Contents

Introduction .............................................................................................................................................. 2
  What’s in this manual................................................................................................................................. 2
  Prerequisites................................................................................................................................................ 2
  Product support ............................................................................................................................................ 2
  Symbols and conventions .......................................................................................................................... 3
  What’s next.................................................................................................................................................. 3

Part 1: Using VBA ...................................................................................................................................... 6

Chapter 1: VBA Overview ....................................................................................................................... 7
  The Visual Basic Editor ............................................................................................................................... 7
  The VBA project.......................................................................................................................................... 8
  Objects.......................................................................................................................................................... 9
  Programming model .................................................................................................................................. 11
  Using VBA with the Modifier ................................................................................................................... 12
  Online help............................................................................................................................................... 12
  Multiple third-party products.................................................................................................................. 12

Chapter 2: Programming Windows ......................................................................................................... 13
  Working with windows in a VBA project ................................................................................................. 13
  Programming a window object............................................................................................................... 14
  Using window events............................................................................................................................... 15
  Window open events.............................................................................................................................. 15
  Window close events ............................................................................................................................. 16
  Window activate events .......................................................................................................................... 17
  Modal dialog events ............................................................................................................................... 17
  Using window properties and methods................................................................................................. 19

Chapter 3: Programming Window Fields .............................................................................................. 23
  Working with window fields in your VBA project ................................................................................... 23
  Programming a window field object ..................................................................................................... 24
  Using window field events..................................................................................................................... 25
  Got focus events..................................................................................................................................... 26
  User changed events.............................................................................................................................. 27
  Changed event......................................................................................................................................... 29
  Lost focus events.................................................................................................................................... 29
  Using window field properties and methods....................................................................................... 31
  Working with field values .................................................................................................................... 33
  Verifying field values ........................................................................................................................... 36
  Window field type reference ................................................................................................................ 39

Chapter 4: Programming Grids .............................................................................................................. 55
  Working with grids in your VBA project ................................................................................................. 55
  Programming a grid object ................................................................................................................... 57
  Types of grids .......................................................................................................................................... 58
### CONTENTS

Using grid events .......................................................................................................................... 59
Using grid properties and methods ............................................................................................ 62

**Chapter 5: Programming Reports** ......................................................................................... 63
  Working with reports in your VBA project .................................................................................. 63
  Using report events ....................................................................................................................... 64
  Start/End events ............................................................................................................................ 65
  Band events .................................................................................................................................... 66
  Using report properties and methods .......................................................................................... 69
  Working with report fields .......................................................................................................... 70
  Exporting data to other applications ......................................................................................... 72

**Chapter 6: Storing Additional Data** ...................................................................................... 75
  Understanding the DUOS ................................................................................................................. 75
  Structure of the DUOS ..................................................................................................................... 76
  Working with DUOS data objects ...................................................................................................... 77
  Retrieving a DUOS data object ....................................................................................................... 78
  Deleting a DUOS data object ........................................................................................................... 78
  Updating a DUOS data object ........................................................................................................... 78
  DUOS properties and methods ....................................................................................................... 79
  Using the DUOS ............................................................................................................................... 80

**Chapter 7: Distributing a Project** ......................................................................................... 87
  Enabling VBA .................................................................................................................................. 87
  References to external components ............................................................................................... 87
  Distributing project files .................................................................................................................. 88
  Package files .................................................................................................................................. 88
  Package file import/export issues .................................................................................................... 91

**Part 2: VBA Object Library** .................................................................................................. 94

**Chapter 8: Window Object** .................................................................................................... 95
  Activate method ............................................................................................................................... 96
  AfterActivate event ........................................................................................................................ 97
  AfterClose event .............................................................................................................................. 98
  AfterModalDialog event .................................................................................................................. 99
  AfterOpen event ............................................................................................................................. 100
  BeforeActivate event .................................................................................................................... 101
  BeforeClose event .......................................................................................................................... 102
  BeforeModalDialog event .............................................................................................................. 103
  BeforeOpen event .......................................................................................................................... 104
  Caption property ............................................................................................................................ 106
  Changed property ............................................................................................................................ 107
  Close method .................................................................................................................................. 108
  EventMode property ....................................................................................................................... 109
  Height property .............................................................................................................................. 110
  Hide method ................................................................................................................................... 111
  IsLoaded property .......................................................................................................................... 112
  Left property ................................................................................................................................... 113
  Move method .................................................................................................................................. 114
  Name property ............................................................................................................................... 115
Chapter 9: Field Object ................................................................. 125
  AfterGotFocus event ................................................................. 126
  AfterLostFocus event ................................................................. 127
  AfterUserChanged event ............................................................ 128
  BeforeGotFocus event ................................................................. 129
  BeforeLostFocus event ............................................................... 130
  BeforeUserChanged event .......................................................... 131
  Caption property ........................................................................ 132
  Changed event ........................................................................... 133
  Empty property ........................................................................... 134
  Enabled property ........................................................................ 135
  Focus method ............................................................................... 136
  FocusSeg method ....................................................................... 137
  Height property .......................................................................... 138
  Left property ............................................................................... 139
  Locked property .......................................................................... 140
  Move method ............................................................................... 141
  Name property ........................................................................... 142
  Object property .......................................................................... 143
  Parent property .......................................................................... 144
  Required property ....................................................................... 145
  TabStop property ....................................................................... 146
  Top property ................................................................................ 147
  Value property ........................................................................... 148
  ValueSeg property ...................................................................... 150
  Visible property .......................................................................... 151
  Width property ........................................................................... 152

Chapter 10: Grid Object ................................................................. 153
  AfterLineChange event ............................................................... 154
  AfterLineGotFocus event ............................................................ 155
  AfterLineLostFocus event ............................................................ 156
  BeforeLineChange event .............................................................. 157
  BeforeLineGotFocus event ........................................................... 158
  BeforeLineLostFocus event .......................................................... 159
  BeforeLinePopulate event ............................................................ 160
  Changed property ........................................................................ 161
  EventMode property .................................................................... 162
  Hide method ................................................................................ 163
  Left property ............................................................................... 164
  Move method ............................................................................... 165
  Name property ........................................................................... 166
  Show method ............................................................................... 167
  Top property ................................................................................ 168
  Visible property .......................................................................... 169

Chapter 11: Report Object ................................................................. 171
  BeforeAF event ........................................................................... 172
  BeforeAH event .......................................................................... 173
Introduction

Microsoft Visual Basic® for Applications, or VBA, is a standard programming language used by Microsoft® Office products, as well as by many other software providers who choose to embed VBA. Applications such as Microsoft Dynamics® GP host VBA as part of the Modifier. The tools available in the VBA environment allow you to customize windows, fields and reports. In addition, you can attach VBA code to fields you create using the Modifier.

What’s in this manual

The VBA Developer’s Guide is designed to teach the basics of using VBA with Microsoft Dynamics GP. The manual is divided into the following parts:

- **Part 1, Using VBA**, provides information on how you can program windows, window fields, grids and reports, as well as store additional application data.
- **Part 2, VBA Object Library**, provides reference information about properties, methods and events for objects in Microsoft Dynamics GP.

Prerequisites

Keep in mind that VBA is a powerful programming language, and a basic knowledge of VBA programming concepts is required. If you haven’t worked with VBA before, we recommend you review one of the many books available that introduces basic programming.

The remainder of the manual assumes that you are already familiar with VBA’s programming concepts and features, as well as Microsoft Dynamics GP. It explains how to use your existing VBA knowledge to work with the VBA capabilities available with the accounting system.

Product support

Technical support for VBA can be accessed by the following methods:

- **Telephone support** – Technical Support at (888) 477-7877 between 8:00 a.m. and 5:00 p.m. Central Time, Monday through Friday. International users can contact Technical Support at (701) 281-0555.
- **Internet** – VBA Technical Support is also available online through CustomerSource or PartnerSource, and is accessible from [www.microsoft.com/Dynamics/GP](http://www.microsoft.com/Dynamics/GP).
Symbols and conventions

To help you use the VBA documentation more effectively, we’ve used the following symbols and conventions within the text to make specific types of information stand out.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Light bulb]</td>
<td>The light bulb symbol indicates helpful tips, shortcuts and suggestions.</td>
</tr>
<tr>
<td>![Warning]</td>
<td>Warnings indicate situations you should be especially aware of.</td>
</tr>
</tbody>
</table>

*Margin notes summarize important information.*

<table>
<thead>
<tr>
<th>Convention</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Part 1, Using VBA</td>
<td>Bold type indicates a part name.</td>
</tr>
<tr>
<td>Chapter 1, “VBA Overview”</td>
<td>Quotation marks indicate a chapter name.</td>
</tr>
<tr>
<td>Setting window fields</td>
<td>Italicized type indicates a section name.</td>
</tr>
<tr>
<td>set 'l_Item' to 1.</td>
<td>This font is used for VBA code examples.</td>
</tr>
<tr>
<td>RUNTIME.EXE</td>
<td>Words in uppercase indicate a file name.</td>
</tr>
<tr>
<td>Visual Basic for Applications (VBA)</td>
<td>Acronyms are spelled out the first time they’re used.</td>
</tr>
<tr>
<td>TAB or ALT+M</td>
<td>Small capital letters indicate a key or a key sequence.</td>
</tr>
</tbody>
</table>

What’s next

Before you can begin using VBA, be sure to complete the following tasks:

1. **Install Microsoft Dynamics GP.**
   The runtime engine available with Microsoft Dynamics GP includes the components necessary to support the VBA environment.

2. **Register the Modifier.**
   To activate the VBA environment, you must register the Modifier. The registration keys to register the Modifier are supplied with the accounting system. After you’ve registered the Modifier, you can access the Visual Basic environment.

3. **Review Chapter 1, “VBA Overview.”**
   This chapter explains the basics of how the accounting system integrates with the Visual Basic environment. After you’ve reviewed the information in this chapter, you can use the remainder of the manual to learn specific ways you can use VBA with the accounting system.
Part 1: Using VBA

Use the information in this portion of the documentation to understand how to begin using VBA with Microsoft Dynamics GP. The following is a list of the topics discussed:

- Chapter 1, “VBA Overview,” explains the basics of the VBA environment and how it integrates with Microsoft Dynamics GP. It also explains the components of Microsoft Dynamics GP you’ll use when creating projects in VBA.

- Chapter 2, “Programming Windows,” explains how to use VBA with the window object.

- Chapter 3, “Programming Window Fields,” explains how to use VBA with the window field objects.

- Chapter 4, “Programming Grids,” explains how to use VBA with the grid object (scrolling window).

- Chapter 5, “Programming Reports,” explains how to use VBA with the report object.

- Chapter 6, “Storing Additional Data,” explains how to store user-defined data using the Dynamic User Object Store (DUOS).

- Chapter 7, “Distributing a Project,” explains how to package and deliver a VBA project.
Chapter 1: VBA Overview

Before you start using VBA, it’s important that you’re familiar with the VBA components specific to Microsoft Dynamics GP. The following sections explain each of these in detail:

- The Visual Basic Editor
- The VBA project
- Objects
- Programming model
- Using VBA with the Modifier
- Online help
- Multiple third-party products

The Visual Basic Editor

You will use the Visual Basic Editor for building and maintaining a VBA project. Once you register the Modifier, the Visual Basic Editor becomes accessible though the Tools submenu of the Microsoft Dynamics GP menu. The following illustration shows this menu:

To open the Visual Basic Editor, point to Customize on the Tools submenu, and then choose Visual Basic Editor.

⚠️ To create or edit VBA code, you must have appropriate user privileges. Typically, this means being part of the Administrators group or the Power Users group. If User Account Control (UAC) is active, it means launching Microsoft Dynamics GP with Administrative privileges.
The following illustration shows how an existing project looks when viewed in the Visual Basic Editor:

The Visual Basic Editor is similar for each host application, so if you’ve used VBA with Microsoft Excel or Word, the Visual Basic Editor in Microsoft Dynamics GP will be familiar.

**The VBA project**

The Visual Basic Editor stores customizations in a project file. The first time you open the Visual Basic Editor, the runtime engine automatically creates a new, empty project file that is named based on the current dictionary. For example, the VBA project created for Microsoft Dynamics GP is DYNAMICS.VBA.

*If you’re working with a third-party application, VBA creates a separate project to store customizations named NAME.VBA, where NAME is the name of the third-party product’s application dictionary.*

You can view a project using the Visual Basic Project Explorer:

Windows, reports and fields are not automatically part of the project. When you begin working with VBA, you’ll select the window, field, and report objects you want to work with, and add them to your project. Once added to your project, you can reference these objects in VBA. In addition, the Visual Basic Editor lets you add VBA user forms, user-defined procedures and user-defined class modules to your VBA project.
The VBA Editor lets you develop and maintain a single project at a time. To work with several different projects, store each project file in a separate location. When you need to work on a project, copy it to the same location of the runtime engine, then rename the file `name.VBA`, where `name` is the name of the dictionary the project is based on.

After you’ve completed a VBA project file, you’ll distribute it to users. This file contains your VBA code, the objects you’ve referenced in your VBA code, and any VBA user forms you choose to use.

**Objects**

Applications expose their functionality to VBA through a set of programmable objects. These objects include windows, reports, grids and Dynamic User Object Store (DUOS) objects. The relationship each object has to another object is expressed in a hierarchical *object model*, shown in the following illustration:

```
Window
  Window fields
Report
  Report fields
Grid
  Window fields
UserInfo
  DUOSObjects
    DUOSObject
      DUOSProperties
        DUOSproperty
```

The organization of the model is important, since you must navigate through the object model within VBA code to access lower-level objects. For example, to access a window field, you must first indicate the field’s window object:

```
' Set a field on a window
CustomerMaintenance.CustomerID.Value = "AARONFIT0001"
```

The following sections explains each object in more detail.
**Windows**
Most of the VBA programming you’ll complete for windows will involve the following objects.

<table>
<thead>
<tr>
<th>Object</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Window</td>
<td>Represents an open window.</td>
</tr>
<tr>
<td>Window field</td>
<td>Represents a field in a window.</td>
</tr>
</tbody>
</table>


**Grids**
Grids are VBA’s definition of a scrolling window. When working with grids, you’ll use the following objects:

<table>
<thead>
<tr>
<th>Object</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grid</td>
<td>Represents a grid (scrolling window) in an open window.</td>
</tr>
<tr>
<td>Window field</td>
<td>Represents a field in a grid.</td>
</tr>
</tbody>
</table>

Refer to Chapter 4, “Programming Grids,” for more information about using VBA with grids, and Chapter 3, “Programming Window Fields,” for information about programming window fields within a grid.

**Reports**
When working with reports, you’ll use the following objects:

<table>
<thead>
<tr>
<th>Object</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Report</td>
<td>Represents a report.</td>
</tr>
<tr>
<td>Report field</td>
<td>Represents a field in a report.</td>
</tr>
</tbody>
</table>

Refer to Chapter 5, “Programming Reports,” for more information about using VBA with reports.

**User information**
When implementing a VBA integration, you may need to retrieve information about the user currently logged into Microsoft Dynamics GP. The UserInfo object provides access to this information. Refer to Chapter 17, “UserInfo Object,” for more information about retrieving user information.

**Dynamic User Object Store (DUOS)**
The Dynamic User Object Store (DUOS) allows you to store and display user-definable data.

<table>
<thead>
<tr>
<th>Object/Collection</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DUOSObjects collection</td>
<td>Contains all user-definable data objects.</td>
</tr>
<tr>
<td>DUOSObject object</td>
<td>Represents a single data object.</td>
</tr>
<tr>
<td>DUOSProperties collection</td>
<td>Contains all properties for a data object.</td>
</tr>
<tr>
<td>DUOSProperty object</td>
<td>Represents a single property for a data object.</td>
</tr>
</tbody>
</table>

Refer to Chapter 6, “Storing Additional Data,” for more information about using VBA with the DUOS.
Programming model

Each object in the object model has defined *methods* and *properties* you can use in VBA code to manipulate the behavior of the object. Methods and properties for objects are declared using standard VBA *object.method* and *object.property* syntax. Most objects also have specific *events* that specify when associated VBA code executes for the object.

