extra spaces from *String*: that is, spaces following the last non-space character or preceding the first non-space character, and duplicate spaces between words. Strings with no extra spaces are not affected.

Examples @TRIM(" too many spaces ") = "too many spaces"
@TRIM("no extra spaces") = "no extra spaces"

TYPE

Format @TYPE(*Value*)

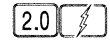
Value = a constant value or an expression.

Use @TYPE returns one of four codes indicating the data type of *Value*:

- **1** Numeric value
- **2** Text value
- **4** Logical value
- **16** Error value

Examples @TYPE(State Tax) = 1, when the State Tax field value is a numeric value
@TYPE('1990 File') = 4, when the 1990 File field value is Yes
@TYPE(2.14159) = 1
@TYPE("Thomas Jefferson") = 2
@TYPE(No) = 4
@TYPE(@ERR) = 16

UNCHECKMENUITEM



Format @UNCHECKMENUITEM("*MenuName*", "*MenuItemName*")

"*MenuName*" = the name of the menu command on the main menu.
"*MenuItemName*" = the name of the item under the menu command.

Use This function removes a check mark from *MenuItemName* in the list of items under the command in the main menu.

Examples @UNCHECKMENUITEM("File", "New") = removes the check mark from New under the File command.

UPDATE

Format @UPDATE("*LinkName*")

"*LinkName*" = an open Paradox, dBASE, or Btrieve link name.

Use @UPDATE writes current ObjectVision values to the current record position of the database link through the write connections. If the link specifies an indexed field, the index is also updated.

@UPDATE will fail if there is no current link position in the database or if there is a key failure. If there is a write filter on the link, the filter expression must be true for the values to be written to the database table.

@UPDATE returns Yes if successful; otherwise, No.

Examples @UPDATE("ORDER"), uses the ORDER link to revise an existing record in the open database.

UPPER

Format @UPPER("*String*")

"*String*" = a string value.

Use @UPPER returns *String* in uppercase characters. Numbers and symbols within a string are unchanged.

Examples
@UPPER(4839) = 4839
@UPPER(@LEFT("johnson",1)) = J
@UPPER("upper") = UPPER
@UPPER("Hello, world.") = HELLO, WORLD.
@UPPER("145 Bancroft Lane") = 145 BANCROFT LANE

VERSION

2.0

Format @VERSION

Use This function returns the version number of ObjectVision that you are using. You can use this function to include or exclude parts of your application based on the version number of your user's program.

Examples @VERSION = 2.0 if you are using this version of ObjectVision.

WEEKDAY

Format @WEEKDAY(*DateTimeNumber*)

DateTimeNumber = a numeric value from –36522 to 73050 representing a date/time serial number.

Use @WEEKDAY returns the day from the week specified by the *DateTimeNumber*. The day is returned as an integer value from 1 (Sunday) to 7 (Saturday).

DateTimeNumber represents the number of days from December 30, 1899 to the date referenced in the expression. The earliest date available is January 1, 1800 and the latest date available is December 31, 2099.

The fractional portion of a date serial number is used by the time functions.

Any illegal dates return ERR as their value. If *DateTimeNumber* is given as text, it is converted to a serial number.

Examples @WEEKDAY(Start Date) = 7, when the Start Date field value is January 27, 1990 (a Saturday)

YEAR

Format @YEAR(*DateTimeNumber*)

DateTimeNumber = a numeric value from –36522 to 73050 representing a date/time serial number.

Use @YEAR returns the year portion of *DateTimeNumber*. The result will be a number from 1800 to 2099. If *DateTimeNumber* is given as a string, it is converted into a serial number.

Examples @YEAR(22222) = 1960
@YEAR(End Date) = 1991, where the End Date field value is 1/27/91

C H A P T E R

9

Menu customization

This chapter teaches you how to customize menus for your applications. You should have an understanding of @functions and event tree before reading this chapter.

Overview

Creating your own menus or simply adding a new menu item to the existing ones gives you the power to shape your application and control what users can do with your application.

There are several @functions that change the main menu. You use these @functions with event trees. The event tree can be on the stack, so when an event occurs to the stack, such as Open, the menus are changed, or the event trees could be on a form or object.

You can use menu functions in any event tree conclusion. Here's a list of the menu customization @functions.

- @ADDMENU("*MenuName*", *Position*)
- @ADDMENUITEM("*MenuName*", "*MenuItemName*", *Position*)
- @CHECKMENUITEM("*MenuName*", "*MenuItemName*")
- @DELETEMENU("*MenuName*")
- @DELETEMENUITEM("*MenuName*", "*MenuItemName*")
- @RESTOREMENU
- @UNCHECKMENUITEM("*MenuName*", "*MenuItemName*")

Although it doesn't change menus, there's another @function,

@SETTITLE("*Title*"), that changes the ObjectVision title.

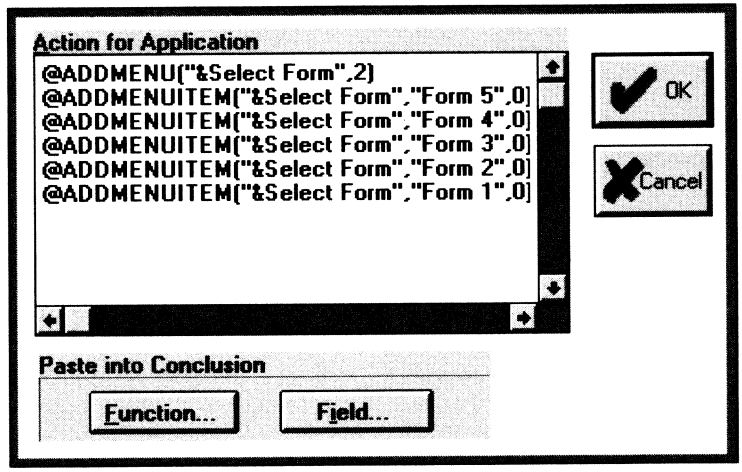
A stack event tree

If you want to customize your application's menu and have those menus working once a user opens your application, you'll have to create an event tree on the stack.

To create an event tree on the stack, open the Form Tool, then right-click on the application title bar.

The event condition is Open. The Open event is sent to the stack each time the application is opened, or if you send the event to the stack using @EVENT. The conclusion would be a series of the above @functions. For example, if you want to add a menu name called Select Form with item names for each form in the stack, your event tree conclusion would look like Figure 9.1.

Figure 9.1
Adding one menu name



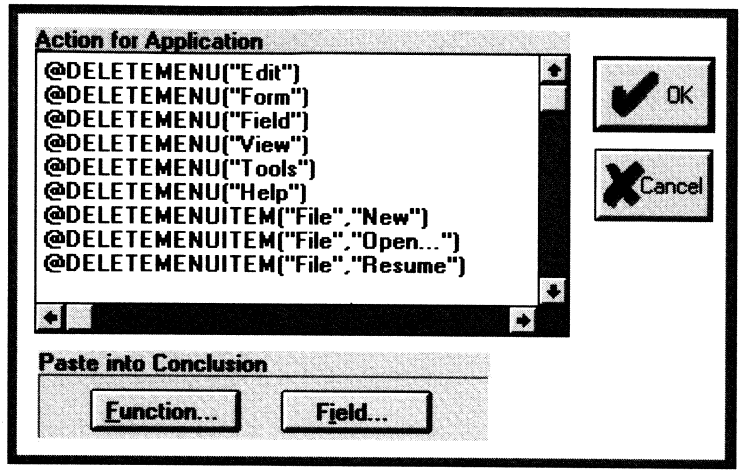
You can combine any set of @functions in one multilined conclusion. You can first delete menus, then add new ones.

If you want to delete the ObjectVision menus, you'll have to delete each menu name. By deleting the menu name using @DELETEMENU, you delete all the menu items below it. If you delete Edit, then Undo, Cut, Copy, Paste, and Clear All are also deleted.

If you only want to delete menu items, such as Edit | Clear All, you'd have to use @DELETEMENUITEM. The example in Figure 9.2

shows an event tree conclusion on the stack that deletes both menu names and items.

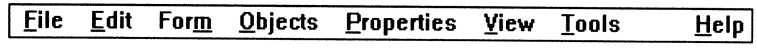
Figure 9.2
Deleting menu names and items



Using menu functions

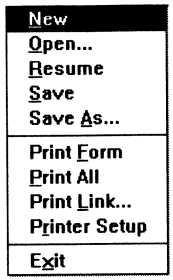
You can use the & symbol to place an underscore under the next letter of a menu name or a menu item. The underscored letter becomes the shortcut key—used with *Alt*. For an explanation of shortcut keys see Appendix E "Keyboard operations."

Figure 9.3
Using & in names and items



If you wanted to underline the F in File (as in Figure 9.3), you'd type `"&File"` for MenuName. Similarly, if you wanted to underline the m in Form, you'd type `"For&m"`. If you want to use the & symbol in the menu itself, you'll need to place two ampersands next to each other.

The position of a menu name or item is a number starting with 0 and continuing to the right for menu names, or continuing down for menu items. You can use as many menu names and items as you want, but for aesthetic reasons you might want to limit the number you use.



In Figure 9.3, File is in position 0, Edit is 1, Form 2, View 3, and Tools 4. If you use –1, ObjectVision will attach menu names to the end of the list after Help in the example. If you use 0 for the position of all menu names, ObjectVision will push existing menu names to the right and place the new menu name in position 0 (see Figure 9.1).

Items work the same. Under File, New is in position 0, Open 1, Resume 2, and so on.

To place a separating line in a list of items, like the one between Save As... and Print Form, you'd type empty quotes "" for the menu item (@ADDMENUITEM("&File","",5)).

Using other @functions in your menus

After you have created a new set of menu names and items, you'll want to give them the power to do something.

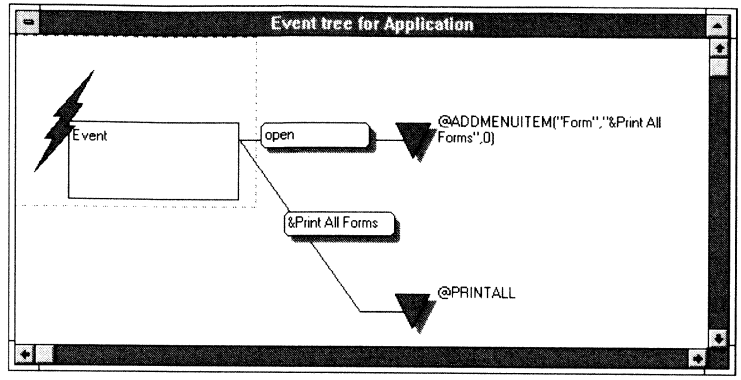
When you select a menu item you create, an event with the same name as the menu item is sent to the stack. You can create a branch on the stack event tree whose event condition is the same name as the menu item so that when you select the menu item, the branch and conclusion in the stack event tree are activated.

For example, if you want to create a menu item that prints all of the forms in a stack, follow these steps:

1. Create an event tree on the stack whose condition is Open. You can also use a control key and a letter, but for this example the menu and items will be created when the application is first opened.
2. To write the event tree conclusion, type `@ADDMENUITEM("Form", "&Print All Forms", 0)`.
3. Next, create another conclusion by clicking conclusion on the Object bar.
4. For the event condition, type `&Print All Forms`.
5. For the event conclusion, type `@PRINTALL`.

The event tree for the stack you just created should look like the one in Figure 9.4.

Figure 9.4
Creating menus



Restoring the menus

The function @RESTOREMENU restores all the default ObjectVision menus. A good way to activate the @RESTOREMENU is to add it to the Stack event tree as a conclusion to the Ctrl+R event. When you use the application, you can restore the menus simply by pressing *Ctrl+R*.

You can delete the Tools menu very easily, but then you won't be able to change the application unless you use an @RESTOREMENU.

If you delete the tools menu without adding an @RESTOREMENU to the application, you can still make changes to the application by opening the Form Tool with a *new application*, then choosing File | Open.

PART 3

Linking

C H A P T E R

10

Linking basics

This chapter introduces the Links Tool. The emphasis of this chapter is on basic linking concepts and creating a simple link.

ObjectVision can link to database tables (from Paradox, dBASE, and Btrieve), as well as to comma-delimited ASCII files and DDE-supported Windows applications. This chapter focuses on linking to databases. If you want to link to ASCII files or DDE-supported applications, you should read Chapter 15, "Linking to ASCII files," or Chapter 16, "Linking through DDE," because those types of links are slightly different from links to database tables.

After reading this chapter, you should read Chapter 11, "Linking options," which discusses linking options, and then read the chapters specific to the database you'll be linking to.

The Links Tool

To use the Links Tool, you start ObjectVision, open an application, then select Tools | Links. If you're designing a form at the same time you want to create links, you can use the Form Tool, then select Tools | Links or click Links on the Object bar.

The links you create can be to one database table or to several different tables with different formats.

When you use the Links Tool, you can create one link to a data file, or you can create multiple links without leaving the tool. You can also create database tables while using the Links Tool. For

more information on creating database tables, see Chapter 17, "Tips for designing database tables."

Linking overview

What is a link? A link is a way of connecting ObjectVision objects such as fields and columns to database tables. With linked ObjectVision objects you can receive, change, delete, and save values to the database table. You can create multiple links of different types (such as a dBASE and a Paradox link) or you can create multiple links to the same database type (such as seven Paradox links in one application).

Link connections

Links are made of connections. A link can have read connections, write connections, or both. Read connections allow the link to pass values *from* the database table to the linked ObjectVision fields.

Write connections allow the link to take values from the ObjectVision fields and write (save) them to the database table.

You can also create read-write connections that allow the link to first read values from the database table, display them in an ObjectVision form where you can edit them, and then save them.

Once the link is connected, values are exchanged between ObjectVision and the database table through these connections. The exchange takes place when *link functions* are performed. For example, @TOP, @BOTTOM, @PREVIOUS, @NEXT, and @STORE are link functions.

Link classification

ObjectVision classifies the links you create as *common*, *primary*, or *secondary*. You create the link, then ObjectVision classifies the link according to what the link is connected to—fields or columns—and in what order the link was created:

Common link	is a link connected to only ObjectVision fields (not to columns in tables).
Primary link	is the first link connected to a column in a table. There can only be one primary link to a table.

This link works with the table directly. This link is responsible for delivering multiple records to the ObjectVision table. When you perform an action on a table (such as pressing *F1*), the table communicates with the primary link.

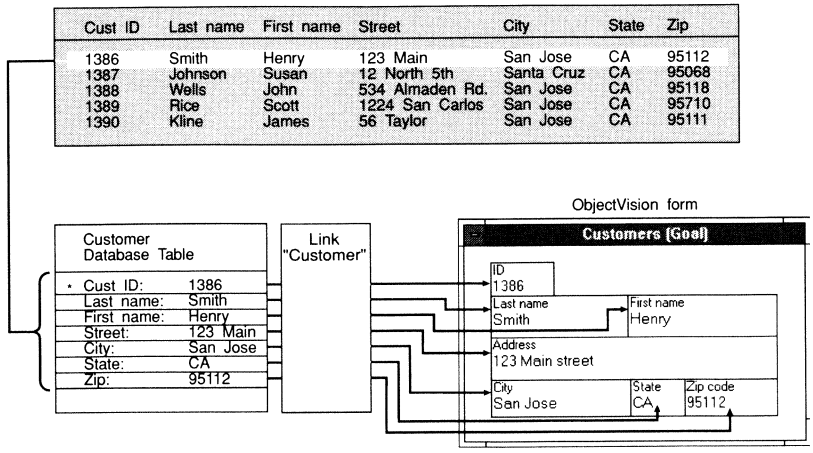
Secondary link is any link connected to a column in a table that already has a primary link. There can be more than one secondary link to a table. The secondary links look up and deliver values for each table row after the primary link has delivered its values. For example, if you had a table where you'd display book orders of customers, the primary link could deliver Items, Dates, and Volumes, then the secondary link could look up titles based on the values of Volumes. See the sample application CUSTORDS.OVD.

When creating links, connect all common links first, then primary links, and finally any secondary links. All link classes are described later in this chapter.

Read connections

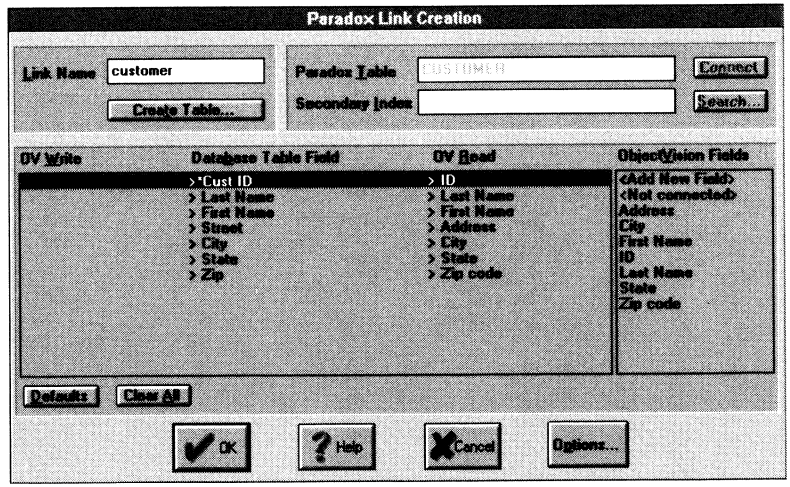
Figure 10.1 shows how ObjectVision reads values from a database table through read connections after a link function is performed, such as @TOP. The database table is a file that contains values. ObjectVision can read those database values and display them on an ObjectVision form. The database table is broken up into fields. ObjectVision links are connections between ObjectVision fields and database table fields. The database table *stores* the values, and ObjectVision *displays* the values.

Figure 10.1
How a read connection works



When you connect to a database table, all the fields in the database table display under Database Table Fields. If you make a read connection from a database field to an ObjectVision field, the ObjectVision field appears to the right of the database field under OV Read as shown in Figure 10.2. Notice that the ObjectVision read fields correspond to the field names on the ObjectVision forms. The Database Table Fields correspond to the field names in the database table. The ObjectVision field names don't have to match the database table names. The entire dialog box will be discussed later in the chapter.

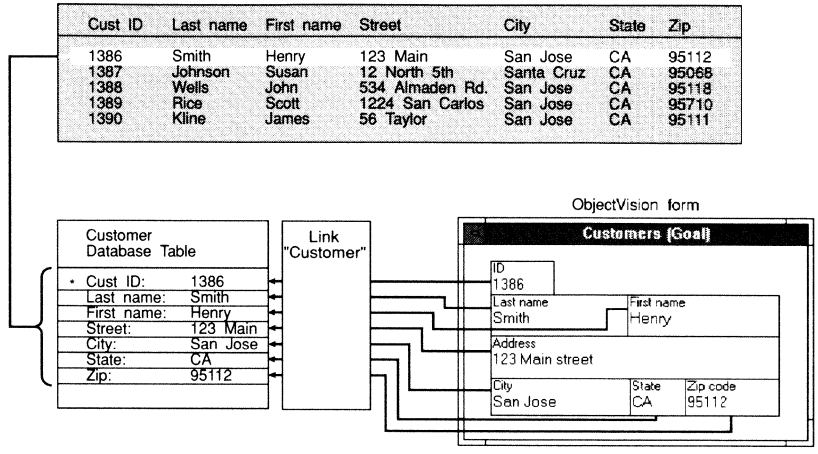
Figure 10.2
A read connection



A write connection

Figure 10.3 shows how ObjectVision writes values to a database table through write connections after a link function is performed. You type the value in the ObjectVision field, then use a link function to save the value to the database table.

