

User's Guide

Formula OneTM ActiveX

High performance software for manipulating data

For Microsoft[®] Visual Basic 5.0TM and 6.0TM, Visual C++ 5.0TM and 6.0TM,
and Other Languages

Version 6.1

Tidestone Technologies, Inc.

Information in this document is subject to change without notice. Companies, names, and data used in examples herein are fictitious unless otherwise noted. No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without the express written permission of Tidestone Technologies, Inc.

Copyright © 1999 Tidestone Technologies, Inc. All rights reserved.

Formula One is a licensed trademark and Visual Components, First Impression, and VisualSpeller are registered trademarks of Tidestone Technologies, Inc.

Microsoft, MS, MS-DOS, Visual Basic, and Windows are registered trademarks and Microsoft Access and Microsoft Excel are trademarks of Microsoft Corporation in the USA and other countries.

Java, 100% Pure Java, and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc., in the U.S. and other countries.

PowerBuilder is a trademark of Sybase, Inc. or its subsidiaries.

TrueType is a registered trademark of Apple Computer, Inc.

All other company and product names mentioned may be trademarks or registered trademarks of the companies with which they are associated.

The Tidestone License Agreement, included with the product, specifies the permitted and prohibited uses of the product. Any unauthorized reproduction or use of the product, or breach of the terms and conditions of the License Agreement, is forbidden. The Tidestone License Agreement sets forth the only warranties applicable to the product and documentation. All warranty disclaimers and exclusions set forth therein apply to the information contained in this document.

Published by:

Tidestone Technologies, Inc.
12980 Metcalf Avenue, Suite 300
Overland Park, Kansas 66213

(913)851-2200
(800)884-8665
(913)851-1390 fax

www.tidestone.com

Contents

Preface Overview	ix
New Features in Formula One	ix
Version 6.0	ix
Version 6.1	x
Installation	x
Installing the Product	xi
What Does The Installation Program Do?	xi
After Installation	xii
If you Experience Installation Problems	xii
Technical Support	xii
Documentation Conventions	xiii
 Chapter 1 Getting Started	 1
Adding the ActiveX Control to Your Application	1
Getting Started in Visual C++	1
Creating a Dialog, CFormView, or CView-Based Application	2
Creating Dialog Box-Based Applications	2
Creating CFormView-Based Applications	3
Creating CView-Based Applications	3
Adding the Formula One Component to Your Project	3
Adding the Component to Your Dialog or CFormView	4
Adding the Formula One Component to Your CView	5
Working With Top-Level Properties and Methods in Visual C++	6
Accessing Formula One Objects in Visual C++	6
Handling Events in your Dialog or CFormView in Visual C++	7
Handling Events in your CView in Visual C++	7
Handling Printing and Previewing in Visual C++	7
Serializing the Control in Visual C++	7
Setting Properties in Visual C++	8
Setting Properties for a Control on a CView	8
Getting Started in Visual Basic	8
Adding the Component to your Visual Basic 5.0 or 6.0 Project	8
Setting Properties in Visual Basic 5.0 or 6.0	9
Getting Started in PowerBuilder	9
OLE 2 Presentation Style	10
Uniform Data Transfer Method	14
Standalone Worksheet Method	18
Using Formula One as a Worksheet in PowerBuilder	18

Working in PowerBuilder	19
Upgrading Formula One	20
Chapter 2 Introducing Formula One	23
Working with API Objects	23
Understanding Workbooks and Worksheets	24
Introducing the Workbook Designer	25
Using Workbooks, Views, and Invisible Workbooks	25
Working With the F1Book Control	26
Working With the F1BookView Control	27
Using Attach Methods	30
Controlling the Display of Workbook Areas	30
Saving View or Invisible Workbook Information	32
Reading and Writing Files	32
Using BLOB access	33
Writing out a Range of Cell Data	34
Chapter 3 Overview of the Workbook Designer	35
Launching the Workbook Designer	36
Docking the Toolbars	37
Using the Workbook Designer Menus	37
Using the Workbook Designer Toolbars	42
Chapter 4 Workbook Fundamentals	45
Setting up Workbooks	45
Displaying Parts of the Workbook Designer	45
Setting the Default Font	46
Setting Up the Color Palette	48
Manipulating Worksheets	48
Inserting Worksheets	49
Selecting Worksheets	50
Working with a Group of Worksheets	51
Inserting Multiple Worksheets	52
Deleting Worksheets	54
Renaming Worksheets	55
Setting Display Options for Worksheets	55
Navigating Through Worksheets	56
Navigating with the Mouse	57
Navigating with the Keyboard	58
Selecting Cells	60
Selecting Cells with the Mouse	60
Selecting Cells with Properties and Methods	61
Selecting Rows and Columns	61
Setting Selection Options	62

Chapter 5 Working With Data	63
Understanding Worksheet Data Entry	63
Adding the Formula Bar	64
Entering Data with Properties	64
Entering Multi-Line Data	65
Understanding Worksheet Data Types	66
Entering Constant Values	66
Entering Formulas	67
Using Formula Operators	68
Understanding Cell References	69
Understanding Worksheet Errors	72
Displaying Formulas	72
Built-In Worksheet Functions	72
Understanding Functions	73
Entering Functions	73
Using Autofill Lists	74
Adding Autofill Lists	75
Deleting Autofill Lists	76
Using Names	76
Calculating Worksheets	77
Setting Automatic Recalculation	77
Setting Minimal Recalculation	78
Solving Circular References	79
Limiting Data Entry	80
Denying Access to a Workbook	80
Denying Access to a Worksheet	81
Denying Access to Certain Cells	81
Denying Access to Row and Column Headings	82
Restricting Cell Data to Certain Values	82
Denying Entry of Formulas in a Worksheet	85
Restricting the Use of Certain Keys	85
Restricting the Use of Certain Mouse Actions	86
Chapter 6 Editing Worksheets	87
Copying, Moving, and Pasting Selections	87
Using Dragging to Move, Copy, and Paste Selections	87
Using Menu Commands to Move, Copy, and Paste Selections	91
Using Methods to Edit, Move, Copy, and Paste Selections	92
Transferring Data via Uniform Data Transfer	93
Finding and Replacing Data in Formula One	94
Inserting Cells, Rows, and Columns	96
Clearing and Deleting Cells, Rows, and Columns	97
Sorting Data in Worksheets	98

Chapter 7 Formatting Worksheets	99
Using Built-in Number Formats	99
Applying Number Formats to Rows and Columns	101
Obtaining Formatted Text Programmatically	101
Creating Custom Number Formats	102
Formatting Fonts	106
Aligning Data	107
Merging Cells	109
Cutting, Copying, and Pasting Merged Cells	109
Changing Row Height and Column Width	110
Setting Default Row Height and Width	110
Sizing Rows and Columns Using Menu Commands	111
Sizing Rows and Columns Using Click and Drag Actions	112
Sizing Rows and Columns with Properties and Methods	113
Freezing Horizontal and Vertical Panes	115
Setting Cell Border and Fill Formats	116
Setting Cell Borders	116
Setting Cell Fill Colors and Patterns	118
Formatting Row and Column Headings	119
Selecting Row and Column Heading Areas	119
Sizing Row and Column Headings	120
Setting Row and Column Heading Text	120
 Chapter 8 Working With Graphical Objects	 123
Creating Graphical Objects	123
Creating Graphical Objects with Methods	123
Interactively Drawing Graphical Objects	124
Picture Objects	125
Setting Mouse Mode	126
Identifying Graphical Objects	126
Naming Graphical Objects	128
Selecting Graphical Objects	128
Interactively Selecting Graphical Objects	129
Selecting Graphical Objects Programmatically	129
Moving, Sizing, and Arranging Graphical Objects	130
Interactively Moving and Sizing Graphical Objects	130
Positioning Graphical Objects Programmatically	131
Determining Graphical Object Position and Size	132
Arranging Overlapping Graphical Objects	132
Formatting Graphical Objects	133
Formatting Colors and Patterns	133
Formatting Lines (Borders) on Graphical Objects	134
Showing and Hiding Graphical Objects	135

Formatting Dropdown List Boxes	135
Formatting Check Boxes	136
Formatting Buttons	137
Selecting Check Box and Dropdown List Box Items	137
Setting Values Interactively	138
Setting Values Programmatically	138
Setting Values by Cell Reference	138
Editing Polygons	139
Chapter 9 Working With Chart Objects	141
Creating Charts	141
Using the Chart Wizard	142
Navigating in the Chart Wizard	142
Using the Gallery Page	143
Using the Style Page	143
Using the Layout Page	144
Using the Axes Page	144
Chart Options	146
Referencing Data on Another Worksheet	147
Chapter 10 Printing Worksheets	149
Printing Worksheets	149
Specifying Print Areas	150
Specifying Print Titles	151
Specifying Page Breaks	152
Specifying Margins	154
Setting Page Numbering	155
Specifying Headers and Footers	156
Setting Page Orientation	157
Setting Up Scaling for Printing	157
Specifying Page Printing Order	158
Choosing Paper Size	158
Specifying Miscellaneous Printing Options	158
Previewing Your Printout	159
Chapter 11 Working With Databases	161
Overview of Formula One Connections	161
Installing the ODBC Drivers	161
Setting up a Data Source	162
Connecting to the Data Source	162
Querying the Data Source	164
Updating or Inserting Data	167
Using PREPARE Statements	168
Binding Worksheet Columns	169

Executing PREPARE Statements	170
Disconnecting from the Data Source	170
Chapter 12 Using Formula One With the Internet	171
Writing out a Worksheet File in HTML Format	171
Embedding Formula One Data in an HTML file	172
Introducing Internet Application Development	173
Viewing a Web Page Containing Formula One	173
Adding Formula One to your Web Page	174
Using Methods and Events for Internet Development	174
Understanding Formula One's IObjectSafety Support	174
Understanding Formula One's Safe Events	175
Chapter 13 Performance Tuning and Specifications	177
Using Performance Tuning	177
Optimizing Formula One	178
Understanding Formula One's Data Structure	178
Allocating and Freeing Memory	179
Filling Worksheets with Data	179
Using Technical Specifications	180
Chapter 14 Creating Add-In Functions	183
Formula One ActiveX Add-Ins in Visual Basic	183
General Design Principles	183
Thread Safety	184
Add-In Function Requirements	184
Visual Basic Example Add-Ins	186
Formula One C++ Add-In API	189
How Add-In Functions Are Declared	189
How Add-In Functions Are Exposed to Formula One	190
How Arguments and Return Values Are Passed	191
IF1AddInArrayEx interface	194
Formula Evaluation Errors	198
C++ Example Add-In	198
Chapter 15 A-Z Worksheet Function Reference	203
Index	297

P R E F A C E

Overview

Formula One is a high-performance workbook control that allows you to create, manipulate, and print worksheets. It contains the tools needed to store, analyze, manipulate, and present your data.

New Features in Formula One

Version 6.0

- **Excel 97 support.** Formula One supports the Excel 97 and 95 file formats. Formula One reads and writes worksheets compatible with Excel 97 and 95.
- **Minimal recalculation.** Formula One now supports minimal recalculations. Under some circumstances, this can cause a dramatic improvement in recalculation speed. For information on minimal recalculation, see “Setting Minimal Recalculation” on page 78.
- **Worksheet size has quadrupled.** Formula One now supports worksheets up to 65,536 rows long. This is four times the number of rows that the previous version of Formula One provided.
- **Cell capacity has increased.** Each cell in a worksheet can contain as much as 16KB of text.
- **Cells can now be merged.** Formula One lets you merge two or more adjacent cells. This opens up a wide range of worksheet formatting possibilities. For example, developers and users can use merged cells to quickly and easily create multicolumn headings and titles, to insert blocks of text in worksheets, or to specify backgrounds and borders. For more information on cell merging, see “Merging Cells” on page 109.
- **New find and replace method.** The Formula One API has a new and improved way of doing find and replace through the API. It uses a new API object, `FIFindReplaceInfo`. A search can be set up by using the new **DefineSearch** method to create a `FindReplaceInfo` object. The search itself is done using the

new **FindNext** method. The new **Replace** method replaces the found data with new data. For information about these new objects and methods, see the Formula One Online Documentation.

- **New format for fractions.** Formula One now allows users to specify the denominator for fractional data. Now you can express a value of 0.8 as 4/5 or as 8/10 or even as 80/100.
- **Enhanced printing features.** Several printing improvements have been added to this version. Included is the new F1PageSetup Object that gives developers a wide range of paper size choices, increased control over worksheet page numbering, and the ability to set the number of copies Formula One prints by default. The Page Setup dialog box has also been redesigned. For more information on the new Page Setup features, see “Printing Worksheets” on page 149.
- **Increased date range.** Dates through the year 9999 are now accepted. The limit in previous versions of Formula One was 2078.
- **Improved cell editing.** When editing a cell, the in-cell edit space expands as needed to accommodate input.
- **New worksheet function.** Formula One now supports the SUMPRODUCT worksheet function. See page 263.
- **Mouse handling.** Support now exists for the IntelliPoint mouse. Hiding the scroll bars will prevent the user from using the mouse wheel to scroll.

Version 6.1

- **Add-In Functions.** Formula One Version 6.1 provides Add-In function support using Visual Basic and C++. This allows developers to add their own functions to Formula One. See “Creating Add-In Functions” on page 183 for details.

Installation

The Installer Program can be used to install both trial and working versions of Formula One.

You are prompted for a valid serial number during installation. If you enter a valid serial number, Formula One successfully installs as a working developer version.

The product is installed as a trial (demo) version if you do one of the following:

- press ENTER at the serial number prompt
- unsuccessfully attempt to enter a serial number three times

After the third attempt, the product installs as a trial version.

The trial version of a product is a **NON-REDISTRIBUTABLE** component and will expire after 30 days. You cannot deploy applications with this version. The trial version displays the About box every 30 minutes reminding you that you are working with an evaluation version of the product. If you wish to continue evaluating the product after the 30-day trial, contact Tidestone at (913) 851-2200 or sales@tidestone.com.

Installing the Product

The Setup program creates new directories and copies product files to your hard disk.

► **To install a Tidestone ActiveX control on your hard disk:**

1. Insert the first disk in your drive.
2. Locate and double-click SETUP.EXE.
3. Follow the Setup program directions.

What Does The Installation Program Do?

The Installer performs the following tasks during the installation process:

- Allows you to identify the components you want to install, select a directory to hold the program files, and specify a folder in which to place the program on your desktop.
- Copies the files to your hard disk.
- Updates system files in your Windows system directory or the location of your choice.
- Records your serial number. During the installation process, you enter the product serial number provided on the installation media and product registration card. The serial number is recorded and displayed in your product's About Box. You are required to provide your serial number to receive technical support and upgrade pricing on future product releases.
- Registers the ActiveX control with the Windows Registration Database. This makes the control visible and available to your development environment.

After Installation

Once you install the product, you can determine if you have a full version of the product or a trial version by displaying the About box. To display the About box, use the **AboutBox** method or choose Help > About VCF1 in the standalone Workbook Designer. Text in the About Box tells you whether you are using an evaluation copy.

After you successfully install a full version on your system, you can distribute that ActiveX control to your end users without worrying that the About box might display on their system every 30 minutes.

If you Experience Installation Problems

If you experience problems installing this product, please read the file INSTPROB.DOC located on the installation media. This file contains suggestions for fixing the most common installation problems. If problems persist, contact Tidestone Technical Support for further assistance.

Technical Support

The Tidestone technical support staff can help you with any problem you encounter installing or using Formula One. If you need assistance, contact Tidestone in any of the following ways:

- **On the World Wide Web.** For best service, send your technical support requests directly to the Tidestone Case Tracking System, which you can access from the Tidestone web site. Point your browser at:

<http://www.tidestone.com/support/tsmain.htm>

- **By telephone.** You can contact our technical support staff at (913) 851-2200 on weekdays between 9:00 a.m. and 4:00 p.m., central time.
- **By fax.** You can contact us by FAX at (913) 851-1390.
- **By mail.** Address your correspondence to:

Tidestone Technical Support Department
12980 Metcalf, Suite 300
Overland Park, KS 66213

- **In Europe, contact:**

Tidestone Europe

Lenexa House

11 Eldon Way

Paddock Wood, Kent

England TN12 6BE

Tel: +44 1892 834343

Fax: +44 1892 835843

Documentation Conventions

Throughout this documentation, typographic conventions are used to define elements and references to Formula One items. .

Convention example	Description
AxisSelected, AllowSelections, Select,	Names of events, properties, and methods, are in proper case and bold font.
➤ To install Formula One:	A series of numbered instructions are preceded by an introductory line. The introductory line begins with an arrowhead.
1.Type a:\setup.	Numbered instructions provide step-by-step directions for performing tasks. The instructions should be performed in the order they are presented. In numbered steps, items you are to enter are shown in Letter Gothic font.
<i>workbook</i>	In general sections, italic text is used for the first occurrence of a new term.
<i>fontname</i>	In reference sections, italic text indicates variable or argument information you must supply.
[<i>axis_id</i>]	In reference sections, italic text surrounded by square brackets indicates optional arguments.
{TRUE FALSE}	In reference sections, text surrounded by braces indicates you must make a choice among the items inside the braces. Choices are separated by vertical bars.
F1Book2.ATTACH F1Book1.Title	Letter Gothic font is used for all code examples.
TTF16.OCX	File names are presented in upper case text.
VtChart1. RowCount 'number of rows	In code examples, an apostrophe precedes a comment.
Format > Sheet > Properties	Choose the Properties option on the Sheet submenu of the Format menu.

Tidestone

C H A P T E R 1

Getting Started

The Formula One control can be used as an ActiveX control with several Windows-based development environments, including Microsoft Visual Basic, Microsoft Visual C++, and PowerBuilder. This chapter describes how to get started using Formula One with these development environments.

Adding the ActiveX Control to Your Application

The process you use to add an ActiveX control to your application varies slightly from one development environment to another.

➤ **To add an ActiveX control, in most cases:**

- add the control to your project
- select the control's tool from the toolbar and draw the control on a form or in a window

For steps about adding the Formula One control to your application, refer to the following sections:

- “Getting Started in Visual C++” on page 1
- “Getting Started in Visual Basic” on page 8
- “Getting Started in PowerBuilder” on page 9

Getting Started in Visual C++

Before using Formula One with Visual C++, you should read the Microsoft Visual C++ documentation and on-line help.

The following section highlights procedures required to use Formula One as an ActiveX control with the Microsoft Visual C++ 5.0 and 6.0 environments.

Important Visual C++ does not read constants such as F1Auto from the ActiveX control, so the file TTF16.h, distributed with Formula One, should be included wherever such constants are used.

Creating a Dialog, CFormView, or CView-Based Application

➤ **To create a Dialog, CFormView, or CView-based ActiveX control application in Visual C++ 5.0 or 6.0:**

1. Start Visual C++.
2. Choose File > New to display the New dialog box.
3. Select the Projects tab.
4. Browse to locate the desired directory path.
5. Type a name for your project in the Name text box,
This creates a sub-directory of that name in the current path.
6. From the Type list, select MFC AppWizard(exe) to create a project based on the MFC library.
7. Click OK.

The MFC AppWizard - Step 1 dialog box appears.

- To create a Dialog-based application select the Dialog radio button and click NEXT. Refer to “Creating Dialog Box-Based Applications” on page 2.
- To create a CFormView-based application select the Single Document or Multiple Documents radio button. Refer to “Creating CFormView-Based Applications” on page 3.
- To create a CView-based application select the Single Document or Multiple Documents radio button. Refer to “Creating CView-Based Applications” on page 3.

Creating Dialog Box-Based Applications

To create a dialog box-based application, be sure to complete the steps in “Creating a Dialog, CFormView, or CView-Based Application” on page 2 before continuing with the following steps:

➤ **To create a dialog-based application in Visual C++ 5.0 or 6.0:**

1. Click the FINISH button to accept the default options. Visual C++ builds your project.

The New Project Information dialog box appears.

2. Click OK.

Creating CFormView-Based Applications

To create a CFormView-based application, be sure to complete the steps in “Creating a Dialog, CFormView, or CView-Based Application” on page 2 before continuing with the following steps:

➤ **To create a CFormView-based application in Visual C++ 5.0 or 6.0:**

1. Click NEXT until you get to the dialog box in step 6.
2. In the Step 6 dialog box, select the class view name from the class list at the top of the dialog box.

CView appears in the Base Class list.

3. In the Base Class list, change CView to CFormView.
4. Click FINISH for Visual C++ to build your project.

Creating CView-Based Applications

To create a CView-based application, be sure to complete the steps in “Creating a Dialog, CFormView, or CView-Based Application” on page 2 before continuing with the following steps:

➤ **To create a CView-based application in Visual C++ 5.0 or 6.0:**

1. Click FINISH to accept the default options. Visual C++ builds your project.

The New Project Information dialog box appears.

2. Click OK.

Adding the Formula One Component to Your Project

When Visual C++ adds components to your project, it creates CPP and H source files defining the classes, properties, and methods for the control. It is a good idea to take a look at these files to understand what they contain. Methods and properties are not accessed the same in C++ as they are in many other languages like Visual Basic. When these files are generated, Visual C++ creates both a Get and Set method for most properties.

➤ **To add a Formula One component to your project in Visual C++ 5.0 or 6.0:**

1. Choose Project > Add To Project > Components and Controls to display the Components and Controls Gallery dialog box.
2. Select the Registered ActiveX Controls folder.

3. If the Formula One Control icon is not visible in the list, then the control was not registered properly and you may need to reinstall or try to register it from the ActiveX Control Test Container, which is available in the Tools menu of Visual C++.

4. Select the control from the Component list and click Insert. Click OK.

The Confirm Classes dialog box appears.

5. Click OK to confirm and exit the dialog box.
6. Click Close to exit the Component Gallery.

The Formula One Control appears in the Control palette.

Adding the Component to Your Dialog or CFormView

➤ To add the component to your dialog or CFormView in Visual C++ 5.0 or 6.0:

1. In the Resource Editor, display the dialog box for which you want to place Formula One.
2. Click the Formula One component in the Editor's Control palette.
3. Draw the component on the dialog box.
4. Size and place the component using the handles around the control.
5. Click the right mouse button to display the context menu.

You can view and modify the design-time properties using the context menu.

Assigning Member Variables

After you add the workbook control to the dialog box, you must assign a member variable to the control to gain access to the methods and properties at runtime.

➤ To assign member variables (for CForm or Dialog-based applications) in Visual C++ 5.0 or 6.0:

1. Choose View > ClassWizard.
2. Select the Member Variables tab.
3. Select the Formula One control in the Control ID window and click the Add Variable button.

The Add Member Variable dialog box displays.

4. Type the member variable name (e.g., something like m_F1) and click OK to accept the default variable category and type.

The MFC ClassWizard dialog box displays the variable in the Control ID window.

5. Click OK in the MFC ClassWizard dialog box to return to your project.

Adding the Formula One Component to Your CView

► **To add the Formula One component to your CView in Visual C++ 5.0 or 6.0:**

1. In the file list, display the header file for the view (<projname>view.h).
2. At the top of the file, include each of the Formula One control header files that were created when you added Formula One to your project.
3. In the Attributes section, as a public member, add the following to create member variables for each of the controls in your view:

```
CTTF1 m_F1;
```

4. Now through the file list, display the C++ source file for the view (<projname>view.cpp).
5. Start the ClassWizard, and make sure the view class is selected as the Class Name.
6. Select the View object in the Object Id list.
7. Select the “Create” message in the Messages list.

The Create handler initially presents the following code:

```
return CWnd::Create(lpszClassName, lpszWindowName, dwStyle, rect,  
    pParentWnd, nID, pContext);
```

Change this to the following:

```
if (CWnd::Create(lpszClassName, lpszWindowName, dwStyle, rect,  
    pParentWnd, nID, pContext) == 0)  
    return FALSE;  
  
if (m_F1.Create("Formula One", dwStyle, rect, this, 1000) == 0)  
    return FALSE;  
return TRUE;
```

8. Start the ClassWizard, and select view class as the Class Name.
9. Select the View object in the Object Id list.
10. Select the WM_SIZE message in the Messages list.
11. Click the Add Function button to create the OnSize handler function for this message.

12. Add the following code to the handler:

```
// TODO: Add your message handler code here
if (m_F1) {
    m_F1.MoveWindow(0, 0, cx, cy);
}
```

Working With Top-Level Properties and Methods in Visual C++

TTF1.H defines a number of properties and methods that affect the Formula One control. Methods are simply called as declared. However, each property has a Get and a Set method. For example, the **Row** property documented in the Formula One online help can be set with the **SetRow** method, and read with the **GetRow** method.

Accessing Formula One Objects in Visual C++

Formula One contains several objects, such as F1CellFormat, F1FileSpec, F1NumberFormat, etc. You can access each of these objects via wrapper classes provided by Visual C++. Wrapper classes are generated when you add the F1 control to the project.

- **To create an object using CreateDispatch, use the following code in Visual C++ 5.0 or 6.0:**

```
CF1BookView SS;
COLEException e;
if (SS.CreateDispatch(CLSID_F1BookView, &e))
{
    // commence
}
else
{
    e.ReportError();
    ::AfxThrowOLEException(e.m_sc);
}
```

The *guids* (see *italics* above) are defined in TTF16.H and are:

```
CLSID_F1FileSpec
CLSID_F10DBCConnect
CLSID_F10DBCQuery
CLSID_F1BookView
```

- **To use an API call to create an object, use the following code in Visual C++ 5.0 or 6.0:**

```
CF1CellFormat fmt(m_F1.CreateNewCellFormat());
```

- To call a method that wants an object argument, use the following code in Visual C++ 5.0 or 6.0:

```
m_F1.SetCellFormat(fmt.m_lpDispatch);
```

Handling Events in your Dialog or CFormView in Visual C++

- To assign message handlers in Visual C++ 5.0 or 6.0:
1. Start ClassWizard.
 2. In the Class Name list, select the Dialog or CFormView class that was created.
 3. In the Messages list, select the desired message to handle and click Add Function to add a handler. For this example, select Click event and click Add Function to add the handler.
 4. Click Edit Code to edit the new function.
 5. Add the following code in the function:

```
AfxMessageBox ("Click Event","You clicked on the workbook");
```
 6. Run the program and when the document is clicked, the message “You clicked on the workbook” is displayed.

Handling Events in your CView in Visual C++

In the view header, declare the Formula One event handlers to be used, in the section with all the other AFX messages. In the view source file, implement the event handlers, and define the `EVENTSINK_MAP` for the workbook. An easy way to get boilerplate for these declarations and definitions is to create a CFormView project with the same name as the CView project and use ClassWizard to generate the event procedures. Then copy them into the CView project.

Handling Printing and Previewing in Visual C++

From ClassWizard, add an `OnPrint` override to the view. Use the Formula One **FilePrintEx** method to send the workbook to the printer. If previewing, you must use the **PrintPreviewDCEx** method.

Serializing the Control in Visual C++

Using the **CopyRangeEx** and **WriteRangeEx** methods, you can read a range of data from a file and write a range of data to a file. Alternatively, you can load and save the Formula One OLE control via its `IPersistStorage` interface.

Setting Properties in Visual C++

You can easily set specific properties for the Formula One control in Visual C++ 5.0 and 6.0.

➤ **To set the properties of a control on a dialog or a CFormView in Visual C++ 5.0 or 6.0:**

1. Right-click the control in your project for which you want to set properties and choose Properties from the context menu.

The Control Properties dialog box appears.

2. Select the appropriate tab for the property settings you want to modify.

Properties are grouped together in categories, such as paragraphs, fonts, and pages.

3. Modify the property settings as needed.

For more information on each property, see “Property and Method Reference” in the Formula One online help.

4. Once you set the properties for the active control, close the Control Properties dialog box to return to your project.
5. Repeat steps 1 through 4 for each control.

Setting Properties for a Control on a CView

Since a control on a CView is created dynamically at runtime, you must call Formula One methods to make any changes to its initial properties. Make these calls in the Create handler for the CView.

Getting Started in Visual Basic

The following sections highlight procedures required to use Formula One as an ActiveX control with Visual Basic, versions 5.0 and 6.0.

Adding the Component to your Visual Basic 5.0 or 6.0 Project

➤ **To insert the component into your project:**

1. Choose Project > Components or press CTRL T from the keyboard.

You may also right-click the component palette and choose Components from the context menu.

The Components dialog box appears.

2. Select the Tidestone Formula One control from the list of available controls.
3. If the Tidestone Formula One component is not visible in the component list, click **BROWSE** to add the component.
4. Click **OK**.

The Visual Basic form is returned, and the Formula One component appears in the Component palette.

5. Double-click the Tidestone Formula One icon in the component palette to drop it on your form.

You can easily set specific properties for the Formula One component, either programmatically or through the Formula One property pages.

Setting Properties in Visual Basic 5.0 or 6.0

➤ **To set properties for a component programmatically:**

1. Select the component in your project for which you wish to set properties.
2. Press F4 to display the properties window.
3. Modify the property settings as needed.

➤ **To set properties for a component via the component properties pages:**

1. Select the component.
2. Click the right mouse button and choose **Properties** from the context menu.

The component property pages are displayed.

3. Within these property pages, you may set general features such as application and table name; determine what editing features are available to end users; and set the appearance features of the workbook.

For more information on each property, refer to your Formula One online help.

Getting Started in PowerBuilder

This section highlights the basic procedures required to begin using Formula One as an ActiveX control with the PowerBuilder environment. For detailed information, consult your PowerBuilder documentation.

You can also consult the Tidestone web site, which contains FAQs and examples on how Formula One works with PowerBuilder. Point your browser at www.tidestone.com/enviro/default.htm and click **PowerBuilder**.

The following sections include tutorials that describe the following methods:

- “OLE 2 Presentation Style” on page 10. This method inserts the Formula One control into the DataWindow and provides data via the DataWindow object.
- “Uniform Data Transfer Method” on page 14. This method simply places the Formula One control in the Application Window. Data is then copied to the clipboard where Formula One can access the data via the Uniform Data Transfer method.
- “Standalone Worksheet Method” on page 18. This method of using Formula One focuses on using it as a standalone spreadsheet without any database connectivity.

OLE 2 Presentation Style

This method inserts the Formula One control into the DataWindow and provides data via the **DataWindow** object.

Database Preparation

Prior to creating an application in PowerBuilder that accesses a database, you must ensure that you have correctly configured your environment. The following sections outline some preliminary measures to take before working with databases in PowerBuilder.

Configuring the ODBC

➤ To configure the ODBC:

1. Click Configure ODBC.
2. Select the appropriate database driver from the list.
3. Click CREATE.

The ODBC Database Driver Setup dialog box appears.

4. Depending on your selected database driver, the exact setup instructions might vary. Please refer to the PowerBuilder interface for instructions on how to locate your database file.

In general, the ODBC Database Driver Setup dialog box prompts you for the following information.

Data Source Name: A string that identifies this data source configuration in ODBC.INI.

Description: An optional long description of a data source name.

Database: An identification of the database file.

5. If you must access a **SELECT** button or a **BROWSE** button to locate your database file, select the database file and click **OK** to return to the ODBC window.
6. Click **OK** to confirm the Setup dialog box information.
7. Click **CLOSE**.

Setting the Database Profile

► To set the database profile:

1. Click **DB Profile**.

The DB Profile Painter window appears.

2. Select the database file you indicated in Step 4 of “Configuring the ODBC” on page 10.
3. If the database does not appear, click **NEW**.
4. Enter a long description of the data source name.
5. Choose **ODBC** in the **DBMS** field.
6. Select the database file from the **SQL Data Sources** window that you indicated in Step 4 of “Configuring the ODBC” on page 10.
7. Click **OK** to exit the New Database window.
8. Click **OK** to exit the DB Profile window.

Creating an Application via a PowerBuilder Generated Application Template

► To create an application via a PowerBuilder generated application template:

1. Click **APPLICATION** to open a new application.
2. Click the **NEW** button.

The Select New Application Library dialog box appears.

3. Type a filename in the text box.
4. Click **SAVE**.

The Save Application dialog box appears.

5. Type an Application name.

6. Click OK.

A message box appears prompting: Would you like PowerBuilder to generate an Application Template?

7. Click YES.
8. Click WINDOW to open a new window.

The Select Window dialog box appears.

9. Select **w_genapp_sheet** from the list.
10. Click OK.

Adding the ActiveX Control to the PowerBuilder DataWindow

► To add the ActiveX control to the PowerBuilder DataWindow:

1. Click DATAWINDOW.

The Select DataWindow dialog box appears.

2. Click NEW.

The New DataWindow dialog box appears.

3. Select QuickSelect as the DataSource.
4. Select OLE 2.0 as the presentation style.
5. Click OK.

The QuickSelect dialog box appears.

6. Select the appropriate table for your database from the list.
7. Select one or more columns from the table, or select the ADD ALL button.
8. Click OK.

The Insert Object dialog box appears.

9. Select the Insert Control tab.
10. Select the Tidestone Formula One control from the list of Control Types.

If the control is not registered, you must register the control by clicking REGISTER NEW.

11. Click OK.

The Formula One Workbook Properties dialog box appears.

12. Click OK to dismiss the Tidestone Formula One Control Properties dialog box.

The Formula One control appears in your DataWindow, and the Ole Object dialog box appears.

13. Select the Data tab.

14. Drag and drop any key Source data to Target data in the appropriate order for assignment. By dragging this information to the Target data window, you are linking the database columns to the Formula One worksheet.

15. Click OK.

16. Right-click the DataWindow and choose Properties.

17. Select the General tab and enter the name of the DataWindow object in the Name text box.

18. Click PREVIEW to preview the DataWindow object and retrieve data from the database to display in Formula One.

19. Close the DataWindow and indicate Yes to save the changes.

20. Type in a name for your DataWindow object.

21. Click OK.

22. Select File > Close to close the DataWindow.

23. The DataWindow prompts you if you want to save changes. Click Yes.

Connecting the DataWindow Object

► To connect the DataWindow object:

1. Select Controls > DataWindow.
2. Drag and drop a DataWindow on the form. Resize as necessary.
3. Right-click on the DataWindow and choose Properties from the context menu.

The DataWindow dialog box appears.

4. Assign a name to your DataWindow in the provided text box or use the default name.
5. Browse to select the name of the DataWindow object created in the section titled “Adding the ActiveX Control to the PowerBuilder DataWindow” on page 12.

The DataWindow control is now bound to the DataWindow object and acts as an interface to the database.

6. Click OK to accept the values and exit back to the DataWindow dialog box.
7. Click OK to dismiss the DataWindow dialog box.

Creating a Transaction Object for the Application Open Event

➤ **To create a transaction object for the Application Open event:**

1. Right-click the form.
2. Choose Script from the context menu.
3. From the Select Event drop-down list, select the Open Event for the form.
4. Append the following script to the Open Event script:

```
transaction DBTrans
DBTrans = Create transaction
DBTrans.DBMS = 'ODBC'
connect;
dw_1.settransobject (SQLCA)
dw_1.retrieve()
```

Note dw_1 represents the default DataWindow control name; supply the name of your DataWindow control as created in the section titled “Creating the DataWindow Object” on page 16.

5. Click the Close box on the Script window.
6. Save the script when prompted.
7. Run your application.

Uniform Data Transfer Method

This method inserts the Formula One control into the DataWindow and provides data via the **DataWindow** object.

Creating a New Application Window

➤ **To create a new application window:**

1. Click APPLICATION to open a new application.
2. Click the NEW button.

The Select Application Library dialog box appears.

3. Type a filename in the text box.
4. Click SAVE.

The Save Application dialog box appears.

5. Type an Application name.
6. Click OK.

A message box appears prompting: Would you like PowerBuilder to generate an Application Template?

7. Click NO.

Modifying the Application Open Event

► To modify the application open event:

1. Click SCRIPT from the PowerBar.
2. From the default Select Event drop-down list, select the Open Event.
3. Modify the Open Event script to populate the SQL object and create a startup application window.

The Open Event script should read as follows:

```
/* Populate sqlca from current PB.INI settings */
sqlca.DBMS = ProfileString ("pb.ini","database","dbms","")
sqlca.database = ProfileString ("pb.ini","database","database","")
sqlca.userid = ProfileString ("pb.ini","database", "userid","")
sqlca.dbpass = ProfileString ("pb.ini","database","dbpass","")
sqlca.logid = ProfileString ("pb.ini","database","logid","")
sqlca.logpass = ProfileString ("pb.ini","database",
    "LogPassWord","")
sqlca.servername = ProfileString
    ("pb.ini","database","servername","")
sqlca.dbparm = ProfileString ("pb.ini","database","dbparm","")
/* Uncomment the following for actual DB connection */
connect;
if sqlca.sqlcode <> 0 then
    MessageBox ("Cannot Connect to Database", sqlca.sqlerrtext)
    return
end if
Open (w_fone_demo) // This should be the application window name
                    that you wish to place the control in and wish to be the
                    startup application window. (eg. W_test_demo) If it does not
                    exist, you must create one.
```

Placing the Control in the Application Window

➤ **To place the control in the application window:**

1. Click the window to display the window painter.
2. To insert an OLE control, choose Controls > OLE.
The Insert Object window appears.
3. Select the Insert Control tab.
4. From the Control Type list, choose the Tidestone Formula One Workbook control.
5. Click OK.
6. Click the form to draw the Formula One control in the window.

Creating the DataWindow Object

➤ **To create the DataWindow object:**

1. Click DATAWINDOW.
2. Click NEW.
3. Select OLE 2.0 and click OK.
4. Select the appropriate table for your database from the list.
5. Select one or more columns from the table, or select ADD ALL.
6. Click OK.

The Insert Object dialog box appears.

7. Select the Insert Control tab.
8. Select the Tidestone Formula One control from the list of Control Types.

If the control is not registered, you must register the control by clicking REGISTER NEW.

9. Click OK.
10. The Formula One control appears in your DataWindow, and the Ole Object dialog box appears.
11. Click the Data tab, if necessary.

12. Drag and drop any key Source data to Target data in the appropriate order for assignment. By dragging this information to the Target data window, you are linking the database columns to Formula One.
13. Click OK.
14. Click PREVIEW to preview the DataWindow object and retrieve data from the database to display in Formula One.
15. Close the DataWindow and indicate Yes to save the changes.
16. Type a name for your DataWindow object.
17. Click OK.
18. Select File > Close to close the DataWindow.
19. The DataWindow prompts you if you want to save changes. Click Yes.

Connecting the Control

► To connect the control:

1. Select Controls > DataWindow.
2. Drag and drop a DataWindow on the form.
3. Right-click the DataWindow control and choose Properties from the context menu. The DataWindow dialog box appears.
4. Assign a name to your DataWindow in the provided text box or use the default name.
5. Click BROWSE to browse and select the name of the DataWindow object that you just created in the previous section titled “Creating the DataWindow Object” on page 16.
6. Click OK.

The DataWindow control is now bound to the DataWindow object and acts as an interface to the database.

The Constructor Event

When the Constructor Event fires, data is copied from the DataWindow object to the clipboard where Formula One can access it via the Uniform Data Transfer method.

1. Right-click the Formula One control and choose Script.
2. From the default Event drop-down list, select the Constructor Event.

3. Type the following code:

```
int li_rc
string ls_data
dw_1.settransobject(sqlca)
li_rc = dw_1.retrieve()
if li_rc > 0 then
    ls_data = dw_1.describe("datawindow.data")
    ole_1.SetData(ClipFormatText!, ls_data)
    ole_1.object.refresh()
end if
```

4. Close the event window and confirm to save changes.

Standalone Worksheet Method

This method of using Formula One focuses on using it as a standalone spreadsheet without any database connectivity.

Using Formula One as a Worksheet in PowerBuilder

You may want to use Formula One as a worksheet in PowerBuilder without benefit of any database connectivity to supply data. You can either enter data in the Formula One worksheet yourself, or provide data to the worksheet through any number of copy methods.

1. Click APPLICATION to open a new application.
2. Click NEW. The Select New Application Library dialog box appears.
3. Type a filename in the text box.
4. Click OK.

The Save Application dialog box appears.

5. Type an Application name.
6. Click OK.

A message box appears prompting: Would you like PowerBuilder to generate an Application Template?

7. Click YES.
8. Click WINDOW to open a new window.

The Select Window dialog box appears.

9. Select **w_genapp_sheet** from the list.
10. Click OK.

Placing the Formula One Control in the Application Window

➤ **To place the Formula One control in the application window:**

1. To insert an OLE control, choose Controls > OLE.

The Insert Object window appears.

2. Select the Insert Control tab.
3. From the Control Type list, choose the Tidestone Formula One Workbook control.
4. Click OK.
5. Click the form to draw the Formula One control in the window.

You can now use Formula One in the PowerBuilder environment.

Working in PowerBuilder

Calling ActiveX Properties and Methods in PowerBuilder

The syntax to access an ActiveX property or method is as follows:

```
<Ole_Object>.object.<Ole Property or Method>
```

Property Example:

```
Ole_1.object.ShowGridlines (False)
```

Method Example:

```
Ole_1.object.SetSelection (1,1,5,5)
```

Converting General Syntax into PowerBuilder Syntax

The Formula One online help provides detailed descriptions of the specific roles of each property and method. It explains all the parameters and their settings. The general syntax for the property or method you are using can be easily converted to PowerBuilder syntax if you follow the guidelines in this section:

➤ **To convert syntax for PowerBuilder:**

1. Go to the script window where you want to call the property or method.
2. Select Design > Browse Object.
3. In the Browser, select the OLE tab.
4. In the Browser window, double-click Ole Custom Controls.

5. Find the ActiveX (OCX) control that you want to access.

Under the ActiveX control is a list of Properties and Functions with the proper PowerBuilder syntax.

Trapping Errors in PowerBuilder

The PowerBuilder ActiveX container has an event called `ExternalException`. This event fires anytime an OCX throws an exception. In the `ExternalException` event, there are arguments that you can check to find out the details of the exception. For example:

```
Script - externalexception for ole_1  
MessageBox ("Exception", description)
```

This displays the description text of the exception that is thrown.

Handling Method Parameters Passed By Reference

To handle an ActiveX method with parameters that are passed by reference, you must place the keyword `REF` in front of the parameter.

The following example shows this with the Formula One **SaveFileDialog** method.

```
string pBuf  
into FileType  
ole_1.object.SaveFileDialog ("Save File", REF pBuf, REF pFileType)
```

Upgrading Formula One

Developers who are upgrading from an earlier version of Formula One to Formula One 6.0 or 6.1 should be aware of the following issues.

- Converting 4.x or 5.x workbooks to 6.x workbooks is as easy as opening and saving a file. To open the file, use the `File > Read` menu command in the Workbook Designer, the `File > Open` command in the standalone Workbook Designer, or the **Read** or **ReadEx** methods in the API. Then save the file in 6.0 format in order to complete the conversion.
- To convert 2.0 workbooks, open the standalone Workbook Designer and use the `File > Import > Formula One 2.x` command. This menu command is only available on the standalone Workbook Designer. After importing the file, save it in the Workbook Designer using the `File > Save` command. This will save it in 5.0 format. Then you may open that 5.0 file and save it in the 6.x format.

Note You can no longer read in a Formula One 2.x file programmatically. You must use the standalone Workbook Designer to import it, as described above.

- If you want to keep both versions of the software installed as components in your container, you will need to ensure that code that refers to objects specifies which version of the software you want to use. This is because the names of the objects are the same in both versions, so the container has no way of knowing which version you want. The following example, which uses the `F1RangeRef` object, specifies that the 6.0 version of the object should be used:

```
Dim Range As TTF160Ctl.F1RangeRef
```

Another way to fix this problem is to use either the 5.0 or the 6.x version of the software, but not both.

For further information about upgrading from previous versions of Formula One, check the Tidestone website at www.tidestone.com.

Tidestone

CHAPTER 2

Introducing Formula One

Formula One provides a high-performance workbook control that allows you to create, manipulate, and print *workbooks*. Workbooks are objects that are drawn on a form or in a window of an application. You can also create *invisible workbooks*, which allow you to perform calculations or other functions without showing a workbook on a form or in a window. These objects are maintained by Formula One.

Formula One also provides the *Workbook Designer*. The Workbook Designer provides a graphical interface that allows you to design and format a workbook for your application by pointing and clicking and using menu commands. The Workbook Designer is also available as a stand-alone application which allows you to create Formula One spreadsheets.

Working with API Objects

Formula One provides a variety of API objects that are useful for the development of your applications. The main objects, `F1Book` and `F1BookView`, contain the lion's share of the Formula One functionality. Of the remaining objects, only `F1FindReplaceInfo` actually performs actions -- the remaining objects merely hold data for the two book objects to access.

The following objects are discussed in this manual and in the Formula One online help:

- **F1Book.** A Formula One workbook object. An `F1Book` object is created when you place a Formula One workbook control on a form.
- **F1BookView.** A Formula One workbook view. An `F1BookView` object is a windowless, invisible workbook. It can contain data of its own, or it can be attached to a workbook, but have its own selection and view settings.
- **F1FindReplaceInfo.** A Formula One find/replace object. An `F1FindReplaceInfo` object contains properties and methods for find and replace operations.

Note The following objects hold data. They must be used in conjunction with **F1Book** or **F1BookView** methods to perform actions.

- **F1CellFormat.** A Formula One cell format object. An F1CellFormat object contains properties that describe a cell format.
- **F1EventArg.** A Formula One event argument object. An F1EventArg object represents a reference to a variant value.
- **F1FileSpec.** A Formula One file spec object. An F1FileSpec object contains properties that describe a workbook file.
- **F1NumberFormat.** A Formula One number format object. An F1NumberFormat object contains properties that describe a number format.
- **F1ObjPos.** A Formula One object position object. An F1ObjPos object contains properties that identify the position of a specified object.
- **F1ODBCConnect.** A Formula One ODBC connect object. An F1ODBCConnect object describes an ODBC connection.
- **F1ODBCQuery.** A Formula One ODBC query object. An F1ODBCQuery object describes a query.
- **F1PageSetup.** A Formula One page setup object. An F1PageSetup object contains properties that describe the page setup for printing worksheets.
- **F1RangeRef.** A Formula One range reference object. An F1RangeRef object contains properties that identify a range of cells.
- **F1Rect.** A Formula One rectangle object. An F1Rect object contains properties that identify the rectangular area of a range or object on a worksheet.
- **F1ReplaceResults.** A Formula One replace results object. An F1ReplaceResults object contains properties that identify how many items have been found and how many have been replaced.

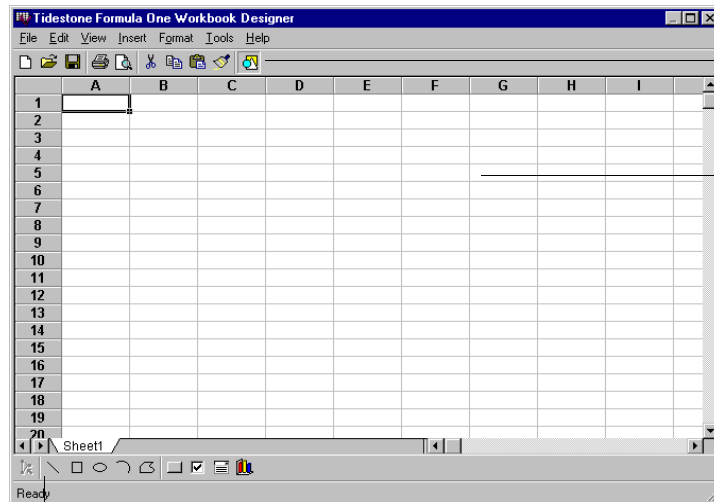
Understanding Workbooks and Worksheets

When you open or create a file in Formula One, you are creating or opening a *workbook*. Workbooks store cell data, cell formulas, workbook formatting information, and workbook-specific information such as printing attributes and calculation attributes. You can open multiple workbooks simultaneously. Formulas in one workbook can refer to cells in other workbooks. The Formula One engine manages all open workbooks.

A workbook is a collection of individual *worksheets*. Worksheets allow you to show and analyze data. Data can be manipulated on several worksheets simultaneously, and you can base calculations on data from multiple worksheets. Worksheets are useful for organizing information into separate groups. For example, you might have the year-end sales figures for each sales region on a different worksheet within the same workbook. Having all the information in one worksheet can be cumbersome; splitting it into separate files makes working with the data inconvenient.

Introducing the Workbook Designer

The Workbook Designer, shown in the following illustration, appears and behaves much like a commercial spreadsheet application. It is useful in different ways; for example, you can use it when you are designing an application or it can be launched from your application during runtime. You can also create Formula One worksheets using the stand-alone version. Refer to “Overview of the Workbook Designer” in this manual for information about the specific functions of the Workbook Designer.



This toolbar provides access to common functions.

Data and formulas are entered in the worksheet.

This toolbar allows you to create drawing objects.

Using Workbooks, Views, and Invisible Workbooks

In Formula One, you can create workbook controls (**F1Book**) and invisible workbook objects (**F1BookView**). This section discusses the purpose and functionality of these controls.

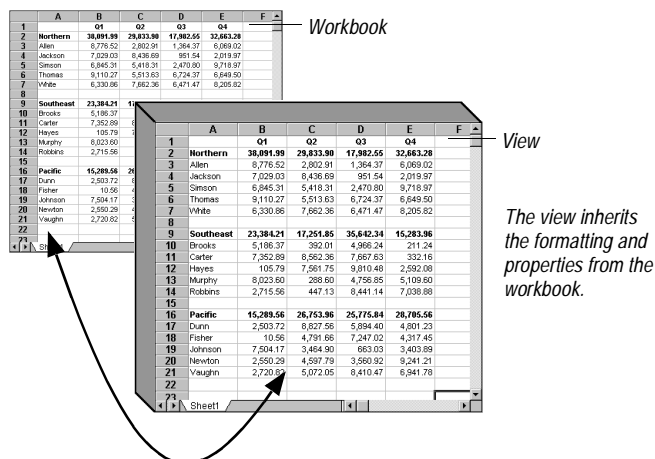
Working With the F1Book Control

When you create a workbook (**F1Book**), a default *view* is automatically created. A view provides a simplified method for showing data with different options displayed. Views contain information about:

- grid line display
- column and row heading display
- fixed row and column specifications
- maximum workbook viewing size

In addition, views can contain information about user permissions such as whether the user is allowed to select cells, enter or edit data, or resize rows and columns.

The following illustrates the concept of one view showing data from one workbook.



When you update data and formulas in the workbook, they are reflected in the view, and vice versa.

A view is attached to a workbook by default; however, you can change the view to which the workbook is attached. Multiple views can be used to display specified data from one workbook, but each view can only display data from one workbook at a time. When multiple views are displaying data from the same workbook, any change made in one view is reflected in the other views. The following illustrates this concept.

The arrows indicate the portion of the workbook that is displayed in each of the views.

	A	B	C	D
2	Northern	38,091.99	29,833.90	17,982.55
3	Allen	8,776.52	2,802.91	1,364.37
4	Jackson	7,029.03	8,436.69	951.54
5	Simson	6,845.31	5,418.31	2,470.80
6	Thomas	9,110.27	5,513.63	6,724.37
7	White	6,330.86	7,662.36	6,471.47

	A	B	C	D
9	Southeast	23,384.21	17,251.85	35,642.34
10	Brooks	5,186.37	392.01	4,966.24
11	Carter	7,352.89	8,562.36	7,667.63
12	Hayes	105.79	7,561.75	9,810.48
13	Murphy	8,023.60	268.60	4,756.85
14	Robbins	2,715.56	447.13	8,441.14

	A	B	C	D
16	Pacific	15,289.56	26,753.96	25,775.84
17	Dunn	2,503.72	8,827.56	5,894.40
18	Fisher	10.56	4,791.66	7,247.02
19	Johnson	7,504.17	3,464.90	663.03
20	Newton	2,560.29	4,597.79	3,560.92
21	Vaughn	2,720.82	5,072.05	8,410.47

The view should not be confused with the **F1BookView** API object. A view can only accommodate data from one workbook at a time. If you want to show one workbook that includes data from multiple workbooks, use the **F1BookView** API object, which is discussed in the following section.

Working With the F1BookView Control

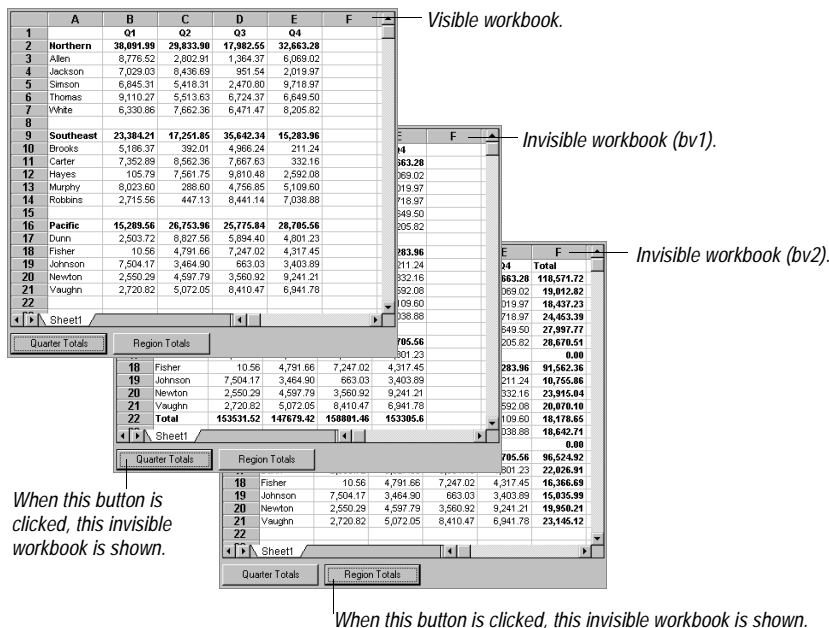
Formula One provides an **F1BookView** API object that allows you to create an *invisible* workbook control. An invisible workbook is windowless, and can only be created programmatically as an **F1BookView** object.

The **F1BookView** object is useful for any situation in which you do not want to interrupt a visible workbook, form, or window. For example, it is useful when you want to perform calculations or formatting “behind the scenes” within the invisible workbook.

In addition, if you want to build an application that shows one workbook, but includes data from multiple workbooks, you can use invisible workbooks to hold the data that you do not want the user to see. This technique works without requiring you to hide the workbooks; therefore, the users are less likely to notice changes to the visible workbook.

When you want an invisible workbook to become visible, use the **Attach** or **AttachToSS** method to attach the **F1BookView** object to an **F1Book** object. When an invisible workbook is attached, the previous attachment is severed. For additional information about attaching, refer to “Using Attach Methods” on page 30.

The following example shows how you can use the **F1BookView** API object. In this example, data shown in the **F1Book** control changes according to the invisible workbook that is attached to this visible workbook. The calculations and data that are displayed do not require cut or clear commands so the user is less likely to notice changes to the visible workbook.



The following code is used to create the invisible workbooks and perform the calculations.

These variables are declared:

```
Dim bv1 As F1BookView
Dim bv2 As F1BookView
```

This code creates the invisible workbooks, and reads in a workbook:

```
Private Sub Form_Load()
    Dim FileName As String
    Set bv1 = F1Book1.CreateBookView
    Set bv2 = F1Book1.CreateBookView
    bv1.ReadEx "wb1.vts"
    bv2.ReadEx "wb1.vts"
End Sub
```

This code is included for the "Quarter Totals" button:

```
Private Sub Command1_Click()
    F1Book1.AttachToSS bv1.SS
    bv1.TextRC(22, 1) = "Total"
```

```

        bv1.FormulaRC(22, 2) = "sum(b2:b21)"
        bv1.SetSelection 22, 2, 22, 5
        bv1.EditCopyRight
    End Sub

```

This code is included for the “Region Totals” button:

```

Private Sub Command2_Click()
    F1Book1.AttachToSS bv2.SS
    bv2.TextRC(1, 6) = "Total"
    bv2.FormulaRC(2, 6) = "sum(b2:e2)"
    bv2.SetSelection 2, 6, 21, 6
    bv2.EditCopyDown
End Sub

```

Using Properties and Methods with F1BookView

For each **F1BookView** API object, you can set or get properties to determine:

- actions and tasks the user can perform
- the appearance of the workbook and its worksheets
- the content and location of cells
- the selection status of cells
- the format of data

In general, most methods and properties that perform actions such as visually formatting a workbook or calling dialog boxes are not supported by the **F1BookView** object because it is an invisible workbook. Specifically, you cannot use the following properties and methods with the **F1BookView** object:

AllowDesigner	BorderStyle	CalculationDlg
CancelEdit	ColorPaletteDlg	ColWidthDlg
DefinedNameDlg	DefRowHeightDlg	DeleteDlg
DoCancelEdit	DoClick	DoDbtClick
DoEndEdit	DoEndRecalc	DoObjClick
DoObjDbtClick	DoObjGotFocus	DoObjLostFocus
DoObjValueChanged	DoRClick	DoRDbtClick
DoSafeEvents	DoSelChange	DoStartEdit
DoStartRecalc	DoTopLeftChanged	Enabled
FileName	FilePageSetupDlg	FilePrintPreview
FilePrintSetupDlg	Find	FindDlg
FindEx	FormatAlignmentDlg	FormatBorderDlg
FormatCellsDlg	FormatDefaultFontDlg	FormatFontDlg

FormatNumberDlg	FormatObjectDlg	FormatPatternDlg
FormatSheetDlg	GotoDlg	hWnd
InsertDlg	LaunchDesigner	LaunchWorkbookDesigner
LineStyleDlg	Mode	ObjNameDlg
ObjOptionsDlg	OptionsDlg	PasteSpecialDlg
PolyEditMode	ProtectionDlg	Replace
ReplaceDlg	ReplaceEx	RowHeightDlg
SaveFileDlg	SaveFileDlgEx	SortDlg
ValidationRuleDlg		

Using Attach Methods

You can change the view or invisible workbook that is attached to your workbook at any time using the **Attach** or **AttachToSS** method. These methods attach a view or invisible workbook to a different workbook and severs the current attachment.

When using attach methods, there are several important rules to remember.

- A view or **F1BookView** invisible workbook can be connected to only one workbook control at a time.
- An **F1Book1** workbook control can have multiple views or **F1BookView** invisible workbooks to which it is attached.
- An **F1Book1** workbook is always attached to a view. It ceases to exist if it is not attached to a view.

► To change the view or invisible workbook in which a workbook is attached:

1. Create a workbook control.
2. Create another workbook control to act as a view. Alternatively, create an invisible workbook.
3. Attach the workbook to the view or invisible workbook using the **Attach** or **AttachToSS** method.

The following code selects a workbook and attaches it to a view using the **Attach** method (*F1Book2* is the view and *F1Book1* is the workbook):

```
F1Book2.ATTACH F1Book1.Title
```

Controlling the Display of Workbook Areas

There are a number of properties you can set to determine which area of the workbook is displayed. You must identify the worksheet and range of cells to appear in the current workbook or view.

If the workbook contains multiple worksheets, you can specify which worksheet you want to display. This is accomplished by setting the **Sheet** property. Set **Sheet** to the index number of the worksheet you want to display. Sheets are indexed from left to right beginning with 1. Do not confuse the index with the worksheet sheet name that appears on the sheet tab.

To prevent users from going to another worksheet in the workbook, you can set the **ShowTabs** property to **F1TabsOff**. This hides the sheet tabs, preventing the user from changing sheets. Alternatively, you could write code within the **SelChange** event to prevent the user from changing worksheets.

You can limit the area of each worksheet that can be be seen by setting the **MinRow**, **MinCol**, **MaxRow**, and **MaxCol** properties for each worksheet. This is particularly useful when you want to use multiple views to display different parts of the worksheet.

The following illustration shows the property settings used to limit the number of rows that can be displayed in a view. The data displayed in all three views is contained in one worksheet. Notice that none of the views have vertical scroll bars. This is to prevent the end users from scrolling beyond the rows they already see.