**Methods**
Methods are *actions* you can perform for a given object. Methods include opening and closing a window or moving a field in a window. For example, the field object uses the *Focus method* to move the focus to a different field in the window:

```
'Move the focus to the salesperson field
SalespersonID.Focus
```

**Properties**
Properties are characteristics you can retrieve or set for a given object. Properties include the title of a window, or the value of a field. For example, you can use the *Caption property* to change the name of a window or field:

```
'Change the prompt for a field
CustomerMaintenance.CustomerID.Caption = "Patient ID"

'Change the title of a window
CustomerMaintenance.Caption = "Patient Maintenance"
```

**Events**
You’ll write the majority of your VBA code within individual *event* procedures. Each event procedure executes VBA code at certain times for a specific object, such as when the user changes the value of a field, clicks a button, prints a report, or opens a window. You can see these predefined event procedures in the VBA Code window.

For example, *BeforeOpen* is an event for a window object. This event executes the associated event procedure when the specified window opens. In the following example, the *BeforeOpen event* executes a procedure when the Receivables Transaction Entry window opens:

```
Private Sub Window_BeforeOpen()
    SortBy.Value = 3 'Set the sort list to "by Date"
    DocumentType = 7 'Set the document type to "Returns"
End If
```
Using VBA with the Modifier

The primary reason to use VBA with the Modifier is to apply programming logic to new fields you add to a window using the Modifier. By using VBA with fields added using the Modifier, you can:

- Create push buttons that perform additional tasks, such as opening a VBA user form, launching another application, or performing calculations. Refer to Chapter 3, “Programming Window Fields,” for more information about using VBA with push button fields.

- Add new data entry fields that store additional data in the Dynamic User Object Store (DUOS). Refer to Chapter 6, “Storing Additional Data,” for more information about using the DUOS.

Online help

After you install and register the Modifier, you’ll have access to online help for the Visual Basic Editor, the VBA programming language, and for objects, methods and properties in Microsoft Dynamics GP. Use either of the following methods to display specific help topics:

- Open the help for Microsoft Dynamics GP. A link to the VBA help can be found there.

- Highlight a keyword in the Code window, such as an object’s property or method and press F1. Context-sensitive help for that item will appear in the help window.

Multiple third-party products

By default, VBA can work with objects from the dictionary that the VBA project is associated with. For example, the Microsoft Dynamics GP VBA project can work with resources from the Microsoft Dynamics GP dictionary (Dynamics.dic). It is possible to have your VBA project reference objects in other dictionaries. To do this, you must create a reference in the VBA project to the other dictionary.

To create a reference from a VBA project to another dictionary, select the VBA project in the VBA Explorer. Choose References from the Tools menu in the VBA environment. Mark the check box for the dictionary you want to make references to.

When creating references to other dictionaries, you can’t have circular references. For example, if your Microsoft Dynamics GP VBA project references the SmartList dictionary, the VBA project for the SmartList dictionary cannot have a reference back to the Microsoft Dynamics GP project.
Chapter 2: Programming Windows

Programming windows using VBA involves adding a window to your project, specifying the window events you’ll use to execute VBA code, and using window methods and properties to customize window display.

Information about programming windows is divided into the following sections:

- Working with windows in a VBA project
- Programming a window object
- Using window events
- Window open events
- Window close events
- Window activate events
- Modal dialog events
- Using window properties and methods

You’ll also likely want to customize a window using the window’s fields and scrolling windows (grids). Refer to Chapter 3, “Programming Window Fields,” and Chapter 4, “Programming Grids,” for detailed information about working with window fields and grids.

Working with windows in a VBA project

In order to use VBA with a Microsoft Dynamics GP window, you’ll first need to add the window to your project. Adding a window creates a corresponding window object in your project, which you can reference in VBA.

Adding a window object

To add a window object to your project, switch to Microsoft Dynamics GP and display the window you want to add. On the Tools menu, point to Customize, then choose Add Current Window to Visual Basic. This will add the currently-active (topmost) window to your project. The following illustration shows this menu:

Switch to the Visual Basic Editor and display the Project Explorer. It should look similar to the following:
You can continue to switch between the Visual Basic Editor and the main application to add windows as needed throughout your project’s development. When viewed in the accounting system, each window you’ve added to your project will appear with a period (.) at the end of the window’s title. This provides technical support and VBA developers with a visual cue that the window is part of the VBA project.

**Removing a window object**

To remove a window object from your project, switch to Microsoft Dynamics GP and display the window you want to remove. On the Tools menu, point to Customize, then choose Remove Current Window From Visual Basic. VBA will remove the window and any window fields associated with the window.

*When you remove a window from a project, be sure that you remove from your project all references to the window.*

**Programming a window object**

There are two primary ways you can write VBA code for a window object: using a window event procedure, or referencing the window in procedures throughout your project.

**Using a window event procedure**

A window event procedure executes VBA code when a user opens, closes or activates a window in Microsoft Dynamics GP. This is useful for setting default window field values when the window opens, or closing other windows when the window closes. In the following example, an event procedure sets default field values when the Invoice Entry window opens:

```vba
Private Sub Window_AfterOpen()
    'Set the sort list to "by Date"
    SortBy = 3
    'Set the document type list to "Returns"
    DocumentType = 2
End Sub
```

The following section, *Using window events*, explains each type of window event you can use to execute VBA code. Windows also contain window fields that you can reference and manipulate through VBA. Refer to *Chapter 3, “Programming Window Fields,”* for more information about using window fields.

**Referencing the window object**

After you create a window object in your project, any VBA code you write in your VBA project has full access to the window object, its properties and its methods. In the following example, a push button on a VBA form opens a window using the window’s `Open` method:

```vba
Private Sub OpenCustomerMaintenance_Click()
    CustomerMaintenance.Open
End Sub
```

The section titled *Using window properties and methods* explains the methods and properties available to procedures in your VBA project.
Using window events

A window event executes a specific VBA procedure (an event procedure) when a user opens, closes or activates a window, or when a message dialog appears for the window. To view the window events available, select a Microsoft Dynamics GP window object in your project, then display the Visual Basic Code window. Select Window in the Object list; window events will appear in the Procedure list.

A window event executes an event procedure either before or after Microsoft Dynamics GP application code runs for the same event. Therefore, window events use names like “BeforeClose” to indicate that the event runs as the window closes but before Microsoft Dynamics GP application code runs for the window. When you choose an event, VBA automatically adds the event procedure syntax to the Code window.

If you use VBA with a window you’ve modified using the Modifier, you must set the EventMode property to emModified if you want events to occur for the modified window. To change the EventMode property for the window object, use the Visual Basic Properties window.

If you delete a modified window from your forms dictionary (or delete the forms dictionary altogether), all VBA references to new fields you add to the modified window will be invalid. If you re-create the modified window, then re-add the new fields using the Modifier, you’ll need to re-add those same fields to your VBA project and re-create any VBA customizations that reference the new fields.

Window open events

VBA window open events occur when the user opens the window, either before or after the Microsoft Dynamics GP code for the window open event runs. Microsoft Dynamics GP uses the window open event to set default field values, such as a default document date or a sorting order.

**BeforeOpen event**

The BeforeOpen event occurs before the Microsoft Dynamics GP code runs for the window open event. It’s useful when you want to set default field values in the window, while still allowing any Microsoft Dynamics GP application code to override your defaults. In the following example, the event procedure sets the Checkbook ID field when the Invoice Batch Entry window opens. If this window displays an existing record when opened (such as when opened from a zoom field), Microsoft Dynamics GP application code overrides this value with the value stored with the record:

```vbp
Select Window in the Object list.
When you choose an event, the Code window adds event procedure syntax automatically.
Window events appear in the Procedures list.

Private Sub Window_BeforeOpen()
    ' Set default field values
    Me.CheckbookID = 12345
End Sub
```
Private Sub Window_BeforeOpen(OpenVisible As Boolean)
  'Set the checkbook ID
  CheckbookID = "PETTY CASH"
End Sub

This event also includes the OpenVisible parameter. When set to False, this specifies that the window should open invisibly.

**AfterOpen event**

The AfterOpen event occurs after the Microsoft Dynamics GP code runs for the window open event. It’s useful for overriding any default values that were set for window fields. In the following example, the event procedure marks both check box controls when the Sales Territory Maintenance window opens. Since Microsoft Dynamics GP application code sets defaults for these fields when the window opens, using the AfterOpen event overrides the defaults:

Private Sub Window_AfterOpen()
  'Set both check boxes when the window opens
  MaintainHistory = 1
  MaintainHistory1 = 1
End Sub

**Window close events**

VBA window close events occur when the user closes the window, either before or after the Microsoft Dynamics GP code for the window close event runs. Microsoft Dynamics GP uses the window close event to display a modal dialog asking the user whether to save changes to the record.

**BeforeClose event**

The BeforeClose event occurs before Microsoft Dynamics GP application code runs for the window close event. The primary use for this event is to cancel the window close if the contents in the window don’t meet certain criteria. The following event procedure for the Invoice Entry window checks whether the user entered a comment ID for the transaction, and displays a message dialog. If the user clicks Yes, the procedure cancels the close using the AbortClose parameter, and the user can enter a comment ID:

Private Sub Window_BeforeClose(AbortClose As Boolean)
  Dim Response As Integer
  If CommentID.Empty = True Then
    'Display a message box
    Response = MsgBox("Do you want to enter a Comment?", vbYesNo)
    If Response = vbYes Then
      'They want to enter a comment
      AbortClose = True
      CommentID.Focus
    End If
  End If
End Sub
**AfterClose event**

The AfterClose event occurs after the Microsoft Dynamics GP application code runs for the window close event. The AfterClose event is a general-purpose event you can use to perform any “clean-up” tasks, such as closing other windows. For example, the following event procedure runs when the Customer Maintenance window closes. It closes the Customers And Prospects lookup window if it’s open (its IsLoaded property is True):

```vba
Private Sub Window_AfterClose()
    If CustomersAndProspects.IsLoaded = True Then
        CustomersAndProspects.Close
    End If
End Sub
```

**Window activate events**

VBA window activate events occur when the user activates a window, either by opening it or by making it the frontmost window.

The BeforeActivate event occurs before any Microsoft Dynamics GP application code runs for the window activate event. The AfterActivate event occurs after any Microsoft Dynamics GP application code runs for the window activate event.

Microsoft Dynamics GP rarely executes application code during a window activate event. Therefore, whether you choose the BeforeActivate or AfterActivate event may be of little consequence.

**Modal dialog events**

A modal dialog is a specific type of window that requires the user to perform some action in order to dismiss the dialog. The most common modal dialog is an ask dialog, where message text and one or more buttons appear in the window. A modal dialog appears in the following illustration:

You apply a modal dialog event to a window in the same manner as other window events. However, instead of occurring when the window opens, closes or activates, the modal dialog event occurs for the window whenever a modal dialog opens.
BeforeModalDialog event

The VBA BeforeModalDialog event occurs when Microsoft Dynamics GP opens a modal dialog, but before it’s displayed. Since the dialog is open, but not visible, this event allows you to dismiss the dialog by programmatically “answering” it for the user. In the following example, if a user enters a non-existent shipping method in the Customer Maintenance window, Microsoft Dynamics GP displays an ask dialog asking whether the user wants to add the shipping method. Normally the user must dismiss the dialog manually, by clicking Add or Cancel. Instead, the event procedure automatically “answers” this dialog and removes a step in the data entry process:

```vba
Private Sub Window_BeforeModalDialog(ByVal DlgType As Boolean, _
PromptString As String, Control1String As String, Control2String _
As String, Control3String As String, Answer As Long)
  If PromptString = "Do you want add this Shipping Method?" Then
    'Click the first button, the Add button
    Answer = dcButton1
  End If
End Sub
```

The PromptString parameter is the message text, which you can use to filter which modal dialog you want to respond to. The Answer parameter uses the constants dcButton1, dcButton2 and dcButton3. When indicated, these “click” the first, second or third button in the dialog.

The BeforeModalDialog event is also useful for altering the contents of the dialog before it’s displayed, including the modal dialog’s message text and button text. The following event procedure changes the message and button text:

```vba
Private Sub Window_BeforeModalDialog(ByVal DlgType As Boolean, _
PromptString As String, Control1String As String, Control2String _
As String, Control3String As String, Answer As Long)
  If PromptString = "Do you want to add this shipping method?" Then
    'Change the message text
    PromptString = "Create this shipping method?"
    'Change "Add" button to "Create"
    Control1String = "Create"
  End If
End Sub
```

AfterModalDialog event

The VBA AfterModalDialog event occurs when Microsoft Dynamics GP opens a modal dialog, but after the user dismisses it. This event allows you to ascertain how the user responded to the dialog and perform any additional tasks based on that response.

The primary use for the AfterModalDialog event is to keep data in the accounting system synchronized with data you’ve stored using the Dynamic User Object Store (DUOS). For instance, if you’ve created a DUOS object that stores Internet address information for a customer, you’ll likely want to save it when the user saves the corresponding customer record.

In the following example, the user attempts to close a window without saving a customer record. In this case, Microsoft Dynamics GP will display a modal dialog asking if the user wants to save the record. If the user clicks the modal dialog’s Save button, the event procedure saves the corresponding DUOS object:

Refer to Chapter 6, “Storing Additional Data” for more information about using the Dynamic User Object Store (DUOS).
Private Sub Window_AfterModalDialog(ByVal DlgType As Long, _ PromptString As String, Control1String As String, _ Control2String As String, Control3String As String, Answer As Long)
    Dim Customers As DUOSObjects
    Dim Customer As DUOSObject
    Dim CustomerProperty As DUOSProperty
    If PromptString = "Do you want to save changes to this " + _
        "customer?" Then
        'The user is trying to save the record using the save dialog.
        If Answer = dcButton1 Then 'The user clicked Save.
            Set Customers = DUOSObjectsGet("Customers")
            Set Customer = Customers(CustomerID)
            Customer.Properties("URL Address") = URLAddress
            Customer.Properties("Contact E-Mail Address") = _
                ContactEMailAddress
        End If
    End If
End Sub

Using window properties and methods

The following table explains the available window methods and properties. The remainder of this section explains some of the more common methods and properties you’ll use when working with windows, as well as additional ways you can use windows.

<table>
<thead>
<tr>
<th>Property/Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate method</td>
<td>Activates the window.</td>
</tr>
<tr>
<td>Caption property</td>
<td>Specifies the window’s title.</td>
</tr>
<tr>
<td>Close method</td>
<td>Closes the window.</td>
</tr>
<tr>
<td>EventMode property</td>
<td>Specifies whether window events occur for the original or modified version of the window.</td>
</tr>
<tr>
<td>Height property</td>
<td>Specifies the height of the window’s client area (in pixels). The client area is the window less the window’s title bar.</td>
</tr>
<tr>
<td>Hide method</td>
<td>Hides an open window.</td>
</tr>
<tr>
<td>IsLoaded property</td>
<td>Specifies whether the window is open (not necessarily displayed).</td>
</tr>
<tr>
<td>Left property</td>
<td>Specifies the horizontal position (in pixels) of the window.</td>
</tr>
<tr>
<td>Move method</td>
<td>Moves a window to a specified set of coordinates (in pixels).</td>
</tr>
<tr>
<td>Name property</td>
<td>Specifies the internal name VBA uses for the window.</td>
</tr>
<tr>
<td>Open method</td>
<td>Opens a window.</td>
</tr>
<tr>
<td>PullFocus method</td>
<td>Removes the focus from the window.</td>
</tr>
<tr>
<td>Show method</td>
<td>Shows a window that’s hidden using the Hide method.</td>
</tr>
<tr>
<td>Top property</td>
<td>Specifies the vertical position of the window (in pixels).</td>
</tr>
<tr>
<td>Visible property</td>
<td>Specifies whether the window is visible. Width property</td>
</tr>
<tr>
<td>Required property</td>
<td>Specifies whether the user entered data in all fields whose Required property is True.</td>
</tr>
</tbody>
</table>
Opening and closing windows

You can open a Microsoft Dynamics GP window directly or indirectly through VBA. To open and close it directly, use the window object’s **Open method** and **Close method**. The following event procedure opens the Customer Maintenance window after the user logs in:

```vba
Private Sub Window_AfterClose()
    'The user logged in. Open the Customer Maintenance window.
    CustomerMaintenance.Open
End Sub
```

The recommended method is to open a window indirectly, using VBA to provide the navigation that displays the window, such as programmatically “clicking” a lookup button that displays a lookup window:

```vba
Private Sub ShippingMethod_AfterGotFocus()
    If ShippingMethod.Empty = True Then
        'No shipping method specified. Click the lookup button
        LookupButtons = 1
    End If
End Sub
```

Microsoft Dynamics GP application code attached to navigational controls (such as the lookup button) prepares the window to display data correctly. “Clicking” these controls using VBA ensures that this processing occurs.