Figure 10.3
How a write link works

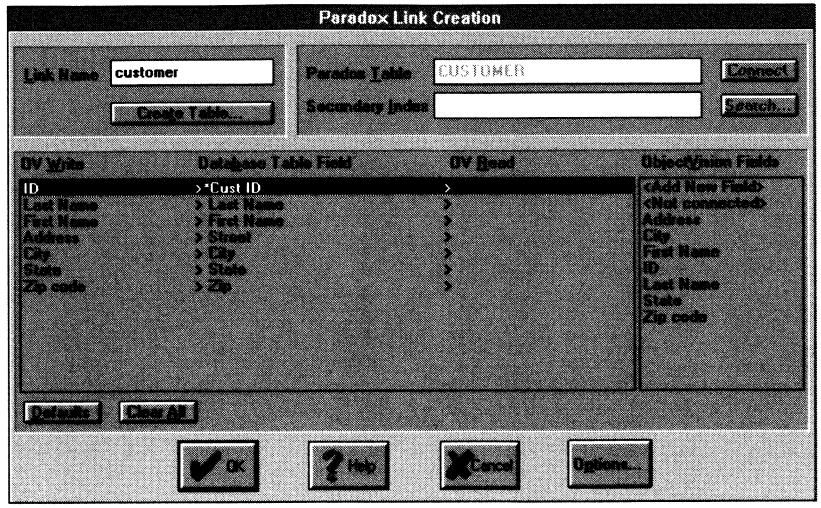


If the link is positioned on a record (that is, a link function has been performed), you can change the value that displays in the ObjectVision field, then update the record that is stored in the database table to match the value in the ObjectVision field.

While you're writing values to a database table, you can also insert new records. You can create new records by typing new values in the ObjectVision fields, then inserting those values into the database table by using the link function @INSERT. Using the data from Figure 10.5 as an example, you could first change Henry Smith's address by editing the ObjectVision fields, then update the existing record through write connections to the database table.

The ObjectVision fields that write values to the database table are displayed under OV Write in the Link Connections dialog box shown in Figure 10.4.

Figure 10.4
A write connection

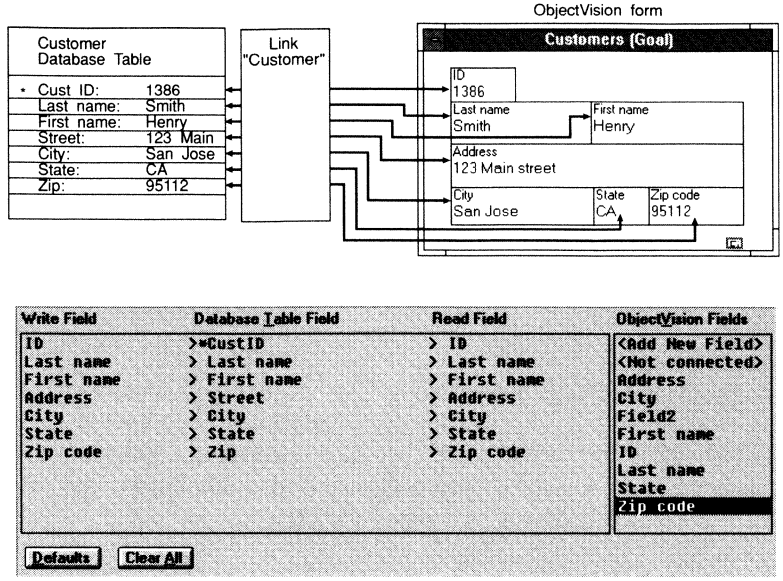


A read and write connection

Figure 10.5 shows the read and write connections to a database record. Using the read and write connections, the link functions can first read values from the database table (using an @TOP for instance), then change the values using the write connections (using an @STORE for instance).

Figure 10.5 contains only the connection section of the Link Connections dialog box.

Figure 10.5
How a read and write connection works



If the ObjectVision fields are read and write connected to a database table, the ObjectVision fields display on both sides of the database fields in the connection list box.

Note that you don't have to make read and write fields the same. You can link a read field to one ObjectVision field, then update or change the information and save it from another ObjectVision field.

Linking to an ObjectVision table

Connecting to a table column is just as easy as linking to a single field. When you link to a table column, the table communicates with the primary link telling it how many values to deliver to the table column.

The first link connected to the table is called the primary link. There can only be one primary link. All other links to that table are considered secondary links.

By using ObjectVision tables, you can display more than one value at a time. However, each table column can only have one current value at a time. This is the value displayed at the row pointer.

The row pointer is a triangle that displays to the left of the current row of values in the table. The row pointer represents the position on the linked database table. If you have a current position on the database table when you use an @function, such as @NEXT, the row pointer moves down the table as well as down the linked database table.

Any functions or trees using the value of a column use the value displayed in the current row.

Table connections

When you link to only ObjectVision fields, you create a common link. The link can read or write values from the database table. But when you connect a link to an ObjectVision table, there are two types of links. The link is classified as either a primary link or a secondary link.

Primary link

A primary link is a link on a ObjectVision table. When there is a link to a table, the first link connected to that table is considered the primary link. It isn't necessary to have more than one link to an ObjectVision table.

The primary link is responsible for communicating with the ObjectVision table. The table tells the primary link how many rows of values to display.

Any action performed on the table (pressing *F7* for instance) is communicated to the primary link, which then performs the action. For example, if the table object is selected in form completion mode, and you press *F4* (a Next), the primary link moves to the next record in its database table.

Anytime the table row pointer changes position by either mouse clicks or function keys (see page 195), the table communicates the change to the primary link. It is also possible to reposition the primary link by using link @functions.

Figure 10.6
A primary link

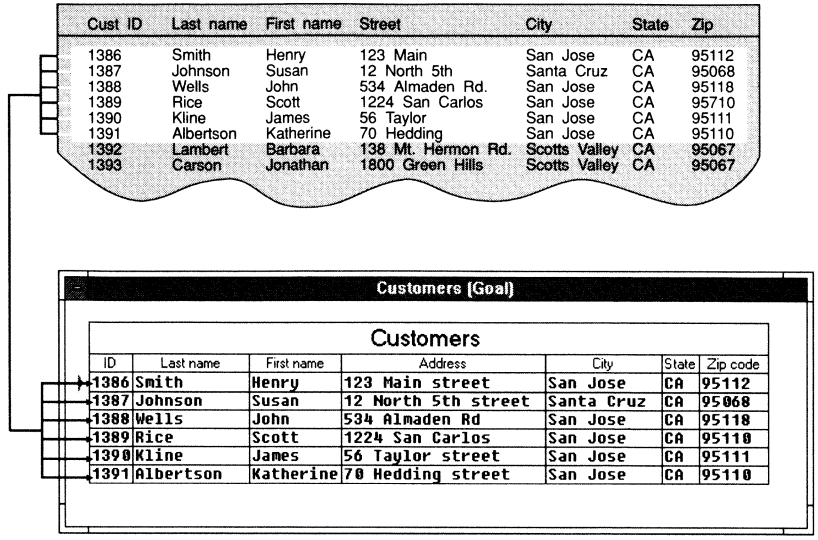


Figure 10.6 shows a link between a database table and an ObjectVision table. Notice that the link is considered primary because it is the first (and in this case, only) link to the ObjectVision table. The link delivers multiple records to the ObjectVision table. ObjectVision reads the values from top down in the database table beginning with the current position.

Linking columns to database tables is the same as linking fields with one exception: the order that you create the links is important.

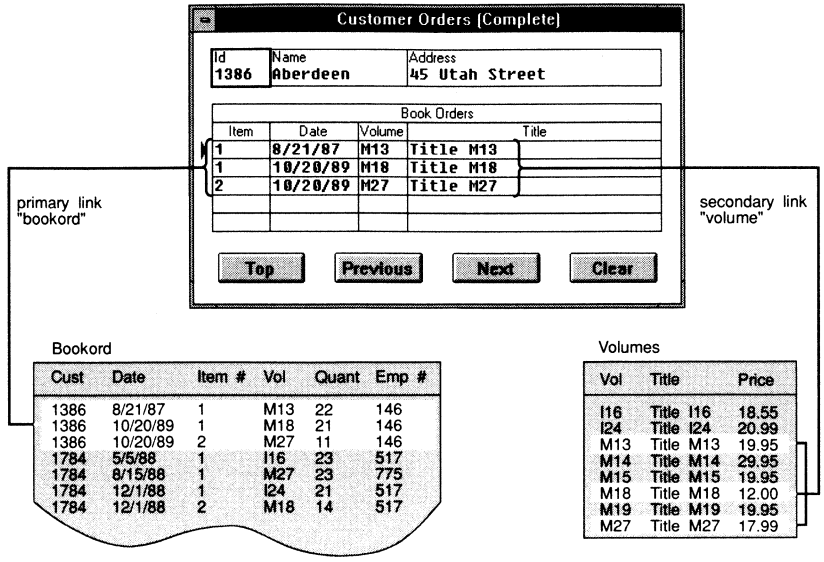
If there are multiple links to a single ObjectVision table, the *first* link created (the primary link) *must* be the one that delivers the multiple records to the table's columns. All other links (secondary links) to the table can be created in any order—they simply fill in information about the values delivered by the primary link.

Secondary link

When more than one link exists *to a table*, there is *one* primary link. All other links to the table are considered secondary links.

Secondary links deliver values after the primary link has delivered its set of values. You can't use link functions with secondary links except @LOCATE or @CLOSE. The secondary link is exclusively used for looking up values.

Figure 10.7
A secondary link



The example in Figure 10.7 shows how the primary link, "BOOKORD" in CUSTORDS.OVD delivers its values first. The primary link delivered three book orders based on the ObjectVision field ID. As the primary link delivered its values to the table, the secondary link looked up each title based on the values in Volume. "Bookord" and "Volumes" are two separate database tables. Each table contains a Vol field that is used to locate values.

Positioning on a database table

After a link is established, the link must be told where to position on the linked database table. You use @functions to tell the link where to position. For instance, you can use @TOP to position on the top record in the database table. All link functions that position use the read connections to deliver values after the link has positioned on the database table.

If the link is primary and a cell in the connected table is the selected field, you can use function keys to reposition the link. See the next two sections for tables of @functions and function keys that are used to reposition a link.

Using @functions to position

The following functions are used for positioning in the database table:

@TOP moves to the first record in the linked table. The first record's values are sent through the read connections to the corresponding ObjectVision fields. If the link is primary, then multiple records will be delivered to the ObjectVision table from top down.

@BOTTOM moves to the last record in the table, then reads the values and sends them back to the linked ObjectVision fields. If the link is primary, it delivers as many records as it can beginning from the bottom of the database table up.

@NEXT moves to the next record in the table. If the link is positioned on the last record when you use @NEXT, you'll see the message "End of table." If there isn't a current record, you'll see "Cannot reposition at this time. No current record."

@PREVIOUS moves to the previous record in the table. If the link is positioned on the first record when you use @PREVIOUS, you'll see the message "Start of table." If there isn't a current record, you'll see "Cannot reposition at this time. No current record."

@PAGEDN moves to the next page in the database table. A page is defined as the number of columns in the ObjectVision table minus one.

@PAGEUP moves to the previous page in the database table. A page is defined as the number of columns in the ObjectVision table minus one.

@CLEAR clears all read fields on the link. After using this function, the link no longer points to a current record.

Using function keys to position

Several function keys work like @functions for positioning a link on a database table. These keys work only in the selected table and with the primary link to that table.

Function key	@function	Action
F3	@PREVIOUS	View previous record (if there is one)
F4	@NEXT	View next record (if there is one)
F5	@PAGEUP	View previous records (number of rows – 1)
F6	@PAGEDN	View the next records (number of rows –1)
F7	@TOP	View first record
F8	@CLEAR	Clear all records from the table object

You can open the sample files to practice moving around in tables. The CUSTTAB.OVD file contains buttons that use all of the link functions for moving in a table.

Editing database values

Once a link is created, you can edit or update database table records by using the following functions:

@INSERT takes the values in the write-connected ObjectVision fields and attempts to insert a record with those values into the database table. If the record can't be inserted, a message will appear (such as "key exists").

@UPDATE takes the current record and attempts to change it based on the values in the current ObjectVision write fields. You can edit the values in the write fields, then use @UPDATE to enter the changes into the database table.

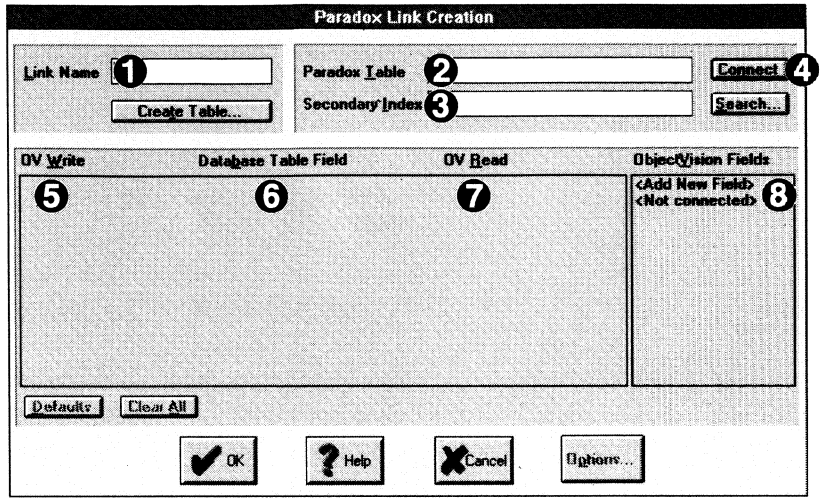
@DELETE removes the current record from the database table, moves to the next closest record in the database table, then does a read. This makes it possible to simply click delete multiple times to remove many records.

@STORE is a combination of insert and update. If there isn't a current record, then @STORE works like an insert. If there is a current record, @STORE works like an update.

Creating a link

1. To create a link, open the Links Tool by clicking Link in the Form Tool or choosing Tools | Link. From this box you can also delete or modify existing links. These buttons are described later in this chapter.
2. Select a link type (Paradox, dBASE, Btrieve, ASCII, or DDE) then click Create. The following instructions are for creating a Paradox link.
3. In the next dialog box, type all the information for the link, then click OK.

Figure 10.8
The Link Connections dialog box for a Paradox link



The following numbers correspond to the positions indicated above:

1. Type a link name then press *Tab*. The link name can be up to 80 characters and can include any ANSI character (see Appendix D).
2. Type the name of the database table, then press *Tab*. It is not necessary to include the extension. If the database table isn't in the current directory, type the entire path name. You can also use the Search button to search for a table (see "Searching for a database table" later in this chapter).
3. If you're linking to a database table that has a secondary index that you want to use, type the name of the index here. If the table doesn't use a secondary index or if you don't want to use a secondary index, leave this area blank. Press *Enter* to connect to the database table.
4. Once the database table is connected, all of the field names for the connected table will appear under Database Table Fields. An asterisk appears next to the field name if it is part of the index.
5. All Write connections will display in this column.
6. Once you click Connect, the database fields will display in this column.
7. All Read connections will display in this column.

8. All fields and columns in your ObjectVision application will display here. You can connect them to the database fields table fields. See the next section for instructions.

Connecting the ObjectVision fields

There are three ways to connect ObjectVision fields to database table fields.

First, make sure you're linked to a database table and that all the database table fields are displayed.

Connecting read or write only

To connect an ObjectVision field to a database table field, highlight the ObjectVision field on the right.

If you want a write connection, double-click under the OVWrite column next to the database table field you want to connect to. If you want a read connection, double-click under the OVRead column to the right of the database table field you want to connect to.

Connecting read and write

There are two ways to make a read/write connection. The first way is to highlight the database table field, then double-click the ObjectVision field you want to connect to the highlighted database table field.

The second way is to highlight an ObjectVision field, then double-click the database table field you want to connect to. The ObjectVision field's name will appear on both sides of the database table field.

After you connect the database table field, the highlight bar moves down to the next database table field. A quick way to connect all the fields is to double-click an ObjectVision field, then once the highlight moves down to the next database table field, you double-click the next corresponding ObjectVision field.

Link Name

The link name can use any ANSI characters and be up to 63 characters long. A link name identifies a link between a database file and your application.

It's a good idea to choose a name that helps you remember the purpose of the link. For example, if you're creating a link to

constant values in a lookup table, you might want to name the link *LOOKUP VALUES*.

File Name

The name of the database table must be a standard DOS file name, with a maximum of eight characters for the name (except for Btrieve tables). Database tables have an extension after the file name, but is isn't necessary for you to type the extension.

If the table you're linking to isn't in the same directory as your ObjectVision application, you must include the path name. The full path name can be up to 63 characters long.

Using <Add New Field>

If you double-click <Add New Field> in the ObjectVision fields list, you can create a new field. A dialog box appears asking for the new field name. Type a new field name, then click OK or press *Enter*. The new field is read and write connected to the highlighted database table field.

Make sure you add this new field to your form. If you don't place it on a form, ObjectVision places it on a Scratchpad form.

Using <Not Connected>

You can disconnect connections in the same three ways you create connections.

If you only want to clear some of the connections, highlight one of the connected rows, then double-click <Not Connected>. The ObjectVision fields on the highlighted row will be disconnected. You can also highlight <Not Connected>, then double-click the database field you want to disconnect.

If you only want to disconnect one of the connections, such as the OVRead connection, highlight <Not Connected>, then double-click under the OVRead column, next to the database table field you want to disconnect.

Using Defaults

The default button in the Link Connections dialog box will connect any ObjectVision fields whose names exactly match the field names in the database table.

If you want to connect matching fields, click Defaults. The matching fields will display as both read and write connections.

Any connection you've already made will not be affected when you use Defaults.

Using Clear All

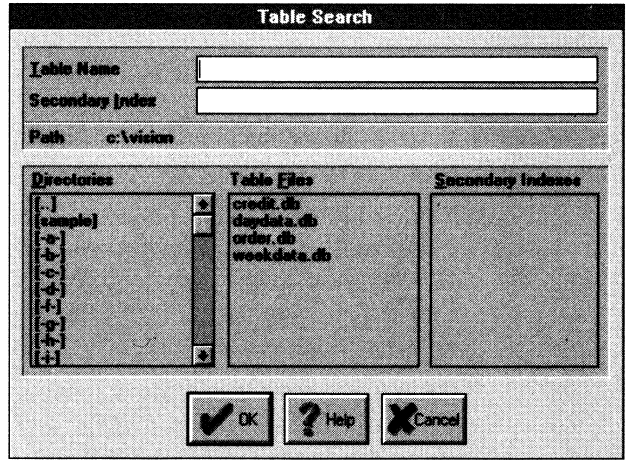
Clear All, when clicked, will clear all Link Connections between ObjectVision fields and database table fields.

Searching for a database table

In the Link Connections dialog boxes, there is a Search button that allows you to search by directory and drive for a database table.

ObjectVision uses the default table extensions to list all the tables in a directory or the Btrieve data dictionary in the current path as shown in the dialog box. You can choose to search different drives or different directories. Figure 10.9 shows the Paradox Table Search dialog box.

Figure 10.9
The Table Search dialog box



The current directory is displayed after you click Search. To move around the directories, double-click on a subdirectory name under Directories.

The double periods [..] let you go back one directory at a time. For instance, if you have a directory called VISION with a

subdirectory called APPS, which is your current directory, and you double-click [..], you'll be returned to VISION.

All of the database tables with the default extensions are displayed under Table Files. If the table you want is listed, highlight that name.

Once a database table is highlighted, any indexes belonging to that table are displayed under Secondary Indexes. If you want to use one of the displayed indexes, highlight it, then click OK to exit the Table Search dialog box.

Creating a database table

See the appropriate chapters for information about the table types you want to create.

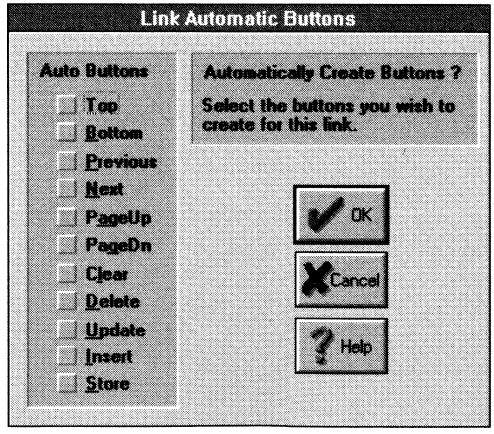
If you want to create a database table, select Tools | Links, choose a database format, then in the dialog box of the link type you're creating, click Create Table.

You may want to read Chapter 17, "Tips for designing database tables," which discusses some database terminology and describes the step-by-step procedures for creating a database table.

Link automatic buttons

Once you've finished creating the link and you click OK, a dialog box appears asking for automatic buttons.