	A	B	C	D
2	Northern	38,091.99	29,833.90	17,982.55
3	Allen	8,776.52	2,802.91	1,364.37
4	Jackson	7,029.03	8,436.69	951.54
5	Simson	6,845.31	5,418.31	2,470.80
6	Thomas	9,110.27	5,513.63	6,724.37
7	White	6,330.86	7,662.36	6,471.47

*MinCol = 1
MinRow = 2
MaxCol = 5
MaxRow = 7*

	A	B	C	D
9	Southeast	23,384.21	17,251.85	35,642.34
10	Brooks	5,186.37	392.01	4,966.24
11	Carter	7,352.89	8,562.36	7,667.63
12	Hayes	105.79	7,561.75	9,810.48
13	Murphy	8,023.60	288.60	4,756.85
14	Robbins	2,715.56	447.13	8,441.14

*MinCol = 1
MinRow = 9
MaxCol = 5
MaxRow = 14*

	A	B	C	D
16	Pacific	15,289.56	26,753.96	25,775.84
17	Dunn	2,503.72	8,827.56	5,894.40
18	Fisher	10.56	4,791.66	7,247.02
19	Johnson	7,504.17	3,464.90	663.03
20	Newton	2,550.29	4,597.79	3,560.92
21	Vaughn	2,720.82	5,072.05	8,410.47

*MinCol = 1
MinRow = 16
MaxCol = 5
MaxRow = 21*

Saving View or Invisible Workbook Information

When a workbook is saved, the settings from the view or invisible workbook that requested the save operation are saved with the workbook. When a view or invisible workbook is attached to a workbook, the view settings are retrieved from the workbook.

Reading and Writing Files

Formula One can read and write a number of file formats. The following table lists the formats and the associated file name extensions.

Format	File Extension	Description
Formula One	.VTS	Formula One 6.x format
Formula One 3, 4, or 5	.VTS	Formula One 3.x, 4.x, or 5.x format
Excel 97	.XLS	Excel 97 format
Excel 5.0 or 95	.XLS	Excel 5.0 or 95 format.
Tabbed-Text	.TXT	Tab-delimited text file including number formatting information.
Tabbed-Text (values only)	.TXT	Tab-delimited text without formatting information.
HTML	.HTM	HTML format including text formatting information. (Write-only format.)
HTML (data only)	.HTM	HTML format without graphics information. (Write-only format.)

Formula One 6.x does not support the Formula One 1.x or 2.x or Excel 4.x file formats.

Since Formula One has some features not supported by Excel, files saved in the VTS file format cannot be read by Excel. The XLS format is based on records where each record represents a unique feature or property of the workbook. If the file you save contains features not supported by Excel, they are removed when the workbook is saved as an XLS file. Likewise, Excel contains features not supported by Formula One. Unsupported features are ignored when Formula One loads an Excel worksheet or workbook.

Important If you load an Excel file that contains features not supported by Formula One, such as charts, drawing objects, or array formulas, those features are ignored. If the imported file is then written from Formula One as an Excel file and subsequently read by Excel, those features are omitted and irretrievable.

Formula One cannot read password-protected Excel files. If you intend to read files from Excel, they should not be password-protected.

The following methods and properties are available for reading and writing files in Formula One applications:

Property/Method	Description
Read, ReadEx	Reads a worksheet from disk.
ReadFromBlob	Reads a worksheet that has been stored in memory in a blob variable.
SaveFileDialog, SaveFileDialogEx	This dialog box allows you to save the current file in Formula One, Excel, tabbed text format or HTML format.
Write, WriteEx	Saves the worksheet to a file.
WriteToBlob, WriteToBlobEx	Writes a worksheet to a blob variable.

Using BLOB access

A Formula One workbook can also read data from or write data to a memory variable defined as a Binary Large Object (BLOB.) This allows you to store worksheets or workbooks in a database table and later retrieve them from the database table.

➤ To retrieve a worksheet or workbook from a database table:

1. Write code outside Formula One to read a worksheet or workbook from a database table into a BLOB variable.
2. Call Formula One's **ReadFromBlob** method to display that worksheet or workbook in the workbook control.

➤ To store a worksheet or workbook in a database table:

1. Call Formula One's **WriteToBlob** or **WriteToBlobEx** method to copy the worksheet or workbook from the Formula One control to a BLOB variable.
2. Write code outside Formula One to write the worksheet or workbook from the blob variable to a database table.

Note When you use the **WriteToBlob** and **WriteToBlobEx** methods, Formula One automatically uses the latest Formula One file format. Older versions of Formula One can't read files saved in the newer versions' formats. This means that developers who use more than one version of Formula One should be careful which version they use to write to BLOB memory variables.

Writing out a Range of Cell Data

Formula One provides the following three methods for writing out a range of cell data.

- **WriteRange** or **WriteRangeEx**. These methods write a range of cell data to a new file.
- **InsertHTML**. This method embeds a range of cell data into an existing HTML file at a specified anchor point.

➤ **To write a range of cell data to a new HTML file:**

Use the **WriteRange** or **WriteRangeEx** method to identify the cell boundaries and worksheet that you want to write data from and specify the type of new file you wish to create.

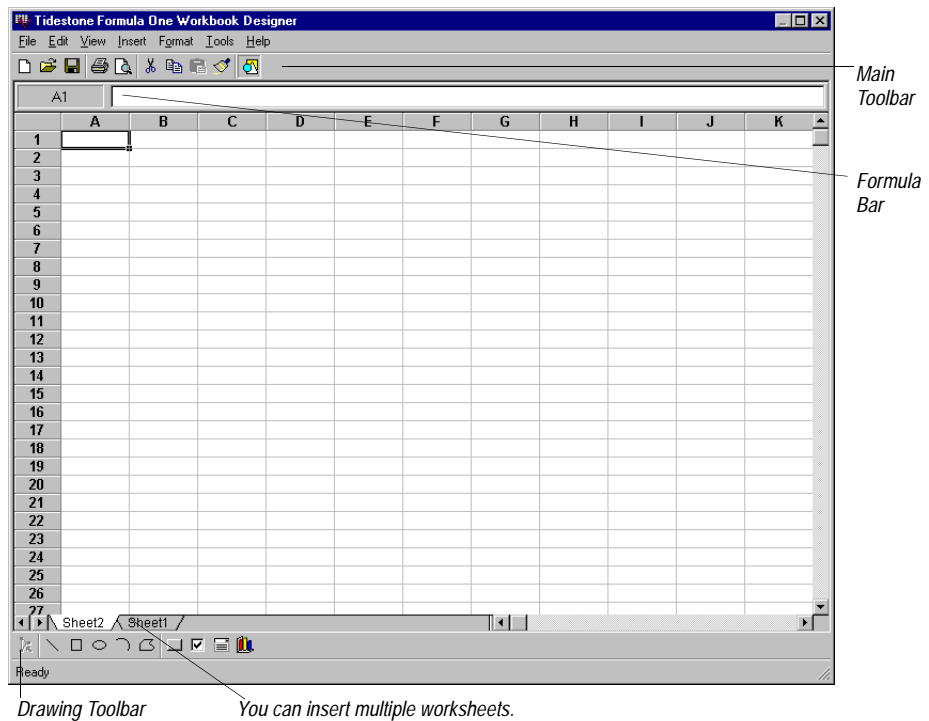
➤ **To embed a range of cell data into an existing HTML file**

Use the **InsertHTML** method to identify the cell boundaries and worksheet that you want to write data from and specify the anchor point in the file where you want the cell data embedded.

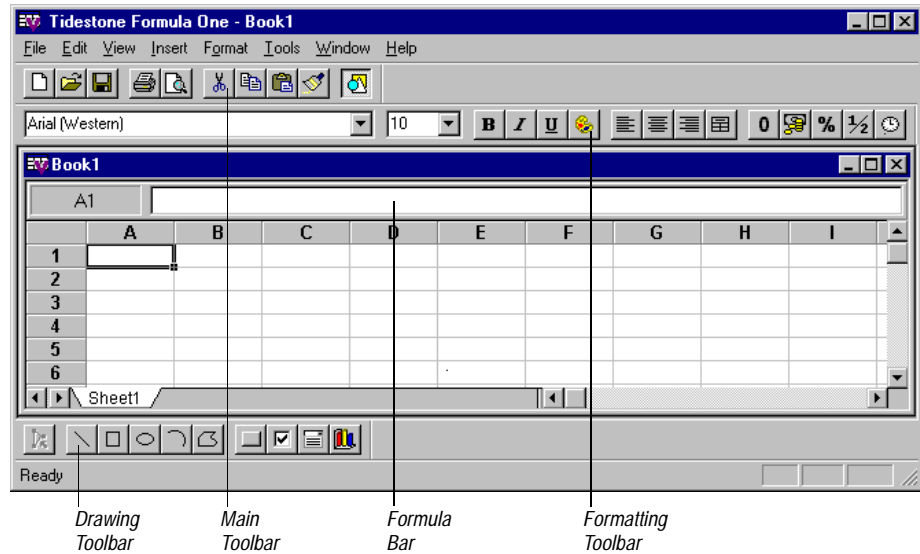
CHAPTER 3

Overview of the Workbook Designer

The Workbook Designer is an interactive program that is available at design time, runtime, or as a standalone spreadsheet application. The Workbook Designer provides access to functions for designing and formatting a workbook by pointing and clicking and choosing commands from menus. The Workbook Designer allows you to manipulate a workbook control just like it was a part of spreadsheet application. The following illustration shows the Workbook Designer as it is displayed at design time or runtime.



The following illustration shows how the Workbook Designer is displayed as a standalone spreadsheet application.



Launching the Workbook Designer

The manner in which you are using the Workbook Designer determines how you launch it. You can launch the Workbook Designer using any of the following techniques:

- **To launch the Workbook Designer during design time:**
 1. Right click on the Formula One control to display the context menu.
 2. Select Workbook Designer from the context menu.
- **To launch the Workbook Designer during runtime:**
 - Double-right click on the Formula One control and the Workbook Designer is displayed.
- **To launch the Workbook Designer as a standalone spreadsheet application:**
 - Double-click on TTF1.EXE in the Windows Explorer, or
 - Select Start > Run and type the path to TTF1.EXE.
- **To launch the Workbook Designer from your application:**
 - Use the **LaunchDesigner** method as shown in the following example:

```
F1Book1.LaunchDesigner
```

Calling this method displays the Workbook Designer, regardless of the setting of the **AllowDesigner** property.

The following sections provide additional information about the Workbook Designer menus and toolbars.

Docking the Toolbars

The toolbars in the standalone Workbook Designer are dockable. This means that you can drag them to a new location within the Workbook Designer, or make them float on top of the Workbook Designer.

► **To move the dockable toolbars:**

1. Select the toolbar you want to move.
2. Drag it to a new location.

If you drag it to an edge of the Workbook Designer, it is docked on that side of the Workbook Designer. If you drag and release the toolbar on top of the Workbook Designer, the toolbar is left floating.

Using the Workbook Designer Menus

The following tables highlight the commands available on the Workbook Designer menu bar and provide a brief description of each command.

File Menu

Command	Description
New	Creates a new file. In the built-in version of the Workbook Designer, this action deletes any information currently in the Workbook Designer. The standalone Workbook Designer allows more than one workbook open at a time.
Read Open (standalone)	Opens a workbook file from disk. Formula One can open files in the following formats: Formula One 3.0 or later (.VTS files), Excel 5.0 or later (.XLS files), and tabbed text (.TXT).
Close	Closes the Workbook Designer
Write Save (standalone) Save As (standalone)	Saves the current worksheet. Formula One can save files in the following formats: Formula One 3.x, 4.x, 5.x, or 6.x (.VTS files), Excel 5.0, 95, or 97 (.XLS files), tabbed text or values-only tabbed text (.TXT), or HTML or data-only HTML (.HTM).
Import... (standalone)	Lets you import documents in the Formula One 2.x and Excel 4 file formats. In order for this to work, the developer or user must have a copy of Formula One 5.0 with a valid license key installed on his or her machine.

Command	Description
Page Setup	Displays the Page Setup dialog box. This dialog box allows you to define header and footer text, page margins, page print order, page centering, worksheet-related print options, and scale.
Print Area	Defines the currently selected range in the active worksheet as the Print_Area user-defined name.
Print Titles	Defines the currently selected range in the active worksheet as the Print_Titles user-defined name.
Print Preview	Displays how the worksheet prints.
Print	Displays the standard Windows Print dialog box. This dialog box allows you to select print options, and print the active worksheet.
[recent files] (standalone)	Displays a list of the last few workbooks that Formula One has opened or created and allows the user or developer to open one or more of them.
Exit (standalone)	Exits the standalone Workbook Designer.

Edit Menu

Command	Description
Cut	Cuts the current worksheet selection to the clipboard.
Copy	Copies the current worksheet selection to the clipboard.
Paste	Pastes the contents of the clipboard to the current worksheet selection.
Paste Special	Pastes the formats, values, or formulas of copied cells into selected cells. In addition, Paste Special controls how data copied from a different application is pasted.
Copy Cell Format	Copies the current cell formatting so you can apply it to another cell. Refer to the FormatPaintMode property in the Formula One on-line help for additional information.
Polygon Points	Toggles between normal polygon editing and polygon point editing. When polygon point editing is enabled, this command is checked.
Select All Objects	Selects all of the graphical objects on the active worksheet.
Sort	Displays the Sort dialog box. This dialog box allows you to set the sorting method and sort keys for data sorting.
Fill	Data in the top or leftmost cell is copied down or to the right to fill the range.
Clear > All	Clears formats and values from the selected cells.
Clear > Formats	Clears only formats from the selected cells.
Clear > Contents	Clears only values from the selected cells.

Command	Description
Delete	Deletes the current selection or selected objects. Cells adjacent to the deleted cells are shifted to fill the space left by the vacated cells.
Delete Sheet	Removes the selected worksheets, and shifts the worksheets to the right of the deleted worksheets to the left.
Find	Searches in selected cells or sheets for the characters that you specify. It selects the first cell that contains those characters.
Replace	Searches in selected cells or sheets for the characters that you specify, and replaces them with your specified replacement characters.
Goto	Displays the Goto dialog box. This dialog box allows you to specify a cell to display in the worksheet window. The specified cell is made the active cell.

View Menu

Command	Description
Toolbar > Standard	Toggles the display of the Main Toolbar.
Toolbar > Drawing and Forms	Toggles the display of the Drawing Toolbar.
Formula Bar	Toggles the display of the Formula Bar.
Status Bar	Toggles the display of the Status Bar.

Insert Menu

Command	Description
Cells	Inserts cells at the location of the current selection. Cells adjacent to the insertion are shifted to make room for the new cells.
Rows	Inserts a new row above the selected cell or row.
Columns	Inserts a new column to the left of the selected cell or column.
Worksheet	Inserts a new worksheet to the left of the selected worksheet. If more than one worksheets are selected, this command inserts the same number of worksheets that are selected. This command fails if non-contiguous sheets are selected.
Chart	Inserts a First Impression chart on the active worksheet.

Command	Description
Page Break	Places a horizontal page break adjacent to the top edge of the active cell and a vertical page break adjacent to the left edge of the active cell. If a row or column is selected, a page break is placed adjacent to the selected row or column.
Name	Displays the Define Name dialog box. This dialog box allows you to add and delete user-defined names.
Drawing Object > Arc	Selects the Arc tool which allows you to draw arcs.
Drawing Object > Line	Selects the Line tool which allows you to draw lines.
Drawing Object > Oval	Selects the Oval tool which allows you to draw ovals.
Drawing Object > Polygon	Selects the Polygon tool which allows you to draw polygons.
Drawing Object > Rectangle	Selects the Rectangle tool which allows you to draw rectangles.
Forms Object > Button	Selects the Button tool which allows you to draw buttons.
Forms Object > Checkbox	Selects the Check Box tool which allows you to draw check boxes.
Forms Object > Dropdown Listbox	Selects the Drop Down List Box tool which allows you to draw drop-down list boxes.
Cancel Insert Object	Allows you to unselect a graphical object tool.

Format Menu

Command	Description
Cells	Displays the Format Cells dialog box which allows you to set cell formatting such as numeric display, alignment, fonts, borders, patterns, protection, and validation.
Row > Height	Displays the Row Height dialog box. This dialog box allows you to set the height of the selected rows, specify default row heights, and specify automatic row height. In addition, you can specify whether the selected rows are shown or hidden.
Row > Hide	Hides the selected rows, which does not delete them from the worksheet.
Row > Unhide	Shows the hidden rows in a selection.
Row > Default Height	Displays the Row Height dialog box which allows you to define a default height of rows.
Column > Width	Displays the Column Width dialog box. This dialog box allows you to set the width of the selected columns, specify default column widths, and specify automatic column width. In addition, you can specify whether the selected columns are shown or hidden.
Column > Autofit Selection	Automatically adjusts the width of all cells in the column to accommodate the size of a text string or value.
Column > Hide	Hides the selected columns, which does not delete them from the worksheet.

Command	Description
Column > UnHide	Shows the hidden columns in a selection.
Column > Default Width	Displays the Column Width dialog box which allows you to define a default width of columns.
Sheet > Properties	Displays the Format Sheet dialog box which allows you to set properties for the active worksheet.
Sheet > Protection	Enables protection for protected cells in the worksheet. A check next to this command means that protection is enabled. Select the command again to disable protection.
Freeze Panes	Freezes the selected columns or rows. Frozen columns and rows do not scroll and cannot be edited.
Unfreeze Panes	Unfreezes frozen panes.
Default Font	Displays the Default Font dialog box. This dialog box allows you to set the default font used to display data in worksheets. In addition to setting the font and font size used to display data in a worksheet, the default font affects the widths of worksheet columns. Column widths can be set in units equal to 1/256th of the character 0 (zero) in the default font.
Object	Displays the Format Object dialog box, and includes the appropriate tabbed pages for the selected object.
Bring to Front	Places the selected objects in front of other objects in the worksheet.
Send to Back	Places the selected objects behind other objects in the worksheet.

Tools Menu

Command	Description
Recalc	Recalculates all open cells, worksheets, and workbooks.
Options	Accesses the Options dialog box. The Options dialog box includes tabbed pages for setting general, calculation, and color options.











A standard Window menu is provided on the standalone version of the Workbook Designer.

Using the Workbook Designer Toolbars

The buttons on Formula One's toolbars provide easy access to some of the most common Formula One menu items. Formula One has three toolbars: Standard, Formatting, and Drawing and Forms. You can display or hide the toolbars by toggling options in the View > Toolbars menu.

Standard Toolbar











Use the buttons on the Standard Toolbar to perform basic workbook functions such as opening, saving, and printing.




Button	Name	Description
	New	Creates a new file.
	Read	Opens an existing file.
	Save	Saves a file.
	Print	Prints the active worksheet.
	Print Preview	Displays the active worksheet in Print Preview mode.
	Cut	Cuts a selection to the clipboard.
	Copy	Copies a selection to the clipboard.
	Paste	Pastes from the clipboard.
	Copy Format	Copies the format of the selected cells.
	Toggle Drawing Toolbar	Toggles the display of the Drawing Toolbar.

Formatting Toolbar

Use the buttons on the Formatting Toolbar to quickly and easily format the data in the selected cell(s). This toolbar is available only with the standalone Workbook Designer.

The Format Toolbar also provides drop-down lists that allow you to select fonts and font sizes.

Button	Name	Description
	Bold	Controls the bold attribute for the currently selected range of data.
	Italic	Controls the italic attribute for the currently selected range of data.
	Underline	Controls the underline attribute for the currently selected range of data.
	Color	Selects a color for the currently selected range of data. The color applies to the numbers and letters in the cell, not to the background or borders of the cell.
	Left Align	Left aligns the selected range of data.
	Center	Centers the selected range of data.
	Right Align	Right aligns the selected range of data.
	Merge and Center	Merges the selected cells and centers the selected data. This is useful for straddling heading information across rows or columns.
	Common Fixed and General Formats	Displays a list of common fixed and general formats. Select a format to apply it to the selected range of data.
	Currency	Displays a list of common currency formats. Select a format to apply it to the selected range of data.

Button	Name	Description
	Percent	Displays a list of common percent formats. Select a format to apply it to the selected range of data.
	Fraction	Displays a list of common fraction formats. Select a format to apply it to the selected range of data.
	Date and Time	Displays a list of common date and time formats. Select a format to apply it to the selected range of data.

Refer to “Interactively Drawing Graphical Objects” on page 124 in this manual for information about the icons on the Drawing Toolbar.

C H A P T E R 4

Workbook Fundamentals

Before you can successfully use a Formula One control, you must understand some basic concepts about the workbook. You must understand how to select worksheets, cells, ranges, rows, and columns, enter and delete data, and display specific sections of a workbook.

This chapter discusses:

- setting up workbook defaults
- adding, inserting, deleting, naming, selecting, and displaying worksheets
- navigating through a sheet with keyboard commands and with mouse actions
- selecting cells and ranges
- selecting entire rows and columns
- setting selection options

Setting up Workbooks

Formula One gives you options for customizing the display and the defaults used in the Workbook Designer. In most cases, you may use the preset defaults, but you may change them if you want using the options described below.

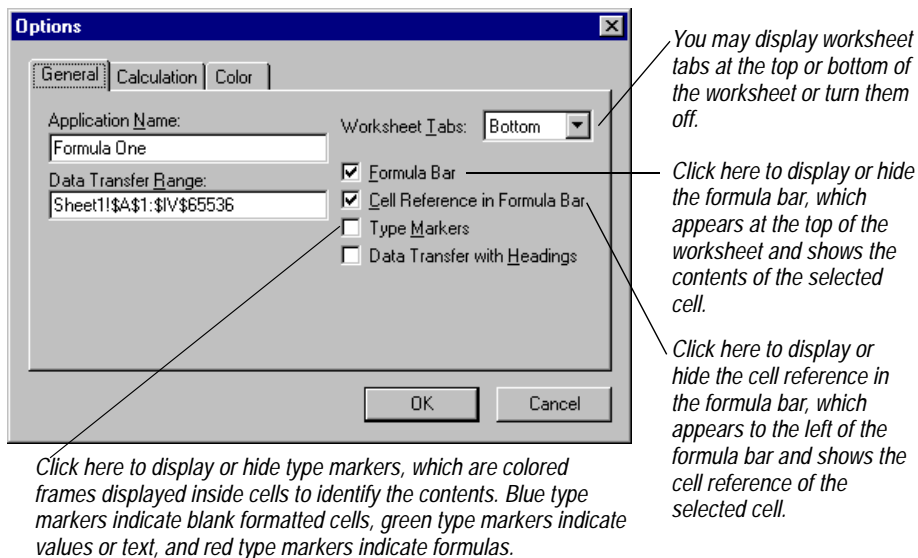
Displaying Parts of the Workbook Designer

You may choose to change whether and how the Workbook Designer displays certain things in a workbook. These options affect all worksheets in the workbook.

More display options are available per individual worksheets. To change how to display different characteristics of individual worksheets, see “Setting Display Options for Worksheets” on page 55.

➤ **To change workbook display options in the Workbook Designer:**

1. Choose Tools > Options and click the General tab, shown below.



2. Choose the options you want, as described above, and click OK.

➤ **To change workbook display options programmatically:**

- Use the **ShowTabs**, **ShowEditBar**, **ShowEditBarCellRef**, and **ShowTypeMarkers** properties.

Note You can hide the worksheet tabs in order to limit users to a single worksheet in a workbook that contains multiple sheets. Use the **Sheet** property to make the sheet you want users to access the active sheet.

Setting the Default Font

You can set a default font, font size, and style that will apply to all worksheets in the workbook. Later you can change these settings for individual cells in the Font tab of the Format Cells dialog box. Changes you make in the Format Cells dialog box will remain, even if you change the default settings.

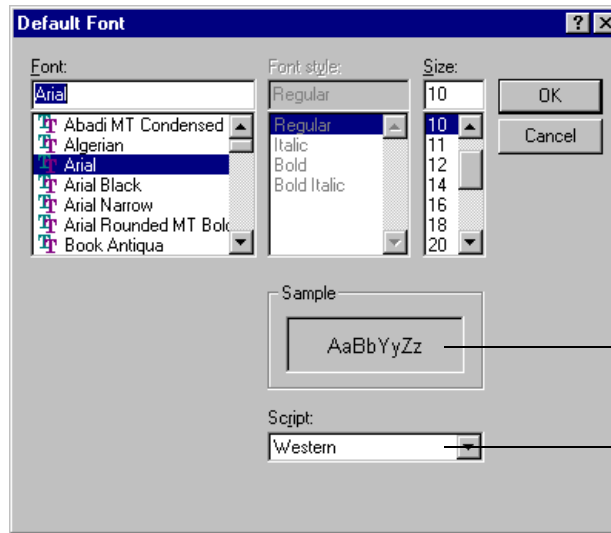
The default font effects the widths of worksheet columns. Column widths are set in units equal to 1/256th of the character 0 (zero) in the default font, or twips, depending on the setting of the **ColWidthUnits** property.

Because the basic unit for measuring columns can change when you change the default font, you may need to adjust the widths of columns (including the row header column) after setting the default font.

Note By default, Formula One uses Arial as the default font. Be sure you always use a TrueType font as the default font in order for print and display scaling to work correctly.

➤ **To set the default font using the Workbook Designer:**

1. Select Format > Default Font. The system will display the Default Font dialog box, shown here.



A text sample with the font, style, and size you chose appears here.

Some fonts provide scripts that allow you to use non-Western alphabets. Choose the script you want here.

2. Choose the default font options you want. Click OK when you finish.

➤ **To set the default font programmatically:**

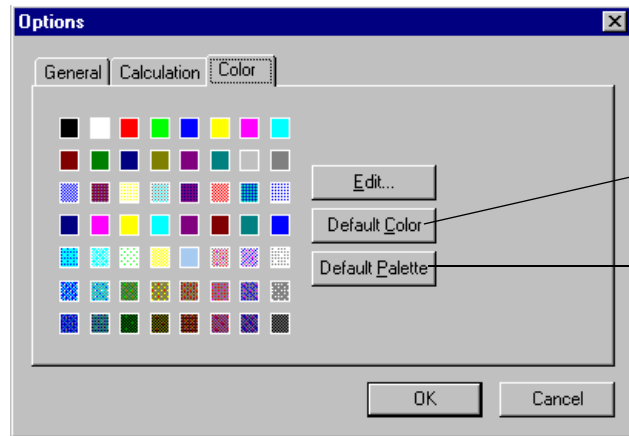
- Use the **SetDefaultFont** method to set basic defaults. Use the **SetDefaultFontEx** method to set basic defaults and specify the character set.

Setting Up the Color Palette

Formula One comes with a predefined color palette of 56 colors. You may use those colors or you may define new colors. After you define new colors you may change them back to the preset colors.

➤ **To change colors in the color palette using the Workbook Designer:**

1. Choose Tools > Options, click the Color tab, and the Color tab will appear, as shown below.



Click here to return the selected color to its default value.

Click here to return all the palette colors to their default values.

2. Double-click on the color you want to change. The Windows color editor will appear, allowing you to change the settings for the selected color.

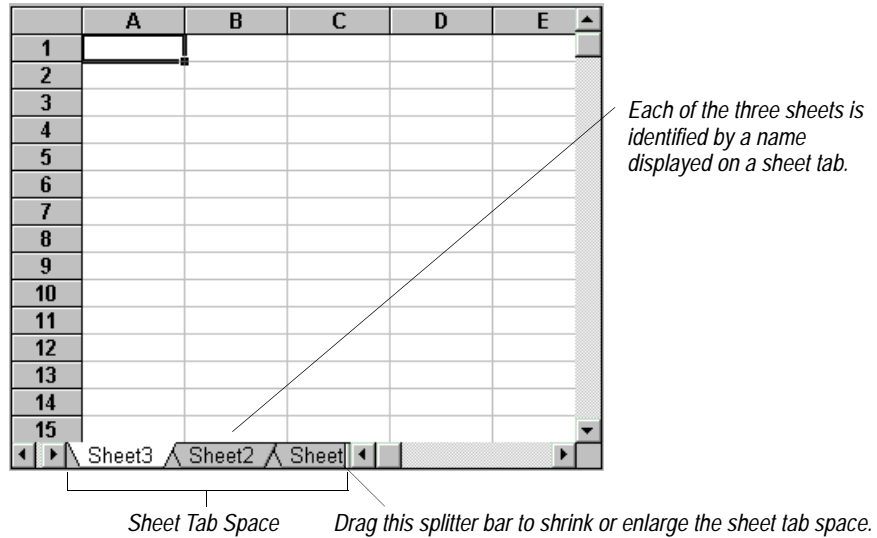
➤ **To change colors in the color palette programmatically:**

- Use the **PaletteEntry** property.

Manipulating Worksheets

Once you create a Formula One workbook, you should understand how to add, insert, delete, name, and select the worksheets that are contained in your workbook. For additional information about workbooks and worksheets, refer to “Understanding Workbooks and Worksheets” on page 24.

The following illustration shows a Formula One workbook with three worksheets:



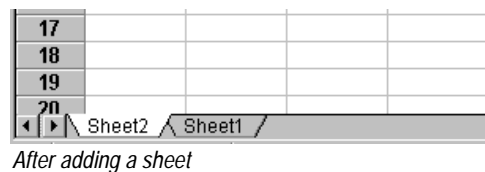
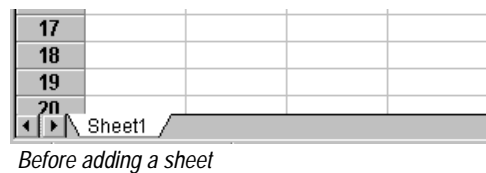
Inserting Worksheets

By default, a workbook contains only one worksheet. You can easily insert additional worksheets via the Workbook Designer or program code.

► To add worksheets using the Workbook Designer:

1. Right-click on the Workbook control to display the context menu.
2. Select Workbook Designer.
3. Select Insert > Worksheet.

One new worksheet is inserted to the left of the selected worksheet as shown in the following illustration:



Sheet Index List

Each workbook maintains an indexed list of the worksheets it contains. Worksheets are indexed from left to right beginning with 1. As you add worksheets, Formula One adjusts the index numbers so the leftmost worksheet is always index 1. Most methods and properties reference worksheets by index rather than name. It is important to remember that the sheet index is different from the name that appears on the sheet tab.

Notice in the previous illustration that the worksheet is given the next available name "Sheet2." However, Sheet2 is index 1 and Sheet1 is index 2.

Selecting Worksheets

Usually, you do most of your work in one worksheet at a time. This is called the *active worksheet*. When you have multiple worksheets in a workbook, you can use the mouse to click on a worksheet's tab to make it the active sheet. The tab is highlighted and moves on top of the other tabs.

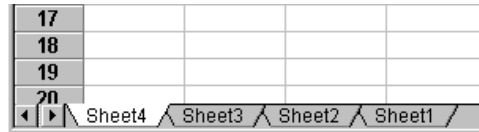
You can save time and effort by performing some tasks on several sheets at once. For example, if you want all three worksheets in your workbook to have the same title information, you can select all three worksheets and enter the titles on the active worksheet. The titles are automatically entered in the corresponding cells in the other selected worksheets as well.

➤ To select multiple worksheets in the Workbook Designer:

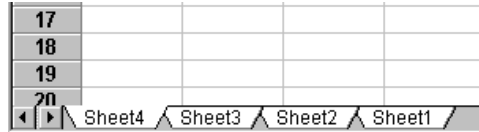
1. Use one of the following key/mouse combinations, depending on whether you want to select adjacent or non-adjacent worksheets:

Action	Result
CTRL-Click on sheet tab	Selects or deselects non-adjacent sheets. Any other selected worksheets remain selected.
SHIFT-Click on sheet tab	Selects all adjacent worksheets between the active worksheet and the worksheet you clicked on. All other worksheets are deselected.

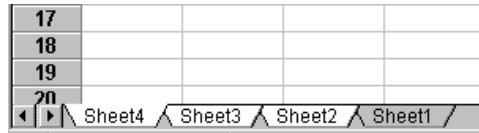
The following illustration shows various groupings of selected worksheets:



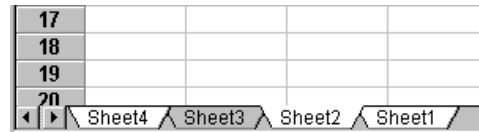
Sheet4 is the active sheet. All other sheets are deselected.



If you hold down the Shift key and select Sheet1, all sheets between the active sheet (Sheet4) and Sheet1 are selected.



If you select all but one sheet, and then make one of the selected sheets active by selecting it, all the other sheets remain selected.



If you hold down the Control key and select Sheet3, it is deselected, but the other sheets remain selected.

► To select multiple sheets programmatically:

- Use the **SheetSelected** property to toggle an individual worksheet's selection status to on.

The following example selects the second and third worksheets in the workbook:

```
SheetSelected (2) = True
SheetSelected (3) = True
```

Working with a Group of Worksheets

When multiple worksheets are selected, you can think of them as a group of worksheets. When you perform some actions, execute some methods, or refer to some properties, they affect all the selected worksheets. Other actions affect only the active worksheet, regardless of how many worksheets are selected.

In the Workbook Designer, the following actions work on all selected worksheets:

- changing cell selection
- entering values via the formula bar
- inserting rows, columns, or ranges of cells
- deleting rows, columns, or ranges of cells
- clearing rows, columns, or ranges of cells
- setting TopLeft/Row/Column header text
- setting column width

- setting row height
- moving and copying with the mouse

Within your application code, the following methods and properties affect all selected worksheets:

Methods that affect all selected worksheets

ClearRange	ColWidthDlg	CopyRange
DefColWidthDlg	DefRowHeightDlg	DeleteDlg
DeleteRange	EditClear	EditDelete
EditInsert	FilePageSetupDlg	FilePageSetupDlgEx
FormatAlignmentDlg	FormatBorderDlg	FormatCellsDlg
FormatFontDlg	FormatNumberDlg	FormatPatternDlg
FormatSheetDlg	InsertDlg	InsertRange
MoveRange	ProtectionDlg	RowHeightDlg
SetAlignment	SetBorder	SetBorderEx
SetCellFormat	SetColWidth	SetFont
SetFontEx	SetPageSetup	SetPattern
SetProtection	SetRowHeight	SetValidationRule
ValidationRuleDlg		

Properties that affect all selected worksheets

ColText	ColWidth	EnableProtection
Entry	EntryRC	EntrySRC
Formula	FormulaLocal	FormulaLocalRC
FormulaLocalSRC	FormulaRC	FormulaSRC
HAlign	HdrWidth	HdrHeight
Logical	LogicalRC	LogicalSRC
Number	NumberFormat	NumberFormatLocal
NumberRC	NumberSRC	ProtectionHidden
ProtectionLocked	RowText	RowHeight
Text	TextRC	TextSRC
TopLeftText	VAlign	WordWrap

Inserting Multiple Worksheets

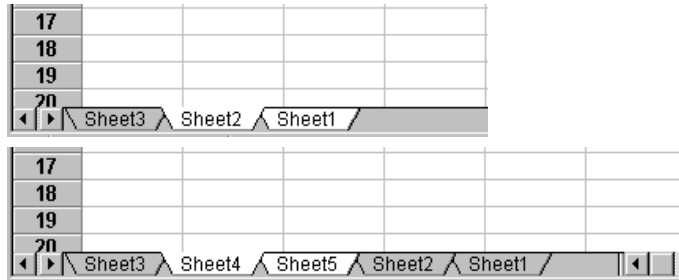
You can insert more than one sheet at a time and at any point in the sheet tab index. The number and position of the inserted sheets depends on number and position of the selected sheets in the workbook.

➤ **To insert worksheets using the Workbook Designer:**

1. Select the worksheet immediately to the right of where you want to insert the new worksheets.
2. Select as many worksheets to the right of that worksheet as the number of worksheets you want to insert.

For example, to insert two worksheets, select two worksheets.

3. Select Insert > Worksheet. The following illustration shows this process:



Since Sheet2 and Sheet1 are selected, two additional worksheets are inserted to the left of Sheet2. Notice that the newly inserted worksheets are given the next available sheet names, regardless of their position in the sheet index list.

➤ **To insert sheets programmatically:**

- Use the **NumSheets** property to increase the number of worksheets in the workbook. Additional worksheets are added to the right of all existing worksheets.
- Use the **InsertSheets** method to add worksheets to the left of a specified worksheet. The following example illustrates how this method can be used to insert 2 worksheets to the left of the third worksheet in the workbook:

```
F1Book1.InsertSheets 3, 2
```

- Use the **SheetSelected** property to select the number of worksheets you want and then use the **EditInsertSheets** method. The following example selects the third and fourth worksheets in the workbook and inserts two worksheets before the third worksheet.

```
F1Book1.SheetSelected (3)
F1Book1.SheetSelected (4)
F1Book1.EditInsertSheets
```

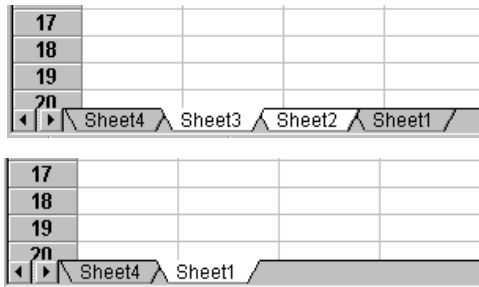
Deleting Worksheets

You can delete one or more worksheets from the sheet index list through the Workbook Designer or through application code.

► To delete worksheets in the Workbook Designer:

1. Select the worksheets you want to delete.
2. Select Edit > Delete Sheet.

The following illustration shows this process:



The selected worksheets are deleted.

► To delete worksheets programmatically:

Two methods delete worksheets: **DeleteSheets** and **EditDeleteSheets**. **DeleteSheets** takes arguments that define the position and number of worksheets to be deleted. **EditDeleteSheets** deletes the currently selected worksheets.

- Use the **NumSheets** property to decrease the total number of worksheets. Worksheets are deleted from the right.
- Use the **DeleteSheets** method to delete specific worksheets. The following example uses this method to delete the third and fourth worksheets from the workbook:

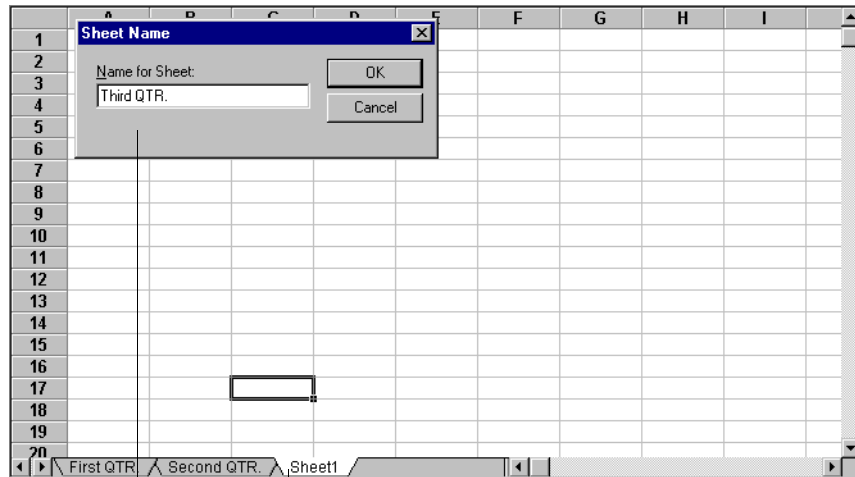
```
F1Book1.DeleteSheets 3, 2
```

- Use the **SheetSelected** property to select sheets and then use the **EditDeleteSheets** method. The following example selects the first and fourth worksheets and then deletes them:

```
F1Book1.SheetSelected (1)
F1Book1.SheetSelected (4)
F1Book1.EditDeleteSheets
```

Renaming Worksheets

Formula One provides a default name for each worksheet. You can change the names to more meaningfully describe the sheets' contents. For example, the sheet names in the following illustration are more descriptive than the worksheets' default names.



*Type a new name in the
Sheet Name dialog box.*

*Double-click the sheet tab to display
the Sheet Name dialog box.*

► To name a worksheet in the Workbook Designer:

1. Select Format > Sheet > Properties and select the General tab.
2. Type a name for the sheet in the Name text box.
3. Click OK.

► To edit a sheet name programmatically:

- Use the **SheetName** property to rename a worksheet identified by index. The following code changes the name of the second worksheet in the workbook to QTR 2 Sales.

```
F1Book1.SheetName (2) = "QTR 2 Sales"
```

Setting Display Options for Worksheets

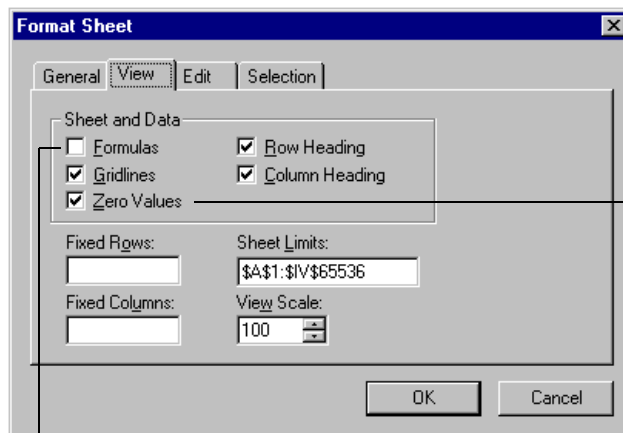
Just as you can display or hide different parts of the Workbook Designer, you can display or hide different parts of worksheets. You may hide the column and row headings and the gridlines.

You may display the worksheet at regular size, shrunk down, or enlarged. You may also choose how many rows and columns you want to display. The default number of rows and columns displayed is the maximum (65,536 rows by 256 columns). You may display fewer. Rows and columns beyond the display limits you specify will not be seen, but they can hold data and formulas.

Finally, you may specify how you want to display formulas and zero values.

➤ **To change display options for worksheets in the Workbook Designer:**

1. Select the worksheet(s) whose display settings you want to change.
2. Select Format > Sheet > Properties and click the View tab. The View tab will appear, as shown below.



To display zeroes in cells, select this option. To leave cells with zero values empty, unselect this option.

To display formulas, select this option. To display values that the formulas calculate, uncheck this box. When formulas are displayed, Formula One automatically doubles the width of columns to accommodate the wider text. When the Formulas option is later unselected, column widths return to their original settings.

3. Select the worksheet and data viewing options you want by selecting or unselecting the options in the Sheet and Data values.
4. To display the worksheet larger or smaller than its actual size, enter a value from 10 to 400 in the View Scale box.
5. To display fewer than the maximum number of rows and columns, enter the range that you want to display, in absolute references, in the Sheet Limits box.
6. Click OK.

➤ **To change display options for worksheets programmatically:**

- Use the **ShowFormulas**, **ShowGridLines**, **ShowZeroValues**, **ShowRowHeading**, **ShowColHeading**, **ViewScale**, **MaxRow**, and **MaxCol** properties.

Navigating Through Worksheets

When working in the Workbook Designer or in a workbook at run time, you can navigate within individual worksheets using mouse actions or keyboard commands.

Navigating with the Mouse

Primarily the mouse is used to select items in a worksheet at run time. The following table lists the mouse actions you can perform in a worksheet at run time or in the Workbook Designer.

Action	Description
Left-Click	Moves the active cell to the pointer position.
Right-Click	In the container's design mode, brings up the context menu.
Wheel-click (on the IntelliPoint mouse) or middle click	Invokes or cancels scrolling mode, which allows users to scroll by moving the mouse over the worksheet. To disallow scrolling mode, hide the scroll bars using the ShowHScrollBar and ShowVScrollBar properties.
Left-Click in Row or Column Headings	Selects entire row or column.
Left-Click in Top Left Corner	Selects entire sheet.
Left Double-Click in Top Left Corner, Row Headings, Column Headings, or Worksheet tabs	Displays a dialog box that allows you to enter a label for the top left corner or the column or row heading, or a new name for the worksheet that was double clicked.
Left Double-Click	In the Workbook Designer, invokes in-cell editing. At run time, a DblClick event is fired.
Right Double-Click	In the Workbook Designer, does nothing. At run time, the Workbook Designer is launched if DoRdblClick is False.
Left-Click and Drag	Selects a range. If other ranges are selected, the previously selected ranges are unselected.
CTRL + Left-Click and Drag	Selects a range. If other ranges are selected they remain selected.
SHIFT + Left-Click and Drag	Extends the current selection.
CTRL + SHIFT Click on Row Headings, Column Headings, or Top Left Corner	Selects the row headings, column headings, or top left corner of the sheet.
Drag a Selection's Copy Handle	Copies the selection into the newly selected area.
Drag a Selection's Border	Moves the selection to a new location.
ALT + Click and Drag an Object or Object's Selection Handles	Repositions or resizes an object and aligns object sides with the cell grid.

Navigating with the Keyboard

In addition to navigating through worksheets, keyboard commands allow you to perform a variety of other tasks.

Keyboard commands allow you to:

- position the active cell in the worksheet
- page through a worksheet
- enter data typed in a cell
- move the active cell within a selected range
- enter and exit edit mode
- recalculate a workbook
- delete data from a selected cell or range

The tables in this section list the keyboard commands you can use when working in the Workbook Designer or a workbook at run time.

The following table lists action keys that allow you to enter and edit data, move the active cell within a selected range, and recalculate the workbook.

Key	Description
ENTER	When in edit mode, accepts the current entry. When a range is selected, and if the EnterMovesDown property is set to True, accepts the current entry and moves active cell vertically to next cell in selection.
SHIFT + ENTER	When in edit mode, accepts the current entry. When a range is selected, and if the EnterMovesDown property is set to True, accepts the current entry and moves active cell vertically to previous cell in selection.
TAB	When in edit mode, accepts the current entry and moves the active cell horizontally to right.
SHIFT + TAB	When in edit mode, accepts the current entry and moves the active cell horizontally to left.
F2	Enters edit mode. While in editing mode, F2 displays the Cell Text dialog box, in which you can enter multi-line data entries.
F9	Recalculates workbook.
DEL	May clear current selection depending on the setting of the AllowDelete property.
Escape	Cancels current data entry or editing operation.

The following table lists the movement keys that allow you to move the active cell within a worksheet and display different sections of the workbook.

Key	Description
Up Arrow	Moves active cell up one row.
Down Arrow	Moves active cell down one row.
Left Arrow	Moves active cell left one column.

Key	Description
Right Arrow	Moves active cell right one column.
CTRL Up/Down/Left/Right	Moves to the next range of cells containing data. If there is no additional data in the direction in which you are moving, moves to the edge of the worksheet.
Page Up	Moves up one screen.
Page Down	Moves down one screen.
CTRL Page Up	Activates the previous worksheet in the current workbook.
CTRL Page Down	Activates the next worksheet in the current workbook.
Home	Goes to first column of current row.
End	Goes to last column of current row that contains data.
CTRL Home	Goes to row 1 column 1.
CTRL End	Goes to last row and column that contains data.

The following table lists the keys that modify the action of the movement keys.

Key	Description
Scroll lock	Causes the view window to scroll without changing current selection with all movement keys except Home, End, CTRL Home, and CTRL End.
SHIFT plus any movement key	Extends the current selection.

Selecting Cells

Many operations require one or more cells to be selected. There are three kinds of selections: a single cell, a range of cells, and multiple ranges of cells (non-adjacent). The following illustration shows the three types of selections.

	A	B	C	D	E
1					
2					
3					
4					
5					
6					
7					
8					

Single cell selection

	A	B	C	D	E
1					
2					
3					
4					
5					
6					
7					
8					

Single range selection

	A	B	C	D	E
1					
2					
3					
4					
5					
6					
7					
8					

Multiple range selection

Selecting Cells with the Mouse

The worksheet cursor is always located on a cell. The cell on which the worksheet cursor is located is called the *active* cell. The active cell is also a selection or part of a selection. Any data the user enters is always placed in the active cell.

- To select a range of cells, click and hold the left mouse button and drag through the range you want to select. When a range is selected, it becomes highlighted.
- To select multiple ranges, press the CTRL key while selecting a range with the mouse. Any previously selected ranges remain selected.

Once a range is selected, you can move the active cell within the range using the ENTER, SHIFT + ENTER, TAB, and SHIFT + TAB keys. When you use these keys to move the active cell, the range remains selected.

Selecting Cells with Properties and Methods

The following properties and methods can select ranges.

- Setting the **Selection** property removes all current selections and selects a range.
- The **AddSelection** method adds a selection to the current selection list. Continue calling **AddSelection** to create multiple selections.

The following example selects two ranges. The range “my_select” is selected, which is the name for A1:D4, and then the E5:H8 range is added.

```
F1Book1.Selection = "my_select" 'Select A1:D4  
F1Book1.AddSelection 5, 5, 8, 8 'add e5:h8
```

In addition, the following properties retrieve information about multiple selections.

- The value of the **SelectionCount** property tells you the number of selections. You can use this if a selection is made by the user and you need to determine how many ranges are selected.
- The value of the **Selection** property gives you all current selections in the form of a formula (e.g. A1:D4,E5:H8).
- The **SelectionEx** property used with the **F1RangeRef** API object returns the row and column of the specified selection.

The following example returns the value of a specified selection and adds another column to the selection.

```
Dim sel As F1RangeRef  
Dim cnt As Integer  
For cnt = 0 To (F1Book1.SelectionCount - 1)  
Set sel = F1Book1.SelectionEx(cnt)  
F1Book1.AddSelection sel.StartRow, sel.StartCol, sel.EndRow,  
sel.EndCol + 1  
Next cnt  
End Sub
```

Selecting Rows and Columns

Entire rows and columns can be selected in the worksheet at run time or in the Workbook Designer using the mouse. To select a row or column, position the pointer on the header of the row or column you want to select. When you click the header, the row or column is selected.

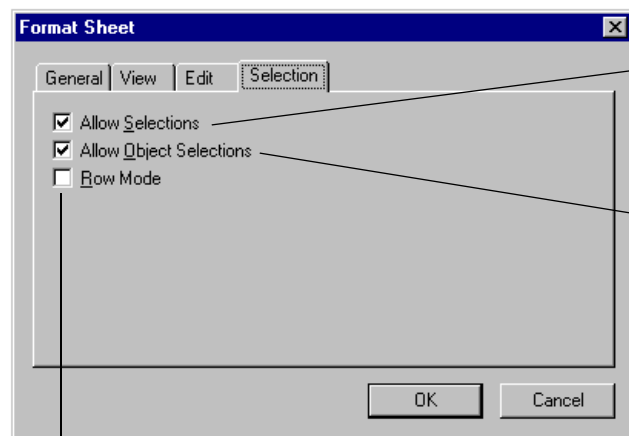
You can also select all rows and columns in the worksheet. To do this, position the pointer on the top left header and click.

Setting Selection Options

You may prevent users of your worksheet from selecting cells and/or objects. You may also choose to display the worksheet in row mode, which specifies that an entire row be selected when a user selects a cell in that row. Row mode is a nice feature in applications where each row represents a particular record: it makes it obvious which row the user is currently working on and facilitates copying and pasting entire rows.

► **To set selection options in the Workbook Designer:**

1. Select the worksheet(s) whose selection options you want to change.
2. Select Format > Sheet > Properties and click the Selection tab. The Selection tab, displayed below, will appear.



To prevent users from selecting cells, rows, columns, graphical objects, or worksheets, uncheck this option.

To prevent users from selecting graphical objects, uncheck this option.

To specify that the entire row should be selected whenever an individual cell in that row is selected, check this option.

3. Choose the selection options you want. Click OK when you are done.

► **To set selection options programmatically:**

- Use the **AllowSelections**, **AllowObjSelections**, and **RowMode** properties.

C H A P T E R 5

Working With Data

Entering and manipulating data is the basis for nearly all work performed in a workbook control. You can enter virtually any type of data and formula. With formulas and built-in functions, you can evaluate and calculate that data and make decisions based on the results of those operations.

This chapter discusses:

- how to enter data directly or with methods and properties
- how to use autofill lists
- the types of constant values that can be entered
- how to construct and use formulas
- the suite of built-in worksheet functions
- using names
- the methods for calculating worksheets
- how to limit user data entry

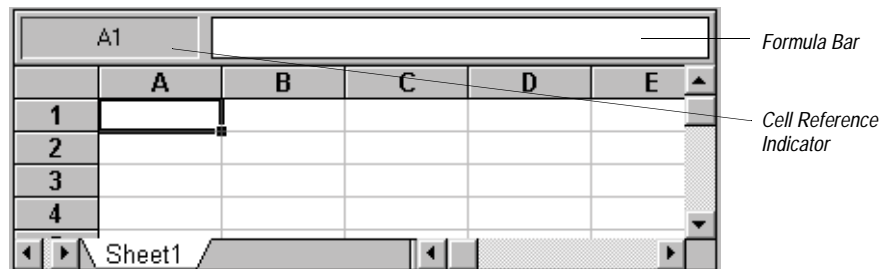
Understanding Worksheet Data Entry

One of the basic tasks encountered when working with a workbook is data entry. Formula One provides several methods for entering data.

- **Direct entry.** This is the most direct method of data entry. Data can be entered directly in a worksheet at run time. Or, you can enter data in the Workbook Designer at design time.
- **Properties and methods.** Several properties and methods allow you to enter data in the active cell or a specified cell.

Adding the Formula Bar

You can enter data into a worksheet by typing directly into a cell, or by typing into the formula bar.



The formula bar appears by default in the Workbook Designer, and you can turn on or off the display of the formula bar and cell reference indicator in your workbook control.

- **To control the display of the formula bar in the Workbook designer:**
 - Select View > Formula Bar to toggle the display of the formula bar.
- **To add a formula bar to your control:**
 - Set the **ShowEditBar** property to True.
 - If the **ShowEditBar** property is True, you can also set the **ShowEditBarCellRef** to True to display the cell reference indicator with the formula bar.

Important The formula bar does not appear on the control unless the container makes it UI Active (provides a window for it).

Entering Data with Properties

Formula One provides a full complement of properties for entering data. In addition to entering data in the active cell, a number of methods allow you to enter data in a cell you specify.

The following table lists the properties involved in entering data.

Property/ Method	Description
Entry	Sets or returns the value of the current cell in edit mode format.
EntryRC	Sets or returns the value of the specified cell in edit mode format.
EntrySRC	Sets or returns the value of the specified cell in edit mode format.
FormattedText	Sets or returns the value of the current cell as it appears in the worksheet.
FormattedTextRC	Returns the value of the specified cell as it appears in the worksheet.

Property/ Method	Description
FormattedTextSRC	Sets or returns the value of the specified cell as it appears in the worksheet.
Formula	Sets or returns the formula in the active cell.
FormulaLocal	Sets or returns the text version of the formula in the active cell, in the user's language.
FormulaLocalRC	Sets or returns the text version of the formula of the specified cell, in the user's language.
FormulaLocal SRC	Sets or returns the text version of the formula of the specified cell, in the user's language.
FormulaRC	Sets or returns the formula in the specified cell.
Logical	Sets or returns the formula in the active cell.
LogicalRC	Sets or returns the logical value of the specified cell.
Logical SRC	Sets or returns the logical (True or False) value of the specified cell.
Number	Sets or returns the numeric value of the active cell.
NumberRC	Sets or returns the numeric value of the specified cell.
Number SRC	Sets or returns the numeric value of the specified cell.
Text	Sets or returns the text value of the active cell.
TextRC	Sets or returns the text in the specified cell.
TextSRC	Sets or returns the text value of the specified cell.

Entering Multi-Line Data

A single cell can contain as many as nine lines or 256 characters of data. When entering the data, new lines of data are specified within the cell by entering carriage return/line feeds.

- When entering data interactively in the Workbook Designer or in a worksheet at run time, press F2 when editing a cell. The Cell Text dialog box is displayed in which you can enter the cell data. To enter a line feed, press RETURN. Click the OK button to accept the entry and return to normal worksheet editing.
- When entering data with properties or methods, you should specify line feeds with the ANSI character code 10.

After entering multi-line data, you may need to resize the row and column in which the entry is placed in order to view all the data. The following example uses the **TextRC** property to enter three lines of data in cell A1.

```
F1Book1.TextRC(1,1) = "Regional Sales" & Chr$(10) & "FY '94" &
Chr$(10) & "Q2"
```

The following illustration shows the results of this example.

	A	B	C
1	Regional Sales FY '94 Q2		
2			
3			
4			
5			

*The multi-line entry is placed in cell A1.
Row 1 and column A have been resized so
the entry is not cropped.*

You can also enter multi-line row and column headers. Refer to Formatting Row and Column Headings in Chapter 7 for information about entering and formatting row and column headers.

Understanding Worksheet Data Types

Cells can contain two types of information: constant values and formulas.

- Constant values are numbers (including dates and times), logical values, error values, and text.
- Formulas are groups of constant values, cell references, names, functions, and operators that result in a new value when calculated or evaluated.

Entering Constant Values

Numbers. Numeric entries can contain numeric characters (e.g., 1, 2, 3, 4, 5, 6, 7, 8, 9, and 0) and the special characters (e.g., +, -, (,), /, \$, %, ., E, and e).

- Negative numbers can be preceded by a minus sign or enclosed in parentheses.
- Commas can be included in numeric entries as thousands separators.
- Numeric entries containing leading dollar signs (or other currency signs, such as £ for pounds) are formatted as currency.
- Numeric entries containing trailing percent signs are formatted as percentages.

Formula One accepts numeric entries as fractions. If the fraction contains a leading integer (e.g., 1 1/3) it can be entered directly. If there is no leading integer, the fraction should be preceded by a zero (e.g., 0 2/3).

Numbers larger than the cell in which they are entered are converted to scientific notation unless a specific format is applied.

Use the **SetColWidthAuto** method to automatically set the column width to the correct size for all data in the column. The following code automatically sets the widths of columns 1 through 10.

```
F1Book1.SetColWidthAuto -1, 1, -1, 10, False
```


Dates and Times. Dates and times are automatically recognized by Formula One. They are entered in the cell as values and automatically formatted. The following date and time formats are automatically recognized.

Entered	Format Assigned
3/15/94	m/d/yy *
15-Mar-94	d-mmm-yy
15-Mar	d-mmm
Mar-94	mmm-yy
9:55 PM	h:mm AM/PM
9:55:33 PM	h:mm:ss AM/PM
21:55	h:mm
21:55:33	h:mm:ss
3/15/94 21:55	m/d/yy h:mm *

* These formats are locale-dependent. For example, in the United Kingdom, 15/3/94 would be recognized as the date format d/m/yy.

Text. Text is any set of characters that Formula One does not recognize as a number, date, or time.

Text that is wider than a cell ordinarily spills over into the cell immediately to the right. You can specify that text should wrap within the cell by enabling word wrap in your alignment format settings.

Logical and Error Values. Logical and error values are not normally entered directly in cells; they are usually the result of a formula. However, entering these values can be useful for testing formulas.

The logical values that can be entered are TRUE and FALSE. The error values that can be entered are #N/A, #VALUE!, #REF!, #NULL!, #DIV/0!, #NUM!, and #NAME?.

Entering Formulas

Formulas are the basic building blocks for analyzing and calculating worksheet data. A formula is a string containing numbers, operators, worksheet functions, cell references, and names. A formula can contain as many as 1024 characters.

- When you manually enter a formula in a worksheet, you must begin the entry with an equal sign (=). Formula One recognizes this entry as a formula.
- When entering a formula using the **Formula**, **FormulaRC**, and **FormulaSRC** properties, exclude the leading equal sign. These entities expect strings.

Numbers in formulas can be followed by a percent sign (%). Numbers with trailing percent signs are treated as percentages (e.g., 100% is evaluated as 1).

If text is encountered when a number is expected, the text is converted to a number. For example, the formula `1 + "3"` returns 4, because "3" is converted to a number. If the text cannot be converted to a valid number (e.g., `1 + "Text"`), #VALUE! is returned.

Likewise, if a number is encountered when text is expected, the number is converted to text. The formula `"The number is "&3` converts to the text string "The number is 3".

The value TRUE always converts to 1; FALSE converts to 0. If a number is encountered when a logical value is expected, a zero is converted to FALSE. All other numbers are converted to TRUE. If text is encountered when a logical value is expected, "TRUE" is converted to TRUE; "FALSE" is converted to FALSE. All other text returns #VALUE!.

Dates and times are recognized and converted to their serial values. For example, "10/10/94" - "10/1/94" equals 9.

Using Formula Operators

When creating formulas, Formula One provides a set of operators for specifying the type of calculation or evaluation to be performed on the formula data. The following table lists the formula operators.

Operator Type	Operator	Description
Arithmetic	+	Addition
	-	Subtraction
	/	Division
	*	Multiplication
	%	Percentage
	^	Exponentiation
Text	&	Concatenation
Comparison	=	Equal to
	>	Greater than
	<	Less than
	>=	Greater than or equal to
	<=	Less then or equal to
	<>	Not equal to
Reference	:, .., .	Range - produces a reference that includes all the cells between the two references (e.g., A1:A5 includes cells A1 and A5 and all cells in between).
	,	Union - produces one reference that includes the two references (e.g., A1:A10,C1:C10).