Working with forms

In Microsoft Dynamics GP, forms are a logical grouping of windows that function together to perform a specific task. For instance, the Customer Maintenance, Customer Account Maintenance and Customer Maintenance Options windows are all part of the same form.

When you open a window, Microsoft Dynamics GP opens all windows in the window’s form invisibly, then displays the first window in the form (the parent window). The child windows in the form remain invisible, but loaded (their **IsLoaded property** is True).

If you open a form’s child window through VBA, Microsoft Dynamics GP displays the form’s parent window as well as the child window. Any other child windows remain loaded and invisible.

When you display a record in the parent window, Microsoft Dynamics GP updates data in the invisible child windows so each contains data specific to the current record. Since these child windows are loaded, you can reference data in these fields using VBA.

For instance, when you open the Customer Maintenance window and display a customer record, you can reference fields in the Customer Maintenance Options window for that customer. In the following example, the event procedure stops the user from saving a customer record in the Customer Maintenance window if the user hasn’t entered a currency ID in the Customer Maintenance Options window:
Private Sub Save_BeforeUserChanged(KeepFocus As Boolean, _
CancelLogic As Boolean)
    If CustomerMaintenanceOptions.CurrencyID.Empty = True Then
        'Cancel the save
        CancelLogic = True
        'Prompt the user to enter a currency ID
        MsgBox "You must enter a currency ID for this customer."
        CustomerMaintenanceOptions.Visible = True
        CustomerMaintenanceOptions.CurrencyID.Focus
    End If
End Sub

Use the Modifier’s Form Definition window to find out which windows are part of a form.

**Activating a window**
The VBA `Activate` method activates a visible, open window, making it the frontmost window, or expands it if it’s minimized. The `Activate` method also causes the window’s activate event to occur. In the following example, the event procedure activates the Invoice Entry window after closing the Invoice Batch Entry window:

```vba
Private Sub Window_AfterClose()
    If InvoiceEntry.Visible = True Then
        InvoiceEntry.Activate
    End If
End Sub
```

**Hiding windows**
The VBA `Hide` method hides a window you’ve opened, making it invisible. The `Visible` property, when set to False, also makes a window invisible. While invisible, the window is open (its `IsLoaded` property is True) and data in the window is accessible.

Making a window invisible is useful if you need to reference data contained in the window without necessarily displaying the window to the user. For example, when the user enters a transaction amount in the Receivables Transaction Entry window, the following event procedure compares the amount with the maximum batch total in an invisible Receivables Batch Entry window:

```vba
Private Sub SalesAmount_BeforeUserChanged(KeepFocus As Boolean, _
CancelLogic As Boolean)
    If BatchID.Empty = False Then
        'The user selected a batch
        'Click the expansion button to open the Batch Entry window
        ExpansionButtons = 1
        'Make the window invisible
        ReceivablesBatchEntry.Visible = False
        'Compare the batch total to the trx amount entered
        If CCur(SalesAmount) > CCur(ReceivablesBatchEntry.__.BatchTotal) Then
            MsgBox "Amount exceeds batch limit. Select another batch."
            'Clear the batch ID field and move the focus there
            BatchID.Empty = True
            BatchID.Focus
        End If
        'Close the window
        ReceivablesBatchEntry.Close
    End If
End Sub
```
The **Show method** displays an invisible window. Setting the window’s **Visible property** to True also displays an open window. You can open a window invisibly by setting the **OpenVisible** parameter of the window’s BeforeOpen event to False.

**Moving and resizing a window**

You can resize or reposition a window using the **Height property**, **Left property**, **Top property** and **Width property**. Resizing and repositioning a window is useful for organizing windows more efficiently within the visible workspace. In the following example, the event procedure runs before the Customer AddressMaintenance window opens. The event procedure verifies the Customer Maintenance window is open, then positions the Customer Address Maintenance window below and to the right:

```vba
Private Sub Window_BeforeOpen(OpenVisible As Boolean)
If CustomerMaintenance.Visible = True Then
    'The Customer Maintenance window is open
    CustomerAddressMaintenance.Left = CustomerMaintenance.Left + 25
    CustomerAddressMaintenance.Top = CustomerMaintenance.Top + 100
End If
End Sub
```

**Changing a window title**

The **Caption property** allows you to set the window’s caption (title). The following example changes the title of the CustomerMaintenance window when the window opens. It also changes the prompt for one of the fields in the window.

```vba
Private Sub Window_BeforeOpen(OpenVisible As Boolean)
    'Change the title of the window
    CustomerMaintenance.Caption = "Client Maintenance"
    CustomerMaintenance.CustomerID.Caption = "Client ID"
End Sub
```

**Using modified or original windows**

You can choose whether window events occur for the modified or original version of a window using the **EventMode property**. If you modify a window using the Modifier, you can change the window’s **EventMode property** to **emModifiedOnly**. This allows VBA events to occur for the modified window only.

If you make no modifications to the window using the Modifier, you can set the **EventMode property** to **emOriginalOnly**. This allows VBA events to occur for the original window only.

**Renaming a window**

The **Name property** allows you to change VBA’s internal reference to a window object. Note that this is not the same as the **Caption property**, which allows you to change the window’s title. If you change the window’s **Name property** using the Visual Basic Properties window, the name will change in your project. You should then recompile your project to replace any references to that window with the new name. You’ll find it necessary to rename a window if its name conflicts with other object names in your project, such as reserved words in VBA.
Chapter 3: Programming Window Fields

Programming window fields using VBA involves adding a window or a grid to your project, then adding selected field objects from that window or grid. Programming window fields also involves determining which window field events you’ll use to execute VBA code, and using window field methods and properties to customize field display characteristics, or to change business logic in a window. Information about programming window fields is divided into the following sections:

- Working with window fields in your VBA project
- Programming a window field object
- Using window field events
- Got focus events
- User changed events
- Changed event
- Lost focus events
- Using window field properties and methods
- Working with field values
- Verifying field values
- Window field type reference

Working with window fields in your VBA project

To use window fields with VBA, you’ll first need to add a window and its window fields to your project.

Adding window fields to your project

To add window fields, switch to Microsoft Dynamics GP and display the window. On the Tools menu, point to Customize, then choose Add Window To Visual Basic. This adds the active (topmost) window to your project. With the window displayed, point to Customize on the Tools menu and choose Add Field To Visual Basic. This will activate field selection mode, and a selection pointer will appear. The following illustration shows the selection pointer:

```
Position the selection pointer on the field (not the prompt) and click.
```

Using the mouse, click on each data entry field, push button or list field you’ll use in your project. Be sure to click on the field itself, and not the field’s prompt. When you’ve finished adding fields, point to Customize on the Tools menu, and choose Add Fields to Visual Basic to deactivate field selection mode.

Field selection mode also deactivates when you switch to the Visual Basic Editor.

Window fields you’ve added to a project won’t appear in the Visual Basic Project Explorer like a window, report or grid object. Instead, they will appear in the Visual Basic Code window for a window object.
The following illustration shows window field objects for the Receivables Customer Maintenance window object:

At this point, you can write VBA code for the window field object using a window field event, or you can reference the window field from other VBA procedures in your project.

**Removing window fields from your project**

You cannot remove individual field objects from your project, only window objects. When you remove a window object from your project, VBA also removes the all window field objects associated with the window.

To remove a window object from your project, switch to Microsoft Dynamics GP and display the window you want to remove. On the Tools menu, point to Customize, then choose Remove Current Window From Visual Basic. VBA will remove the window and any window fields associated with it.

**Programming a window field object**

There are two primary ways you can write VBA code for a window field object: using a window field event procedure, and referencing the window field in procedures throughout your project.

**Using window field event procedures**

A window field event lets you execute VBA code when the user moves to, changes or moves out of a specific field. This is useful when you want to perform actions based on entries made in the field. The following section, *Using window field events*, explains window field events in more detail.

**Referencing the window field**

To reference a window’s fields, the field must be an object in your project, and the field’s window must be open (the window’s **IsLoaded** property is True).

If the field you’re referencing is not in the current window, you must reference the field using a qualified field name. A qualified name explicitly specifies the location of the field you’re referencing. In the following example, an event procedure for a VBA user form opens the Invoice Batch Entry window, then clicks a lookup button using the lookup button field’s qualified name:

```vba
Private Sub CommandButton1_Click()
    'Open the batch entry window
    InvoiceBatchEntry.Open
    'Click the lookup button to display the lookup
    InvoiceBatchEntry.LookupButton1 = 1
    UserForm1.Hide
End Sub
```
Using window field events

Window field events execute VBA procedures when a user moves to, changes, or moves out of a window field, or a field you’ve added using the Modifier. To view field events, choose a field object in the Code window’s Object list; all the window field events will appear in the Procedure list.

When you choose an event, VBA automatically adds the event procedure syntax to the Code window. You can then write the event procedure using that event syntax.

There are four basic kinds of field events you’ll work with in VBA: got focus, user changed, changed and lost focus. Each event occurs relative to when the user moves the focus into a field (a got focus event), when the value of a field changes (a user changed or changed event) or when the user moves the focus out of a field (a lost focus event).

These events are useful when you want to respond to data the user enters in a field by performing other operations in the window. For example, the following AfterUserChanged event procedure runs when the user changes information in the State field:

```vba
Private Sub State_AfterUserChanged()
    If State = "NJ" Then
        'Default the salesperson and territory
        SalespersonID = "MARILYN H."
        TerritoryID = "TERRITORY 4"
    End If
End Sub
```

Field events also respond to actions performed by the user, such as clicking a button. In the following example, a new button (OpenMSWord) added to a window using the Modifier uses the AfterUserChanged event to open Microsoft Word:

```vba
Private Sub OpenMSWord_AfterUserChanged()
    Dim App As Word.Application
    Set App = CreateObject("Word.Application")
    App.Visible = True
End Sub
```

Each window field event you define occurs either before or after the Microsoft Dynamics GP code for the field event runs. Therefore, field events have names like “BeforeUserChanged” to indicate that the event runs when the user changed the contents, but before the Microsoft Dynamics GP application code runs.
Got focus events

VBA got focus events occur when the user initially enters a field, using the TAB key or the mouse. You can use two VBA events, BeforeGotFocus and AfterGotFocus, to execute VBA event procedures before and after the code for the got focus event runs.

The BeforeGotFocus, AfterGotFocus and BeforeUserChanged events are the only three window field events where VBA allows you to set the value of the current field (the field whose events are running).

The following illustration shows the progression of got focus events:

If Microsoft Dynamics GP runs application code during the got focus event, it typically checks the value of the field that’s gaining focus, and performs calculations or sets other field values based on that value. However, Microsoft Dynamics GP uses this event infrequently to do such operations.

BeforeGotFocus event

The BeforeGotFocus event occurs before the Microsoft Dynamics GP code for the got focus event. Use this event to set or evaluate the contents of the field gaining focus, and cancel any further got focus events from running (the Microsoft Dynamics GP got focus event and the VBA AfterGotFocus event). When set to True, the CancelLogic parameter will cancel the field’s other got focus events. In this example, the BeforeGotFocus event procedure cancels the AfterGotFocus event for the Batch ID field (shown in the description of the AfterGotFocus event below):

```vba
Private Sub BatchID_BeforeGotFocus(CancelLogic As Boolean)
    If DocumentType = 7 Then
        'The document type is a return. Don’t use a batch
        BatchID.Empty = True
        'Cancel the AfterGotFocus event, which opens the lookup
        CancelLogic = True
        MsgBox “Post returns individually, not in a batch.”
        DocumentDate.Focus
    End If
End Sub
```
CHAPTER 3  PROGRAMMING WINDOW FIELDS

AfterGotFocus
The AfterGotFocus event occurs after the Microsoft Dynamics GP code runs for the got focus event. Use this event to set or evaluate the contents of a field gaining focus. In the following example, the AfterGotFocus event for the Batch ID field checks whether the field is empty when the user moves to it. If it is, the event procedure opens the batch lookup window:

Private Sub BatchID_AfterGotFocus()
    If BatchID.Empty = True Then
        'The field is empty. Click the lookup button
        LookupButton3 = 1
    End If
End Sub

User changed events
VBA user changed events occur when the user changes the contents of a field, then moves the focus out of the field, using the TAB key or the mouse. In fields that toggle, such as push buttons and visual switches, the user changed event occurs when the user clicks the field. In list fields, the user changed event occurs when the user selects an item in the list.

You can use two VBA events, BeforeUserChanged and AfterUserChanged, to execute VBA events either before or after the Microsoft Dynamics GP code for the user changed event runs.

The BeforeGotFocus, AfterGotFocus and BeforeUserChanged events are the only three window field events where VBA allows you to set the value of the current field (the field whose events are running).

The following illustration shows the progression of user changed events in an editable field:

User enters information in the Name field and presses the TAB key:
1. VBA BeforeUserChanged event occurs for the Name field.
2. Microsoft Dynamics GP user changed event occurs for the Name field.
3. VBA AfterUserChanged event occurs for the Name field.

It’s common for Microsoft Dynamics GP to use the user changed event to verify the contents of a field when the field changes; however, this isn’t done for all fields.

In fields that toggle, such as push buttons and visual switches, the user changed event occurs when the user clicks the field. Microsoft Dynamics GP always executes application code during the user changed event for these fields.

User clicks the Save button:
1. VBA BeforeUserChanged event occurs for the button.
2. Microsoft Dynamics GP user changed event occurs for the button.
3. VBA AfterUserChanged event occurs for the button.
In list fields, such as button drop lists and list boxes, user changed events occur when the user selects an item from the list, or presses the up arrow or down arrow keys to move to a different list item.

User selects an item in the list:
1. VBA BeforeUserChanged event occurs for the list.
2. Microsoft Dynamics GP user changed event occurs for the list.
3. VBA AfterUserChanged event occurs for the list.

**BeforeUserChanged**

The BeforeUserChanged event occurs before the Microsoft Dynamics GP code runs for the user changed event. Use this event to evaluate the value of the field and bypass, or cancel, any Microsoft Dynamics GP processing that occurs for the same field.

Use caution when canceling Microsoft Dynamics GP processing for a field’s user changed event. Canceling the user changed event processing inhibits the accounting system’s ability to verify the contents of a field.

In the following example, the BeforeUserChanged event procedure runs for the Invoice Entry window’s Trade Discount field. Using the CancelLogic parameter, it cancels processing for the Trade Discount field if the user enters an amount greater than 4% of the invoice subtotal. It also uses the KeepFocus parameter to place the focus in the same field, allowing the user to enter a lower amount:

```vba
Private Sub TradeDiscount_BeforeUserChanged(KeepFocus As Boolean, CancelLogic As Boolean)
    If TradeDiscount > Subtotal * 0.04 Then
        'The trade discount is greater than 4% of the subtotal
        'Cancel the invoice calculation
        CancelLogic = True
        TradeDiscount.Value = Subtotal * 0.04
        KeepFocus = True
        MsgBox "You cannot enter a discount greater than 4% of " + _
            "the subtotal."
    End If
End Sub
```

**AfterUserChanged**

The AfterUserChanged event occurs after the Microsoft Dynamics GP code for the user changed event runs. Use this event to evaluate information the user entered in a field. The following example uses the AfterUserChanged event to check the value of the Customer ID field in the Invoice Entry window, then disable the Trade Discount field:

```vba
Private Sub CustomerID_AfterUserChanged()
    If CustomerID = "ADVANCED0002" Then
        'Don't offer a trade discount
        TradeDiscount.Enable = False
    End If
End Sub
```
**Changed event**

The VBA *Changed* field event *always* occurs for a field when its contents change. This includes:

- When the user changes the field directly
- When Microsoft Dynamics GP application code updates the window field
- When a field’s value changes using an external tool, such as VBA

The most common situation to use the *Changed* event is when you want to execute VBA code for window fields that Microsoft Dynamics GP updates. This occurs most often when the accounting system retrieves a record from a table, then displays the record’s contents in window fields. The *AfterUserChanged* and *BeforeUserChanged* events won’t occur for these fields, since the application, not the user, changes the contents of the field.