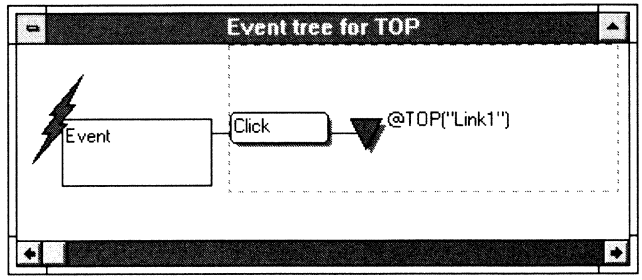
Figure 10.10
Link Automatic Buttons dialog box



You can check any highlighted box and ObjectVision will automatically create buttons with the link name that you just created.

ObjectVision only highlights the buttons you can use with a particular link type. Each button is created with an event tree whose condition is Click and whose conclusion is the @function name of the button for the link.

Figure 10.11
Event trees for link buttons



Modifying an existing link

If you want to link to another table, you'll have to create another link.

When you first select the Links Tool, a dialog box appears that lists any current links. To modify an existing link, highlight the link name, then click Modify or double-click the link name.

You can follow the instructions in the section "Creating a link," earlier in this chapter, to edit an existing link. You can change the link name and change the connected fields, but you can't change the table name or any secondary indexes.

Deleting a link

To delete an existing link, open the Links Tool, highlight the link name, then click Delete. You might want to delete a link if you decided to change linked database tables. Since you cannot modify a link to connect to another table, you'll need to create a new link, then delete the old link.

C H A P T E R

11

Linking options

This chapter describes the linking options. You should be familiar with linking and writing expressions before you read this chapter.

Options overview

When you create a link and you're using the Link Connections dialog box, you'll see the Options button next to Cancel. When you click this button, you'll go to a set of dialog boxes that let you alter the options and properties of the links you create.

There are five options: Auto Insert, Auto Update, Secondary Lookup, Cascade Deletes, and Cascade Updates.

There are three properties: Locates, Filters, and VFields. Since options are based on properties, properties are discussed first.

Locates

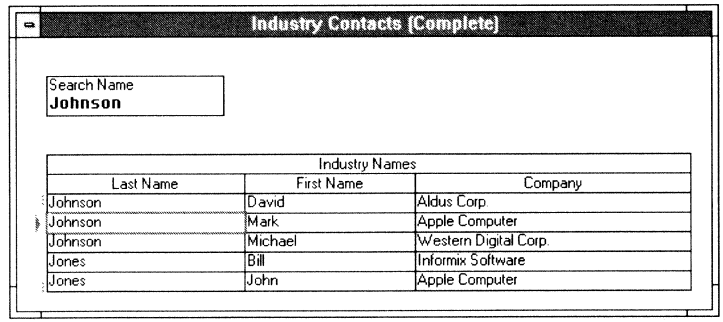
Locates monitor ObjectVision fields that are connected to database table fields. When a ObjectVision field's value changes, the locate immediately goes to the database table and attempts to position on that record. After the link repositions on the database table, the link reads the values from top to bottom beginning at the new location.

If there is a value change in the connected ObjectVision field, the link notices the changes and may perform an operation depending on the locate options you select.

To use locates with your database tables, your tables must be indexed and you can only locate on an indexed field.

For example, Figure 11.1 shows the Industry Names table from the CONTACTS.OVD application. The table is linked to a database table of contacts. (This sample application is included with ObjectVision.) The database table is indexed on Last Name and First Name. Search Name is monitored by a locate and is connected to the database field Last Name. It is linked to a field on the same database table of customers. When the value of Search Name changes, the link monitoring the field through a locate attempts to position itself on that value in the database table. The link to the ObjectVision table then reads values beginning with the new position on the database table (see Figure 11.2).

Figure 11.1
Using Locates with tables



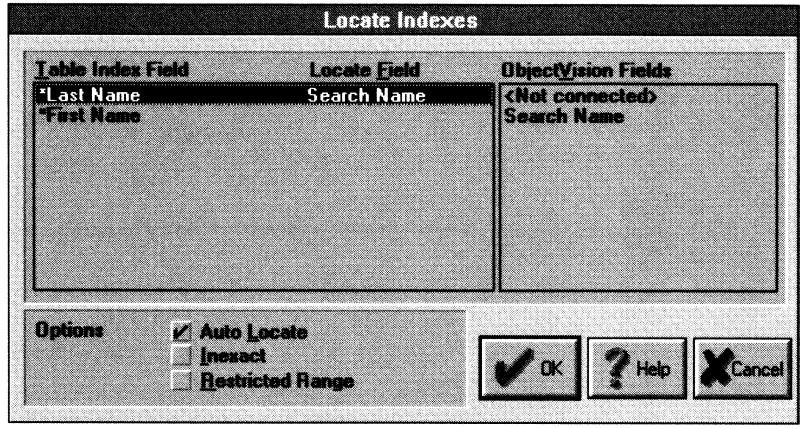
You can open CONTACTS.OVD to see how you can use the Locate options described below.

1. You might want to type a name in Search Name then have the link position on that record in the database table and read the values beginning with that position in the database. You'd do this with an Auto Locate.
2. If you don't know the exact name to search with, you might want to type only a letter, have the link position on the first record that begins with that letter, then read the values in the database table from that position on. You'd do this with Auto Locate and Inexact checked.

3. If you only want to see records that *exactly* match what you type in Search Name, you'd check Auto Locate and Restricted Range. If you typed "Johnson" in Search Name, then you'd only see records with Johnson as the Last Name.

Figure 11.2 shows the Locate dialog box for the table shown in Figure 11.1. You can open the CONTACTS.OVD application and check Inexact or Restricted Range to see how each option works. To get to this dialog box, open the Links Tool, double-click Links (the link name for this example), click the Options button, then click Locates.

Figure 11.2
Using the Locates options



The Table Index Field area lists the database table fields that are part of the link index. The asterisks (also used in Link Connections dialog boxes) mark that the field is part of the index. Locate Field lists the ObjectVision fields that are locate-connected to the database table index fields. To make the connection, highlight the table index field, then double-click the ObjectVision fields that can be monitored by the link. Not all ObjectVision fields are listed. Primary links cannot locate on ObjectVision columns that are part of its table. Common and secondary links can locate on any field or column.

The Locates options are described in more detail in the following sections.

Auto Locate

If you check this option, the link attempts to position on the record on the database table that matches the new ObjectVision

field value. This occurs automatically every time the value in the connected ObjectVision field changes.

This option is checked by default. If you uncheck Auto Locate, the link won't perform any operations when there is a value change in the connected ObjectVision field.

Inexact

Inexact is a locate option that searches for an inexact match. This option can't be used at the same time that a restricted range is used because it's impossible to look up an inexact indexed value whose range is then restricted.

For example, if Inexact was checked in the Figure 11.2, you could type J in Search Name and get any record that starts with J.

The Inexact search is dependent on how each database type defines an inexact search because each database type is different.

Restricted range

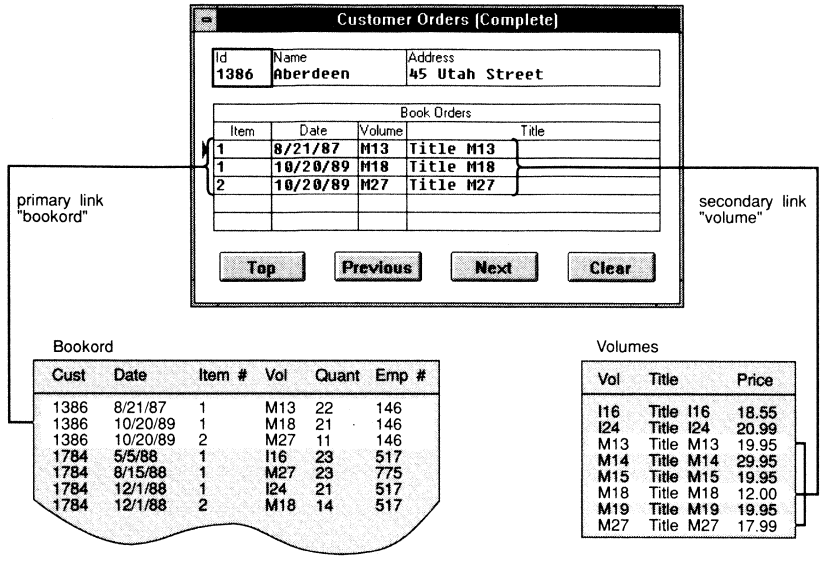
You can modify the locate to only accept a restricted range of values. In the example in Figure 11.2 you could restrict the range to just the Johnson's. Then you wouldn't see records for Jensen or Jones.

Restricted range doesn't work with Inexact because you can't restrict the range of something that is inexact.

Typing Johnson in Search Name causes the link to locate Johnson in the linked database table if Auto Locate is checked. If Restricted Range is also checked, the only values that pass through the link are records whose Last Name *exactly* match Johnson. Each connected locate field, and *only* locate fields, are checked for exact matches.

Restricted range allows you to make relationships between links as shown in Figure 11.3:

Figure 11.3
Customer Orders



In Figure 11.3, there are three links. The first link is a common link that delivers values to ID, Name, and Address. The second link is a primary link called "Bookord" that delivers values to Item, Date, and Volume to the Book Orders table. The third link is a secondary link to the Book Orders table that delivers values to Title.

"Bookord" is restricted in range to only show values that exactly match the locate field ID. Anytime ID changes, "Bookord" Auto Locates all records that exactly match the new ID field value.

Filters

A filter expression can be any ObjectVision expression.

Filters allow you to type a logical expression that is evaluated before a record is passed through the link. Filters have options that let you filter values coming from a database table into ObjectVision, or going from ObjectVision into a database table, or both.

If the filter evaluates to a nonzero value (the value passes the filter), then that record is delivered through the link. If the expression evaluates to zero (the value does not pass the filter), then the link either positions to another record in the database table and evaluates it (if the filter is a read filter), or a message appears telling you that the record you're trying to send to the database table doesn't pass the filter (if the filter is a write filter).

Filters can work with unindexed database tables. This is slower than a locate because each record must be evaluated with the expression before it can be passed to ObjectVision.

You should use locates whenever possible—especially with indexed tables—but there may be times when you'll need a filter.

You can have more than one filter for any link you create, but only one filter can be active at a time. You can select a filter in the Links Tool, or you can select a filter using the @functions @FILTERACTIVATE and @FILTERDEACTIVATE.

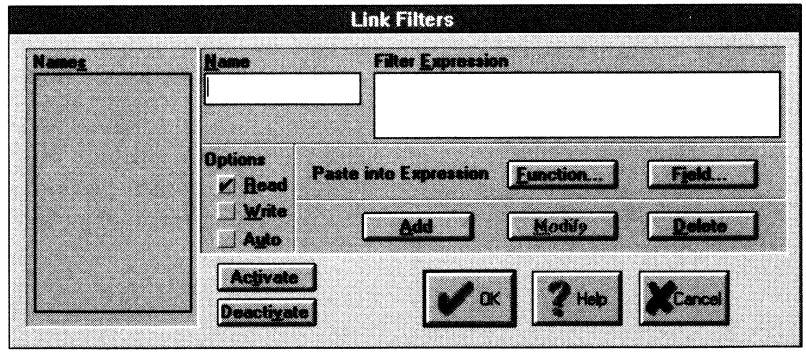
For example, if you have a database table of customers linked to an ObjectVision application, you could filter so you'd only view customers from California. This filter expression might look like (STATE="CA") where STATE is an ObjectVision field that is read-connected to the database field STATE.

If the database field value you want to use in the filter isn't read-connected to ObjectVision, you must use @LINKVALUE. For example, if the database field STATE isn't read-connected to ObjectVision, the filter expression might look like (@LINKVALUE("STATE")="CA").

Creating a filter

When you click Filter from the Link Options dialog box, you'll see a dialog box where you can write filter expressions, select filter options, and activate or deactivate filters.

Figure 11.4
The Filter dialog box



To create a filter, follow these instructions:

1. In the Name box, type the name you want to give to the filter.

2. In the Filter Expression box, type the expression for the filter. You can click Function to paste a function to the expression. You can also click Field to paste a field to the expression. To learn how to write expressions, see Chapter 6, "Writing expressions."
3. Click Add to add the filter you just created to the list of filters. The first filter you create automatically becomes the active filter, and an asterisk appears next to its name to show that it is the selected filter. Any other filters you create are not active.

Using Modify and Delete

To modify an existing filter, highlight the filter name under Names. The filter name and expression appear to the right, where you can edit them. You can edit the expression or name, then click Modify.

To delete a filter from the list, highlight the filter, then click Delete.

To undo a deletion, click Cancel, then click Filter to return to the Filter dialog box.

Using Activate and Deactivate

Since you can have more than one filter for each link, you can activate a new filter or deactivate the current filter.

To activate a filter, highlight its name in the list of filters, then click Activate. If another filter is active, it will be deactivated because only one filter can be active at a time.

To deactivate a filter, click Deactivate. The active filter, selected or not, becomes deactivated.

You can create several filters, but you don't have to have one active. You can have all filters deactivated if you want.

The Read option

The Read optino in the Filter dialog box uses the filter expression only when a record is read from a database table to ObjectVision.

The link evaluates the active filter expression for each record on the database table. If the expression evaluates to nonzero (true), the record is sent through the link to ObjectVision. If the expression evaluates to zero (false), the record does not pass the filter expression and the link positions to the next record and evaluates it.

The Write option

If the Write option is checked, the filter expression is used during writes from ObjectVision to the linked database table.

If the record you want to save to the database table doesn't pass the filter expression, a message will appear saying that the record doesn't pass. You'll need to either correct the values in the ObjectVision fields so they pass the filter expression or change the filter expression.

You can check both Read and Write options to use the filter as a read filter, and again as a write filter when those records are written to the database table.

The Auto option

If there is a value change in any ObjectVision field that appears in the filter expression and Auto is checked, the link will automatically perform a Top on the linked database table and begin evaluating records if the filter is active.

For example, in Figure 11.1, there is a locate connected to Search Name. If you changed the locate to a filter and checked the Auto option, then every time you type a new value in Search Name, the link will perform a top on the database table and return any values that pass the filter expression. The filter expression for this would be (Last Name = Search Name).

When this option is checked, every time a value changes in an ObjectVision field that appears in an active filter expression, the link performs a Top.

VFields

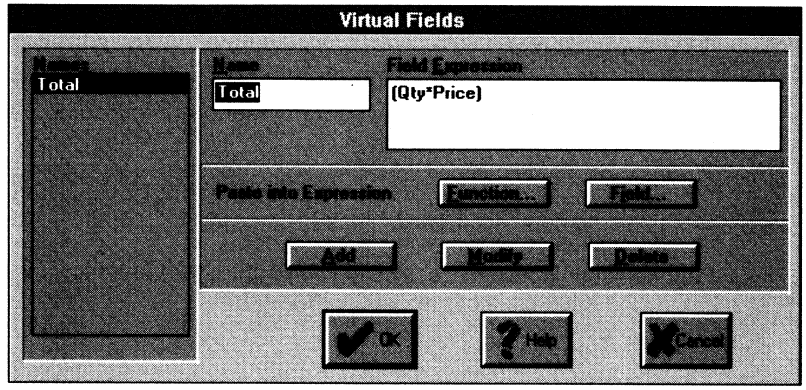
A VField is a virtual database field calculated from an ObjectVision expression. Technically, the field itself doesn't exist—only its expression exists, so the values in a VField aren't saved on the database.

The VField extends the database fields that you can connect to. Since a VField value can't be saved, *VFields can only be read-connected.* VFields display an equal sign next to the database field name in the Link Connections dialog box.

A VField is calculated each time a record is read from the database table by the link.

For example, open CUSTVIRT.OVD. This sample application is connected to a database table that has the fields Qty and Price. In CUSTVIRT, Price and Qty are multiplied. The result is placed in a VField called Total. Because Price and Qty database fields are read-connected to fields in ObjectVision, the *values* for Price and Qty that ObjectVision uses in the VField come from the current record in the link. If Price and Qty weren't read-connected, the values for the VFields would come from the ObjectVision field values.

Figure 11.5
Using VFields



ObjectVision links to the VField Total. Those values are read to the Book Orders table. Each time a record is delivered through the link, the expression (Qty*Price) is evaluated for the current record, and that value is used for the value of Total.

The field Sum of Total gets its value from a value tree with the conclusion @LINKSUM("bookorder","Total").

To get a value for Sum of Total without a VField, you could create a column that has a value tree with the conclusion @COLUMNSUM. However, this brings in the problem of only summing the *displayed* values.

VFields are noted with an equal mark instead of < or > in the Link Connections dialog box.

Creating a VField

When you click VField from the Link Options dialog box, you'll see a dialog box where you can write the VField expression. To learn how to write expressions, see Chapter 6, "Writing expressions."

To create a VField, follow these instructions:

1. In the Name box, type the field name for the VField.
2. In the Field Expression box, type the expression for the VField. You can paste functions and field names into your expressions. In Figure 11.5, the expression is (Qty * Price). Remember to use the correct expression syntax.
3. Click Add to add the VField you just typed to the list of VFields.

Using Modify and Delete

To modify an existing VField, highlight the VField under Names. The VField name and expression appear to the right where you can edit them. When you've finished editing the VField, click Modify.

To delete a VField, highlight the VField under Names, then click Delete.

To undo a deletion, click Cancel, then click VFields to return to the VField dialog box.

Auto Insert

This option automatically inserts a record when you've typed values in a blank record, then move off the record. This option works like @ INSERT. This is a quick and easy method for data entry.

Auto Insert works best with primary links, although it can work with common links. Auto Insert can't work with secondary links.

For example, if you've created a table on a form and you link it to a database table with write connections, you can type records in the ObjectVision table. The records will be automatically inserted to the database table as soon as you move off the record.

In an ObjectVision table, you move off the record by pressing *Enter* or by using *Tab* to get to the next row of the table. You can also use a Next to move to another record. Moving off the table by selecting another form object is also a way of moving off a record.

You can use Auto Insert to quickly add values to a database table. The record is sorted automatically if the database table is indexed.

Auto Insert is for inserting records only. If you want to read, edit, then write record values, use Auto Update. If you leave a record blank, ObjectVision does not insert that blank record to the database table.

Auto Update

This option automatically updates records in the database table. It works like @UPDATE, so if you use Auto Update, you don't have to use an @UPDATE button. Auto Update will only update values that are write connected.

How does Auto Update work? If the link is positioned on a database record, then you try to reposition the link (by using function keys, @functions, or mouse clicks) any changes in the ObjectVision write fields will be automatically saved before the link is repositioned.

This option works best with primary links. It can also be used with common links, but it can't be used with secondary links.

Secondary Lookup

If you check this option, the link you create becomes a secondary link to the table. This option works only for links to tables.

Why would you want to force a link to be secondary? In the example below, the table gets its primary values from a value tree. These values need to be the primary values for that table because they are the values that control the ObjectVision table.

Figure 11.6
Using Secondary Lookup

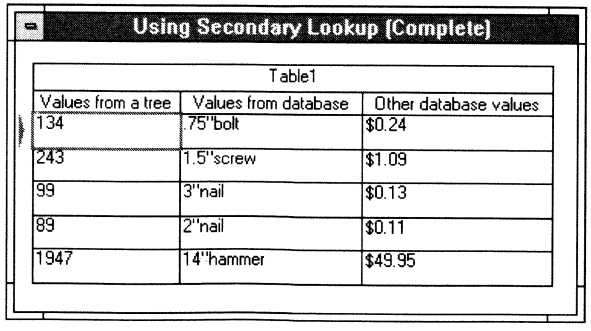


Table1		
Values from a tree	Values from database	Other database values
134	.75"bolt	$0.24
243	1.5"screw	$1.09
99	3"nail	$0.13
89	2"nail	$0.11
1947	14"hammer	$49.95

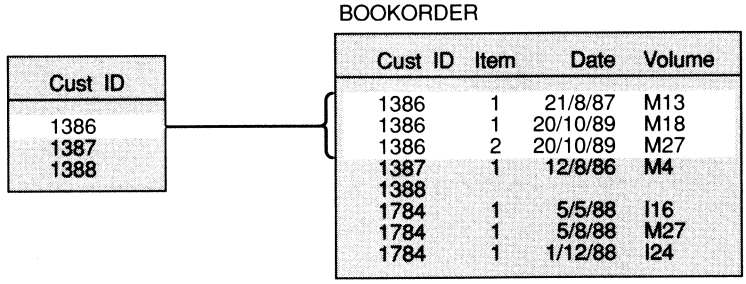
Column "Values from a tree" gets its value from a value tree. The values in Columns "Values from Database" and "Other database values" are looked up by the link. The link uses the values in "Values form a tree to locate the corresponding values in the linked database table."