Using Operator Precedence

When combining operators in a formula, Formula One uses a specific order of precedence to calculate the formula. The following table lists the order of precedence for formula operators.

Operator	Description
()	Parentheses
:, ..., .	Range
,	Union
-	Negation (single operand)
%	Percentage
^	Exponentiation
* and /	Multiplication and Division
+ and -	Addition and Subtraction
&	Text concatenation
= < > <= >= <>	Comparison

Operators of like precedence are evaluated left to right. Parentheses should be used when it is necessary to change the order of evaluation. The following example illustrates how the result of a formula can be altered by adding parentheses to change the order of precedence.

Formula	Result
1+2*37	75
(1+2)*37	111

As illustrated in the previous table, the multiplication operator (*) has higher precedence than the addition operator (+). It is evaluated first unless parentheses are used to force the addition to take place first.

Understanding Cell References

A reference identifies a cell by referring to the row and column coordinates of the cell. References are based on the row and column headings. For example, A1 refers to the cell at the intersection of row 1 and column A. References can be used in formulas to access data from a worksheet.

A range of cells is specified by placing a colon (:) between two cell references. For example, the reference A1:C3 refers to the range anchored by cells A1 and C3. The range includes all cells in columns A, B, and C of rows 1, 2, and 3.

Absolute and Relative References

There are two types of cell references: relative and absolute.

- Relative references point to a cell based on its relative position to the current cell. When the cell containing the reference is copied, the reference is adjusted to point to a new cell with the same relative offset as the originally referenced cell.
- Absolute references point to a cell at an exact location. When the cell containing the formula is copied, the reference does not change. Absolute references are designated by placing a dollar sign (\$) in front of the row and column that is to be absolute.

References can be part absolute and part relative. These are called mixed references. The following table lists the reference types.

Reference	Type
A1	Relative reference pointing to cell A1.
\$A\$1	Absolute reference pointing to cell A1.
\$A1	Absolute column reference, relative row reference pointing to cell A1.
A\$1	Relative column reference, absolute row reference pointing to cell A1.

The reference operators can be used to specify multiple ranges in the same reference. For example, A1:C1,A10:C10 specifies the three cells A1, B1, and C1 and the three cells A10, B10, and C10. The formula =SUM(A1:C1,A10:C10) adds the values in all six cells.

References to Other Worksheets

You can reference cells in other worksheets in the same or different workbooks.

To reference a worksheet in the same workbook, use the following syntax:

Sheet3!A1

To reference a worksheet in a different workbook, use the following syntax:

[F1Book1]Sheet1!A1:B2

Workbooks are referenced by the value of their **Title** property, and must be loaded in another control for the reference to work.

References to Multiple Worksheets

You can also refer to multiple worksheets, or ranges in multiple worksheets. The following example references cell A1 in Sheet1 and cell A1 in Sheet2.

Sheet1:Sheet2!A1

Important Worksheets must be referenced in index order. For example, the reference to the sheet indexed 1 must come before the reference to the sheet indexed 2. Remember that the worksheet index is usually different than the sheet name that appears on the sheet tab. Worksheets are indexed from left to right, beginning with 1.

The following syntax references the range A1 to B2 in Sheet1 and the range A1 to B2 in Sheet2.

```
Sheet1:Sheet2!A1:B2
```

You can also reference multiple worksheet ranges in different workbooks by referencing the workbook name at the beginning of the syntax.

```
[F1Book1]Sheet1:Sheet2!A1  
[SSView2]Sheet1:Sheet2!A1:B2
```

External References

References can point to cells in other workbooks. This type of reference is called an external reference. An external reference is created by placing a workbook name in brackets, followed by the worksheet name and an exclamation point, and finally a cell reference. The following table shows examples of external references.

Reference	Type
[Sales]Sheet1!A1	Relative reference pointing to cell A1 in the first worksheet of a workbook titled Sales.
[FY91]Sheet2!\$A\$1	Absolute reference pointing to cell A1 in the second worksheet of a workbook titled FY91.
[Q1]Sheet1:Sheet2!\$A1	Absolute column reference, relative row reference pointing to cell A1 in the first and second worksheets in a workbook titled Q1.
[Store1]Sheet1:Sheet4!A1:F1	Relative row and column reference, pointing to the range A1 to F1 in a workbook titled Store1.

Automatically Entering Cell References

Cell references can be automatically entered as you enter a formula.

- **To automatically enter a cell reference:**
1. Enter the formula to the point of the range reference.

2. With the mouse, select the cell or range you want to reference.
- The reference of the range you select is automatically placed in the formula.
- When you enter a cell reference in this manner, Formula One assumes it is a relative reference.

Understanding Worksheet Errors

When a formula cannot be properly calculated, an error is returned in the cell. The following table lists the errors that can be generated.

Error	Cause
#ARRAY_FORMULA!	Formula One read an Excel file that contained an array formula. Since this feature is not supported in Formula One, this error value is placed in the cell that used to contain the array formula.
#DIV/0!	Divide by zero. May be caused by a reference to a blank cell or a cell containing zero.
#N/A	No value is available. May be caused by inappropriate values in the formula or a reference to a cell containing the #N/A value.
#NAME?	Name is not recognized. May be caused by a user defined name that is not defined.
#NULL!	Null intersection. An intersection of two ranges was defined that does not intersect.
#NUM!	Number problem. May be caused by inappropriate numbers in functions, an iteration that cannot solve for a value, or a formula that results in a number too large or too small to represent.
#REF!	Reference error. May be caused by referring to a cell that was deleted.
#VALUE!	Wrong argument type. May be caused by entering text where a number was expected, or supplying a range to an operator or function that was expecting a single value.

Displaying Formulas

It is often convenient to display formula text instead of the values they produce. Setting **ShowFormulas** to True, causes the worksheet to display formula text instead of formula results. Displaying formula text can help you debug formula- related problems.

The following example enables and disables the display of formulas.

```
F1Book1.ShowFormulas = TRUE 'Displays formulas  
F1Book1.ShowFormulas = FALSE 'Displays formula text
```

Built-In Worksheet Functions

Formula One contains a set of 143 built-in worksheet functions that provide the ability to perform complex calculations with very little work.

Worksheet functions:

- calculate and evaluate data
- can be used alone or in a formula
- are entered directly in the worksheet

Like formulas, worksheet functions return data to the cell in which they are entered.

Each function performs a specific calculation. The **SQRT** function is an example of a built-in function. With this function, you can easily calculate the square root of a number. The following example calculates the square root of 118:

```
=SQRT(118)
```

Understanding Functions

Most worksheet functions are composed of keywords and arguments. Every worksheet function contains a keyword, but not all functions require arguments.

- The keyword identifies the function and tells the worksheet what type of calculation or evaluation is performed. Each function keyword is unique.
- Arguments provide the data for the function to calculate or evaluate. The arguments for a function immediately follow the function keyword and are enclosed in parentheses.

Entering Functions

When entering functions in a worksheet, all functions are preceded by an equal sign (=). The leading equal sign tells the worksheet that the following information is to be evaluated or calculated.

The function keyword follows the equal sign. It can be entered in lowercase or uppercase characters. After the function is entered, the worksheet records the function keyword in uppercase characters, regardless of how it was entered.

If a function requires multiple arguments, the arguments are separated by commas. Some functions contain optional arguments. If you omit an optional argument, a default value is assumed for the argument.

Functions that do not require arguments still require a set of parentheses following the function keyword.

Nesting Functions

A function can be used as an argument for another function. When a function is used in this manner, you are *nesting functions*. The nested function must return the appropriate type of data for the function in which it is nested. You must also provide the necessary arguments for the nested function.

In the following example, the **AVERAGE** function is used as an argument for the **SUM** function. In this case, **AVERAGE** is nested in **SUM**.

```
=SUM(5.23, 6.82, AVERAGE(2.45, 5.62, 7.74), 8.95, 9.01)
```

Entering Arguments

The arguments for a function can be:

- Numerical values
- Logical values
- Text strings
- Error values
- References to cells or ranges

Each argument requires a specific type of data. Refer to Chapter 14, A-Z Worksheet Function Reference, to determine the type of data required for the function you are entering.

For most arguments, you can substitute a cell or range reference for the data required by an argument. For example, if an argument requires a number, you can substitute a reference to a cell that contains a number. The number in the referenced cell is used in the calculation of the function. The data in the referenced cell must be appropriate for the argument for which it is used.

Syntax Errors

If the worksheet function you enter contains syntax errors, Formula One does not allow the function to be entered. You must correct the errors before proceeding with other tasks.

Using Autofill Lists

If you frequently use lists of names, months, or days of the week in your worksheet, you can let Formula One do some of the work for you by using the autofill feature.

Formula One's default autofill lists contain frequently used series of text such as months and days of the week. When you enter one of the elements in these lists and drag the autofill handle, Formula One enters the rest of the data from the list as needed to fill the range you mark.

	A	B	C	D	E
1	Jan				
2					
3					

When Formula One recognizes this text as part of an autofill list, and . . .

	A	B	C	D	E	F
1	Jan	Feb	Mar	Apr	May	
2						
3						

you drag the fill handle to this position, the cells in the marked range are automatically filled with items from the list.

Once Formula One has recognized the text as an item from an autofill list, pressing Tab puts the next list item in the next cell to the right, or Enter puts the next list item in the next cell below.

Adding Autofill Lists

You can add custom autofill lists that include frequently used series of text with the **AutoFillItems** property. You can also access the **AutoFill** tab using the **OptionsDlg** method (F1AutoFillPage).

➤ **To add a new autofill list using the AutoFillItems property:**

- Use the **AutoFillItemsCount** method to determine the number of autofill lists. Then increment by one and use the **AutoFillItems** property to specify the new list. The following example illustrates:

```
F1Book1.AutoFillItems (F1Book1.AutoFillItemsCount+1) =  
    ➔ "A;B;C;D;E;F;G;H;I;J;K;L;M;N;O;P;Q;R;S;T;U;V;W;X;Y;Z"
```

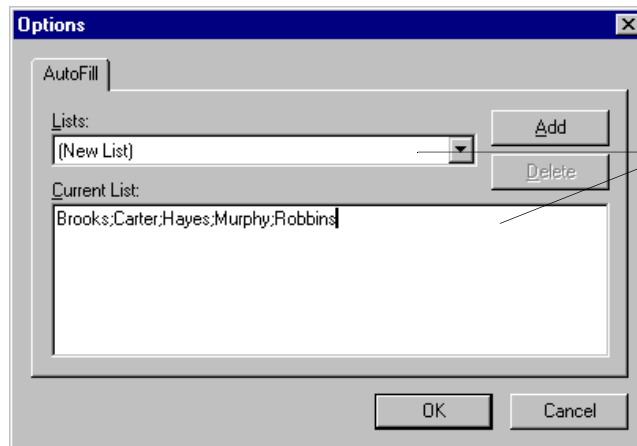
Note Values recognized as data cannot be used as autofill list items, such as 1973;1974, 1a;1b, 1%;2%.

➤ **To add a new autofill list using the OptionsDlg method:**

1. Use the following code:

```
F1Book1.OptionsDlg (F1AutoFillPage)
```

2. The following dialog box is displayed:



When (New List) is selected, you can type your custom list in the Current List text box.

You can also edit default lists using this text box.

3. Select (New List) from the Lists list.
4. Type your new autofill list in the Current List text box, separating each item with a semi-colon.
5. Click Add.

Deleting Autofill Lists

You can delete an autofill list using the **AutoFill** tab by selecting a list and clicking the Delete button. You can also delete a default or custom autofill list using the **DeleteAutoFillItems** method.

➤ **To manipulate an autofill list programmatically:**

- Get the current string of text that makes up a list by returning the value of the **AutoFillItems** property.

```
list = F1Book1.AutoFillItems(2)
```

- Replace a list by setting the **AutoFillItems** property.

```
list = "1st Qtr;2nd Qtr;3rd Qtr;4th Qtr"
F1Book1.AutoFillItems(2) = list
```

- Delete a list with the **DeleteAutofillItems** method.

```
F1Book1.DeleteAutoFillItems (F1Book1.AutoFillItemsCount -2)
```

Using Names

User-defined names are an easy way to identify a cell, a group of cells, a value, or a formula. For example, the formula “= Sales - Expenses” is much clearer than “A10 - A6”.

You can also use names to identify constants and formula expressions. For example, you might define the name **LightSpeed** as 186000. You could then use the name **LightSpeed** in all your formulas. Or, you could define the name **SqrtTwo** as the formula **SQRT(2)**.

➤ **To define names in the Workbook Designer:**

1. If you are naming a range, select it.
2. Select Insert > Name.
3. Enter a name that describes the cell reference in the Name text box. Do not use spaces in the name.

A cell or range reference that represents your selection from step 1 is displayed in the Formula text box. You can edit this reference, if desired.

4. Click Add to add the new name to the list.

You can delete any name from the list by selecting it in the list and clicking Delete.

➤ **To define names programmatically:**

- Use the **DefinedName** property. The following code uses this property to define a name.

```
F1Book1.DefinedName ("Sales") = "$A$10"
```

This example defines the name “Sales” as \$A\$10. The name “Sales” can then be used in formulas instead of the reference.

Formula One has a set of built-in names. These names are used by the print functions. The built-in names are listed in the following table.

Built in Name	Purpose
Print_Area	Defines the print area used during printing. This name can contain one or more ranges (e.g., A1:C3,A11:C13).
Print_Titles	Defines the row and column titles that are printed on each new page. Entire rows and columns must be selected when you define this name.

Calculating Worksheets

Formula One calculates cells in natural order. In natural order calculation, formulas are calculated in such a way that all dependencies are calculated before their dependents. This ensures that the formula results are always correct.

When a worksheet is edited, Formula One first adjusts formula references so they point to the correct cells. Then, Formula One determines the natural order of the formulas.

When a change is made to a cell, the formulas are recalculated to keep all worksheets in the workbook current, ensuring that data is always valid.

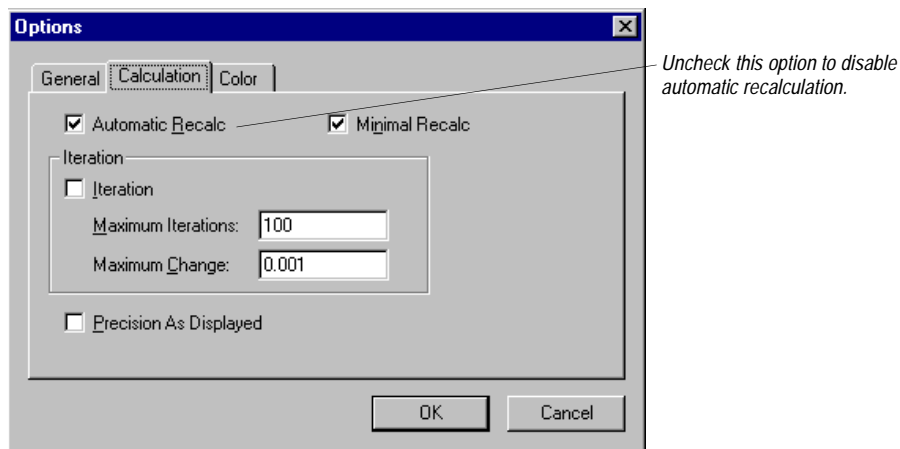
Setting Automatic Recalculation

Normally, automatic recalculation is enabled, which means the worksheet is recalculated each time a cell is changed and system processing is idle. For moderately sized worksheets, recalculation operations happen in a fraction of a second. But for large worksheets or situations where many cells are changed by code, this reorganization and recalculation process can slow system processing.

In these situations, you may disable automatic recalculation while your code operates on the worksheet. After the operation is completed, you may enable automatic recalculation to update the worksheet. Automatic recalculation can be disabled using the Workbook Designer or with the **AutoRecalc** property or the **OptionsDlg** method.

➤ **To change automatic recalculation using the Workbook Designer:**

1. Select Tools > Options and click the Calculation tab, shown below.



2. Select the Automatic Recalc option to turn auto recalc on, or unselect it to turn it off.
3. Click OK.

➤ **To change automatic recalculation programmatically:**

- The following code uses the **AutoRecalc** property to disable the automatic recalculation setting.

```
F1Book1.AutoRecalc = FALSE 'Disables automatic recalculation
```

Setting Minimal Recalculation

Minimal recalculation means that, when Formula One needs to recalculate the formulas on the worksheet, it only recalculates cells that refer to cells that have changed. Cells and references to cells that have not changed are left alone. This will often cause a dramatic improvement in recalculation speed. By default, minimal recalculation is on.

When minimal recalculation happens is determined by the setting of the **AutoRecalc** property. When **AutoRecalc** is set to True, minimal recalc will be called whenever the worksheet changes. If **AutoRecalc** is off, minimal recalc will be called when the user presses F9 or uses the **Recalc** method.

The CALL, COLUMNS, INDEX, INDIRECT, NOW, OFFSET, RAND, ROWS, and TODAY worksheet functions must be reevaluated during every recalc to ensure accurate results. Formulas that use these functions are reevaluated during every recalc operation regardless of whether minimal recalc is on or off.

You may encounter an unusual instance where minimal recalculation slows worksheet processing speed. In these cases, you may want to turn minimal recalculation off.

- **To change minimal recalculation in the Workbook Designer:**
 1. Select Tools > Options and click the Calculation tab, shown on page 78.
 2. Select the Minimal Recalc option to turn minimal recalc on, or unselect it to turn it off.
 3. Click OK
- **To change minimal recalculation programmatically:**
Use the **MinimalRecalc** property.

Solving Circular References

In some circumstances, a formula refers to its own cell, either directly or indirectly. This is called a circular reference. To solve a formula that contains a circular reference, *iteration* must be used. Iteration is the process of repeatedly calculating a worksheet until a specific condition is met.

Formula One supports iteration using the **IterationEnabled**, **IterationMax**, and **IterationMaxChange** properties; the **GetIteration** and **SetIteration** methods; and the **OptionsDlg** (F1CalculationPage) method. These properties and methods allow you to specify the maximum number of iterations and the maximum change between iterations. The iteration continues until one of those two conditions is met.

The following example involves a circular reference:

Suppose your small business has 10,000 shares of stock owned by four shareholders. You decide to let a fifth shareholder enter your partnership. In return for his investment, you give him 10 percent of the company. How many more shares will the company have to issue to give the new investor 10% of the company?

The following illustration shows the results of this example as it is entered in a worksheet.

	A	B	C	D
1	Old Shares	10,000		
2	Total Shares	= Old Shares + New Shares		
3	New Shares	= Total Shares * 10%		
4				

	A	B	C	D
1	Old Shares	10,000		
2	Total Shares	11,111		
3	New Shares	1,111		
4				

The formulas in B2 and B3 create a circular reference in this example worksheet.

The first worksheet shows the formula text, the second worksheet shows the results of the formulas.

- **To control the number of times a circular reference is calculated using the Workbook Designer:**
 1. Select Tools > Options and select the Calculation tab, shown on page 78.
 2. Click the Iteration check box to limit iteration for calculating circular references.

3. In the Maximum Iterations text box, type the maximum number of iterations you want Formula One to execute.
4. In the Maximum Change text box, type the maximum change between iterations. The smaller the number, the more accurate your answer is.
5. Click OK.

Limiting Data Entry

Some applications might require that the user not be allowed to enter or edit data. Formula One gives you many options for restricting data entry.

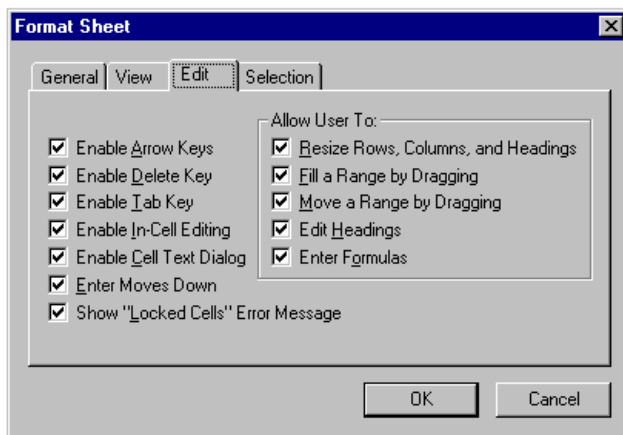
Denying Access to a Workbook

You may make an entire workbook and all the worksheets in it read-only. Any data manipulation in such a workbook must be done programmatically. You may deny access to a workbook using the Workbook Designer or using properties and methods

► To use the Workbook Designer to make a workbook read-only:

You must hide the formula bar and disable all of the user's editing options.

1. To hide the formula bar, select View > Formula bar. Make sure the option is unselected.
2. To disable users' editing options, first select all the worksheets in the workbook.
3. Choose Format > Sheet > Properties and click on the Edit tab. The Edit tab, shown below, will appear.



4. Unselect the following options: Enable Delete Key, Enable In-Cell Editing, Enable Cell Text Dialog, and all of the options in the Allow User To box.
5. Click OK.

- **To use properties and methods to make a workbook read-only:**
 - Set all of the following properties to False: **ShowEditBar**, **AllowDelete**, **AllowInCellEditing**, **AllowCellTextDlg**, **AllowResize**, **AllowFillRange**, **AllowMoveRange**, **AllowEditHeaders**, and **AllowFormulas**. Data and formula entry and editing is thus prevented. Any data manipulation must be performed programmatically.

Denying Access to a Worksheet

To deny access to a worksheet, hide the worksheet tabs. For information on how to do this in the Workbook Designer and programmatically, see “Displaying Parts of the Workbook Designer” on page 45.

Denying Access to Certain Cells

To deny access to one, a few, or all of the cells in a worksheet, lock the cells and enable worksheet protection. Locked cells may be selected, but they cannot be changed, moved, resized, or deleted. You may protect the entire worksheet or just certain cells in the worksheet. You can protect the worksheet in the Workbook Designer or programmatically.

Protecting the worksheet is a two-step process: First you choose which cells you want to be locked. Then you enable worksheet protection. Locking the cells has no effect until you enable the protection.

By default, worksheet cells are locked and protection is disabled.

- **To protect the worksheet in the Workbook Designer:**
 1. Select the cells you want to lock or unlock.
 2. Select Format > Cells and click the Protection tab.

The Protection tab of the Format Cells dialog box appears.
 3. Check the Locked check box to lock the selected cells. Uncheck the Locked check box to unlock the selected cells. Click OK.
 4. Select Format > Sheet > Protection. A check by the Protection menu item means that protection is enabled and locked cells cannot be entered and changed.
- **To set editing permissions on selected cells programmatically:**
 1. Select the cells you want to lock and use the **GetCellFormat** or **SetCellFormat** method of the **F1Book** object to create an **F1CellFormat** object for the selected cells.
 2. Use the **ProtectionLocked** property of the **F1CellFormat** object to lock or unlock the selected cells. (Note: You can use the **ProtectionHidden** property of the **F1CellFormat** object to hide cells.)
 3. Use the **EnableProtection** property to enable protection.

Note Worksheet cells are locked by default. If you want to protect only selected cells, you must unlock the rest of the worksheet cells, then enable worksheet protection.

Working With Locked Cells

By default, in the Workbook Designer, when a user selects a locked cell and tries to enter data, Formula One will beep and display the message “Locked cells cannot be modified.” You can turn off the error message so that Formula One just beeps in this situation. To turn off the error message, uncheck the Show “Locked Cells” Error Message in the Edit tab of the Format Sheet dialog box. To turn off the error message programmatically, set the **ShowLockedCellsError** property to False.

Also in the Workbook Designer, when a locked cell is selected, the ENTER, SHIFT-ENTER, TAB, and SHIFT-TAB keys advance the selection to the next unlocked cell.

Denying Access to Row and Column Headings

You may keep the user from changing row and column heading text in a worksheet.

➤ **To use the Workbook Designer to deny access to row and column headings:**

1. Select the worksheet(s) you want to restrict access to headings in.
2. Choose Format > Sheet > Properties and click the Edit tab. The Edit tab, shown on page 80, will appear.
3. Uncheck the Edit Headings box and click OK.

➤ **To use properties to deny access to row and column headings:**

- Set the **AllowEditHeaders** property to False.

Restricting Cell Data to Certain Values

You can restrict the user to entering only specific values in a cell by specifying a validation rule for the cell. A validation rule consists of a formula to test, and text to display if the validation fails. If the formula returns True, the value is entered. If the formula returns a text string, the string is displayed and the value is not entered. If the formula returns False, the value is not entered and the validation text is displayed in an error dialog box.

For example, you can limit the range of values a user can enter in a cell by creating a rule that fails if the user enters a number under 100 and displays the message “Enter a value greater than 100.”

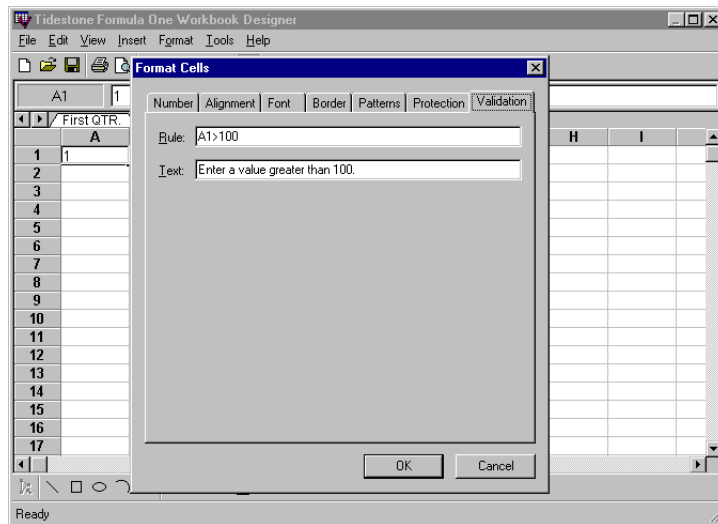
You can use relative references in validation rules. These references are considered to be relative to the active cell. This allows a validation rule to be properly applied to an entire range.

Note Validation rules are only checked if data is entered manually, or through use of the **Entry** or **EntryRC** property. Any other way of entering data, such as selecting a value from a check box, bypasses validation.

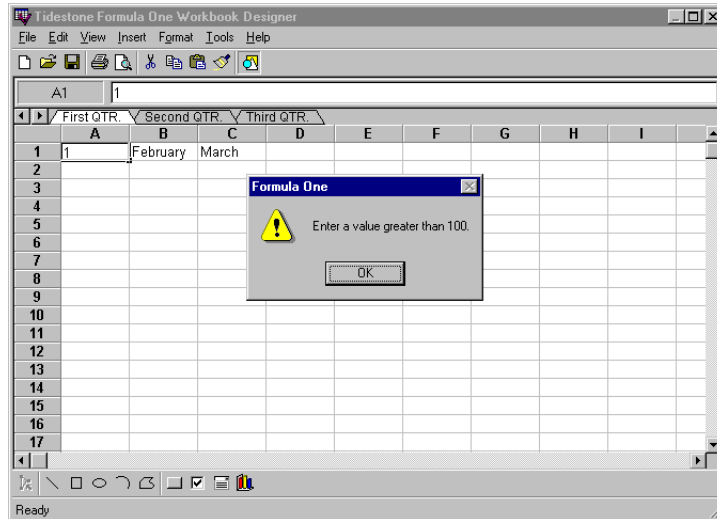
Note Formula One's validation rules are incompatible with Microsoft Excel's validation rules.

➤ **To create a validation rule in the Workbook Designer:**

1. Select the cell for which you want to create a validation rule.
2. Select Format > Cells to display the Format Cells dialog box.
3. Select the Validation tab.
4. Enter a validation formula in the Rule text box.
5. Enter text to be displayed if the data fails the validation test in the Text box.
6. Click OK.



This illustration shows the validation rule created for cell A1.



This illustration shows the message displayed when the data entered in cell A1 fails the validation rule.

➤ **To work with validation rules programmatically:**

- Use the **ValidationRule** property of the **F1CellFormat** object to create, edit, or return the validation rule associated with the specified cell. (You can also use the **ValidationRuleLocal**, **ValidationRuleRC**, and **ValidationRuleLocalRC** properties to set or return validation rules.)
- Use the **ValidationText** property of the **F1CellFormat** object to create, edit, or return the text to be displayed if the validation rule fails.
- Use the **FormatCellsDlg** method to display the Format Cells dialog box. For example, to only display the Validation tab of the Format Cells dialog box, use the following code:


```
F1Book1.FormatCellsDlg (64)
```
- Use the **ValidationFailed** event to respond programmatically when data fails. You can even change the value of the data within the **ValidationFailed** event and attempt to re-validate.

All of these properties, methods, and events are documented in detail in the Formula One Online Documentation.

The Validation Formula

The validation formula follows basically the same rules as those for defined names (see “Using Names” on page 76). It must also be a worksheet formula that evaluates to True or False. Following are a number of examples of validation formulas.

```
SUM(A6:A7) > A5  
AND(A6>1, A6 <100)  
IF (A7>1,A7<100,A7>0)  
OR(ISLOGICAL (A7), A7=1,A7=0)
```

Denying Entry of Formulas in a Worksheet

You may prevent users from entering formulas in a worksheet while still allowing them to enter constant values. You may do this in the Workbook Designer or through code.

- **To use the Workbook Designer to deny users the ability to enter formulas:**
 1. Select the worksheet(s) in which you don't want users to enter formulas.
 2. Choose Format > Sheet > Properties and click the Edit tab. The Edit tab, shown on page 80, will appear.
 3. Uncheck the Enter Formulas box and click OK.
- **To use properties and methods to deny users the ability to enter formulas:**
 - Set the **AllowFormulas** property to False. Setting this property to False does not affect the entry and editing of constant values.

Restricting the Use of Certain Keys

You may enable and disable a variety of navigation and data entry keys for particular worksheets. You may do this in the Workbook Designer and in code.

- **To use the Workbook Designer to disable the use of certain keys:**
 1. Select the worksheets to which you want to apply the changes.
 2. Select Format > Sheet > Properties and click the Edit tab. The Edit tab, shown on page 80, will appear.
 3. To disable use of the arrow, delete, and/or tab keys, unselect the Enable Arrow Keys, Enable Delete Key, and/or Enable Tab Key options.
 4. To prevent the Cell Text dialog box from appearing when the user presses F2 twice, unselect the Enable Cell Text Dialog option. (The Cell Text dialog allows the user to enter multi-line data.)
 5. To leave the active cell selected when a user presses Enter, unselect the Enter Moves Down option. To move the active cell down one row when the user presses Enter, check the enter Moves Down box.
 6. Click OK.

- **To use properties and methods to disable the use of certain keys:**
 1. Use the following properties: **AllowArrows**, **AllowDelete**, **AllowTabs**, **AllowCellTextDlg**, and **EnterMovesDown**.

Restricting the Use of Certain Mouse Actions

For any of the worksheets in a workbook, you can prevent the user from resizing cells and headings, from filling a range of cells by dragging with the mouse, and from moving ranges by dragging. You can prevent these actions using the Workbook Designer or using code.

- **To use the Workbook Designer to prevent certain mouse actions:**
 1. Select the worksheet(s) that you want to deny access to mouse actions to.
 2. Select Format > Sheet > Properties and click the Edit tab. The Edit tab, shown on page 80, will appear.
 3. To keep users from changing the size of rows, columns, and row and column headings, unselect the Resize Rows, Columns, and Headings option.
 4. To keep users from filling a range by dragging, unselect the Fill a Range by Dragging option.
 5. To keep users from moving a cell or range by dragging, unselect the Move a Range by Dragging option.
 6. Click OK.
- **To use properties to prevent certain mouse actions:**
 - Use the **AllowResize**, **AllowFillRange**, and **AllowMoveRange** properties.

CHAPTER 6

Editing Worksheets

Formula One provides a variety of ways to edit data in worksheets.

- You can copy, move, and paste selections interactively or programmatically.
- Methods allow you to insert data in and delete data from ranges, rows, and columns.
- A selected range of data can be sorted according to keys that you specify.

Copying, Moving, and Pasting Selections

You can copy, move, and paste selections interactively or programmatically. Selections in a worksheet contain many attributes such as:

- **Formulas.** Establish the value in a cell. Formulas are displayed in the formula bar when the cell is active.
- **Values.** What displays in the cell.
- **Formats.** How the cells and values are shown, such as red text applied to a selection.

Note Formula One maintains its own internal clipboard and also supports text on the Windows clipboard. The internal clipboard is more flexible than the Windows clipboard. The internal clipboard retains formulas and allows cell references to be adjusted when cells are pasted. The Windows clipboard only holds text and formatting; cell references are not maintained by the Windows clipboard.

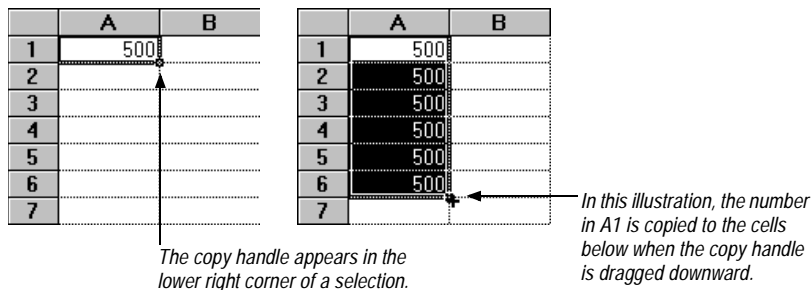
Using Dragging to Move, Copy, and Paste Selections

If you are moving or copying a selection within the same worksheet, dragging or dragging-and-dropping are the most simple techniques to use. This section discusses how to:

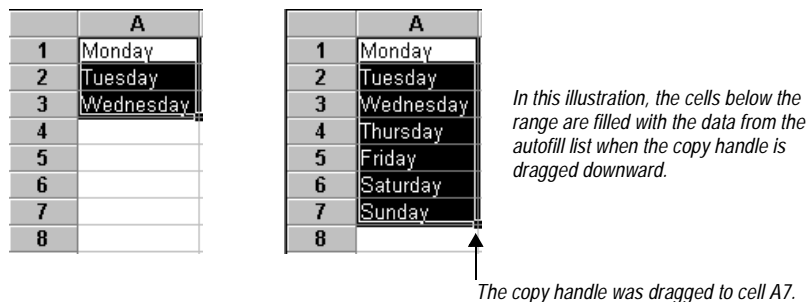
- copy a selection to the right or down using dragging
- drag-and-drop a selection to copy or move it

Copying a Selection Using Dragging

You can copy a selection to the right or down by dragging the copy handle of a selection. The copy handle is the small knob in the lower right corner of a selection. When you copy data using the copy handle, the pointer changes to a small crosshair. The following illustrations describe how to copy by dragging.



You can also copy a selected range in a worksheet. If an autofill list is defined for the data in the range, the data is copied using the autofill list, as shown in the following illustration.



Moving or Copying a Selection Using Drag-and-Drop

The following steps describe how to move or copy a selection using Formula One's drag-and-drop. Formula One's drag-and-drop functionality is different from this type of functionality in other applications. For information about drag-and-drop in other applications, refer to "Using Drag-and-Drop with Other Applications" on page 90.

► To move or copy a selection using drag-and-drop:

1. Select the cells that contain the data, object, or chart you want to move or copy.
2. Position the pointer on the border of the selection.

When placed on the selection border, the pointer changes to an arrow, as shown in the following illustration.

	A	B	C	D	E	F	G	H
1		Q1	Q2	Q3	Q4			
2	Americas	7026.05	5482.56	5517.19	4412.46			
3	Asia	7899.55	7733.54	9452.51	9367.54			
4	Europe	4734.25	4975.25	6587.21	5571.25			
5		19659.85	18191.35	21556.91	19351.25			
6								
7								
8								
9								
10								

The pointer appears as an arrow when positioned on a selection border.

3. When the pointer changes to an arrow,
 - to move the selection, hold down the mouse button and drag-and-drop the selection to the new location.
 - to copy the selection, hold down the CTRL key and drag-and-drop the selection to the new location

An outline of the selection moves as you drag the pointer, as shown in the following illustration.

	A	B	C	D	E	F	G	H
1		Q1	Q2	Q3	Q4			
2	Americas	7026.05	5482.56	5517.19	4412.46			
3	Asia	7899.55	7733.54	9452.51	9367.54			
4	Europe	4734.25	4975.25	6587.21	5571.25			
5		19659.85	18191.35	21556.91	19351.25			
6								
7								
8								
9								
10								
11								
12								
13								
14								
15								

When you drag-and-drop a selection border, the highlighted area moves to indicate the placement of the selection.

Tip To place the selection on an area of the worksheet that is not visible, move the cursor to the edge of the worksheet and scroll to the appropriate location.


4. When the pointer is at the appropriate location, release the mouse button.

The original selection is not moved if you hold down the CTRL key. However, The original selection is moved to the new location if do not hold down the CTRL key.



Note You can disable the user's ability to copy data by setting the **AllowFillRange** property to False. This disables interactive data copying.

Using Drag-and-Drop with Other Applications



You can easily drag text and Windows metafile pictures onto a Formula One worksheet from other Windows applications, such as Excel.

Note The drag-and-drop functionality only works with applications that support Windows CF_TEXT and CF_METAFILEPICT data. The mouse pointer changes to  to indicate that the data cannot be placed in the selected location. For example, if you select data and drag it to the Desktop, the pointer indicates that the data cannot be dropped onto the desktop.

➤ **To drag information from another application and drop it into Formula One:**

1. Arrange the application windows so that the source and destination documents are open and visible. You must be able to see the information you want to drag and the Formula One worksheet you want to place it on.
2. Select the information you want to move or copy.
 - To move the information, point to the selected information, and then hold down the mouse button. When the pointer appears, drag the pointer to the new location, and then release the mouse button. The pointer changes to  to indicate a move.
 - To copy the information, hold down the CTRL key, point to the selected information, and then hold down the mouse button. When the pointer appears, drag the pointer to the new location, and then release both the mouse button and the CTRL key. The pointer changes to  to indicate a copy.

➤ **To drag a selection from Formula One to another application:**

1. Arrange the application windows so that the source and destination documents are open and visible. You must be able to see the data on the Formula One worksheet.
2. Select the information you want to move or copy.
 - To move the selection, point to it, and then hold down the mouse button. When the pointer appears, drag the pointer to the new location, and then release the mouse button. The pointer changes to  to indicate a move.
 - To copy the selection, hold down the CTRL key, point to the selection, and then hold down the mouse button. When the pointer appears, drag the pointer to the new location, and then release both the mouse button and the CTRL key. The pointer changes to  to indicate a copy.

Using Menu Commands to Move, Copy, and Paste Selections

If you are moving or copying a selection for a longer distance, such as another workbook, worksheet, or application you might want to use menu commands. Also, in some instances you might not want to paste all cell attributes to the other location. Formula One allows you to choose whether to paste formulas, values, or formats. Furthermore, Formula One allows you to copy formatting from a cell or range and apply that formatting to another cell or range.

This section describes how to use menu commands to:

- move, or copy and paste a selection
- copy formatting of a selection and apply it to a cell or range
- paste individual attributes of a selection to another location

Formula One also provides toolbar buttons that perform the same functions as the menu commands.

➤ **To move or copy a selection using menu commands:**

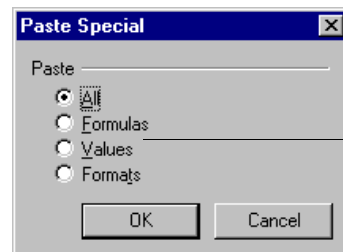
1. Select the cells that you want to cut.
2. Select Edit > Cut to move a selection. Alternatively, select Edit > Copy to copy a selection.
3. Select the upper left cell where you want to move or paste the selection and select Edit > Paste.

This pastes all cell attributes.

➤ **To paste individual cell attributes:**

1. Select the upper left cell of where you want to paste the selection and select Edit > Paste Special.

The Paste Special dialog box is displayed as shown in the following illustration:



Select the option button for the attribute that you want to paste.

2. Click OK.

➤ **To copy only the formatting of a selection and apply it to a cell or range:**

1. Select the cells that contain the formatting that you want to apply.
2. Select Edit > Copy Cell Format.
3. Highlight the cell or range for which you want to apply the formatting and click the left mouse button.

The formatting is applied.

Using Methods to Edit, Move, Copy, and Paste Selections

Selections can be edited, moved, copied, or pasted programmatically using the techniques described in the following sections.

Using Methods to Edit Ranges

Ranges of data can be edited using one of several editing methods. Formula One automatically adjusts cell references when cells are moved. Thus, the integrity of worksheet formulas remains intact. The following table describes the methods that interact with the clipboards.

Method	Operation
ClearClipboard	Frees Formula One resources that are on the Windows clipboard.
CopyAll	Copies the contents of the active worksheet in the specified view to the active worksheet in the current view.
EditCopy	Copies the current selection to the internal clipboard and the Windows clipboard (in text format only). If there is more than one selection, only the first selection is copied.
EditCut	Cuts the current selection to the internal clipboard. If there is more than one selection, only the first selection is cut.
EditPaste	Pastes the contents of the internal clipboard to the current selection. If the internal clipboard is empty, text is pasted from the Windows clipboard. You can also paste tab-delimited blocks of data.
EditPasteValues	Pastes values from the clipboard to the current worksheet selection. Any formatting applied to the values is ignored. In addition, only formula results are pasted; formulas are ignored.
EditPasteSpecial	Pastes formats, formulas, values, or everything from the clipboard to the selected range.

- If you cut a cell to which formulas refer, the formula references are maintained while the cell remains in the clipboard. If the cell is subsequently pasted, references in the original formulas are adjusted to point to the cell's new location.
- If a cell containing a formula is copied and subsequently pasted, its relative references are adjusted to point to a new location.

Using Methods to Copy Selections Across

Four methods copy selections within and between worksheets. The following table describes these methods.

Method	Operation
EditCopyDown	Copies the top row of the selection down. Relative references are automatically adjusted.
EditCopyRight	Copies the left column of the selection right. Relative references are automatically adjusted.
CopyRange	Copies a range from one range to another, within the same workbook or between workbooks.
CopyRangeEx	Copies a range from one worksheet to another, within the same workbook or between workbooks.

Using Methods to Move Data

There are several ways you can move ranges of data. The easiest way is to use the **MoveRange** method. When you use this method, the integrity of formula cell references is maintained.

If there is special processing that must be performed when data is moved, you can use a loop in C or C++ code to move the data. However, cell references are not adjusted using this technique.

Using Methods to Copy Data Between Worksheets and Arrays

Formula One provides two methods for copying data between a worksheet and a variant array.

- The **CopyDataFromArray** method allows you to quickly copy data from an array to a specified range of cells in a worksheet.
- The **CopyDataToArray** method allows you to quickly copy data from a specified range of cells in a worksheet to a variant array.

Transferring Data via Uniform Data Transfer

In development environments which support Uniform Data Transfer, Formula One exposes its **IDataObject** for Uniform Data Transfer support. Data can be transferred between the ActiveX control and other applications that support UDT in the CF_TEXT format. When an application requests data from Formula One, the current selection is sent in tab-delimited format. When CF_TEXT data is sent to Formula One, it is copied into the worksheet starting at Cell A1 unless the **DataTransferRange** property has been set.

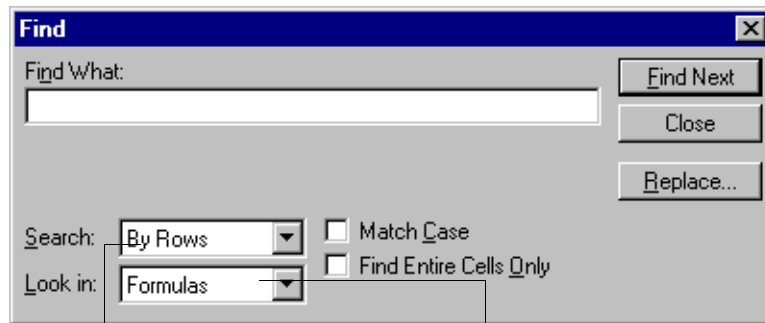
Finding and Replacing Data in Formula One

Formula One allows you to find and replace data within a worksheet in formulas and values. You may invoke the Find dialog and a separate Replace dialog in the Workbook Designer or programmatically within the application. You may also find and replace data in a worksheet using properties and methods.

Whichever way you choose to do the find and replace, Formula One gives you several search options: You may search by row or by column; you may search for values or formulas or both; and you may specify search options like matching case.

► **To find data in a worksheet using the Workbook Designer:**

1. Invoke the Workbook Designer by selecting that option on the context menu.
2. Choose Edit > Find or issue the Ctrl F keyboard shortcut. The Find dialog, shown below, will appear.



The Find dialog allows you to search for data in a worksheet by row order or by column order.

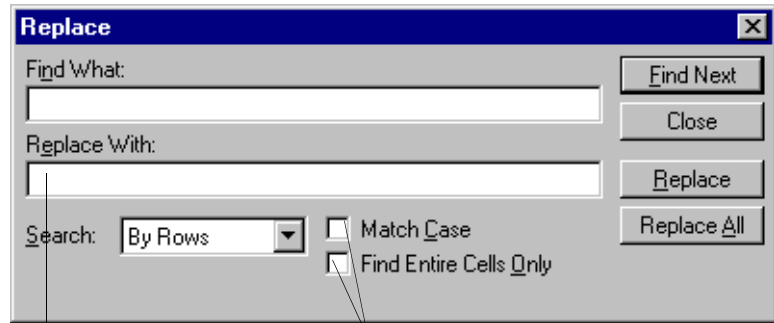
In a Formula One worksheet, you can search for data in cell Values or Formulas.

3. Enter the data you want Formula One to search for in the Find What dialog box. Choose any special search options you want, then click the Find Next button.
4. Formula One will select the first cell that contains the search string you chose. If it finds no instances of the string, it will display the message "Cannot find matching data."
5. If you want to replace the data, click Replace. The Replace dialog box, shown below, will appear.
6. Click Close when you finish.

► **To replace data in a worksheet using the Workbook Designer:**

1. Invoke the Workbook Designer by selecting that option on the context menu.

- Choose Edit > Replace or issue the Ctrl H keyboard shortcut. The Replace dialog, shown below, will appear.



The Replace dialog provides a text box for entering the data string you wish to replace with.

Select Match Case if you want the search to find data with matching case.

Select Find Entire Cells Only if you want the search to locate cells containing only the string. Otherwise, Formula One will also locate cells that contain the search string and other data.

- Enter the data you want Formula One to search for in the Find What dialog box. Enter the data you want to replace that data with in the Replace With box. Choose any special search options you want, then click Find Next.
- Formula One will select the first cell that contains the search string you chose. If it finds no instances of the string, it will display the message "Cannot find matching data."
- Click Replace if you want to replace the data, or click the Find Next button to search for the next instance of the search string.
- Click Close when you finish.

➤ **To invoke the Find and Replace dialogs programmatically:**

- Use the **FindDlg** and the **ReplaceDlg** methods.

➤ **To find and replace data in a worksheet using properties and methods:**

Formula One provides an API object, **F1FindReplaceInfo**, to help you programmatically search for data in a worksheet.

Use the **DefineSearch** method to create an **F1FindReplaceInfo** object. The **DefineSearch** method sets up the search area and criteria. You may indicate the string you wish to find, the range of cells you wish to search within the worksheet, and any special flags you want to set such as matching case or matching the entire cell.

The search itself is done by the **FindNext** method of the **F1FindReplaceInfo** object. The **FindNext** method returns True when it finds a matching string. It also sets the **Row** and **Col** properties for the **F1FindReplaceInfo** object so that they return the row and column in which the match was found.

The **Replace** method of the **F1FindReplaceInfo** object replaces the search string found by the **FindNext** method with the replacement string of your choice.

For more information regarding find and replace objects, properties, and methods, review the sections on the `F1FindReplaceInfo` object in the Formula One on-line help.

Inserting Cells, Rows, and Columns

You can insert cells, rows, or columns using the Workbook Designer or programmatically.

➤ **To insert cells using the Workbook Designer:**

1. Select a range of cells in the size and place of the new cells that you want to insert.
2. Select **Insert > Cells** to display the Insert dialog box.
3. Select an Insert option button to specify a direction that you want the surrounding rows, columns, or cells to shift.
4. Click **OK**.

➤ **To insert rows using the Workbook Designer:**

1. Select the same number of columns directly below where you want to add the new row or rows.
2. Select **Insert > Rows**.

➤ **To insert columns using the Workbook Designer:**

1. Select the same number of columns directly to the right of where you want to add the new column or columns.
2. Select **Insert > Columns**.

➤ **To insert new cells programmatically:**

- Use the **InsertRange** method to insert new cells in a worksheet. You supply a range where new cells are inserted and specify how the current cells in that range should be shifted to make room for the new cells.

The following example inserts a two by two block of cells starting at B2. The current cells in the range B2:C3 are shifted downward to make room for the new cells.

```
F1Book1.InsertRange 2, 2, 3, 3, F1ShiftVertical
```

- Use the **EditInsert** method to insert cells, rows, and columns. You specify whether rows, columns, or cells should be inserted. This method call uses the currently selected range to determine how many rows, columns, or cells to insert.

When new cells are inserted, cell references in formulas are adjusted so the formulas remain correct.

The next four examples assume the range A4:B5 is selected (a two by two range). In the following code, data in all columns and rows 4 and below is shifted down two rows to allow room for the inserted cells.

```
F1Book1.EditInsert F1ShiftRows
```

The following code shifts all data in the worksheet right two columns to allow room for the inserted cells.

```
F1Book1.EditInsert F1ShiftCols
```

In the following code, data in all columns of rows 4 and 5 is shifted right two columns to allow room for the inserted cells

```
F1Book1.EditInsert F1ShiftHorizontal
```

In the following code, data in columns A and B in rows 4 and below is shifted down two rows to allow room for the inserted cells

```
F1Book1.EditInsert F1ShiftVertical
```

Clearing and Deleting Cells, Rows, and Columns

You can delete or clear cells, rows, or columns in the Workbook Designer or programmatically. Deleting cells removes the cells and shifts the surrounding data to fill the space. Clearing cells leaves the cells but deletes the data.

➤ **To delete cells in the Workbook Designer:**

1. Select the cells, rows, or columns you want to delete.
2. Select Edit > Delete to display the Delete dialog box.
3. Select a Delete option button to specify a direction you want the surrounding rows, columns, or cells to shift.
4. Click OK.

➤ **To clear cells in the Workbook Designer:**

1. Select the cells, rows, or columns you want to clear.
2. Choose one of the following, depending on what you want to clear from the cells:
Select Edit > Clear > Contents to clear only the data and formulas.
Select Edit > Clear > Formats to clear only the cell formats.
Select Edit > Clear > All to clear the content and formats.
3. Click OK.

➤ **To delete or clear cells programmatically:**

Several methods delete and clear data. The following table lists these methods.

Method	Operation
EditDelete	Deletes the current selection.
DeleteRange	Deletes the specified range.
EditClear	Clears the current selection.
ClearRange	Clears the specified range.

- **EditDelete** is similar to the **EditInsert** method. For **EditDelete**, you specify whether cells, rows, or columns should be deleted. The number of cells, rows, or columns deleted is determined from the current selection. For example, to delete rows (based on the current selection), you could use the following code:

```
F1Book1.EditDelete F1ShiftRows
```

If you delete cells (e.g., using **EditDelete** or **DeleteRange**) to which a formula refers, those formulas return a #REF! error because the referenced cells no longer exist.

- To delete a specific range instead of the current selection, use the **DeleteRange** method. This method allows you to explicitly specify the range to delete. The following code uses this method.

```
F1Book1.DeleteRange 1, 1, 3, 3, F1ShiftRows
```

Clearing a cell can clear the value or format in a cell, or both. You can also specify whether or not formulas are cleared. Clearing does not shift other cells in the worksheet. The cleared cell has a value of zero. Formulas that refer to cleared cells obtain a value of zero from those cells.

- You can use **EditClear** or **ClearRange** to clear a cell or range of cells. The following example clears the current selection.

```
F1Book1.EditClear F1ClearAll
```

Alternately, you can use the following example to clear specific rows or columns instead of the current selection.

```
F1Book1.ClearRange 1, 1, 3, 3, F1ClearAll
```

Sorting Data in Worksheets

You can sort the data in a worksheet and specify the keys by which the data is sorted. **SortDlg** displays a dialog box that allows the user to specify sort keys, sort rows or columns, and ascending or descending sort order. Before using the sort dialog box, a range in a worksheet must be selected. The data in the selected range is the data that is sorted.

You can also sort worksheet data using the **Sort** or **Sort3** methods. Refer to the Formula One Online Documentation for information about these methods.

CHAPTER 7

Formatting Worksheets

Formula One supports a rich set of data formatting capabilities. You can use number formats to display numbers in a certain way -- as dates, for example, or as dollars. You can use cell formats to change the size of cells and to add colors and borders. You can use font formats to change the typeface and style of the characters in your worksheet.

Using Built-in Number Formats

Number formats determine how numbers look when they appear in the worksheet. When you create a new worksheet, by default all cells in that worksheet use the General format, which means that as you enter data in the worksheet, Formula One determines the type of data and applies the appropriate number format (e.g., if you enter a date, a date format is applied).

To specify formats other than General, you can apply a number of built-in number formats using the Workbook Designer. If you do not find the number format you want, you can create your own custom number format.

➤ **To apply number formats using the Workbook Designer:**

1. Select the cells for which you want to change the number format.
2. Select Format > Cells and select the Number tab, if necessary.
3. Select a category for the number format from the Category list.
4. Type a number format or select a format type from the Type combo box.

You can type a built-in format or a custom format. Refer to “Creating Custom Number Formats” on page 102 for more information about custom formats.

5. Click OK.

The following table shows the built-in number formats for a US English locale and the result after the format is applied to a positive, negative, and decimal number.

Category and Format	3	-3	.3
General	3	-3	.3
Currency			
\$\$,##0_);(\$\$,##0)	\$3	(\$3)	\$0
\$\$,##0_);[Red](\$\$,##0)	\$3	(\$3) in red	\$0
\$\$,##0.00_);(\$\$,##0.00)	\$3.00	(\$3.00)	\$0.30
\$\$,##0.00_);[Red](\$\$,##0.00)	\$3.00	(\$3.00) in red	\$0.30
\$(* #,##0);(\$* #,##0);_(\$* "-"_)			
:_(@_)	\$ 3	(\$ 3)	\$ 0
\$(* #,##0.00);(\$* #,##0.00);_(\$* "-"??_)			
:_(@_)	\$ 3.00	(\$ 3.00)	\$ 0.30
Fixed			
0	3	-3	0
0.00	3.00	-3.00	0.30
#,##0	3	-3	0
#,##0.00	3.00	-3.00	0.30
#,##0_);(#,##0)	3	(3)	0
#,##0_);[Red](#,##0)	3	(3) in red	0
#,##0.00_);(#,##0.00)	3.00	(3.00)	0.30
#,##0.00_);[Red](#,##0.00)	3.00	(3.00) in red	0.30
\$(* #,##0);(* #,##0);_(* "-"_);:_(@_)	3	(3)	0
\$(* #,##0.00);(* #,##0.00);_(* "-"??_);:_(@_)	3.00	(3.00)	0.30
Percent			
0%	300%	-300%	30%
0.00%	300.00%	-300.00%	30.00%
Fraction			
# ?/?	3	-3	2/7
# ??/??	3	-3	3/10
Scientific			
0.00E+00	3.00E+00	-3.00E+00	3.00E-01
##0.0E+0	300.0E-2	-300.0E-2	300.0E-3

The following table shows the built-in date formats for a US English locale and the result after the format is applied to a date.

Format	04/18/95
m/d/yy	4/18/95
d-mmm-yy	18-Apr-95
d-mmm	18-Apr
mmm-yy	Apr-95
m/d/yy h:mm	4/18/95 0:00

The following table shows the built-in time formats for a US English locale and the result after the format is applied to a time.

Format	12:02:02
h:mm AM/PM	12:02 PM
h:mm:ss AM/PM	12:02:02 PM
h:mm	12:02 PM
h:mm:ss	12:02:02
m/d/yy h:mm	4/18/95 12:02 PM
mm:ss	02:02
[h]:mm:ss	12:02:02
mm:ss.0	02:02.0

Applying Number Formats to Rows and Columns

If you apply a number format to a row or column, that format is applied to all cells in the row or column. When you enter data in a cell in a formatted row or column, the data assumes the designated format.

Formula One allocates memory by rows. Formatting empty rows or columns does not use memory. A format is merely attached to a row or column. Formatting empty ranges is treated differently. If you format a range of empty cells, a group of formatted, empty cells is created. Each new formatted, empty cell consumes memory.

For more information about memory use in Formula One, see chapter 13, “Performance Tuning and Specifications”.

Obtaining Formatted Text Programmatically

You can obtain the formatted text in a cell by retrieving the value of the **FormattedText**, **FormattedTextRC**, or **FormattedTextSRC** properties. These properties return text exactly as it is displayed in the worksheet.

Creating Custom Number Formats

In addition to the built-in number formats, you can define custom formats. Each custom format can have as many as four sections: one for positive numbers, one for negative numbers, one for zeros, and one for text. Each section is optional; the sections are separated by semicolons. The following example shows a custom format.

```
#,###;(#,###);0;"Error: Entry must be numeric"
```

➤ **To define a custom number format in the Workbook Designer:**

1. Select the cells for which you want to create the custom number format.
2. Select Format > Cells and select the Number tab, if necessary.
3. Select a category for the custom number format from the Category list.
4. Type a custom number format built from the custom format characters described later in this chapter in the Type combo box.
5. Click OK.

➤ **To define a custom number format programmatically:**

- Use the **NumberFormat** property. The following code sets **NumberFormat** to format numbers in the current selection with two decimal places and negative numbers with parentheses.

```
Dim Fmt As F1CellFormat
Set Fmt = F1Book1.CreateNewCellFormat
Fmt.NumberFormat = "#,##0.00_);(#,##0.00)"
F1Book1.SetCellFormat Fmt
```

- Use **FormatCellsDlg (F1NumberPage)** to display the Format Cells dialog box. The **Number** tab of this dialog box allows you to select existing formats as well as define custom formats. The selected format is applied to all selections. The following code displays the Format Cells dialog box with the **Number** tab displayed.

```
F1Book1.FormatCellsDlg (F1NumberPage)
```

The following table lists the format symbols that can be used in a custom format string.

Format Symbol	Description
General	Displays the number in General format.
0	Digit placeholder. If the number contains fewer digits than the format contains placeholders, the number is padded with 0's. If there are more digits to the right of the decimal than there are placeholders, the decimal portion is rounded to the number of places specified by the placeholders. If there are more digits to the left of the decimal than there are placeholders, the extra digits are retained.
#	Digit placeholder. This placeholder functions the same as the 0 placeholder except the number is not padded with 0's if the number contains fewer digits than the format contains placeholders.
?	Digit placeholder. This placeholder functions the same as the 0 placeholder except that spaces are used to pad the digits.
. (period)	Decimal point. Determines how many digits (0's or #'s) are displayed on either side of the decimal point. If the format contains only #'s left of the decimal point, numbers less than 1 begin with a decimal point. If the format contains 0's left of the decimal point, numbers less than 1 begin with a 0 left of the decimal point.
%	Displays the number as a percentage. The number is multiplied by 100 and the % character is appended.
, (comma)	Thousands separator. If the format contains commas separated by #'s or 0's, the number is displayed with commas separating thousands. A comma following a placeholder scales the number by a thousand. For example, the format 0, scales the number by 1000 (e.g., 10,000 would be displayed as 10).
E- E+ e- e+	Displays the number as scientific notation. If the format contains a scientific notation symbol to the left of a 0 or # placeholder, the number is displayed in scientific notation and an E or an e is added. The number of 0 and # placeholders to the right of the decimal determines the number of digits in the exponent. E- and e- place a minus sign by negative exponents. E+ and e+ place a minus sign by negative exponents and a plus sign by positive exponents.
\$ - + / () : space	Displays that character. To display a character other than those listed, precede the character with a back slash (\) or enclose the character in double quotation marks (" "). You can also use the slash (/) for fraction formats.
\	Displays the next character. The backslash is not displayed. You can also display a character or string of characters by surrounding the characters with double quotation marks (" "). The backslash is inserted automatically for the following characters: ! ^ & ` (left quote) ' (right quote) ~ { } = < >
* (asterisk)	Repeats the next character until the width of the column is filled. You cannot have more than one asterisk in each format section.