*Exercise caution when using the Changed event.* In many cases, Microsoft Dynamics GP may cause the *Changed* event to occur numerous times for the same field even though the value of the field may not visibly change. This will execute the associated VBA event procedure each time the event occurs.

For example, as the user browses through records in the Receivables Transaction Entry window, Microsoft Dynamics GP sets window field values for each record. The following *Changed* event procedure runs for the Document Date field. Each time the user moves to a new record, it checks the value of the document date, and displays a message if the document is older than 30 days:

```vba
Private Sub DocumentDate_Changed()
    Dim DaysOld As Integer
    'Check the document date using the VBA Date function
    If CDate(DocumentDate) < Date - 30 Then
        'The document is at least 30 days old
        DaysOld = Date - CDate(DocumentDate)
        MsgBox "This document is " + Str(DaysOld) + " days old." + _
        "Please post."
    End If
End Sub
```

**Lost focus events**

VBA lost focus events occur when the user exits a field, using the TAB key or the mouse, regardless of whether the field’s contents changed. You can use two VBA events, *BeforeLostFocus* and *AfterLostFocus*, to execute VBA event procedures before and after the Microsoft Dynamics GP code for the lost focus event runs.

The following illustration shows the progression of lost focus events:

*User exits the Name field:*
1. VBA *BeforeLostFocus* event occurs for the Name field.
2. Microsoft Dynamics GP lost focus event occurs.
3. VBA *AfterLostFocus* event occurs.
Microsoft Dynamics GP rarely uses the lost focus event. When it is used, it typically locks control fields (fields that control the display of a record).

**BeforeLostFocus**
The **BeforeLostFocus** event occurs before the Microsoft Dynamics GP code for the lost focus event. You can use the **BeforeLostFocus** event to cancel any subsequent lost focus events for the field. When set to True, the **CancelLogic** parameter cancels the Microsoft Dynamics GP lost focus and VBA **AfterLostFocus** events.

The following **BeforeLostFocus** event procedure runs for the Document Number field in the Receivables Transaction Entry window. This is a control field, and normally the lost focus event locks this field. In this case, the procedure cancels all other lost focus events, and the Document Number field remains unlocked and editable:

```vba
Private Sub Number_BeforeLostFocus(KeepFocus As Boolean, _
    CancelLogic As Boolean)
    CancelLogic = True
    Description.Focus = True
End Sub
```

When set to True, the **KeepFocus** parameter allows you keep the focus in the current field.

**AfterLostFocus**
The **AfterLostFocus** event occurs after the Microsoft Dynamics GP code runs for the lost focus event. You can use the **AfterLostFocus** event to evaluate the value of the field losing focus, and perform any additional operations in the window. The following **AfterLostFocus** event procedure checks whether the Payment Terms field is empty. If it is, the procedure displays a VBA dialog that asks the if the user wants to enter a payment term. If the user clicks Yes, the procedure opens the Payment Terms Lookup window:

```vba
Private Sub PaymentTerms_AfterLostFocus()
    Dim Response As Integer
    If PaymentTerms.Empty = True Then
        'Set a default payment term
        Response = MsgBox("Do you want to enter a payment term?", _
            vbYesNo)
        If Response = vbYes Then
            LookupButtons = 1
        End If
    End If
End Sub
```
Using window field properties and methods

The following table explains the available methods and properties for window fields. The remainder of this section explains some of the more common methods and properties you’ll use when working with window fields.

<table>
<thead>
<tr>
<th>Property/Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Caption property</td>
<td>Specifies the field’s caption (prompt).</td>
</tr>
<tr>
<td>Empty property</td>
<td>Specifies whether the field is empty.</td>
</tr>
<tr>
<td>Enabled property</td>
<td>Specifies whether the field is enabled.</td>
</tr>
<tr>
<td>Focus method</td>
<td>Moves the focus to a field.</td>
</tr>
<tr>
<td>FocusSeg method</td>
<td>Moves the focus to a segment in a composite.</td>
</tr>
<tr>
<td>Height property</td>
<td>Specifies the height of the field (in pixels).</td>
</tr>
<tr>
<td>Left property</td>
<td>Specifies the horizontal position of a field (in pixels).</td>
</tr>
<tr>
<td>Locked property</td>
<td>Specifies whether the current field is locked.</td>
</tr>
<tr>
<td>Move method</td>
<td>Moves the field to a specified set of coordinates within the window’s client area.</td>
</tr>
<tr>
<td>Name property</td>
<td>Specifies the internal name VBA uses to reference the field.</td>
</tr>
<tr>
<td>Object property</td>
<td>Returns a field object without the standard properties and methods extended to it by Visual Basic.</td>
</tr>
<tr>
<td>Parent property</td>
<td>Returns a window or report object containing a specified field object.</td>
</tr>
<tr>
<td>Required property</td>
<td>Specifies whether the field requires data for the window to save the record properly.</td>
</tr>
<tr>
<td>TabStop property</td>
<td>Specifies whether the field is in the window’s tab sequence.</td>
</tr>
<tr>
<td>Top property</td>
<td>Specifies the vertical position of a field (in pixels).</td>
</tr>
<tr>
<td>Value property</td>
<td>Specifies the value of a field.</td>
</tr>
<tr>
<td>ValueSeg method</td>
<td>Specifies the value of a segment in a composite field.</td>
</tr>
<tr>
<td>Visible property</td>
<td>Specifies whether the field is visible.</td>
</tr>
<tr>
<td>Width property</td>
<td>Specifies the width (in pixels) of the field.</td>
</tr>
</tbody>
</table>

Disabling and locking fields

The Enabled property and Locked property allow you to make a field read-only and inaccessible to the user. To disable a field, set the Enabled property to False. A disabled field will appear dimmed, and any related controls (such as lookup or expansion buttons) will be inaccessible. To lock a field, set its Locked property to True. A locked field will appear with a gray background, and its related controls will still be accessible.

In the following example, an event procedure for the Account Maintenance window disables the Delete button and locks the Account Number field:

```vba
Private Sub Window_AfterOpen()
    If UserInfoGet.UserID = "LESSONUSER1" Then
        Delete.Enabled = False 'Disable the Delete button
        Account.Locked = True 'Lock the account number
    End If
End Sub
```

In this case, the user can browse information in the window, but not create new account records or delete existing ones.
You can use VBA to set the value of a field you’ve locked or disabled using VBA. However, you cannot set the value of a field locked or disabled by Microsoft Dynamics GP, nor can you unlock or enable these fields. Microsoft Dynamics GP disables fields when they don’t apply in a given situation, and lock fields to preserve the integrity of accounting data (such as document totals).

**Controlling the focus**
The **Focus method** and **FocusSeg method** let you control the movement of the focus in a window.

The **Focus method** moves the focus to a specified field. This will also activate the field’s **BeforeGotFocus** and **AfterGotFocus** events. In the following example, the **Focus method** moves the focus to the Check field if the current customer uses Check as the payment term:

```vba
Private Sub Tax_AfterLostFocus()
    If PaymentTerms = "Check" Then
        'The customer pays with checks. Move to the Check field
        Check.Focus
    End If
End Sub
```

The **FocusSeg method** moves the focus between segments in a composite. This method uses an index corresponding to the segment you want the focus to move to. The following example uses both the **ValueSeg property** and **FocusSeg method** to set a default value for an account number field:

```vba
Private Sub CashAccount_AfterGotFocus()
    If CheckbookID = "PAYROLL" Then
        'Default payroll account segments
        CashAccount.ValueSeg(1) = "100"
        CashAccount.ValueSeg(2) = "1500"
        'Move the focus to the third segment
        CashAccount.FocusSeg(3)
    End If
End Sub
```

When working with windows, you cannot move the focus using the window’s **BeforeOpen** event. Since the window isn’t open yet, there’s no place for the focus to be placed. You can, however, move the focus using the window’s **AfterOpen** event.

**Showing and hiding fields**
The **Visible property** allows you to show and hide fields in a window. To hide a field, set the **Visible property** to False; the field and its caption will still be in the window, but won’t be visible to the user.

The following example hides sales fields when a given user opens the Salesperson Maintenance window:

```vba
Private Sub Window_AfterOpen()
    If UserInfoGet.UserID = "LESSONUSER1" Then
        CommissionedSales.Visible = False
        CostofSales.Visible = False
        NonCommissionedSales.Visible = False
        TotalCommissions.Visible = False
    End If
End Sub
```
You can still return or set the values of fields you make invisible with VBA. However, you cannot set the value of fields hidden by Microsoft Dynamics GP, nor can you show these fields.

### Renaming fields

The `Name property` allows you to change the name you use in VBA code to reference the field. Note that this is not the same as the `Caption property`, which allows you to change the field’s caption. You can change a field’s name using the Visual Basic Properties window, and the name will change in your project. You should then recompile your project to replace any references to that field with the new name.

You’ll find it necessary to rename fields if the field’s name conflicts with a reserved word in VBA. For example, the Date field has the same name as a reserved word in VBA. You may also want to rename fields to make code more readable.

### Making fields required

The `Required property` specifies whether the field must contain data before the user can save information in the window. To make a field required, set its `Required` property to True. The following example sets fields in the Customer Maintenance window to required:

```vba
Private Sub Window_BeforeOpen(OpenVisible As Boolean)
    Name.Required = True
    ShortName.Required = False
    StatementName.Required = True
End Sub
```

Microsoft Dynamics GP will prompt the user to enter data in a required field if the field is empty when the user attempts to save the record. Fields marked as required by VBA are displayed with the same caption styles as fields marked as required by the accounting system.

You cannot set the `Required property` to False for fields already marked as required by Microsoft Dynamics GP. These are fields necessary for the accounting system to store the record properly.

### Working with field values

You return and set window field values using the `Value property`. If your code specifies the field name, but omits the `Value property`, VBA assumes you’re returning or setting the value of the field. The following example explicitly uses the `Value property`:

```vba
BatchID.Value = "DEBITMEMOS"
```

The same code can also omit the `Value property` and have the same effect:

```vba
BatchID = "DEBITMEMOS"
```

The majority of the examples in this manual use the abbreviated form of this syntax. You can choose to use or ignore the `Value property` in your own project.
Using the ValueSeg property

The ValueSeg property specifies the value of a given segment in a composite field (typically, an account number field). This is useful when you want to evaluate only a single segment of the composite. The Value property can evaluate the value of the entire composite, but not a single segment.

In the following example, if “600” is entered as the first segment in the Cash Account field, the event procedure locks the maximum amount for checks with this account:

```vba
Private Sub CashAccount_AfterLostFocus()
    If CashAccount.ValueSeg(1) = "600" Then
        'This is an expense account
        'Set the maximum payables amount for this account
        Amount = "$1000.00"
        Amount.Locked = True
    End If
End Sub
```

You can also use the ValueSeg property to set individual segments in a composite. However, there’s little difference in using the Value and ValueSeg properties to do this. In the following example, the ValueSeg property is used to set the first and second segments in the Account field.

```vba
Private Sub Account_AfterGotFocus()
    CashAccount.ValueSeg(1) = "100"
    'Set the second segment
    CashAccount.ValueSeg(2) = "1100"
    'Move the focus to the third segment
    CashAccount.FocusSeg(3)
End Sub
```

The following event procedure has the same effect, but uses the Value property to set the composite’s value:

```vba
Private Sub Account_AfterGotFocus()
    CashAccount.Value = "100-1100"
    CashAccount.FocusSeg (3)
End Sub
```

Using the Empty property

The Empty property specifies whether a field’s value is empty. If you set a field’s Empty property to True, VBA clears the field. Using the Empty property in this manner is the same as setting the field’s Value property to a null value (“”). The following example uses the Empty property to automatically open a lookup window if the Salesperson ID field is empty:

```vba
Private Sub SalespersonID_AfterGotFocus()
    If SalespersonID.Empty = True Then
        'Click the lookup button to display the lookup window
        LookupButton8 = 1
    End If
End Sub
```
Returning field values
You can use the Value property to return window field’s value, providing the window that contains the field is open (the window’s IsLoaded property is True). If the window is open but invisible, you can return field values from that window. If you’ve disabled, locked or hidden the field using VBA, you can still return the field’s value. If Microsoft Dynamics GP hides the field, you cannot return the field’s value.

VBA returns all field values as strings. For instance, VBA returns the values of a currency field as “$30.75”, a date field as “5/1/00”, and an integer field as “100”.

Performing numeric calculations
Since all field values returned to VBA are strings, numeric calculations or comparisons won’t function properly unless you convert string values to the appropriate numeric data type first.

For example, if you compare two integer fields, Quantity1 (value of 70), and Quantity2 (value of 100), VBA compares the string values of those fields, not their integer values. VBA uses alphanumeric precedence in comparing strings, and compares the “7” in “70” to the “1” in “100”. Since 7 is greater than 1, the expression results in “70” being the larger value, which is incorrect:

If Quantity1 > Quantity2 Then
  'Quantity1 (70) in this case is greater than Quantity2 (100)

To avoid this problem, you must convert these values using VBA’s numeric conversion functions prior to performing the comparison. In the following example, VBA’s CInt() function converts the ItemQuantity1 and ItemQuantity2 fields, resulting in a correct comparison:

If CInt(Quantity1) > CInt(Quantity2) Then
  'Quantity1 (70) is now less than Quantity2 (100)

Likewise, when performing a calculation using a Microsoft Dynamics GP value and a VBA value, you must convert the value from the accounting system before performing the calculation. For example, to calculate the number of days between a Microsoft Dynamics GP date value (Document Date) and the current system date returned using the VBA Date function, convert the Microsoft Dynamics GP date using the CDate() function:

Private Sub DocumentDate_Changed()
  Dim DaysOld As Integer
  If CDate(DocumentDate) < Date - 30 Then
    'The document is at least 30 days old
    DaysOld = Date - CDate(DocumentDate)
    MsgBox "This document is " + Str(DaysOld) + " days old." + _
    'Please post."
  End If
End Sub
The following table shows the VBA functions available for converting string values to a specific data type:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CInt()</td>
<td>Converts a Microsoft Dynamics GP string value to an integer.</td>
</tr>
<tr>
<td>CDate()</td>
<td>Converts a Microsoft Dynamics GP string value to a date.</td>
</tr>
<tr>
<td>CCur()</td>
<td>Converts a Microsoft Dynamics GP string value to a currency.</td>
</tr>
<tr>
<td>CLng()</td>
<td>Converts a Microsoft Dynamics GP string value to a long integer.</td>
</tr>
</tbody>
</table>

### Setting field values

You can use the **Value property** to set a field’s value for any field in a Microsoft Dynamics GP window, providing the window that contains the field is open (the window’s **IsLoaded** property is True). If the window is open but invisible, you can set field values from that window. If you’ve disabled, locked or hidden the field using VBA, you can still set the field’s value. If the accounting system disables, locks or hides the field, you cannot set the field’s value.

Each type of field accepts specific values from VBA. For setting data entry fields, such as string, date and currency fields, you typically use a string value from VBA:

```vba
' Set a the Batch ID string field
BatchID = "DEBITMEMOS"
' Set the Document Date field
DocumentDate = "02/07/97"
' Set the Discount Amount currency field
DiscountAmount = "120.95"
```

Other fields, such as list fields, check boxes, push buttons and radio buttons, accept numeric values:

```vba
' Set the Document Type drop-down list to item 2
DocumentType = 2
' Mark the Hold check box
Hold = 1
' Push the Save button
Save = 1
' Select the first option in the Posting Type radio button group
PostingType = 0
```

### Verifying field values

Microsoft Dynamics GP verifies data entered in many fields to check whether the data is valid. When you enter data in a field manually, verification takes place in the Microsoft Dynamics GP user changed event (when the user enters the data, then moves out of the field).