When the link for "Values from database" was created, it was marked as primary because it was the first link to the ObjectVision table. The secondary lookup option forces the link to be secondary.

Cascade Deletes

If a record in a database table is deleted and Cascade Deletes is checked, then ObjectVision looks for any other link that *auto locates* on any ObjectVision read-connected field within the link. The figure below shows how Cascade Deletes work.

Figure 11.7
Using Cascade Deletes



You can't cascade from or to a secondary link.

The first link (read-connected to a database table of customers) is the link that controls the cascade. The second link (the Bookorder link, which Auto Locates on and restricts its range to CUST ID) receives the cascade delete. The receiving link deletes all records in its restricted range. If the receiving link has its cascade option checked, the link continues the cascade.

When the customer is deleted, ObjectVision looks for all the book orders that belonged to that customer.

Cascade Updates

Cascade Updates works like Cascade Deletes except it updates all Auto Located and restricted range links.

If you check Cascade Updates, any changes you make to a record are updated in any Auto Locate records. For example, in Figure 11.7, if you changed the customer number, ObjectVision goes to the book order table and updates all the book orders for that customer.

The link for Cascade Updates must be Auto Locate and Restricted Range.

C H A P T E R

12

Linking to Paradox tables

This chapter describes linking to Paradox tables. It assumes you've read Chapter 10, "Linking basics."

Paradox overview

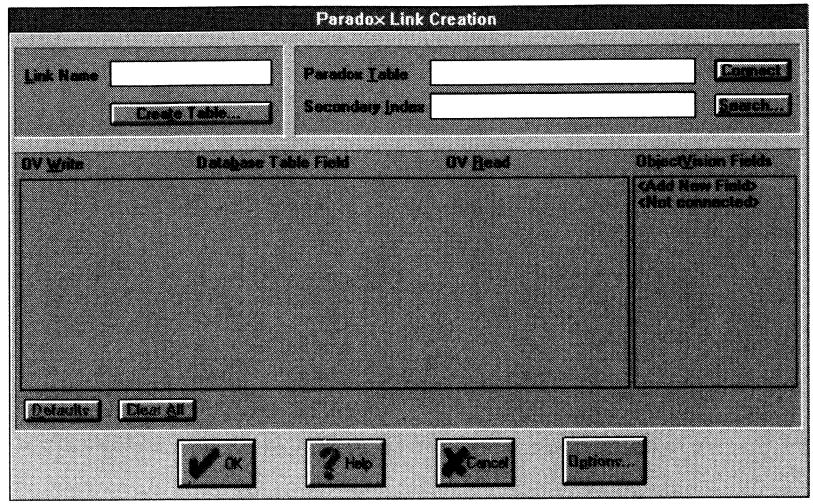
You can link to existing Paradox tables, or you can create your own Paradox tables. You don't need Paradox to create a table. See Chapter 17, "Tips for designing database tables," for information on creating database tables.

ObjectVision is 100% compatible with Paradox 3.0 and 3.5.

Linking to Paradox tables

To create a link to a Paradox table, select Tools | Links or click Link on the Object bar in the Form Tool. When the Link dialog box appears, select Paradox, then click Create. The Paradox dialog box shown in Figure 12.1 appears.

Figure 12.1
Paradox dialog box



To create the link, follow these steps. Some steps are described in more detail in Chapter 10, "Linking basics."

1. Type a link name in the Link Name box. Press *Tab* or select Paradox Table.
2. Type the name of the Paradox table file you want to link to. You don't need to type the .DB extension. If you don't know the name of the Paradox table file, you can use the Search button (see Chapter 10, "Linking basics," for information on this button). Press *Tab* or select Secondary Index.
3. Type the name of the secondary index field name you want to use. If you don't type in a secondary index, ObjectVision uses the primary index if there is one.
4. Press *Enter* or click Connect to connect to the Paradox table file. All fields in the table will display in the Links dialog box. For information on connecting ObjectVision fields to Paradox table fields, see Chapter 10, "Linking basics."

Primary index

ObjectVision automatically uses the primary index if one exists for a linked Paradox table unless you type a secondary index. The primary index file has a .PX extension and must be in the same directory as the .DB table file.

Secondary Index Field Name

Paradox supports secondary indexes to provide different sort orders. To use an existing secondary index, type the name of the single Paradox field that you want to use to locate records or control record sequence. You can use Search to find out if a table has any secondary indexes. In the Search dialog box, secondary indexes for the table display as soon as you select the table.

Automatic buttons

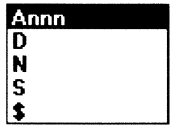
Once you're done connecting positions to ObjectVision fields, click OK. ObjectVision asks if you want to put buttons for the Paradox link on your form. You can select each button or just a few.

For more information on automatic buttons, see Chapter 10, "Linking basics."

Creating Paradox tables

If you want to create a Paradox table, you should read Chapter 17, "Tips for designing database tables."

This section describes the field types that Paradox uses. When you use ObjectVision to create a Paradox table, you'll see a Field Type box. When you click the arrow to its right, a list of the available Paradox field types will display as shown to the left. These field types are described below.



A*nnn* Alphanumeric of size *nnn* where *nnn* is less than 255. You must specify a length from 1 to 255 characters. Alphanumeric fields can contain almost any character including letters, numbers, punctuation marks, special characters such as +, –, %, and $, and any ASCII character (except null).

D Date. Date fields can only contain dates. Note that this field type can only contain dates from 1 January 100 to 31 December 9999. If you connect a Paradox date field to an ObjectVision date/time field, the time (decimal portion) will be lost. You should connect ObjectVision date/time fields to Paradox numeric fields.

N Number. This field type can be up to 15 significant digits (including decimal places) from $\pm 10^{-307}$ to $\pm 10^{308}$. Values with more digits are rounded and saved in scientific notation.

S Short numeric. This field type can only contain integers between –32,767 and 32,767. Short numerics use less disk space than normal numerics.

$ Currency. This field type works exactly like numerics in Paradox with only two decimal places.

C H A P T E R

13

Linking to dBASE tables

This chapter describes linking to dBASE and dBASE-compatible tables. This chapter assumes you've read Chapter 10, "Linking basics."

dBASE overview

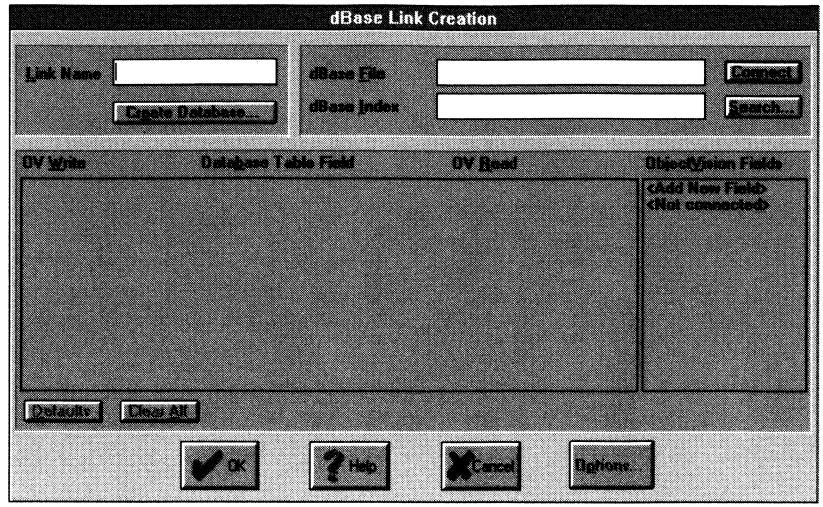
You can link to existing dBASE-compatible files or you can create your own dBASE tables.

ObjectVision is compatible with dBASE III, dBASE III+, dBASE IV (with the exception of the "float" field type), Clipper, and Fox .DBF files. ObjectVision doesn't support dBASE IV's .MDX index files, Clipper's .NTX files, or Fox's .IDX files.

Linking to dBASE tables

To create a link to a dBASE table, select Tools | Links or click Links on the Object bar in the Form Tool. Figure 13.1 shows the dBase dialog box that appears when you create a new dBASE link.

Figure 13.1
dBase dialog box



To create the link, follow these steps. Some steps are described in more detail in Chapter 10, "Linking basics."

1. Type a link name in the Link Name box. Press *Tab* or select dBASE File.
2. Type the name of the dBASE table you want to link to. You don't need to type the .DBF extension. If you don't know the name of the dBASE table, you can use the Search button (see Chapter 10, "Linking basics," for information on this button). Press *Tab* or select dBASE Index.
3. Type the name of the index you want to use. If you don't select an index, ObjectVision won't use an index with the dBASE table.
4. Press *Enter* or click Connect to connect to the dBASE table. All fields in the table will display in the Links dialog box. For information on connecting ObjectVision fields to dBASE table fields, see Chapter 10, "Linking basics."

Index files

An index file controls the sort order of the data file. A dBASE index file has the extension .NDX, but it isn't necessary to type the extension.

Automatic buttons

Once you're done connecting positions to ObjectVision fields, click OK. ObjectVision asks if you want to put buttons for the dBASE link on your form.

For more information on automatic buttons, see Chapter 10, "Linking basics."

Creating dBASE tables

If you want to create a dBASE table, you should read Chapter 17, "Tips for designing database tables."

This section describes the field types that dBASE uses. When you use ObjectVision to create a dBASE table, you'll see a Field Type box. When you click the arrow to its right, a list of the available dBASE field types will display. These field types are described below.

CHAR(*nnn*)	Character/text. This field can store characters, including letters, numbers, special symbols, and blank spaces. The field must be less than 254 characters. You control the number of characters using *nnn*. For example, CHAR(20) holds 20 characters.
NUM(*nn,nn*)	Numeric. This field can hold up to 15 significant digits. The first number (*nn*) is for size in characters. Any decimal points or negatives are included in the size. The second number is for decimal digits. For example, NUM(7,2) can hold a number like 1234.56, which is seven characters and two digits.
DATE	Date. This field only accepts dates between 01/01/0000 and 12/31/9999. The format for dates is mm/dd/yy.
LOGICAL	This field holds a single value that represents a true or false value.
MEMO	ObjectVision fields hold up to 4,096 characters. Although Memo fields hold more than that, if you exceed limit this you might cut off information when you link to ObjectVision fields.

C H A P T E R

14

Linking to Btrieve tables

This chapter describes linking to Btrieve tables. This chapter assumes you've read Chapter 10, "Linking basics."

Btrieve overview

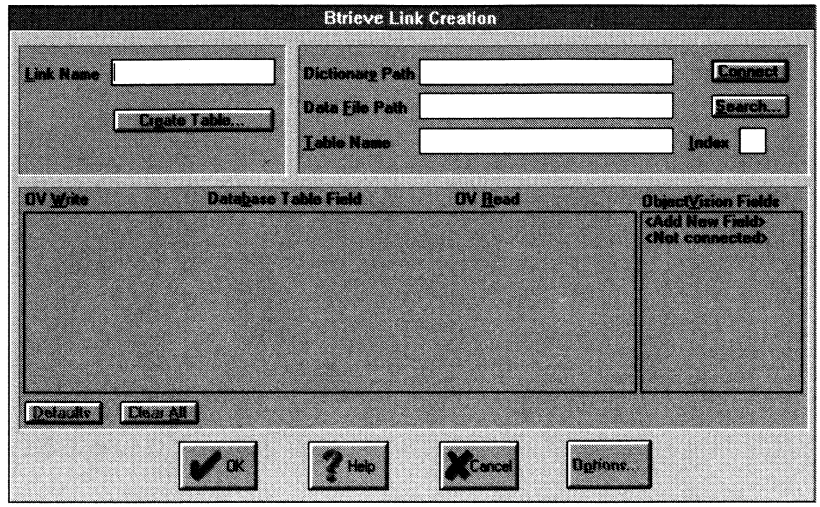
This chapter explains how you to link to existing Btrieve tables. It also describes the Btrieve field types used when you create a Btrieve table.

ObjectVision is 100% compatible with Btrieve tables.

Linking to Btrieve tables

To create a link to a Btrieve table, select Tools | Links or click Links on the Object bar in the Form Tool. Figure 14.1 shows the Btrieve dialog box that appears when you create a new Btrieve link. The steps for creating a Btrieve link are similar to those used to create other database links.

Figure 14.1
Btrieve dialog box



If you use ObjectVision to create your Btrieve database files, you don't need either Novell's Btrieve or XQL applications. The tables and the data dictionary files are automatically created using the information you supply in the Table Creation dialog box.

However, you will need Btrieve and XQL if you want to modify the file structure.

ObjectVision links to Btrieve data through Novell's data dictionary. If you have existing Btrieve files with no data dictionary, you must use Novell's Xtrieve application to create dictionary entries for your Btrieve files. If you use Novell's Xtrieve or XQL to create you data, the data dictionary entries are created automatically.

Searching for the data dictionary

ObjectVision first looks for the data dictionary in the path you specify when you create the link. If it can't find the data dictionary there, ObjectVision looks in the current directory.

Searching for the data file

When the data dictionary is opened, ObjectVision receives the data location and the name of the Btrieve data file as it is stored by Xtrieve or XQL. ObjectVision searches for the data file in this order:

1. the stored data path you typed in the ObjectVision link
2. the Data Dictionary Path you typed in the ObjectVision link

3. the stored data path in the data dictionary
4. the current directory

To create the link, follow these steps. Some steps are described in more detail in Chapter 10, "Linking basics."

1. Type a link name in the Link Name box. Press *Tab* or select Dictionary Path.
2. Type the path to the dictionary that contains the three .DDF files. When you attempt to open the dictionary, ObjectVision looks in the Dictionary Path. After you've typed in the path, press *Tab* or select Data File Path.
3. Type the path to the data file. If ObjectVision can't locate the dictionary in the path from step 2, it will look in the Data File Path. After you've typed the path to the data file, press *Tab* or select Table Name.
4. Type the name of the Btrieve table you want to link to. The table name must follow the XQL standard with a maximum of 20 characters. If you don't know the name of the table, you can use the Search button (see Chapter 10, "Linking basics," for information on this button). Press *Tab* or select Index.
5. Type the number (from 0 to 23) of the index you want to use. If you don't select an index, ObjectVision won't use an index with the table.
6. Press *Enter* or click Connect to connect to the Btrieve table. All fields in the table will display in the link dialog box. For information on connecting ObjectVision fields to Btrieve table fields, see Chapter 10, "Linking basics."

Dictionary path

The dictionary path directory must contain the set of three .DDF files (FIELD.DDF, FILE.DDF, and INDEX.DDF) that contain the field definitions for the table you want to link to.

The file name must be a standard DOS file name, with a maximum of 8 characters for the name. The full path name can be up to 63 characters long.

Table Name

The table name must be a standard XQL table name, with a maximum of 20 characters for the name. The table's field name and field type information must be contained in the Dictionary.

Index Number

The index number specifies which index you want to use. The index number is the number assigned when you created the index in (an existing index number from 0 to 23).

Automatic buttons

Once you're done connecting positions to ObjectVision fields, click OK. ObjectVision asks if you want to put buttons for the Btrieve link on your form. For more information on automatic buttons, see Chapter 10, "Linking basics."

Creating Btrieve tables

If you want to create a Btrieve table, you should read Chapter 17, "Tips for designing database tables."

This section describes the field types that Btrieve uses. When you use ObjectVision to create a Btrieve table, you'll see a Field Type box. When you click the arrow to its right, a list of the available Btrieve field types will display. Sizes are in bytes.

CHAR(*nnn*)	Character field of size *nnn*. The size of the field must be between 1 and 255, and can contain any ASCII character.
NUM(*nn,nn*)	Numeric field of size *nn* and decimal digits *nn*. The size must be between 1 and 15. The number is stored in Btrieve numeric format.
DEC(*nn,nn*)	Decimal number of size *nn*, where *nn* is between 1 and 10, and of decimal digits *nn*. The number is stored as a parked decimal.
INT(*n*)	Integers of size *n*. The size specifies a range for the integers: 1 is for integers from 0-255, 2 is –32,768-32,767, and 4 is approximately –2 billion to +2 billion.

FLOAT(*n*)	IEEE standard for single-and-double precision real numbers where the range is *n*. The range is either 4 for 5.9E-39 to 1.7E+38, or 8 for 1.1E-308 to 1.1E+307.
ZSTRING(*nnn*)	A string of size *nnn* where *nnn* is between 1 and 255. ZSTRING corresponds to a C string.
LSTRING(*nnn*)	A string of size *nnn* where *nnn* is between 1 and 255. LSTRING corresponds to a Pascal string.
MONEY(*nn*)	Decimal number with only 2 decimal places where *nn* is the size from 1 to 10. The number is stored as a parked decimal.
DATE	A date.
TIME	A time stored as Hours:Minutes:Seconds:Hundredths.
LOGICAL(*n*)	A logical value where 0 represents false and anything else is true. The number *n* represents the size of either 1 or 2 bytes.
BFLOAT(*n*)	A basic floating point.
NOTE(*nnnnn*)	ObjectVision can create the NOTE field, but it can only read and write 4096 characters.
LVAR(*nnnnn*)	ObjectVision can create the LVAR field, but it can only read and write 4096 characters.
AUTO(*n*)	Auto increment where *n*, can be size 2 or 4. Each time the field is read, its value increments.

C H A P T E R

15

Linking to ASCII files

This chapter describes ASCII files and explains how to link to them. This chapter assumes you've read Chapter 10, "Linking basics."

What are ASCII files?

ASCII files can't be used by more than one user at a time.

ASCII files are simple, text-based files. ASCII files can be created by any text editor such as the Windows Notepad. Most text editors create a .TXT file for the ASCII file. Unless you specify an extension such as .TXT when you're creating an ASCII file, ObjectVision will create the file without an extension.

ObjectVision links either read data from or write data to an ASCII file, but *not* both. If you want your application to read from and write to an ASCII file, you must build two links—one for the read connection and one for the write connection.

ASCII file structure

ASCII files are structured as lines of text, and each line is a record containing a set of positions. For example, each time a form's values are written to an ASCII file, one record is created. Each position in a record is separated, or delimited, by a comma. When a position value contains a comma, that position must be enclosed in double quotation marks:

Doe,Jane,H.,"Apartment, duplex, or single-family"

When a position value contains a double quotation mark, that position must have two double quotation marks for each occurrence:

Doe,John,R.,""Fixer-upper""

ObjectVision links create connections between the ASCII file's positions and your application's fields according to the positions in the ASCII file. ObjectVision reads or writes your application's field values to the ASCII file as specified by the link.

Unlike database tables, ASCII positions aren't labeled with names. ObjectVision refers to ASCII fields by numbers beginning with one.

ASCII link options

ObjectVision automatically creates an ASCII file if you create a Write or Append link to a file that doesn't exist. See "Creating an ASCII file" later in this chapter.

A Write link *overwrites* the ASCII file with the new values. An Append link inserts the new values at the end of the file.

ASCII files, unlike database files, aren't structured for optimal speed. ASCII files are best used for small sets of values. For example, an ASCII file might be used to get constant values.

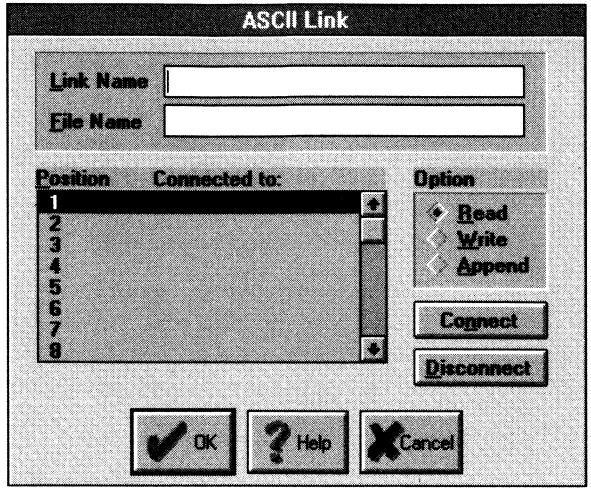
Supporting single users

ObjectVision links to ASCII files don't support multi-user activity. When ObjectVision opens a link to an ASCII file, it assumes no one else is using that file. If another application changes the file while your application is linked to it, errors will result.