Format Symbol	Description
_ (underline)	Skips the width of the next character. For example, to make negative numbers surrounded by parentheses align with positive numbers, you can include the format _) for positive numbers to skip the width of a parenthesis.
"text"	Displays the text inside the quotation marks.
@	Text placeholder. If there is text in the cell, the text replaces the @ format character.
m	Month number. Displays the month as digits without leading zeros (e.g., 1-12). Can also represent minutes when used with h or hh formats.
mm	Month number. Displays the month as digits with leading zeros (e.g., 01-12). Can also represent minutes when used with the h or hh formats.
mmm	Month abbreviation. Displays the month as an abbreviation (e.g., Jan-Dec).
mmmm	Month name. Displays the month as a full name (e.g., January-December).
d	Day number. Displays the day as digits with no leading zero (e.g., 1-2).
dd	Day number. Displays the day as digits with leading zeros (e.g., 01-02).
ddd	Day abbreviation. Displays the day as an abbreviation (e.g., Sun-Sat).
dddd	Day name. Displays the day as a full name (e.g., Sunday-Saturday).
yy	Year number. Displays the year as a two-digit number (e.g., 00-99).
yyyy	Year number. Displays the year as a four-digit number (e.g., 1900-2078).
g	If you are using a Japanese locale, this displays the Latin letter for an era.
gg	If you are using a Japanese locale, this displays the first character of an era name.
ggg	If you are using a Japanese locale, this displays the full era name.
e	If you are using a Japanese locale, this displays the full era year.
ee	If you are using a Japanese locale, this displays the full era year with a leading 0 if the year is less than 10.
h	Hour number. Displays the hour as a number without leading zeros (e.g., 0-23). If the format contains one of the AM or PM formats, the hour is based on a 12-hour clock. Otherwise, it is based on a 24-hour clock.
hh	Hour number. Displays the hour as a number with leading zeros (e.g., 00-23). If the format contains one of the AM or PM formats, the hour is based on a 12-hour clock. Otherwise, it is based on a 24-hour clock.
m	Minute number. Displays the minute as a number without leading zeros (e.g., 0-59). The m format must appear immediately after the h or hh symbol. Otherwise, it is interpreted as a month number.
mm	Minute number. Displays the minute as a number with leading zeros (e.g., 00-59). The mm format must appear immediately after the h or hh symbol. Otherwise, it is interpreted as a month number.
s	Second number. Displays the second as a number without leading zeros (e.g., 0-59).

Format Symbol	Description
ss	Second number. Displays the second as a number with leading zeros (e.g., 00-59).
AM/PM am/pm A/P a/p	12-hour time. Displays time using a 12-hour clock. Displays AM, am, A, or a for times between midnight and noon; displays PM, pm, P, or p for times from noon until midnight.
[h]	Outputs total number of hours.
[m]	Outputs total number of minutes.
[s]	Outputs total number of seconds.
s.0, s.00, s.000, ss.0, ss.00, ss.000	Outputs fractional part of second.
[Black]	Displays cell text in black.
[Blue]	Displays cell text in blue.
[Cyan]	Displays cell text in cyan.
[Green]	Displays cell text in green.
[Magenta]	Displays cell text in magenta.
[Red]	Displays cell text in red.
[White]	Displays cell text in white.
[Yellow]	Displays cell text in yellow.
[Color <i>n</i>]	Displays cell text using the corresponding color in the color palette. <i>n</i> is a color in the color palette.
[conditional value]	Each format can have as many as four sections: one each for positive numbers, negative numbers, zeros, and text. Using the conditional value brackets ([]), you can designate a different condition for each section. For example, you might want positive numbers displayed in black, negative numbers in red, and zeros in blue. The following string formats a number for these conditions: [>0][Black]General; [<0][Red]General; [Blue]General

The following table shows some examples of custom number formats and numbers displayed using the custom formats.

Format	Cell Data	Display
#.#	123.456	123.46
	0.2	.2
#.0	123.456	123.46
	123	123.0
#,##0"CR";#,##0"DR";0	1234.567	1,235CR
	0	0
	-123.45	123DR
#,	10000	10

Format	Cell Data	Display
"Sales="0.0	123.45	Sales=123.5
	-123.45	-Sales=123.5
"X="0.0;"X=" -0.0	-12.34	x=-12.3
\$* #,##0.00;\$* -#,##0.00	1234.567	\$ 1,234.57
	-12.34	\$ -12.34
000-00-0000	123456789	123-45-6789
"Cust. No." 0000	1234	Cust. No. 1234
:::	Anything	(Not Displayed)
"The End"	123.45	The End
	-123.45	-The End
	text	text
m-d-yy	2/3/94	2-3-94
mm dd yy	2/3/94	02 03 94
mmm d, yy	2/3/94	Feb 3, 94
mmm d, yyyy	2/3/94	February 3, 1994
d mmmm yyyy	2/3/94	3 February 1994
hh"h" mm"m"	1:32 AM	01h 32m
h.mm AM/PM	14:56	2.56 PM
hhmm "hours"	3:15	0315 hours
#?/?	1.25	1 1/4
#?/8	1.25	1 2/8

Formatting Fonts

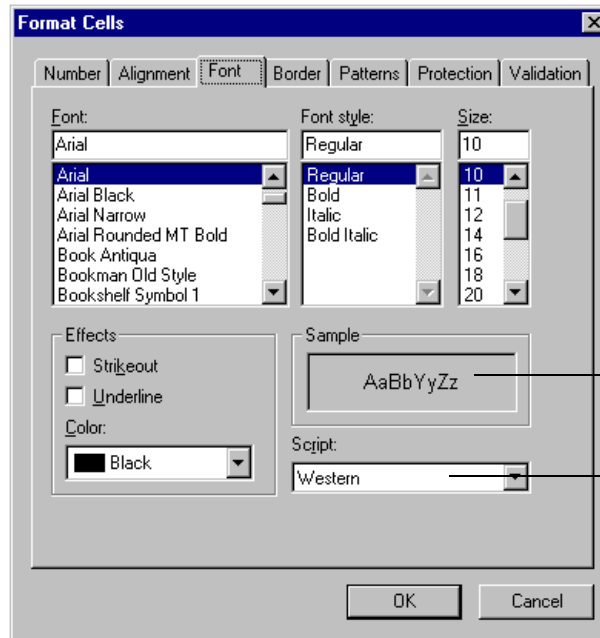
You may choose from many different fonts, font sizes, font styles, and font colors for any cells in your workbook.

You can set a default font, font size, and style that will apply to all cells in all worksheets in the workbook. Later you can change these settings for individual cells. Changes you make to individual cells will remain, even if you change the default settings. For more information on changing the default font, see "Setting the Default Font" on page 46.

Note By default, Formula One uses Arial as the default font. Be sure you always use a TrueType font as the default font in order for print and display scaling to work correctly.

➤ **To set font formats for individual cells using the Workbook Designer:**

1. Select the cells you wish to format.
2. Choose Format > Cells and click the Font tab, shown below.



A text sample with the font, style, and size you chose appears here.

Some fonts provide scripts that allow you to use non-Western alphabets. Choose the script you want here.

3. Select the font format settings you want, and click OK.

➤ **To set font formats for individual cells programmatically:**

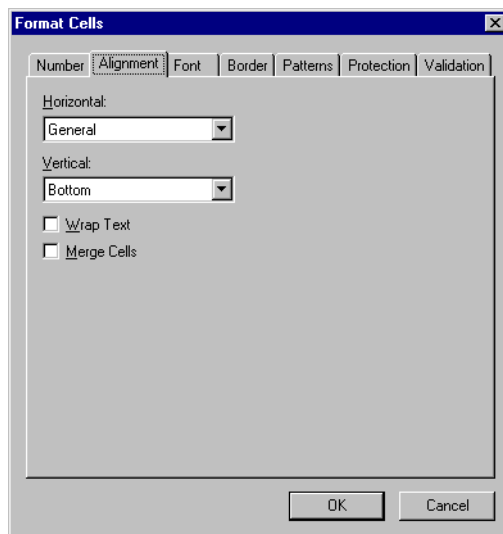
- Use the **SetFontEx** method.

Aligning Data

Formula One allows you to specify how data is aligned within a cell. The standard alignment places text along the left edge of the cell and numbers along the right edge of the cell. Logical and error values are centered.

➤ **To align text in the Workbook Designer:**

1. Select the cells for which you want to align the contents.
2. Select Format > Cells and select the Alignment tab, shown below.



3. Specify the horizontal and vertical alignment of data in the selected cells using the Horizontal and Vertical lists.
4. Select the Wrap Text option to wrap long strings of data to multiple lines within the cell.
5. Click OK.

➤ **To align text programmatically:**

- Use the **AlignHorizontal**, **AlignVertical**, and **WordWrap** properties of the **F1CellFormat** object to set horizontal and vertical alignment and word wrapping for data in the selected cells.

To align the text in the currently selected range(s), you can use the following code:

```
F1CellFormat.AlignHorizontal = F1HAlignCenter 'Centers the text
                                         horizontally
F1CellFormat.AlignVertical = F1VAlignCenter 'Centers the text
                                         vertically
F1CellFormat.WordWrap = FALSE 'Disables word wrapping
```

- Use **FormatCellsDlg** with the **Alignment** tab displayed.

The following code invokes the Format Cells dialog box with the **Alignment** tab displayed.

```
F1Book1.FormatCellsDlg (F1AlignmentPage)
```

Merging Cells

You may merge two or more cells in order to create headings that span many columns or a column entries that span many rows. You can use merged cells to, for example, create a heading for several different columns of data, or to insert a block of text on a worksheet. You may merge cells in the Workbook Designer, or you may use properties and methods.

When you merge cells, Formula One removes the cell borders between the merged cells and replaces any data in the cells with the data in the top left cell in the selection. To include all data in the range in the merged cell, copy all the data into the upper-leftmost cell within the range before merging.

Merged cells function as a single cell on the worksheet, with the row/column reference of the cell in the top-left corner of the range. For example, if you merge cells A1:B5, the resulting cell will have the cell reference A1.

- **To merge cells in the workbook designer:**
 1. Select the cells you want to merge.
 2. Select Format > Cells and select the Alignment tab, shown on page 108.
 3. Select the Merge Cells option.
 4. Click OK.
- **To merge cells using properties and methods:**
 - Use the **MergeCells** property of the **F1CellFormat** object.

Cutting, Copying, and Pasting Merged Cells

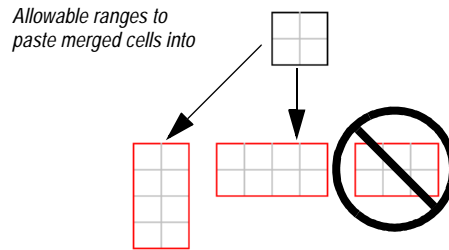
Merged cells behave a little differently than normal cells when it comes to cutting, copying, and pasting. Be aware of the following guidelines when you work with merged cells.

Pasting the entire merged cell range

Whenever you paste a range that contains merged cells, Formula One requires you to paste one or more copies of the entire range. It will not paste portions of the range containing the merged cell(s). This means that you must select a destination range (the range you are pasting into) that is:

- equal to the number of rows and columns in the source range, or
- a multiple of the number of rows and columns in the source range, or
- a single cell.

For example: A range two rows deep and two columns wide that contains merged cells can be pasted into a range four rows deep and two columns wide, or two rows deep and four columns wide. It cannot, however, be pasted into a range two rows deep and three columns wide.



Paste Special with merged cells

Some of the Paste Special command do not work when the source or destination of the paste is a range that contains merged cells. Specifically, you cannot use Paste Special to paste formulas or values to or from a range that contains merged cells.

Changing Row Height and Column Width

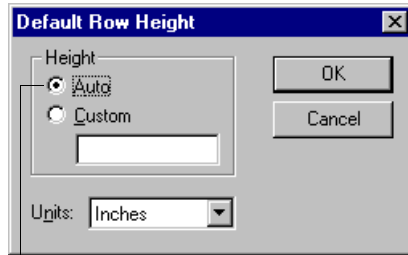
You can set the width of columns and the height of rows using menu commands, click and drag actions, or properties and methods. In addition, Formula One provides commands that allow you to define the default row height and default column width for your worksheet. Column and row sizing can be performed in the Workbook Designer at design time or in a workbook at runtime.

Setting Default Row Height and Width

Formula One provides menu commands that allow you to define the default row height and default column width for your entire worksheet. This section describes how to define these default settings.

➤ **To define the default row height of a worksheet:**

1. Select Format > Row > Default Height to display the Default Row Height dialog box which is shown in the following illustration.

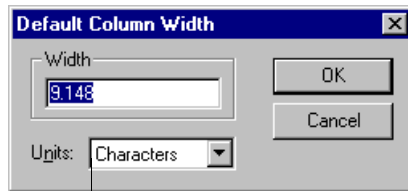


For Formula One to automatically adjust the height of rows based on the values in cells, select the Auto option button.

2. Select the Custom option button define the default height of rows.
3. Enter a custom setting for the row height in the Custom text box.
4. Select whether the custom row height is entered as inches or centimeters from the Units drop-down list.
5. Click OK.

➤ **To define the default column width of a worksheet:**

1. Select Format > Column > Default Width to display the Default Column Width dialog box which is shown in the following illustration.



Select Characters, Centimeters, or Inches for the unit in which the width is entered.

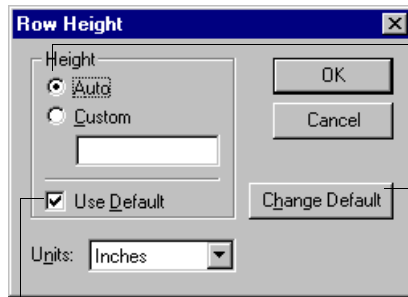
2. Enter a default width for columns and select a unit from the Units drop-down list.
3. Click OK.

Sizing Rows and Columns Using Menu Commands

You can set the width of selected columns and the height of selected rows using menu commands. This section describes the commands that allow you to perform these functions.

➤ **To set the row height of a selection:**

1. Select the rows for which you want to set the height.
2. Select Format > Row > Height to display the Row Height dialog box which is shown in the following illustration.



For Formula One to automatically adjust the height of rows based on the values in cells, select the Auto option button.

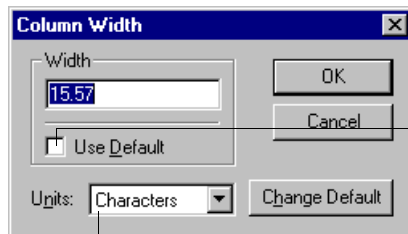
Click this button to display the Default Row Height dialog box.

To use the default settings, check this check box.

3. Select the Custom option button define a custom row height for the selected rows.
4. Enter a custom setting for the row height in the Custom text box.
5. Select whether the custom row height is entered as inches or centimeters from the Units drop-down list.
6. Click OK.

➤ **To set the column width of a selection:**

1. Select Format > Column > Width to display the Column Width dialog box which is shown in the following illustration.



To use the default settings, check this check box.

Select Characters, Centimeters, or Inches for the unit in which the width is entered.

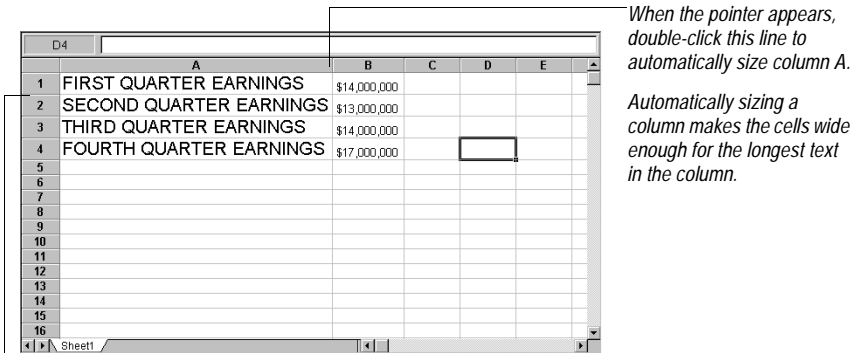
2. Enter a custom width for the selected columns and select a unit from the Units drop-down list.
3. Click OK.

Sizing Rows and Columns Using Click and Drag Actions

When you position the pointer on the right edge of a column heading or the bottom edge of a row heading, the pointer changes to a double arrow to indicate that the row or column can be resized. Simply click and drag to resize the column or row.

If multiple rows are selected when you resize a row, all selected rows are resized as you drag a row border. Multiple columns can be resized in the same manner.

Double-click the bottom border of a row heading to automatically adjust the height of all cells in the row to accommodate the largest font size in the row. Double-click the right side of a column heading to automatically adjust the width of all cells in the column to accommodate the largest entry. This is shown in the following illustration.



When the pointer appears, double-click this line to automatically size row 1.
Automatically sizing a row makes the cells tall enough for the largest text in the row.

When the pointer appears, double-click this line to automatically size column A.
Automatically sizing a column makes the cells wide enough for the longest text in the column.

You can also set the size of a selected group of columns or rows to match the size of an existing row or column. First, select the group of rows or columns you want to resize, including the row or column whose size you want to match. Then, click the right border of the column heading or the bottom border of the row whose size you want to match. The selected rows are resized to match the size of the row or column you clicked.

Note You can disable interactive sizing of rows and columns by setting the **AllowResize** property to False.

Sizing Rows and Columns with Properties and Methods

The following table lists the properties and methods that allow you to size rows and columns.

Property/Method	Operation
ColHidden	Sets or returns the display status of an individual column.
ColWidth	Sets or returns the width of a single column in units of 1/256 of a average character's width in the default font or twips (1/1440th of an inch) depending on the setting of ColWidthUnits .
ColWidthDlg	Displays the Column Width dialog box.
ColWidthTwips	Sets or returns the width of the specified columns in twips.

Property/Method	Operation
ColWidthUnits	Sets or returns whether column widths are stored and displayed in twips or character units.
SetColWidth	Sets the width of the specified columns in units of 1/256 of an average character's width in the default font or twips (1/1440th of an inch) depending on the setting of ColWidthUnits .
SetColWidthAuto	Automatically sets the width of the specified columns to accommodate the largest data in the column.
SetColWidthTwips	Changes the width of one or more columns to the specified number of twips.
RowHeight	Sets or returns the height of a single row in twips.
RowHidden	Sets or returns the display status of an individual row.
SetRowHeight	Sets the height of the specified rows in twips (one twip equals 1/1440 inch).
SetRowHeightAuto	Automatically sets the height of the specified rows to accommodate the tallest data in the row.
RowHeightDlg	Displays the Row Height dialog box.

SetRowHeight and **SetColWidth** set the size of one or more rows or columns. For example, the following code sets the height of rows 1 through 10 to 1/2 inch, and the width of columns 1 through 10 (A through J) to 10 characters wide.

```
F1Book1.SetRowHeight 1, 10, 720, FALSE
F1Book1.SetColWidth 1, 10, 2560, FALSE
```

SetColWidthAuto and **SetRowHeightAuto** automatically size rows and columns to accommodate the largest data in the row or column. For example, the following code automatically sets the row and column sizes of rows 1 through 10, and columns 1 through 10 (A through J).

```
F1Book1.SetRowHeightAuto 1, 1, 10, 10, TRUE
F1Book1.SetColWidthAuto 1, 1, 10, 10, TRUE
```


Freezing Horizontal and Vertical Panes

To scroll through your worksheet and see designated headings for columns or rows, split the worksheet into panes by “freezing” them. The following illustration shows a worksheet with frozen panes.

	A	B	C	D	E	F	
1			1993				
2			Q1	Q2	Q3	Q4	
3	Eastern	New England	1449.04	4573.43	5681.87	4085.19	
4		Mid Atlantic	1827.12	3748.45	8042.25	8049.24	
5		Southeast	1772.45	2097.15	7948.42	2553.15	
6	Central	Great Lakes	8398.15	2626.89	6117.96	4480.14	
7		Midwest	5937.66	3671.86	7470.59	2623.64	
8		Texas	8227.45	4167.80	8104.55	6783.30	
9	Western	Rockies	5647.47	4190.54	5779.47	9674.10	
10		Southwest	1898.54	3919.69	7348.04	3366.32	
11		Pacific NW	2258.25	8321.59	1205.87	5376.54	
12			43,137.00	42,358.00	43,367.42	52,311.10	

Columns A and B are frozen in this worksheet. When horizontal scrolling is performed, columns A and B do not scroll.

Before horizontal scrolling

After horizontal scrolling

Data contained in frozen panes cannot be edited. You must perform any data editing in these panes prior to freezing. If you attempt to select a cell in a frozen row or column, the entire row or column is selected, just as if you selected a row or column heading. This is shown in the following illustration.

	A	B	G	H	I	J	
1			1994				
2			Q1	Q2	Q3	Q4	
3	Eastern	New England	4085.19	9956.62	4949.14	6232.05	
4		Mid Atlantic	8049.24	3573.06	2596.09	7505.63	
5		Southeast	2553.15	2482.89	1029.15	6607.76	
6	Central	Great Lakes	4480.14	2073.95	4884.52	4141.76	
7		Midwest	5937.66	3671.86	7470.59	2623.64	
8		Texas	8227.45	4167.80	8104.55	6783.30	
9	Western	Rockies	5647.47	4190.54	5779.47	9674.10	
10		Southwest	1898.54	3919.69	7348.04	3366.32	
11		Pacific NW	2258.25	8321.59	1205.87	5376.54	
12			43,137.00	42,358.00	43,367.42	52,311.10	

Data in frozen rows and columns cannot be edited.

When you attempt to select a cell in a frozen row or column, the entire row or column is selected.

► To freeze horizontal panes in the Workbook Designer:

1. Select a cell in the row below where you want to split the panes.
2. Select Format > Freeze Panes.

The rows above the split are frozen.

➤ **To freeze vertical panes in the Workbook Designer:**

1. Select a cell in the column to the right of where you want to split the panes.
2. Select Format > Freeze Panes.
The columns to the left of the split are frozen.

➤ **To freeze panes using the Format Sheet dialog box in the Workbook Designer:**

1. Select Format > Sheet > Properties and select the View tab.
2. Enter a range for the cells that you want to freeze in the Fixed Rows and Fixed Columns text boxes.

For example, to freeze columns A - B and rows 1- 4 enter \$1:\$4 in the Fixed Rows text box and \$A:\$B in the Fixed Columns text box.

3. Click OK.

➤ **To freeze rows or columns programmatically:**

- Setting the **FixedRow** and **FixedCol** properties specify the starting row and column to freeze. The **FixedRows** and **FixedCols** properties determine how many rows or columns are frozen from the first row or column. The following example freezes A1:A4.

```
F1Book1.FixedRow = 1  
F1Book1.FixedCol = 1  
F1Book1.FixedCols = 1  
F1Book1.FixedRows = 4
```

Setting Cell Border and Fill Formats

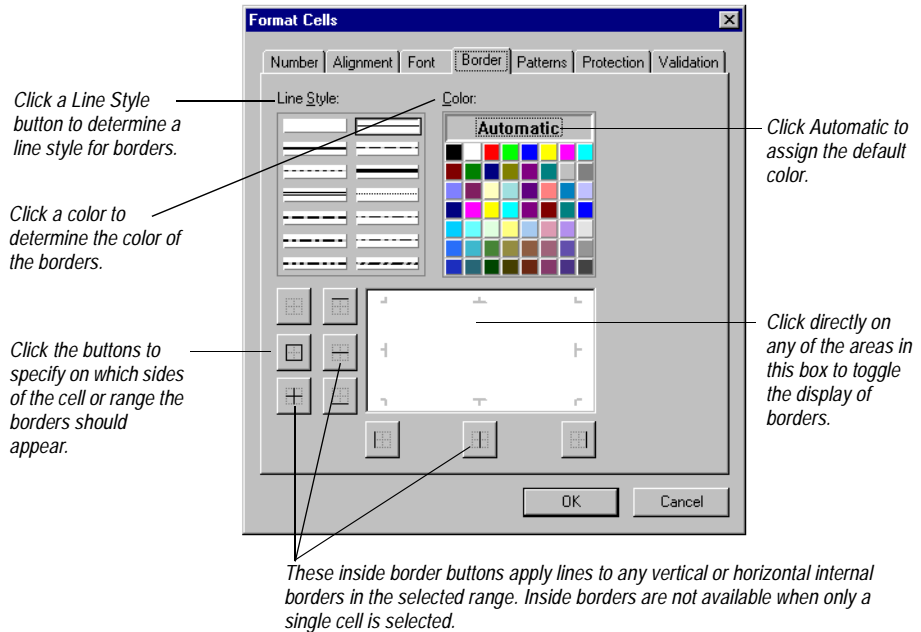
Cells and ranges can be formatted with borders of different outlines, line styles, and colors. In addition, cells and ranges can be formatted with colors and patterns. The following sections describe how to apply these attributes to cells.

Setting Cell Borders

Borders can be applied to the top, bottom, left, and right sides of a cell. When you add a border to a range, you can place a border around the outside of the range. You can set borders using the Workbook Designer or programmatically using properties and methods. To programmatically set formatting for cell borders, use the **F1CellFormat** API object.

➤ **To format cells with borders using the Workbook Designer:**

1. Select the cells you want to format.
2. Select Format > Cells and select the Border tab from the Format Cells dialog box. The following illustration shows an example of the Border tab.



Note You must select line style and color before selecting the location of the borders.

3. Select borders using the instructions in the previous illustration.
4. Click OK.

➤ **To format cell borders programmatically:**

- Use the **F1CellFormat** API object, **BorderColor** and **BorderStyle** properties, **CreateNewCellFormat** method, and the **SetCellFormat** method to create border styles for a cell or range.

The following example selects a range and places a thick light green border around the range.

```
Dim cellfrmt As F1CellFormat
Set cellfrmt = F1Book6.CreateNewCellFormat
'thick border around the selection
cellfrmt.BorderStyle(F1RightBorder) = F1BorderThick
cellfrmt.BorderStyle(F1LeftBorder) = F1BorderThick
cellfrmt.BorderStyle(F1TopBorder) = F1BorderThick
cellfrmt.BorderStyle(F1BottomBorder) = F1BorderThick
'green border around the selection
```

```

cellfmt.BorderColor(F1RightBorder) = QBColor(10)
cellfmt.BorderColor(F1LeftBorder) = QBColor(10)
cellfmt.BorderColor(F1TopBorder) = QBColor(10)
cellfmt.BorderColor(F1BottomBorder) = QBColor(10)
F1Book6.SetSelection 1, 1, 3, 3
F1Book6.SetCellFormat cellfmt

```

- Use **FormatCellsDlg** to invoke the Format Cells dialog box. The following code displays this dialog box with the **Border** tab displayed:

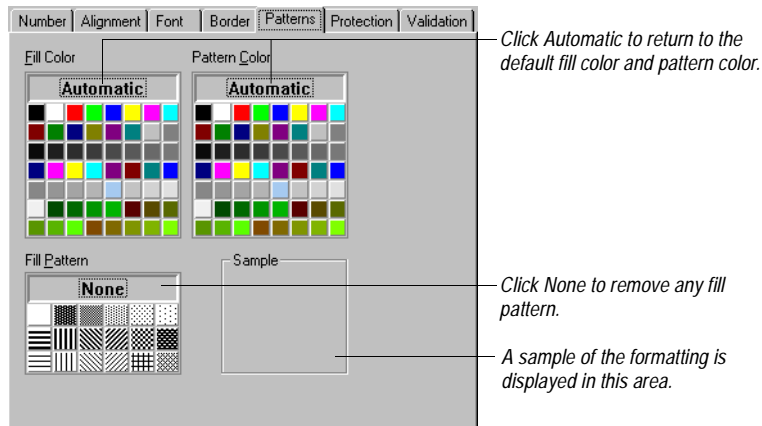
```
F1Book1.FormatCellsDlg (F1BorderPage)
```

Setting Cell Fill Colors and Patterns

When you apply colors and patterns to a cell or range, you specify the pattern and foreground and background colors used to fill the cells. These attributes can be set using the Workbook Designer or via code using properties and methods.

➤ To format cells with colors and patterns using the Workbook Designer:

1. Select the cells you want to format.
2. Select Format > Cells and select the Patterns tab from the Format Cells dialog box. The following illustration shows an example of the Patterns tab.



3. Click a fill color in the Fill Color palette to select it.
4. Click a pattern color in the Pattern Color palette to select it.
5. Click a pattern style in the Fill Pattern palette to select it.
6. Click OK.

➤ To format cells with colors and patterns programmatically:

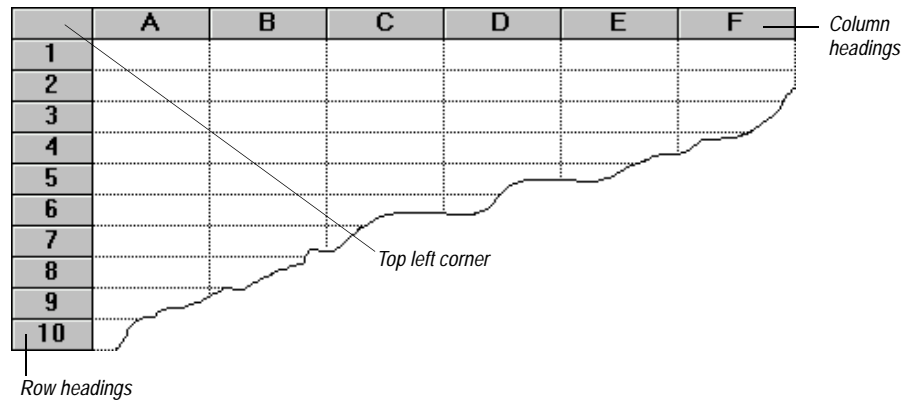
- Use **PatternBG**, **PatternFG**, and **PatternStyle** properties of the **F1CellFormat** object to choose the background and foreground colors and pattern style for the selected cells.

- Use **FormatCellsDlg** to invoke the Format Cells dialog box. The following code displays this dialog box with the **Patterns** tab displayed:

```
F1Book1.FormatCellsDlg (F1PatternsPage)
```

Formatting Row and Column Headings

In addition to formatting worksheet cells, many aspects of row and column headings can be formatted. Worksheet headings contain three areas: the row headings, column headings, and the box in the top left corner of the worksheet where the row and column headings intersect. The following illustration highlights these three areas.



Selecting Row and Column Heading Areas

Row and column headings can be selected interactively or programmatically.

➤ To select headings interactively:

- Press CTRL+Shift and click the heading area.

➤ To select a heading area programmatically:

- Use **SetHdrSelection**. The following code selects the column heading area.

```
F1Book1.SetHdrSelection FALSE, FALSE, TRUE
```

- Set the **SelHdrRow**, **SelHdrCol**, and **SelHdrTopLeft** properties to true. The following code selects the top left gray rectangle in the worksheet where the rows and columns intersect.

```
F1Book1.SelHdrTopLeft = TRUE
```

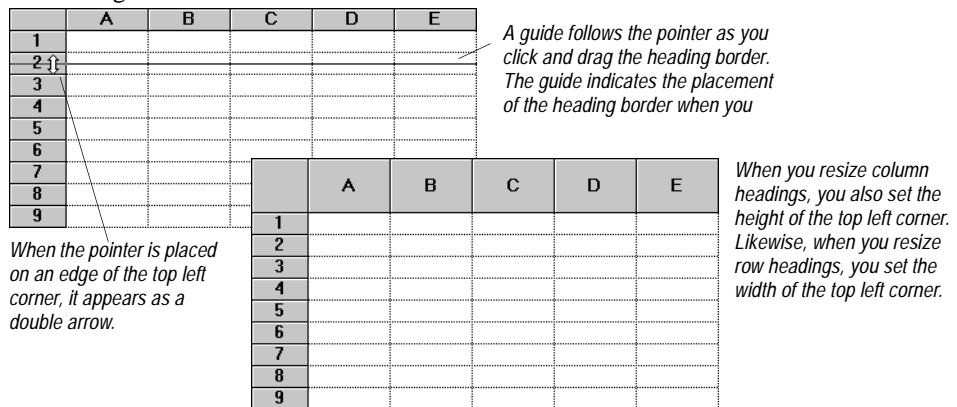
After a heading area is selected, you can set:

- the alignment of the heading text

- the font and color of the heading text
- the pattern and fill color of the heading area
- the border used to frame heading cells

Sizing Row and Column Headings

The size of row, column, and top left headings can be set interactively and programmatically. The following illustration shows interactive resizing of the column headings.



- **To interactively change the size of column headings:**
 - Click and drag the bottom edge of the top left corner.
- **To interactively change the size of row headings:**
 - Click and drag the right edge of the top left corner.
- **To change the size of column headings programmatically:**
 - Use **HdrHeight**.
- **To change the size of row headings programmatically:**
 - Use **HdrWidth**.

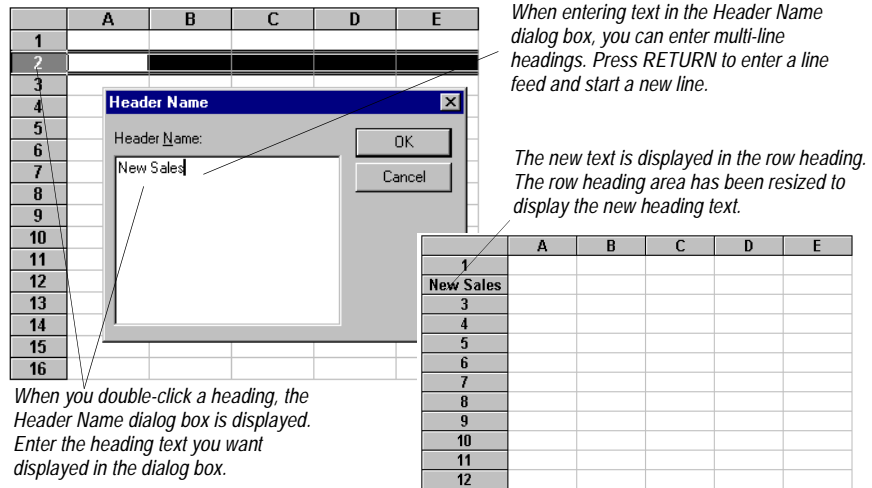
Setting Row and Column Heading Text

Like other column and row heading attributes, the text displayed in heading cells can be changed interactively or programmatically.

- **To interactively change the text for a row or column heading:**
 1. Double-click the heading for which you want to enter text to display the Header Name dialog box.

2. Enter one or more lines of text to serve as the heading name.
3. Click OK.

The heading text is displayed, as shown in the following illustration



You can control interactive editing of headings by setting the **AllowEditHeaders** property. You can return the value of **AllowEditHeaders** to determine whether interactively editing headings is allowed.

► **To set or return the text displayed in column headings programmatically:**

- Use **ColText**. Use **RowText** to set or return the text displayed in row headings. **TopLeftText** sets or returns the text displayed in the top left corner.

With **ColText** and **RowText**, you must specify the column or row for which you are setting heading text. The following example sets the heading for column 4 to "Orders" instead of the default "C".

```
F1Book1.ColText (3) = "Orders"
```

The following illustration shows the result of the example code.

	A	B	Orders	D	E
1					
2					
3					
4					
5					
6					
7					

The heading text for column C is replaced with "Orders" by the **ColText** property.

Note Rows, columns, and cells retain their default numbers and letters in functions, properties, and formulas even if the heading text for rows and columns is changed. For example, the cell at the intersection of column B and row 2 is still referred to as B2 even if the heading text for row 2 has been set to "New Sales."

Tidestone

CHAPTER 8

Working With Graphical Objects

Formula One provides the ability to create graphical objects in a worksheet. Among the graphical objects you can create are lines, rectangles, ovals, arcs, polygons, buttons, check boxes, dropdown list boxes, and charts. As with other worksheet elements, Formula One provides a wide range of options for formatting and manipulating the appearance of graphical objects you create.

For specific information about creating and formatting charts, refer to “Working With Chart Objects” in this manual.

Note Formula One’s graphical objects are incompatible with graphical objects in Microsoft Excel 95 and 97. You may still read and write Excel worksheets, but any graphical objects on those worksheets will not be transferred with the file. If you read or write an Excel file and save the file, any graphical objects that appeared in the original file will be lost.

Creating Graphical Objects

Formula One provides several methods for creating graphical objects in worksheets:

- Graphical objects can be created by calling methods in your code.
- The Workbook Designer allows graphical objects to be created interactively.
- You can set the mode of the mouse to allow graphical objects to be created at any time.

The following sections describe each of these methods for creating graphical objects.

Creating Graphical Objects with Methods

Formula One provides several methods that allow you to create graphical objects from your application code.

- Use **ObjNew** or **ObjCreate** to create lines, rectangles, ovals, arcs, buttons, check boxes, dropdown list boxes, and charts.

- **Use ObjNewPicture or ObjCreatePicture** to create a new metafile picture object on a worksheet.

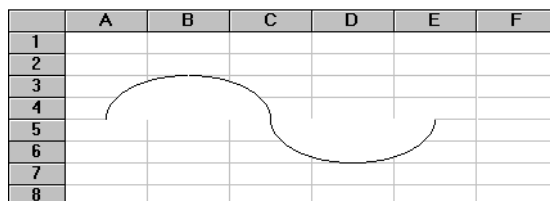
The following example creates four adjoining arcs using the **ObjCreate** method:

```
Dim pid As Long
pid = F1Book1.ObjCreate(F1ObjArc, 0.5, 2, 1.5, 4)
pid = F1Book1.ObjCreate(F1ObjArc, 2.5, 2, 1.5, 4)
pid = F1Book1.ObjCreate(F1ObjArc, 2.5, 6, 3.5, 4)
pid = F1Book1.ObjCreate(F1ObjArc, 4.5, 6, 3.5, 4)
```

The following example creates four adjoining arcs using the **ObjNew** method:





```
Dim objID1 As Long
Dim objID2 As Long
Dim objID3 As Long
Dim objID4 As Long
F1Book1.ObjNew F1ObjArc, .5, 2, 1.5, 4, objID1
F1Book1.ObjNew F1ObjArc, 2.5, 2, 1.5, 4, objID2
F1Book1.ObjNew F1ObjArc, 2.5, 6, 3.5, 4, objID3
F1Book1.ObjNew F1ObjArc, 4.5, 6, 3.5, 4, objID4
```







The following illustration shows the result of the example code.



Interactively Drawing Graphical Objects

The Workbook Designer toolbar contains tools that allow you to interactively create and edit graphical objects. The following table describes the tools.

Button	Name	Description
	Polygon point editing tool	Toggles between normal polygon editing and polygon point editing. When polygon point editing is enabled, use this tool to move the points of a polygon.
	Line tool	Draws lines.
	Rectangle tool	Draws rectangles and squares.
	Oval tool	Draws ovals and circles.

Button	Name	Description
	Arc tool	Draws arcs.
	Polygon tool	Draws polygons.
	Button tool	Draws buttons.
	Check box tool	Draws check boxes.
	Dropdown list box tool	Draws dropdown list boxes.
	Chart tool	If Tidestone's First Impression charting software is installed on your computer, this button accesses First Impression to create a chart based on the selected range of data.

Interactively drawing graphical objects in the Workbook Designer is as simple as point, click, and drag.

► **To create a graphical object in the Workbook Designer:**

1. Click the tool for the graphical object you want to create.
The pointer appears as a small cross when positioned in the worksheet.
2. Position the pointer at the point where you want to begin drawing.
3. Click and drag to create the graphical object.
An outline of the graphical object you are creating appears and moves as you drag the mouse.
4. Release the mouse button to set the graphical object in place.

Note When creating a graphical object, press ALT to align the graphical object to the cell grid.

Picture Objects

You can place a picture object on a worksheet and fill it with a metafile. Formula One provides two methods for doing this: one for drawing the picture object, and one for filling it with a metafile.

- Use **ObjNewPicture** or **ObjCreatePicture** to create a picture object on the current worksheet. You must specify the position for the new picture object.

When specifying the location of the picture object, integers place the edge of the picture object on a row or column border; fractional numbers place the edge of the picture object between borders.

- Use **ObjSetPicture** to place a metafile in an existing graphical object. You must provide a handle to the metafile and the ID number of the picture object into which you want the metafile placed. Any metafile previously contained by the picture object is freed from memory.

These methods also pass information about the dimensions of the picture and whether the picture can be stretched. Formula One manages the memory associated with a metafile once a picture object has been created, including freeing memory when the graphical object is deleted.

You should be familiar with Windows metafiles and their structure before using these methods.

Setting Mouse Mode

By setting the mode for mouse actions, you can allow graphical objects to be drawn interactively in a worksheet.

- Set the **Mode** property to change the mouse mode. You can specify that the mouse draw charts, lines, rectangles, ovals, arcs, buttons, polygons, check boxes, and dropdown list boxes. You can also specify that the mouse assume normal worksheet editing mode.
- Return the value of the **Mode** property to return the current mouse mode.

The following example uses **Mode** to set the mouse mode to rectangle drawing when you double-click a worksheet.

```
Private Sub F1Book1_DblClick(ByVal nRow As Long, ByVal nCol As Long)
    F1Book1.Mode = F1ModeRectangle
End Sub
```

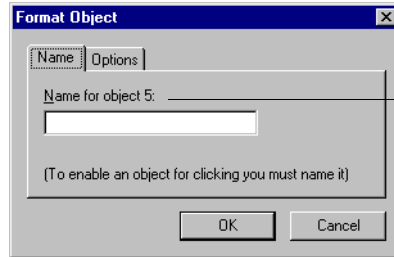
Identifying Graphical Objects

When you create a graphical object – whether by method, in the Workbook Designer, or by setting the mouse mode – the graphical object is assigned an identification number. Many methods or properties require a graphical object identification number to tell Formula One on which graphical object to operate.

If you are uncertain of a graphical object's identification number, there are several ways to determine it.

➤ **To determine a graphical object's number in the Workbook Designer:**

1. Select the graphical object for which you want to determine the identification number.
2. Select Format > Object to display the Format Object dialog box.
3. Select the Name tab. The Name tab displays the identification number of the selected graphical object, as shown in the following illustration.



The Name tab displays the identification number for the selected graphical object.

4. Click OK.

➤ **To determine a graphical object's number programmatically, use one of the following properties or methods:**

- If a graphical object is selected, the **ObjGetSelection** method returns the identification number of the selected object.
- Formula One maintains a list of graphical objects in each worksheet within a workbook. The order of the graphical objects in that list is determined by the order in which graphical objects are drawn in the worksheet. The farther to the back a graphical object is placed or drawn, the higher the graphical object is placed in the list; the closer to the front a graphical object is placed or drawn, the lower the graphical object is placed in the list. Therefore, when you call **ObjFirstID**, this method returns the identification number of the first graphical object in the list, which is the graphical object created farthest back in the layers of worksheet graphical objects. Then, **ObjNextID** returns the identification number of the next closest graphical object, and so on.

Note When you use the **ObjBringToFront** and **ObjSendToBack** methods or the Bring To Front and Send To Back commands from the Format menu in the Workbook Designer, you alter the order of the graphical object list maintained by the worksheet.

- If an graphical object has been named, you can return the identification number of the named graphical object with the **ObjNameToID** method.

Naming Graphical Objects

As mentioned in the previous section, graphical objects can be named after they are created. Graphical object names do not take the place of graphical object identification numbers. Rather, graphical object names are used as a supplement to identification numbers, making it easier to track and manipulate graphical objects.

Another purpose for naming graphical objects is to enable the **ObjClick**, **ObjDbtClick**, **ObjGotFocus**, **ObjLostFocus**, and **ObjValueChanged** events. Before these events can fire, you must name the graphical object receiving the action.

Graphical objects can be named in the Workbook Designer or programmatically.

➤ **To name a graphical object in the Worksheet Designer:**

1. Select the graphical object you want to name.
2. Select **Format > Object** to display the **Format Object** dialog box.
3. Click the **Name** tab.
4. Enter the name to assign to the graphical object in the **Name for Object** text box.

➤ **To name a graphical object programmatically:**

- Set the **ObjName** property to name a graphical object. Once a graphical object is named, return the value of **ObjName** to obtain the name assigned to a graphical object. You can also call **FormatObjectDlg** (F1NamePage) to display the **Format Object** dialog box with the **Name** tab displayed.

Note After a graphical object is named, you must press **CTRL** to select it at runtime. In addition, if the **ObjClick** or **ObjDbtClick** events are enabled, you must press **CTRL** when clicking the graphical object.

Selecting Graphical Objects

When working in the Workbook Designer or in a workbook at runtime, selecting graphical objects is an important first step when you are formatting, moving, or resizing graphical objects. In addition, many properties and methods – such as **SetPattern**, **SetLineStyle**, **ObjPatternStyle**, **ObjPatternFG**, **ObjPatternBG**, **ObjBringToFront**, and **ObjSendToBack** – require that a graphical object be selected for the property or method to work.

Interactively Selecting Graphical Objects

Selecting graphical objects interactively in the Workbook Designer or in a worksheet at runtime is as simple as pointing and clicking.

➤ **To interactively select a graphical object:**

1. Position the pointer on the graphical object you want to select.

The pointer appears as an arrow when positioned on a graphical object.

2. Click the graphical object.

When the graphical object is selected, selection handles appear at the edges of the bounding box that surrounds the graphical object.

Note For check boxes, dropdown list boxes, and buttons that have been named or if the **ObjClick** or **ObjDbClick** events are enabled, you must press CTRL when selecting the graphical object. In addition, in a worksheet at runtime, you must press CTRL to select arcs, lines, ovals, polygons, and rectangles that have been named.

To select multiple graphical objects, press SHIFT as you select the graphical objects. For graphical objects that have been named, you must press CTRL+SHIFT to select multiple graphical objects.

In the Workbook Designer, you can choose Edit > Select All Objects to select all the graphical objects in an active worksheet.

Limiting Interactive Graphical Object Selection

Setting the **AllowObjSelections** property allows you to specify whether users can interactively select graphical objects in the Workbook Designer or in a worksheet at runtime. Return the value of **AllowObjSelections** to determine if graphical object selections are allowed.

Selecting Graphical Objects Programmatically

➤ **To select graphical objects programmatically, use one or more of the following:**

- **ObjSetSelection** selects the graphical object you specify by graphical object identification number. This method unselects any previously selected graphical objects or worksheet ranges.
- **ObjGetSelection** returns the identification number of the selected object. If more than one object is selected, you must indicate for which object to return an identification number.
- **ObjAddSelection** selects the specified graphical object; all previously selected objects remain selected.

- **ObjSelection** returns the identification number of the selected graphical object. If more than one graphical object is selected, you must indicate for which graphical object to return an identification number.
- **ObjGetSelectionCount** returns the number of graphical objects currently selected.

Moving, Sizing, and Arranging Graphical Objects

After a graphical object is created, you can change both the position and size of the graphical object. Formula One allows these changes to be made interactively or programmatically.

Interactively Moving and Sizing Graphical Objects

Interactively, graphical objects can be moved and resized at runtime or in the Workbook Designer.

➤ **To interactively move and size a graphical object:**

1. Select the graphical object to be moved or sized.

Note To select a named button, check box, or dropdown list box, you must press CTRL when selecting the graphical object. Or, if the **ObjClick** or **ObjDbClick** events are enabled, you must press CTRL when clicking the graphical object.

When you select a graphical object, selection handles appear along the bounding box that surrounds the graphical object.

2. If moving the graphical object, position the pointer anywhere in the area occupied by the graphical object. If resizing the graphical object, position the pointer on one of the selection handles.

When positioned in the graphical object area, the pointer appears as an arrow. When positioned on a selection handle, the pointer appears as a two-headed arrow, indicating the direction in which the graphical object can be resized.

3. Click and drag to move or resize the graphical object.

An outline of the graphical object moves with the pointer as you drag the mouse.

Note When moving or resizing a graphical object, you can align it to the worksheet grid by pressing ALT as you click and drag.

4. Release the mouse to set the graphical object at its new position or at its new size.

Positioning Graphical Objects Programmatically

Use the **ObjSetPos** method to set the position of a graphical object. Because the **EndCol**, **EndRow**, **StartCol**, and **StartRow** properties of the **F1ObjPos** API object and the **ObjSetPos** method set the placement of both anchor points of the bounding box that surrounds a graphical object, the graphical objects are also resized.

If you are uncertain of an object's location, use the **F1ObjPos** API object or the **ObjGetPos** method to return the position of a graphical object. The following examples use **ObjGetPos** and **F1ObjPos** to obtain the position of the object. Then, **ObjSetPos** resizes the object by moving the second anchor point left one and a half columns and up five rows.

ObjGetPos and ObjSetPos Example:

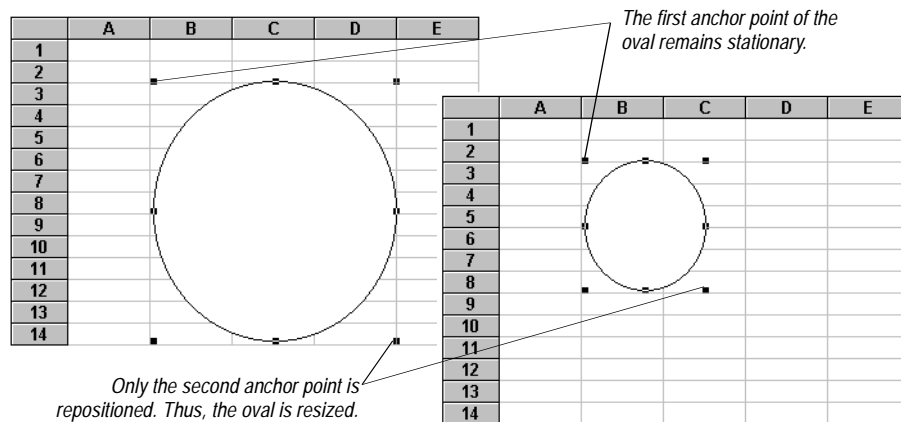
```
Dim x1 As Single
Dim y1 As Single
Dim x2 As Single
Dim y2 As Single
```

```
F1Book1.ObjGetPos 1, x1, y1, x2, y2
F1Book1.ObjSetPos 1, x1, y1, (x2 - 1.5), (y2 - 5)
```

F1ObjPos and ObjSetPos Example:

```
Dim id As Long
Dim F1ObjPos As Single
Dim position As F1ObjPos
id = F1Book1.ObjFirstID
Set position = F1Book1.ObjGetPosEx(id)
F1Book1.Text = position.startcol
F1Book1.Text = position.startrow
F1Book1.Text = position.endcol
F1Book1.Text = position.endrow
F1Book1.Text = position.Cols
F1Book1.Text = position.Rows
F1Book1.ObjSetPos id, position.startcol, position.startrow,
    (position.endcol - 1.5), (position.endrow - 5)
```

The following illustration shows the results of these examples.



You can also use the **ObjPosShown** method to return the graphical object's display status.

Determining Graphical Object Position and Size

The **ObjPosToTwipsEx** and **ObjPosToTwips** methods return the placement and size of a graphical object in twips. For these methods, you provide the position of a graphical object's anchor points in relation to the rows and columns of a worksheet. They also determine whether the graphical object is shown, partially shown, or hidden given the current dimensions of the view.

These methods do not reference a specific graphical object on a worksheet and have no effect on any graphical objects.

Arranging Overlapping Graphical Objects

When you have multiple graphical objects on a worksheet, they appear to be drawn on the same plane. However, when two graphical objects overlap, the previously drawn object is covered by the latter-drawn object. You can change the order of graphical object layering in a worksheet by sending a graphical object behind other graphical objects or bringing a graphical object to the front of other graphical objects.

➤ To arrange graphical objects in the Workbook Designer:

1. Select the graphical object you want to move.
2. Select **Format > Bring to Front** or **Format > Send to Back** to move the graphical object to the direction you specify.

➤ To arrange graphical objects programmatically:

- Call the methods **ObjBringToFront** or **ObjSendToBack** to arrange graphical objects. These methods move only the selected graphical objects.

If multiple graphical objects are selected when using these commands and methods, the order of graphical objects within the selection remains unchanged. The selected graphical objects are placed in front of or behind only the unselected graphical objects.

Arranging Graphical Object Order

When you use the **ObjBringToFront** and **ObjSendToBack** methods or the Format > Bring To Front and Format > Send To Back commands in the Workbook Designer, you alter the order of the graphical object list maintained by the worksheet.

Refer to “Identifying Graphical Objects” on page 126 for information about how this affects the identification of graphical objects.

Formatting Graphical Objects

Many elements of graphical objects can be formatted including:

- fill patterns and colors; line colors; and widths and styles for arcs, lines, ovals, polygons, and rectangles
- the lists of items contained by dropdown list boxes
- the text displayed by check boxes or buttons

Formatting Colors and Patterns

To set the fill colors and patterns for graphical objects – arcs, ovals, polygons, and rectangles – use the following methods:

- **To set fill colors and patterns for selected graphical objects in the Workbook Designer:**
 1. Select the graphical object.
 2. Select Format > Object to display the Format Object dialog box.
 3. Select the Patterns tab.
 4. Click a fill color in the Fill Color palette. This assigns a background color to the graphical object.
 5. If you want to assign a pattern to the object, click a pattern style in the Fill Pattern palette, then click a pattern color in the Pattern Color palette. The pattern color will be the foreground color in the pattern.
 6. Click OK.
- **To set fill colors and patterns for selected graphical object programmatically:**
 - Use **SetPattern** to set the fill colors and pattern for the selected objects.
 - Use **ObjPatternStyle**, **ObjPatternBG**, and **ObjPatternFG** to set the fill colors and pattern for the selected graphical objects.

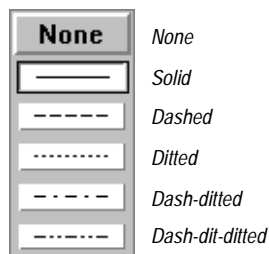
- Call the **FormatObjectDlg** (F1PatternsPage) to display the Format Object dialog box with the **Patterns** tab displayed.

Formatting Lines (Borders) on Graphical Objects

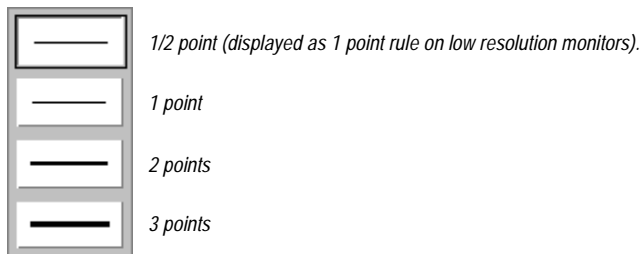
To set the line style, color, and weight for line graphical objects and the lines that form the borders of arcs, ovals, polygons, and rectangles, use the following approaches.

► To format lines for selected graphical objects in the Workbook Designer:

1. Select the graphical object.
2. Select Format > Object to display the Format Object dialog box.
3. Select the Line Style tab.
4. Click a line style or click None to not specify a line style. Line styles can be applied to line objects and the borders of arcs, ovals, polygons, and rectangles. The line styles are shown in the following illustration:



5. Click a line weight. Solid lines can be 1/2 point, 1 point, 2 points, or 3 points in weight. Styled lines can be 1/2 point in weight. The line weights are shown in the following illustration:



6. Click a color in the Color palette. This assigns a color to the line.
7. Click OK.

- **To format lines for selected graphical objects programmatically:**
 - Use **SetLineStyle** to set the line style, color, and weight for the selected objects.
 - Use the **LineStyle**, **LineColor**, and **LineWeight** properties to set the line style, color, and weight for the selected graphical objects.
 - Call **FormatObjectDlg** (F1LineStylePage) to display the Format Object dialog box with the **Line Style** tab displayed.

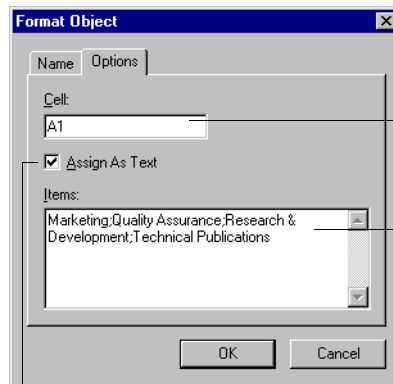
Showing and Hiding Graphical Objects

A graphical object on a worksheet can be hidden by setting the **ObjVisible** property to False. To display a hidden graphical object, set the property to True. If you are uncertain whether a graphical object is shown or hidden, return the value of **ObjVisible**.

Formatting Dropdown List Boxes

Dropdown list boxes can be formatted by setting or changing the cell they reference, whether a selection appears in the referenced cell as a value or text, and specifying the list of selections in the dropdown list box.

- **To set or edit dropdown list box items in the Workbook Designer:**
 1. Create a dropdown list box, or select an existing one. When you draw the dropdown list box, draw an area wide enough to display the longest item in the list and as tall as the list of items you want to display. Formula One automatically sets the height of the list box; you set the height of the dropdown list of items.
 2. Select **Format > Object** to display the Format Object dialog box.
 3. Select the Options tab. The following illustration shows an example of the Options tab.

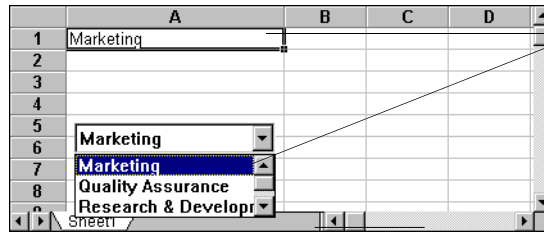


If you want the results of a selection to be displayed in a cell, type the cell location in the Cell text box.

Enter or edit the list of items contained by the dropdown list box. The items must be entered as a semicolon-delimited list.

Check the Assign As Text check box to assign the dropdown list box items as text, rather than values.

4. Click OK. The following illustration shows the results.



When you select Marketing, the text for Marketing is displayed in the cell.

► **To set and manipulate the items contained in a dropdown list box programmatically, use the following properties:**

- Set **ObjItems** to specify a list of items for a dropdown list box. For this property, you provide a semicolon-delimited list of items. The list you provide replaces any previously specified lists. Return the value of **ObjItems** to get a semicolon-delimited list of items from a list box.
- To change an item in a list of items, set **ObjItem**. For this property, you provide the number of the item you want to change and the new value for the item. Return the value of **ObjItem** to get a specific item from a dropdown list box.
- To add an item to a list of items, use **ObjAddItem**. This method adds an item to the end of the current list. Use **ObjInsertItem** to add an item at a specific location within a list.
- Use **ObjDeleteItem** to delete an item from a dropdown list box.
- **ObjGetItemCount** returns the number of items contained by a dropdown list box.
- Use **FormatObjectDlg** (F1OptionsPage) to display the Format Object dialog box with the **Options** tab displayed.

Formatting Check Boxes

The text displayed by a check box can be set either through the Workbook Designer or programmatically.

► **To edit check box text in the Workbook Designer:**

1. Select a check box.
2. Select Format > Object to display the Format Object dialog box.
3. Select the Options tab.
4. Edit the text displayed by the check box.
5. Click OK.

- **To set the text displayed in a check box programmatically:**
 - Set the **ObjText** property. Return the value of **ObjText** to get the text displayed by a check box.

Formatting Buttons

The text displayed on a button can be set either through the Workbook Designer or programmatically.

- **To edit button text in the Workbook Designer:**
 1. Select a button.
 2. Select Format > Object to display the Format Object dialog box.
 3. Select the Options tab.
 4. Edit the text displayed on the button.
 5. Click OK.
- **To set the text displayed on a button programmatically:**
 - Set the **ObjText** property. Return the value of **ObjText** to get the text displayed on a button.

Selecting Check Box and Dropdown List Box Items

Formula One provides a variety of methods for checking or unchecking check box objects and selecting items from dropdown list box objects.

- At runtime, items can be checked or selected interactively using the mouse.
- Properties and methods can set the value of a check box or dropdown list box object.
- By assigning a cell to a graphical object, you can set the value in the cell by making a selection from the graphical object. Likewise, if you enter a value that is in the list associated with a dropdown list box, you can change the value displayed in the dropdown list box. The cell reflects the value to which the graphical object is set, regardless of the method used to set the value.

When you check or uncheck a check box, or select an item from a dropdown list box, you set the value in the assigned cell.

- For a check box, the value of the assigned cell is True if the graphical object is checked or False if unchecked.
- In a dropdown list box, you can choose to have the value of the assigned cell set to the number or the text of the selected item. Items in a dropdown list box are numbered starting with 0 (e.g., the first item is item 0, the second item is item 1, and so on). -1 means that no item is selected in the dropdown list box.