When you set the value of a field using VBA, this verification will still occur, but has different results depending on whether you set the field from “inside” the field, or “outside” the field. The following sections describe data verification issues in more detail.
Setting a value from “inside” the field

If you set a value from “inside” a field (after the field gains focus), using the field’s BeforeGotFocus, AfterGotFocus, or BeforeUserChanged events, Microsoft Dynamics GP field verification can perform normally. This is the recommended method for setting a field value, since it most closely matches how the user enters data, and how the accounting system verifies that data. The following illustration shows the progression of events when setting a value from “inside” the field:

Field event order:
1. VBA BeforeGotFocus
2. VBA AfterGotFocus
3. VBA BeforeUserChanged
4. Microsoft Dynamics GP user changed
5. VBA AfterUserChanged
6. VBA Changed
7. VBA BeforeLostFocus
8. VBA AfterLostFocus

Use these events to set the field value.
The accounting system

Use these events to read or reverify the field

In the following example, the AfterGotFocus event procedure for the Shipping Method field sets the value of the Shipping Method field:

Private Sub ShippingMethod_AfterGotFocus()
    If ShippingMethod.Empty = True Then
        ShippingMethod = "NEW"
    End If
End Sub

When the user moves out of the Shipping Method field, the Microsoft Dynamics GP code for the user changed event checks whether the field’s data is valid. In this case, it will make sure “NEW” is an existing shipping method. Since this is a new shipping method, a dialog will appear asking whether the user wants to add it.

You cannot set the field’s value using events that follow the Microsoft Dynamics GP user changed event (AfterUserChanged, AfterLostFocus or BeforeLostFocus events). Microsoft Dynamics GP doesn’t have the opportunity to verify values set in these events, and a VBA error will occur. You can use these events to read the value of the field.

Setting a value from “outside” the field

If you set a field’s value from “outside” the field (before the field gains focus), such as by using the window’s BeforeOpen or AfterOpen event, or another field’s BeforeUserChanged or AfterUserChanged event, VBA automatically runs the Microsoft Dynamics GP user changed event for the field you’re setting. This is necessary so the application code associated with these events can verify the field’s value.

For many fields, Microsoft Dynamics GP does little or no field verification, and setting the field’s value from “outside” the field using VBA will not generate errors. However, Microsoft Dynamics GP will perform verification for fields that affect business logic (such as an invoice discount percent, or a tax amount) or for add-on-the-fly fields. Typically, these are fields that generate Microsoft Dynamics GP alert dialogs when you enter invalid data, indicating that the value entered is incorrect. If the field is an add-on-the-fly field, a dialog typically appears asking whether you want to add a new record.
If you set a field value from “outside” the field using VBA, and Microsoft Dynamics GP determines that its value is invalid, the accounting system will first display its alert dialog. This will be followed by a VBA error dialog, shown in the following illustration.

![VBA Error Dialog](image)

The VBA error occurs because the accounting system attempts to place the focus in the field and restore the field’s previous value. Since the current field does not have focus, Microsoft Dynamics GP cannot place the focus and the VBA error results.

This type of error will commonly occur when you set an add-on-the-fly field to a new value from “outside” the field. The following `AfterOpen` event procedure for the Receivables Transaction Entry window sets the value of an add-on-the-fly field (Shipping Method):

```vba
Private Sub ReceivablesTransactionEntry_AfterOpen()
    'Set the Shipping Method field to a value that doesn’t exist.
    ShippingMethod = "NEW"
End Sub
```

In this case, the accounting system will display a dialog asking whether you want to add the new shipping method. When you dismiss the dialog, VBA generates an error, since the system cannot place the focus in the Shipping Method field.

There are two ways to avoid this type of error:

- Whenever possible, set a field’s value from “inside” the field, after its gained focus, using the field’s `AfterGotFocus` or `BeforeGotFocus` field event. If your event procedure sets an invalid value, the accounting system will still display a dialog, but VBA won’t generate the error:

  ```vba
  Private Sub ShippingMethod_AfterGotFocus()
      'Set the Shipping Method field to a value that doesn’t exist.
      ShippingMethod = "NEW"
  End Sub
  ```

- If you set the value “outside” the field, before the field has gained focus, use the `Focus method` with the `setvalue` parameter in your event procedure. The `Focus method` moves the focus to the field, then sets its value to the `setvalue`:

  ```vba
  Private Sub CustomerID_AfterUserChanged()
      'Move the focus to the field, then set the value
      ShippingMethod.Focus("NEW")
  End Sub
  ```
Window field type reference

This section describes the common window field type in Microsoft Dynamics GP, as well as those you can create using the Modifier.

Finding additional field information
Use the Resource Descriptions tool for locating additional information about a Microsoft Dynamics GP field. To access this tool, switch to Microsoft Dynamics GP window. On the Tools menu, point to Resource Descriptions, then choose Fields. Choose the field you want additional information for, then click Field Info.

This information can help you understand the type of field you’re working with (string, integer, list box) as well as the field’s keyable length, the format applied to the field, and any static values applied to the field.

Field list
The fields described are listed below. Detailed explanations of each follow.

- Button drop list
- Check box
- Combo box
- Composite
- Currency
- Date
- Drop-down list
- Integer
- List box
- Multi-select list box
- Push button
- Radio button
- String
- Text
- Visual switch
### Button drop list

**Description**
A button drop list allows a user to select one item from the list. The following illustration shows a button drop list.

![Button drop list](image)

**Events**
The **BeforeUserChanged** and **AfterUserChanged** events occur when the user selects an item from the list. The **BeforeGotFocus** and **AfterGotFocus** events occur as the user presses the button. The **BeforeLostFocus** and **AfterLostFocus** events occur when the button returns to its “unpressed” state. Setting the value of a button drop list through VBA runs the list’s **Changed** event.

**VBA usage**
VBA uses a numeric value (1 to n) when referencing an item in a button drop list. In most cases, the value corresponds to the order of the selected item in the list. You cannot reference the name of an item in the list, nor can you rearrange, remove or add items to a button drop list using VBA.

**Comments**
In certain instances, the position of an item in a button drop list doesn’t necessarily to its numeric value. This is especially true if the list is sorted in alphanumeric order. To determine the correct value, select the item in the list, then view the field’s **Value property** using the Visual Basic Editor’s Property sheet.

**Example**
The following event procedure chooses the first item in the Write Letters button drop list in the Customer Maintenance window.

```vba
Private Sub PrintCollections_Click()
    'Print a collections letter
    CustomerMaintenance.coWriteLettersBDL = 1
End Sub
```
**Check box**

**Description**
A check box allows the user to mark or unmark an item. The following illustration shows a check box.

![Check box](image)

**Events**
All VBA field events function for a focusable check box field.

**VBA usage**
VBA uses numeric values when referencing a check box field. If 1, the check box is marked; if 0, the check box is unmarked.

**Comments**
You can change a check box caption using the **Caption property**. For two or more check boxes linked to a single caption (such as the Calendar Year and Fiscal Year check boxes linked to the Maintain History caption), you can change only the check box group caption, not the captions for individual check boxes.

**Example**
The following event procedure runs when the Sales Territory Maintenance window opens. It marks the first check box (Calendar Year) in the Maintain History group, and unmarks the second check box (Fiscal Year):

```vba
Private Sub Window_AfterOpen()
    'Mark this check box when the window opens
    MaintainHistory = 1
    'Unmark this check box when the window opens
    MaintainHistory1 = 0
End Sub
```
Combo box

Description
A combo box allows the user to enter a text item or choose one from the combo box list. The following illustration shows a combo box.

Events
The BeforeUserChanged and AfterUserChanged events run each time the user selects an item in the combo box list, or presses the up arrow or down arrow key to move to a different list item. All other VBA field events function normally for a focusable combo box field.

VBA usage
VBA uses string values for each item in a combo box. When you set the value of a combo box, VBA sets the value to the combo box field. If the value doesn’t exist in the item list, VBA won’t add it to the list. However, Microsoft Dynamics GP application code typically asks the user if they want to save a new item in the combo box field when the user dismisses the window.

Example
The following event procedure checks the value of the Budget Year combo box field in the Budget Maintenance window, then enables or disables the Display radio group:

```vba
Private Sub BudgetYear_AfterUserChanged()
    'Check the Budget Year combo box
    If BudgetYear = "2004" Then
        'Disable the Display radio group
        Display.Enabled = False
    Else
        Display.Enabled = True
    End If
End Sub
```
Composite

Description
A composite field displays information from multiple fields called segments. Microsoft Dynamics GP primarily uses composite fields for account numbers. The following illustration shows a composite field.

![Composite field diagram]

Events
The BeforeUserChanged and AfterUserChanged events occur for the composite field when the user changes information in any segment, then moves the focus out of the composite. No events occur when the user changes the value of a segment, then moves to another segment in the same composite. All other VBA field events occur when you move the focus into or out of the composite.

VBA usage
You can return the value of a composite using either the Value property or the ValueSeg property. If you use the Value property, VBA returns a formatted string value. If you use the ValueSeg property, VBA returns the unformatted value of the segment you specify.

You can also set the value of a composite using either the Value property or the ValueSeg property. Use the Value property to set the value of the entire composite, including any formatting characters, such as dashes. Use the ValueSeg property to set the value of individual segments of the composite, without including any formatting. Use the FocusSeg method to move the focus between segments of a composite field.

Examples
The following example uses the Value property to set the Cash Account composite field to an existing account number:

```vba
Private Sub CheckbookID_AfterUserChanged()
    If CheckbookID = "FIRST NATIONAL" And CashAccount.Enabled = True Then
        CashAccount = "100-5100-00"
        CashAccount.Enabled = False
    Else
        CashAccount.Enabled = True
    End If
End Sub
```

The following event procedure uses the ValueSeg property to set the default value of the first segment in the Cash Account composite field. It then uses the FocusSeg method to place the focus on the second segment in the composite:

```vba
Private Sub CashAccount_AfterGotFocus()
    CashAccount.ValueSeg(1) = "100"
    CashAccount.FocusSeg(2)
End Sub
```
## Currency

### Description
A currency field displays a value as a currency amount. The following illustration shows a currency field.

<table>
<thead>
<tr>
<th>Caption</th>
<th>Total Commissions</th>
<th>$4,500.00</th>
</tr>
</thead>
</table>

### Events
All VBA field events function for a focusable currency field.

### VBA usage
VBA uses a formatted string value when referencing a currency field. The value can have a maximum of 14 characters to the left of a decimal separator and 5 to the right.

When you set a currency value, specify the location of the decimal separator; the accounting system automatically provides all other formatting. When you return a currency value, VBA returns the formatted string value that appears in the window, including the currency symbol, thousands separator and the decimal separator.

### Example
The following VBA code defaults the value of a currency field to a fixed amount in the Checkbook Maintenance window:

```vba
Private Sub CheckbookID_AfterUserChanged()
    If NextCheckNumber = 1 Then
        'This is a new checkbook.
        'Set the max check amount to $2,000.00
        Amount = "2000.00"
    End If
End Sub
```
Date

Description
A date field displays a date, in a format specified by your operating system settings for date and time. The following illustration shows a date field.

Events
All VBA field events function for a focusable date field.

VBA usage
VBA uses a formatted string value when referencing a date field. When you set a date value, the text value must match the operating system settings for date and time, such as “12/31/96” (using MM/DD/YY format) or “31-12-96” (using DD-MM-YY format).

Examples
In the following example, VBA code sets the value of a date field to a fixed string value:

```vba
Private Sub Window_BeforeOpen(OpenVisible As Boolean)
    PostingDate = "07/28/97"
End Sub
```

In the following example, VBA code combines a string with a date field to construct a new batch ID in the Invoice Entry window:

```vba
Private Sub DocumentType_AfterGotFocus()
    If DocumentType = 2 Then
        'The user selected returns
        'Place the focus in the Batch ID field and set the value
        BatchID.Focus("RET-" + Str(UserInfoGet.UserDate))
        'Constructs "RET-07/28/97" as the batch ID
    End If
End Sub
```
Drop-down list

Description

A drop-down list field “expands” to show a list of items, then collapses to show only the selected item. The following illustration shows a drop-down list.

Events

The BeforeUserChanged and AfterUserChanged events run each time the user selects an item in the drop-down list, or presses the up arrow or down arrow key to move to a different list item. All other VBA field events function normally for a focusable drop-down list field.

VBA usage

VBA uses a numeric value (1 to n) when referencing items in a drop-down list. In most cases, the value corresponds to the position of the item in the list. You cannot reference the name of the item in the list, nor can you rearrange, remove or add items to a drop-down list using VBA.

Comments

In certain instances, the position of an item in a drop-down list doesn’t necessarily correspond to its numeric value. This is especially true if the list is sorted in alphanumeric order. To determine the correct value, select the item in the list, then view the field’s Value property using the Visual Basic Editor’s Property sheet.

Example

In the following example, an event procedure sets the default position of the sort list and the Document Type drop-down list fields in the Receivables Transaction Entry window:

Private Sub Window_BeforeOpen(OpenVisible As Boolean)
    SortBy = 4  'Sets the sort list to "By Batch ID"
    DocumentType = 7  'Sets the Document Type to "Returns"
End Sub
CHAPTER 3 PROGRAMMING WINDOW FIELDS

Integer

**Description**

An integer field displays a number between -32,768 and 32,767. The following illustration shows an integer field.

<table>
<thead>
<tr>
<th>Caption</th>
<th>Discount Grace Period</th>
<th>30</th>
</tr>
</thead>
</table>

**Events**

All VBA events function for a focusable integer field.

**VBA usage**

VBA uses a string value when referencing an integer field. If the integer field includes a format (such as a percent field), VBA returns the formatted string value, including the decimal separator and percent sign (such as “3.15%”). To set the value of a formatted integer field, only include the decimal separator (“3.15”). The accounting system adds any other formatting characteristics.

**Examples**

In the following example, VBA code sets the Transaction Total integer field to 50 when the user enters a new batch in the Sales Batch Entry window:

```vba
Private Sub BatchID_AfterUserChanged()
    If Origin = 0 Then
        'No batch origin specified. This is a new batch.
        TransactionTotal = "50"
    End If
End Sub
```

This example sets an integer field (Percent) that’s formatted with two decimal places. It calculates a new commission percent based on commissioned sales:

```vba
Private Sub CommissionedSales_BeforeUserChanged(KeepFocus As Boolean, CancelLogic As Boolean)
    If CCur(CommissionedSales) + CCur(NonCommissionedSales) > 200000 Then
        Percent = "$4.50" 'Set the percent to 4.50%
    Else
        Percent = "$3.00" 'Set the percent to 3.00%
    End If
End Sub
```
List box

Description
A list box displays up to 32,767 items in a list, one of which the user can select. The following illustration shows a list box.

Events
The BeforeUserChanged and AfterUserChanged events run each time the user selects an item in the list box, or presses the up arrow or down arrow key to move to a different list item. All other VBA field events function normally for a focusable list box field.

VBA usage
VBA uses a numeric value (1 to n) when referencing list box fields. In most cases, the value corresponds to the position of an item in the list. You cannot reference the name of the item in the list, nor can you rearrange, remove or add items to a list box using VBA.

Comments
In certain instances, the position of an item in a list box doesn’t necessarily correspond to its numeric value. This is especially true if the list is sorted in alphanumeric order. To determine the correct value, select the item in the list, then view the field’s Value property using the Visual Basic Editor’s Property sheet.

Example
The following example sets the default selection in the Currencies list box when the Multicurrency Access Setup window opens:

```vba
Private Sub Window_AfterOpen()
    Currencies = 8
    'Selects Z-US$
End Sub
```
Multi-select list box

Description
A multi-select list box displays up to 32 items in a list, from which the user can choose multiple items. The following illustration shows a multi-select list box.

Events
The `BeforeUserChanged` and `AfterUserChanged` events run each time the user selects an item in the multi-select list box, or presses the up arrow or down arrow keys to move to a different list item. All other VBA field events function normally for a focusable multi-select list box field.

VBA usage
VBA uses a 32-bit numeric value when referencing the items selected in a multi-select list box. The numeric value takes into account both the number of items selected and their position in the list. The following table shows the position of an item in a multi-select list box, the item’s corresponding numeric value, and the calculation that derives the value:

<table>
<thead>
<tr>
<th>List item</th>
<th>Numeric value</th>
<th>Calculated as...</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>$2^0$</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>$2^1$</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>$2^2$</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>$2^3$</td>
</tr>
<tr>
<td>:</td>
<td>:</td>
<td>:</td>
</tr>
<tr>
<td>31</td>
<td>1,073,741,824</td>
<td>$2^{30}$</td>
</tr>
</tbody>
</table>

The value of the list is the sum of the selected items’ numeric values. If you select items 1, 3 and 4, the value of the multi-select list box is 13 (total = $1 + 4 + 8$). An easier way to determine the value of the list is to select multiple items, then view the field’s `Value property` in the Visual Basic Properties window.