Linking to ASCII files

To create a link to an ASCII file, select Tools | Links or click Links on the Object bar in the Form Tool. When the Link dialog box appears, select ASCII then click create. The ASCII dialog box shown in Figure 15.1 appears.

Figure 15.1
ASCII dialog box



To create the link connections follow the steps below. Each step is described in more detail in the next paragraphs.

1. Type a link name in the Link Name box. Press *Tab* or select File Name.
2. Type the name of the ASCII file you want to link to in the File Name box. If you're creating a Write or Append file, you can create a new ASCII file by typing a unique file name. If the file isn't in the current directory, type the full path name.
3. Select the connection option: Read, Write, or Append. These options are described below.
4. Highlight a position of the ASCII field that you want to connect to, then click Connect. You can also double-click the position. Note that 64 positions is the limit.
5. The Field Name dialog box appears. Either select the field you want to connect to the position you chose in step 4 or select <Add New Field>. You can highlight the field name and click OK, or you can double-click the field to connect it.
6. After you connect a field, you return to the ASCII Link dialog box where the connected field appears next to the position. Repeat the above steps for every position you want to connect to.

Link Name

First, type a unique link name, which can use any ANSI characters and be up to 63 characters long.

It's a good idea to choose a name that helps you remember the purpose of the link. For example, if you are creating a link to append records to an existing file, you might want to name the link ADD RECORDS.

When you create a link, avoid using double quotation marks in the link name. When a link name is used in an expression or as an argument, it must be enclosed in double quotation marks. If a link name contains a double quotation mark, the double quotation marks must be typed twice.

File Name

Next, type the name of the ASCII file you want to link to. The file name must be a standard DOS file name—a maximum of eight characters for the name and three characters for the file extension.

If the file you are linking to is not in the same directory as your ObjectVision application, you must include the path name. The full path name can be up to 63 characters long.

Option

Option defines the type of links you want between an ObjectVision application and the ASCII file. An ASCII file can have links for reading, writing, or appending data.

Read

This option creates a read-only link. The ASCII file can't be modified through a Read link. ObjectVision can't create an ASCII file through a read link if one doesn't exist.

When a read connection is created, the values in the first record (the first line of the ASCII file) display in the connected ObjectVision fields.

Unless a full path name is provided, ObjectVision looks for the ASCII file in the current directory. If ObjectVision can't find the ASCII file in the current directory, ObjectVision searches the DOS path. If it still can't find the file, you'll need to create it first.

Write

When an ASCII file is opened for writing, the file is overwritten. Each time the link is opened, the old file is replaced with a new file. If the ASCII file you typed in File Name doesn't exist, ObjectVision creates it automatically.

The new ObjectVision field values are not written to the ASCII file until an insert occurs.

Append

When an ASCII file is opened for appending, ObjectVision field values are written to the end of that file each time you use @INSERT. All existing records in the ASCII file remain unchanged when you append new records.

Position

The Position column in the ASCII dialog box displays numbers corresponding to the order of field positions in the ASCII file. For example, the first field in the ASCII file is labeled 1 in the Position column.

Connect

The limit on positions is 64.

To connect a field in the ASCII file to an ObjectVision field, you first select a number in the Position column, then click Connect. The Field Name dialog box lists all fields in the ObjectVision application. Select the field you want to link to the selected field position in the ASCII file.

You can continue linking fields until you connect all the fields you want. When you choose OK, ObjectVision links the fields to the positions in the ASCII file.

Disconnect

To remove an existing connection to an ASCII position, highlight the connection, then click Disconnect. This button only disconnects one connection at a time.

Automatic buttons

Once you're done connecting positions to ObjectVision fields, click OK. ObjectVision asks if you want to put buttons for the ASCII link on your form. You can select certain buttons depending on the type of connection you created. Next and Top

can be added for Read connections, or Clear and Insert can be added for Write and Append links.

For more information on automatic buttons, see Chapter 10, "Linking basics."

Creating an ASCII file

When you create a link to a file that ObjectVision can't locate, ObjectVision will create the ASCII file automatically.

ObjectVision creates a position for each field that you connect. For instance, if you connect Field1 with position 1 and Field2 with position 2, and then click OK, the ASCII file will contain only two positions.

The ASCII file is created as soon as you click Connect or OK.

C H A P T E R

16

Linking through DDE

This chapter outlines linking between ObjectVision and a DDE-supported program. It also briefly describes DDE and how it works.

DDE basics

Dynamic Data Exchange (DDE) is a Windows protocol for exchanging information between applications. ObjectVision can use DDE to exchange data with any other Windows program that supports DDE, including other ObjectVision applications.

Because DDE links can transfer data instantly from one Windows application to another, they are an effective way for different applications to share constantly changing information. Windows programs can also use DDE to send commands to other programs. For example, an ObjectVision application can use DDE to cause a recalculation in a Windows spreadsheet.

You can use DDE links to break large ObjectVision applications into separate, smaller applications. Using DDE, other applications can request values from your application. When you don't want another application to receive the values, you can set your ObjectVision application to ignore DDE requests by checking Ignore Remote DDE Requests in the Links dialog box.

ObjectVision supports three DDE functions:

- Reading values from other Windows applications
- Writing values to other applications
- Sending commands to other applications

By using Tools | Links, Edit | Paste Link, or DDE @functions you can create DDE links to other applications. To change another application's values or send commands to other applications you must use DDE @functions.

What's different about DDE links

Unlike the other links supported by ObjectVision, a DDE link is an interactive conversation between two programs. If the other application is not loaded in memory when ObjectVision tries to connect a DDE link, ObjectVision starts the other application.

When ObjectVision works with a dBASE file, ObjectVision does all the work. It searches through the data file and finds the records it needs. The dBASE program itself is not involved in the process.

If the application is not already running, ObjectVision starts it, then links to it.

When ObjectVision wants information from a DDE-linked Windows spreadsheet, it gets the spreadsheet application to do some of the work. ObjectVision first links to the running application. ObjectVision then tells the application what values it wants. From that point on, the spreadsheet sends the appropriate values to ObjectVision every time those values change.

ObjectVision monitors the link for changes and requests any updates through **hot links** to the application.

DDE and memory use

DDE offers many useful features that other data links don't provide, but it demands greater performance from your computer. Since DDE requires both programs that are exchanging data to be running at the same time, it uses more memory than other types of data exchange.

DDE background information

A **DDE conversation** is two applications using DDE to exchange information.

Client A client application starts the conversation and usually receives data from the other application.

Server The server application responds to the client.

For example, if you create a DDE link in an ObjectVision application to receive data from a Windows spreadsheet, ObjectVision is called the client and the spreadsheet is called the server.

Conversation terms

In ObjectVision, every DDE conversation has an application, a topic, one or more items, and a link name.

- The *application* is usually the name of the server's .EXE file, but don't include the extension with the name. For instance, the application name for ObjectVision is VISION.
- The *topic* is the name of the server's document or file containing the requested information. When using DDE to link to an ObjectVision application, the topic is the name of the .OVD file. You must specify the extension when you name the topic.
- The *item* is the place in the topic where the information is stored. In ObjectVision the item is the name of a field. The *value* in the item is exchanged between the client and server.
- The *link name* is the name you give to the link. It is referenced in @functions and the Links Tool.

DDE link types

DDE protocol defines two types of links: hot links and warm links.

In a *hot link*, the server immediately sends the changed data to the client.

In a *warm link* the server notifies the client that the data has changed, but does not send the information until the client requests it.

ObjectVision always creates a hot link when it is the client. When it is the server, it accepts either a hot or warm link from the requesting client.

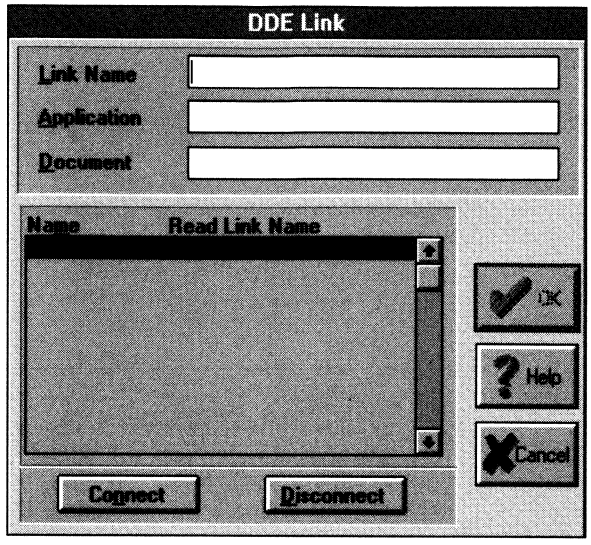
Using the Links Tool

The Links Tool is one way to create DDE links. You can also paste links from other ObjectVision applications or use a DDE @function to create DDE links, as discussed later in this chapter.

The following steps describe how to create a DDE link. The terms and instructions used in the steps are explained in more detail in the following sections.

1. Choose Tools | Links from the menu, or click Links on the Object Bar.
2. In the Links Creation box, select DDE from the selection list then click Create. If you don't want your application to act as a server, be sure to check Ignore Remote Links.
3. Type a Link Name then press *Tab*.
4. Type Application name (usually the .EXE file, but you don't need to type .EXE). Press *Tab*. If this file isn't in the DOS path, type the entire path name.
5. Type the Document name with its extension. Click Connect.
6. A dialog box requesting Remote Name appears. Type in the item name, then click OK.
7. Another dialog box appears, this time requesting an ObjectVision Field Name. You can select one from the list, or you can add a new field to your application. Click OK. This connects the ObjectVision field to the Remote Name item. You can highlight another line in the connections box then repeat steps 6 and 7 until you connect all the items to fields that you want.
8. The item names you connected should now appear in the DDE Links dialog box. Click OK to connect your links.
9. A dialog box may appear if the DDE application and topic aren't currently running. Press *Enter* to start the application.
10. Once the DDE application is running the DDE linked fields in your ObjectVision application should contain values.

Figure 16.1
DDE dialog box



Link Name

Type a unique link name in the Link Name box. The Link Name is the name that you will use in ObjectVision to refer to the link you are creating. The name can contain any ANSI character and be up to 63 characters long.

The name you choose should help you remember the purpose of the link. Try to avoid double-quotation marks in link names; it will make writing expressions easier (see Chapter 6, "Writing expressions").

Application

In the Application box, type the name of the Windows application (usually the .EXE file), but don't type the extension. For example, to create a DDE link to another ObjectVision application, type the application name `VISION`.

Remember that the application you are linking to must be specified in your DOS path. If it isn't listed in the path, you can enter the complete path name as the application name.

Document

Enter the name and extension of the topic in the Document box. To link to another ObjectVision application, type the .OVD file name. If the file you are linking to is not where the DDE application can find it, you must include the complete path name.

If ObjectVision can't find the application or the application can't find its document, ObjectVision can't create the link. If this occurs, check to see that you have entered the application name and document name correctly.

Name

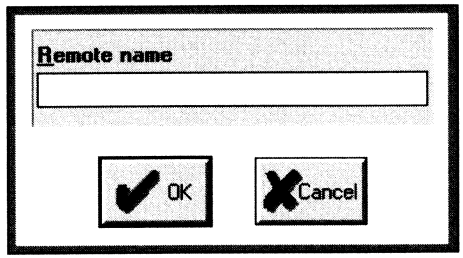
Use the Name box to connect fields in your ObjectVision application to items in other applications.

To begin, select the first row in the Name box, then click Connect.

Connect

After you click Connect, a dialog box prompts you to enter a Remote Name. This is an item in the server's topic.

Figure 16.2
Remote Name box for the server's items



The Remote Name varies depending on the DDE application. For example, in ObjectVision, a Remote Name is a field name. Other DDE servers will have different Remote Names. Check the server's documentation for the naming conventions of its items.

After you click OK, the Field Name dialog box appears listing the fields in your ObjectVision application. Select the field to receive the value.

You can also create a new ObjectVision field from the Read Field dialog box by selecting <Add New Field> and typing a name in the New Field Name dialog box.

Click OK when you have either selected an ObjectVision field from the Read Field dialog box or created a new field.

When you return to the DDE Link dialog box, you'll see the name of the ObjectVision field on the right under Read Link Name and the item it is connected to on the left under Name. See Figure 16.1.

Disconnect

To disconnect a field from a DDE link, highlight the field name and item in the DDE Link dialog box then click Disconnect.

Disconnect only disconnects the highlighted field. All other fields in the link remain connected.

Using Paste link

The fastest way to create a DDE link is by pasting one. You must be running both the client and the server to do this.

1. Highlight the item in the server then copy it to the Clipboard. Most servers use Edit | Copy to place a copy of the object on the Clipboard.
2. Return to ObjectVision (the client in this case).
3. Select the ObjectVision field that will receive its value from the server.
4. While in the Form Tool, choose Edit | Paste Link from the menu.

If you want to check that the field has been linked, select Tools | Links from the menu or click Links on the Object bar to verify that the Remote Name appears.

When you create DDE links with Paste Link, ObjectVision uses the item name for the link name. For example, if you're linking a cell (r1c1) in a Windows spreadsheet to a field (Field1) in ObjectVision, the link name will be the same as the server's item name (in this case r1c1).

If you're creating and linking two ObjectVision applications with Paste Link, the server application must be saved before it can be linked. If you try to link an unsaved (Untitled) server document, ObjectVision won't have a document name to use in the link, and therefore can't create the link.

DDE @functions

There are three DDE specific @functions: @DDEEXECUTE, @DDEOPEN, @DDEPOKE. @CLOSE is a general link function that can be used to close DDE links. This section discusses their use and purpose. See also Chapter 8, "@Function descriptions".

Linking with DDE @functions

@DDEOPEN works like the Links tool but is more flexible. You can use @DDEOPEN in combination with @CLOSE to connect and disconnect a link. You can do this several ways. Here are some suggestions:

- Use the @functions with buttons. Click one button to link, the other to close the link.
- Create an event tree on a field. When it's selected, the link is connected. Create a second field with an event tree to close the link.
- Write an event tree on a special field. When it's selected, automatically close the link. After certain fields are complete, reconnect the link (use @ISCOMPLETE).

For example, you could link one field (Contract Number) in an ObjectVision application to another field (Cont#) in another application. To do this you could use one of the above suggestions with @DDEOPEN. The event tree conclusion for this example is

```
@DDEOPEN("ContractLink", "VISION", "BUILDER.OVD",
                "Cont#", "Contract Number")
```

The value in Cont# is read to the field Contract Number.

This example assumes that you'll call the link ContractLink. It also assumes that BUILDER.OVD is the name of the application you're linking to and BUILDER.OVD is in the current directory.

Sending values with @DDEPOKE

You can use @DDEPOKE to change the value of an item in a server. The DDE link must be connected before you can use @DDEPOKE.

For example, you can use @DDEPOKE to change the value in another application every time the value of a field exceeds the value of the field in a remote application.

Follow the example for @DDEOPEN. The event tree conclusion looks like

@DDEPOKE("ContractLink", "Cont#", "The New Value for Cont#")

The Cont# field receives the string value "The New Value for Cont#". Instead of sending a string value, you could send the value from a field in your application. You could reverse the @DDEOPEN example. Instead of Contract Number receiving its value from Cont#, you could use @DDEPOKE to send Cont# the value in Contract Number:

@DDEPOKE("ContractLink", "Cont#", Contract Number)

Sending commands with @DDEEXECUTE

Using @DDEEXECUTE you can send commands to another application that is linked through DDE. Like @DDEPOKE, @DDEEXECUTE requires a connected DDE link.

@DDEEXECUTE sends a character string to the DDE server application. The server interprets the string as a series of commands, then performs the commands.

You can send commands to any DDE application. The example below shows how you could send commands from one ObjectVision application to another.

Remember to use two double quotes to represent a single double quote in the command that is a constant string. For more information on writing expressions, see Chapter 6, "Writing expressions."

You can send any single-lined or multilined command from one ObjectVision application to another. The only difference between DDE commands and event tree conclusion expressions is that each line of a DDE command must be enclosed by square brackets.

For example, if you have a DDE link called DemoLink to another ObjectVision application, you could send the following commands:

- @DDEEXECUTE("DemoLink", "[@CLEARALL]")
- @DDEEXECUTE("DemoLink", "[@FORMCLEAR(""FormA"")] [@FORMCLEAR(""FormB"")]")

For information on syntax or instructions for sending commands to other DDE programs, consult the manuals of the other programs.

An example of a DDE link

The included applications DDE1.OVD and DDE2.OVD provide a simple example of the basic DDE functions.

DDE1 contains four buttons at the bottom of the form. The event trees for these buttons are listed and explained below.

The button event trees

Open DDE Link

@DDEOPEN("DDE1toDDE2", "VISION", "DDE2.OVD",
"LastName, MailingName, Address1, Address2, City, State, Zip, Phone", "LastName, MailingName, Address1, Address2, City, State, Zip, Phone")

This expression creates a DDE link named DDE1toDDE2. Notice that the link name indicates which applications are being linked. DDE1toDDE2 links the fields in DDE2 to fields with the same name in DDE1.

Close DDE Link

@CLOSE("DDE1toDDE2")

This removes the DDE link.

Change Server Values

@DDEPOKE("DDE1toDDE2","LastName",LastName)
@DDEPOKE("DDE1toDDE2","MailingName",MailingName)
@DDEPOKE("DDE1toDDE2","Address1",Address1)
@DDEPOKE("DDE1toDDE2","Address2",Address2)
@DDEPOKE("DDE1toDDE2","City",City)
@DDEPOKE("DDE1toDDE2","State",State)
@DDEPOKE("DDE1toDDE2","Zip",Zip)
@DDEPOKE("DDE1toDDE2","Phone",Phone)

The Change Server Values button uses @DDEPOKE to update the values in the server application, DDE2, with values from DDE1. Notice that there is a separate @DDEPOKE command for every field.

Clear All in Server

@DDEEXECUTE("DDE1toDDE2","[@FIELDCLEAR(""LastName"")]")
@DDEEXECUTE("DDE1toDDE2","[@FIELDCLEAR(""MailingName"")]")
@DDEEXECUTE("DDE1toDDE2","[@FIELDCLEAR(""Address1"")]")
@DDEEXECUTE("DDE1toDDE2","[@FIELDCLEAR(""Address2"")]")
@DDEEXECUTE("DDE1toDDE2","[@FIELDCLEAR(""City"")]")
@DDEEXECUTE("DDE1toDDE2","[@FIELDCLEAR(""State"")]")
@DDEEXECUTE("DDE1toDDE2","[@FIELDCLEAR(""Zip"")]")
@DDEEXECUTE("DDE1toDDE2","[@FIELDCLEAR(""Phone"")]")

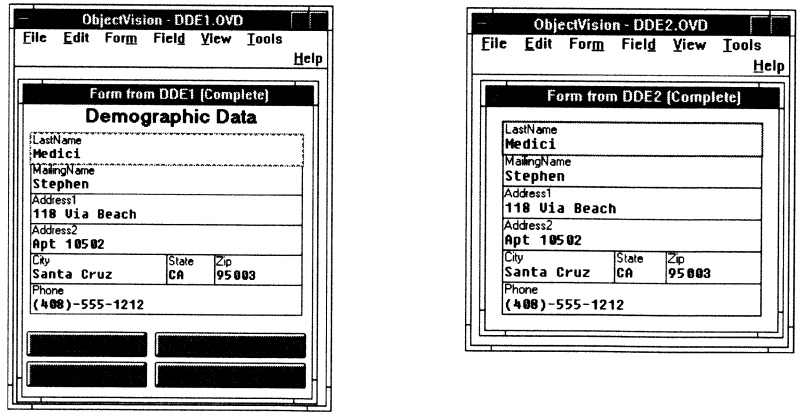
The Clear All in Server button uses @DDEEXECUTE to send the @FIELDCLEAR commands to the DDE2 application causing it to remove all the field values.

Using the sample application

To see how these applications work together, open DDE1.OVD then click the Open DDE Link button. A dialog box appears asking you if you want to start VISION.EXE. Choose Yes. An icon for DDE2.OVD appears on the screen and the values from DDE2 appear in DDE1.

Double-click the icon for DDE2 and resize the windows for both applications so that you can see them both on the screen at the same time.

Figure 16.3
Linking DDE1 to DDE2



When you type a value in any field in DDE2 and press *Tab* or *Enter*, the value instantly appears in the corresponding field in DDE1.