Setting Values Interactively

To set the value of a check box or dropdown list box interactively, the graphical object itself cannot be selected and the pointer must be in normal worksheet editing mode.

- To set the value of a check box, position the pointer anywhere in the check box area and click. The check box toggles between checked and unchecked states.
- To set the value of a dropdown list box, position the pointer anywhere in the dropdown list box area and click. The list of items contained by the dropdown list box is displayed. If the list area is not large enough to display all the items, you may have to click the scroll areas to view the entire list. Then, click the item you want to select.

Setting Values Programmatically

Set the **ObjValue** property to set the value of a check box or dropdown list box specified by graphical object ID number.

- For check boxes, provide 1 to check a check box; provide 0 to uncheck the graphical object.
- For dropdown list boxes, provide the number of the item you want to select. Dropdown list box items are numbered consecutively, starting with 0. If you specify -1, no item is selected in the dropdown list box.

Return the value of **ObjValue** to get the value of the current selection.

- For dropdown list boxes, you can also use **ObjText** to set or return the text displayed on a button or next to a check box.

Setting Values by Cell Reference

For both check boxes and dropdown list boxes, you can specify a cell that references the graphical object. The referenced cell works in two ways:

- If you select an item from the graphical object, the value or text of that selection is displayed in the referenced cell. For example, if you uncheck a check box, FALSE is placed in the cell the check box references.
- If you enter a valid value in the referenced cell, the current selection of the graphical object reflects the cell value. For example, if you enter 0 in the cell, a dropdown list box that references the cell displays its first item as the current selection.

If multiple graphical objects reference the same cell, making a selection in one graphical object makes the same selection in all graphical objects that reference the cell.

The cell reference for a graphical object can be set in the **Options** tab of the Format Object dialog box.

► To display or locate a cell reference interactively:

1. Select Format > Object to display the Format Object dialog box.
2. Select the Options tab.
3. If desired, type a cell reference in the Cell text box.
4. Click OK.

You can also specify a cell reference by calling the **ObjSetCell** method. **ObjGetCell** returns an object cell reference. Furthermore, you can specify a cell reference by using the **ObjCellType**, **ObjCellRow**, and **ObjCellCol** properties. For dropdown list boxes only, you can specify whether the value or text of the selection is displayed in the cell.

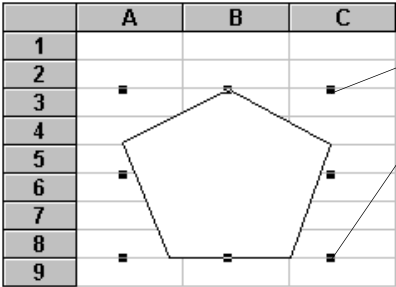
Editing Polygons

When editing polygons, there are two editing modes:

- **Normal Polygon Editing.** This mode allows you to resize and move polygons. Editing of polygon points is not allowed in this mode.
- **Polygon Point Editing.** This mode allows you to reposition polygon points, thus changing the shape of the polygon.

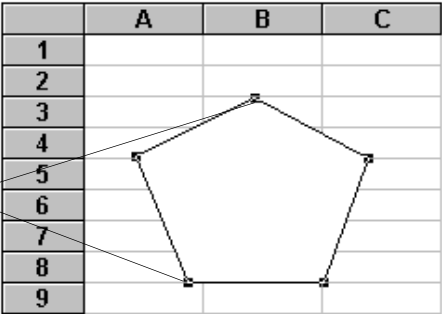
You can set the polygon editing mode by setting the **PolyEditMode** property. To determine the current polygon editing mode, return the value of **PolyEditMode**.

In the Workbook Designer, to use normal polygon editing click the polygon and move the selection handles appropriately. If you want to use polygon point editing, click the polygon point editing tool and move the selection handles of the polygon. The following illustration shows a selected polygon when normal polygon editing and polygon point editing modes are enabled.



When normal polygon editing mode is enabled, the selection handles appear at the edges of the bounding box that surrounds the polygon.

In this mode, the polygon can be resized and moved.



When polygon point editing mode is enabled, a selection handle appears at each point along the border of the polygon.

In this mode, the polygon points can be repositioned and the polygon can be moved.

➤ **To interactively reshape a polygon:**

1. Select the polygon to be reshaped.
2. Make certain that polygon editing mode is enabled.

When polygon editing mode is enabled, selection handles appear at each point along the border of the selected polygon.

3. Position the pointer on the polygon point that you want to move.
4. Click the point and drag the mouse.

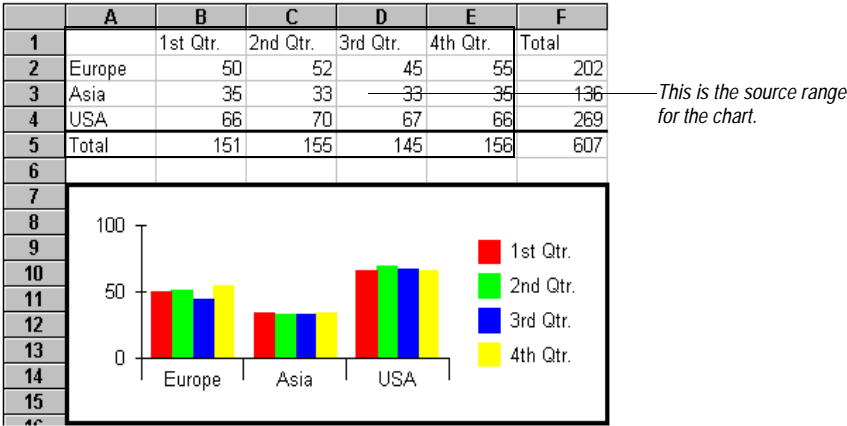
An outline of the lines adjoining the point move as you drag the polygon point.

5. Release the mouse button to place the point at its new location.

CHAPTER 9

Working With Chart Objects

If you have also purchased Tidestone’s First Impression ActiveX control, you can automatically chart worksheet data. In order to draw a chart, First Impression must be properly installed on your system. The following illustration shows an example of a chart on a worksheet.



Creating Charts

- **To create a chart programmatically:**
 - Use the **ObjCreate** with the **F1ObjChart** constant or the **ObjNew** method. These methods draw a chart object on the worksheet in the position you specify and chart the currently selected range.
 - To change a chart’s data range, you must select it, open the Format Object dialog box, select the Options tab, and provide a range reference or defined name for the chart.
- **To interactively chart data:**
 1. Select a range of data to be charted.
 2. Click the chart tool in the toolbar or select Insert > Chart.

3. Using the chart tool, draw a rectangle where you want to place the chart.
The Chart Wizard appears to assist you in designing the chart appearance.
4. Make any necessary selections from the Chart Wizard and click Finish.

Using the Chart Wizard

The pages within the Chart Wizard guide you through most common design decisions required when you create or modify a chart. The Chart Wizard is displayed automatically when you draw a new chart on a worksheet. You can also display the Chart Wizard at any time to assist you in formatting an existing chart.

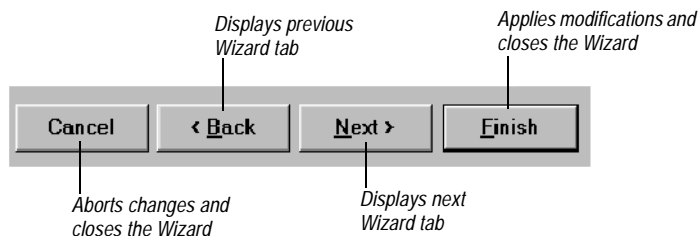
Important Each chart type has individual requirements as to how the data must be laid out. Separate formatting options are also available for different chart types. You should read your First Impression User's Guide to familiarize yourself with the requirements of various chart types.

➤ **To access the Chart Wizard:**

1. Double-click on the chart to activate it.
2. Right-click the chart to display the context menu and select **Wizard**.

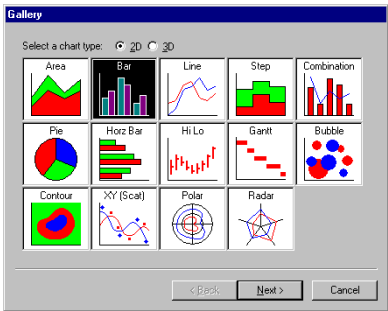
Navigating in the Chart Wizard

The Chart Wizard allows you to control various design aspects such as choosing a chart type, setting chart options, controlling chart layout, and specifying chart and axis titles. Use the navigation buttons at the bottom of the Wizard pages to navigate through the Chart Wizard.



Using the Gallery Page

The Gallery Page allows you to select the type of chart you wish to design. Two radio buttons allow you to differentiate between 2D chart types and 3D chart types.



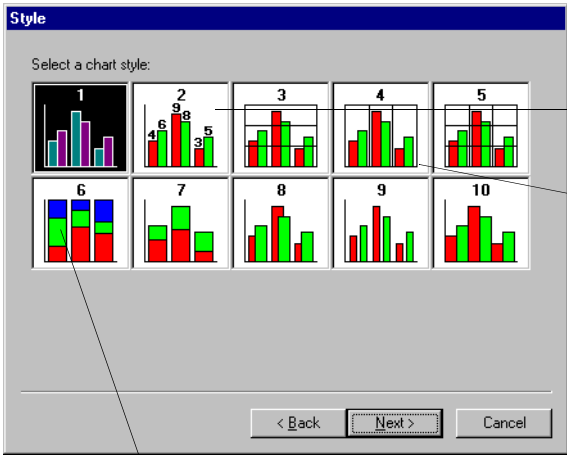
2D Charts



3D Charts

Using the Style Page

The Style Page lets you set the style for the selected chart type. Using the style page, you can easily set chart display options such as series labels, stacking, and bar gap.



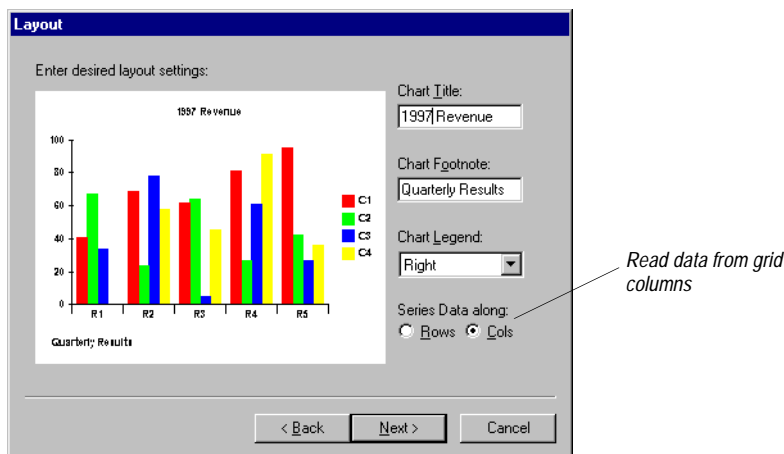
Series Labels

Bar Gap

Series Stacking

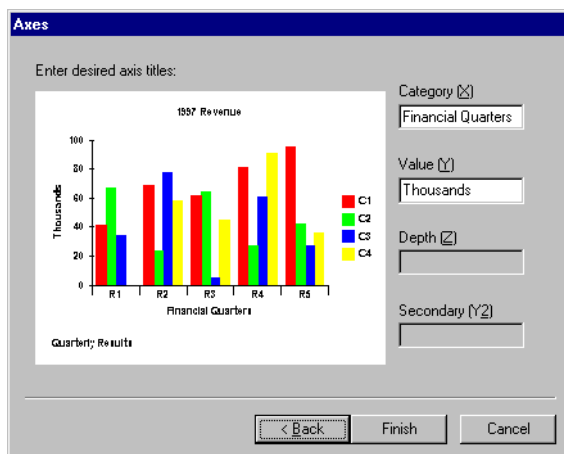
Using the Layout Page

The Layout Page provides methods for determining the elements and layout of the chart plot such as chart titles, chart footnotes, and chart legends.



Using the Axes Page

The Axes Page allows you to optionally label chart axes. The chart preview image on this page shows you how the chart looks with your settings.



Important When you use the Chart Wizard to modify existing charts, the Wizard reverts the chart to its default settings and then restores only those features it controls in the **Gallery**, **Layout**, and **Axes** tabs. Exercise care when modifying charts that may have originally been created without the Wizard. You may need to manually adjust some chart settings after using the Wizard.

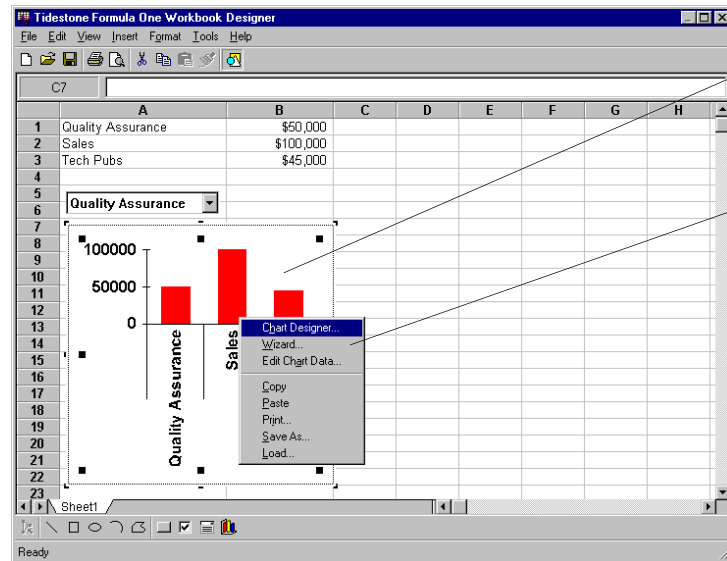
The Chart Wizard provides a quick and easy method for applying some of the most common formatting options to your chart. Additional formatting options are available by using the First Impression Chart Designer. The Chart Designer is displayed by double-clicking on chart elements in an activated chart object, or selecting Chart Designer from the First Impression context menu.

You can modify chart data by selecting **Edit Chart Data** from the First Impression context menu. The Data Grid Editor will appear.

Consult your First Impression documentation for more information about the Chart Designer, the Data Grid Editor, and other First Impression functions.

► **To access First Impression functions:**

1. Double-click on the chart object to activate it.
2. Right-click on the chart to display the context menu.
3. Make a selection from the menu.



Important One of the options on the First Impression context menu is Edit Chart Data. This displays First Impression's Data Grid Editor. Using this tool you can modify the size and content of the chart's data source. Any changes you make in the Data Grid Editor are reflected in the chart, but are NOT reflected in the worksheet. Also, if you use the Data Grid Editor to edit the chart data and then recalculate the workbook in Formula One while the chart still references a range in

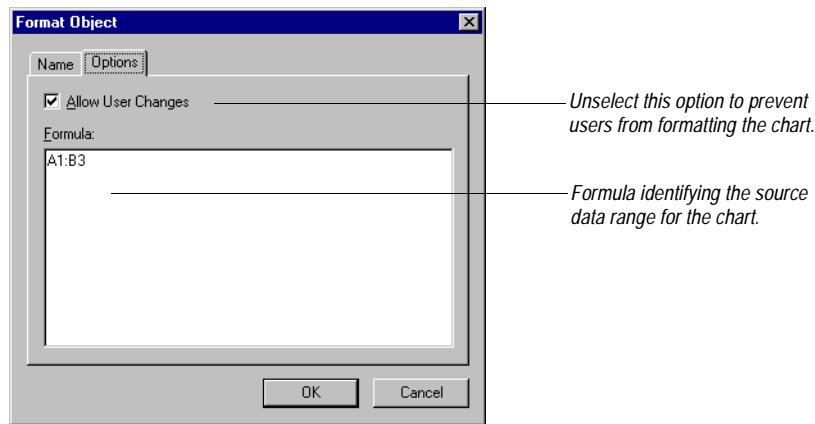
the worksheet, the worksheet data overwrites your changes. To prevent this, select the chart, choose Format > Object, and replace the chart's formula with an empty string (" ") before using the Data Grid Editor to edit the data.

Chart Options

Within Formula One, you can control whether or not users can edit the chart and change the source data range for a chart using the Options tab of the Format Object dialog box.

► **To edit chart options:**

1. Select the chart.
2. Choose Format > Object to display the Format Object dialog box.
3. Select the Options tab.

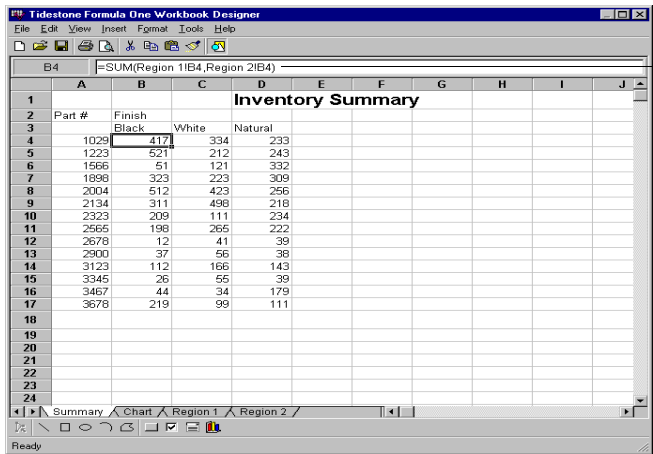


Developer's Note If you make the Workbook Designer available to your end users, they will have access to the Allow User Changes check box. To prevent them from modifying a chart regardless of this setting, set the **AllowObjSelections** property to False.

Referencing Data on Another Worksheet

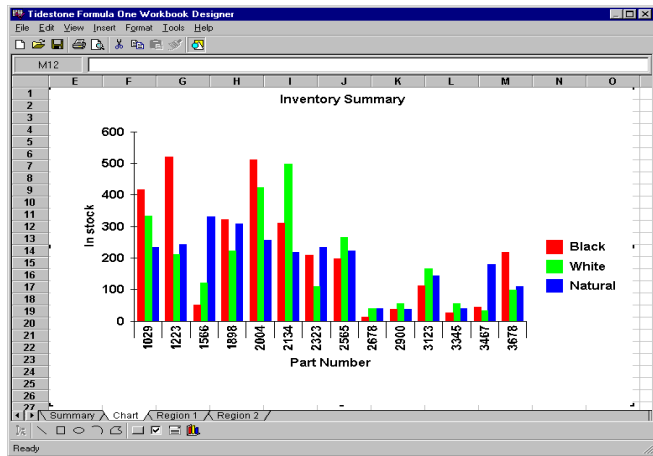
You can also display a chart on one worksheet that references data on a separate worksheet.

- **To create a chart on a separate worksheet:**
 1. Select the chart.
 2. Choose Format > Object to display the Format Object dialog box.
 3. Select the Options tab.
 4. Enter the formula referencing the data source on the other spreadsheet.



This worksheet provides summary information about the inventory data stored on the worksheets named Region 1 and Region 2.

These cells contain formulas that sum information from the worksheets individual region.



The data source formula for this chart object is Summary!B3:D17

Tidestone

CHAPTER 10

Printing Worksheets

Formula One provides many options for printing worksheets and setting printing specifications.

- You can print worksheets through the Workbook Designer.
- You can use properties and methods of the `F1PageSetup` object to print worksheets directly. See the Formula One online documentation for more information on the `F1PageSetup` object and its properties. You can also use methods to display the Page Setup and Printer Setup dialog boxes.

Printing Worksheets

You can print the active worksheet from the Workbook Designer, or from code.

➤ **To print the active sheet from the Workbook Designer:**

1. Select File > Print to display the Print dialog box.
2. You may preview the printout by pressing the Preview button. Formula One will display the Print Preview dialog, shown on page 159.
3. Make any necessary adjustments to the settings and click OK.

➤ **To print the active sheet from code:**

- Use the **FilePrint** method. The following code uses this method to print a worksheet.

```
F1Book1.FilePrint TRUE
```

When you call **FilePrint**, the Print dialog box can be displayed, allowing you to specify the pages to print, the number of copies to print, and other related items.

- Use **FilePageSetupDlgEx** to display the Page Setup dialog box, which gives easy access to setting margins, headers, footers, headings, grid printing, page ordering, and output alignment. The following code displays the Page Setup dialog box.

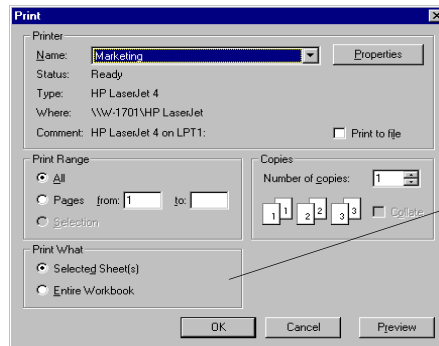
```
F1Book1.FilePageSetupDlgEx
```

➤ **To print selected sheets or the entire workbook from code:**

- Use the **FilePrintEx** method. The following code uses this method to print selected worksheets or the entire workbook as a single document.

```
F1Book1.FilePrintEx bShowPrintDlg, bPrintWorkbook
```

When you call **FilePrintEx**, the Print dialog box can be displayed, allowing you to specify the pages to print, the number of copies to print, and other related items.



The Print dialog default printing selection can be changed by setting the appropriate flag in the FilePrintEx method.

- Use **FilePrintSetupDlg** to invoke the Print Setup dialog box; the standard Windows printer setup dialog box is displayed. It allows you to select a printer, select the paper source, and select the page orientation (portrait or landscape). The following code displays the Print Setup dialog box.

```
F1Book1.FilePrintSetupDlg
```

Specifying Print Areas

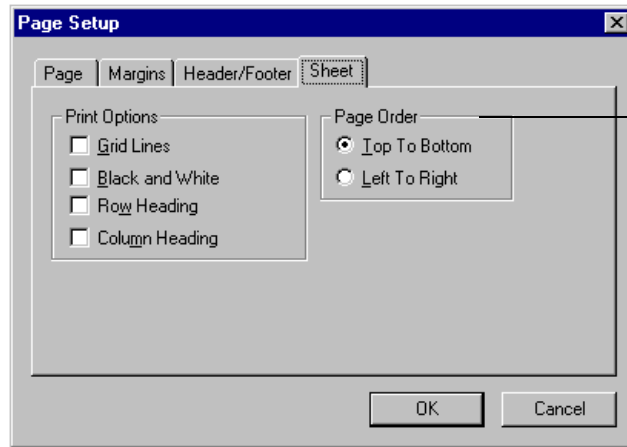
Formula One prints the entire active worksheet unless you specify the ranges you want to print. To specify the areas you want to print, you must set the **Print_Area** name to reflect the worksheet area to be printed.

➤ **To set the print area with menu commands in the Workbook Designer:**

1. Select the ranges to print.
2. Select **File > Print Area > Set Print Area**.

➤ **To set the print area using the Page Setup dialog box in the Workbook Designer:**

1. Select File > Page Setup and click the Sheet tab. The Sheet tab of the Page Setup dialog box will appear, as shown below.



2. In the Print Area text box, enter the range(s) on the selected worksheet(s) that you want to print. You may enter absolute or relative cell references. Separate non-contiguous ranges with commas. When you finish, click OK.

➤ **To set the print area in code:**

- Use the **PrintArea** and **PrintAreaLocal** properties of the **F1PageSetup** object.

The following example uses the **PrintArea** property to set A1:D25 as the area to be printed.

```
Dim pgSetup As F1PageSetup

Set pgSetup = F1Book1.CreateNewPageSetup
pgSetup.PrintArea = "A1:D25"
F1Book1.SetPageSetup pgSetup
```

You can select multiple ranges to print. The ranges do not have to be adjacent. For example, a print area could be comprised of two ranges, A1:D4 and F5:I8.

Specifying Print Titles

You can specify row or column titles that you want printed on each page of your worksheet. If you select a row, it is printed at the top of each page. If you select a column, it is printed at the left edge of each page. You can select multiple rows or columns, but they must be adjacent.

Important When setting print titles, you must select entire rows and columns.

➤ **To set the print titles with menu commands in the Workbook Designer:**

1. Select the rows or columns to use as print titles. You must select entire rows or columns.
2. Select File > Print Titles > Set Print Titles.

➤ **To set the print titles using the Page Setup dialog box in the Workbook Designer:**

1. Select File > Page Setup and click the Sheet tab. The Sheet tab of the Page Setup dialog box will appear, as shown on page 151.
2. In the Print Titles text box, enter the row and/or column range(s) on the selected worksheet(s) that you want to appear on every page of the printout. You may enter absolute or relative cell references. Separate non-contiguous ranges with commas. When you finish, click OK.

➤ **To set the print titles in code:**

- Set the **PrintTitles** or **PrintTitlesLocal** property of the **F1PageSetup** object.

The following example uses the **PrintTitles** property to set rows 1 and 2 and column A as print titles.

```
Dim pgSetup As F1PageSetup

Set pgSetup = F1Book1.CreateNewPageSetup
pgSetup.PrintTitles = "A1:IV2,A1:A65536"
F1Book1.SetPageSetup pgSetup
```

Specifying Page Breaks

Both horizontal and vertical page breaks can be specified on a worksheet. You can specify page breaks interactively using the Workbook Designer, or you can use methods and properties.

In the Workbook Designer, page breaks are always placed adjacent to the active cell. When using methods, page breaks can be placed adjacent to the active cell or a cell that you specify.

- For horizontal (row) page breaks, the page break will be placed above the active or specified cell.
- For vertical (column) page breaks, the page break will be placed to the left of the active or specified cell.

➤ **To set page breaks in the Workbook Designer:**

1. Select the cell adjacent to which you want to place page breaks.
2. Select Insert > Page Break.

➤ **To remove page breaks in the Workbook Designer:**

1. To remove a horizontal page break, select a cell in the row below the break. To remove a vertical page break, select a cell in the column to the right of the break.
2. Select Insert > Remove Page Break.

➤ **To set and remove page breaks in code:**

There are several categories of page break methods. The **AddPageBreak** and **RemovePageBreak** methods add page breaks adjacent to the active cell. The following example uses these methods:

```
F1Book1.AddPageBreak  
F1Book1.RemovePageBreak
```

AddRowPageBreak, **AddColPageBreak**, **RemoveRowPageBreak**, and **RemoveColPageBreak** add and remove page breaks adjacent to the row or column that you specify in the method. The following example uses the **AddRowPageBreak** method:

```
Dim theRow As Long  
theRow = 8  
F1Book1.AddRowPageBreak theRow
```

NextRowPageBreak returns the next page break below the row that you specify in the method. **NextColPageBreak** returns the next page break to the right of the column that you specify in the method. The following example uses the **NextRowPageBreak** method:

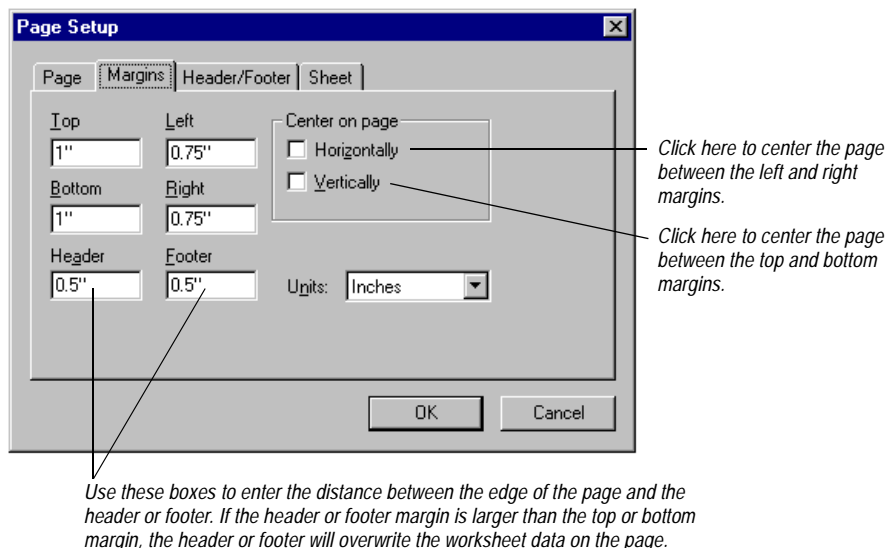
```
Dim nextBreak As Long  
Dim theRow As Long  
theRow = 20  
nextBreak = F1Book1.NextRowPageBreak (therow)
```

Specifying Margins

You may set margins in inches or centimeters, or you may tell Formula One to use whatever margins it takes to print the print area in the center of the worksheet. You may also establish margins for headers and footers.

➤ **To specify margins in the Workbook Designer:**

1. Choose File > Page Setup. Click on the Margins tab. The Margins tab of the Page Setup dialog box appears, as shown below.



2. Select the options you want. Click OK when you are done.

➤ **To specify margins programmatically:**

- Use the following properties of the F1PageSetup object: **TopMargin**, **LeftMargin**, **BottomMargin**, **RightMargin**, **HeaderMargin**, **FooterMargin**, **CenterHoriz**, and **CenterVert**.

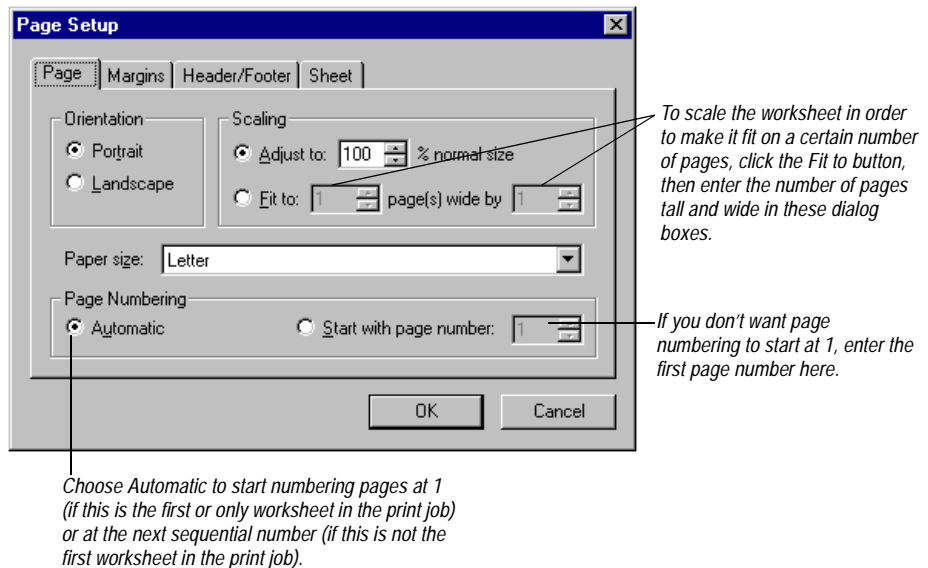
Setting Page Numbering

You may number the worksheet pages automatically starting at 1 or at any number you choose.

Note By default, the page numbers you set here appear on the worksheet footer. To change the location of the page number on the printout, change the worksheet footer and/or header.

► To specify page numbering in the Workbook Designer:

1. Choose File > Page Setup. Click on the Page tab. The Page tab of the Page Setup dialog box appears, as shown below.



2. Select the options you want. Click OK when you are done.

► To specify page numbering programmatically:

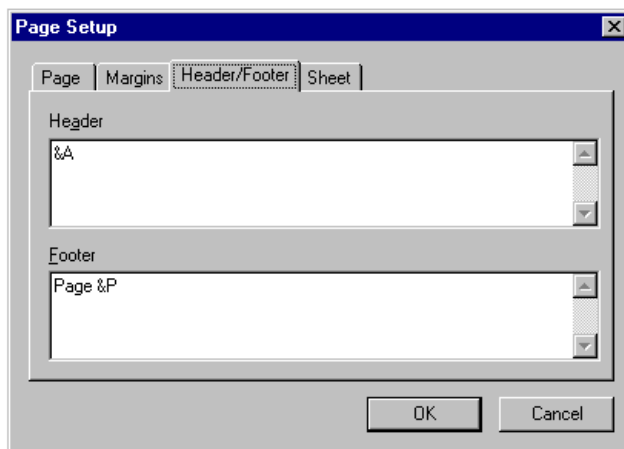
- Use the **AutoPageNumber** and **FirstPageNumber** properties of the F1PageSetup object.

Specifying Headers and Footers

Headers are printed at the top of each page and footers are printed at the bottom. You can define the alignment, contents, and formatting of headers and footers.

➤ **To specify headers and footers in the Workbook Designer:**

1. Choose File > Page Setup. Click the Header/Footer tab. The Header/Footer tab of the Page Setup dialog box appears, as shown below.



2. Type text and special formatting codes in the Header and Footer boxes. See below for information on special formatting codes. When you finish, click OK.

➤ **To specify headers and footers programmatically:**

- Use the **Footer** and **Header** properties of the F1PageSetup object.

Formatting Codes for Headers and Footers

Header and footer codes allow you to format and align the text and to insert worksheet-specific information, like page numbers and the worksheet name.

The alignment codes (&L, &C, and &R) must go before font-related codes (like &B for bold and &I for italic). Codes for printing worksheet-specific information like worksheet title go last. If you put the codes in the wrong order, Formula One may ignore some of them. Enter codes in upper or lower case and separate them with a space.

By default, codes and text are centered unless &L or &R is specified.

Format Code	Description
&L	Left-aligns the characters that follow.
&C	Centers the characters that follow.
&R	Right-aligns the characters that follow.
&B	Use a bold font.

Format Code	Description
&I	Use an italic font.
&U	Underline the header.
&S	Strikeout the header.
&O	Ignored.
&H	Ignored.
&"fontname"	Use the specified font.
&nn	Use the specified font size (must be a two-digit number)
&D	Prints the current date.
&T	Prints the current time.
&F	Prints the workbook name.
&A	Prints the worksheet name.
&P	Prints the page number.
&P+ <i>number</i>	Prints the page number plus <i>number</i> .
&P- <i>number</i>	Prints the page number minus <i>number</i> .
&&	Prints an ampersand.
&N	Prints the total number of pages in the document.

Setting Page Orientation

You may print pages in portrait or landscape orientation.

➤ To specify page orientation in the Workbook Designer:

1. Choose File > Page Setup. Click the Page tab. The Page tab will appear, as shown on page 155.
2. Choose Portrait or Landscape, then click OK.

➤ To specify page orientation programmatically:

- Use the **Landscape** property of the F1PageSetup object. If you do not set this property, landscape printing can be handled using your system's default Print dialog box.

Setting Up Scaling for Printing

You may specify a specific scale percentage to print your chart. You can also force the chart to print on a specified number of pages.

➤ To specify worksheet size and scale in the Workbook Designer:

1. Choose File > Page Setup and click the Page tab. The Page tab will appear, as shown on page 155.
2. To specify a scale factor, enter it in the Adjust To text box.

3. To force the chart to fit on a specified number of pages, click the Fit to button, then enter the number of pages tall and pages wide you want the worksheet to fit into.
4. When you finish choosing options, click OK.

➤ **To specify worksheet size and scale programmatically:**

- Use the **PrintScale**, **FitPages**, **PagesWide**, and **PagesTall** properties of the F1PageSetup object.

Specifying Page Printing Order

You may use the Page Setup dialog box to specify the order in which to print your worksheet pages, or you may specify page order programmatically.

➤ **To specify page order in the Workbook Designer:**

1. Choose File > Page Setup and click the Sheet tab. The Sheet tab will appear, as shown on page 151.



2. To print from the top to bottom margins, click Top to Bottom. The data in the leftmost columns prints before the data in columns farther out.



3. To print from the left to right margins, click Left to Right. The data in the uppermost rows prints before the data in lower rows.

4. Click OK.

➤ **To specify page order programmatically:**

- Use the **LeftToRight** property of the F1PageSetup object.

Choosing Paper Size

➤ **To specify paper size in the Workbook Designer:**

1. Choose File > Page Setup and click the Page tab. The Page tab will appear, as shown on page 155.
2. Choose the paper size from the dropdown list box, then click OK.

➤ **To specify paper size programmatically:**

- Use the **PaperSize** property of the F1PageSetup object.

Specifying Miscellaneous Printing Options

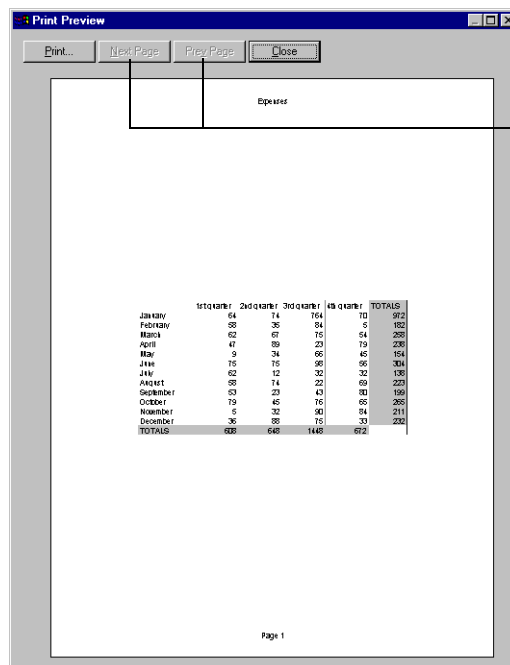
Formula One lets you choose whether or not you want certain worksheet page elements to appear on the printout and whether you want your printout in color or black and white.

- **To specify miscellaneous printing options in the Workbook Designer:**
 1. Choose File > Page Setup and click the Sheet tab. The Sheet tab will appear, as shown on page 151.
 2. Check the Grid Lines box if you want grid lines to appear on the printout.
 3. Check the Black & White box if you want any colored elements on the worksheet to be printed in shades of gray.
 4. Check the Row Heading and Column Heading boxes to make row and column headings appear on the printout.
 5. Click OK when you are done.
- **To specify miscellaneous printing options programmatically:**
 - Use the **BlackAndWhite**, **GridLines**, **RowHeadings**, and **ColHeadings** properties.

Previewing Your Printout

Formula One lets you preview the pages that are going to print. You can preview the printout of your worksheet(s) from the Workbook Designer, or you can use properties and methods.

- **To preview your printout in the Workbook Designer:**
 1. Choose File > Print Preview. The Print Preview screen appears, as shown below.



If your printout has multiple pages, you can use the Next Page and Prev Page buttons to view the different pages.

- An image of what the printout will look like appears on the screen, showing the margins, headers and footers, print titles, page breaks, and other printing features discussed in this chapter. When you finish, click Close.
- To print the worksheet(s) directly from this screen, click Print. Formula One prints all pages as shown, without showing you the Print dialog box, and closes Print Preview.

➤ **To preview your printout using properties and methods:**

Formula One provides a three methods to simplify adding print preview functionality to your application. As a developer, you can set the **PrintPreviewEx** method to preview worksheets prior to printing.

This method allows a device context to be passed for painting the sheet. With **PrintPreviewEx** a window handle and a rectangle are passed in, along with the page number to display. The **PrintPreviewEx** method allows VB programmers and others to add print preview functionality.

A simpler method of previewing worksheets utilizes the **FilePrintPreview** method. By calling the **FilePrintPreview** method, you can display a dialog that previews the printed page.

For MFC users, the **PrintPreviewDCEX** method can be used with the print preview mechanism provided by MFC. The **PrintPreviewDCEX** method allows a device context to be passed in for painting the sheet. A page number is also passed in and the total number of pages in the sheet is passed back to the user.

CHAPTER 11

Working With Databases

Database connectivity is one of Formula One's most powerful features. You can use Formula One, along with ODBC drivers, to retrieve data from a database and use it to populate a Formula One worksheet at the starting row and column position you specify. This ODBC connection offers incredible speed and flexibility in populating your worksheet.

Overview of Formula One Connections

Database connections are made on a per worksheet basis. This means that you can populate each worksheet in a workbook with data from a different query, or even a different database. This is a powerful feature because Formula One worksheet functions work across multiple worksheets.

As an example, you can query district sales information, and place information from each district in a separate worksheet within the same workbook. On each worksheet, you can perform summary calculations to show statistics for that district. You can also add additional worksheets to the workbook that execute summaries or analysis on any combination of the districts.

If you also purchase Tidestone's First Impression charting tool, you can select a range of data in a worksheet and draw a chart illustrating that data on the Formula One worksheet.

Installing the ODBC Drivers

In order to connect a Formula One worksheet to a database via ODBC, you must have the 32-bit version of the ODBC drivers installed on your system. These drivers come with most 32-bit development environments such as Visual Basic 4.0, Office95, and Visual C++ 2.x. When you install these environments be sure to select the ODBC option. If you have already installed your development environment, you can re-install and check only the ODBC option. Be aware that different environments offer different drivers. Please contact your development environment vender for information about the drivers available.

Setting up a Data Source

You can create a new data source from within Formula One, or you might find it more convenient to set up your data sources outside of Formula One.

➤ **To set up a data source outside of Formula One:**

1. Double-click the ODBC icon in the Control Panel to bring up the Data Sources dialog box.
2. Click Add to add a connection to the database you want to reach.
3. In the Add Data Source dialog box, choose the appropriate 32-bit driver for the database you are connecting to and click Finish. This brings up an ODBC dialog box based on the driver you chose.
4. Enter a Data Source Name and Database Name that describe the database you want to connect to and click OK.

Connecting to the Data Source

Formula One provides the **F1ODBCConnect** API object used with the **ODBCConnectEx** method to connect the active worksheet to a database. In addition, Formula One provides the **ODBCConnect** method to connect the active worksheet to a database.

If you are using the **ODBCConnectEx** method to connect the active worksheet to a data source, you provide ODBC connection with the following:

- a connect object (**F1ODBCConnect**).
- a boolean that controls whether SQL errors are displayed.

The following code example shows the use of this object and method:

```
If Command1 = True Then
    Dim pConnect As New F1ODBCConnect
    Dim Flobdcconnect As String
    pConnect.ConnectStr = F1ODBCConnect
    F1Book1.ODBCConnectEx pConnect, True

connecterror:
    MsgBox Error
```

To allow the user to select the database at runtime, the **F1ODBCConnect** object should be equal to a null string.

If you are using the **ODBCConnect** method to connect the active worksheet to a data source, you provide ODBC connection with the following:

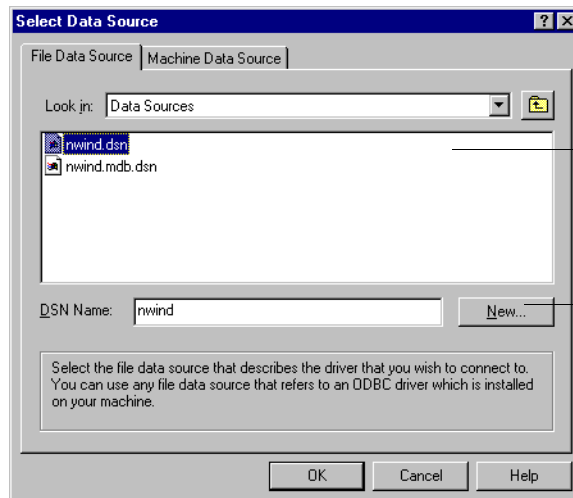
- a variable containing a connect string.
- a boolean that controls whether SQL errors are displayed.
- a variable that receives the returned SQL status code.

The following code example shows the use of this method:

```
On Error GoTo ConnectError
Dim returnCode As Integer, pConnect As String
F1Book1.ODBCConnect pConnect, True, returnCode
Exit Sub
ConnectError:
MsgBox Error
```

The connect string must be a variable. This allows you to let the user select the database at runtime and returns the data source in the connect string.

Next, a dialog box is displayed, allowing the user to select or create a file data source or machine data source at runtime. The following illustration shows the **File Data Source** tab of the Select Data Source dialog box



This selection describes the driver for which you want to connect. You can use any file data source that refers to an ODBC driver on your machine.

Click New to create a new data source.

If you want to create a new data source, click New. This launches the ODBC Datasource Administrator Utility for your database. For more information, consult your database administrator documentation.

For more information about **ODBCConnect** or **ODBCConnectEx**, refer to the Formula One online documentation.

Querying the Data Source

Once you connect to a data source, you can build and execute a query. This is accomplished using Formula One's **F1ODBCQuery** API object and **ODBCQueryEx** method or the **ODBCQuery** method.

If you are using **ODBCQueryEx**, you must provide the following information:

- an ODBC query object.
- if the query string is null, the ODBC Query dialog box is displayed to allow the user to build the query at runtime.
- row and column coordinates that identify where in the active worksheet the returned data is to be placed.
- a boolean that determines whether the ODBC Query dialog box is displayed.
- variables that control whether data and data type information is used to format the worksheet size, column headings, column width, and cell formatting.
- a variable that received the returned status code.

Below is the code that implements this in the demo.

```
On Error GoTo FetchError
Dim pConnect As New F1ODBCConnect
Dim pQuery As New F1ODBCQuery
Dim bShowDlg As Boolean

'connect to SouthCreek database
pConnect.ConnectStr = "DSN=SouthCreek"
F1Book1.ODBCConnectEx pConnect, True

'prompt for query at runtime
'initially , query Is Empty
'Cell format, column names, column width, and the maximum
'number of rows and columns displayed is determine
'from the data that is retrieved from the database

pQuery.SetColFormats = True
pQuery.SetColNames = True
pQuery.SetColWidths = True
pQuery.SetMaxRC = True
pQuery.QueryStr = ""
bShowDlg = True

F1Book1.ODBCQueryEx pQuery, 1, 1, bShowDlg

Exit Sub
FetchError:
MsgBox Error
```

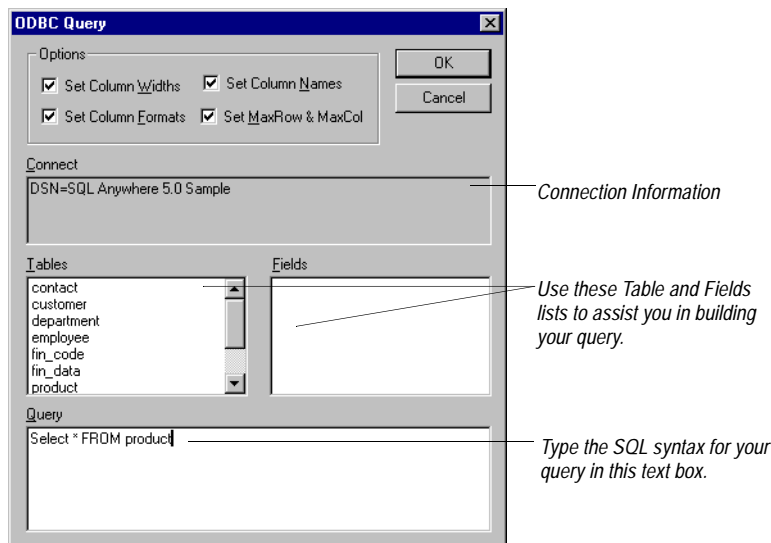
If you are using **ODBCQuery**, you must provide the following information:

- a variable that specifies the query syntax. If you pass a null string for this argument, the ODBCQuery dialog box is displayed to allow the user to build the query at runtime. The query string must be passed as a variable. This allows you to let the user build the query at runtime and returns the final query string.
- row and column coordinates that identify where in the active worksheet the returned data is to be placed.
- a boolean that determines whether the ODBCQuery dialog box is displayed.
- variables that control whether data and data type information is used to format the worksheet size, column headings, column width, and cell formatting.
- a variable that received the returned status code.

Below is the code that implements this in the demo:

```
On Error GoTo FetchError
Dim returnCode As Integer, query As String
Dim setColNames As Boolean, setColFormats As Boolean
Dim setColWidths As Boolean, setMaxRC As Boolean
Let query = cboQueries.TEXT
setColNames = chkSetColNames.Value
setColFormats = chkSetColFormats.Value
setColWidths = chkSetColWidths.Value
setMaxRC = chkSetMaxRC.Value
FlBook1.ODBCQuery query, Val(txtStartRow.TEXT),
    Val(txtStartCol.TEXT), optShowDialog.Value, setColNames,
    setColFormats, setColWidths, setMaxRC, returnCode
Exit Sub
FetchError:
MsgBox Error
```

If the query string is null or you set the *bShowDialog* value to True, the query dialog is displayed as shown in the following illustration.



- **Set Column Widths.** Check this box to automatically set the width of each column to be wide enough to display the widest data in the column.
- **Set Column Names.** Check this box to display field names instead of the standard alphabetic column headings. Even though field names are displayed as the column headings, formulas must still use the standard cell referencing conventions (e.g., A1).
- **Set Column Formats.** Check this box to have formats for date, time, and currency fields set automatically when data is placed in the worksheet. If you do not check this box, you must set the formats for these columns manually.
- **Set MaxRow & MaxCol.** Check this box to have the maximum number of worksheet rows and columns set to the number of records and fields returned by the query.

When you click OK in the ODBC Query dialog box, the query is executed. The following example shows the results of a returned query.

	id	name	description	size	color	quantity	unit_price
1	300	Tee Shirt	Tank Top	Small	White	28	9
2	301	Tee Shirt	V-neck	Medium	Orange	54	14
3	302	Tee Shirt	Crew Neck	One size fits all	Black	75	14
4	400	Baseball Cap	Cotton Cap	One size fits all	Black	112	9
5	401	Baseball Cap	Wool cap	One size fits all	White	12	10
6	500	Visor	Cloth Visor	One size fits all	White	36	7
7	501	Visor	Plastic Visor	One size fits all	Black	28	7
8	600	Sweatshirt	Hooded Sweatshirt	Large	Green	39	24
9	601	Sweatshirt	Zippered Sweatshirt	Large	Blue	32	24
10	700	Shorts	Cotton Shorts	Medium	Black	80	15

Notice that Set Column Names and Set MaxRow and Max Column have been set to True. Column labels have been replaced with database field names and the size of the worksheet has been adjusted to the number of returned columns and rows.

Note To maintain Excel compatibility, each cell is limited to 256 characters. If the returned data exceeds this limit, the text is truncated to fit in the cell

Updating or Inserting Data

Formula One supports the use of prepared SQL statements to update database data, or insert or delete database records. Following is an overview of this process:

- Establish a database connection using **ODBCConnectEx** or **ODBCConnect**.
- Build a PREPARE statement using **ODBCPrepareEx** or **ODBCPrepare**.
- If necessary, bind the PREPARE statement parameters to worksheet columns using one or more **ODBCBindParameterEx** or **ODBCBindParameter** methods.
- Execute the PREPARE statement with **ODBCExecuteEx** or **ODBCExecute**.

In addition, Formula One provides tools for handling errors that occur during this process:

- **ODBCError**. **ODBCError** can be checked after failure of any Formula One ODBC method.
- **ODBCErrorMsg**. The **ODBCErrorMsg** property returns a detailed message of the error that occurred.
- **ODBCNativeError**. The **ODBCNativeError** property displays ODBC native error information to the user following an ODBC method failure.

- **ODBCExecuteError.** The **ODBCExecuteError** event allows you to determine what happens if an error is encountered during the execution of the prepared statement.
- **ODBCSQLState.** The **ODBCSQLState** property shows an SQL state error number after failure of any Formula One ODBC method.

A common way you might want to incorporate these features into a Formula One application is to populate a worksheet with database data using **ODBCQueryEx** or **ODBCQuery** which allows the user to edit the information, and then update the database with the user updates. You can use the **EndEdit** event to determine if the user has made changes, or the **SelChanged** event to determine when a user leaves a particular row.

Using PREPARE Statements

ODBCPrepareEx or **ODBCPrepare** sends an SQL string to the database via ODBC. For specific information about creating PREPARE statements, refer to your SQL documentation.

The string sent by **ODBCPrepareEx** or **ODBCPrepare** can be any valid SQL statement. Formula One does not attempt to parse or modify the statement before sending it to SQLPrepare. **ODBCPrepareEx** or **ODBCPrepare** are specifically designed to provide you a way to perform UPDATE, INSERT, and DELETE operations from within your Formula One code. It is NOT recommended that you use **ODBCPrepareEx** or **ODBCPrepare** to send SELECT statements from the database in order to populate a worksheet with data. **ODBCQueryEx** or **ODBCQuery** are optimized to retrieve data from a database and display it in worksheet columns more efficiently.

A PREPARE statement can contain static or dynamic information. You would use static parameters when you have a single, well-defined change to make to the database.

The following code for **ODBCPrepareEx** adds a new record to the Product table.

```
retcode = F1Book1.ODBCPrepareEx ("INSERT INTO Product
                                   (id,name,description,size,color,quantity,unit_price) VALUES
                                   (701,Sunglasses,wrap-around,Medium,Black,80,75)")
```

The following code for **ODBCPrepare** adds a new record to the Product table.

```
F1Book1.ODBCPrepare "INSERT INTO Product
                     (id,name,description,size,color,quantity,unit_price)",
                     "VALUES (701,Sunglasses,wrap-around,Medium,Black,80,75)",
                     retCode
```

However, when working with worksheets, you more commonly want to supply data from the worksheet as a parameter in your PREPARE statement. For example, if you have a number of new authors listed in a worksheet and you want to add them all to your database you could use variable parameters to make your PREPARE statement more dynamic. In this way you can supply data from the worksheet as the statement is executed.

The following code for **ODBCPrepareEx** allows the user to supply data to the worksheet as the statement is executed:

```
retcode = F1Book1.ODBCPrepareEx ("INSERT INTO Product SET (id = ?,
    name = ?, description = ?, size = ?, color = ?, quantity =
    ?, unit_price = ?)")
```

The following code for **ODBCPrepare** allows the user to supply data to the worksheet as the statement is executed:

```
F1Book1.ODBCPrepare "INSERT INTO Product SET id = ?, name = ?,
    description = ?, size = ?, color = ?, quantity = ?,
    unit_price = ?", retCode
```

In this case, each of the question marks is a variable parameter tag that must be bound to a worksheet column using the **ODBCBindParameterEx**.

Binding Worksheet Columns

When you use variable parameter tags in an **ODBCPrepareEx** or **ODBCPrepare** statement, you must bind each parameter to a column in the worksheet and identify the type of data that is located in that worksheet column. If the **ODBCPrepareEx** or **ODBCPrepare** statement contains two variable parameter tags, it must be followed by two separate **ODBCBindParameterEx** or **ODBCBindParameter** statements.

The following example uses **ODBCBindParameter** to creates a PREPARE statement and binds parameters in the PREPARE statement to columns in the illustrated worksheet.

	id	name	description	size	color	quantity	unit_price
1	300	Tee Shirt	Tank Top	Small	White	28	9
2	301	Tee Shirt	V-neck	Medium	Orange	54	14
3	302	Tee Shirt	Crew Neck	One size fits all	Black	75	14
4	400	Baseball Cap	Cotton Cap	One size fits all	Black	112	9
5	401	Baseball Cap	Wool cap	One size fits all	White	12	10
6	500	Visor	Cloth Visor	One size fits all	White	36	7
7	501	Visor	Plastic Visor	One size fits all	Black	28	7
8	600	Sweatshirt	Hooded Sweatshirt	Large	Green	39	24
9	601	Sweatshirt	Zipped Sweatshirt	Large	Blue	32	24
10	700	Shorts	Cotton Shorts	Medium	Black	80	15

Sheet1

The following examples binds these columns.

```
F1Book1.ODBCPrepare "INSERT INTO Product SET id = ?, name = ?,
    description = ?, size = ?, color = ?, quantity = ?, unit_price =
    ?", retCode
```

Binds the first question mark to the second worksheet column and specifies that the data type as Long.

```
F1Book1.ODBCBindParameter 1, 2, F1CDataLong, retcode
```

Binds the second question mark to the second worksheet column and specifies the data type as Character.

```
F1Book1.ODBCBindParameter 2, 2, F1CDataChar, retcode
```

If you do not provide a bind for each question mark in the PREPARE statement, the **ODBCExecute** or **ODBCExecuteEx** statements fail.

Formula One does no checking to verify that the data in the column matches the type you have specified. If there is a data conversion error during an **ODBCExecuteEx** or an **ODBCExecute**, the **ODBCExecuteError** event is fired. **ODBCExecuteError** allows you to trap for data conversion errors and determine a course of action. The error handler also returns the affected row and column in the worksheet so that you can determine which cell is causing the problem.

Executing PREPARE Statements

Once you create a PREPARE statement and bind any variable parameters to worksheet columns, you are ready to execute the PREPARE statement.

ODBCExecuteEx and **ODBCExecute** use any information provided in **ODBCBindParameterEx** or **ODBCBindParameter** statements to fill in the variables in the **ODBCPrepareEx** or **ODBCPrepare** statement. They then attempt to execute the PREPARE statement. If you are filling in parameters with worksheet data, the **ODBCExecuteEx** or **ODBCExecute** methods also identify which worksheet rows to process.

Important If any Formula One ODBC statement other than **ODBCError**, **ODBCSQLState**, **ODBCNativeError**, **ODBCErrorMsg**, **ODBCBindParameterEx**, or **ODBCBindParameter** are placed between **ODBCPrepareEx** or **ODBCPrepare** and **ODBCExecuteEx** or **ODBCExecute**, the PREPARE string you were attempting to build is lost and the execute fails.

The following example for **ODBCExecute** processes the bound data in the first 11 rows of the spreadsheet through the PREPARE statement.

```
F1Book1.ODBCExecute 1, 11, retCode
```

If an error occurs during an **ODBCExecuteEx** or **ODBCExecute**, an **ODBCError**, **ODBCSQLState**, **ODBCNativeError**, or **ODBCErrorMsg** is executed. You can provide code to determine what happens when an error is encountered.

Disconnecting from the Data Source

After your query you should disconnect from the database. This is done with the **ODBCDisconnect** method. Following is a code example showing the use of this method.

```
F1Book1.ODBCDisconnect
```


CHAPTER 12

Using Formula One With the Internet

Using Formula One, you can save an entire worksheet in HTML format for use on the Internet, or embed a Formula One worksheet in an existing HTML file. You can write only the HTML data or pass the full design capabilities of the Workbook Designer to your HTML document.

Formula One can also be used by containers for Internet and corporate Intranet application development. The Formula One 5.0 ActiveX workbook control has been digitally signed by Verisign Commercial Software Publishers CA and supports the to IObjectSafety interface for secure data.

There are a number of tools available to assist you in creating the necessary files for displaying the Formula One ActiveX workbook control on your web page. For an up-to-date list of tools and instructions on how to obtain them, visit our website at <http://www.tidestone.com/internet>.

Writing out a Worksheet File in HTML Format

Formula One users can save a worksheet in HTML format using the File > Write command.

The Write dialog box gives them the option of saving the document as:

- **HTML.** HTML format including data formatting, font and color information.
- **HTML (Data Only).** HTML format includes data formats, but excludes font and color information.

➤ **To write a document to HTML programmatically:**

- Use the **SaveFileDialog** or **SaveFileDialogEx** method to call the Write dialog box. The following example uses **SaveFileDialogEx** to call the Write dialog box. The default file type is set to HTML.

```
Dim pFileInfo As New FIFileSpec
pFileInfo.Name = F1Book1.Title
pFileInfo.Type = FIFileHTML
F1Book1.SaveFileDialogEx "Save As HTML", pFileInfo
```

Embedding Formula One Data in an HTML file

► To embed a Formula One worksheet in an existing HTML file:

1. Create a Formula One project and write some code utilizing the **InsertHTML** method to embed the worksheet into an HTML file.
2. Create an anchor point (location) in the HTML file where you want the worksheet embedded.

Utilizing the InsertHTML Method

The most important aspect of the **InsertHTML** method is declaring an *AnchorName* variable. *AnchorName* identifies the location in your HTML file where you wish to insert your Formula One worksheet. The following example declares *AnchorName* = "data1" and inserts a worksheet into the HTML document titled "mortgage."

```
Private Sub Command1_Click()
    Dim nRow1 As Long
    Dim nCol1 As Long
    Dim nRow2 As Long
    Dim nCol2 As Long
    Dim nSheet As Long
    Dim pAnchorName As String
    pAnchorName = "data1"
    nRow1 = 1
    nCol1 = 1
    nRow2 = 10
    nCol2 = 5
    nSheet = 1
    FlBook1.InsertHTML nRow1, nCol1, nRow2, nCol2, nSheet, App.Path &
        "\mortgage.HTML", True, pAnchorName
End Sub
```

Creating an Anchor in your HTML Source File

Depending on the HTML editor you are using, the technique used to create an anchor or target location in your HTML document might vary. Consult your HTML editor documentation for details on how to edit HTML files.