Examples
To select a single item in a multi-select list box, set the multi-select list box’s value to the item’s numeric value (noted in the previous table). The following example selects list item 4 by setting the list box’s value to 8:

```vba
Private Sub Window_BeforeOpen(OpenVisible As Boolean)
    'Select Payroll in the list
    IncludeinLookup = 8
End Sub
```

To select multiple items in a multi-select list box, add the list items numeric values together. The following example selects list items 1, 3 and 4 by setting the field to the sum of the items’ integer values ($1 + 4 + 8 = 13$):

```vba
Private Sub Window_BeforeOpen(OpenVisible As Boolean)
    'Select Sales, Purchasing and Payroll
    IncludeinLookup = 13
End Sub
```
## Push button

### Description
A push button performs actions, such as executing Microsoft Dynamics GP application code or VBA code. The following illustration shows a push button.

![Push button](image)

### Events
The **BeforeUserChanged** and **AfterUserChanged** events occur when the user clicks a push button. The **Changed** event occurs when you set a push button using VBA. No other VBA field events occur for push button fields.

### VBA usage
VBA uses numeric values when referencing push buttons. The value indicates the state of the push button. The “up,” or normal state of the button is 0; the “down,” or pushed state is 1. Setting the button’s value to 1 in effect “pushes” the button, and performs the same operation as if the user clicked the button.

### Comments
You can change a text push button’s caption using the **Caption property**.

### Examples
The following example displays a lookup window when the user moves to the Batch ID field and the field is empty. To display the lookup window, the procedure sets the lookup button field to 1:

```vba
Private Sub BatchID_AfterGotFocus()
    If BatchID.Empty = True Then
        'The field is empty. Press the lookup button.
        LookupButton3 = 1
    End If
End Sub
```
Radio button

Description A radio button field appears within a radio button group. You can select only one button in the group. The following illustration shows a radio button group.

Events When you select a radio button in a radio button group, the BeforeUserChanged and AfterUserChanged events occur for the entire group. All other VBA field events occur for the entire radio button group, as you move the focus to and from the group.

VBA usage VBA uses numeric values when referencing a radio button group. The numeric value indicates the selected radio button in the group and corresponds to the order each radio button appears in the tab sequence, starting with 0 for the first field in the sequence. In the Typical Balance radio button group pictured above, the Debit field is 0 and the Credit field is 1.

Comments You can change the radio button group’s caption using the Caption property. You cannot change the captions for individual radio buttons.

Example The following example selects the Code radio button control in the Method Of Entry radio button group when the window opens:

```vba
Private Sub Window_BeforeOpen
    'Set the Code radio button
    MethodOfEntry = 1
End Sub
```
String

Description
A string field allows you to enter and view up to 255 text characters. The following illustration shows a string field.

![String field](image)

Events
All VBA field events function for a focusable string field.

VBA usage
VBA uses a string value when referencing string fields. If you set the value of a string field that uses a format (such as a phone number field), include all formatting characters, including parentheses, static text, dashes and spaces. When you return the value of a formatted string value, VBA returns the formatted string value.

Example
The following example sets the value of the Batch ID string field when the user chooses a document type from the Document Type drop-down list in the Receivables Transaction Entry window:

```vba
Private Sub DocumentType_AfterUserChanged()
    Select Case DocumentType
        'Move the focus to the Batch ID field and set the value
        Case 1
            BatchID.Focus("DAILYSLS")
        Case 7
            BatchID.Focus("DAILYRET")
        Case Else
            BatchID.Focus("DAILYMISC")
    End Select
End Sub
```

In the following example, the procedure sets the value of the Phone 1 field using a formatted string:

```vba
Private Sub CustomerID_Changed()
    If CustomerID = "AARONFIT0001" Then
        Phone1 = "(701) 555-7890 Ext. 1234"
    End If
End Sub
```
Text

Description
A text field displays up to 32,000 text characters. The following illustration shows a text field.

Events
All VBA field events function for a focusable text field.

VBA usage
VBA uses a string value when referencing a text field. The string value cannot exceed 32,000 characters.

Comments
Microsoft Dynamics GP uses text fields primarily in Note windows.

Example
The following example adds a note to an Invoice record when the customer purchases over $2,000. The message added to the text field in the Note window explains that the customer received a 4% trade discount:

```vba
Private Sub TradeDiscount_AfterGotFocus()
    If Subtotal >= 2000 And TradeDiscount = 0 Then
        'Click the note button to open the Note window
        NoteAbsentButtonWindowArea = 2
        'Add a message to the text field
        Note.TextField = "Customer " + Name1 + " purchased " + _
        'over $2,000. The trade discount for Customer " + Name1 + _
        'was calculated and applied at 4%.
        'Click the Attach button to save the note
        Note.Attach = 2
        'Set the trade discount to 4% of the subtotal
        InvoiceEntry.TradeDiscount = Subtotal * 0.04
    End If
End Sub
```
Visual switch

Description
A visual switch displays a series of text or picture items. Clicking the field displays the next item in the series. The following illustration shows a visual switch field.

Events
The BeforeUserChanged and AfterUserChanged events run each time the user selects an item a visual switch. All other VBA field events function normally for a focusable visual switch field.

VBA usage
VBA uses numeric values when referencing visual switch fields. The numeric value corresponds to the currently-displayed item in the series, starting with 1 and incremented by 1. For a visual switch with two pictures in the series, the first picture in the list has a value of 1 and the second has a value of 2.

Example
The following example expands the scrolling window in the Invoice Entry window when the window initially opens:

```vba
Private Sub Window_BeforeOpen(OpenVisible As Boolean)
    ' ScrollWindowExpandButton2 = 2
End Sub
```
Chapter 4: Programming Grids

VBA refers to Microsoft Dynamics GP scrolling windows as “grids.” Programming grids involves adding a grid to your project, determining which grid events you’ll use to execute application code, and working with the window fields that appear in the grid.

Information about grids is divided into the following sections:

- Working with grids in your VBA project
- Programming a grid object
- Types of grids
- Using grid events
- Using grid properties and methods

Working with grids in your VBA project

Grids display multiple lines of data. Each line in a grid corresponds to one record in a Microsoft Dynamics GP table. Conversely, a Microsoft Dynamics GP window displays only a single record at a time. Since the functionality provided by a window is quite different from that provided by a grid, VBA references a window and a grid that appears in it as separate objects.

It’s also important to note that Microsoft Dynamics GP grids do not function like traditional grids in VBA. You can reference fields on only the current line in a Microsoft Dynamics GP grid, and grid events occur only for the current line. VBA grids typically allow you to work with specified rows and cells within the grid.

When you expand a grid, each line displays two or more rows of data for each record. You expand and shrink a grid using the expand and shrink buttons.
Adding a grid object
When you add a window containing a grid to your project, VBA adds both the window and the grid. Although the two work together to display information in the accounting system, they are independent objects in your VBA project.

To add a grid object to your project, switch to Microsoft Dynamics GP and display the window containing the grid you want to add. On the Tools menu, point to Customize, then choose Add Current Window To Visual Basic. This will automatically add the currently active (topmost) window and the window’s grid to your project.

When you switch back to Visual Basic, the Project Explorer will display both the grid and window object.

Removing a grid object
To remove a grid object from your project, switch to Microsoft Dynamics GP and display the window containing the grid you want to remove. On the Tools menu, point to Customize, then choose Remove Current Window From Visual Basic. VBA will remove the window and grid object, as well as any window field objects associated with the window and grid objects.

Be sure you remove all references to a grid after you’ve removed it from the project.

Adding fields
Fields that appear within a grid use the same properties, methods and events as fields that appear in a window. The only difference is that the VBA references the grid’s fields within the context of the grid, and references the window’s fields within the context of the window.

To add a grid’s fields to your project, switch to Microsoft Dynamics GP and display the window containing the grid.

Click the Expand button if you want to add fields from a line when it’s expanded. The expanded line will show two or more rows, each containing multiple fields.

Be sure to select a line in the grid. On the Tools menu, point to Customize, then choose Add Field To Visual Basic. This will activate field selection mode, and a selection pointer will appear. The following illustration shows the selection pointer.
Using the mouse, click on the fields you'll use in your project. You only need to add fields from one line to your project. References you make to the field in your VBA project will be for the line currently focused by the user, so it's not necessary to add the same field from multiple lines in the grid. When you've finished adding fields, point to Customize on the Tools menu and choose Add Fields to Visual Basic to deactivate field selection mode.

Field selection mode also deactivates when you switch to the Visual Basic Editor.

Programming a grid object

When programming the grid object, you can access only the current line in a grid through VBA. This means that grid events occur only for the current line, and you can set or return the values of fields that appear in the current line. Keep in mind that you won’t know which line in the grid is actually the current line; you must rely on the data values of the fields in the line to ascertain which line is selected. There are several ways you can use VBA to interact with a grid object in your project.

Use grid event procedures

Grid events execute a procedure when the accounting system fills a grid, when the user moves to a line, changes the contents of a line or exits the line. You’ll find grid events especially useful if you want to filter the information displayed in a grid, or if you want to set or return field values in a line as the user interacts with that line. The section titled Using grid events explains each type of grid event you can use to execute VBA code.

Reference fields that appear in the grid

Like windows, grids contain window fields that you can reference and manipulate through VBA. For grids whose fields are editable, window field events occur when the user moves the focus between fields in a line.

If you reference window fields in windows or grids outside the current grid, you must use a qualified field name. A qualified name explicitly specifies the location of the field you’re referencing. Refer to Chapter 3, “Programming Window Fields,” for more information about using window fields.

Reference the grid from other procedures

After you define a grid as an object in your project, any VBA code you write in your VBA project has full access to the grid object, its properties and its methods. You reference a grid in the same manner as a window, using the grid’s name as a qualifier for any grid fields you reference.
Types of grids

Three types of grids can be found in Microsoft Dynamics GP: browse-only, editable and adds-allowed. Each type has unique characteristics.

**Browse-only grids**
Browse-only grids allow the user to view information but not change it. Microsoft Dynamics GP uses browse-only grids in lookup windows.

![Browse-only grid example]

These are captions for each field in a grid line, and are not part of the grid. A browse-only grid contains non-editable lines.

**Editable grids**
Editable grids allow the user to change the contents of the selected line. Microsoft Dynamics GP uses these types of grids to change multiple options for one or more lines.

![Editable grid example]

You can change the information in an editable grid.

**Adds-allowed grids**
Adds-allowed grids let the user add additional lines of information to the grid. The grid stores the new information in a table. Microsoft Dynamics GP commonly uses adds-allowed grids in transaction entry windows, where the user can add multiple line items for a transaction.

![Adds-allowed grid example]

An adds-allowed grid has a blank line that allows users to add items.
Using grid events

A grid event executes a specific VBA procedure (an *event procedure*) only for the *current* line in a grid. The current line is the line that has the focus. To view the grid events available, select a grid object in your project, then display the Visual Basic Code window. Select Grid in the Object list; window events will appear in the Procedure list.

There are four basic kinds of grid events you’ll work with in VBA: *line got focus*, *line changed*, *line lost focus* and *line populate* events. Events occur when the user displays a new line in a grid (a *line populate* event) moves the focus into a grid line (a *line got focus* event), when the value of a line changes (a *line changed* event), or when the user moves the focus out of a line (a *line lost focus* event).

**Using modified or original windows**

If you use the Modifier to modify a window containing a grid (such as a lookup window), VBA grid events will occur only if you set both the window’s and the grid’s *EventMode property* to *emModifiedOnly*. Changing this property allows events to occur only for the modified version of the window and the grid. To change the *EventMode property*, use the Visual Basic Properties window.

**Line got focus events**

VBA line got focus events occur when the user moves to a line in a grid. The VBA *BeforeLineGotFocus* and *AfterLineGotFocus* events execute event procedures *before* or *after* Microsoft Dynamics GP code runs for the got focus event. Microsoft Dynamics GP typically uses the got focus event to verify whether the user can enter the line. Since you cannot cancel the Microsoft Dynamics GP got focus event from VBA, the VBA got focus event you choose may be of little consequence.

Line got focus events are useful for checking the value of a field in the line gaining focus. In the following example, the *BeforeLineGotFocus* event checks the value of the Item Number field in the Invoice Entry grid. If the field is empty, the event procedure opens the Items lookup window:

```vba
Private Sub Grid_BeforeLineGotFocus()
    If ItemNumber.Empty = True Then
        'Open the lookup window
        InvoiceEntry.LookupButton6 = 1
    End If
End Sub
```
**Line change events**

VBA line change events occur when the user changes the contents of a line in an editable or adds-allowed grid, then moves the focus out of the line. The VBA `BeforeLineChange` and `AfterLineChange` events occur before or after Microsoft Dynamics GP code runs for the line change event.

Microsoft Dynamics GP uses the line change event to save data in the line to a table when the user leaves the line. An example of this is the Invoice Entry window, where each grid line represents a separate transaction record in the invoice. As the user moves to a new line, the line change event saves the invoice line item to a table.

**BeforeLineChange**

The `BeforeLineChange` event occurs before Microsoft Dynamics GP code runs for the line change event. Use this event when you want to check the value of fields in a line, then cancel the Microsoft Dynamics GP line change event if specified criteria aren’t met. The `KeepFocus` parameter cancels the line change event that normally saves the contents of the line, and moves the focus to the field that last had focus in the line.

In the following example, the `BeforeLineChange` event checks the value of the Markdown Amount field in the Invoice Entry grid. If the user attempts to enter a markdown amount greater than 20%, the `KeepFocus` parameter stops the Microsoft Dynamics GP line change event and moves the focus back to the current line:

```vba
Private Sub Grid_BeforeLineChange(KeepFocus As Boolean)
    If CCurr(MarkdownAmount) > CCurr(UnitPrice) * 0.2 Then
        'The markdown was greater than 20%
        KeepFocus = True
        MsgBox "You cannot enter a markdown greater than 20% of the unit price."
    End If
End Sub
```

**AfterLineChange**

The `AfterLineChange` event occurs after Microsoft Dynamics GP code runs for the line change event. Use this event to perform other actions in the window after the accounting system saves data in the line. For instance, when the user enters line item information in the Invoice Entry grid, then moves to the next line item, the following `AfterLineChange` event procedure checks the Subtotal field. If the invoice subtotal is greater than $1000, it assigns a specific batch ID to the transaction:

```vba
Private Sub Grid_AfterLineChange()
    If CCurr(InvoiceEntry.Subtotal) > 1000 Then
        'Set the batch ID to an existing batch
        InvoiceEntry.BatchID = "BIGSALES"
        InvoiceEntry.BatchID.Locked = True
    End If
End Sub
```
Line lost focus events
VBA line lost focus events occur when the user moves the focus out of a line. The VBA BeforeLineLostFocus and AfterLineLostFocus events occur before or after the Microsoft Dynamics GP code runs for the lost focus event. Microsoft Dynamics GP typically uses the lost focus event to prepare the next line to accept data. Since you cannot cancel the got focus event from VBA, the VBA line lost focus event you choose may be of little consequence.

In the following example, an AfterLineLostFocus event “shrinks” the grid in the Invoice Entry window when the user moves to the next line.

```vba
Private Sub Grid_AfterLineLostFocus()
    InvoiceEntry.ScrollingWindowExpandButton = 1
End Sub
```

Line populate event
The VBA BeforeLinePopulate event occurs each time Microsoft Dynamics GP displays a new line in a grid. When the accounting system initially displays a grid, the BeforeLinePopulate event occurs repeatedly until the grid is full. It also occurs each time the user displays a new line in the grid, by scrolling up or down in the grid.

The primary reason to use the BeforeLinePopulate event is to filter out any lines you don’t want to display in a grid. For instance, the following event procedure displays only Illinois customers in the Customers and Prospects lookup window:

```vba
Private Sub Grid_BeforeLinePopulate(RejectLine As Boolean)
    If State <> "IL" Then
        RejectLine = True
    End If
End Sub
```

The RejectLine parameter lets you selectively filter records from the grid when it’s filled. Limit use of the RejectLine parameter, since it can have an impact on application performance.