If you type a new value in DDE1 then click the Change Server Values button, the new value appears in DDE2. The Clear All in Server button removes all values from DDE2. Close Link disconnects the link.

C H A P T E R

17

Tips for designing database tables

This chapter will help you design a database table. It includes definitions and examples to help you understand database tables.

What is a database table?

A table is an organized set of information. Typically, you view the values in horizontal *rows* and vertical *columns*.

Columns are given labels, generically called *field names*, that explain what type of information is stored beneath them. In Figure 17.1, the column labels are Last Name, First Name, Address, City, State, and Zip Code.

Figure 17.1
A table

Contacts					
Last Name	First Name	Address	City	State	ZIP Code
Mioreille	Aspi	1078 Center Street	Santa Cruz	CA	95066
Henderson	Michael	123 Main street	San Francisco	CA	95108

Each column is given a *field type* that tells the database program how to save and display the information. You should try to use the field type that best fits your information because you don't want to give each field more room than it needs.

What is an index?

An index is a table reference that allows a database program and ObjectVision to locate records more quickly.

You can think of an unindexed table as a telephone directory in which the names appear in random order. To find a number you'd have to start at page one and read all the numbers until you found the one you want.

Searching this way, called a *sequential search*, takes a long time. With an index, however, you don't have to look at each record.

An index allows you to search on key fields, just as you'd search a phone book by last name, then first name, and possibly, by address.

ObjectVision uses indexes when it tries to locate a record. You cannot use a Locate if your table is not indexed. You can index more than one field in a table (just as in a phone book, you could index last name, then first name, then city).

Indexed fields must be in top-down order for Paradox. This means that you couldn't index Last name and City without indexing First name and Address as well. dBASE and Btrieve are exceptions, because they let you index nonsequential fields such as indexing Last name, not indexing the next two fields (First name and Address), then indexing City.

A secondary index

Paradox allows you to use two indexes. If you create a secondary index in Paradox it can be either manually or automatically updated.

You'll want a secondary index if you plan to sort your tables in more than one way. For example, if you wanted to see who lives in a particular city by using the phone book, neither the alphabetical listing nor the random order would be much help. But if you created a secondary index on the field City, you could easily locate records by city.

What is normalization?

Normalization is a way of arranging information in tables so that the information can be easily related (the next section discusses how to relate tables). If you have a lot of columns of information

to store, you may want to save it in different tables and relate the tables.

A table with lots of columns can get difficult to manage. For example, in Figure 17.2 there are 10 columns, each describing one employee.

Figure 17.2
A very large table

STAFF	Last name	First name	Position	Location	Phone	State1	State2	State3	Salary
1	Milford	John	Rep	Boston	617 555-3733	ME	NH	UT	36,000.00
2	Janacek	Sue	Sr Rep	Chicago	312 555-6700	IL			42,000.00
3	Rorem	Blake	Mgr	Atlanta	404 555-4321				48,000.00
4	Bacon	Mary	Rep	Atlanta	404 555-4321	MS	LA	AR	33,000.00

This may appear to be OK now, but here are some reasons why you shouldn't create a huge table like this:

- Suppose the eastern region relocates. You'll have to change the phone number for every employee.
- Suppose you want to know who's responsible for customers in a state. You'll have to search three fields (State1, State2, and State3).
- Suppose some of the staff resign. How will you know what areas they were responsible for?

Relating tables

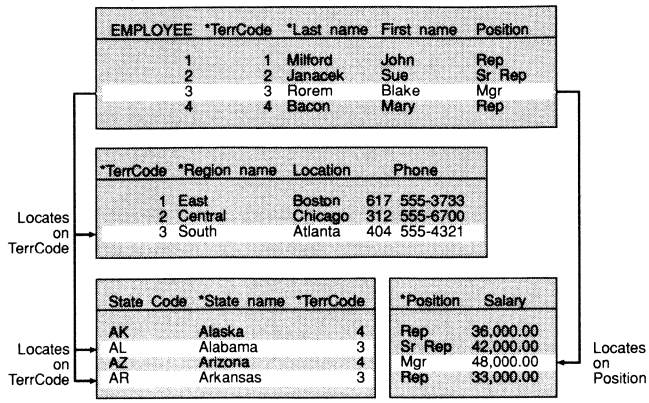
A better way to organize information is to create several smaller tables and relate them. Related tables need to have at least one field in common. The name of the field doesn't need to be the same—just the values.

Using the information from the above table as an example, you could create four smaller tables. You may ask *why create four tables when you can create one?* One answer is that you may not want everyone to use all the information. For example, you might want data entry people to be able to change employee addresses, but not to see each employee's salary.

For more information on auto locates and using @LOCATE see Chapter 11, "Linking options."

Figure 17.2 shows all four tables. You could relate them using ObjectVision. You would use one field, Emp Code, to locate Region and Phone Number in another table.

Figure 17.3
Relating tables



Before you create a table

Planning is always the first step in creating a table. You need to decide what information the table should contain and how it should be laid out.

This is an important step when creating a database table with ObjectVision because ObjectVision can't restructure a table. If you create a table with ObjectVision then decide later you need to add or delete fields, you'll need to use the database program itself to do this.

In ObjectVision, you can plan a table, rearrange it, and add or delete fields, but once you click OK in the Database Table Creation dialog box the table can't be changed with ObjectVision.

Tips on creating tables

Before you create a table, you should consider these suggestions:

- Be clear about what type of table you want to create. For example, do you want a Paradox, Btrieve, dBASE, or ASCII table? If you have a database program, you'll most likely use that type of table, but if you don't own a database program consider how you'll manage the table before choosing a type.
- Be thorough. Try to only include fields you'll use. Don't clutter your table.
- Try not to needlessly place everything in one table. It's usually better to create several tables, each with a few fields of related information, rather than a single large table. Since ObjectVision

can locate on other tables, small tables are easier to work with than tables with many fields.

- Keep tables familiar. It's often best to create tables that correspond to the kinds of information you already use. They can reflect files or forms you currently keep on paper.
- Build ways of joining tables such as keeping one common (indexed) field (remember only the values need to match, not the field names).
- Avoid redundancy. Beyond common fields for locating, don't duplicate information.
- Remember limitations. Every database type has different limitations for field types and memory. The limits for each database are discussed in that database's respective chapter.

Creating a table with ObjectVision

You can create a database table by either creating the ObjectVision fields and columns first, or you can create the fields for the database table while you create the table.

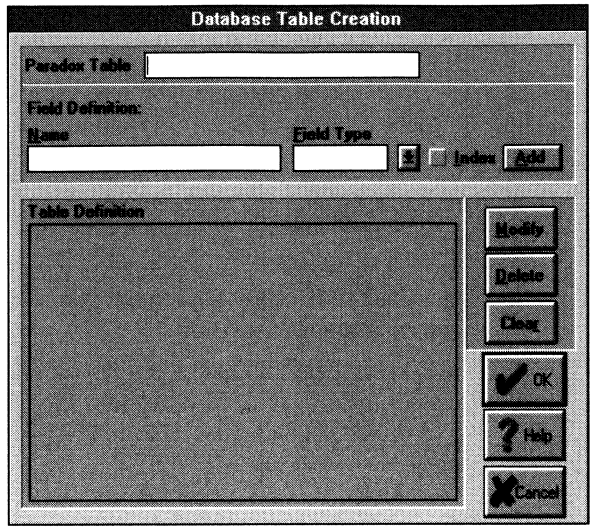
Overview

1. Before you can create a table, open the Links Tool. Select the type of database table you want to create (under the link type), then click Create.
2. When the Link Connections dialog box appears, click Create table.
3. A dialog box appears where you add database table fields, modify them, or delete them. If you've created ObjectVision fields or columns, they will be displayed where you can delete them from the database table, or modify them to fit the table. (See the following section for more details.)

Database table creation

Figure 17.4 shows the Database Table Creation dialog box. This is the dialog box you use to create a database table. The box shown in Figure 17.4 is the dialog box for Paradox.

Figure 17.4
Creating a Paradox table



If you've created ObjectVision fields, they will be listed under Table definition. You can then delete from this list the fields you don't want in your table, or you can modify existing fields.

If you haven't created any ObjectVision fields, the area under Table definition will be blank.

1. In the first box titled Paradox table, type the name you want to give your table. The name must follow the eight-character DOS standard. You don't need to specify the extension—ObjectVision adds the default extension for each table type.
2. Type the name of the database table field under Name. This name can correspond to an ObjectVision field.
3. In the field type box, type the field type you want for that field. You can click the arrow next to the box to get a generic list of the available field types for the table type you've chosen. If you aren't familiar with the field types of the table you're creating, you might want to read the chapter that discusses that type of database table before you create the table.
4. If you want the field to be an indexed field, check Index. If not, leave Index blank.
5. Once you've typed all the above information, click Add to add the field to the database table. Repeat the above sequence for as many fields as you need for your table. Once you've added all the fields, click OK to create the table.

Editing the table

You can only modify, add, or delete any fields *before* you create the database table. All the edit buttons are described below.

Modifying a field

You can modify any field listed in the Table definition area.

1. Highlight the field in the Table definition list.
2. The field information will appear in the field definition area. You can edit any of the displayed values. You can change the field name, type or size, or you can check or uncheck Index.
3. When you've edited the field definition area, click Modify. The changes immediately appear in the Table definition list.

Deleting a field

You can delete any field listed under Table definition. Highlight the field in the list, then click Delete. The field will disappear from the Table definition list.

If you created ObjectVision fields before creating the table, but you don't want them as fields in your database table, delete them from the Table definition list. They won't be deleted from the ObjectVision form you've already created.

Using Clear

The Clear button clears the entire dialog box. Any fields listed under Table definition or in Field definition are erased. The table name is also cleared.

Using OK and Cancel

When you're done modifying the database table fields and want to create the actual table, click OK. Once you click OK you can't change the database table fields.

Cancel closes the Database Table Creation dialog box without creating the database table. Any editing of the table is lost.

PART 4

Creating @functions

C H A P T E R

18

Using @REGISTER with DLLs

This chapter explains how to use @REGISTER to register a dynamic-link library (DLL). It also explains a DLL, when and why ObjectVision uses a DLL, then gives an example of a DLL.

DLL Overview

A DLL is a program that allows Windows programs to share code that performs common tasks.

ObjectVision uses DLLs to create new @functions. Users with sufficient programming experience can write their own DLLs; less experienced users can use DLLs written by someone else.

Using @REGISTER to link DLL functions

The @REGISTER function links DLL routines to a ObjectVision application so that the routines appear as @functions. Functions from DLLs must be registered each time the ObjectVision application is opened.

Registered functions don't become a permanent part of ObjectVision; that's why they are called dynamic-link libraries. Once you close an application that registered a DLL function, the link is gone and the function cannot be used again unless it is reregistered.

Where to use @REGISTER

You can use @REGISTER to link DLL functions in a variety of ways. The most common way is to include @REGISTER in the open event tree conclusion for the application stack. This way, the DLL functions are registered every time you open that application.

You can also use @REGISTER with a button, so that when the button is clicked, the function is registered.

Or, you can register the function after a form or field is completed. You can register the function however you want to, but you must always register the function before you use it.

The @REGISTER arguments

The arguments for @REGISTER tell ObjectVision what name to give the new @function, what arguments it accepts, and where to find the DLL.

The format for @REGISTER is:

@REGISTER("OVAlias", "ArgTypes", "ArgHelp", "LibName", "FuncName", Type)

OVAlias

OVAlias is the @function name that will be assigned to the new DLL function.

The OVAlias must follow these rules:

- it must begin with the @ character
- it must contain only letters, numbers, periods, and underlines
- it must be less than or equal to 254 characters including the @ character

After the function is registered, this name will appear in the Paste Function list in ObjectVision.

ArgTypes

ArgTypes are the codes for the argument data types required by the new function. The first character is the value returned by the function itself. All other characters are the arguments that will be passed to the registered function.

ArgTypes are case-sensitive and must be from the table below.

Table 18.1
Argument types

Code	Type	Passed	Returned
A	Boolean	int	int
B	IEEE float	double	double far*
C	C string	char far*	char far*
F	C string buffer	char far*	char far*
H	Unsigned integer	unsigned int	unsigned int
I	Signed integer	int	int
J	Unsigned long	unsigned long	unsigned long
o	GetCurrentRow	FARPROC	
p	NextCol	FARPROC	
q	GetLastRow	FARPROC	
r	IsColumn	FARPROC	
s	SetCurrentRow	FARPROC	
t	SetLinkRow	FARPROC	
u	ResetCalculated	FARPROC	
v	ResetOverride	FARPROC	
w	Put	FARPROC	
x	Get	FARPROC	
y	hMainWindow	HANDLE	
z	hInst	HANDLE	

Argument type F represents a 4,096 byte buffer allocated by ObjectVision as a work space for the DLL. ObjectVision sets the memory aside for the DLL, then gives the DLL a pointer to the buffer. The DLL puts a value in the buffer then tells ObjectVision that the value is there. ObjectVision then retrieves the value and deallocates the memory.

The arguments o through x are pointers to ObjectVision functions that a DLL function can call back. They do not require a user-supplied value.

ArgHelp

ArgHelp is the text that appears when a user checks Paste Arguments while pasting the function in an expression.

If the function doesn't use a user-entered value in its arguments, enter an empty set of double quotes.

LibName

LibName is the name of the DLL containing the function you are registering. If the DLL is not in the current directory or in the Windows program directory, include the complete path name.

FuncName

FuncName is the name of the function in the DLL. Don't confuse this name with the OVAlias, which refers to the function name in ObjectVision, although you might want to use the same name for both OVAlias and FuncName, leaving off the @ in FuncName, of course.

Type

Type describes whether the function is a value or event function. For value functions use 0. For event functions use 1.

@REGISTER Example

To see how this function works, open the CHOICE.OVD sample application.

The example below shows how you'd register a function.

@REGISTER("@CHOICE", "FACCF", "Condition, TrueString, FalseString", "CHOICE.DLL", "ChooseString", 0)

Once the function is registered, you could use this function in an expression as follows:

@CHOICE(Cost>4576, "We cannot purchase at this time", "We can purchase now.")

A sample DLL written in C

This section contains some of the C source code for the DLL CHOICE.DLL, which is included with ObjectVision. If you plan to only use DLLs written by someone else, you may want to skip the rest of this chapter.

CHOICE.DLL contains the function ChooseString.

Before looking at the code, note these two points:

- Every DLL has an entry point LibMain and an exit point WEP.
- To use any of the callbacks (data types o through x) you must include the file CHOICE.H.

```
#include <windows.h>
#include "choice.h"

// Turn off "Parameter never used" warning
#pragma argsused

int FAR PASCAL LibMain(HANDLE hInstance, WORD wDataSegment,
                       WORD wHeapSize, LPSTR lpszCmdLine)
```

```
{
  // The startup code for the DLL initializes the local heap (if
  // there is one) with a call to LocalInit which locks the data
  // segment.
  if (wHeapSize != 0)
      UnlockData(0);
  return 1;  // Indicate the DLL was initialized successfully.
}

// Turn off "Parameter never used" warning
#pragma argsused

int FAR PASCAL WEP (int bSystemExit)
{
  return 1;
}

// Argument template = "FACCF"
LPSTR FAR PASCAL _export ChooseString(BOOL bChoice,
                                      LPSTR szTrueString,
                                      LPSTR szFalseString,
                                      LPSTR sOvBuffer)
{
  if (bChoice)
  {
     // ObjectVision 2.0 strings can be up to 4096 characters long
     if (lstrlen(szTrueString) <= 4096)
     {
        lstrcpy(sOvBuffer, szTrueString);
     }
     else
     {
        *sOvBuffer = '\0';
     }
  }
  else
  {
     if (lstrlen(szFalseString) <= 4096)
        {
     lstrcpy(sOvBuffer, szFalseString);
     }
   else
     {
     *sOvBuffer = '\0';
     }
  }
  return sOvBuffer;
}
```

Compiling CHOICE.C

To compile CHOICE.C with Borland C++, use the commands and arguments that follow. The arguments for other C compilers should be very similar.

```
bcc -v -c -ms! -WDE -IPATHNAME  sampl_c.c
tlink -Twd -v -c -Lc:\bc\lib c0ds sampl_c, sampl_c,, cwins cs
import, sampl_c
rc sampl_c.dll
```

The associated .DEF file is listed below:

```
LIBRARY        CHOICE
DESCRIPTION    'Sample ObjectVision DLL'
EXETYPE        WINDOWS
CODE           PRELOAD MOVEABLE DISCARDABLE
DATA           PRELOAD MOVEABLE SINGLE
HEAPSIZE       1024
```

Sample PASCAL DLL source code

This DLL, written in Turbo Pascal for Windows, contains a simple function—Random Number. You can find this source code in the file RAND.PAS. The compiled DLL is named RAND.DLL. To use the registered function, open the RAND.OVD sample application.

```
{$N+}

library Rand;

var D : Double;

function RandomNumber: pointer; export;
begin
  D:=Random
  RandomNumber := @D;
end;

exports
  RandomNumber;

begin
  Randomize;
end.
```

A note to DLL programmers

If you'll be distributing your DLLs, you might want to follow these tips:

- Write a text file for users that gives the @REGISTER syntax. Try not to assume that your users will automatically know how to do this.
- Suggest that users copy and paste the @REGISTER syntax you've written. This way they shouldn't have problems remembering the syntax and argument types.
- You might want to create a sample application that contains only your @REGISTER functions. You can place your functions in a conclusion of an event tree on the stack so that users of your DLL can either copy what you've written, or begin designing their applications from the one you started.

PART 5

Appendixes and Glossary

A P P E N D I X

A

Summary of 2.0 changes

This appendix summarizes the features that are new in version 2.0. If you are a 1.0 user, this section should help you get a quick overview idea of what changes were made to ObjectVision. If you are not a 1.0 user, you might want to skip this chapter.

User interface changes

The entire user interface has been improved for easier use. You will immediately notice the changes.

Object bar

The form and stack tools now include an object bar for inserting new objects and performing other frequent operations. The buttons on this bar are a quicker way of performing a command.

To insert a new object, click the object's button, then position the new object.

The Object bar is described in more detail beginning on page 30.

Properties

In addition to the Properties menu, you can quickly change an object's properties by clicking the object with the right mouse button. A list of that object's properties appears. This action is referred to as Inspecting the objects properties.

You can right-click on a form title bar or any blank area in the form window to change its properties. Forms only have two properties: event trees and titles.

To change the stack's only property (an event tree), right-click on the application title bar or any blank part of the ObjectVision window outside of any forms.

You can still select Properties from the menu bar, but you will need to choose Object, Form, or Stack, then the particular property.

Trees have been added to the Properties list. This makes it easy for you to create a tree based on an object, a form, or a stack. Once a tree has been created, a check mark appears next to the tree in the object's Properties list.

Guided completion

Guided completion will select buttons automatically. In version 1.0, buttons were excluded from guided completion.

Tree changes

There are now two types of trees: value trees and event trees. Value trees return a value to the field based on conditions. Event trees perform an action after an event occurs.

In order to create or edit a tree you must be using the Form Tool. From there you inspect the properties by clicking the right mouse button or using the menu.

ObjectVision 2.0 has a new type of branch—the unrestricted branch. You can use this branch when you want to write a condition that doesn't depend on a single field's value. Hence, the branch is unrestricted by a field. See page 59 for a more detailed explanation of unrestricted branches.

Conversion from 1.0

If you created trees on buttons using ObjectVision 1.0, they will be converted to event trees with the event click. All other ObjectVision 1.0 trees are converted to value trees.

Editing trees

You can copy trees between applications. To copy a tree, select the tree, use Edit | Copy to copy the tree or tree section to the Clipboard, then open another ObjectVision application. Select the object to receive the copied tree, select the tree type from its property list, then use Edit | Paste.

When you paste an event tree, you'll have to specify a new event for the object receiving the tree.

Events

A typical event for a button is *click*. For a form or a field, an event might be *select*. For instance, you could program an event on a button so that when a user clicks a button a message appears, or you may want to have another form appear.

You can also use 26 key combinations from CTRL+A through CTRL+Z. These events are to the stack only. For a table of the available events see page 57.

If you customize your own menus, each new menu item becomes an event on the stack.

You can define your own events by using @EVENT. User-defined events are discussed beginning on page 58.

All of the tree changes are discussed in Chapters 4, "Tree basics" and 5, "Creating trees."