The following example shows the HTML markup code for an anchor and anchor description "data1".

```
<CENTER><P><A NAME="data1"></A></P></CENTER>
```

HTML Document Design

The final HTML document design depends on the limitations of your HTML editor and your own creative ability. Formula One allows you to pass the full design capabilities of the Workbook Designer to your HTML document through the **Write** method and the HTML file type. When writing a worksheet file, you can save your file as a number of file types. By choosing the HTML file type, you ensure that any worksheet design characteristics such as font, text alignment, and color are passed to your HTML document. If you choose to write your file as an HTML (Data only) file type, you save text without any design attributes.

Introducing Internet Application Development

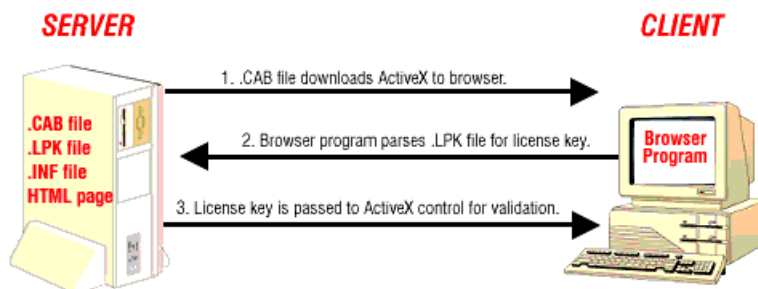
Formula One can be used by containers for Internet and corporate Intranet application development. The Formula One ActiveX workbook control has been digitally signed by Verisign Commercial Software Publishers CA and supports the to IObjectSafety interface for secure data.

Viewing a Web Page Containing Formula One

When a browser encounters a web page containing an ActiveX control, a unique exchange of information must take place between the browser and the web server before the ActiveX control is rendered on the page.

First, the Browser program checks to see if a copy of the ActiveX control resides on the client's computer. If not, the browser extracts the Formula One ActiveX control (and any dependent files) from the Cabinet file that resides on the server and copy them to the client machine.

Next, the browser program parses a License Pack File on the server in order to validate the license that was embedded in the Formula One control at design time. The browser then passes the extracted license key to the Formula One control on the client machine. If the license key from the .LPK file matches the Formula One control's license, the control is rendered on the web page.



Adding Formula One to your Web Page

In order to successfully add Formula One to your web page you must completing the following tasks:

1. Create an HTML web page that contains the Formula One ActiveX control.
2. Create a License Pack File to use for registering the control on the browser machine.
3. Create an .INF File that lists all of the control's dependencies.
4. Create a Cabinet File to compress and store the files needed for downloading by the browser program.
5. Copy the License Pack and Cabinet Files to your Internet Server where they can be accessed by the browser program.

For specific instructions on how to create these files, visit our website at <http://www.tidestone.com/internet>.

Using Methods and Events for Internet Development

Some containers (such as Microsoft's Internet Explorer) cannot use ByRef parameters. Formula One provides alternatives to the ByRef methods and events currently offered in the API. Refer to your Formula One on-line help for more information about the methods and events that are available as alternatives for Internet development.

Understanding Formula One's IOjectSafety Support

Formula One supports the IOjectSafety interface for Internet application development. The IOjectSafety interface allows a control to instruct a container that it is safe for initialization or scripting. On an Internet download, the browser can request that the control enter safe mode. When this request is detected, the following methods are disabled so they do not harm the user's system:

Draw	FilePrint	FilePrintEx
FilePrintPreview	GetTabbedText	InsertHTML
ObjCreatePicture	ObjNewPicture	ObjSetPicture
ODBCBind Parameter	ODBCBindParameterEx	ODBCConnect
ODBCConnectEx	ODBCDisconnect	ODBCError
ODBCErrorMsg	ODBCExecute	ODBCExecuteEx
ODBCNativeError	ODBCPrepare	ODBCPrepareEx
ODBCQuery	ODBCQueryEx	ODBCSQLState
OpenFileDialog	OpenFileDialogEx	PrintDevMode
PrintPreview	PrintPreviewDC	PrintPreviewDCEX
PrintPreviewEx	Read	ReadEx

ReadFromBlob	SaveFileDialog	SafeFileDialogEx
Write	WriteEx	WriteRange
WriteRangeEx	WriteToBlob	WriteToBlobEx

The following worksheet functions are also disabled in your browser when the safe request is detected:

CALL
REGISTER.ID

Note If you want to load a workbook from an Internet address, use the URL property. For additional information about this property, refer to the Formula One Online Documentation.

Understanding Formula One’s Safe Events

Formula One provides alternative events that are safe for use on the Internet. These events are only fired if you set the **DoSafeEvents** property to True. The following table includes a list of the events that are replaced with their safe counterparts.

Events	Internet Alternative Events
BeforeReplace	SafeBeforeReplace
Found	SafeFound
EndEdit	SafeEndEdit
ODBCExecuteError	SafeODBCExecuteError
StartEdit	SafeStartEdit
ValidationFailed	SafeValidationFailed

Tidestone

CHAPTER 13

Performance Tuning and Specifications

Using Performance Tuning

The following tips can help you make the most efficient use of memory and get the best performance from Formula One.

- **Avoid formatting blank cells.** It is more efficient to format an entire row or column because no cells are created. When you format a blank range that does not consist of whole rows or whole columns, Formula One must create empty cells before it can apply the format. To find and eliminate blank formatted cells in your worksheet, show cell markers by using the **ShowTypeMarkers** property. (You can do this in the Workbook Designer by choosing Tools > Options, clicking the General tab, and checking the Show Markers box.) With markers turned on, Formula One will display a blue frame inside blank, formatted cells.
- **Build worksheets by rows instead of columns.** Formula One allocates memory by rows. You can save memory by building tables a row at a time, rather than a column at a time. For example, fill cells in row 1 before moving to row 2, and so on, rather than filling cells in column A before moving to column B, and so on.
- **Build ranges from the lower right corner.** When building a table one cell at a time from code, it is faster and more efficient to start in the lower right corner of the area in which you are working. This ensures that the row pointers are allocated simultaneously instead of one at a time. Likewise, each row is allocated once instead of being reallocated as each cell is added.
- **Use values instead of formulas whenever possible.**
- **Avoid adding empty rows and columns for white space.** Adjust the row height or column width to create white space instead of adding empty rows or columns. If you must have additional white space on your worksheet, empty rows are more efficient than empty columns.
- **Disable repainting when performing a series of operations.** When performing a number of sequential operations on a worksheet, disable repainting with the **Repaint** property so the screen does not repaint after each operation. This increases the speed of the operation and avoids unnecessary screen flashing.

- **Use methods to copy and move data.** Use **EditCopyRight**, **EditCopyDown**, **CopyRange**, **CopyRangeEx**, and **MoveRange** to copy and move cells. These methods are much faster than using the clipboard. In addition, these methods update cell references to maintain the integrity of your formulas.

Optimizing Formula One

Formula One allocates an array of row pointers for each row that has data. For each row, it then allocates an array of column pointers. Rows that do not have data in them only have a row pointer allocated for that row and not an array of column pointers for that row. Therefore, it is better to fill the sheet by rows instead of columns when possible.

Since the row and column pointers are allocated based upon the last row or column with data in them, it is best to load the sheet from the bottom-right up to the top-left. This allows Formula One to allocate the entire array of row and column pointers before it fills the sheet. With these arrays pre-allocated then it only has to allocate space for each cell depending on the cell size.

If the Formula One sheet is not loaded from the bottom-right to the top-left, then the arrays of row and column pointers must grow as the number of rows and columns grow. Each time these arrays need to grow, it must find a contiguous block of memory to hold the entire array. This leads to disk swapping and decreased speed.

If it is not possible to load the sheet from the bottom-right, then another way to pre-allocate the arrays is to put a value in the bottom-right hand corner of the sheet. This will pre-allocate the row pointers. In order to pre-allocate all the arrays of column pointers, the entire column on the right would need to be loaded with a value. This would be recommended if the number of columns is large. After the data has been loaded into the sheet, then the cell at the bottom-right can be deleted. The sheet will then re-evaluate the amount of memory that it needs and give back the rest. This will speed up load times considerably.

Understanding Formula One's Data Structure

There are three basic parts to a Formula One data structure: the row pointers, the cell pointers, and the cells.

Formula One's data structure starts with a set of row pointers. This row pointer array is a contiguous block of memory containing one 4-byte pointer for each row. For example, if you have a spreadsheet that contains 10 rows, the row pointer array contains 10 pointers of 4 bytes each. As you add rows, this array expands.

Each row that is not blank consists of an array of cell pointers large enough to point to the last cell in a row. The cell pointer array is a contiguous block of memory containing one 4-byte pointer for each cell. So, if the last cell in a row is located in column H (the eighth column), that row contains eight 4 byte pointers (for columns A through H).

Finally, there are the cells themselves, which are small data structures containing the cells contents. The structures include a cells value, its format, its formula, its font, its alignment, and other cell attributes. Cells only exist in memory if they contain formulas or data or are formatted differently from the row or column in which they are located.

Allocating and Freeing Memory

Generally speaking, it is an expensive operation to get memory from and return memory to Windows (referred to as allocating and freeing memory). Every time a spreadsheet has to obtain a new chunk of memory, time is consumed. Additionally, allocating chunks of memory in varying sizes tends to fragment the memory pool. At some point, you may need to obtain a piece of memory that is a certain size, and find that it does not exist, although there is plenty of free memory scattered throughout your system in small pieces.

Filling Worksheets with Data

Consider the case of filling a 1,000 row by 10 column spreadsheet with a set of values or formulas. If you start from the top left corner, the first cell you create causes the row pointer array to allocate one pointer to the first row. Then, the first cell pointer is allocated in row 1. And finally, the first cell (A1) is allocated. When you fill cell B1, the cell pointer array in row 1 increases to two pointers and cell B1 is created. This continues until all 10 cells in row one are created. If you are keeping score, you will note that Formula One performed 21 memory allocations: one for the row pointer, 10 for the cell pointers, and 10 for the cells themselves.

This operation can be optimized by filling the row from right to left. This causes the cell pointer array to be allocated once, eliminating 9 memory allocations (and the associated memory copying and freeing operations).

Continuing with the spreadsheet filling, row 1 is completed and you are ready to fill row 2. Before you start filling the row, the row pointer array is expanded to 2 pointers (one for each row). Then, row 2 is created just like row 1. Remember, our sample spreadsheet contains 1,000 rows. So, the row pointer array is expanded 1,000 times. This is unnecessary and very time consuming. By the time you reach the one-thousandth row, Formula One is looking for a block of memory 4,000 bytes long (since there is 4 bytes allocated for each row pointer).

This part of the operation can be optimized by filling the rows from bottom to top. This causes the row pointer array to be allocated once instead of 1,000 times.

Employing these two simple optimizations reduces the number of memory allocations from 21,000 (for 1,000 row pointers, 10,000 cell pointers, and 10,000 cells) to 11,001 (1 row pointer, 1,000 cell pointers, and 10,000 cells). But reducing the number of memory allocations is only half the story.

When you expand the row pointer array or a cell pointer array, Formula One must ask for a larger piece of memory, copy the contents of the current array into the new memory array, and then return the previous memory block to the memory pool. The 9,999 memory operations you eliminated are the most expensive operations.

How can this be? The secret is that the optimized method for filling the worksheet avoided dealing with fragmented memory, as happened with the slow method. As the cell pointer arrays grew larger, the remaining chunks of free memory were not big enough for the growing arrays. This caused Windows to start paging memory to disk to produce large enough memory chunks.

You can achieve the same results without actually filling a worksheet from bottom to top. When you start filling data, make sure the maximum-needed row pointer array is allocated by entering a cell in the last row you are going to use. This forces a row pointer array to be created that will accommodate your entire worksheet. Then, it doesn't matter what row you fill next because this array will not change. Likewise, if you always create the last cell in any row first, the maximum-needed cell pointer array is created. Then, you can enter the cells in that row in any order without affecting performance.

Using Technical Specifications

The following table lists the technical specifications for the Formula One control.

Specifications

Maximum worksheet size	65,536 rows by 256 columns
Column width	0 to 255 characters
Row height	0 to 409 points
Text length	16,383 characters
Formula length	1024 characters
Number precision	15 digits
Largest positive number	9.999999999999999E307
Largest negative number	-9.999999999999999E307
Smallest positive number	1E-307
Smallest negative number	-1E-307
Maximum number of iterations	32,767
Maximum number of colors	56
Maximum number of available colors	Limited by display card and monitor
Maximum number of fonts per workbook	256
Maximum number of selected ranges	2048
Maximum number of names per workbook	Limited by available memory
Maximum length of name	255
Maximum number of function arguments	30

Specifications

Maximum length of format string	255
Maximum number of tables (workbooks)	Limited by system resources (windows and memory)
Excel file format version	Excel 5.0, 95, and 97

Tidestone

CHAPTER 14

Creating Add-In Functions

This chapter provides information about add-in functions—small programs that extend the capabilities of Formula One. The examples presented here demonstrate how to extend Formula One’s functions by adding the capability to use array arguments in functions using Visual Basic and C++.

Formula One ActiveX Add-Ins in Visual Basic

General Design Principles

Formula One implements Add-Ins as ActiveX DLLs.

To function as a Formula One Add-In, an ActiveX DLL must, at a minimum, implement a creatable class named **F1Functions**. When the DLL is registered with Formula One either through the API or through the Add-In Manager dialog, Formula One attempts to locate and create this class. Formula One only recognizes the DLL as a valid add-in if the DLL implements a creatable **F1Functions** class.

Note In Visual Basic, the standard way to make a class creatable in an ActiveX DLL is to set the class’s **Instancing** property to a value of 5. Visual Basic describes such a class as *MultiUse*.

Formula One treats any member of the **F1Functions** class that follows the guidelines set forth in this chapter as a worksheet function and ignores any functions not meeting the guidelines.

All arguments must also be passed ByVal.

Note “ByVal” means “by value.” When a parameter is passed by value, anyone referencing that parameter is prevented from modifying that variable’s value.

As Formula One's add-in capabilities increase, future versions may look for new classes in addition to **F1Functions**. To maximize compatibility with future versions of Formula One, add-in developers should refrain from using the prefix **F1** on internal class names in their add-in DLLs—especially on classes that are creatable.

Formula One presents the ActiveX DLL's project description to end-users via the Add-In Manager dialog; therefore, the project description should be an informative phrase (typically no longer than a short sentence) describing the add-in DLL. The Project Description field can be found on Visual Basic's Project Properties dialog under the General page.

Thread Safety

Although Formula One is single-threaded, future versions of Formula One may support multithreading. If so, Formula One's calc engine will create a separate instance of **F1Functions** for each thread. This means that if your add-in functions use only automatic variables or data defined within the **F1Functions** class, you will not need to be concerned about thread safety. However, if you must refer to data outside of **F1Functions** (for example, if all threads must share common data), you are responsible for ensuring that your code is thread-safe.

Add-In Function Requirements

All arguments to an add-in function must be passed **ByVal** or the function will not be recognized by Formula One. To use the **F1AddInArray** and **F1AddInArrayEx** types, the Formula One control must be added to the project.

Formula One recognizes functions that use the following data types (all arguments must be **ByVal**):

Type	Allowed as Argument	Allowed as Return Value
Boolean	Yes	Yes
Double	Yes	Yes
F1AddInArray	Yes	No
F1AddInArrayEx	Yes	No
String	Yes	Yes
Variant*	Yes	Yes

*Use **Variant** when a single function can accept or return values of different types. A **Variant** may also be of type **vbEmpty**. A **Variant** argument cannot accept a reference to more than one cell. If you need to accept such references, you may use **F1AddInArray** or **F1AddInArrayEx** instead.

F1AddInArray

Use the type **F1AddInArray** when using an area reference as an argument. You must add the Formula One control to your project to use this type. **F1AddInArray** will only accept a “simple” reference—a reference to a single 2d area. If you specify a 3d area or union reference, the formula evaluator returns **#VALUE!** without calling the add-in function. Use **F1AddInArrayEx** for support of 3d area or union references.

The members of **F1AddInArray** are:

```
Function Rows() As Long

Function Cols() As Long

Function GetArrayType() As Long

Function GetValue(ByVal Row As Long, ByVal Col As Long)

Function IterStart() As Boolean

Function IterNext() As Boolean

Function IterGetValue() As Variant

Function IterGetValueEx(Row As Long, Col As Long) As Variant
```

F1AddInArray.GetValue always returns **F1AddIn2dArea**.

You may use **IterStart** and **IterNext** in a loop to iterate through the non-empty elements of the array. This can be much faster than examining each element individually when a sparsely populated array is expected.

Example Code

```
Dim Found As Boolean
Found = TheArray.IterStart()
While Found

    ' Use IterGetValue or IterGetValueEx to
    ' retrieve the value of the current element.

    Found = TheArray.IterNext()
Wend
```

F1AddInArrayEx

You may also use the type **F1AddInArrayEx** when using an area reference as an argument. Like **F1AddInArray**, **F1AddInArrayEx** accepts “simple” 2d area references. However, **F1AddInArrayEx** also accepts 3d area and union references. To only allow 2d area references, use **F1AddInArray**.

You must add the Formula One control to your project to use this type.

The members of **F1AddInArrayEx** are:

```
Function Areas() As Long ' Number of Areas in Array
Function Rows(ByVal Area As Long) As Long 'Nbr Rows in Area
Function Cols(ByVal Area As Long) As Long 'Nbr Cols in Area
Function GetArrayType() As Long
Function GetValue(ByVal Area As Long, ByVal Row As Long, _
                  ByVal Col As Long)
Function IterStart() As Boolean
Function IterNext() As Boolean
Function IterGetValue() As Variant
Function IterGetValueEx(Area As Long, Row As Long, Col As Long) _
                        As Variant
```

F1AddInArrayEx.GetArrayType returns **F1AddIn2dArea**, **F1AddIn3dArea** or **F1AddInRegion**.

Note You may use **IterStart** and **IterNext** in a loop to iterate through the non-empty elements of the array. This can be much faster than examining each element individually when a sparsely populated array is expected.

Visual Basic Example Add-Ins

Example 1

```
' =ADDTHESE(1,2)
' All arguments must be ByVal
Function AddThese(ByVal X As Double, ByVal Y As Double) _
As Double

AddThese = X + Y
End Function
```


Example 2

```
' =CONCATENATETHESE("ABC","XYZ")
' All arguments must be ByVal
Function ConcatenateThese(ByVal X As String, _
    ByVal Y As String) As String

    ConcatenateThese = X + Y
End Function
```

Example 3

```
' =SUMOFRANGE(A1:C5)
' All arguments must be ByVal
Function SumOfRange(ByVal Range As FlAddInArrayEx) _
    As Double

    On Error GoTo ErrorHandler

    Dim Sum As Double
    Sum = 0

    Dim GotOne As Boolean

    GotOne = Range.IterStart
    While GotOne
        Sum = Sum + CDb1(Range.IterGetValue)
        GotOne = Range.IterNext
    Wend

    SumOfRange = Sum
    Exit Function

ErrorHandler:
    Err.Raise FlAddInValueError

End Function
```

Example 4

```
' =MAKEERROR(TRUNC(RAND()*7)+1)
' All arguments must be ByVal
Function MakeError(ByVal WhichOne As Double) _
    As Double

    Select Case WhichOne
        Case 1
            Err.Raise FlAddInNullError ' #NULL!

        Case 2
            Err.Raise FlAddInDivZeroError ' #DIV/0!
```

```
        Case 4
            Err.Raise FlAddInRefError ' #REF!

        Case 5
            Err.Raise FlAddInNameError ' #NAME?

        Case 6
            Err.Raise FlAddInNumError ' #NUM!

        Case 7
            Err.Raise FlAddInNaError ' #N/A

        Case Else
            Err.Raise FlAddInValueError ' #VALUE!

    End Select
End Function
```

Formula One C++ Add-In API

How Add-In Functions Are Declared

Formula One Add-Ins written in C++ implement each add-in function as an individual callback function. They are “callback” functions because they will not be exported by the DLL—they will be enumerated by a single exported function called `F1AddInInit`.

Each add-in function implementation must adhere to the following declaration:

```
HRESULT CALLBACK F1AddInFunction(  
    LPVARIANTARG    pResult,  
    int              nReserved,  
    int              nArgs,  
    LPVARIANTARG    pArgs);
```

Where:

`pResult` is a pointer to the variant that holds the add-in function’s result.

`nReserved` is reserved for future use. The add-in function should ignore this argument.

`nArgs` is the number of arguments being passed to the add-in function.

`pArgs` is a pointer to an array of `nArgs` variants, which are the arguments being passed to the add-in function. The first argument is at index 0, the second is at index 1, etc. If `nArgs` is 0, then this pointer is invalid.

Note Formula One’s formula evaluator always clears the variants pointed to by *pResult* and *pArgs*. The add-in function should not clear these.

Return Value

The only valid return values for an add-in function are `S_OK` and `E_OUTOFMEMORY`. To report a formula evaluation error, set *pResult*’s type to `VT_ERROR` and its value to one of the error codes listed in “Formula Evaluation Errors” on page 198, and return `S_OK`.

How Add-In Functions Are Exposed to Formula One

The following functions are used to initialize the plug-in for use within Formula One.

F1AddinInit

The add-in DLL must export the **F1AddinInit** function, which is declared as follows:

```
extern "C"
{
    HRESULT __declspec(dllexport) __stdcall F1AddinInit(
        F1AddinRegisterInfoProc      RegisterInfoProc,
        F1AddinRegisterFunctionProc  RegisterFunctionProc,
        int                          nReserved1,
        int                          nReserved2);
};
```

Where:

`RegisterInfoProc` points to a callback function implemented by Formula One. **F1AddinInit** calls this function to provide global information about the add-in DLL such as its name and a short description.

`RegisterFunctionProc` also points to a callback function implemented by Formula One. **F1AddinInit** provides information about the DLL's add-in functions to Formula One by calling this function once for each add-in function that the DLL implements.

`nReserved1` and `nReserved2` are reserved for future use. The add-in should ignore these arguments.

F1AddinRegisterInfoProc

This function is called from the add-in's **F1AddinInit** code to register information about the add-in. A pointer to this callback is paired to the add-in through the **F1AddinInit** functions.

```
HRESULT CALLBACK F1AddinRegisterInfoProc(
    LPWSTR  pwszName,
    LPWSTR  pwszDescription,
    int     nReserved1,
    int     nReserved2);
```

Where:

`pwszName` is a Unicode string specifying the name of the add-in DLL. This should be a short name of (typically) one or two words.

`pwszDescription` is a Unicode string describing the add-in DLL. This should be a short descriptive phrase, no longer than a single sentence.

`nReserved1` and `nReserved2` must be zero so that the add-in ignores these arguments.

F1AddInRegisterFunctionProc

Each function in the add-in must be registered by calling the `F1AddInRegisterFunctionProc` function which is paired to the **F1AddinInit** entry point.

```
HRESULT CALLBACK F1AddInRegisterFunctionProc(
    LPWSTR                pwszName,
    int                   nReserved,
    F1AddInFunction        pFunction,
    int                   nArgs);
```

Where:

`pwszName` is a Unicode string specifying the name of the add-in function. It is not case sensitive.

`nReserved` must be zero.

`pFunction` points to an add-in function declared according to the description in this chapter.

`nArgs` is the number of arguments expected by the function. If the add-in function can accept a variable number of arguments, this should be `-1`; otherwise, Formula One will not attempt to call the function with any number of arguments other than `nArgs`.

How Arguments and Return Values Are Passed

Arguments and return values are passed between Formula One and the add-in function using the **VARIANT** structure. In Formula One, an argument may have any of the following value types:

```
VT_EMPTY
VT_R8
VT_BSTR
VT_BOOL
VT_ERROR
VT_UNKNOWN
```

Formula One accepts any of the above value types as an add-in function's result except **VT_UNKNOWN**.

When an add-in function encounters an argument of type **VT_UNKNOWN**, it should use **QueryInterface** to obtain an interface it can use to retrieve the data. At this time, the only interfaces implemented for this purpose are **IF1AddInArray** and **IF1AddInArrayEx**.

- Query for **IF1AddInArray** on an argument where you only want to accept two-dimensional area references.
- Query for **IF1AddInArrayEx** on an argument where you want to accept three-dimensional area references. This interface also provides all the functionality of **IF1AddInArray**.

If the add-in function fails to obtain a suitable interface through **QueryInterface**, it should return the formula evaluation error code **F1_E_VALUE**. See “General Design Principles” on page 183, for information on returning formula evaluation errors.

IF1AddInArray interface

When an argument to an add-in function is a two-dimensional area reference, Formula One passes that reference to the add-in function as an **IF1AddInArray** interface. (The **IF1AddInArrayEx** interface is also implemented; the add-in function may use whichever interface best suits its needs.)

IF1AddInArray, not technically an array, provides a mapping mechanism for exposing a range in the workbook. In future versions of Formula One, the same interface may expose true arrays to the add-in function.

This section describes the members of **IF1AddInArray**.

IF1AddInArray::Rows

Returns the number of rows in the array.

```
int Rows(void);
```

IF1AddInArray::Cols

Returns the number of columns in the array.

```
int Cols(void);
```

IF1AddInArray::GetArrayType

Returns the type of the array.

```
Int GetArrayType(void);
```

Return Value

IF1AddinArray returns one of the following constants:

F1ADDIN_AREA Formula One returns this constant when the argument is a range reference, as in the formula `=SUM(A1:C5)`.

F1ADDIN_ARRAY Not implemented in Formula One, this constant would represent an array, as in the Excel formula `=SUM({1,2,3;4,5,6})`.

IF1AddInArray::GetValue

Retrieves the value of the specified element in the array.

```
HRESULT GetValue(
    int          nRow,
    int          nCol,
    LPVARIANTARG pResult);
```

Where:

`nRow` is the row index of the element to be retrieved.

`nCol` is the column index of the element to be retrieved.

`pResult` points to the variant that will receive the element's value.

Return Value

If `nSheet`, `nRow`, or `nCol` fall outside of the array's boundaries, the return value is `F1_E_REF`. In this case, `pResult`'s variant is also set to `F1_E_REF`. If the function succeeds, Formula One returns `S_OK`, and sets `pResult`'s variant the value of the specified element in the array.

IF1AddInArray::IterStart

IF1AddInArray::IterNext

Iterates through the non-empty elements of the array.

```
BOOL IterStart(void);
BOOL IterNext(void);
```

To quickly iterate through the non-empty elements of an array, call

IF1AddInArray::IterStart for the first element and **IF1AddInArray::IterNext** for each additional element until either of these functions returns **FALSE**. During each iteration, call **IF1AddInArray::IterGetValue** or **IF1AddInArray::IterGetValueEx** to retrieve the actual data. Formula One returns the elements in an arbitrary order.

Routines that care about the location of the data should use

IF1AddInArray::IterGetValueEx, which indicates the row and column of the current element.

Using these functions may be much faster than individually examining each element in situations where a sparsely-populated array is expected.

Return Value

TRUE if successful; FALSE if there are no more non-empty elements.

```
for (BOOL bGotOne = pAddInArray->IterStart();
     bGotOne;
     bGotOne = pAddInArray->IterNext())
{
    // use IterGetValue or IterGetValueEx to
    // get the value of the current element
}
```

IF1AddInArray::IterGetValue**IF1AddInArray::IterGetValueEx**

Retrieves the value of the array element last selected by **IF1AddInArray::IterStart** or **IF1AddInArray::IterNext**.

```
BOOL IterGetValue(LPVARIANTARG    pResult);
```

```
BOOL IterGetValueEx(
    LPINT    pRow,
    LPINT    pCol,
    LPVARIANTARG    pResult);
```

Where:

pRow points to an integer that will receive the element's row index.

pCol points to an integer that will receive the element's column index.

pResult points to the variant that will receive the element's value.

Return Value

If the last value returned by **IF1AddInArray::IterStart** or **IF1AddInArray::IterNext** was TRUE, the return value is TRUE.

If the last value returned by **IF1AddInArray::IterStart** or **IF1AddInArray::IterNext** was FALSE, or if neither of those functions has yet been called, the return value is FALSE. When the return value is FALSE, no values will be placed in *pRow*, *pCol*, or *pResult*.

IF1AddInArrayEx interface

When an argument to an add-in function is a two- or three- dimensional area reference or a union, Formula One passes that reference or union to the add-in function as an **IF1AddInArrayEx** interface. (For two-dimensional area references, Formula One also implements the **IF1AddInArray**; the add-in function may use the interface that best suits its needs.)

IF1AddInArrayEx, not technically an array, provides a mapping mechanism for exposing a range in the workbook. In future versions of Formula One, the same interface may expose true arrays to the add-in function.

This section describes the members of **IF1AddInArrayEx**.

IF1AddInArrayEx::Areas

Returns the number of areas in the array.

```
int Areas(void);
```

IF1AddInArrayEx::Rows

Returns the number of rows in an area of the array.

```
int Rows(  
    int    nArea);
```

Where:

`nArea` is the index of the area whose row count will be returned.

IF1AddInArrayEx::Cols

Returns the number of columns in an area of the array.

```
int Cols(  
    int    nArea);
```

Where:

`nArea` is the index of the area whose column count will be returned.

IF1AddInArrayEx::GetArrayType

Returns the type of the array.

```
int GetArrayType(void);
```

Return Value

IF1AddInArrayEx returns one of the following constants:

F1ADDIN_AREA: Formula One returns this constant when the argument is a 2D area reference, as in the formula `=SUM(A1:C5)`.

F1ADDIN_AREA3D: Formula One returns this constant when the argument is a 3D area reference, as in the formula `=SUM(Sheet1:Sheet3!A1:C5)`.

F1ADDIN_ARRAY: Not implemented in Formula One, this constant would represent an array, as in the Excel formula `=SUM({1,2,3;4,5,6})`.

F1ADDIN_REGION: Formula One returns this constant when the argument is a region reference, as in the formula
`=SUM((A1 : C5 , A10 : C12))`.

IF1AddInArrayEx::GetValue

Retrieves the value of the specified element in the array.

```
HRESULT GetValue(
    int          nArea,
    int          nRow,
    int          nCol,
    LPVARIANTARG pResult);
```

Where:

`nArea` is the area index of the element to be retrieved.

`nRow` is the row index of the element to be retrieved.

`nCol` is the column index of the element to be retrieved.

`pResult` points to the variant that will receive the element's value.

Return Value

If `nArea`, `nRow`, or `nCol`, fall outside of the array's boundaries, the return value is `F1_E_REF`. In this case, `pResult`'s variant is also set to `F1_E_REF`. If the function succeeds, the return value is `S_OK`, and `pResult`'s variant is set to the value of the specified element in the array.

IF1AddInArray::IterStart

IF1AddInArray::IterNext

Iterates through the non-empty elements of the array.

```
BOOL IterStart(void);
BOOL IterNext(void);
```

To quickly iterate through the non-empty elements of an array, call

IF1AddInArrayEx::IterStart for the first element and

IF1AddInArrayEx::IterNext for each additional element until either of these functions returns **FALSE**. During each iteration, call

IF1AddInArrayEx::IterGetValue or **IF1AddInArrayEx::IterGetValueEx** to retrieve the actual data.

Formula One returns the elements in an arbitrary order.

Routines that care about the location of the data should use

IF1AddInArrayEx::IterGetValueEx, which indicates the row and column of the current element.

Using these functions may be much faster than individually examining each element in situations where a sparsely-populated array is expected.

Return Value

TRUE if successful; FALSE if there are no more non-empty elements.

Example

```
for (BOOL bGotOne = pAddInArrayEx->IterStart();
     bGotOne;
     bGotOne = pAddInArrayEx->IterNext())
{
    // use IterGetValue or IterGetValueEx to
    // get the value of the current element
}
```

IF1AddInArrayEx::IterGetValue IF1AddInArrayEx::IterGetValueEx

Retrieves the value of the array element last selected by **IF1AddInArrayEx::IterStart** or **IF1AddInArrayEx::IterNext**.

```
BOOL IterGetValue(LPVARIANTARG pResult);
```

```
BOOL IterGetValueEx(
    LPINT      pArea,
    LPINT      pRow,
    LPINT      pCol,
    LPVARIANTARG pResult);
```

Where:

pArea points to an integer that will receive the element's area index.

pRow points to an integer that will receive the element's row index.

pCol points to an integer that will receive the element's column index.

pResult points to the variant that will receive the element's value.

Return Value

If the last value returned by **IF1AddInArrayEx::IterStart** or **IF1AddInArrayEx::IterNext** was TRUE, the return value is TRUE.

If the last value returned by **IF1AddInArrayEx::IterStart** or **IF1AddInArrayEx::IterNext** was FALSE, or if neither of those functions has yet been called, the return value is FALSE. When the return value is FALSE, no values will be placed in *pRow*, *pCol*, or *pResult*.

Formula Evaluation Errors

An argument to or return value of an add-in function of the type VT_ERROR indicates a formula evaluation error. The errors recognized by Formula One are:

Value	Error Type
F1_E_NULL	#NULL!
F1_E_DIVZERO	#DIV/0!
F1_E_VALUE	#VALUE!
F1_E_REF	#REF!
F1_E_NAME	#NAME?
F1_E_NUM	#NUM!
F1_E_NA	#N/A

C++ Example Add-In

```
#include <windows.h>

#include <initguid.h>
#include "fladdin.h"

.
.
.
////////////////////////////////////
// RETURN.ME function (::ReturnMe)
//
// RETURN.ME accepts a single argument and returns that argument's
// value. If the argument is a range reference, the top-left cell's
// value is returned.
//
HRESULT CALLBACK ReturnMe(
    LPVARIANTARG pResult,    // Pointer to the VARIANT receiving
                             // the function's result
    int,                    // Ignore this argument; it's
                             // reserved for future use
    int nArgs,               // Number of arguments
    LPVARIANTARG pArgs)      // Pointer to an array of nArgs
                             // VARIANTS, each corresponding to
                             // an argument being passed to the
                             // add-in function. pArgs[0] is the
                             // first (left-most) argument.

{
    if (nArgs == 1)
    {
        // VT_UNKNOWN requires special handling
        if (pArgs[0].vt == VT_UNKNOWN)
        {
```

```

        // We handle arrays; we don't know about any other
        // interfaces, so all others are ignored here.
        IF1AddInArrayEx* pArray;
        if (SUCCEEDED(pArgs[0].punkVal->QueryInterface(
            IID_IF1AddInArrayEx, (LPVOID*)&pArray)))
        {
            // Copy the value of the top-left cell of the first
            // sheet or range to the result.
            pArray->GetValue(0, 0, 0, pResult);
            pArray->Release();

            return S_OK;
        }
    }
    else
        // Copy the argument's value to the result.
        return ::CheckReturnValue(::VariantCopy(pResult,
            pArgs));
    }
    return ::MakeErrorResult(pResult, F1_E_VALUE);
}

////////////////////////////////////
// GET.IN.RANGE function (::GetInRange)
//
// GET.IN.RANGE returns an item at a specified location
// with a range.
// Formula                                Cell Returned
// -----
// =GET.IN.RANGE(A11:C13,1,1)              A11
// =GET.IN.RANGE(A11:C13,2,3)              C12
// =GET.IN.RANGE(Sheet1:Sheet3!A11:C13,2,1,3) Sheet2!C11
// =GET.IN.RANGE((A1:C3,A11:C13),1,3,2)    B3
//
HRESULT CALLBACK GetInRange( LPVARIANTARG pResult, int, int nArgs,
                             LPVARIANTARG pArgs)
{
    if (nArgs == 3)
    {
        IF1AddInArray* pArray;

        // Try to get the array interface
        if (pArgs[0].vt == VT_UNKNOWN && SUCCEEDED(
            pArgs[0].punkVal->QueryInterface(IID_IF1AddInArray,
                (LPVOID*)&pArray)))
        {
            // Both coordinates must be of type double
            if (pArgs[1].vt == VT_R8 && pArgs[2].vt == VT_R8)
            {
                // If the coordinates are invalid, GetValue will put
                // #REF! in pResult, which is what we want.

```

```

        pArray->GetValue((int)pArgs[1].dblVal - 1,
                        (int)pArgs[2].dblVal - 1, pResult);
        pArray->Release();
        return S_OK;
    }
    // Release the array; we can't use it.
    pArray->Release();
}
}
else if (nArgs == 4) // Four arguments: a three-dimensional
                    // array
{
    IF1AddInArrayEx* pArray;

    // Try to get the array interface
    if (pArgs[0].vt == VT_UNKNOWN && SUCCEEDED(
        pArgs[0].punkVal->QueryInterface(IID_IF1AddInArrayEx,
                                          (LPVOID*)&pArray)))
    {
        // All coordinates must be of type double
        if (pArgs[1].vt == VT_R8 && pArgs[2].vt == VT_R8 &&
            pArgs[3].vt == VT_R8)
        {
            pArray->GetValue((int)pArgs[1].dblVal - 1,
                            (int)pArgs[2].dblVal - 1,
                            (int)pArgs[3].dblVal - 1, pResult);
            pArray->Release();
            return S_OK;
        }
        pArray->Release();
    }
}
return ::MakeErrorResult(pResult, F1_E_VALUE);
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// F1AddInInit
//
// This exported function (see AddIn.def) registers the add-in's
// functions and other general information with Formula One.
// Formula One calls this function when it loads the add-in.
//
HRESULT __stdcall F1AddInInit(
    F1AddInRegisterInfoProc    RegisterInfoProc,
    F1AddInRegisterFunctionProc RegisterFunctionProc,
    int,
    int)
{
    HRESULT hr = S_OK;

    if (SUCCEEDED(hr))

```

```

        hr = RegisterFunctionProc(
            L"RETURN.ME", // Name of the function as seen by the
                        // end-user
            0,           // Reserved for future use; must be 0
            ReturnMe,    // Name of the function as implemented
                        // in the DLL
            1);          // Number of arguments expected.

    if (SUCCEEDED(hr))
        hr = RegisterFunctionProc(
            L"GET.IN.RANGE",
            0,
            GetInRange,
            -1);

    if (SUCCEEDED(hr))
        hr = RegisterInfoProc(
            FlAddInName, // Name of the add-in (defined at top
                        // of the file)
            FlAddInDesc, // Description of the add-in
            0,           // Reserved for future use; must be 0
            0);          // Reserved for future use; must be 0

    return hr;
}
////////////////////////////////////
// DllMain
//
// Initialization may be done here or in FlAddInInit--whichever best
// suits your needs. See the Win32 API doc for more details about
// this function.
//
BOOL APIENTRY DllMain(HANDLE hModule, DWORD ul_reason_for_call,
                     LPVOID lpReserved)
{
    return TRUE;
}

```

Tidestone

CHAPTER 14

A-Z Worksheet Function Reference

This chapter provides a complete alphabetical reference for the Formula One worksheet functions. Refer to **Working With Data** in this manual for additional information about using worksheet functions.

ABS

Description Returns the absolute value of a number.

Syntax **ABS** (*number*)

Parameter	Description
<i>number</i>	Any number.

Remarks An absolute value does not display a positive or negative sign.

Examples These functions both return 1:

ABS(−1)
ABS(1)

See Also **SIGN**

ACOS

Description Returns the arc cosine of a number.

Syntax **ACOS** (*number*)

Parameter	Description
<i>number</i>	The cosine of the angle. The cosine can range from 1 to −1.

Remarks	The resulting angle is returned in radians (from 0 to π). To convert the resulting radians to degrees, multiply the radians by 180/PI().
Examples	<p>This function returns 1.05:</p> <p>ACOS(.5)</p> <p>This function returns 1.77:</p> <p>ACOS(- .2)</p>
See Also	COS

ACOSH

Description	Returns the inverse hyperbolic cosine of a number.				
Syntax	ACOSH (<i>number</i>)				
	<table><tr><th>Parameter</th><th>Description</th></tr><tr><td><i>number</i></td><td>Any number equal to or greater than 1.</td></tr></table>	Parameter	Description	<i>number</i>	Any number equal to or greater than 1.
Parameter	Description				
<i>number</i>	Any number equal to or greater than 1.				
Examples	<p>This function returns .62:</p> <p>ACOSH(1.2)</p> <p>This function returns 1.76:</p> <p>ACOSH (3)</p>				
See Also	ASINH ATANH COSH				

ADDRESS

Description Creates a cell address as text.

Syntax **ADDRESS** (*row*, *column*, *ref_type* [, *a1*] [, *sheet*])

Parameter	Description										
<i>row</i>	The row number for the cell address.										
<i>column</i>	The column number for the cell address.										
<i>ref_type</i>	The cell reference type. Following are the valid values for this argument. <table><tr><th>Value</th><th>Description</th></tr><tr><td>1</td><td>Absolute</td></tr><tr><td>2</td><td>Absolute row, relative column</td></tr><tr><td>3</td><td>Relative row, absolute column</td></tr><tr><td>4</td><td>Relative</td></tr></table>	Value	Description	1	Absolute	2	Absolute row, relative column	3	Relative row, absolute column	4	Relative
Value	Description										
1	Absolute										
2	Absolute row, relative column										
3	Relative row, absolute column										
4	Relative										
<i>a1</i>	The reference format. This argument must be TRUE() to represent an A1 reference format; Formula One does not support the R1C1 reference format.										
<i>sheet</i>	The name of an external worksheet view control. Omitting this argument assumes that the reference exists in the current spreadsheet.										

Examples This function returns \$F\$5:
`ADDRESS(5, 6, 1)`
This function returns SALES!F5:
`ADDRESS(5, 6, 4, TRUE(), SALES.)`

See Also **COLUMN**
OFFSET
ROW

AND

Description Returns True if all arguments are true; returns False if at least one argument is false.

Syntax **AND** (*logical_list*)

Parameter	Description
<i>logical_list</i>	A list of conditions separated by commas. You can include as many as 30 conditions in the list. The list can contain logical values or a reference to a range containing logical values. Text and empty cells are ignored. If there are no logical values in the list, the error #VALUE! is returned.

Examples This function returns True because both arguments are true:

AND(1+1=2, 5+5=10)

This function returns False:

AND(TRUE(), FALSE())

See Also **NOT**
OR
ROW

ASC

Description In DBCS (Far-East) systems, this functions returns a copy of text in which the double-byte characters are converted to single-byte characters, if possible. Characters that cannot be converted are left unchanged.

Syntax **ASC** (*text*)

Parameter	Description
<i>text</i>	Text converted from double-byte characters to single-byte characters.

Remarks On non-DBCS systems, the text returns unchanged.

See Also **DBCS**

ASIN

Description Returns the arcsine of a number.

Syntax **ASIN** (*number*)

Parameter	Description
<i>number</i>	The sine of the resulting angle, ranging from -1 to 1.

Remarks The resulting angle is returned in radians (ranging from $-\pi/2$ to $\pi/2$). To convert the resulting radians to degrees, multiply the radians by 180/PI().

Examples This function returns 1.57:

ASIN(1)

This function returns .41:

ASIN(.4)

See Also **ASINH**
PI
SIN

ASINH

Description Returns the inverse hyperbolic sine of a number.

Syntax **ASINH** (*number*)

Parameter	Description
<i>number</i>	Any number.

Examples This function returns 2.37:

ASINH(5.3)

This function returns -2.09:

ASINH(-4)

See Also **ACOSH**
ASIN
ATANH
SINH

ATAN

Description Returns the arctangent of a number.

Syntax **ATAN** (*number*)

Parameter	Description
<i>number</i>	The tangent of the angle.

Remarks The resulting angle is returned in radians, ranging from $-\pi/2$ to $\pi/2$. To convert the resulting radians to degrees, multiply the radians by $180/\text{PI}()$.

Examples This function returns 1.29:

ATAN(3.5)

This function returns -1.33 :

ATAN(4)

See Also **ATAN2**
ATANH
PI
TAN

ATAN2

Description Returns the arctangent of the specified coordinates.

Syntax **ATAN2** (*x*, *y*)

Parameter	Description
<i>x</i>	The x coordinate.
<i>y</i>	The y coordinate.

Remarks The arctangent is the angle from the x axis to a line with end points at the origin (0, 0) and a point with the given coordinates (*x*, *y*). The angle is returned in radians, ranging from $-\pi$ to π , excluding $-\pi$.

Examples This function returns 1.11:

ATAN2(3, 6)

This function returns 3.04:

ATAN2(-1, .1)

See Also **ATAN2**

ATANH
PI
TAN

ATANH

Description Returns the inverse hyperbolic tangent of a number.

Syntax **ATANH** (*number*)

Parameter	Description
<i>number</i>	A number between –1 and 1, excluding –1 and 1.

Examples This function returns .55:

ATANH(.5)

This function returns –.26:

ATANH(–.25)

See Also **ACOS**
ASINH
TANH

AVERAGE

Description Returns the average of the supplied numbers. The result of **AVERAGE** is also known as the arithmetic mean.

Syntax **AVERAGE** (*number_list*)

Parameter	Description
<i>number_list</i>	A list of numbers separated by commas. As many as 30 numbers can be included in the list, and the list can contain numbers or a reference to a range that contains numbers. Text, logical expressions, or empty cells in a referenced range are ignored. All numeric values (including 0) are used.

Examples This function returns 8.25:

AVERAGE(5, 6, 8, 14)

This function returns 134, the average of the values in the range C15:C17:

AVERAGE(C15:C17)

See Also **MIN**
MAX

CALL

Description Calls a procedure in a dynamic link library. There are two syntax forms of this function. When **CALL** is used with **REGISTER.ID**, as shown in syntax 1, the DLL is loaded and remains loaded until the program is dismissed. When **CALL** is used alone, as shown in syntax 2, the DLL is loaded, the function is called, and then the DLL is unloaded.

Important This function is provided for advanced users only. If you use the **CALL** function incorrectly, you could cause errors that will require you to restart your computer.

Syntax 1 Used with **REGISTER.ID**
CALL(*register_id*, *argument1*, ...)

Syntax 2 Used alone
CALL(*module_text*, *procedure*, *type_text*, *argument1*, ...)

Parameter	Description
<i>register_id</i>	The value returned by a previously executed REGISTER.ID function.
<i>argument1</i>	The arguments to be passed to the procedure.
<i>module_text</i>	Quoted text or reference specifying the name of the dynamic link library (DLL) that contains the procedure.
<i>procedure</i>	Text specifying the name of the function in the DLL in Formula One. The function name is case dependent in Formula One.
<i>type_text</i>	Text specifying the data type of the return value and the data types of all arguments to the DLL or code resource. The first letter of <i>type_text</i> specifies the return value. The data types you use for <i>type_text</i> are described in the following table.

Data Type	Description	Pass by	C Declaration
A	Logical (False = 0, True = 1)	Value	short int
B	IEEE 8-byte floating point number	Value	double
C	Null-terminated string (255 characters maximum)	Reference	char*
D	Byte-counted string (first byte contains string length; 255 characters maximum)	Reference	unsigned char*
E	IEEE 8-byte floating point number	Reference	double*
F	Null-terminated string (255 characters maximum)	Reference	char*
G	Byte-counted string (first byte contains string length; 255 characters maximum)	Reference	unsigned char *
H	Unsigned 2-byte integer	Value	unsigned short int
I	Signed 2-byte integer	Value	short int
J	Signed 4-byte integer	Value	long int
L	Logical (False = 0, True = 1)	Reference	short int *
M	Signed 2-byte integer	Reference	short int *
N	Signed 4-byte integer	Reference	long int *

Remarks

For declarations made in C, it is assumed that your compiler defaults to 8-byte doubles, 2-byte short integers, and 4-byte long integers. In the Windows programming environment, all pointers should be far pointers.

Pascal calling conventions are used for all functions called from DLLs. For most C compilers, you must add the **-Pascal** keyword to the function declaration.

If the return value for your custom function uses a pass-by-reference data type, a null pointer can be passed as the return value. The null pointer is interpreted as the #NUM! error value.

For the F and G data types, a custom function can modify an allocated string buffer. If the return value type code is F or G, the value returned by the function is ignored. The list of function arguments is searched for the first data type that corresponds to the return value type. The current contents of the allocated string buffer is taken for the return value. 256 bytes is allocated for the argument; therefore, a function can return a larger string than it receives.

You can use a single digit (n), with a value from 1 to 9, as the code for data_type. The variable in the location pointed to by the nth argument is modified instead of the return value; this process is referred to as modifying in place. The nth argument must be a pass-by-reference data type. In addition, you must declare the function void. For most C compilers, you can add the Void keyword to the function declaration.

Examples

Syntax 1

The following macro formula registers the GetTickCount function from 32-bit Microsoft Windows. GetTickCount returns the number of milliseconds that have elapsed since Microsoft Windows was started.

```
REGISTER.ID("Kernel32","GetTickCount","J")
```

Assuming that this **REGISTER.ID** function is in cell A5, after your macro registers GetTickCount, you can use the **CALL** function to return the number of milliseconds that have elapsed since Windows was started:

```
CALL(A5)
```

Syntax 2

On a worksheet, you can use the following **CALL** formula (syntax 2) to call the GetTickCount function:

```
CALL("Kernel32","GetTickCount","J!")
```

The ! in the type_text argument forces Formula One to recalculate the **CALL** function every time the worksheet recalculates. This updates the elapsed time whenever the worksheet recalculates.

CEILING

Description

Rounds a number up to the nearest multiple of a specified significance.

Syntax

CEILING (*number*, *significance*)

Parameter	Description
<i>number</i>	The value to round.
<i>significance</i>	The multiple to which to round.

Remarks

Regardless of the sign of the number, the value is rounded up, away from zero. If *number* is an exact multiple of *significance*, no rounding occurs.

If *number* or *significance* is non-numeric, the error #VALUE! is returned. When the arguments have opposite signs, the error #NUM! is returned.

Examples

This function returns 1.25:

```
CEILING(1.23459, .05)
```

This function returns 150:

```
CEILING(148.24, 2)
```

See Also	EVEN FLOOR INT ODD ROUND TRUNC
----------	---

CHAR

Description	Returns a character that corresponds to the supplied ASCII code.
-------------	--

Syntax	CHAR (<i>number</i>)
--------	-------------------------------

Parameter	Description
<i>number</i>	A value between 1 and 255 that specifies an ASCII character.

Remarks	The character and associated numeric code are defined by Windows in the ASCII character set.
---------	--

Examples	This function returns F:
----------	--------------------------

CHAR(70)

This function returns #:

CHAR(35)

See Also	CODE
----------	-------------

CHOOSE

Description Returns a value from a list of numbers based on the index number supplied.

Syntax **CHOOSE** (*index*, *item_list*)

Parameter	Description
<i>index</i>	A number that refers to an item in <i>item_list</i> .
<i>item_list</i>	A list of numbers, formulas, or text separated by commas. This argument can also be a range reference. You can specify as many as 29 items in the list.

Remarks *Index* can be a cell reference; *index* can also be a formula that returns any value from 1 to 29. If *index* is less than 1 or greater than the number of items in *item_list*, #VALUE! is returned. If *index* is a fractional number, it is truncated to an integer.

Examples This function returns Q2:

CHOOSE(2,"Q1", "Q2", "Q3", "Q4")

This function returns the average of the contents of range A1:A10:

AVERAGE(CHOOSE(1, A1:A10, B1:B10, C1:C10))

See Also **INDEX**

CLEAN

Description Removes all nonprintable characters from the supplied text.

Syntax **CLEAN** (*text*)

Parameter	Description
<i>text</i>	Any worksheet information.

Remarks Text that is imported from another environment may require this function.

Examples This function returns Payments Due because the character returned by CHAR (8) is nonprintable:

CLEAN("Payments " & CHAR(8) & "Due")

See Also **CHAR**
TRIM

CODE

Description Returns a numeric code representing the first character of the supplied string.

Syntax **CODE** (*text*)

Parameter	Description
<i>text</i>	Any string.

Remarks The numeric code and associated string are defined in your computer's character set.

Examples This function returns 65:

CODE("A")

This function returns 98:

CODE("b")

See Also **CHAR**

COLUMN

Description Returns the column number of the supplied reference.

Syntax **COLUMN** (*reference*)

Parameter	Description
<i>reference</i>	A reference to a cell or range. Omitting the argument returns the number of the column in which COLUMN is placed.

Examples This function returns 2:

COLUMN(B3)

This function returns 4 if the function is entered in cell D2:

COLUMN()

See Also **COLUMNS**
ROW

COLUMNS

Description Returns the number of columns in a range reference.

Syntax **COLUMNS** (*range*)

Parameter	Description
<i>range</i>	A reference to a range of cells.

Example This function returns 4:

COLUMNS(A1:D5)

See Also **COLUMN**
ROWS

CONCATENATE

Description Joins several text items into one item.

Syntax **CONCATENATE** (*text1*, *text2*,)

Parameter	Description
<i>text1</i> , <i>text2</i> , ...	Up to 30 text items to be joined into a single text item. The text items can be strings, numbers, or single-cell references.

Remarks The “&” operator can be used instead of **CONCATENATE** to join text items.

Examples The following example returns “Sale Price” it is the same as typing “Sale”& “ ” & “Price”:

```
CONCATENATE ("Sale ", "Price")
```

Suppose in an inventory worksheet, C2 contains “extruder1”, C5 contains “gaskets”, and C8 contains the number 15. The following example returns “Inventory currently holds 15 gaskets for extruder1.”:

```
CONCATENATE ("Inventory currently holds ", C8, " ", C5," for ", C2)
```

See Also **COLUMN**
ROWS

COS

Description Returns the cosine of an angle.

Syntax **COS** (*number*)

Parameter	Description
<i>number</i>	The angle in radians. If the angle is in degrees, convert the angle to radians by multiplying the angle by PI()/180.

Examples This function returns .126:

```
COS(1.444)
```

This function returns .28:

```
COS(5)
```

See Also **ACOS**
ASINH
ATANH
COSH
PI

COSH

Description Returns the hyperbolic cosine of a number.

Syntax **COSH** (*number*)

Parameter	Description
<i>number</i>	Any number.

Examples This function returns 4.14:

COSH(2.10)

This function returns 1.03:

COSH(.24)

See Also **ASINH**
ATANH
COS

COUNT

Description Returns the number of values in the supplied list.

Syntax **COUNT** (*value_list*)

Parameter	Description
<i>value_list</i>	A list of values. The list can contain as many as 30 values.

Remarks **COUNT** only numerates numbers or numerical values such as logical values, dates, or text representations of dates. If you supply a range, only numbers and numerical values in the range are counted. Empty cells, logical values, text, and error values in the range are ignored.

Examples This function returns 2:
`COUNT(5, 6, "Q2")`

This function returns 3:
`COUNT("03/06/94", "06/21/94", "10/19/94")`

See Also **AVERAGE**
COUNTA
SUM

COUNTA

Description Returns the number of nonblank values in the supplied list.

Syntax **COUNTA** (*expression_list*)

Parameter	Description
<i>expression_list</i>	A list of expressions. As many as 30 expressions can be included in the list.

Remarks **COUNTA** returns the number of cells that contain data in a range. Null values (" ") are counted, but references to empty cells are ignored.

Examples This function returns 4:
`COUNTA(32, 45, "Earnings", "")`

This function returns 0 when the specified range contains empty cells:
`COUNTA(C38:C40)`

See Also **AVERAGE**
COUNT
PRODUCT
SUM

COUNTIF

Description Returns the number of cells within a range which meet the given criteria.

Syntax **COUNTIF** (*range*, *criteria*)

Parameter	Description
<i>range</i>	Range of cells you want to count.
<i>criteria</i>	Number, expression, or text that defines which cells are counted.

See Also **AVERAGE**
COUNTA
SUM
SUMIF

DATE

Description Returns the serial number of the supplied date.

Syntax **DATE** (*year*, *month*, *day*)

Parameter	Description
<i>year</i>	A number from 1900 to 2078. If <i>year</i> is between 1920 to 2019, you can specify two digits to represent the year; otherwise specify all four digits.
<i>month</i>	A number representing the month (for example, 12 represents December). If a number greater than 12 is supplied, the number is added to the first month of the specified year.
<i>day</i>	A number representing the day of the month. If the number you specify for <i>day</i> exceeds the number of days in that month, the number is added to the first day of the specified month.

Examples This function returns 34506:

DATE(94, 6, 21)

This function returns 36225:

DATE(99, 3, 6)

See Also **DATEVALUE**
DAY
MONTH
NOW
TIMEVALUE
TODAY

YEAR

DATEVALUE

Description Returns the serial number of a date supplied as a text string.

Syntax **DATEVALUE** (*text*)

Parameter	Description
<i>text</i>	A date in text format between January 1, 1900, and December 31, 2078. If you omit the year, the current year is used.

Examples This function returns 34399:

DATEVALUE("3/6/94")

This function returns 35058:

DATEVALUE("12/25/95")

See Also **NOW**
TIMEVALUE
TODAY

DAY

Description Returns the day of the month that corresponds to the date represented by the supplied number.

Syntax **DAY** (*serial_number*)

Parameter	Description
<i>serial_number</i>	A date represented as a serial number or as text (for example, 06-21-94 or 21-Jun-94).

Examples This function returns 6:

DAY(34399)

This function returns 21:

DAY("06-21-94")

See Also **NOW**
 HOUR
 MINUTE
 MONTH
 SECOND
 TODAY
 WEEKDAY
 YEAR

DAYS360

Description Returns the number of days between two dates based on a 360-day year (twelve 30-day months). Use this function to help compute payments if your accounting system is based on twelve 30-day months.

Syntax **DAYS360** (*start_date*, *end_date*, [*method*])

Parameter	Description
<i>start_date</i> , <i>end_date</i>	The two dates between which you want to know the number of days.
<i>method</i>	A logical value that specifies whether the European or US method should be used in the calculation. If False (or omitted), the US (NASD) method is used. If True, the European method is used. The default is based on the local translation. It should be correct for your location.

Remarks *start_date* and *end_date* can be text strings using numbers to represent the month, day, and year (for example, "1/30/93" or "1-30-93"), or they can be serial numbers representing the dates.

If *start_date* occurs after *end_date*, DAYS360 returns a negative number.

If *method* is set to False and *start_date* is the 31st of a month, it becomes equal to the 30th of the same month. If *end_date* is the 31st of a month and *start_date* is less than the 30th of a month, the ending date becomes equal to the 1st of the next month, otherwise the ending date becomes equal to the 30th of the same month.

If *method* is set to True, *start_dates* or *end_dates* which occur on the 31st of a month become equal to the 30th of the same month.

Note To determine the number of days between two dates in a normal year, you can use normal subtraction--for example, "12/31/93"- "1/1/93" equals 364.

Example DAYS360("1/30/93", "2/1/93") equals 1

DB

Description Returns the real depreciation of an asset for a specific period of time using the fixed-declining balance method.

Syntax **DB** (*cost*, *salvage*, *life*, *period* [, *months*])

Parameter	Description
<i>cost</i>	The initial cost of the asset.
<i>salvage</i>	The salvage value of the asset.
<i>life</i>	The number of periods in the useful life of the asset.
<i>period</i>	The period for which to calculate the depreciation. The time units used to determine <i>period</i> and <i>life</i> must match.
<i>months</i>	The number of months in the first year of the item's life. Omitting this argument assumes there are 12 months in the first year.

Example This function returns 1451.52:

DB(10000, 1000, 7, 3)

See Also **DDB**
SLN
SYD
VDB

DBCS

Description In DBCS (Far-East) systems, this functions returns a copy of text in which the single-byte characters are converted to double-byte characters, if possible. Characters that cannot be converted are left unchanged.

Syntax **ASC** (*text*)

Parameter	Description
<i>text</i>	Text converted from single-byte characters to double-byte characters.

Remarks On non-DBCS systems, the text returns unchanged.

See Also **ASC**

DDB

Description Returns the depreciation of an asset for a specific period of time using the double-declining balance method or a declining balance factor you supply.

Syntax **DDB** (*cost*, *salvage*, *life*, *period* [, *factor*])

Parameter	Description
<i>cost</i>	The initial cost of the asset.
<i>salvage</i>	The salvage value of the asset.
<i>life</i>	The number of periods in the useful life of the asset.
<i>period</i>	The period for which to calculate the depreciation. The time units used to determine <i>period</i> and <i>life</i> must match.
<i>factor</i>	The rate at which the balance declines. Omitting this argument assumes a default factor of 2, the double-declining balance factor.

Remarks The double-declining balance method uses an accelerated rate where the highest depreciation occurs in the first period, decreasing in successive periods.

All arguments for this function must be positive numbers.

Example This function returns 1457.73:

DDB(10000,1000, 7, 3)

See Also **DB**
SLN
SYD
VDB

DOLLAR

Description Returns the specified number as text, using the local currency format and the supplied precision.