A more elaborate use of the BeforeLinePopulate event could make use of a new field added to a lookup window using the Modifier. For example, you could use the Modifier to add a “Fill By State” field to the Customers and Prospects lookup window. A VBA event procedure for the Fill By State field fills the window by “clicking” the Redisplay button in the lookup window:

```vba
Private Sub FillByState_BeforeUserChanged(KeepFocus As Boolean, _
    CancelLogic As Boolean)
    Redisplay = 1
End Sub
```

As the fill occurs, the following procedure uses the BeforeLinePopulate event’s RejectLine parameter to reject records not indicated in the Fill By State field:

```vba
Private Sub Grid_BeforeLinePopulate(RejectLine As Boolean)
    If State <> CustomersandProspects.FillByState Then
        RejectLine = True
    End If
    CustomersandProspects.FillByState.Focus
End Sub
```
Using grid properties and methods

There are two properties available for the grid object: the **Name property** and the **EventMode property**.

### Changing a grid’s name

The **Name property** allows you to change the name you use in VBA code to reference the grid. You can change a grid’s name using the Visual Basic Properties window, and the name will change in your project. You should then recompile your project to replace any references to the previous grid name. You may want to rename a grid to make it more readable.

### Using the EventMode property

You can choose whether window events occur for the modified or original version of a grid using the **EventMode property**. If you use the Modifier to modify a window containing a grid, and you want VBA grid events to occur for the modified window, change the **EventMode property** for both the window and the grid object to **emModifiedOnly**. If you make no modifications to a Microsoft Dynamics GP window or grid using the Modifier, the default value for the **EventMode** property is **emOriginalOnly**.
Chapter 5: Programming Reports

Programming reports using VBA involves adding the report and report fields to your project, choosing the report events you’ll use to execute VBA code, and working with methods and properties to customize report display. Information about programming reports is divided into the following sections:

- Working with reports in your VBA project
- Using report events
- Start/End events
- Band events
- Using report properties and methods
- Working with report fields
- Exporting data to other applications

Working with reports in your VBA project

To use VBA with a report, you’ll first need to add the report to your project. Adding a report creates a corresponding report object in your project, which you can reference in VBA.

Adding a report object

To add a report to your project, launch Report Writer. Create a new or modified version of the report you want to work with, then display the report’s layout. With the Report Layout window active, choose Add Report To Visual Basic from the Tools menu.

After you’ve added a report, switch to the Visual Basic Editor and display the Project Explorer. It should look similar to the following:

Although you created a modified report using Report Writer, this was only necessary to access the report’s layout and create the report object in VBA. You can determine whether you want VBA code to run for either the modified or the original version of the report using the report’s EventMode property. See the section titled Using modified or original reports on page 65 for more information about the EventMode property.

Removing a report object

To remove a report object from your project, launch Report Writer and open the report layout for the report you want to remove. From the Tools menu, point to Customize, then choose Remove Current Report From Visual Basic. VBA will remove the report and any report fields associated with the report from your project.
Adding report field objects

You can add two types of report fields to your VBA project: table fields and calculated fields. To add a report field to your project, select the field in the report layout using the arrow tool, then choose Add Fields To Visual Basic from the Tools menu.

If the same field appears multiple times on a report, it’s important that you select the field located in the report section from which you are going to reference the field. For example, a report field may appear in the report body and in the page footer. If you will be referencing the field from the **BeforePF event**, be sure you’re adding the field located in the page footer.

If the field is not a table field or a calculated field, the Add Fields To Visual Basic menu item will appear disabled. To add several fields at once, hold down the **SHIFT** key and select the fields, then choose Add Fields To Visual Basic.

After you’ve added a report field to a project, you can reference it only from within a report event procedure. Report fields, unlike window fields, have no associated field events. Therefore, you cannot write VBA code specifically for a report field.

Using report events

A report event executes a specific procedure (an event procedure) when a report prints. To view the report events available, select a report object in the Project Explorer and display the Visual Basic Code window. Select Report in the Object list; report events will appear in the Procedure list.

There are two kinds of report events you’ll work with: **start/end events** and **band events**. A **Start** event always occurs at the beginning of a report and an **End** event always occurs at the end of a report. **Band** events occur when a particular section of a report, or a **band**, prints.
CHAPTER 5  PROGRAMMING REPORTS

Using modified or original reports
Keep in mind that you only need to use Report Writer to add report and report field objects to VBA. If you don’t modify the report directly using Report Writer, you can use VBA with the original report in Microsoft Dynamics GP.

Start/End events

Start/end events occur when the report starts and the report ends. The only report data you have access to from start/end events are report legends, which you can set or return in the Start event. To access report fields, you must use band events.

Start event

The Start event occurs just before the report starts to print, before any data actually prints. The primary use of the Start event is to set report legends using the Legend property. Legends are fields whose data is persistent throughout the report, and which must be passed to the report as it starts. The following example sets the value of a legend field from the Start event:

Private Sub Report_Start()
    RMCustomerReport.Legend(2) = "Aging Period Amount"
End Sub

You can also use the report’s Start event to initialize any module-level variables you’ll use for the report. The following examples use a module-level variable named Count to restrict a report to 10 customer records. The Start event sets the counter to 0, then the BeforeAH event increments the counter. After 10 records print, the BeforeAH event cancels the report using the Cancel method.

' Declare a module-level variable
Private Count As Integer
---------------------------------------------------------------
Private Sub Report_Start()
    ' Initialize the count
    Count = 0
End Sub
---------------------------------------------------------------
Private Sub Report_BeforeAH(ByVal Level As Integer, _
    SuppressBand As Boolean)
    ' Increment the count
    Count = Count + 1
    If Count = 11 Then
        RMCustomerReport.Cancel
    End If
End Sub
End event
The End event occurs after a report prints. Use this event to perform any clean-up activities for the report, launch other applications, or open and close other windows. For example, the following End event procedure launches Microsoft Outlook® after the RM Customer Report prints:

```vba
Private Sub Report_End()
    Dim Response As Integer
    Dim RetVal As Variant
    Response = MsgBox("Do you want to launch MS Outlook?", vbYesNo)
    If Response = vbYes Then
        RetVal = Shell("C:\Program Files\Microsoft Office\ + _
"Office\Outlook.exe", vbNormalFocus)
    End If
End Sub
```

Band events
A report is composed of several sections called bands. In VBA, each report band has a corresponding band event. As a report prints, each band event occurs just before the corresponding data within the band prints. The following illustration shows the different bands that compose a report:

The primary use of a band event is to return or set the value of a field in a corresponding band using the Value property. You can use a band event to return a value for any report table field or calculated field. You can set a value only for a user-defined calculated field you’ve added using the Report Writer.
In the following illustration, the Customer Name table field and EMailAddress user-defined calculated field appear within the RM Customer Report’s additional header. In this case, only the **BeforeAH** event can reference these fields:

The **BeforeAH** event procedure can return the value of the Customer Name table field, and set EMailAddress user-defined calculated field:

```vbnet
Private Sub Report_BeforeAH(ByVal Level As Integer, SuppressBand As Boolean)
    If CustomerName = "Aaron Fitz Electric" Then
        'Set the user-defined calculated field
        EMailAddress = "aftiz@aol.com"
    End If
End Sub
```

Each band event includes a **SuppressBand** parameter. When set to True, the **SuppressBand** parameter stops the current instance of the corresponding band from printing. If a band prints multiple times in a report (such as the report body), you can conditionally suppress a given instance of the band. For example, the following **BeforeBody** procedure for the RM Salesperson Report suppresses the current instance of the report body if the value of the Commissioned Sales To Date field is less than $200,000:

```vbnet
Private Sub Report_BeforeBody(SuppressBand As Boolean)
    If CCur(CommissionedSalesToDate) < 200000 Then
        SuppressBand = True
    End If
End Sub
```

If the report uses an accumulator field to calculate report totals or count the number of bodies in a report, the accumulator field will still sum or count data from fields in a suppressed band.

**BeforePH event**

The **BeforePH** event occurs before items in the **page header** print. Items in the page header are placed at the top of every report page. Page number, date and time fields are commonly placed in this section of a report.

**BeforeRH event**

The **BeforeRH** event occurs before items in the **report header** print. Items in the report header appear only on the first page of a report. The title of the report and introductory information are often included in this section. If a page header is also included on the first page, the report header will appear after the page header.
BeforeAH event
The BeforeAH event occurs before the report’s additional header prints. A report may have none, one or several additional headers, each indicated on the report layout by H1, H2 and so on. Each prints when the data in the field it is based on changes.

If a report uses multiple additional headers, use the BeforeAH event’s Level parameter to specify an index corresponding to the number of the additional header the event occurs for. In the following example, the BeforeAH event procedure runs for the second additional header on a report. The procedure also returns the value of the Checkbook ID field in the second additional header:

```vba
Private Sub Report_BeforeAH(ByVal Level As Integer, SuppressBand As Boolean)
    If Level = 2 Then
        If CheckbookID = "PAYROLL" Then
            BankDepositPostingJournal.Cancel
        End If
    End If
End Sub
```

If the report has only one additional header, the BeforeAH event occurs only for that additional header, and you don’t need to specify a value for the Level parameter.

BeforeBody event
The BeforeBody event occurs before each instance of the report’s body prints. Microsoft Dynamics GP uses the report body for the bulk of a report, which typically is made up of table fields. A report prints the body repeatedly until all records in the report have printed.

BeforeRF event
The BeforeRF event occurs before the report footer prints. Items in the report footer appear only on the last page of a report. Summary information is often included the report footer. If a page footer is also included on the last page, the report footer will print before the page footer.

BeforePF event
The BeforePF event occurs before the report’s page footer prints. Items in the page footer are placed at the bottom of every report page. The page footer often includes administrative information, such as the name of the person running the report.

BeforeAF event
The BeforeAF event occurs before the report’s additional footer prints. A report may have none, one or several additional footers, each indicated on the report layout by F1, F2 and so on. Each prints when the data in the field it is based on changes. Microsoft Dynamics GP uses the additional footer to display summary data, such as a total of all records in the report’s body.

If a report uses more than one additional footer, use the BeforeAF event’s Level parameter to specify an index corresponding to the number of the additional header you want the event to occur for. In the following example, the BeforeAF event procedure runs for the second additional footer that appears on the report:
Private Sub Report_BeforeAF(ByVal Level As Integer, SuppressBand As Boolean)
    If Level = 2 Then
        If CustomerName = "Aaron Fitz Electric" Then
            'Set the user-defined calculated field
            EMailAddress = "aftiz@aol.com"
        End If
    End If
End Sub

If the report has only one additional footer, the BeforeAF event occurs only for that additional footer, and you don’t need to specify a value for the Level parameter.

Using report properties and methods

The following table explains the available report methods and properties. The remainder of this section explains some of the more common methods and properties you’ll use in when working with reports:

<table>
<thead>
<tr>
<th>Property/Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cancel method</td>
<td>Cancels the report from any report event.</td>
</tr>
<tr>
<td>EventMode property</td>
<td>Specifies whether report events occur for the original or modified version of the report.</td>
</tr>
<tr>
<td>Legend property</td>
<td>Specifies the value of a report legend.</td>
</tr>
<tr>
<td>Name property</td>
<td>Specifies the internal name VBA uses for the report.</td>
</tr>
</tbody>
</table>

Canceling a report

You cancel a report using the Cancel method. The following example cancels a report from the report’s BeforeBody event:

Private Sub Report_BeforeBody(SuppressBand As Boolean)
    If SalespersonID = "ERIN J." Then
        RNSalespersonReport.Cancel
    End If
End Sub

If you cancel a report from a report’s band event, the report stops at that band. If you cancel a report from a report’s Start event, no data prints for the report. In all cases, the report’s End event will occur for a canceled report. If the report is printed to the screen, the Cancel method closes the Screen Output window automatically.

Working with legend fields

Use the Legend property to set or return the value of a legend field in a report. You don’t add a legend field to your project; instead, you use the Legend property’s index parameter to refer to a specific legend field on the report. In addition, you can set or return the value of a legend only in the report’s Start event.

A legend field contains data that’s passed to the report at runtime, before it prints. These fields typically include information about the report, such as the range of records the user chose for the report. The following illustration shows legend fields used to show the range of customer records that print for the RM Customer Report:

This is the legend’s index.
The Legend property references the legend’s index, not the legend field.

Customer ID: [Legend 5]
Salesperson ID: [Legend 3]
Customer Class: [Legend 7]
For example, to return the value of the Customer Class legend field, the following Start event procedure includes a reference to the field’s index:

```vba
Private Sub Report_Start()
    If RMCustomerReport.Legend(5) = "AARONFIT0001 - AARONFIT0001" Then
        'Cancel the report
        RMCustomerReport.Cancel
    End If
```

**Working with report fields**

You can use the *Value property* to return the value of a table field, a Microsoft Dynamics GP calculated field, or a user-defined calculated field. You can also use the *Value property* to set the value of a user-defined calculated field. For any field, the report event you use must correspond to the report band containing the field.

**Table fields**

Table fields are fields whose values derive directly from a Microsoft Dynamics GP table. You can return only the value of table fields.

**Calculated fields**

Calculated fields are defined using table fields, constants and operators. Microsoft Dynamics GP typically uses calculated fields to conditionally print data on a report. You can only return the value of a Microsoft Dynamics GP calculated field. You can set or return the value of a calculated field you added to the report using Report Writer.

**Accumulator fields**

You can return the values of “accumulator” fields in a report. You can also set the value of calculated fields you add to the report that function as accumulator fields. An accumulator field is a table field or calculated field that performs a specific function in the report, and whose field type is not DATA. The field type for an accumulator field determines its function. For instance, an accumulator field can count the number of occurrences of a field (a COUNT field type), or calculate the sum total of each occurrence of a numeric field (a SUM field type). Use the Report Writer’s Report Field Options window to determine the field type for a selected field.

**Field formats**

With the exception of composite fields, VBA returns report field values without any formatting. For instance, VBA returns the phone number (701) 555-0100 Ext. 1772 as 70155501001772. Likewise, VBA returns the currency value $10.56 as 10.56000.

To format a report field value, use VBA’s *Format* function. The *Format* function allows you to define a format string for a specific field type (string or numeric). The following example uses the *Format* function to format the fax number string field:

```vba
MsgBox Fax
    'Displays 70155501001772
    'Use the format function to format the string
    Format(Fax, "(@@@) @@@-@@@@  Ext. @@@@")
MsgBox Fax
    'Displays (701) 555-0100 Ext. 1772
```
Hiding and showing report fields
Use the field object’s Visible property to hide and show report fields. You can hide any table field or calculated field on a report. However, you can only show fields you’ve hidden using VBA. If invisible, a field’s value is still available to VBA, but will not appear in the report.

The following example uses the Visible property to hide the Total Sales YTD field in the RM Customer Report if the Territory ID field is TERRITORY 1:

```vba
Private Sub Report_BeforeAH(ByVal Level As Integer, SuppressBand As Boolean)
    If SalesTerritory = "TERRITORY 1" Then
    'Hide the YTD sales amount
    TotalSalesYTD.Visible = False
    Else
    TotalSalesYTD.Visible = True
    End If
End Sub
```

Returning fields from multiple bands
Since you only have access to field values as the field’s band prints, you’ll likely find it necessary to use module-level variables to store report information from multiple bands. As each band prints, you can store selected values from the band using the module-level variables.

The following example exports data from the Microsoft Dynamics GP Item Price List to a text file. The procedure declares several module-level variables to store report data. It then uses VBA events (the BeforeAH and BeforeBody events) to return field values from their respective bands and set module-level variables:

```vba
'Declare module-level variables
Private QTY As Integer
Private ItmDesc, ItmNum, VenName As String

Private Sub Report_BeforeAH(ByVal Level As Integer, SuppressBand As Boolean)
    If CInt(QTYOnHand) < 10 Then
    'Set the module-level variables for fields in this band
    ItmDesc = ItemDescription
    ItmNum = ItemNumber
    QTY = QTYOnHand
    Else
    QTY = 0
    End If
End Sub
```
Private Sub Report_BeforeBody(SuppressBand As Boolean)
    'Check the value of the QTY variable
    If QTY <> 0 Then
        'Set the module-level variable for the field in this band
        VenName = VendorName
        'Create a text file. You could also export these to Excel
        Open "ItemQuantities.TXT" For Append As #1
        Print #1, "Item Number: " + ItmNum
        Print #1, "Item Description: " + ItmDesc
        Print #1, "Avail. QTY: " + Str(QTY)
        Print #1, "Vendor Name: " + VenName
        Print #1, ""
        Close #1
    End If
End Sub

Exporting data to other applications

Reports are useful for exporting multiple records from the accounting system to other VBA-compliant applications, such as Microsoft Excel, Outlook or Word. A report’s output can include several records, such as customer, item or transaction records. You can use VBA to analyze each record in the report as the report prints, then export specific field values to another application.