Improved form design

You can now add color to your objects and customize your menus. With the addition of Grid and Rulers, you have more control over object placement.

Rulers and grids

While using the Form Tool, you can choose View | Grid to select which underlying grid to use—Coarse, Medium, or Fine. Coarse uses a one-character by one-line grid pattern, Medium a one-half-character by one-half-line grid, and Fine a one-fourth-character by one-fourth-line grid. When you create or place an object on a form, it is constrained to the grid.

The Coarse grid was used in ObjectVision 1.0.

You can attach rulers to a form to help guide you during design. You can select a ruler for the top or the left of a form or both. Rulers can display in inches, centimeters, or characters. For characters, the ruler displays one average character width along the top and one average character height along the left.

Color

You can now add color to your objects. Once you have created an object and placed it on a form, you can select Color from the object's list of properties.

Color is separated into four types—Label, Value, Background, and Border. Some objects, like fields, accept all four color types. Other objects, like lines, only accept one color type. For more information on Color see page 36.

Fonts

You can change both the label and the value font. These are two field properties. The default label font is Helvetica 8. The default value font is also Courier 12.

New field types

Three field types have been added:

- Alphanumeric, which is similar to General.
- ComboBox, which is similar to Selection list except you can supply a value not in the list.
- Radio buttons, which are like Check boxes except they're round.

Menu customization

You can customize the menus on your applications. This gives your applications a personal touch. You now have control over which choices your users can make. Menu customization is discussed in Chapter 9, "Menu customization."

Object defaults

You can select defaults for every object except Button. Once a default is set for an object, every object of that type that you place on a form from then on will have those defaults.

For example, you could change the default settings for a field to

- Percent (Field type)
- Centered (Alignment)
- Helvetica 14 (Label Font)
- Blue (Label Color)

The defaults are saved, so if you exit ObjectVision and restart, you will still have your object defaults. The defaults are saved in a file called VISION.INI. You can look at this file by using any text editor such as the Windows Notepad, although you should only change defaults from ObjectVision.

Setting defaults

There are two ways to set defaults. The easiest is to right-click on the Object bar button. A list of that object's current defaults appears. To set a default, select it from the list.

The second way is to use the menus. Select Objects | Defaults then select the object from a list. If you want to reset the defaults for *all* objects, select Objects | Defaults | Reset.

Additional @functions

Ninety-three @functions have been added to ObjectVision. They are divided into two categories—value functions and event functions—that correspond with the division of trees. Event functions can only be used in event trees. Value functions, however, can be used in both value and event trees.

All functions are described on page 89, but here's a short list describing the basic types of @functions that were added:

- 7 menu-customization event functions
- 12 menu equivalent functions
- 18 financial value functions

Chapter 8, @Function descriptions, contains a more detailed explanation of all @functions. Every new @function is tagged with the 2.0 icon for easy identification.

Password protection

Now you can protect your applications with a password. When you save your application, simply type a password in the password box. When someone wants to *use* your application they don't need the password. But if they want to change the design or copy parts of it, they will need the password.

Basically, without the password you can't use any of the tools.

Table form objects

Tables are a new form object. Tables consist of columns and rows that you can add or delete graphically. You can't assign properties to whole rows, but you can assign properties to columns and column cells just as you can to fields. You can also assign properties to individual cells in a column. Fields and columns, however, share the same properties.

If there is a feature or a function (such as @COLUMNSUM) that can only be used in columns, this manual will use the term **column**. Otherwise, the term **field** will be used. For more on tables, see page 41.

To select a table, click on the table title. Now you can change the number of columns or rows in the table by dragging one of the table handles. You can also move the entire table by clicking and dragging it.

To select a column, click in the column title box. To select a cell click anywhere inside the cell. Once you've selected the column or the cell, you can adjust its height and width.

Printing forms with tables

If a form contains a table connected to a database table and you want to print the form, it will be printed multiple times until every value of the database table has been printed. The number of

records printed can be limited by using a filter or a restricted range.

Additions to linking

When designing database tables with ObjectVision, you can define fields, then edit them before you actually create the table.

You can also create one index for each table you create. You can have more than one field indexed. In fact, you could index all fields in a table, although this is strongly discouraged.

There are three linking options: locates, filters, and VFields. They are thoroughly described in Chapter 10, "Linking basics," and 11, "Linking options," but the following paragraphs should give you a general idea of how they work.

Locates

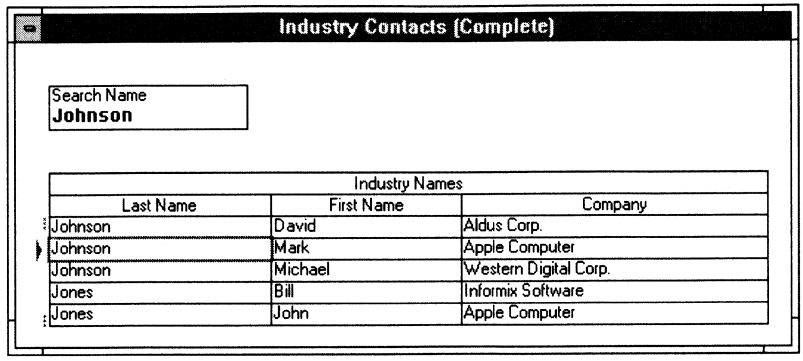
Locates monitor an ObjectVision field that is connected to an indexed database table field. When the ObjectVision field's value changes, the locate immediately attempts to position on that record in the database table (if Auto Locate is checked). The link then returns values.

In version 2.0, you can turn off Auto Locate. You can also use @LOCATE to perform a locate based on an event (such as a button click).

To use locates with your database tables, your tables must be indexed and you can only locate on an indexed field.

For example, in the following figure Industry Names is linked to a database table of customers. Search Name is monitored by a locate. It is linked to a field on the same database table. When the value of Search Name changes, the link that monitors that field through a locate attempts to position on that record in the database table. The link to the ObjectVision table then retrieves values beginning with the new position on the database table.

Figure A.1
Using Locates with tables



Restricted range

You can modify the locate to only accept a restricted range of values. In the above example you could restrict it to just the Johnson's. Then you wouldn't see Jones.

Inexact

Inexact is a locate option that will search for an inexact match. This option can't be used at the same time that a restricted range is used.

For example, in the above table, you could type J in Search Name and get the first record that starts with J.

Filters

Filters allow you to type an expression which is evaluated before a database record is passed through the link. If the value matches the filter expression, that value is read through the link. If the value doesn't match the expression, the link positions to the next database record and evaluates it.

Filters can work with unindexed database tables. This is much slower than a locate since each record must be evaluated with the expression before it can be passed to ObjectVision.

You should use locates whenever possible—especially with indexed tables—but there may be times when you'll need a filter.

Any value that doesn't match the filter expression is ignored by ObjectVision.

VFields

A VField is a field calculated from fields in the database table. Technically, the field itself doesn't exist—only its expression exists, so the values in a VField aren't saved.

A VField is calculated each time the application is opened and each time a value that affects the VField changes.

Linking options filters and VFields use expressions. Chapter 6, "Writing expressions," explains how to write expressions.

New DDE functions

You can create links from one application to another by copying the information in an application (for example, cells in a spreadsheet) to the Clipboard. Once the information is on the Clipboard, you can choose Edit | Paste Link to automatically create a DDE link.

Some Windows applications support a feature called Object Linking and Embedding (OLE). If the application you link to supports OLE (it copies both an OLE and a DDE object to the Clipboard), ObjectVision will use an OLE link instead of a DDE link.

There are two new DDE functions:

@DDEEXECUTE
@DDEPOKE

You can send values to another application using @DDEPOKE. You can send commands to an application by using @DDEEXECUTE.

Printing with links

You can print a set of forms for each record in a link by using File | Print Link or @PRINTLINK. You can reduce the number of forms that print by using a filter or a restricted range. See Chapter 10, "Linking basics," for more information.

Creating @functions

You can create your own @functions and automatically add them to the list of ObjectVision functions. Once a function is registered it appears in the Paste function box.

You can write a DLL that uses the Pascal calling conventions to define @functions. *You can also use a DLL that someone else has written*—you don't have to be able to write a DLL to make use of one that someone else has written.

You use @REGISTER in ObjectVision to register the function from the DLL to ObjectVision.

After you register the function, you can use it in your applications. For more information, see Chapter 18, "Using @REGISTER with DLLs."

A P P E N D I X

B

Error messages

This appendix describes error messages you might see while working with ObjectVision. Not all error messages are listed here since many are self-explanatory or documented in other manuals.

For example, ObjectVision relays error messages from linked databases and linked DDE applications to you. If an error occurs in a linked program, you can look up that error in that application's manual.

A disk write error occurred while writing to *filename*. You will not be able to open and run the resulting application file.

The file you are writing to may be corrupted, or the physical disk may be flawed. You might want to check your disk by using a hard disk utility. If the disk is OK, you may want to delete the corrupted file and re-create the application.

Access denied to file *filename*.

If you are using a network, you may be trying to access a locked file. You may be trying to access a network file to which you don't have rights. In this case, you will need to gets rights from the Network Administrator.

Cannot allocate memory for bitmap. Cannot allocate memory for *filename*.

You have exceeded your available RAM. Try closing other applications. If this does not work, you may need to use a bitmap that uses less memory.

Cannot continue this application. It is too large.

You have exceeded your global memory. You may want to split your application into two or more smaller applications.

Cannot create file *filename*.

You may have a disk error. ObjectVision needs to write to the disk to create the file, but it can't. Consider running a hard disk utility application. Also, the directory may be full.

This can also occur if the disk is write-protected.

Cannot open file *filename*.

You may be trying to open a file that was not created using ObjectVision. If you are sure the file you are trying to open was created using ObjectVision, then the file may be corrupted and can no longer be used.

Cannot read graphic file *filename*.

ObjectVision can only import Windows metafiles and bitmaps. You may be trying to import another graphic type.

Cannot read next ASCII record—link cleared.

The ASCII file from which you are reading may be corrupted, or it may just be in a format ObjectVision cannot recognize. The ASCII file should be comma-delimited. If it isn't, you will need to alter the ASCII file so that it is comma-delimited.

Cannot register add-in function, library file does not contain *filename*.

The DLL file you've written is missing a crucial part. Add the necessary part and try to register the function again.

Cannot register add-in function, argument list is invalid.

When you used @REGISTER, you used the wrong argument count. You might want to check the DLL to see what the arguments are. You also might have exceeded the 13 argument limit.

Cannot register add-in function, name is invalid.

The name you have used for your add-in function is invalid. You may have tried to use a non-ASCII character, or the name could be too long, or the name could be a duplicate of an already registered function.

Cannot reposition at this time. No current record.

You are at an unspecified location in the database while attempting to perform a Next or a Previous. You must be on a record to perform either of these commands. Use @TOP or @BOTTOM to locate a record, then retry Next or Previous.

Cannot use event function in conditions, value tree conclusions, or filters.

You have tried to use an event function where it is invalid. You can only use event functions in event conclusions. If you always use the Paste Function button you will see which functions are available, and you can avoid seeing this message.

Clipboard does not contain graphic. Use .OVG file?

If you want a graphic on your ObjectVision application, you must first copy it to the Clipboard, then transfer it to ObjectVision. The graphic must be either a bitmap or a metafile. ObjectVision saves the graphics with a default .OVG extension. Once an .OVG file has been created, it can be used on different ObjectVision applications.

To select an existing .OVG file, choose Yes. ObjectVision lists the .OVG files in the current directory. From this dialog box you can also change directories or drives.

Condition contains more than one line.

Conditions can only be one line long. Conclusions, however, can consist of multiple lines.

Could not create ASCII link for reading. Could not create ASCII link for writing. Could not create ASCII link for appending.

You don't have access to the directory you have specified for the ASCII link. You will need to get access to that directory or specify another directory.

Could not print.

You may not be attached to a printer, or the printer you are attached to has had a failure. Check to see that you are attached to your printer, then make sure the printer is online and active. Also, the printer might not be installed. To install the printer, use the Windows Control Panel, and consult your Windows manual. You might also want to read the printing section in Chapter 1, "Using ObjectVision applications."

Dangerously low system resources—close other applications before continuing.

You may be running too many applications for you computer to handle at once. You'll have to close other applications to free some of your RAM before you can continue with ObjectVision.

Data dictionary path does not exist or is invalid. Data dictionary path is invalid. Specify an existing absolute path.

The Btrieve table you are linked to has a data dictionary file that ObjectVision needs. You must specify the correct path to this file. Check your Btrieve link to ensure that you have typed the correct path to the data dictionary file.

Database already locked—will try again.

The database table you are trying to use from the network is being used by someone else.

Disk error accessing ASCII link. Link is no longer usable. Disk error reading from ASCII link. Link is no longer usable. Disk error writing to ASCII link. Link is no longer usable.

The disk containing the ASCII files is damaged. Try running a hard disk utility.

Disk full.

You have filled your disk. You will need to backup or shrink your files before you can save anything else to this disk. You might also want to store data on a floppy disk.

DOS error *Error* while opening file *Filename*.

Consult your DOS manual for all DOS errors. There may be an error with the file type or disk.

ERR

This value appears in fields when ObjectVision evaluates an expression and an error occurs. If you divide by zero, you'll also see this value. You might want to check the expressions in your trees of that field.

Error in formula.

There is an error somewhere in the expression you've written. Check the expression to find the error. You may have misspelled a field or link name, missed a comma in an argument, or forgot a

quotation mark or parenthesis. You may also want to read Chapter 6, "Writing expressions."

Error loading Btrieve DLL. Error loading Paradox engine DLL.

ObjectVision can't locate the DLL file. You should make sure that the file is in the same directory as the .EXE or .OVD file or in the Windows subdirectory.

Exceeded internal memory limit. Cannot continue.

You ran out of local memory. You will need to increase your RAM or reconfigure it for optimal use with Windows.

Expression is not complete.

See Chapter 6, "Writing expressions," for instructions writing expressions.

Expression too complex.

Your expression has exceeded 4K. This could be because of long field names, so you'll want to shorten your field names.

Field name already exists.

You must select another name for this field. You cannot have duplicate field names or table names on one form.

Field value is incomplete.

You receive this message when typing values in Picture fields. You must complete the field's requirements, or press *Esc* to leave the field blank. For instance, a picture file for a phone number might look like (###)###-####. You must have 10 numbers to fill in this field. See page 20 for a list of Picture field symbols and their meanings.

Field value must be a number.

This field is a numeric type. You must enter a number or redefine the field type.

Field value must be a date/time value.

This field is a date/time type. Either enter a date or time value or redefine the field type.

Form already exists. Form name already exists.

You must select another name for this form. You cannot have duplicate form names in a stack.

Index file is out of date.

The dBASE index isn't up to date with the data. You may need to rebuild the index.

Insufficient system resources. Too many applications are currently open to operate properly.

Shut down any other applications that you don't need before continuing.

Internal error encountered. Please report the following error code:

If you see this message, copy the code and the steps to reproduce the error and either send the information to Borland or report it to Technical Support (see Contacting Borland in *Getting Started*).

Invalid file format for ASCII link.

The file you are linked to is either not an ASCII file or the ASCII file is not a comma-delimited file. You cannot link to this file.

Invalid filename.

You have typed a filename that contains unrecognized characters. Enter a filename that follows the 8-character DOS parameters.

Invalid function on link *Linkname*.

You may have tried an @function on a table that can't take that function. For example you could've used @PREVIOUS on an ASCII link.

Link is not yet created. Link name not found.

The link name used in an link @function doesn't match any of the open links. Verify that the correct link name is used in the function.

Maximum open files.

Check your CONFIG.SYS file for the statement FILES=*nn*. Increase the number slightly, then restart your computer.

Memory allocation error.

Close any applications you don't need before retrying.

NA

This error appears in a field when ObjectVision evaluates a tree and can't find a true condition. Adding an Otherwise condition to

the tree will fix this problem. However, this error may also occur if a DDE link is opened, but the server application can't recognize the DataField.

Nesting too deep.

Expressions can only nest to eight levels. You'll need to rewrite the expression that caused this error message.

Not enough disk space for picture file.

You don't have enough room on your disk to save the picture file. You will need to erase, backup, or shrink the files on this disk before you can save anything else to this disk.

Not enough memory to sort.

Paradox requires 2MB of hard disk space to sort.

Paradox table name contains invalid characters.

Paradox table names follow the DOS 8-character standard. Type in a table name that follows this format. The .DB extension will be automatically attached, so you don't need to type it.

Problem with expression.

The expression you've typed is not a valid expression. You will need to follow the expression guidelines in Chapter 6, "Writing expressions" before your expression can be accepted.

Remote data not accessible. Start application?

You have tried to create a DDE link to an application that isn't running. If you click OK, ObjectVision will start that application so the link can continue.

Remote links to document exist. Close remote links?

You're trying to shut down an application that has a DDE link to it. By clicking OK you shut down that application, but you also clear all the DDE links from that application to your ObjectVision application.

Seek on database with no index file.

You have tried to search through a database that does not contain an index. To solve this you will need to either create an index for that table or use an ObjectVision filter to retrieve the information you need from the database table.

Table name already exists.

Each table name must be unique. Type another name, or add a number to the end of the table name. For instance, you could call a table Table1, Table2, and so on.

Too many copies of *Filename* responded to request for remote data. Extra response ignored.

You have several copies of one application open in Windows. To prevent this from happening, only open one copy of each application at a time. ObjectVision simply ignores the responses from other copies of the same application.

Too many open files. Can't open *filename.*

To prevent this from occurring, modify the FILES=*nn* line in your CONFIG.SYS to include more files than it currently handles. Make sure you restart your computer after you make changes to CONFIG.SYS. If you're using Paradox, update the Paradox engine (see *Getting Started*).

Too many records in sort.

The database table you've created needs more memory than your computer can handle for a sort. To prevent this, limit your tables to a smaller size or increase you computer's memory.

Tried to open a link with a preexisting link name.

Link names must be unique. Try typing another name.

Tried to read before the start of a dBASE link.

You were at the top of the dBASE table when you pressed the Previous button. There is no previous record to read.

Tried to read data from an ASCII link that was created for writing.

When you created the ASCII link, you created it for writing only. You will need to change the link either to read or to append. You could also create another link for reading only.

Tried to read past the end of a dBASE link.

You were at the bottom of the dBASE file when you pressed the Next button. There is no next record to read.

Tried to rewind an ASCII link that was created for appending.

Most likely you pressed the TOP button or used @TOP with an ASCII link that was created for appending or "attaching" records

to the end of the file. If you need to see the top of the file, create another link for reading to this same file.

Tried to write data to an ASCII link that was created for reading.

When you created the ASCII link, you created it for reading only. You will need to change the link either to write or to append or create another link for writing or appending.

Tried to read past end of ASCII file.

When you're at the end of the ASCII file and use either the NEXT button or @NEXT, you'll get this error. This is because you're at the end of the file and there are no more records to view.

Unable to open ASCII file as specified.

You have probably typed an incorrect path name. Check to see that the file you specified is really in that path. If it is, the file you're trying to open is either not an ASCII file or it is not a comma-delimited ASCII file.

Unable to open database as specified.

This could mean one of two things. The path you specified could be incorrect, or the path to the index file is incorrect. Check for both, then retype the correct path.

Unable to open DDE link as specified.

Check to see if the DDE application is in your DOS path. Is the application's document also in your path? If not, either add them to your DOS path, or type the entire path name in the link.

Unable to write record. Does not pass *Expression* filter

You've created a write filter to the database table and you're trying to write values to the table that do not pass the filter. You'll either have to change the values you're trying to write or rewrite the filter expression.

Unrecognized function in expression. Unrecognized function name.

If you've tried to use an @function in an expression, you may have mistyped it. You might also have tried to use an unregistered function. To prevent from seeing this message again, use the Paste Function button.

Unrecognized operator in expression.

You may have typed a character that ObjectVision doesn't recognize. Check your expression for any unwanted characters.

Unrecognized value in expression.

You may have typed a value that ObjectVision doesn't recognize. ObjectVision can't use scientific notation.

Unterminated string in expression.

When you created the expression, you included a string but forgot to include a concluding double quotation mark. Find the string in your expression, then add the double quote.

Wrong argument count.

Check the arguments for the function you are trying to use. ObjectVision only allows up to 13 arguments in a function. To prevent this message from occurring again, check the Paste Function box that includes the list of arguments needed for that function.