Syntax **DOLLAR** (*number* [, *precision*])

Parameter	Description
<i>number</i>	A number, a formula that evaluates to a number, or a reference to a cell that contains a number.
<i>precision</i>	A value representing the number of decimal places to the right of the decimal point. Omitting this argument assumes two decimal places.

Note “Local” currency refers to the currency format for the current system, i.e., the one specified in Regional Settings in Control Panel

Remarks	Dollar will return the specified number format as text using currency format for the current system. If you wish to always convert to the US Dollar format, regardless of the language of your system, then use the USDOLLAR worksheet function.
US Example	<p>When using a US setting in Windows, this function returns \$1023.79:</p> <pre>DOLLAR(1023.789)</pre> <p>This function returns \$500:</p> <pre>DOLLAR(495.301, -2)</pre>
UK Example	<p>When using a British setting in Windows, this function returns £1023.8:</p> <pre>DOLLAR(1023.789)</pre> <p>This function returns £500:</p> <pre>DOLLAR(495.301, -2)</pre>
German Example	<p>When using a German setting in Windows, this function returns 1023,8 DM</p> <pre>DOLLAR(1023.789)</pre> <p>This function returns 500 DM:</p> <pre>DOLLAR(495.301, -2)</pre>
See Also	FIXED TEXT VALUE USDOLLAR

ERROR.TYPE

Description	Returns a number corresponding to an error.
Syntax	ERROR.TYPE (<i>error_ref</i>)

Parameter	Description
<i>error_ref</i>	A cell reference.

Remarks	The following error text or numbers can be returned by this function.																		
	<table> <tr> <th>Number</th><th>Description</th></tr> <tr> <td>1</td><td>#NULL!</td></tr> <tr> <td>2</td><td>#DIV/0!</td></tr> <tr> <td>3</td><td>#VALUE!</td></tr> <tr> <td>4</td><td>#REF!</td></tr> <tr> <td>5</td><td>#NAME?</td></tr> <tr> <td>6</td><td>#NUM!</td></tr> <tr> <td>7</td><td>#N/A</td></tr> <tr> <td>#N/A</td><td>Other</td></tr> </table>	Number	Description	1	#NULL!	2	#DIV/0!	3	#VALUE!	4	#REF!	5	#NAME?	6	#NUM!	7	#N/A	#N/A	Other
Number	Description																		
1	#NULL!																		
2	#DIV/0!																		
3	#VALUE!																		
4	#REF!																		
5	#NAME?																		
6	#NUM!																		
7	#N/A																		
#N/A	Other																		
Example	<p>This function returns 2 if the formula in cell A1 attempts to divide by zero:</p> <p><code>ERROR.TYPE(A1)</code></p>																		
See Also	<p>ISERR</p> <p>ISERROR</p>																		

EVEN

Description	Rounds the specified number up to the nearest even integer.				
Syntax	<p>EVEN (<i>number</i>)</p> <table> <tr> <th>Parameter</th><th>Description</th></tr> <tr> <td><i>number</i></td><td>Any number, a formula that evaluates to a number, or a reference to a cell that contains a number.</td></tr> </table>	Parameter	Description	<i>number</i>	Any number, a formula that evaluates to a number, or a reference to a cell that contains a number.
Parameter	Description				
<i>number</i>	Any number, a formula that evaluates to a number, or a reference to a cell that contains a number.				
Examples	<p>This function returns 4:</p> <p><code>EVEN(2.5)</code></p> <p>This function returns 2032:</p> <p><code>EVEN(2030.45)</code></p>				
See Also	<p>CEILING</p> <p>FLOOR</p> <p>INT</p> <p>ODD</p> <p>ROUND</p> <p>TRUNC</p>				

EXACT

Description Compares two expressions for identical, case-sensitive matches. True is returned if the expressions are identical; False is returned if they are not.

Syntax **EXACT** (*expression1*, *expression2*)

Parameter	Description
<i>expression1</i>	Any text.
<i>expression2</i>	Any text.

Examples This function returns True:

EXACT("Match", "Match")

This function returns False:

EXACT("Match", "match")

See Also **LEN**
SEARCH

EXP

Description Returns e raised to the specified power. The constant e is 2.71828182845904 (the base of the natural logarithm).

Syntax **EXP** (*number*)

Parameter	Description
<i>number</i>	Any number as the exponent.

Examples This function returns 12.18:

EXP(2.5)

This function returns 20.09:

EXP(3)

See Also **LN**
LOG

FACT

Description Returns the factorial of a specified number.

Syntax **FACT** (*number*)

Parameter	Description
<i>number</i>	Any non-negative integer. If you supply a real number, FACT truncates the number to an integer before calculation.

Examples This function returns 2:

FACT(2.5)

This function returns 720:

FACT(6)

See Also **PRODUCT**

FALSE

Description Returns the logical value False. This function always requires the trailing parentheses.

Syntax **FALSE** ()

See Also **TRUE**

FIND

Description Searches for a string of text within another text string and returns the character position at which the search string first occurs.

Syntax **FIND** (*search_text*, *text* [, *start_position*])

Parameter	Description
<i>search_text</i>	The text to find. If you specify an empty string (""), FIND matches the first character in text.
<i>text</i>	The text to be searched.
<i>start_position</i>	The character position in <i>text</i> where the search begins. The first character in <i>text</i> is character number 1. When you omit this argument, the default starting position is character number 1.

Remarks	FIND is case-sensitive. You cannot use wildcard characters in the <i>search_text</i> .
Examples	<p>This function returns 12:</p> <pre>FIND("time", "There's no time like the present")</pre> <p>This function returns 19:</p> <pre>FIND("4", "Aisle 4, Part 123-4-11", 9)</pre>
See Also	EXACT LEN MID SEARCH

FINDB

Description	Searches for a string of text within another text string and returns the byte position at which the search string first occurs.								
Syntax	FINDB (<i>search_text</i> , <i>text</i> [, <i>start_position</i>])								
	<table> <tr> <th>Parameter</th><th>Description</th></tr> <tr> <td><i>search_text</i></td><td>The text to find. If you specify an empty string (""), FINDB matches the first byte in <i>text</i>.</td></tr> <tr> <td><i>text</i></td><td>The text to be searched.</td></tr> <tr> <td><i>start_position</i></td><td>The byte position in <i>text</i> where the search begins. The first byte in <i>text</i> is byte number 1. When you omit this argument, the default starting position is byte number 1.</td></tr> </table>	Parameter	Description	<i>search_text</i>	The text to find. If you specify an empty string (""), FINDB matches the first byte in <i>text</i> .	<i>text</i>	The text to be searched.	<i>start_position</i>	The byte position in <i>text</i> where the search begins. The first byte in <i>text</i> is byte number 1. When you omit this argument, the default starting position is byte number 1.
Parameter	Description								
<i>search_text</i>	The text to find. If you specify an empty string (""), FINDB matches the first byte in <i>text</i> .								
<i>text</i>	The text to be searched.								
<i>start_position</i>	The byte position in <i>text</i> where the search begins. The first byte in <i>text</i> is byte number 1. When you omit this argument, the default starting position is byte number 1.								
Remarks	<p>FINDB is case-sensitive. You cannot use wildcard characters in the <i>search_text</i>.</p> <p><i>start_position</i> and return value are expressed in bytes, so these values might differ on DBCS systems. On non-DBCS systems, these functions are identical, but FINDB should only be used in special applications that require distinctions between single-byte and double-byte characters.</p>								
Examples	<p>This function returns 12:</p> <pre>FINDB("time", "Theres no time like the present")</pre> <p>This function returns 19:</p> <pre>FINDB("4", "Aisle 4, Part 123-4-11", 9)</pre>								

FIXED

Description Rounds a number to the supplied precision, formats the number in decimal format, and returns the result as text.

Syntax **FIXED** (*number* [, *precision*][, *no_commas*])

Parameter	Description
<i>number</i>	Any number.
<i>precision</i>	The number of digits that appear to the right of the decimal place. When this argument is omitted, a default precision of 2 is used. If you specify negative precision, <i>number</i> is rounded to the left of the decimal point. You can specify a precision as great as 127 digits.
<i>no_commas</i>	Determines if thousands separators (commas) are used in the result. Use 1 to exclude commas in the result. If <i>no_commas</i> is 0 or the argument is omitted, thousands separators are included (for example, 1,000.00).

Examples This function returns 2,000.500:

`FIXED(2000.5, 3)`

This function returns 2010:

`FIXED(2009.5, -1, 1)`

See Also **DOLLAR**
ROUND
TEXT
VALUE

FLOOR

Description Rounds a number down to the nearest multiple of a specified significance.

Syntax **FLOOR** (*number*, *significance*)

Parameter	Description
<i>number</i>	The value to round.
<i>significance</i>	The multiple to which to round.

Remarks Regardless of the sign of the *number*, the value is rounded down, toward zero. If *number* is an exact multiple of *significance*, no rounding occurs.

If *number* or *significance* is non-numeric, #NAME? is returned. When the arguments have opposite signs, #NUM! is returned.

Examples

This function returns 1.2:

```
FLOOR(1.23459, .05)
```

This function returns -148:

```
FLOOR(-148.24, -2)
```

See Also

CEILING
EVEN
INT
ODD
ROUND
TRUNC

FV

Description

Returns the future value of an annuity based on regular payments and a fixed interest rate.

Syntax

FV (*interest*, *nper*, *payment* [, *pv*] [, *type*])

Parameter	Description
<i>interest</i>	The fixed interest rate.
<i>nper</i>	The number of payments in an annuity.
<i>payment</i>	The fixed payment made each period.
<i>pv</i>	The present value, or the lump sum amount, the annuity is currently worth. When you omit this argument, a present value of 0 is assumed.
<i>type</i>	Indicates when payments are due. Use 0 if payments are due at the end of the period or 1 if payments are due at the beginning of the period. When you omit this argument, 0 is assumed.

Remarks

The units used for *interest* must match those used for *nper*. For example, if the annuity has an 8 percent annual interest rate over a period of 5 years, specify 8 percent/12 for *interest* and 5*12 for *nper*.

Cash paid out, such as a payment, is shown as a negative number. Cash received, such as a dividend check, is shown as a positive number.

Examples

This function returns 4,774.55:

```
FV(5%, 8, -500)
```

This function returns 531,550.86:

```
FV(10%/12, 240, -700, 1)
```

See Also

IPMT
NPER
PMT
PPMT
PV
RATE

HLOOKUP

Description Searches the top row of a table for a value and returns the contents of a cell in that table that corresponds to the location of the search value.

Syntax **HLOOKUP** (*search_item*, *search_range*, *row_index*)

Parameter	Description
<i>search_item</i>	A value, text string, or reference to a cell containing a value that is matched against data in the top row of <i>search_range</i> .
<i>search_range</i>	A reference to the range (table) to be searched. The cells in the first row of <i>search_range</i> can contain numbers, text, or logical values. The contents of the first row must be in ascending order (for example, -2, -1, 0, 2...A through Z, False, True). Text searches are not case-sensitive.
<i>row_index</i>	The row in <i>search_range</i> from which the matching value is returned. <i>row_index</i> can be a number from 1 to the number of rows in <i>search_range</i> . If <i>row_index</i> is less than 1, the error #VALUE! is returned. When <i>row_index</i> is greater than the number of rows in the table, the error #REF! is returned.

Remarks **HLOOKUP** compares the information in the top row of *search_range* to the supplied *search_item*. When a match is found, information located in the same column and supplied row (*row_index*) is returned.

If *search_item* cannot be found in the top row of *search_range*, the largest value that is less than *search_item* is used. When *search_item* is less than the smallest value in the first row of the *search_range*, the error #REF! is returned.

Examples The following examples use this worksheet.

	A	B	C	D	E	
1		Midwest	Northeast	Pacific	South	
2	Q1	48.23	278.21	61.97	164.80	
3	Q2	163.83	22.63	161.73	183.96	
4	Q3	43.96	233.56	278.16	171.98	
5	Q4	245.69	167.09	245.23	163.00	
6						

Sheet1

This function returns 22.63:

HLLOOKUP("Northeast", B1:E5, 3)

This function returns #REF!:

HLLOOKUP("Pacific", B1:E5, 7)

See Also

INDEX
LOOKUP
MATCH
VLOOKUP

HOUR

Description Returns the hour component of the specified time in 24-hour format.

Syntax **HOUR** (*serial_number*)

Parameter	Description
<i>serial_number</i>	The time as a serial number. The decimal portion of the number represents time as a fraction of the day.

Remarks The result is an integer ranging from 0 (12:00 AM) to 23 (11:00 PM).

Examples This function returns 9:

HOUR(34259.4)

This function returns 23:

HOUR(34619.976)

See Also

DAY
MINUTE
MONTH
NOW
SECOND
WEEKDAY
YEAR

IF

Description Tests the condition and returns the specified value.

Syntax **IF** (*condition*, *true_value*, *false_value*)

Parameter	Description
<i>condition</i>	Any logical expression.
<i>true_value</i>	The value to be returned if <i>condition</i> evaluates to True.
<i>false_value</i>	The value to be returned if <i>condition</i> evaluates to False.

Example This function returns Greater if the contents of A1 is greater than 10 and Less if the contents of A1 is less than 10:

IF(A1>10, "Greater", "Less")

See Also **AND**
FALSE
NOT
OR
TRUE

INDEX

Description Returns the contents of a cell from a specified range.

Syntax **INDEX** (*reference* [, *row*] [, *column*] [, *range_number*])

Parameter	Description
<i>reference</i>	A reference to one or more ranges. If <i>reference</i> specifies more than one range, separate each reference with a comma and enclose <i>reference</i> in parentheses. For example, (A1:C6, B7:E14, F4). If each range in <i>reference</i> contains only one row or column, you can omit the <i>row</i> or <i>column</i> argument. For example, if <i>reference</i> is A1:A15, you can omit the column argument INDEX(A1:A15, 3,, 1).
<i>row</i>	The row number in <i>reference</i> from which to return data.
<i>column</i>	Column number in <i>reference</i> from which to return data.
<i>range_number</i>	Specifies the range from which data is returned if <i>reference</i> contains more than one range. For example, if <i>reference</i> is (A1:A10, B1:B5, D14:E23), A1:A10 is <i>range_number</i> 1, B1:B5 is <i>range_number</i> 2, and D14:E23 is <i>range_number</i> 3.

Remarks If *row*, *column*, and *range_number* do not point to a cell within reference, #REF! is returned. If *row* and *column* are omitted, **INDEX** returns the range in *reference* specified by *range_number*.

Examples The following examples use this worksheet.

	A	B	C	D	E	
1	Sales Group 1			Sales Group 2		
2	Adams	\$1,225.14		Cash	\$1,819.47	
3	Baker	\$1,415.35		Johnson	\$1,733.67	
4	Martinez	\$1,573.57		Nelson	\$1,138.23	
5	Smith	\$1,469.78		Randall	\$1,634.58	
6	White	\$1,390.89		Schultz	\$1,093.82	
7						

This function returns \$1415.35:

INDEX(A2:B6, 2, 2)

This function returns \$1634.58:

INDEX((A2:B6, D2:E6), 4, 2, 2)

See Also

CHOOSE
HLOOKUP
LOOKUP
MATCH
VLOOKUP

INDIRECT

Description Returns the contents of the cell referenced by the specified cell.

Syntax **INDIRECT** (*ref_text* [, *a1*])

Parameter	Description
<i>ref_text</i>	A reference to a cell that references a third cell. If <i>ref_text</i> is not a valid reference, the error #REF! is returned.
<i>a1</i>	The reference format. This argument must be TRUE() to represent an A1 reference format; Formula One does not support the R1C1 reference format.

Example This function returns the contents of the cell that C1 references. If C1 contains “D1,” then the contents of D1 is returned:

INDIRECT(C1)

See Also **OFFSET**

INT

Description Rounds the supplied number down to the nearest integer.

Syntax **INT** (*number*)

Parameter	Description
<i>number</i>	Any real number.

Examples This function returns 10:

INT(10.99)

This function returns -11:

INT(-10.99)

See Also **CEILING**
FLOOR
MOD
ROUND
TRUNC

IPMT

Description Returns the interest payment of an annuity for a given period, based on regular payments and a fixed periodic interest rate.

Syntax **IPMT** (*interest*, *per*, *nper*, *pv*, [*fv*], [*type*])

Parameter	Description
<i>interest</i>	The fixed periodic interest rate.
<i>per</i>	The period for which to return the interest payment. This number must be between 1 and <i>nper</i> .
<i>nper</i>	The number of payments.
<i>pv</i>	The present value, or the lump sum amount the annuity is currently worth.
<i>fv</i>	The future value, or the value after all payments are made. If this argument is omitted, the future value is assumed to be 0.
<i>type</i>	Indicates when payments are due. Use 0 if payments are due at the end of the period or 1 if payments are due at the beginning of the period. When you omit this argument, 0 is assumed.

Remarks	<p>The units used for <i>interest</i> must match those used for <i>nper</i>. For example, if the annuity has an 8 percent annual interest rate over a period of 5 years, specify 8 percent/12 for <i>interest</i> and 5*12 for <i>nper</i>.</p> <p>Cash paid out, such as a payment, is shown as a negative number. Cash received, such as a dividend check, is shown as a positive number.</p>
Examples	<p>This function returns -117.87:</p> <pre>IPMT(8%/12, 2, 48, 18000)</pre> <p>This function returns -117.09:</p> <pre>IPMT(8%/12, 2, 48, 18000, 0, 1)</pre>
See Also	<p>FV PMT PPMT RATE</p>

IRR

Description	Returns internal rate of return for a series of periodic cash flows.						
Syntax	IRR (<i>cash_flow</i> [, <i>guess</i>])						
	<table> <tr> <th>Parameter</th><th>Description</th></tr> <tr> <td><i>cash_flow</i></td><td>A reference to a range that contains values for which to calculate the internal rate of return. The values must contain at least one positive and one negative value. During calculation, IRR uses the order in which the values appear to determine the order of the cash flow. Text, logical values, and empty cells in the range are ignored.</td></tr> <tr> <td><i>guess</i></td><td>The estimate of the internal rate of return. If no argument is supplied, a rate of return of 10 percent is assumed.</td></tr> </table>	Parameter	Description	<i>cash_flow</i>	A reference to a range that contains values for which to calculate the internal rate of return. The values must contain at least one positive and one negative value. During calculation, IRR uses the order in which the values appear to determine the order of the cash flow. Text, logical values, and empty cells in the range are ignored.	<i>guess</i>	The estimate of the internal rate of return. If no argument is supplied, a rate of return of 10 percent is assumed.
Parameter	Description						
<i>cash_flow</i>	A reference to a range that contains values for which to calculate the internal rate of return. The values must contain at least one positive and one negative value. During calculation, IRR uses the order in which the values appear to determine the order of the cash flow. Text, logical values, and empty cells in the range are ignored.						
<i>guess</i>	The estimate of the internal rate of return. If no argument is supplied, a rate of return of 10 percent is assumed.						
Remarks	<p>The internal rate of return is the interest rate received for an investment consisting of payments (specified by negative numbers) and investments (specified by positive numbers).</p> <p>IRR is calculated iteratively, cycling through the calculation until the result is accurate to .00001 percent. If the result cannot be found after 20 iterations, #NUM! is returned. When this occurs, supply a different value for <i>guess</i>.</p>						

Examples

The following examples use this worksheet.

	A	B
1	Investment	(\$60,000.00)
2	1989 income	\$9,590.00
3	1990 income	\$10,580.00
4	1991 income	\$12,790.00
5	1992 income	\$15,830.00
6	1993 income	\$18,930.00
7		
Sheet1		

This function returns 3.72 percent:

IRR(B1:B6)

This function returns -49.26 percent:

IRR(B1:B3, -20%)

See Also

MIRR
NPV
RATE

ISBLANK

Description

Determines if the specified cell is blank.

Syntax

ISBLANK (*reference*)

Parameter	Description
<i>reference</i>	A reference to any cell.

Remarks

If the referenced cell is blank, True is returned. False is returned if the cell is not blank.

Example

This function returns True if A1 is a blank cell:

ISBLANK(A1)

See Also

ISERR
ISERROR
ISLOGICAL
ISNA
ISNONTEXT
ISNUMBER
ISREF
ISTEXT

ISERR

Description Determines if the specified expression returns an error value.

Syntax **ISERR** (*expression*)

Parameter	Description
<i>expression</i>	Any expression.

Remarks If the expression returns any error except #N/A!, True is returned. Otherwise, False is returned.

Example This function returns True if A1 contains a formula that returns an error such as #NUM!:

ISERR(A1)

See Also

ISBLANK
ISERROR
ISLOGICAL
ISNA
ISNONTEXT
ISNUMBER
ISREF
ISTEXT

ISERROR

Description Determines if the specified expression returns an error value.

Syntax **ISERROR** (*expression*)

Parameter	Description
<i>expression</i>	Any expression.

Remarks If *expression* returns any error value, such as #N/A!, #VALUE!, #REF!, #DIV/0!, #NUM!, #NAME?, or #NULL!, True is returned. Otherwise, False is returned.

Examples This function returns True:

ISERROR(4/0)

This function returns False if A1 contains a formula that does not return an error.

ISERROR(A1)

See Also **ISBLANK**

ISERR
ISLOGICAL
ISNA
ISNONTEXT
ISNUMBER
ISREF
ISTEXT

ISLOGICAL

Description Determines if the specified expression returns a logical value.

Syntax **ISLOGICAL** (*expression*)

Parameter	Description
<i>expression</i>	Any expression.

Remarks If *expression* returns a logical value, True is returned. Otherwise, False is returned.

Example This function returns True because **ISBLANK** returns a logical value:

ISLOGICAL(ISBLANK(A1))

See Also **ISBLANK**
ISERR
ISERROR
ISNA
ISNONTEXT
ISNUMBER
ISREF
ISTEXT

ISNA

Description Determines if the specified expression returns the value not available error.

Syntax **ISNA** (*expression*)

Parameter	Description
<i>expression</i>	Any expression.

Remarks If *expression* returns the #N/A! error, True is returned. Otherwise, False is returned.

Example This function returns True if cell A1 contains the NA () function or returns the error value #N/A!:

ISNA(A1)

See Also

ISBLANK
ISERR
ISERROR
ISLOGICAL
ISNONTEXT
ISNUMBER
ISREF
ISTEXT

ISNONTEXT

Description

Determines if the specified expression is not text.

Syntax

ISNONTEXT (*expression*)

Parameter	Description
<i>expression</i>	Any expression.

Remarks

If *expression* returns any value that is not text, True is returned. Otherwise, False is returned.

Examples

This function returns True if cell F3 contains a number or is a blank cell:

ISNONTEXT(F3)

This function returns False:

ISNONTEXT("text")

See Also

ISBLANK
ISERR
ISERROR
ISLOGICAL
ISNA
ISNUMBER
ISREF
ISTEXT

ISNUMBER

Description Determines if the specified expression is a number.

Syntax **ISNUMBER** (*expression*)

Parameter	Description
<i>expression</i>	Any expression.

Remarks If *expression* returns a number, True is returned. Otherwise, False is returned. If *expression* returns a number represented as text (for example, "12"), False is returned.

Examples This function returns True:

ISNUMBER(123.45)

This function returns False:

ISNUMBER("123")

See Also **ISBLANK**
ISERR
ISERROR
ISLOGICAL
ISNA
ISNONTEXT
ISREF
ISTEXT

ISREF

Description Determines if the specified expression is a range reference.

Syntax **ISREF** (*expression*)

Parameter	Description
<i>expression</i>	Any expression.

Remarks If *expression* returns a range reference, True is returned. Otherwise, False is returned.

Example This function returns True:

ISREF(A3)

See Also **ISBLANK**
ISERR

ISERROR
ISLOGICAL
ISNA
ISNONTEXT
ISTEXT

ISTEXT

Description Determines if the specified expression is text.

Syntax **ISTEXT** (*expression*)

Parameter	Description
<i>expression</i>	Any expression.

Remarks If *expression* returns text, True is returned. Otherwise, False is returned.

Example This function returns True:

ISTEXT("2nd Quarter")

See Also **ISBLANK**
ISERR
ISERROR
ISLOGICAL
ISNA
ISNONTEXT
ISNUMBER
ISREF

LEFT

Description Returns the leftmost characters from the specified text string.

Syntax **LEFT** (*text* [, *num_chars*])

Parameter	Description
<i>text</i>	Any text string.
<i>num_chars</i>	The number of characters to return. This value must be greater than or equal to zero. If <i>num_chars</i> is greater than the number of characters in <i>text</i> , the entire string is returned. Omitting this argument assumes a value of 1.

Examples This function returns 2:

LEFT("2nd Quarter")

This function returns 2nd:

```
LEFT("2nd Quarter", 3)
```

See Also

MID
RIGHT

LEFTB

Description

Returns the leftmost byte from the specified text string.

Syntax

LEFTB (*text* [, *num_bytes*])

Parameter	Description
<i>text</i>	Any text string.
<i>num_bytes</i>	The number of bytes to return. This value must be greater than or equal to zero. If <i>num_bytes</i> is greater than the number of bytes in <i>text</i> , the entire string is returned. Omitting this argument assumes a value of 1.

Remarks

num_bytes and return value are expressed in bytes, so these values might differ on DBCS systems. On non-DBCS systems, these functions are identical, but **LEFTB** should only be used in special applications that require distinctions between single-byte and double-byte characters.

Examples

This function returns 2:

```
LEFTB("2nd Quarter")
```

This function returns 2nd:

```
LEFTB("2nd Quarter", 3)
```

LEN

Description

Returns the number of characters in the supplied text string.

Syntax

LEN (*text*)

Parameter	Description
<i>text</i>	Any text string. Spaces in the string are counted as characters.

Examples

This function returns 11:

```
LEN("3rd Quarter")
```

This function returns 3:

```
LEN("1-3")
```

See Also

EXACT
SEARCH

LENB

Description

Returns the number of bytes in the supplied text string.

Syntax

LENB (*text*)

Parameter	Description
<i>text</i>	Any text string. Spaces in the string are counted as bytes.

Remarks

text and return value are expressed in bytes, so these values might differ on DBCS systems. On non-DBCS systems, these functions are identical, but **LENB** should only be used in special applications that require distinctions between single-byte and double-byte characters.

Examples

This function returns 11:

```
LENB("3rd Quarter")
```

This function returns 3:

```
LENB("1-3")
```

LN

Description

Returns the natural logarithm (based on the constant e) of a number.

Syntax

LN (*number*)

Parameter	Description
<i>number</i>	Any positive real number.

Remarks

LN is the inverse of the **EXP** function.

Examples

This function returns 2.50:

```
LN(12.18)
```

This function returns 3.00:

```
LN(20.09)
```

See Also **EXP**
 LOG
 LOG10

LOG

Description Returns the logarithm of a number to the specified base.

Syntax **LOG** (*number* [, *base*])

Parameter	Description
<i>number</i>	Any positive real number.
<i>base</i>	The base of the logarithm. Omitting this argument assumes a base of 10.

Examples This function returns 0:

LOG(1)

This function returns 1:

LOG(10)

See Also **EXP**
 LN
 LOG10

LOG10

Description Returns the base-10 logarithm of a number.

Syntax **LOG10** (*number*)

Parameter	Description
<i>number</i>	Any positive real number.

Examples This function returns 2.41:

LOG10(260)

This function returns 2:

LOG10(100)

See Also **EXP**
 LN

LOG

LOOKUP

Description Searches for a value in one range and returns the contents of the corresponding position in a second range.

Syntax **LOOKUP** (*lookup_value*, *lookup_range*, *result_range*)

Parameter	Description
<i>lookup_value</i>	The value for which to search in the first range.
<i>lookup_range</i>	The first range to search and contains only one row or one column. The range can contain numbers, text, or logical values. To search <i>lookup_range</i> correctly, the expressions in the range must be placed in ascending order (for example, -2, -1, 0, 1, 2...A through Z, False, True). The search is not case-sensitive.
<i>result_range</i>	A range of one row or one column that is the same size as <i>lookup_range</i> .

Remarks If *lookup_value* does not have an exact match in *lookup_range*, the largest value that is less than or equal to *lookup_value* is found and the corresponding position in *result_range* is returned. When *lookup_value* is smaller than the data in *lookup_range*, #N/A is returned.

Examples The following examples use this worksheet.

	A	B	
1	Region	Headquarters	
2	Midwest	Kansas City	
3	North	Detroit	
4	Northeast	Philadelphia	
5	Pacific	Portland	
6	South	Atlanta	
7	Southwest	Phoenix	
8			

This function returns Detroit:

LOOKUP("North", A2:A7, B2:B7)

This function returns #N/A:

LOOKUP("Alabama", A2:A7, B2:B7)

See Also

HLOOKUP
INDEX
VLOOKUP

LOWER

Description Changes the characters in the specified string to lowercase characters. Numeric characters in the string are not changed.

Syntax **LOWER** (*text*)

Parameter	Description
<i>text</i>	Any string.

Examples This function returns 3rd quarter:

LOWER("3rd Quarter")

This function returns john doe:

LOWER("JOHN DOE")

See Also **PROPER**
UPPER

MATCH

Description A specified value is compared against values in a range. The position of the matching value in the search range is returned.

Syntax **MATCH** (*lookup_value*, *lookup_range*, *comparison*)

Parameter	Description
<i>lookup_value</i>	The value against which to compare. It can be a number, text, or logical value or a reference to a cell that contains one of those values.
<i>lookup_range</i>	The range to search and contains only one row or one column. The range can contain numbers, text, or logical values.
<i>comparison</i>	<p>A number that represents the type of comparison to be made between <i>lookup_value</i> and the values in <i>lookup_range</i>. When you omit this argument, comparison method 1 is assumed.</p> <p>When <i>comparison</i> is 1, the largest value that is less than or equal to <i>lookup_value</i> is matched. When using this comparison method, the values in <i>lookup_range</i> must be in ascending order (for example, ...-2, -1, 0, 1, 2..., A through Z, False, True).</p> <p>When <i>comparison</i> is 0, the first value that is equal to <i>lookup_value</i> is matched. When using this comparison method, the values in <i>lookup_range</i> can be in any order.</p> <p>When <i>comparison</i> is -1, the smallest value that is greater than or equal to <i>lookup_value</i> is matched. When using this comparison method, the values in <i>lookup_range</i> must be in descending order (for example, True, False, Z through A, ...2, 1, 0, -1, -2...).</p>

Remarks When using comparison method 0 and *lookup_value* is text, *lookup_value* can contain wildcard characters. The wildcard characters are * (asterisk), which matches any sequence of characters, and ? (question mark), which matches any single character.

When no match is found for *lookup_value*, #N/A is returned.

Examples The following examples use this worksheet.

	A	B
1	Mfr. Code	Stock No.
2	BAJ	0677
3	DOD	0753
4	FMH	0816
5	JMP	0913
6	PLY	7534
7	TJL	7763
8		

Sheet1

This function returns 5:

```
MATCH(7600, B2:B7,1)
```

This function returns 2:

```
MATCH("D*", A2:A7,0)
```

See Also **HLOOKUP**
INDEX
LOOKUP
VLOOKUP

MAX

Description Returns the largest value in the specified list of numbers.

Syntax **MAX** (*number_list*)

Parameter	Description
<i>number_list</i>	<p>A list of as many as 30 numbers, separated by commas.</p> <p>The list can contain numbers, logical values, text representations of numbers, or a reference to a range containing those values.</p> <p>Error values or text that cannot be translated into numbers return errors.</p> <p>If a range reference is included in the list, text, logical expressions, and empty cells in the range are ignored.</p> <p>If there are no numbers in the list, 0 is returned.</p>

Examples This function returns 500:

```
MAX(50, 100, 150, 500, 200)
```

This function returns the largest value in the range:

```
MAX(A1:F12)
```

See Also **AVERAGE**
MIN

MID

Description Returns the specified number of characters from a text string, beginning with the specified starting position.

Syntax **MID** (*text*, *start_position*, *num_chars*)

Parameter	Description
<i>text</i>	The string from which to return characters.
<i>start_position</i>	The position of the first character to return from <i>text</i> . If <i>start_position</i> is 1, the first character in <i>text</i> is returned. If <i>start_position</i> is greater than the number of characters in <i>text</i> , an empty string ("") is returned. If <i>start_position</i> is less than 1, #VALUE! is returned.
<i>num_chars</i>	The number of characters to return. If <i>num_chars</i> is negative, #VALUE! is returned.

Remarks If *start_position* plus the number of characters in *num_chars* exceeds the length of *text*, the characters from *start_position* to the end of *text* are returned.

Examples This function returns Expenses:

MID("Travel Expenses", 8, 8)

This function returns 45:

MID("Part #45-7234", 7, 2)

See Also

CODE
FIND
LEFT
RIGHT
SEARCH

MIDB

Description Returns the specified number of bytes from a text string, beginning with the specified starting position.

Syntax **MIDB** (*text*, *start_position*, *num_bytes*)

Parameter	Description
<i>text</i>	The string from which to return bytes.
<i>start_position</i>	The position of the first byte to return from <i>text</i> . If <i>start_position</i> is 1, the first byte in <i>text</i> is returned. If <i>start_position</i> is greater than the number of bytes in <i>text</i> , an empty string ("") is returned. If <i>start_position</i> is less than 1, #VALUE! is returned.
<i>num_bytes</i>	The number of bytes to return. If <i>num_bytes</i> is negative, #VALUE! is returned.

Remarks If *start_position* plus the number of bytes in *num_bytes* exceeds the length of text, the bytes from *start_position* to the end of text are returned.

start_position, *num_bytes*, and return value are expressed in bytes, so these values might differ on DBCS systems. On non-DBCS systems, these functions are identical, but **MIDB** should only be used in special applications that require distinctions between single-byte and double-byte characters.

Examples This function returns Expenses:

MIDB("Travel Expenses", 8, 8)

This function returns 45:

MIDB("Part #45-7234", 7, 2)

MIN

Description Returns the smallest value in the specified list of numbers.

Syntax **MIN** (*number_list*)

Parameter	Description
<i>number_list</i>	<p>A list of as many as 30 numbers, separated by commas. The list can contain numbers, logical values, text representations of numbers, or a reference to a range containing those values.</p> <p>Error values or text that cannot be translated into numbers return errors.</p> <p>If a range reference is included in the list, text, logical expressions, and empty cells in the range are ignored.</p> <p>If there are no numbers in the list, 0 is returned.</p>

Examples This function returns 50:

MIN(50, 100, 150, 500, 200)

This function returns the smallest value in the range:

MIN(A1:F12)

See Also **AVERAGE**
MAX

MINUTE

Description Returns the minute that corresponds to the supplied date.

Syntax **MINUTE** (*serial_number*)

Parameter	Description
<i>serial_number</i>	The time as a serial number. The decimal portion of the number represents time as a fraction of the day.

Remarks The result is an integer ranging from 0 to 59.

Examples This function returns 36:

MINUTE(34506.4)

This function returns 48:

MINUTE(34399.825)

See Also **DAY**
 HOUR
 MONTH
 NOW
 SECOND
 WEEKDAY
 YEAR

MIRR

Description Returns the modified internal rate of return for a series of periodic cash flows.

Syntax **MIRR** (*cash_flows*, *finance_rate*, *reinvest_rate*)

Parameter	Description
<i>cash_flow</i>	<p>A reference to a range that contains values for which to calculate the modified internal rate of return. The values must contain at least one positive and one negative value.</p> <p>During calculation, MIRR uses the order in which the values appear to determine the order of cash flow.</p> <p>Values that represent cash received should be positive; negative values represent cash paid.</p> <p>Text, logical values, and empty cells in the range are ignored.</p>
<i>finance_rate</i>	The interest rate paid on money used in the cash flow.
<i>reinvest_rate</i>	The interest rate received on money reinvested from the cash flow.

Remarks The modified internal rate of return considers the cost of the investment and the interest received on the reinvestment of cash.

Examples The following examples use this worksheet.

	A	B
1	Investment	(\$60,000.00)
2	1989 income	\$9,590.00
3	1990 income	\$10,580.00
4	1991 income	\$12,790.00
5	1992 income	\$15,830.00
6	1993 income	\$18,930.00

This function returns 5.20 percent:

MIRR(B1:B6, 12%, 8%)

This function returns -40.93 percent:

MIRR(B1:B3, 12%, 8%)

See Also

IRR
NPV
RATE

MOD

Description

Returns the remainder after dividing a number by a specified divisor.

Syntax

MOD (*number*, *divisor*)

Parameter	Description
<i>number</i>	Any number.
<i>divisor</i>	Any nonzero number. If <i>divisor</i> is 0, #DIV/0! is returned.

Examples

This function returns 1:

MOD(-23, 3)

This function returns -2:

MOD(-23, -3)

See Also

INT
ROUND
TRUNC

MONTH

Description

Returns the month that corresponds to the supplied date.

Syntax

MONTH (*serial_number*)

Parameter	Description
<i>serial_number</i>	The date as a serial number or as text (for example, 06-21-94 or 21-Jun-94).

Remarks

MONTH returns a number ranging from 1 (January) to 12 (December).

Examples

This function returns 6:

MONTH("06-21-94")

This function returns 10:

MONTH(34626)

See Also **DAY**
 HOUR
 MINUTE
 NOW
 SECOND
 TODAY
 WEEKDAY
 YEAR

N

Description Tests the supplied value and returns the value if it is a number.

Syntax **N** (*value*)

Parameter	Description
<i>value</i>	A value or a reference to a cell containing a value to test.

Remarks Numbers are returned as numbers, serial numbers formatted as dates are returned as serial numbers, and the logical function TRUE() is returned as 1. All other expressions return 0.

Examples This function returns 32467:

N(32467)

This function returns 1 if A4 contains the logical function TRUE:

N(A4)

See Also **T**
 VALUE

NA

Description Returns the error value #N/A, which represents “not available.”

Syntax **NA** ()

Remarks Use **NA** to mark cells that lack data without leaving them empty. Empty cells may not be correctly represented in some calculations.

Although **NA** does not use arguments, you must supply the empty parentheses to correctly reference the function.

See Also **ISNA**

NOT

Description Returns a logical value that is the opposite of its value.

Syntax **NOT** (*logical*)

Parameter	Description
<i>logical</i>	An expression that returns a logical value such as True or False.

Remarks If *logical* is false, **NOT** returns True. Conversely, if *logical* is true, **NOT** returns False.

Examples This function returns False:

NOT(TRUE())

This function returns False:

NOT(MONTH("12/25/94") = 12)

See Also **AND**
IF
OR

NOW

Description Returns the current date and time as a serial number.

Syntax **NOW** ()

Remarks In a serial number, numbers to the left of the decimal point represent the date; numbers to the right of the decimal point represent the time. The result of this function changes only when a recalculation of the worksheet occurs.

See Also **DATE**
DAY
HOUR
MINUTE
MONTH
SECOND
TODAY
WEEKDAY
YEAR

NPER

Description Returns the number of periods of an investment based on regular periodic payments and a fixed interest rate.

Syntax **NPER** (*interest*, *pmt*, *pf* [, *fv*] [, *type*])

Parameter	Description
<i>interest</i>	The fixed interest rate.
<i>pmt</i>	The fixed payment made each period. Generally, <i>pmt</i> includes the principle and interest, not taxes or other fees.
<i>pf</i>	The present value, the lump-sum amount that a series of future payments is currently worth.
<i>fv</i>	The future value, the balance to attain after the final payment. Omitting this argument assumes a future balance of 0.
<i>type</i>	Indicates when payments are due. Use 0 if payments are due at the end of the period or 1 if payments are due at the beginning of the period. When you omit this argument, 0 is assumed.

Examples This function returns 36.67:

NPER(12%/12, -350, -300, 16000, 1)

This function returns 36.98:

NPER(1%, -350, -300, 16000)

See Also **FV**
IPMT
PMT
PPMT
PV
RATE

NPV

Description Returns the net present value of an investment based on a series of periodic payments and a discount rate.

Syntax **NPV** (*discount_rate*, *value_list*)

Parameter	Description
<i>discount_rate</i>	The rate of discount for one period.
<i>value_list</i>	<p>A list of as many as 29 arguments or a reference to a range that contains values that represent payments and income.</p> <p>During calculation, NPV uses the order in which the values appear to determine the order of cash flow.</p> <p>Numbers, empty cells, and text representations of numbers are included in the calculation. Errors and text that cannot be translated into numbers are ignored.</p> <p>If <i>value_list</i> is a range reference, only numeric data in the range is included in the calculation. Other types of data in the range, such as empty cells, logical values, text, and error values, are ignored.</p>

Remarks The time span **NPV** uses for calculation begins one period before the first cash flow date and ends when the last cash flow payment is made. This function is based on future cash flows. When your first cash flow occurs at the beginning of the first period, the first value must be added to the **NPV** result, not supplied as a value in *value_list*.

Example This function returns 811.57:

NPV(8%, -12000, 3000, 3000, 3000, 7000)

See Also **FV**
IRR
PV

ODD

Description Rounds the specified number up to the nearest odd integer.

Syntax **ODD** (*number*)

Parameter	Description
<i>number</i>	Any number, a formula that evaluates to a number, or a reference to a cell that contains a number.

Examples This function returns 5:

`ODD(3.5)`

 This function returns 7:

`ODD(6)`

See Also **CEILING**
EVEN
FLOOR
INT
ROUND
TRUNC

OFFSET

Description Returns the contents of a range that is offset from a starting point in the spreadsheet.

Syntax **OFFSET** (*reference*, *rows*, *columns* [, *height*] [, *width*])

Parameter	Description
<i>reference</i>	A reference to a cell from which the offset reference is based. If you specify a range reference, #VALUE! is returned.
<i>rows</i>	The number of rows from <i>reference</i> that represents the upper-left cell of the offset range. A positive number represents rows below the starting cell; a negative number represents rows above the starting cell. If <i>rows</i> places the upper-left cell of the offset range outside the spreadsheet boundary, #REF! is returned.
<i>columns</i>	The number of columns from <i>reference</i> that represents the upper-left cell of the offset range. A positive number represents columns right of the starting cell; a negative number represents columns left of the starting cell. If <i>columns</i> places the upper-left cell of the offset range outside the spreadsheet boundary, #REF! is returned.
<i>height</i>	A positive number representing the number of rows to include in the offset range. Omitting this argument assumes a single row.
<i>width</i>	A positive number representing the number of columns to include in the offset range. Omitting this argument assumes a single column.

Remarks **OFFSET** does not change the current selection in the worksheet. Because it returns a reference, **OFFSET** can be used in any function that requires or uses a cell or range reference as an argument.

Examples

This function returns the contents of cell D4:

```
OFFSET(B1, 3, 2, 1, 1)
```

This function returns the sum of the values in the range E3:F5:

```
SUM(OFFSET(A1, 2, 4, 3, 2))
```

OR

Description

Returns True if at least one of a series of logical arguments is true.

Syntax

OR (*logical_list*)

Parameter	Description
<i>logical_list</i>	A list of conditions separated by commas. You can include as many as 30 conditions in the list. The list can contain logical values or a reference to a range containing logical values. Text and empty cells are ignored. If there are no logical values in the list, the error value #VALUE! is returned.

Example

This function returns True because one of the arguments is true:

```
OR(1 + 1 = 1, 5 + 5 = 10)
```

See Also

AND
IF
NOT

PI

Description

Returns the value of pi (π), which is approximately 3.14159265358979 when calculated to 15 significant digits.

Syntax

PI ()

Remarks

Although **PI** does not use arguments, you must supply the empty parentheses to correctly reference the function.

See Also

COS
SIN
TAN

PMT

Description Returns the periodic payment of an annuity, based on regular payments and a fixed periodic interest rate.

Syntax **PMT** (*interest*, *nper*, *pv* [, *fv*] [, *type*])

Parameter	Description
<i>interest</i>	The fixed periodic interest rate.
<i>nper</i>	The number of periods in the annuity.
<i>pv</i>	The present value, or the amount the annuity is currently worth.
<i>fv</i>	The future value, or the amount the annuity will be worth. When you omit this argument, a future value of 0 is assumed.
<i>type</i>	Indicates when payments are due. Use 0 if payments are due at the end of the period or 1 if payments are due at the beginning of the period. When you omit this argument, 0 is assumed.

Remarks **PMT** returns only the principal and interest payment, it does not include taxes or other fees.

The units used for *interest* must match those used for *nper*. For example, if the annuity has an 8 percent annual interest rate over a period of 5 years, specify 8 percent/12 for *interest* and 5*12 for *nper*.

Cash paid out, such as a payment, is shown as a negative number. Cash received, such as a dividend check, is shown as a positive number.

Examples This function returns –439.43:

PMT(8%/12, 48, 18000)

This function returns –436.52:

PMT(8%/12, 48, 18000, 0, 1)

See Also **IPMT**
FV
NPER
PPMT
PV
RATE

PPMT

Description Returns the principle paid on an annuity for a given period.

Syntax **PPMT** (*interest*, *per*, *nper*, *pv*, [*fv*], [*type*])

Parameter	Description
<i>interest</i>	The fixed periodic interest rate.
<i>per</i>	The period for which to return the principle.
<i>nper</i>	The number of periods in the annuity.
<i>pv</i>	The present value, or the amount the annuity is currently worth.
<i>fv</i>	The future value, or the amount the annuity will be worth. When you omit this argument, a future value of 0 is assumed.
<i>type</i>	Indicates when payments are due. Use 0 if payments are due at the end of the period or 1 if payments are due at the beginning of the period. When you omit this argument, 0 is assumed.

Remarks The units used for *interest* must match those used for *nper*. For example, if the annuity has an 8 percent annual interest rate over a period of 5 years, specify 8 percent/12 for *interest* and 5*12 for *nper*.

Examples This function returns −321.56:

PPMT(8%/12, 2, 48, 18000)

This function returns −319.43:

PPMT(8%/12, 2, 48, 18000, 0, 1)

See Also

IPMT
FV
NPER
PMT
PPMT
PV
RATE

PRODUCT

Description Multiplies a list of numbers and returns the result.

Syntax **PRODUCT** (*number_list*)

Parameter	Description
<i>number_list</i>	<p>A list of as many as 30 numbers, separated by commas.</p> <p>The list can contain numbers, logical values, text representations of numbers, or a reference to a range containing those values.</p> <p>Error values or text that cannot be translated into numbers return errors.</p> <p>If a range reference is included in the list, text, logical expressions, and empty cells in the range are ignored.</p> <p>All numeric values, including 0, are used in the calculation.</p>

Example This function returns 24:

PRODUCT(1, 2, 3, 4)

See Also **FACT**
SUM

PROPER

Description Returns the specified string in proper-case format.

Syntax **PROPER** (*text*)

Parameter	Description
<i>text</i>	Any string.

Remarks In proper-case format, the first alphabetic character in a word is capitalized. If an alphabetic character follows a number, punctuation mark, or space, it is capitalized. All other alphabetic characters are lowercase. Numbers are not changed by **PROPER**.

Examples This function returns 3Rd Quarter:

`PROPER("3rd Quarter")`

This function returns John Doe:

`PROPER("JOHN DOE")`

See Also **LOWER**
UPPER

PV

Description Returns the present value of an annuity, considering a series of constant payments made over a regular payment period.

Syntax **PV** (*interest*, *nper*, *pmt* [, *fv*] [, *type*])

Parameter	Description
<i>interest</i>	The fixed periodic interest rate.
<i>nper</i>	The number of payment periods in the investment.
<i>pmt</i>	The fixed payment made each period.
<i>fv</i>	The future value, or the amount the annuity will be worth. When you omit this argument, a future value of 0 is assumed.
<i>type</i>	Indicates when payments are due. Use 0 if payments are due at the end of the period or 1 if payments are due at the beginning of the period. When you omit this argument, 0 is assumed.

Remarks The units used for *interest* must match those used for *nper*. For example, if the annuity has an 8 percent annual interest rate over a period of 5 years, specify 8 percent/12 for *interest* and 5*12 for *nper*.

Cash paid out, such as a payment, is shown as a negative number. Cash received, such as a dividend check, is shown as a positive number.

Examples This function returns -17999.89:

`PV(8%/12, 48, 439.43)`

This function returns 17999.89:

`PV(8%/12, 48, -439.43)`

See Also **FV**
 IPMT
 NPER
 PMT
 PPMT
 RATE

RAND

Description Returns a number selected randomly from a uniform distribution greater than or equal to 0 and less than 1.

Syntax **RAND ()**

Remarks Although **RAND** does not use arguments, you must supply the empty parentheses to correctly reference the function.

Example This function returns a random number greater than or equal to 0 and less than 10:

 `RAND () * 10`

RATE

Description Returns the interest rate per period of an annuity, given a series of constant cash payments made over a regular payment period.

Syntax **RATE (*nper*, *pmt*, *pv* [, *fv*] [, *type*] [, *guess*])**

Parameter	Description
<i>nper</i>	The number of periods in the annuity.
<i>pmt</i>	The fixed payment made each period. Generally, <i>pmt</i> includes only principle and interest, not taxes or other fees.
<i>pv</i>	The present value of the annuity.
<i>fv</i>	The future value, or the amount the annuity will be worth. When you omit this argument, a future value of 0 is assumed.
<i>type</i>	Indicates when payments are due. Use 0 if payments are due at the end of the period or 1 if payments are due at the beginning of the period. When you omit this argument, 0 is assumed.
<i>guess</i>	Your estimate of the interest rate. If no argument is supplied, a value of 0.1 (10 percent) is assumed.

Remarks **RATE** is calculated iteratively, cycling through the calculation until the result is accurate to .00001 percent. If the result cannot be found after 20 iterations, #NUM! is returned. When this occurs, supply a different value for *guess*.

Example The following example returns the monthly interest rate of .0067; the annual interest rate (.0067 multiplied by 12) is 8 percent:

```
RATE(48, -439.43, 18000)
```

See Also

FV
IPMT
NPER
PMT
PPMT
PV

REGISTER.ID

Description Returns the register ID of the specified dynamic link library (DLL) that has been previously registered. If the DLL has not been registered, this function registers the DLL, and then returns the register ID.

Syntax **REGISTER.ID**(*module_text*, *procedure*, *type_text*)

Parameter	Description
<i>module_text</i>	The text specifying the name of the DLL that contains the function in Formula One.
<i>procedure</i>	The text specifying the name of the function in the DLL in Formula One. The function name is case dependent in 32-bit Formula One.
<i>type_text</i>	The text specifying the data type of the return value and the data types of all arguments to the DLL. The first letter of <i>type_text</i> specifies the return value. If the function or code resource is already registered, you can omit this argument. For a complete list of the data types available, see the <i>type_text</i> parameter of the CALL worksheet function.

Remarks

For declarations made in C, it is assumed that your compiler defaults to 8-byte doubles, 2-byte short integers, and 4-byte long integers. In the Windows programming environment, all pointers should be far pointers.

Pascal calling conventions are used for all functions called from DLLs. For most C compilers, you must add the **–Pascal** keyword to the function declaration.

If the return value for your custom function uses a pass-by-reference data type, a null pointer can be passed as the return value. The null pointer is interpreted as the #NUM! error value.

For the F and G data types, a custom function can modify an allocated string buffer. If the return value type code is F or G, the value returned by the function is ignored. The list of function arguments is searched for the first data type that corresponds to the return value type. The current contents of the allocated string buffer is taken for the return value. 256 bytes is allocated for the argument; therefore, a function can return a larger string than it receives.

You can use a single digit (n), with a value from 1 to 9, as the code for `data_type`. The variable in the location pointed to by the `nth` argument is modified instead of the return value; this process is referred to as modifying in place. The `nth` argument must be a pass-by-reference data type. In addition, you must declare the function void. For most C compilers, you can add the `Void` keyword to the function declaration.

Examples The following formula registers the `GetTickCount` function with Formula One and returns the register ID:

```
REGISTER.ID("Kernel32", "GetTickCount", "J!")
```

Assuming that `GetTickCount` was already registered on another sheet using the preceding formula, the following formula returns the register ID for `GetTickCount`:

```
REGISTER.ID("Kernel32", "GetTickCount")
```

REPLACE

Description Replaces part of a text string with another text string.

Syntax **REPLACE** (*orig_text*, *start_position*, *num_chars*, *repl_text*)

Parameter	Description
<i>orig_text</i>	The original text string.
<i>start_position</i>	The character position where the replacement begins. If <i>start_position</i> is greater than the number of characters in <i>orig_text</i> , <i>repl_text</i> is appended to the end of <i>orig_text</i> .
<i>num_chars</i>	If <i>start_position</i> is less than 1, #VALUE! is returned. The number of characters to replace. If this argument is negative, #VALUE! is returned.
<i>repl_text</i>	The replacement text string.

Examples This function returns “For the year: 1994”:

```
REPLACE("For the year: 1993", 18, 1, "4")
```

See Also **MID**
SEARCH
TRIM

REPLACEB

Description Replaces part of a text string with another text string.

Syntax **REPLACEB** (*orig_text*, *start_position*, *num_bytes*, *repl_text*)

Parameter	Description
<i>orig_text</i>	The original text string.
<i>start_position</i>	The byte position where the replacement begins. If <i>start_position</i> is greater than the number of bytes in <i>orig_text</i> , <i>repl_text</i> is appended to the end of <i>orig_text</i> . If <i>start_position</i> is less than 1, #VALUE! is returned.
<i>num_bytes</i>	The number of bytes to replace. If this argument is negative, #VALUE! is returned.
<i>repl_text</i>	The replacement text string.

Remarks **REPLACEB** is case-sensitive. You cannot use wildcard characters in the *orig_text*.

start_position, *num_bytes*, and return value are expressed in bytes, so these values might differ on DBCS systems. On non-DBCS systems, these functions are identical, but **REPLACEB** should only be used in special applications that require distinctions between single-byte and double-byte characters.

Examples This function returns “For the year: 1994”:

```
REPLACEB("For the year: 1993", 18, 1, "4")
```

REPT

Description Repeats a text string the specified number of times.

Syntax **REPT** (*text*, *number*)

Parameter	Description
<i>text</i>	Any text string.
<i>number</i>	The number of times you want <i>text</i> to repeat. If <i>number</i> is 0, empty text ("") is returned.

Remarks The result of **REPT** cannot exceed 255 characters.

Example This function returns error-error-error-:

```
REPT("error-", 3)
```

RIGHT

Description Returns the rightmost characters from the given text string.

Syntax **RIGHT** (*text* [, *num_chars*])

Parameter	Description
<i>text</i>	Any text string.
<i>num_chars</i>	The number of characters to return. The value must be greater than or equal to zero. If <i>num_chars</i> is greater than the number of characters in <i>text</i> , the entire string is returned. Omitting this argument assumes a value of 1.

Examples This function returns r:

```
RIGHT("2nd Quarter")
```

This function returns Quarter:

```
RIGHT("2nd Quarter", 7)
```

See Also **LEFT**
MID

RIGHTB

Description Returns the rightmost bytes from the given text string.

Syntax **RIGHTB** (*text* [, *num_chars*])

Parameter	Description
<i>text</i>	Any text string.
<i>num_bytes</i>	The number of bytes to return. The value must be greater than or equal to zero. If <i>num_bytes</i> is greater than the number of bytes in <i>text</i> , the entire string is returned. Omitting this argument assumes a value of 1.

Remarks *num_bytes* and return value are expressed in bytes, so these values might differ on DBCS systems. On non-DBCS systems, these functions are identical, but **RIGHTB** should only be used in special applications that require distinctions between single-byte and double-byte characters.

Examples This function returns r:
`RIGHTB("2nd Quarter")`

This function returns Quarter:
`RIGHTB("2nd Quarter", 7)`

ROUND

Description Rounds the given number to the supplied number of decimal places.

Syntax **ROUND** (*number*, *precision*)

Parameter	Description
<i>number</i>	Any value.
<i>precision</i>	The number of decimal places to which <i>number</i> is rounded. When a negative precision is used, the digits to the right of the decimal point are dropped and the absolute number of significant digits specified by <i>precision</i> are replaced with zeros. If <i>precision</i> is 0, <i>number</i> is rounded to the nearest integer.

Example This function returns 123.46:
`ROUND(123.456, 2)`

This function returns 9900:
`ROUND(9899.435, -2)`

See Also **CEILING**
FLOOR
INT
MOD
ROUND
ROUNDDOWN
ROUNDUP
TRUNC

ROUNDDOWN

Description Rounds a number down.

Syntax **ROUNDDOWN** (*number*, *numberOfDigits*)

Parameter	Description
<i>number</i>	Any real number you want to round.
<i>numberOfDigits</i>	The number of decimal places to which <i>number</i> is rounded. When a negative precision is used, the digits to the right of the decimal point are dropped and the absolute number of significant digits specified by <i>precision</i> are replaced with zeros. If <i>precision</i> is 0, <i>number</i> is rounded down to the nearest integer.

Example This function returns 31.141:

ROUNDDOWN(3.14159, 3)

This function returns 31.400:

ROUNDDOWN(31415.92654, -2)

See Also **CEILING**
FLOOR
INT
MOD
ROUND
ROUNDUP
TRUNC

ROUNDUP

Description Rounds the given number up to the supplied number of decimal places.

Syntax **ROUNDUP** (*number*, *numberOfDigits*)

Parameter	Description
<i>number</i>	Any value you want to round up.
<i>numberOfDigits</i>	The number of decimal places to which <i>number</i> is rounded. When a negative precision is used, the digits to the right of the decimal point are dropped and the absolute number of significant digits specified by <i>precision</i> are replaced with zeros. If <i>precision</i> is 0, <i>number</i> is rounded up to the nearest integer.

Example This function returns 77:

ROUNDUP(76.9,0)

This function returns 3150:

ROUNDUP(31415.92654, -2)

See Also **CEILING**
FLOOR
INT
MOD
ROUND
ROUNDDOWN
TRUNC

ROW

Description Returns the row number of the supplied reference.

Syntax **ROW** (*reference*)

Parameter	Description
<i>reference</i>	A cell or range reference. Omitting this argument returns the row number of the cell in which ROW is entered.

Examples This function returns 3:

ROW(B3)

See Also **COLUMN**
ROWS

SEARCH("?5", "Bin b45")

This function returns 5:

SEARCH("b", "Bin b45", 4)

See Also

FIND
MID
REPLACE
SUBSTITUTE

SEARCHB

Description Locates the position of the first byte of a specified text string within another text string.

Syntax **SEARCHB** (*search_text*, *text* [, *start_position*])

Parameter	Description
<i>search_text</i>	The text to find. To search for an asterisk or question mark, include a tilde (~) before the character. The search string can contain wildcards. The available wildcard characters are * (asterisk), which matches any sequence of characters, and ? (question mark), which matches any single character.
<i>text</i>	The text to be searched.
<i>start_position</i>	The character position where the search begins. If the number you specify is less than 0 or greater than the number of bytes in text, #VALUE! is returned. Omitting this argument assumes a starting position of 1.

Remarks Text is searched from left to right, starting at the position specified. The search is not case-sensitive. If text does not contain the search string, #VALUE! is returned.

start-position and return value are expressed in bytes, so these values might differ on DBCS systems. On non-DBCS systems, these functions are identical, but **SEARCHB** should only be used in special applications that require distinctions between single-byte and double-byte characters.

Examples This function returns 6:

SEARCHB("?5", "Bin b45")

This function returns 5:

SEARCHB("b", "Bin b45", 4)

SECOND

Description Returns the second that corresponds to the supplied date.

Syntax **SECOND** (*serial_number*)

Parameter	Description
<i>serial_number</i>	The time as a serial number. The decimal portion of the number represents time as a fraction of the day.

Examples This function returns 58:

SECOND(.259)

This function returns 46:

SECOND(34657.904)

See Also **DATE**
DAY
HOUR
MINUTE
MONTH
NOW
TODAY
WEEKDAY
YEAR

SIGN

Description Determines the sign of the specified number.

Syntax **SIGN** (*number*)

Parameter	Description
<i>number</i>	Any number.

Remarks **SIGN** returns 1 if the specified number is positive, -1 if it is negative, and 0 if it is 0.

Examples This function returns -1:

SIGN(-123)

This function returns 1:

SIGN(123)

See Also **ABS**

SIN

Description Returns the sine of the supplied angle.

Syntax **SIN** (*number*)

Parameter	Description
<i>number</i>	The angle in radians. If the angle is in degrees, convert the angle to radians by multiplying the angle by PI()/180.

Examples This function returns .85:

SIN(45)

This function returns .89:

SIN(90)

See Also **ASIN**
PI

SINH

Description Returns the hyperbolic sine of the specified number.