Wrong number of parameters in expression.

Check the parameters in the expression you've written.

You need a more recent version to open and run —.

You are trying to use an ObjectVision application that was created by a later version of ObjectVision than yours. You won't be able to use this application unless you upgrade to a more recent version of ObjectVision.

A P P E N D I X

C

Application limits

This appendix lists the limits for ObjectVision applications. It also discusses Help | About, which displays memory use and other information about an open application.

Table C.1
ObjectVision limits

Element	Limit
Application memory use	16,768,000 bytes
Number of fields	Limited by memory
Number of forms	Limited by memory
Number of tree nodes	Limited by memory
Number of tables	Limited by memory
Number of links	Limited by memory
Conclusions	4,096 characters
Conditions	4,096 characters
Filter expressions	4,096 characters
VField expressions	4,096 characters
Field value	4,096 characters
Help text	4,096 characters
Text value	4,096 characters
Form name	254 characters
Field name	254 characters
Date range	1/1/1800 to 12/31/2099, inclusive
@Function arguments	14 (13 arguments + 1 return value)
Numeric values	18 digits, excluding the decimal point
Path name length	63 characters

Getting application information

Help | About lists the following information about an open application:

- number of forms
- number of fields
- number of tree nodes
- memory use

As you design your application, use Help | About to track its size. If it nears the maximum permitted memory use (16,768,000 bytes), you might consider dividing your application into two or more related applications. You can link your application segments with DDE links. DDE links are explained in detail in Chapter 16, "Linking through DDE."

A P P E N D I X

D

The ANSI character set

The ANSI character set defines character codes for 256 characters. The ASCII character set is the same as the first 128 ANSI characters (0-127). The numeric keypad *must* be used when you create an ANSI character. To create one of the characters, hold down *Alt*, type 0 (zero), type the corresponding code number, and then release *Alt*.

Code	Char	Code	Char	Code	Char	Code	Char	Code	Char	Code	Char
0-31†		64	@	97	a	147-159†		192	À	225	á
32	(space)	65	A	98	b	160	(space)	193	Á	226	â
33	!	66	B	99	c	161	¡	194	Â	227	ã
34	"	67	C	100	d	162	¢	195	Ã	228	ä
35	#	68	D	101	e	163	£	196	Ä	229	å
36	$	69	E	102	f	164	¤	197	Å	230	æ
37	%	70	F	103	g	165	¥	198	Æ	231	ç
38	&	71	G	104	h	166	¦	199	Ç	232	è
39	′	72	H	105	i	167	§	200	È	233	é
40	(	73	I	106	j	168	¨	201	É	234	ê
41	)	74	J	107	k	169	©	202	Ê	235	ë
42	*	75	K	108	l	170	ª	203	Ë	236	ì
43	+	76	L	109	m	171	«	204	Ì	237	í
44	,	77	M	110	n	172	¬	205	Í	238	î
45	-	78	N	111	o	173	-	206	Î	239	ï
46	.	79	O	112	p	174	®	207	Ï	240	ð
47	/	80	P	113	q	175	¯	208	Ð	241	ñ
48	0	81	Q	114	r	176	°	209	Ñ	242	ò
49	1	82	R	115	s	177	±	210	Ò	243	ó
50	2	83	S	116	t	178	²	211	Ó	244	ô
51	3	84	T	117	u	179	³	212	Ô	245	õ
52	4	85	U	118	v	180	´	213	Õ	246	ö
53	5	86	V	119	w	181	µ	214	Ö	247	÷
54	6	87	W	120	x	182	¶	215	×	248	ø
55	7	88	X	121	y	183	·	216	Ø	249	ù
56	8	89	Y	122	z	184	¸	217	Ù	250	ú
57	9	90	Z	123	{	185	¹	218	Ú	251	û
58	:	91	[	124	|	186	º	219	Û	252	ü
59	;	92	\	125	}	187	»	220	Ü	253	ý
60	<	93	]	126	~	188	¼	221	Ý	254	þ
61	=	94	^	127-144†		189	½	222	Þ	255	ÿ
62	>	95	_	145	‘	190	¾	223	ß		
63	?	96	`	146	’	191	¿	224	à		

†Characters not supported by Windows

A P P E N D I X

E

Keyboard operations

This appendix lists the keys, key combinations, and general rules for using ObjectVision with the keyboard.

Getting online help

Alt H	Opens the Help menu.
F1	Displays Help for a highlighted field *if* the form designer created help for that field. If a command or tool is highlighted, help for that topic appears. Otherwise, *F1* opens the index to ObjectVision help.
Esc *or* Alt F4	Closes the Help window.
Alt Spacebar	Opens the Help Control menu.
Alt F4 *or* Alt F X	Closes the Help window.
Tab	Selects the next underlined help topic.
Shift Tab	Selects the previous underlined help topic.

Enter Chooses the selected help topic. If the topic has a solid underline, help for that topic appears. If the topic has a dashed underline, the definition of that term appears onscreen as long as you hold down the *Enter* key.

Choosing menu commands

Once you start ObjectVision, you can select the menus with the keyboard by following this general rule: *Use Alt + the underlined letter of the menu command you want to use.*

You can press Esc to unselect the menu.

Alt selects the menu. Once it's selected, you can use the arrow keys to move around the menus. Once a menu item is highlighted, you can select it by pressing *Enter*.

For example, to open an application, press *Alt+F* to select File from the main menu, then press *O* to select Open.

This rule applies for dialog boxes as well. All underlined letters represent the shortcut key for that command or selection. For example, to create a link, you'd select Tools | Links by pressing *Alt+T+L*. The Links dialog box appears. To create a DDE link, press *Alt+E* to select DDE, then press *Alt+R* to create the link (*Alt+R* clicks the Create button).

Menu shortcut keys

When a menu command has a shortcut key, you can use the keys to carry out the action. The shortcut key, if one exists, follows the command name on the menu. For example, on the ObjectVision main menu, *Alt+F4* follows File | Close, so you can use *Alt+F4* instead of pressing *Alt+F+O*, which also selects File | Close.

Esc Exits from a menu without choosing a command.

Positioning and Sizing windows

You can move the ObjectVision window or the form window by selecting the Control-menu box, then selecting Move.

Alt Spacebar Opens the ObjectVision Control menu.

Alt + - Opens the Control menu.

When the Control menu appears, select Move. The pointer changes from a single arrow to a four-pointed arrow. You can use the arrow keys to move the entire window to where you want it.

When the ObjectVision window is smaller than the screen, you can size the window by selecting the control menu then choosing Size. Use the arrow keys to size the window. Press *Enter* when the window is the size you want.

You can use the following shortcuts for any form window in a ObjectVision application:

Ctrl + F7 Lets you move the form window with the arrow keys.

Ctrl + F8 Lets you resize the form window with the arrow keys.

Ctrl + F4 Closes the form window.

Form Tool

Placing a new object on a form:

- Select the object (for example, press *Alt+O+F* to select a field object). The pointer will change from an arrow to a crosshair with a graphic of the object you chose to place on the form.
- Use the arrow keys to move to where you want a corner of the object to be.
- Press *Enter*. The pointer changes to a single crosshair.
- Use the arrow keys to move to where you want the diagonal corner of the object to be, or press *Enter* to get the default size.
- When the object is the size you want, press *Enter*. You can always resize the object later.

Selecting an existing form object:

Tab Moves to the next object, going from left to right and from top to bottom.

Shift + Tab Moves to the next object, going from bottom to top and from right to left.

Ctrl + Tab Lets you select multiple objects on a form, going from left to right and from top to bottom.

Shift Ctrl Tab — Lets you select multiple objects on a form, going from bottom to top and from right to left.

Positioning a form object:

↑ ↓ ← or → — Moves a selected form object in the direction of the key.

Shift ↑ / Shift ← / Shift → or Shift ↓ — Lets you size the selected form object. *Shift+* an arrow key moves the object's lower right corner.

Scrolling:

Ctrl PgUp or Ctrl PgDn — Scrolls a form horizontally.

PgUp or PgDn — Scrolls a form vertically.

Using dialog boxes

Esc — Closes the dialog box and leaves the settings unchanged. You can also press *Alt+C* to click the Cancel button.

Spacebar — Toggles a selected check box between checked and unchecked.

Enter — Activates a highlighted command button.

Tab	Moves to the next named option or group of options. Or, press *Alt* plus an underlined letter in the option name to select that option directly.
Shift Tab	Moves to the previous named option or group of options.
Ctrl Enter	Moves to a new line in a text box.

Fields

The following commands work in form completion mode only.

Moving between fields:

Alt Backspace	Backs up to the previous field and restores its previous value (only if pressed *immediately* after pressing *Enter* or *Tab*).
Enter	Enters a typed value in the field and moves to the next field selected by guided completion.
Tab	Enters a typed value in the field and moves to the next field (going from left to right and from top to bottom).
Shift Tab	Moves to the previous field (going from bottom to top and from right to left).

Selecting field values:

Spacebar	Activates the highlighted button or checks the highlighted check box or radio button.
Enter *or* Tab	Checks a highlighted check box or radio button. To highlight a check box or radio button, press ↓ or ↑ when the field is selected, or press the first letter of the response you want (repeating if necessary until you move to the response you want).
Spacebar	Toggles a true/false field between checked and unchecked.

Editing text in fields:

Shift → *or* Shift ←	Highlights text. Highlighted text is replaced by the next characters you type. You can also use *Home* or *End* instead of ↑ or ↓ to highlight text.

Esc	Restores the previous field value (*only* if you are still in the field and haven't pressed *Enter* to enter the new value).
Home	Moves to the left of the first character in the field.
End	Moves to the right of the last character in the field.
Del	Erases the character immediately to the right of the pointer, or erases any highlighted text.
Backspace	Erases the character immediately to the left of the pointer, or erases any highlighted text.

Viewing trees

Moving between nodes:

Home	Moves to the root node of the tree.
↑	Moves to the previous node at the same nesting level in the tree.
↓	Moves to the next node at the same nesting level in the tree.
←	Moves to the node at the preceding nesting level in the tree.
→	Moves to the first node at the subsequent nesting level in the tree.

Scrolling:

Ctrl PgUp	Scrolls a tree horizontally.
or	
Ctrl PgDn	
PgUp	Scrolls a tree vertically.
or	
PgDn	

Changing the display:

Ctrl Home	Enlarges the size of the tree.
Ctrl End	Reduces the size of the tree.

A P P E N D I X

F

Using the Runtime disk

This appendix is an overview of ObjectVision Runtime and how you use it when you copy your applications.

ObjectVision Runtime has the same system requirements as ObjectVision. See Getting Started for this list.

ObjectVision Runtime, which is included with ObjectVision, allows you to give your applications to users who don't have this development version. A user can install ObjectVision Runtime and use your applications. Remember that if you're giving your applications to someone who has the development version of ObjectVision, you can save your applications with a password to get the same effect as a run-time application.

If you copy DDE-linked documents, make sure your users have a licensed copy of the application (the .EXE file).

Copy the applications you want to distribute to the ObjectVision Runtime disk. Make sure that you include all database tables, ASCII files, DDE-linked application documents, and .OVG files. Copy the run-time disk as many times as you need.

The install program on the run-time disk will install the run-time version, the README.TXT, and any applications you copy to the run-time disk in a directory called VISIONR.

Deleting OVSAM.PAK gives you extra space on the run-time disks for your applications.

The sample applications on the run-time disk are copied to a subdirectory called SAMPOLE. If you don't want to distribute them, delete the OVSAM.PAK file from the run-time disk.

If you need to give your users an explanation of your applications, you can modify the README.TXT file using a text editor. When users install ObjectVision Runtime, they'll be asked if they want to read this file.

G L O S S A R Y

All of the terms listed below are described in greater detail in the sections of the manual in which they appear.

A

application An .OVD file that you create using ObjectVision. An application consists of a stack of one or more forms, each containing objects such as fields, text, and graphics.

In DDE, the application is usually the name of the server's .EXE file. For instance, the application name for ObjectVision is VISION.

argument Information an @function needs to perform its calculation. Some @functions don't require arguments.

B

branch A node in a tree that determines the evaluation path of the tree. A branch (except the root branch) is the node that determines from its condition which path (or branch) the tree will take to make its conclusion.

button An object that is used to start an action. Buttons can be automatically created by ObjectVision when you create a link.

C

calculated field A field that uses a tree to calculate its value. A calculated field is indicated by a solid outline with an inner dotted line when it is selected.

cell A place within a column of a table object that holds one value.

client A client application starts a DDE conversation and usually receives data from the other application.

Clipboard The Windows place for temporarily storing deleted objects. After you cut or copy something in ObjectVision, it is placed in the Clipboard until you cut or copy something else. You can paste the Clipboard contents as many times as you need—a copy always remains there.

column Part of a table object. Columns contain one or more cells that display values; however, only one value in a column is current at a time.

concatenation The joining (adding) of two or more text strings into a single text string.

conclusion The part of a tree that determines the result (value or action) of an object's tree. The conclusion expression is the last node in the tree.

condition The part of a tree that determines what node is selected next.

connections When you create a link, you also create connections. Connections describe the types of values the connected fields will receive (read only, write only, or both). See Chapter 10, "Linking basics," for more information of connections and their types.

current value In a table object, only one row can be the current (selected) row. Current values are indicated in a table object by a right-pointing arrow next to the table's left border.

D

database table A table that contains saved values. You can create links to these tables so you can read (view) the values, write (change or save) values, or both. Don't confuse this with table objects, which ObjectVision uses to *display* values.

DateTimeNumber An @function argument that is a number between –36,522 (January 1, 1800) and 73,050 (December 31, 2099). The decimal portion of DateTimeNumber represents the time from 12:00 noon to 11:59:59 p.m. and is computed as a fraction of a 24-hour day.

DDE (Dynamic Data Exchange) A Windows protocol for exchanging information between Windows applications. ObjectVision can create DDE links between two ObjectVision applications, or between ObjectVision and another DDE-supported application.

DDE conversation An exchange of information between Windows applications.

decision trees See trees.

default properties Object properties that define the display format and appearance of an object can be set to default values. These default properties are attached to each object that you place on a form.

E

empty node Appears in a field's tree when the tree is empty (first created) or when a branch has no nodes under it. If an empty node is evaluated, an error value is returned.

event tree A tree that performs an action after an event has occurred to the tree's object. See also trees.

expression Used in trees and link options, expressions use one or more constant values, @functions, or strings separated by operators to perform mathematical, logical, or string operations.

F

field A uniquely named object that contains a value. Fields are the basic elements of a form.

font A typestyle used for a field label or value, or a text object.

form What ObjectVision uses to hold objects such as fields, tables, or graphics. You can think of an ObjectVision form as an electronic form that does much more than a paper form.

form completion mode The mode a user is in when filling in and using an ObjectVision application.

Form Tool The tool you use to create and edit forms and their objects.

@functions Perform calculations or operations. These functions can be used in expressions. There are two categories of @functions: value and event. Event functions can only be used in event tree conclusions.

G

Goal form The top form in the application's stack of forms. The Goal form appears when you open the application. When you select another form, it becomes the Goal form.

graphic A bitmap (.BMP) or metafile (.WMF) image that you can place on a ObjectVision form. The graphic is converted and saved to an .OVG file once you name it and place it on a form.

guided completion The technique ObjectVision uses to automatically select fields for a user. When you press *Enter*, only a field requiring input is selected. If you interrupt guided completion by selecting another field, you can restore guided completion by selecting File | Resume.

H

handles In the Form Tool, small black squares in the corners of a selected object used to size or change the object's shape or length.

I

Item In DDE links, the *item* is the place in the topic where the information is stored. In ObjectVision the item is the name of a field. The *value* in the item is exchanged between the client and server.

L

label A constant string value that doesn't need to be enclosed in double quotation marks.

label prefix A single quotation mark (') is used as the first character of an expression to force that expression to be evaluated as a label.

links The ObjectVision term for connecting to database tables, ASCII files, or DDE-supported Windows applications. Links are created using Tools | Links.

Links Tool The ObjectVision dialog boxes where you can create, modify, or delete links.

locate option A link option that triggers a link to automatically deliver values from a connected data source.

logical expressions An expression that evaluates to a logical result—either 1 (true) or 0 (false). Logical expressions are typically used in conditions or filters.

M

multiple selection In the Form Tool, highlighting several objects so subsequent actions (such as deletes or property assignments) are carried out on all selected objects at once.

N

nesting level In a value or event tree, the number of nodes away from the root node. In @functions or expressions, the number of parentheses used where each set of parentheses is one nesting level.

node A generic term for a segment of a tree. There are branch and conclusion nodes and the root node.

O

object Something you can place on a form, such as tables, fields, text, filled or rounded rectangles, lines, or graphics. You can assign objects different properties such as color, borders, or fonts.

operators Mathematical symbols used to express a relationship between two or more values in an expression. Operators have an order of precedence that affects the result of an expression.

override When you type a new value over a calculated one, the new value is an override value. After you override a calculated value, the field displays a gray dot pattern. You can restore the calculated value by using Field | Calculate.

P

picture A character field type that lets you specify a pattern to control values that users type in a field during form completion mode.

points A way of measuring font size, roughly equal to $\frac{1}{72}$ of an inch. The default Label Font is 8-point Helvetica, and the default Value Font is 12-point Courier.

precedence The order in which operations are evaluated in an expression.

property inspector In the Form Tool, you can inspect the properties for an object, a form, or a stack, by clicking the object with the right mouse button. The inspector lists all properties for the selected object, and lets you modify the settings.

protection A field property you assign to prevent users from changing the value of the field or viewing the field's tree.

R

record A set of fields in a database table. Records are related values (such as three fields Last Name, First Name, and Address all relating to one person).

reserved words In expressions, reserved words are words that have special meaning to ObjectVision. If you use a reserved word in an expression you must enclose it in double quotes. Reserved words include Yes, No, True, False, and in conditions, Otherwise.

restricted range A link option that only reads records from a database table that *exactly* match the range you specify.

root node The first node in a tree. The root node is the only node that doesn't have a condition.

S

Scratchpad form The form ObjectVision automatically creates to display any field you create but don't place on a form.

select With objects, to position the pointer on and click the object. The selected object receives the next action that you perform.

selected field In form completion mode, the selected field receives the values you type in. The selected field is highlighted with a solid line or a solid line combined with a dot pattern or dotted line.

In the Form Tool, a selected object is surrounded with a dashed line and has small black squares on its corners.

server The server application responds to the client in a DDE conversation.

stack The set of forms in the .OVD application. Think of a stack as you would think of a set of stapled forms.

Stack Tool The window that displays the order of forms in your application. In this tool, you can add, copy, paste, delete, or rearrange forms.

status The application title bar displays the form name and the status of the form: `[Goal]` is the first form in the stack, `[Complete]` is a finished form, `[Prompt]` is the status of any Scratchpad that prompts you for a value, and `[Edit]` is the form's status in the Form Tool.

syntax The format for writing expressions or @functions in ObjectVision.

T

table An object where columns of related values are displayed at the same time. Don't confuse this with database tables, which ObjectVision links to. Database table values are typically displayed in a ObjectVision table object.

title bar The highlighted horizontal bar at the top of a window. The title bar contains the name of the active application, form, or tool. Form title bars also contain the status of the form.

trees A graphical display of calculations and actions. The graphic trees are similar to flow charts because they evaluate a condition, then based on the result, perform the next branch or conclusion. See also value trees and event trees.

Topic Used with DDE, the topic is the name of the server's document or file containing the requested information. When using DDE to link to an ObjectVision application, the topic is the name of the .OVD file. You must specify the extension when you call the topic.

V

values The information contained in fields. Values can be typed in, selected from a lists, calculated in a tree, or read from a database table.

value trees A tree used to calculate a value for its field. Value trees are properties of fields and columns only. See also trees.

I N D E X

C

D

E

F

G

H

M

N

O

P

Q

R

S

W

Y

OBJECTVISION™

REFERENCE GUIDE

CORPORATE HEADQUARTERS: 1800 GREEN HILLS ROAD, P.O. BOX 660001, SCOTTS VALLEY, CA 95067-0001,
(408) 438-5300. OFFICES IN: AUSTRALIA, DENMARK, FRANCE, GERMANY, ITALY, JAPAN, NEW ZEALAND,
SINGAPORE, SWEDEN AND THE UNITED KINGDOM. • PART # 25MN-OBV02-20 • BOR 2842