Syntax **SINH** (*number*)

Parameter	Description
<i>number</i>	Any number.

Examples This function returns 1.18:

SINH(1)

This function returns 10.02:

SINH(3)

See Also **ASINH**
PI

SLN

Description Returns the depreciation of an asset for a specific period of time using the straight-line balance method.

Syntax **SLN** (*cost*, *salvage*, *life*)

Parameter	Description
<i>cost</i>	The initial cost of the asset.
<i>salvage</i>	The salvage value of the asset.
<i>life</i>	The number of periods of the useful life of the asset.

Example This function returns 1285.71:

SLN(10000, 1000, 7)

See Also **DDB**
SYD
VDB

SQRT

Description Returns the square root of the specified number.

Syntax **SQRT** (*number*)

Parameter	Description
<i>number</i>	Any positive number. If you specify a negative number, the error #NUM! is returned.

Examples This function returns 3:

SQRT(9)

This function returns 1.58:

SQRT(2.5)

See Also **SUMSQ**

STDEV

Description Returns the standard deviation of a population based on a sample of supplied values. The standard deviation of a population represents an average of deviations from the population mean within a list of values.

Syntax **STDEV** (*number_list*)

Parameter	Description
<i>number_list</i>	A list of as many as 30 numbers, separated by commas. The list can contain numbers or a reference to a range that contains numbers.

Example This function returns .56:

STDEV(4.0, 3.0, 3.0, 3.5, 2.5, 4.0, 3.5)

See Also **STDEVP**
VAR
VARP

STDEVP

Description Returns the standard deviation of a population based on an entire population of values. The standard deviation of a population represents an average of deviations from the population mean within a list of values.

Syntax **STDEVP** (*number_list*)

Parameter	Description
<i>number_list</i>	A list of as many as 30 numbers, separated by commas. The list can contain numbers or a reference to a range that contains numbers.

Example This function returns .52:

STDEVP(4.0, 3.0, 3.0, 3.5, 2.5, 4.0, 3.5)

See Also **STDEV**
VAR
VARP

SUBSTITUTE

Description Replaces a specified part of a text string with another text string.

Syntax **SUBSTITUTE** (*text*, *old_text*, *new_text* [, *instance*])

Parameter	Description
<i>text</i>	A text string that contains the text to replace. You can also specify a reference to a cell that contains text.
<i>old_text</i>	The text string to be replaced.
<i>new_text</i>	The replacement text.
<i>instance</i>	Specifies the occurrence of <i>old_text</i> to replace. If this argument is omitted, every instance of <i>old_text</i> is replaced.

Examples This function returns “Second Quarter Results”:

SUBSTITUTE("First Quarter Results", "First", "Second")

This function returns "Shipment 45, Bin 52":

SUBSTITUTE("Shipment 45, Bin 45", "45", "52", 2)

See Also **REPLACE**
TRIM

SUM

Description Returns the sum of the supplied numbers.

Syntax **SUM** (*number_list*)

Parameter	Description
<i>number_list</i>	<p>A list of as many as 30 numbers, separated by commas.</p> <p>The list can contain numbers, logical values, text representations of numbers, or a reference to a range containing those values.</p> <p>Error values or text that cannot be translated into numbers return errors.</p> <p>If a range reference is included in the list, text, logical expressions, and empty cells in the range are ignored.</p>

Examples

This function returns 6000:

SUM(1000, 2000, 3000)

This function returns 4000 when each cell in the range contains 1000:

SUM(A10:D10)

See Also

AVERAGE
COUNT
COUNTA
PRODUCT
SUMSQ

SUMIF

Description

Returns the sum of the specified cells based on the given criteria.

Syntax

SUMIF (*range*, *criteria*, *sum_range*)

Parameter	Description
<i>range</i>	The range of cells you want evaluated.
<i>criteria</i>	A number, expression, or text that defines which cells are added. For example, <i>criteria</i> can be expressed as 15, “15”, “>15”, “cars”.
<i>sum_range</i>	The actual cells to sum. These cells are only summed if their corresponding cells in <i>range</i> match the criteria. If this argument is omitted, the cells in <i>range</i> are summed.

See Also

AVERAGE
COUNT
COUNTA
COUNTIF
PRODUCT
SUMSQ
SUM

SUMPRODUCT

Description Multiplies the corresponding cells in the given ranges, then returns the sum of those products.

Syntax SUMPRODUCT (*range_list*)

Parameter	Description
<i>range_list</i>	Two or more range references that provide the sets of numbers you want to multiply. The first cell in the first range is multiplied with the first cell in the second range. Then all the products are summed.

Remarks All the ranges in *range_list* must contain the same number of cells in the same arrangement. That is, if the first range is three rows deep and three columns wide, the second and subsequent ranges must also be three rows deep and three columns wide.

Examples The following examples use this worksheet.

	A	B	C	D
1	1	4	7	
2	2	5	8	
3	3	6	9	
4				
5				

This formula returns 32:

SUMPRODUCT(A1:A3,B1:B3)

That is, $1 \times 4 = 4$
 $2 \times 5 = 10$
 $3 \times 6 = 18$
 32

This formula returns 630:

SUMPRODUCT(A1:C1,A2:C2,A3:C3)

That is, $1 \times 2 \times 3 = 6$
 $4 \times 5 \times 6 = 120$
 $7 \times 8 \times 9 = 504$
 630

This formula returns 50:

SUMPRODUCT(A1:A3,C1:C3)

SUMSQ

Description Squares each of the supplied numbers and returns the sum of the squares.

Syntax **SUMSQ** (*number_list*)

Parameter	Description
<i>number_list</i>	<p>A list of as many as 30 numbers, separated by commas.</p> <p>The list can contain numbers, logical values, text representations of numbers, or a reference to a range containing those values.</p> <p>Error values or text that cannot be translated into numbers return errors.</p> <p>If a range reference is included in the list, text, logical expressions, and empty cells in the range are ignored.</p>

Example This function returns 302:

SUMSQ(9, 10, 11)

See Also **SUM**

SYD

Description Returns the depreciation of an asset for a specified period using the sum-of-years method. This depreciation method uses an accelerated rate, where the greatest depreciation occurs early in the useful life of the asset.

Syntax **SYD** (*cost*, *salvage*, *life*, *period*)

Parameter	Description
<i>cost</i>	The initial cost of the asset.
<i>salvage</i>	The salvage value of the asset.
<i>life</i>	The number of periods in the useful life of the asset.
<i>period</i>	The period for which to calculate the depreciation. The time units used to determine <i>period</i> and <i>life</i> must match.

Example This function returns 1607.14:

SYD(10000, 1000, 7, 3)

See Also **DDB**
SLN
VDB

T

Description Tests the supplied value and returns the value if it is text.

Syntax **T** (*value*)

Parameter	Description
<i>value</i>	The value to test.

Remarks Empty text (" ") is returned for any value that is not text.

Examples This function returns Report:
`T("Report")`

This function returns empty text (" ") if A4 contains a number:
`T(A4)`

See Also **N**
VALUE

TAN

Description Returns the tangent of the specified angle.

Syntax **TAN** (*number*)

Parameter	Description
<i>number</i>	The angle in radians. To convert a number expressed as degrees to radians, multiply the degrees by PI()/180.

Examples This function returns 0.752:
`TAN(0.645)`

This function returns 1:
`TAN(45*PI()/180)`

See Also **ATAN**
ATAN2
PI
TANH

TANH

Description Returns the hyperbolic tangent of a number.

Syntax **TANH** (*number*)

Parameter	Description
<i>number</i>	Any number.

Examples This function returns $-.96$:

TANH(-2)

This function returns $.83$:

TANH(1.2)

See Also
ATANH
COSH
SINH
TAN

TEXT

Description Returns the given number as text, using the specified formatting.

Syntax **TEXT** (*number*, *format*)

Parameter	Description
<i>number</i>	Any value, a formula that evaluates to a number, or a reference to a cell that contains a value.
<i>format</i>	A string representing a number format. The string can be any valid format string including "General," "M/DD/YY," or "H:MM AM/PM." The format must be surrounded by a set of double quotation marks. Asterisks cannot be included in <i>format</i> .

Examples This function returns 123.620:

TEXT(123.62, "0.000")

This function returns 10/19/94:

TEXT(34626.2, "MM/DD/YY")

See Also **DOLLAR**
FIXED
T
VALUE

TIME

Description Returns a serial number for the supplied time.

Syntax **TIME** (*hour, minute, second*)

Parameter	Description
<i>hour</i>	A number from 0 to 23.
<i>minute</i>	A number from 0 to 59.
<i>second</i>	A number from 0 to 59.

Examples This function returns .52:

TIME(12, 26, 24)

This function returns .07:

TIME(1, 43, 34)

See Also **HOURL**
MINUTE
NOW
SECOND
TIMEVALUE

TIMEVALUE

Description Returns a serial number for the supplied text representation of time.

Syntax **TIMEVALUE** (*text*)

Parameter	Description
<i>text</i>	A time in text format.

Examples This function returns .07:

TIMEVALUE("1:43:43 am")

This function returns .59:

`TIMEVALUE("14:10:07")`

See Also

HOUR
MINUTE
NOW
SECOND
TIME

TODAY

Description

Returns the current date as a serial number.

Syntax

TODAY ()

Remarks

This function is updated only when the worksheet is recalculated.

See Also

DATE
DAY
NOW

TRIM

Description

Removes all spaces from text except single spaces between words.

Syntax

TRIM (*text*)

Parameter	Description
<i>text</i>	Any text string or a reference to a cell that contains a text string.

Remarks

Text that is imported from another environment may require this function.

Example

This function returns Level 3, Gate 45:

`TRIM(" Level 3, Gate 45 ")`

See Also

CLEAN
MID
REPLACE
SUBSTITUTE

TRUE

Description	Returns the logical value True. This function always requires the trailing parentheses.
Syntax	TRUE ()
See Also	FALSE

TRUNC

Description	Truncates the given number to an integer.						
Syntax	TRUNC (<i>number</i> [, <i>precision</i>])						
	<table><tr><th>Parameter</th><th>Description</th></tr><tr><td><i>number</i></td><td>Any value.</td></tr><tr><td><i>precision</i></td><td>The number of decimal places allowed in the truncated number. Omitting this argument assumes a <i>precision</i> of 0.</td></tr></table>	Parameter	Description	<i>number</i>	Any value.	<i>precision</i>	The number of decimal places allowed in the truncated number. Omitting this argument assumes a <i>precision</i> of 0.
Parameter	Description						
<i>number</i>	Any value.						
<i>precision</i>	The number of decimal places allowed in the truncated number. Omitting this argument assumes a <i>precision</i> of 0.						
Remarks	TRUNC removes the fractional part of a number to the specified precision without rounding the number.						
Examples	<p>This function returns 123.45:</p> <p><code>TRUNC(123.456, 2)</code></p> <p>This function returns 9800:</p> <p><code>TRUNC(9899.435, -2)</code></p>						
See Also	CEILING FLOOR INT MOD ROUND						

TYPE

Description Returns the argument type of the given expression.

Syntax **TYPE** (*expression*)

Parameter	Description
<i>expression</i>	Any expression.

Remarks The valid values returned by this argument are:

Number	Description
1	Number
2	Text string
4	Logical value
16	Error value

Examples This function returns 1 if cell A1 contains a number:

TYPE(A1)

This function returns 2:

TYPE("Customer")

See Also

ISBLANK
ISERR
ISERROR
ISLOGICAL
ISNA
ISNONTEXT
ISNUMBER
ISREF
ISTEXT

UPPER

Description Changes the characters in the specified string to uppercase characters.

Syntax **UPPER** (*text*)

Parameter	Description
<i>text</i>	Any string.

Remarks	Numeric characters in the string are not changed.
Examples	<p>This function returns 3RD QUARTER:</p> <p>UPPER("3rd Quarter")</p> <p>This function returns JOHN DOE:</p> <p>UPPER("JOHN DOE")</p>
See Also	LOWER PROPER

USDOLLAR

Description	Returns the specified number as text using the US Dollar format and the supplied precision. Omitting the precision argument assumes two decimal places.						
Syntax	USDOLLAR (<i>number</i> [, <i>precision</i>])						
	<table><tr><th>Parameter</th><th>Description</th></tr><tr><td><i>number</i></td><td>A number, a formula that evaluates to a number, or a reference to a cell that contains a number.</td></tr><tr><td><i>precision</i></td><td>A value representing the number of decimal places to the right of the decimal point. Omitting this argument assumes two decimal places.</td></tr></table>	Parameter	Description	<i>number</i>	A number, a formula that evaluates to a number, or a reference to a cell that contains a number.	<i>precision</i>	A value representing the number of decimal places to the right of the decimal point. Omitting this argument assumes two decimal places.
Parameter	Description						
<i>number</i>	A number, a formula that evaluates to a number, or a reference to a cell that contains a number.						
<i>precision</i>	A value representing the number of decimal places to the right of the decimal point. Omitting this argument assumes two decimal places.						
Examples	<p>This function returns \$1023.79:</p> <p>USDOLLAR(1023.789)</p> <p>This function returns \$500:</p> <p>USDOLLAR(495.301, -2)</p>						
See Also	FIXED TEXT VALUE DOLLAR						

VALUE

Description Returns the specified text as a number.

Syntax **VALUE** (*text*)

Parameter	Description
<i>text</i>	Any text string, a formula that evaluates to a text string, or a cell reference that contains a text string. You can also specify a date or time in a recognizable format (for example, M/DD/YY for dates or H:MM AM/PM for time). If the format is not recognized, #VALUE! is returned.

Examples This function returns 9800:

VALUE(9800)

This function returns 123:

VALUE("123")

See Also **DOLLAR**
FIXED
TEXT

VAR

Description Returns the variance of a population based on a sample of values.

Syntax **VAR** (*number_list*)

Parameter	Description
<i>number_list</i>	A list of as many as 30 numbers, separated by commas. The list can contain numbers or a reference to a range that contains numbers.

Example This function returns .31:

VAR(4.0, 3.0, 3.0, 3.5, 2.5, 4.0, 3.5)

See Also **STDEV**
STDEVP
VARP

VARP

Description Returns the variance of a population based on an entire population of values.

Syntax **VARP** (*number_list*)

Parameter	Description
<i>number_list</i>	A list of as many as 30 numbers, separated by commas. The list can contain numbers or a reference to a range that contains numbers.

Example This function returns .27:
 VARP(4.0, 3.0, 3.0, 3.5, 2.5, 4.0, 3.5)

See Also **STDEV**
STDEVP
VAR

VDB

Description Returns the depreciation of an asset for a specified period using a variable method of depreciation.

Syntax **VDB** (*cost, salvage, life, start_period, end_period* [, *factor*] [, *method*])

Parameter	Description
<i>cost</i>	The initial cost of the asset.
<i>salvage</i>	The salvage value of the asset.
<i>life</i>	The number of periods in the useful life of the asset.
<i>start_period</i>	The beginning period for which to calculate the depreciation. The time units used to determine <i>start_period</i> and <i>life</i> must match.
<i>end_period</i>	The ending period for which to calculate the depreciation. The time units used to determine <i>end_period</i> and <i>life</i> must match.
<i>factor</i>	The rate at which the balance declines. Omitting this argument assumes a default of 2, which is the double-declining balance factor.
<i>method</i>	A logical value that determines if you want to switch to straight-line depreciation when depreciation is greater than the declining balance calculation. Use True to maintain declining balance calculation; use False or omit the argument to switch to straight-line depreciation calculation.

Example This function returns 1041.23:

VDB(10000, 1000, 7, 3, 4)

See Also DDB
SLN
SYD

VLOOKUP

Description Searches the first column of a table for a value and returns the contents of a cell in that table that corresponds to the location of the search value.

Syntax VLOOKUP (*search_item*, *search_range*, *column_index*)

Parameter	Description
<i>search_item</i>	A value, text string, or reference to a cell containing a value that is matched against data in the top row of <i>search_range</i> .
<i>search_range</i>	The reference of the range (table) to be searched. The cells in the first column of <i>search_range</i> can contain numbers, text, or logical values. The contents of the first column must be in ascending order (for example, -2, -1, 0, 2...A through Z, False, True). Text searches are not case-sensitive.
<i>column_index</i>	The column in the search range from which the matching value is returned. <i>column_index</i> can be a number from 1 to the number of rows in the search range. If <i>column_index</i> is less than 1, #VALUE! is returned. When <i>column_index</i> is greater than the number of rows in the table, #REF! is returned.

Remarks VLOOKUP compares the information in the first column of *search_range* to the supplied *search_item*. When a match is found, information located in the same row and supplied column (*column_index*) is returned.

If *search_item* cannot be found in the first column of *search_range*, the largest value that is less than *search_item* is used. When *search_item* is less than the smallest value in the first column of the *search_range*, #REF! is returned.

Examples

The following examples use this worksheet.

	A	B	C	D	E	
1	Employee	StartDate	Emp. No.	Salary	Exempt	
2	Anderson	10/15/84	2348	\$37,800	Y	
3	Clark	2/6/90	4891	\$28,700	N	
4	Davis	6/21/80	2480	\$46,950	Y	
5	Franklin	4/20/88	3793	\$30,275	Y	
6	Lee	8/30/89	3961	\$25,000	N	
7	Olson	11/1/81	2578	\$45,780	Y	
8	Turner	2/15/93	5129	\$26,100	N	
9	Wilson	9/1/89	3965	\$31,650	Y	
10						

Sheet1

This function returns \$28,700:

`VLOOKUP("Clark", A2:E9, 4)`

This function returns 3961:

`VLOOKUP("Lee", A2:E9, 3)`

See Also

HLOOKUP
INDEX
LOOKUP
MATCH

WEEKDAY

Description

Returns the day of the week that corresponds to the supplied date.

Syntax

WEEKDAY (*serial_number*)

Parameter	Description
<i>serial_number</i>	The date as a serial number or as text (for example, 06-21-94 or 21-Jun-94).

Remarks

WEEKDAY returns a number ranging from 1 (Sunday) to 7 (Saturday).

Examples

This function returns 1, indicating Sunday:

`WEEKDAY(34399.92)`

This function returns 3, indicating Tuesday:

`WEEKDAY("06/21/94")`

See Also

DAY
NOW
TEXT
TODAY

YEAR

Description

Returns the year that corresponds to the supplied date.

Syntax

YEAR (*serial_number*)

Parameter	Description
<i>serial_number</i>	The date as a serial number or as text (for example, 06-21-94 or 21-Jun-94).

Examples

This function returns 1993:

YEAR(34328)

This function returns 1994:

YEAR("06/21/94")

See Also

DAY
NOW
HOUR
MINUTE
MONTH
SECOND
TODAY
WEEKDAY

Tidestone

Index

A

About box
 displaying every 30 minutes xi, xii
 viewing xii
 AboutBox method xii
 ABS function 203
 Absolute cell references 70
 ACOS function 203
 ACOSH function 204
 Active
 cell 57, 58, 60
 worksheet 50, 59
 ActiveX
 Add-Ins in Visual Basic 183–188
 controls, adding to your application 1
 AddColPageBreak method 153
 Add-In functions
 creating 183
 formula evaluation errors, chart 198
 requirements for 184
 AddPageBreak method 153
 ADDRESS function 205
 AddRowPageBreak method 153
 AddSelection method 61
 AlignHorizontal property 108
 Aligning data 107–108
 Center toolbar button 43
 Left Align toolbar button 43
 Merge and Center toolbar button 43
 Right Align toolbar button 43
 SetAlignment method 52
 AlignVertical property 108
 Allocating memory 179
 AllowArrows property 86
 AllowCellTextDlg property 81, 86
 AllowDelete property 58, 81, 86
 AllowDesigner property 37
 AllowEditHeaders property 81, 82, 121
 AllowFillRange property 81, 86
 AllowFormulas property 81, 85
 AllowInCellEditing property 81
 AllowMoveRange property 81, 86
 AllowObjSelections property 62, 129, 146
 AllowResize property 81, 86, 113
 AllowSelections property 62
 AllowTabs property 86
 Anchors in HTML files 171

AND function 206
 Arc command (Insert menu) 40
 Arcs
 creating 123–125
 tool for 125
 Area references as arguments 185
 Arguments
 2D area reference 192
 2D, 3D, and union 192
 and return values 191
 in functions 73, 74
 Arithmetic operators 68
 Array arguments in functions 183
 #ARRAY_FORMULA! error 72
 Arrow keys 58, 85
 ASIN function 206
 ASINH function 207
 ATAN function 208
 ATAN2 function 208
 ATANH function 209
 Attach method 27, 30
 AttachToSS method 27, 30
 Autofill lists 74
 adding 75
 deleting 76
 dragging for filling 88
 AutoFillItems property 75, 76
 AutoFillItemsCount method 75
 Automatic recalculation 77
 AutoPageNumber property 155
 AutoRecalc property 77, 78
 AVERAGE function 209
 Axes page, in Chart Wizard 144

B

Background color, for graphical objects 133
 Bar gap, on charts 143
 Binding worksheet columns 169
 Black and white printing 159
 BlackAndWhite property 159
 BLOBs
 reading and writing workbooks as 33
 using multiple versions of Formula One and 33
 Bold toolbar button 43
 Boolean 184
 BorderColor property 117

Borders

- on cells 116–118
- on graphical objects 134–135

BorderStyle property 117**BottomMargin** property 154**Bring to Front** command (Format menu) 41, 132**Built-in**

- formats 99–100
- functions 72–74

Button command (Insert menu) 40**Buttons**

- creating 123–125
- formatting 137
- tool for 125

ByVal 183

C

C++ Add-In API 189–198**Calculating workbooks** 77–79

- automatically 77
- circular references when 79

CALL function 78**Cancel Insert Object** command (Insert menu) 40**Cancelling data entry** or editing 58**CEILING** function 212**Cell reference** in formula bar 46, 64**Cell references** 69–71

- as function arguments 74
- automatically entering 71
- for check boxes 138
- for dropdown list boxes 135, 138
- in formulas 69–71
- in validation rules 82
- of merged cells 109
- to other workbooks 71
- to other worksheets 70
- with named row and column headings 121

Cells

- active 57, 58, 60
- aligning data in 107–108
- attributes of 87
- borders on 116–118
- clearing 97
- deleting 98
- displaying data types, with type markers 46
- formatting to optimize memory 177
- inserting 96
- limiting number displayed on a worksheet 55–56
- locking 81–82
- merged 109–110
- naming 76
- preventing resizing of 86

Cells (*continued*)

- preventing selection of 62
 - programatically clearing 98
 - selecting 60–61
 - selecting with properties and methods 61
 - selecting with the keyboard 58–59
 - selecting with the mouse 60
 - type markers 46
- Cells** command (Format menu) 40, 81, 83, 99, 102, 107, 108, 109, 117, 118
- Cells** command (Insert menu) 39
- Center Across** toolbar button 43
- Center** toolbar button 43
- CenterHoriz** property 154
- CenterVert** property 154
- CFormView**-based applications, in Visual C++ 2–3
- adding Formula One to 4–5
- CHAR** function 213
- Chart** command (Insert menu) 39, 141
- Chart Designer** 145
- Chart Wizard** 142–146
- labeling chart axes in 144
 - modifying existing charts with 144
 - navigating in 142
 - selecting chart types in 143
 - selecting styles in 143
 - setting chart elements in 144
- Charts** 141–147
- changing chart data 145
 - creating 123–125, 142–147
 - data range for 146
 - data requirements for different types 142
 - formatting with the Chart Designer 145
 - modifying data 145
 - modifying, with the Chart Wizard 144
 - referencing data on a different worksheet 147
 - selecting styles 143
 - selecting types 143
 - tool for 125
- Check boxes**
- creating 123–125
 - formatting 136
 - input value cells for 138
 - selection values for 137–139
 - tool for 125
- Checkbox** command (Insert menu) 40
- CHOOSE** function 214
- Circles.** *See* Ovals
- Circular references**, calculating 79
- CLEAN** function 215
- Clear All** command (Edit menu) 38
- Clear Contents** command (Edit menu) 38
- Clear Formats** command (Edit menu) 38

- ClearClipboard method 92
- Clearing, cells 97
- ClearRange method 52, 98
- Clipboard, pasting from, with methods 92
- Close command (File menu) 37
- CODE function 215
- Col property (FindReplaceInfo object) 95
- ColHeadings property 159
- ColHidden property 113
- Colon (:), in range references 69
- Color palette 48
- Color toolbar button 43
- Colors
 - defining, in color palette 48
 - for borders of graphical objects 134
 - for cell borders 117
 - for cell data 107
 - for filling cells 118–119
 - for graphical objects 133–134
 - in number formats 105
 - maximum number, per workbook 180
 - printing as shades of gray 159
- ColText property 52, 121
- Column Autofit Selection command (Format menu) 40
- Column Default Width command (Format menu) 41
- COLUMN function 216
- Column headings
 - formatting 119
 - hiding 56
 - preventing changes to 82
 - preventing resizing of 86
 - printing 159
 - selecting 119
 - sizing 120
 - text of 120–121
- Column Hide command (Format menu) 40
- Column Unhide command (Format menu) 41
- Column width
 - affected by default font 46
 - affected by displaying formulas 56
- Column Width command (Format menu) 40
- Columns
 - applying number formats 101
 - building to optimize memory 177
 - deleting 98
 - freezing 116
 - headings (*see* column headings)
 - inserting 96
 - maximum width 180
 - preventing selection of 62
 - print titles and 151
 - selecting 61
 - setting widths of 110–114
- Columns (*continued*)
 - automatically 113
 - default 111
- Columns command (Insert menu) 39
- COLUMNS function 78, 216
- ColWidth property 52, 113
- ColWidthDlg method 52, 113
- ColWidthTwips property 113
- ColWidthUnits property 46, 114
- Commas
 - in cell references 70
 - in functions 73
- Common Fixed and General Formats toolbar button 43
- Comparison operators 68
- CONCATENATE function 217
- Conditional values, in number formats 105
- Connect string 163
- Constant values
 - entering 66–67
 - naming 76
 - using to optimize memory 177
- Constants, not read in Visual C++ 2
- Context menu 57
- Converting from previous versions of Formula One 20
- Copy Cell Format command (Edit menu) 38
- Copy command (Edit menu) 38
- Copy Format toolbar button 42
- Copy toolbar button 42
- CopyAll method 92
- CopyDataFromArray method 93
- CopyDataToArray method 93
- Copying
 - cell references 92
 - data
 - across ranges 93
 - and optimizing performance 178
 - by dragging 57, 88
 - with drag-and-drop 88
 - with menu commands 91
 - with methods 92, 93
 - formulas 92
- CopyRange method 52, 93, 178
- CopyRangeEx method 7, 93, 178
- COS function 217
- COSH function 218
- COUNT function 218
- COUNTA function 219
- COUNTIF function 220
- CreateNewCellFormat method 117
- Currency toolbar button 43
- Currency, formatting numbers as 100
- Custom data formats 102–106
- Cut command (Edit menu) 38

Cut toolbar button 42

Cutting

- cell references 92

data

- and optimizing performance 178

- with menu commands 91

- with methods 92, 93

- formulas 92

CView-based applications, in Visual C++ 2–3

- adding Formula One to 5–6

D

Data

- aligning 107–108

- clearing 98

- filling worksheets with, and optimizing memory 179

- formatting 99–108

- built-in formats for 99–100

- custom formats for 102–106

- inserting or updating in databases 167

- ranges of (*see* ranges)

- retrieving from database 161–170

- sorting 98

- types of 66

- validating 82

Data entry 63–65

- constant values and 66–67

- formulas and 67–72

- limiting 80–86

- multi-line 65

- properties for 64

- with check boxes and dropdown list boxes 137

Data Grid Editor 145

Data structure 178

Data types, displaying with type markers 46

Databases 161–170

- binding worksheet columns in 169

- connecting to 161, 162

- disconnecting from 170

- installing ODBC drivers for 161

- PREPARE statements for 168, 170

- querying 164

- setting up 162

- workbooks in 33

DataTransferRange property 93

Date and Time toolbar button 44

DATE function 220

Dates

- entering 67

- formatting numbers as 101, 104

- printing, in headers and footers 157

- range of, accepted by Formula One x

DATEVALUE function 221

DAY function 221

Days of the week, automatically entering names of 74

DB function 223

DDB function 223

Declaring Add-In Functions in C++ 189

Default Font command (Format menu) 41

Default Height command (Format menu) 111

Default Width command (Format menu) 111

Defaults

- column width 111

- font 46

- for data alignment 107

- for locking cells 82

- number format 99

- row height 111

- workbook display options 45–48

- worksheet display options 55–56

DefColWidthDlg method 52

DefinedName property 76

DefineSearch method ix, 95

DefRowHeightDlg method 52

Delete command (Edit menu) 39

Delete Sheet command (Edit menu) 39, 54

DeleteAutofillItems method 76

DeleteDlg method 52

DeleteRange method 52, 98

DeleteSheets method 54

Deleting

- autofill lists 76

- cells 97, 98

- columns 98

- page breaks 152–153

- ranges 98

- rows 98

- worksheets 54

Development environments, for Formula One 1

Dialog-based applications, in Visual C++ 2–3

- adding Formula One to 4–5

Disconnecting from databases 170

Displaying

- cell reference in formula bar 46, 64

- cells on a worksheet 55–56

- data types, with type markers 46

- formula bar 46, 64

- formulas 72

- parts of the workbook 45–48

- parts of worksheets 55–56

- type markers 46

- worksheet tabs 46

- zero values 56

#DIV/0! error 72

Docking the toolbars 37

- Documentation conventions xiii
- DOLLAR function 224
- Dollar sign (\$), in cell references 70
- DoSafeEvents property 175
- Double 184
- Drag-and-drop
 - copying data using 88
 - copying from other applications using 90
 - moving data using 88
- Dragging
 - copying using 88
 - filling from autofill lists using 88
 - preventing users from 86
- Drawing and Forms Toolbar command (View menu) 39
- Drawing toolbar 36
- Drawing, graphical objects interactively 124
- Drivers, ODBC 161
- Dropdown List box command (Insert menu) 40
- Dropdown list boxes
 - creating 123–125
 - formatting 135
 - input value cells for 138
 - items in 136
 - numbering scheme for items 137
 - selection values for 137–139
 - tool for 125

E

- Edge of worksheet, moving the active cell to 59
- Edit menu, in Workbook Designer 38
- Edit mode 58
 - See also* In-cell editing
- EditClear method 52, 98
- EditCopy method 92
- EditCopyDown method 93, 178
- EditCopyRight method 93, 178
- EditCut method 92
- EditDelete method 52, 98
- EditDeleteSheets method 54
- EditInsert method 52, 96, 98
- EditInsertSheets method 53
- EditPaste method 92
- EditPasteValues method 92
- Embedding Data in HTML files 171
- EnableProtection property 52, 81
- EndCol property 131
- EndRow property 131
- EnterMovesDown property 58, 86
- Entry property 52, 64, 83
- EntryRC property 52, 64, 83
- EntrySRC property 52

- Equal sign (=)
 - in formulas 67
 - in functions 73
- Error values, default alignment of 107
- ERROR.TYPE function 225
- Errors
 - entering 67
 - function syntax 74
 - trapping, in PowerBuilder 20
 - worksheet formula 72
- Escape key 58
- EVEN function 226
- Events, Formula One, in Visual C++ 7
- EXACT function 227
- Excel
 - 4.x file format 32, 37
 - features ignored in Formula One 32
 - graphical objects 123
 - password-protected files 32
 - reading and writing files 32
 - validation rules 83
- Exclamation mark (!), in sheet references 70
- Executing PREPARE statements 170
- Exit command (File menu) 38
- EXP function 227
- External references 71

F

- F1AddInArray 184, 185
- F1AddInArrayEx 184, 185
- F1AddInInit 190
- F1AddInRegisterFunctionProc 191
- F1AddInRegisterInfoProc 190
- F1Book API object 23
 - guidelines for using 30
 - understanding the 26–27
- F1BookView API object 23
 - guidelines for using 30
 - properties and methods that cannot be used with 29
 - understanding the 27–30
- F1CellFormat API object 24, 116, 117
- F1EventArgs API object 24
- F1FileSpec API object 24
- F1FindReplaceInfo API object 23
- F1Functions 183
- F1NumberFormat API object 24
- F1ObjPos API object 24, 131
- F1ODBCConnect API object 24, 162
- F1ODBCQuery API object 24, 164
- F1PateSetup API object 24
- F1RangeRef API object 24, 61
- F1Rect API object 24

- FlReplaceResults API object 24
- FACT function 228
- FALSE function 228
- File menu, in Workbook Designer 37
- FilePageSetupDlg method 52, 149
- FilePageSetupDlgEx method 52
- FilePrint method 149
- FilePrintEx method 7, 150
- FilePrintPreview method 160
- FilePrintSetupDlg method 150
- Fill command (Edit menu) 38
- Filling
 - preventing users from, with mouse 86
 - ranges 74
 - worksheets with data, and optimizing memory 179
- Find and replace 94
 - using properties and methods 95
 - using the Workbook Designer 94
- Find command (Edit menu) 39
- Find dialog box 94
- FIND function 228
- FINDB function 229
- FindDlg method 95
- FindNext method x, 95
- First Impression ActiveX control 141–147, 161
 - Chart Designer 145
 - Data Grid Editor 145
 - using to draw charts 125, 141
- FirstPageNumber property 155
- FitPages property 158
- FIXED function 230
- FixedCol property 116
- FixedRow property 116
- Fixing rows and columns. *See* Freezing panes
- FLOOR function 230
- Fonts
 - changing default 46
 - default affects column width 46
 - formatting 106–107
 - Formula One's default 47
 - maximum number, per workbook 180
 - scripts in 47, 107
- FooterMargin property 154
- Footers
 - formatting codes for 156–157
 - margins for 154
 - printing 156–157
- Footnotes, for charts 144
- Foreground color, for graphical objects 133
- Format menu, in Workbook Designer 40
- FormatAlignmentDlg method 52
- FormatBorderDlg method 52
- FormatCellsDlg method 52, 84, 102, 108, 118, 119
- FormatFontDlg method 52
- FormatNumberDlg method 52
- FormatObjectDlg method 128, 134, 135, 136
- FormatPaintMode property 38
- FormatPatternDlg method 52
- FormatSheetDlg method 52
- FormattedText property 101
- FormattedTextRC property 101
- FormattedTextSRC property 101
- Formatting
 - cell borders 116–118
 - cells, to optimize memory 177
 - column widths 110–114
 - programmatically 114
 - data 99–108
 - fonts 106–107
 - graphical objects 133–137
 - row and column headings 119
 - row heights 110–114
 - programmatically 114
 - with built-in data formats 99–100
 - with custom data formats 102–106
- Formatting toolbar 36, 43
- Formula bar 36
 - displaying 46, 64, 80
- Formula Bar command (View menu) 39, 64
- Formula evaluation errors 198
- Formula One
 - 1.x file format 32
 - 2.x file format 20, 32, 37
 - adding the component to PowerBuilder 12–13, 16, 19
 - adding the component to Visual Basic 8–9
 - adding the component to Visual C++ 3–6
 - data structure of 178
 - documentation conventions of xiii
 - features ignored in Excel 32
 - installation of x–xii
 - new features of ix
 - objects 23–24
 - technical support of xii
 - trial version of x, xii
 - upgrading from previous versions 20
 - using multiple versions of 21, 33
- Formula property 52, 65, 67
- FormulaLocal property 52
- FormulaLocalRC property 52
- FormulaLocalSRC property 52
- FormulaRC property 52, 65, 67
- Formulas
 - cell references in 69–71
 - copying data and 92
 - cutting data and 92
 - displaying 56, 72

Formulas (*continued*)

- entering 67–72
- errors in 72
- external cell references in 71
- functions and 72–74
- limiting entry of 85
- maximum length 180
- naming 76
- operators in 68
- operators precedence in 69

FormulaSRC property 52, 67

Fraction toolbar button 44

Fractions, formatting numbers as x, 100

Freeing memory 179

Freeze Panes command (Format menu) 41, 115, 116

Freezing panes 115–116

- example of 115
- programmatically 116

Functions 72–74, 203–295

- arguments for 74
- cell references in 74
- entering 73
- maximum number of arguments 180
- nesting 73
- syntax errors in 74
- using array arguments in 183

FV function 231

G

Gallery page, in Chart Wizard 143

GetArrayType 185

GetCellFormat method 81

GetIteration method 79

Goto command (Edit menu) 39

Graphical objects

- arranging layers of 132–133
- borders 134–135
- creating 123–125
- drawing interactively 124
- editing polygons 139–140
- fill colors and patterns 133–134
- formatting 133–137
- hiding 135
- identifying 126
- input value cells for 138
- moving 130–132
- naming 128
- not supported in Excel 123
- preventing selection of 62, 129
- repositioning 57
- resizing 57
- selecting 128–130

Graphical objects (*continued*)

- selection values for 137–139
- sizing 130–132
- using mouse mode when creating 126

Grid lines

- aligning graphical objects to 125, 130
- hiding 56
- printing 159

GridLines property 159

H

HAlign property 52

HdrHeight property 52, 120

HdrWidth property 52, 120

HeaderMargin property 154

Headers

- formatting codes for 156–157
- margins for 154
- printing 156–157
- row and column (*see* row headings, column headings)

Headings, creating using merged cells 109

Height command (Format menu) 112

Hiding

- calculations 27
- column headings 56
- graphical objects 135
- grid lines 56
- row headings 56
- type markers 46
- worksheet tabs 31, 46
- zero values 56

HLOOKUP function 232

HOUR function 233

HTML 171

- anchors 171
- reading and writing file types 32
- source files 171
- writing a range to an HTML file 34

I

IDataObject 93

Identifying graphical objects by number 126

IF function 234

Images, importing 125

Import command (File menu) 37

In-cell editing 57

- disabling 85
- in-cell edit space x
- See also* Edit mode

INDEX function 78, 234

- Index list of worksheets 50, 53, 71
- Index numbers, of worksheets 31
- INDIRECT function 78, 235
- Insert menu, in Workbook Designer 39
- InsertDlg method 52
- InsertHTML method 34, 172
- Inserting
 - cells 96
 - columns 96
 - database data 167
 - rows 96
 - worksheets 49, 52
- InsertRange method 52, 96
- InsertSheets method 53
- Installing, Formula One x–xii
- Instanting property 183
- INSTPROB.DOC file xii
- INT function 236
- IntelliPoint mouse 57
- Internet development
 - events for 174
 - methods for 174
 - performing 173–175
- Intranet development
 - events for 174
 - methods for 174
 - performing 173–175
- Invisible workbooks
 - attaching 30
 - See also* F1BookView API object
- IObjectSafety interface 171, 174
- IPMT function 236
- IRR function 237
- ISBLANK function 238
- ISERR function 239
- ISERROR function 239
- ISLOGICAL function 240
- ISNA function 240
- ISNONTEXT function 241
- ISNUMBER function 242
- ISREF function 242
- ISTEXT function 243
- Italic toolbar button 43
- Items, in dropdown list boxes 136
- IterationEnabled property 79
- IterationMax property 79
- IterationMaxChange property 79
- Iterations
 - circular references and 79
 - maximum number allowed in worksheet 180
- IterNext and IterStart 185
- IterStart and IterNext 185

K

- Keyboard commands 58
 - limiting user access to 85
- Keywords, in functions 73

L

- Landscape printing orientation 157
- LaunchDesigner method 36
- Launching the Workbook Designer 36
- Layout page, in Chart Wizard 144
- Left Align toolbar button 43
- LEFT function 243
- LEFTB function 244
- LeftMargin property 154
- Left-to-right printing 158
- LeftToRight property 158
- Legends, for charts 144
- LEN function 244
- LENB function 245
- Limiting data entry 80–86
- Line command (Insert menu) 40
- Line feeds, cell entries and 65
- LineColor property 135
- Lines
 - as borders on graphical objects 134–135
 - creating 123–125
 - formatting 134–135
 - styles 134
 - tool for 124
- LineStyle property 135
- LineWeight property 135
- List boxes. *See* Dropdown list boxes
- List, in dropdown list box 135
- LN function 245
- Locking cells 81–82
 - displaying message 82
- LOG function 246
- LOG10 function 246
- Logical property 52, 65
- Logical values
 - default alignment of 107
 - entering 67
- LogicalRC property 52, 65
- LogicalSRC property 52
- LOOKUP function 247
- LOWER function 248

M

- Main toolbar 36
- Margins 154
- MATCH function 249
- MAX function 250
- MaxCol property 31, 56
- MaxRow property 31, 56
- Member variables, in Visual C++ 4–5
- Memory
 - allocating 179
 - freeing 179
 - optimizing use of 177–180
- Menus, in Workbook Designer 37–41
- Merge and Center toolbar button 43
- MergeCells property 109
- Merging cells 109–110
- MID function 251
- MIDB function 252
- MIN function 253
- MinCol property 31
- Minimal recalculation 78
- MinimalRecalc property 79
- MinRow property 31
- MINUTE function 253
- MIRR function 254
- MOD function 255
- Mode property 126
- MONTH function 255
- Months, automatically entering names of 74
- Mouse
 - actions of 57
 - drawing graphical objects 125
 - IntelliPoint mouse x
 - limiting access to certain actions 86
 - selecting graphical objects 129
 - setting mode of 126
 - with checkboxes and dropdown list boxes 138
- MoveRange method 52, 93, 178
- Moving
 - active cell 58
 - data
 - and optimizing performance 178
 - with drag-and drop 88
 - with dragging 57
 - with menu commands 91
 - with methods 92, 93
 - graphical objects 130–132
 - worksheet tabs 46
- Multi-line data entry 58, 65, 108
- preventing 85

N

- N function 256
- #N/A error 72
- NA function 256
- Name command (Insert menu) 40, 76
- #NAME? error 72
- Names
 - built-in 77
 - cell 76
 - column heading 57
 - constant value 76
 - defining 76
 - formula 76
 - graphical object 128
 - maximum length 180
 - maximum number, per workbook 180
 - range 76
 - row heading 57
 - top left corner 57
 - worksheet 55, 57
 - worksheet and workbook, in headers and footers 157
- Navigating in worksheets 56–59
 - with keyboard commands 58
 - with mouse actions 57
 - with properties and methods 61
- Nesting functions 73
- New command (File menu) 37
- New toolbar button 42
- NextColPageBreak method 153
- NextRowPageBreak method 153
- NOT function 257
- NOW function 78, 257
- NPER function 258
- NPV function 259
- #NULL! error 72
- #NUM! error 72
- Number property 52, 65
- NumberFormat property 52, 102
- NumberFormatLocal property 52
- Numbering, pages 155
- NumberRC property 52, 65
- Numbers
 - as currency 100
 - as dates 101
 - as entries 66
 - as fractions 100
 - as percentages 103
 - as percents 100
 - as times 101
 - in scientific notation 100, 103
- NumberSRC property 52
- NumSheets property 53, 54

O

- ObjAddItem method 136
- ObjAddSelection method 129
- ObjBringToFront method 127, 128, 132
- ObjCellCol property 139
- ObjCellRow property 139
- ObjCellType property 139
- ObjClick event 128, 129
- ObjCreate method 123, 124
- ObjCreatePicture method 124, 126
- ObjDbClick event 128, 129
- ObjDeleteItem method 136
- Object command (Format menu) 41, 127, 128, 133, 134, 136, 137, 139, 146, 147
- Objects
 - Formula One 23–24
 - in Visual C++ 6
 - graphical (*see* graphical objects)
- ObjFirstID method 127
- ObjGetCell method 139
- ObjGetItemCount method 136
- ObjGetPos method 131, 132
- ObjGetSelection method 127, 129
- ObjGetSelectionCount method 130
- ObjGotFocus event 128
- ObjInsertItem method 136
- ObjItem property 136
- ObjItems property 136
- ObjLostFocus event 128
- ObjName property 128
- ObjNameToID method 127
- ObjNew method 123, 124, 141
- ObjNewPicture method 124, 126
- ObjNextID method 127
- ObjPatternBG property 128, 133
- ObjPatternFG property 128, 133
- ObjPatternStyle property 128, 133
- ObjPosToTwips method 132
- ObjPosToTwipsEx method 132
- ObjSelection property 130
- ObjSendToBack method 127, 128, 132
- ObjSetCell method 139
- ObjSetPicture method 126
- ObjSetPos method 131
- ObjSetSelection method 129
- ObjText property 137, 138
- ObjValue property 138
- ObjValueChanged event 128
- ObjVisible property 135
- ODBC 161–170
 - binding of worksheet columns 169
 - configuring, in PowerBuilder 10

- ODBC (*continued*)
 - connecting to the data source 162
 - drivers 161
 - PREPARE statements 168, 170
 - setting up the data source 162
- ODBCBindParameter method 167, 169, 170
- ODBCBindParameterEx method 167, 169, 170
- ODBCConnect method 162, 167
- ODBCConnectEx method 162, 167
- ODBCDisconnect method 170
- ODBCError method 167, 170
- ODBCErrorMsg property 167, 170
- ODBCExecute method 167, 170
- ODBCExecuteError event 168, 170
- ODBCExecuteEx method 167, 170
- ODBCNativeError property 167, 170
- ODBCPrepare method 167, 168, 169, 170
- ODBCPrepareEx method 167, 168, 169, 170
- ODBCQuery method 164, 168
- ODBCQueryEx method 164, 168
- ODBCSQLState property 168, 170
- ODD function 259
- OFFSET function 78, 260
- Open command (File menu) 37
- Operators
 - formula 68
 - precedence of 69
- Optimizing Formula One 177–180
- Options command (Tools menu) 41, 78, 79
- OptionsDlg method 75, 77, 79
- OR function 261
- Oval command (Insert menu) 40
- Ovals
 - creating 123–125
 - tool for 124
- Overlapping graphical objects 132–133

P

- Page Break command (Insert menu) 40, 152
- Page breaks, setting and removing 152–153
- Page numbers 155
 - in headers and footers 157
- Page Setup command (File menu) 38, 151, 152, 154, 155, 156, 157, 158
- PagesTall property 158
- PagesWide property 158
- PaletteEntry property 48
- Paper size 158
- PaperSize property 158
- Parameters passed by reference, in PowerBuilder 20
- Parentheses, in functions 73
- Paste command (Edit menu) 38

- Paste Special command (Edit menu) 38
 - with merged cells 110
 - Paste toolbar button 42
 - Pasting
 - data, with methods 92
 - merged cells 109
 - Patterns
 - for filling cells 118–119
 - for graphical objects 133–134
 - Percent toolbar button 44
 - Percentages, formatting numbers as 103
 - Percents, formatting numbers as 100
 - PI function 261
 - Picture objects 124, 125
 - PMT function 262
 - Point, as measure for line weight 134
 - PolyEditMode property 139
 - Polygon command (Insert menu) 40
 - Polygon point editing 124, 139
 - Polygon Points command (Edit menu) 38
 - Polygons
 - editing 139–140
 - tool for 125
 - tool for edit mode 124
 - Portrait printing orientation 157
 - PowerBuilder 9–20
 - and application templates 11
 - and Formula One, web page 9
 - and ODBC 10
 - errors in 20
 - syntax 19
 - tutorials for Formula One 10–18
 - working in 19–20
 - working with databases in 10–11, ??–14
 - PPMT function 263
 - PREPARE statements 168, 170
 - Print area 77
 - Print Area command (File menu) 38
 - Print command (File menu) 38, 149
 - Print preview 159–160
 - in Visual C++ 7
 - Print Preview command (File menu) 38, 159
 - Print Preview toolbar button 42
 - Print titles 77
 - Print Titles command (File menu) 38
 - Print toolbar button 42
 - PrintArea property 151
 - PrintAreaLocal property 151
 - PrintFooter property 156
 - PrintHeader property 156
 - Printing 149–160
 - from print preview screen 160
 - headers and footers 156–157
 - Printing (*continued*)
 - in Visual C++ 7
 - margins 154
 - page breaks, setting and removing 152–153
 - page numbers 155
 - page order 158
 - paper size 158
 - portrait or landscape 157
 - print area 150–151
 - print preview 159–160
 - row and column headings 159
 - scaling data 157
 - sizing data to fit pages 155
 - titles 151
 - PrintLandscape property 157
 - PrintPreviewDCEX method 7, 160
 - PrintPreviewEx method 160
 - PrintScale property 158
 - PrintTitles property 152
 - PrintTitlesLocal property 152
 - PRODUCT function 264
 - PROPER function 264
 - Properties command (Format menu) 41, 55, 56, 62, 82, 85
 - Properties command (Format Sheet menu) 86
 - Properties of Formula One
 - in PowerBuilder 19
 - in Visual Basic 9
 - in Visual C++ 8
 - Properties, entering data with 64
 - Protecting the worksheet 81–82
 - Protection command (Format menu) 41, 81
 - ProtectionDlg method 52
 - ProtectionHidden property 52, 81
 - ProtectionLocked property 52, 81
 - PV function 265
- ## Q
- Querying databases 164
 - QueryInterface 192
- ## R
- RAND function 78, 266
 - Ranges
 - building to optimize memory 177
 - copying data across 93
 - deleting 98
 - editing, using methods 92
 - filling 74
 - naming 76
 - references as function arguments 74

Ranges (*continued*)

- references in formulas 69–71
- selecting 57, 60
- source, for charts 141
- writing to file 34

RATE function 266

Read command (File menu) 37

Read method 20, 33

Read toolbar button 42

ReadEx method 20, 33

ReadFromBlob method 33

Reading

- BLOBs 33
- Excel-compatible files 32
- Formula One files 32
- HTML files 32
- tabbed text files 32

Read-only, making workbook 80–81

Recalc command (Tools menu) 41

Recalc method 78

Recalculation

- automatic workbook 77
- circular references and 79
- minimal 78
- workbook 58, 77–79

Rectangle command (Insert menu) 40

Rectangles

- creating 123–125
- tool for 124

#REF! error 72

Reference operators (for cell ranges) 68

References to cells. *See* Cell references

Relative cell references 70

- in validation rules 82

Remove Page Break command (Insert menu) 153

RemoveColPageBreak method 153

RemovePageBreak method 153

RemoveRowPageBreak method 153

Renaming worksheets 55

Repaint property 177

Repaint, disabling to optimize memory 177

Replace command (Edit menu) 39

Replace dialog box 94

REPLACE function 267

Replace method *x*, 95

REPLACEB function 269

ReplaceDlg method 95

REPT function 269

Retrieving data from databases 164

Right Align toolbar button 43

RIGHT function 270

RightMargin property 154

ROUND function 271

ROUNDUP function 273

Row Default Height command (Format menu) 40

ROW function 272, 273

Row headings

- formatting 119
- hiding 56
- preventing changes to 82
- preventing resizing of 86
- printing 159
- selecting 119
- sizing 120
- text of 120–121

Row Height command (Format menu) 40

Row Hide command (Format menu) 40

Row mode 62

Row property (F1FindReplaceInfo object) 95

Row Unhide command (Format menu) 40

RowHeadings property 159

RowHeight property 52, 114

RowHeightDlg method 52, 114

RowHidden property 114

RowMode property 62

Rows

- applying number formats 101
- building to optimize memory 177
- deleting 98
- freezing 115
- headings (*see* row headings)
- inserting 96
- maximum height 180
- preventing selection of 62
- print titles and 151
- selecting 61
- selecting, with row mode 62
- setting heights of 110–114
 - automatically 111, 112, 113
 - default 111
 - dialog box for 112

Rows command (Insert menu) 39

ROWS function 78, 274

RowText property 52, 121

S

Safe events, understanding 175

Save As command (File menu) 37

Save command (File menu) 37

Save toolbar button 42

SaveFileDialog method 33, 171

SaveFileDialogEx method 33, 171

Scale

- for printing data 157
- for viewing worksheets 56

- Scientific notation, formatting numbers as 100, 103
- Scripts, in fonts 47, 107
- Scroll lock key 59
- Scrolling mode 57
- SEARCH function 274
- SEARCHB function 275
- Searching for and replacing data 94
- SECOND function 276
- SelChange event 31
- Select All Objects command (Edit menu) 38, 129
- Selecting
 - cells 60–61
 - graphical objects 128–130
 - row and column headings 119
 - rows and columns 57, 61
 - user options for 62
 - using the mouse 57
 - with properties and methods 61
 - with the keyboard 58–59
 - with the mouse 60
 - worksheets 50–51, 57
- Selection property 61
- SelectionCount property 61
- SelectionEx property 61
- SelHdrCol property 119
- SelHdrRow property 119
- SelHdrTopLeft property 119
- Send to Back command (Format menu) 41, 132
- Serial number, for Formula One x, xi
- Series labels, on charts 143
- Set Print Area command (File menu) 150
- Set Print Titles command (File menu) 152
- SetAlignment method 52
- SetBorder method 52
- SetBorderEx method 52
- SetCellFormat method 52, 81, 117
- SetColWidth method 52, 114
- SetColWidthAuto method 66, 114
- SetColWidthTwips method 114
- SetDefaultFont method 47
- SetDefaultFontEx method 47
- SetFont method 52
- SetFontEx method 52, 107
- SetHdrSelection method 119
- SetIteration method 79
- SetLineStyle method 128, 135
- SetPageSetup method 52
- SetPattern method 52, 118, 128, 133
- SetProtection method 52
- SetRowHeight method 52, 114
- SetRowHeightAuto method 114
- Setup.exe file xi
- SetValidationRule method 52
- Sheet limits 56
- Sheet Properties command (Format menu) 41, 55, 56, 62, 82, 85, 86, 116
- Sheet property 31, 46
- Sheet Protection command (Format menu) 41, 81
- SheetName property 55
- SheetSelected property 51, 53, 54
- ShowColHeading property 56
- ShowEditBar property 46, 64, 81
- ShowEditBarCellRef property 46, 64
- ShowFormulas property 56, 72
- ShowGridLines property 56
- ShowHScrollBar property 57
- ShowLockedCellsError property 82
- ShowRowHeading property 56
- ShowTabs property 31, 46
- ShowTypeMarkers property 46, 177
- ShowVScrollBar property 57
- ShowZeroValues property 56
- SIGN function 276
- SIN function 277
- SINH function 277
- Sizing graphical objects 130–132
- SLN function 278
- Sort command (Edit menu) 38
- Sort method 98
- SortDlg method 98
- Sorting data 98
- Specifications 180
- SQRT function 278
- SSClearRange function call 98
- SSDeleteRange function call 98
- SSEditClear function call 98
- Stacking, on charts 143
- Standalone Workbook Designer 36
- Standard Toolbar command (View menu) 39
- StartCol property 131
- StartRow property 131
- Status Bar command (View menu) 39
- STDEV function 279
- STDEVP function 279
- String 184
- Strings, in formulas 68
- Style page, in Chart Wizard 143
- SUBSTITUTE function 280
- SUM function 280
- SUMIF function 281
- SUMPRODUCT function 282
- SUMSQ function 282
- SYD function 283

T

- T function 284
- Tabbed text, reading and writing files 32
- Tabs, worksheet
 - displaying 46
 - enlarging space for 49
 - hiding 31, 46
- TAN function 284
- TANH function 285
- Technical specifications 180
- Templates, for applications in PowerBuilder 11
- Text
 - displayed next to check boxes 136
 - displayed on buttons 137
 - entering 67
 - obtaining formatted 101
- TEXT function 285
- Text operators 68
- Text property 52, 65
- TextRC property 52, 65
- TextSRC property 52
- Thread safety
 - and F1Functions 184
- 3D charts 143
- Tidestone Technologies, Inc.
 - Case Tracking System xii
 - contact information xii
 - Europe office xiii
 - website 171
- TIME function 286
- Times
 - entering 67
 - formatting numbers as 101, 104
 - printing, in headers and footers 157
- TIMEVALUE function 286
- Title property 70
- Titles, for charts 144
- TODAY function 78, 287
- Toggle Drawing Toolbar toolbar button 42
- Toolbars 36, 42–44
 - docking 37
- Tools menu, in Workbook Designer 41
- TopLeftText property 52, 121
- TopMargin property 154
- Top-to-bottom printing 158
- TRIM function 287
- TRUE function 288
- TrueType fonts 47, 106
- TRUNC function 288
- TTF16.h file 2
- Tutorials, using PowerBuilder 10–18
- 2D charts 143

- TYPE function 289
- Type markers 46

U

- Underline toolbar button 43
- Unfreeze Panes command (Format menu) 41
- Uniform Data Transfer 93
- Updating database data 167
- Upgrading 20
- UPPER function 289
- USDOLLAR function 290
- Users
 - limiting access 46, 80–86
 - to cells 81–82
 - to certain keys 85
 - to changing row and column headings 82
 - to charts 146
 - to entering certain values 82
 - to entering formulas 85
 - to graphical objects 129
 - to mouse actions 86
 - to workbook areas 30
 - to workbooks 80–81
 - setting selection options for 62

V

- Validating data 82
- Validation rules 82
- ValidationFailed event 84
- ValidationRule property 84
- ValidationRuleDlg method 52
- ValidationRuleLocal property 84
- ValidationRuleLocalRC property 84
- ValidationRuleRC property 84
- ValidationText property 84
- VAlign property 52
- #VALUE! error 72
- VALUE function 291
- VAR function 291
- Variant 184
- VARP function 292
- VDB function 292
- Versions of Formula One 21, 33
- View menu, in Workbook Designer 39
- View scale, for worksheets 56
- Views
 - attaching 27, 30
 - information stored with 26
 - saving 32
 - working with 26

- ViewScale property 56
- Visual Basic 8–9
 - adding the Formula One component 8–9
 - properties and methods, for Formula One 9
- Visual basic
 - using add-ins in 183
- Visual C++ 1–8
 - adding the Formula One component 3–6
 - Formula One objects in 6
 - handling Formula One events 7
 - printing and previewing in 7
 - properties and methods, for Formula One 6, 8
 - serializing the Formula One control in 7
- VLOOKUP function 293

W

- Web pages containing Formula One 173
- WEEKDAY function 294
- Weight, line 134
- White space 177
- Width command (Format menu) 112
- Windows metafiles, importing 125
- Windows Registration Database xi
- WordWrap property 52, 108
- Workbook Designer 25, 35–44
 - Edit menu commands 38
 - File menu commands 37
 - Format menu commands 40
 - Insert menu commands 39
 - keyboard commands 58
 - launching 36
 - menus 37–41
 - mouse actions 57
 - standalone 36
 - toolbars 42
 - Tools menu commands 41
 - View menu commands 39
- Workbooks
 - attaching views to 26, 30
 - calculating 77–79
 - automatically 77
 - circular references and 79
 - deleting worksheets from 54
 - denying user access to 80–81
 - description of 24
 - display options 30, 45–48
 - index sheet list in 50, 53
 - inserting worksheets in 49, 52
 - reading 32
 - retrieving, from database tables 33
 - selecting worksheets in 50–51
 - writing 32
- Workbooks (*continued*)
 - See also* F1Book API object
- Worksheet command (Insert menu) 39, 53
- Worksheet functions. *See* Functions
- Worksheets
 - active 50
 - building to optimize memory 177
 - cells in
 - borders for 116–118
 - clearing 98
 - data alignment in 107–108
 - deleting 98
 - inserting 96
 - locking 81–82
 - data entry of 63–65
 - constant values in 66–67
 - formulas in 67–72
 - limiting 80–86
 - multi-line 65
 - data types in 66
 - deleting 54
 - display options 55–56
 - external references in 71
 - formatting 99–108
 - index list of 31, 50, 53, 71
 - inserting 49, 52
 - limiting rows and columns displayed 55–56
 - maximum size 180
 - naming 55
 - navigating in 56–59
 - performing tasks on multiple 50
 - preventing selection of 62
 - printing 149
 - referencing in cell references 70
 - selecting 50–51
 - selection in
 - cells 60–61
 - rows and columns 61
 - sorting data in 98
 - tabs on 46
 - working with a group 51
- Wrapper classes, in Visual C++ 6
- Wrapping text 108
- Write command (File menu) 37
- Write method 33
- WriteEx method 33
- WriteRange method 34
- WriteRangeEx method 7, 34
- WriteToBlob method 33
- WriteToBlobEx method 33
- Writing
 - a range of cell data 34
 - BLOBs 33

Writing (*continued*)

Excel-compatible files 32

Formula One files 32

HTML files 32, 171

tabbed text files 32

X

X axis labels, on charts 144

Y

Y axis labels, on charts 144

YEAR function 295

Z

Z axis labels, on charts 144

Zero values, displaying or hiding 56

Zooming in and out

in printing worksheets 157

in viewing worksheets 56