Before you can work with another application’s objects, you must set a reference to that application’s object library using the References dialog box in the Visual Basic Editor.

The following example uses the RM Customer Report to create customer contacts in Microsoft Outlook. The example first declares two module-level variables used to create Outlook contact objects. The report’s Start event then creates and returns a reference to Outlook using VBA’s CreateObject() function.

The RM Customer Report’s additional header contains the majority of customer information, so the procedure uses the BeforeAH event to return customer information. As Report Writer prints each additional header (one per customer record), the procedure checks the Sales Territory field. For each customer in “TERRITORY1,” the procedure creates a contact object in Microsoft Outlook, and sets properties in the contact object to fields in the report:

'Declare module-level variables in the general section
Dim DynItem, DynOlApp As Object
---------------------------------------------------------------------
Report_Start()
    'Create the Outlook object
    Set DynOlApp = CreateObject("Outlook.Application")
End Sub
---------------------------------------------------------------------
Private Sub Report_BeforeAH(ByVal Level As Integer, _
    ByVal Level As Integer, _
    SuppressBand As Boolean)
    If SalesTerritory = "TERRITORY 1" Then
        'Add customer information to Outlook
        Set DynItem = DynOlApp.CreateItem(olContactItem)

        'Set contact object properties to window field values
        DynItem.FullName = ContactPerson
        DynItem.CompanyName = CustomerName
    End If
End Sub
DynItem.BusinessAddressStreet = Address1
DynItem.BusinessAddressCity = City
DynItem.BusinessAddressState = State
DynItem.BusinessAddressPostalCode = Zip

'Format report fields
DynItem.BusinessTelephoneNumber = Format(Phone1, "(@@@) @@@-@@@@  Ext. @@@@")
DynItem.BusinessFaxNumber = Format(Fax, "(@@@) @@@-@@@@  Ext. @@@@")
'Save the contact object
DynItem.Save
End If
End Sub
Chapter 6: Storing Additional Data

The Dynamic User Object Store (DUOS) lets you use VBA to create, store and retrieve user-definable data objects. You can use the DUOS to store data entered in existing Microsoft Dynamics GP fields, new fields added using the Modifier, or fields entered in a VBA user form. You can then retrieve a data object from the DUOS and display it in a VBA user form or in a Microsoft Dynamics GP window. Information about the DUOS is divided into the following sections:

- Understanding the DUOS
- Structure of the DUOS
- Working with DUOS data objects
- Retrieving a DUOS data object
- Deleting a DUOS data object
- Updating a DUOS data object
- DUOS properties and methods
- Using the DUOS

Understanding the DUOS

The VBA object model uses two collections and two objects to define DUOS data objects. The DUOSObjects collection contains one or more user-definable collections you create to store related data objects, such as customer, vendor or item collections. The DUOSObject object represents a single data object within a collection, such as a customer object within a customer collection. The following illustration shows the DUOS portion of the VBA object model:

```
DUOSObjects
   DUOSObject
      DUOSProperties
         DUOSProperty

Object
Collection Object
```

The second collection, the DUOSProperties collection, contains all the properties for the data objects in the DUOSObjects collection. The DUOSProperty object represents a single property for a single object in the DUOSProperties collection.

Refer to Part 2, VBA Object Library, for detailed information about the methods and properties available for the DUOS objects and collections.
Structure of the DUOS

The DUOS is composed of user-defined collections, each of which can store several data objects. Each data object can contain several properties and property values. The following table shows the structure of the DUOS that contains two user-defined collections (Customers and Items):

<table>
<thead>
<tr>
<th>Collection name</th>
<th>Object ID</th>
<th>Property name</th>
<th>Property value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customers</td>
<td>ALTONMAN0001</td>
<td>URL Address</td>
<td><a href="http://www.altonman.com">www.altonman.com</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Contact E-Mail Address</td>
<td><a href="mailto:jrossini@aol.com">jrossini@aol.com</a></td>
</tr>
<tr>
<td>ADVANCED0001</td>
<td>URL Address</td>
<td><a href="http://www.advanced.com">www.advanced.com</a></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Contact E-Mail Address</td>
<td><a href="mailto:mmonat@aol.com">mmonat@aol.com</a></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Customer Type</td>
<td>RETAIL</td>
<td></td>
</tr>
<tr>
<td>Items</td>
<td>ITM001</td>
<td>Item Weight</td>
<td>2.45</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Item Color</td>
<td>Blue</td>
</tr>
<tr>
<td></td>
<td>ITM004</td>
<td>Item Weight</td>
<td>3.50</td>
</tr>
</tbody>
</table>

- The **Collection name** is a string specifying the user-defined collection where you’re storing the data object. This name cannot exceed 30 characters. You can create or specify the collection you want to work with using the global DUOSObjectsGet method.

- The **Object ID** is a unique value for a data object within its collection. This ID cannot exceed 60 characters. You can specify the object ID using the Item property from the DUOSObjects collection.

- The **Property name** is a string that specifies the name of the property for the object. The property name cannot exceed 30 characters. You can define several property names for a data object. Each object in a collection may contain different property names. You can specify the property name using the Item property in the DUOSProperties collection.

- A data object must have at least one property name to remain persistent in the DUOS.

- The **Property value** is a string that specifies the value of the property. This value cannot exceed 132 characters. The DUOS stores all property values as strings. You can specify a property value using the Value property from the DUOSProperty object.

Microsoft Dynamics GP stores DUOS data objects in a table named SY_User_Object_Store (SY90000.*). This table is company-specific, so Microsoft Dynamics GP stores data objects you create for each company in separate tables, located in separate directories. The combination of the collection name, object ID, property name and property value all make up a single record in this table:

<table>
<thead>
<tr>
<th>Collection name</th>
<th>Object ID</th>
<th>Property name</th>
<th>Property value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Items</td>
<td>ITM000</td>
<td>Item Weight</td>
<td>3.5</td>
</tr>
<tr>
<td>Items</td>
<td>ITM000</td>
<td>Item Color</td>
<td>Red</td>
</tr>
<tr>
<td>Items</td>
<td>ITM001</td>
<td>Item Weight</td>
<td>2.45</td>
</tr>
<tr>
<td>Items</td>
<td>ITM001</td>
<td>Item Color</td>
<td>Blue</td>
</tr>
</tbody>
</table>
If two additional properties and property values are added to the ITM001 data object, the accounting system adds two records to this table:

<table>
<thead>
<tr>
<th>Collection name</th>
<th>Object ID</th>
<th>Property name</th>
<th>Property value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Items</td>
<td>ITM000</td>
<td>Item Weight</td>
<td>3.5</td>
</tr>
<tr>
<td>Items</td>
<td>ITM000</td>
<td>Item Color</td>
<td>Red</td>
</tr>
<tr>
<td>Items</td>
<td>ITM001</td>
<td>Item Weight</td>
<td>2.45</td>
</tr>
<tr>
<td>Items</td>
<td>ITM001</td>
<td>Item Color</td>
<td>Blue</td>
</tr>
<tr>
<td>Items</td>
<td>ITM001</td>
<td>Item Height</td>
<td>4.5</td>
</tr>
<tr>
<td>Items</td>
<td>ITM001</td>
<td>Item Width</td>
<td>2.75</td>
</tr>
</tbody>
</table>

Since Microsoft Dynamics GP stores DUOS data objects in a table, you can use table maintenance windows to perform routine table maintenance.

**Working with DUOS data objects**

To create a DUOS data object, use the DUOS global `DUOSObjectsGet method` to specify a collection. If the collection doesn’t exist, this method creates it. The following example specifies a collection named “Additional Item Info.”

The example then uses the collection’s `Item property` to assign a unique ID to the data object (ITM002). If an object with that ID already exists, the `Item property` specifies an existing object in the collection with that ID. Finally, this example creates two new properties for the data object using the DUOSObjects collection’s `Item property`, and assigns values to each:

```vba
Dim ItemCollection As DUOSObject
Dim ItemObject As DUOSObject

'Specify the collection
Set ItemCollection = DUOSObjectsGet("Additional Item Info")

'Specify an object in the collection with a unique object ID
Set ItemObject = ItemCollection.Item("ITM002")

'Create properties and property values for this object
ItemObject.Properties.Item("Item Weight") = "3.5"
ItemObject.Properties.Item("Item Color") = "Red"
```

If you’re storing values entered from window fields (either in a Microsoft Dynamics GP window or from a VBA user form), set the object’s property values to the values of the window fields. A portion of the previous example would then look like this:

```vba
'Set the object ID using the ItemID field
Set ItemObject = ItemCollection.Item(ItemID)

'Set property values using two window fields
ItemObject.Properties.Item("Item Weight") = ItemWeight
ItemObject.Properties.Item("Item Color") = ItemColor
```
Retrieving a DUOS data object

To retrieve a DUOS data object, you return an object from the object’s collection, then set window fields to the property values for the object:

```vba
Dim ItemCollection As DUOSObjects
Dim ItemObject As DUOSObject

'Specify the collection
Set ItemCollection = DUOSObjectsGet("Additional Item Info")
'Specify the object to display in the window
Set ItemObject = ItemCollection("ITM002")
'Set two window fields
ItemWeightField = ItemObject.Properties.Item("Item Weight")
ItemColorField = ItemObject.Properties.Item("Item Color")
```

Deleting a DUOS data object

You delete a DUOS object using the collection’s `Remove` method. In the following example, the procedure removes the data object with the ID “ITM002” from the `ItemCollection`:

```vba
Dim ItemCollection As DUOSObjects

Set ItemCollection = DUOSObjectsGet("Additional Item Info")
ItemCollection.Remove("ITM002")
```

Updating a DUOS data object

You can update a DUOS data object when window field values change. In this example, the procedure updates a data object’s properties, and adds a property named “Date Changed”:

```vba
Dim ItemCollection As DUOSObjects
Dim ItemObject As DUOSObject

Set ItemCollection = DUOSObjectsGet("Additional Item Info")
Set ItemObject = ItemCollection.Item("ITM002")

'Set property values from window fields for this object
ItemObject.Properties.Item("Item Weight") = ItemWeight
ItemObject.Properties.Item("Item Color") = ItemColor

'Set the date changed property to the VBA system date
ItemObject.Properties.Item("Date Changed") = Date
```
You can also globally update DUOS data objects by looping through each object in the collection. In this example, the `For...Next` statement in VBA checks the `Item Color` property for each item in the `ItemCollection`. If the value is Red, the collection's `Remove` method deletes the object:

```vba
Dim ItemCollection As DUOSObjects
Dim ItemObject As DUOSObject

Set ItemCollection = DUOSObjectsGet("Additional Item Info")
For Each ItemObject In ItemCollection
    If ItemObject.Properties.Item("Item Color") = "Red" Then
        'Delete the object
        ItemCollection.Remove(ItemObject.ID)
    End If
Next
```

### DUOS properties and methods

Properties and methods you’ll use for the DUOS are members of three objects and two collections in VBA. The following table lists each object, its member properties and methods, and a brief description of each:

<table>
<thead>
<tr>
<th>Object/Collection</th>
<th>Property/Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Global object</td>
<td>DUOSObjectCombineID method</td>
<td>Combines two strings to generate a unique object ID.</td>
</tr>
<tr>
<td></td>
<td>DUOSObjectExtractID method</td>
<td>Extracts two strings from a combined object ID.</td>
</tr>
<tr>
<td></td>
<td>DUOSObjectsGet method</td>
<td>Returns or creates a user-defined collection in the DUOS.</td>
</tr>
<tr>
<td>DUOSObjects collection</td>
<td>Exists property</td>
<td>Returns whether a data object exists in a user-defined collection.</td>
</tr>
<tr>
<td></td>
<td>Item property</td>
<td>Returns a data object from a user-defined collection.</td>
</tr>
<tr>
<td></td>
<td>SortByExisting property</td>
<td>Specifies the property used to sort the data objects in a user-defined collection.</td>
</tr>
<tr>
<td></td>
<td>Type property</td>
<td>Returns the name of a user-defined collection the data object belongs to.</td>
</tr>
<tr>
<td></td>
<td>Remove method</td>
<td>Removes a data object from a user-defined collection.</td>
</tr>
<tr>
<td>DUOSObject object</td>
<td>ID property</td>
<td>Returns the object ID for a data object.</td>
</tr>
<tr>
<td></td>
<td>Properties property</td>
<td>Returns properties for a data object.</td>
</tr>
<tr>
<td>DUOSProperties collection</td>
<td>Count property</td>
<td>Specifies the number of properties in a data object.</td>
</tr>
<tr>
<td></td>
<td>Exists property</td>
<td>Specifies whether a property exists for a data object</td>
</tr>
<tr>
<td></td>
<td>Item property</td>
<td>Returns or creates a user-defined property in a data object.</td>
</tr>
<tr>
<td></td>
<td>Remove method</td>
<td>Removes a user-defined property from a data object.</td>
</tr>
<tr>
<td>DUOSProperty object</td>
<td>Name property</td>
<td>Specifies the name of a user-defined property.</td>
</tr>
<tr>
<td></td>
<td>Value property</td>
<td>Specifies the value of a user-defined property.</td>
</tr>
</tbody>
</table>
Using the DUOS

Using the DUOS typically involves adding fields to a window using the Modifier, and writing VBA code that saves, displays or deletes the data object in the DUOS when the user saves, displays or deletes a corresponding record in the window. The remainder of this section explains how to address each of these issues in more detail.

A URL Address field and Contact E-mail Address field were added to the Customer Maintenance window with the Modifier. The examples in the remainder of this section will create a DUOS customer collection and data objects to store and display information in these fields.

Saving a DUOS data object

There are two situations where you’ll need to save a DUOS data object: when the user clicks the Save button in a window, and when the user attempts to close the window or display another record without saving changes in the window (such as when using browse buttons). When you store a DUOS data object, you’ll need to address both of these situations.

Using the Save button

The following BeforeUserChanged field event procedure runs when the user clicks the Save button in the Customer Maintenance window. It first checks whether the user entered all required fields in the window using the window’s Required property. It then specifies the collection where it will store the object using the DUOSObjectsGet method.
To ensure the object has a unique ID, the ObjectID parameter of the **Item property** uses the value of the Customer ID field. The procedure defines the DUOS data object's property names ("URL Address" and "Contact E-Mail Address") and sets its property values to the values entered in the new fields:

```vba
Private Sub Save_BeforeUserChanged(KeepFocus As Boolean, _
CancelLogic As Boolean)
    Dim CustomerCollection As DUOSObjects
    Dim CustomerObject As DUOSObject

    If CustomerMaintenance.Required = True Then
        'The user entered all required fields. Create/get the collection
        Set CustomerCollection = DUOSObjectsGet("CustomerCollection")

        'Create the object, using the customer ID as the object ID
        Set CustomerObject = CustomerCollection.Item(CustomerID)

        'Set the object's property names and values
        CustomerObject.Properties("URL Address") = URLAddress
        CustomerObject.Properties("Contact E-Mail Address") = ContactEMailAddress
    End If
End Sub
```

**Using the Save dialog**
The Microsoft Dynamics GP save dialog appears when the user makes changes to a record, then attempts to move to another record or close the window without saving changes. This dialog appears in the following illustration.

The user can click the Save button to save the changes, the Discard button to ignore the changes, or the Cancel button to close the dialog and cancel the save operation. In this situation, you’ll need to know when the user clicks the Save button, and save any changes to the corresponding DUOS data object. To do this, use a window **AfterModalDialog** event.

In the following example, the **AfterModalDialog** event procedure runs when a modal dialog appears for the Customer Maintenance window. It uses the **PromptString** parameter to ascertain whether the Microsoft Dynamics GP save dialog appeared. If the user clicks the Save button, the procedure specifies the collection, then specifies the data object using the collection’s **Item property**: 
Private Sub Window_AfterModalDialog(ByVal DlgType As Long, _ PromptString As String, Control1String As String, Control2String As String, Control3String As String, Answer As Long)
    Dim CustomerCollection As DUOSObjects
    Dim CustomerObject As DUOSObject
    If PromptString = "Do you want to save changes to this " + _
        "customer?" Then
        'The save dialog appeared, asking if the user wants to save
        'changes
        If Answer = dcButton1 Then
            'The user clicked Save. Get the collection
            Set CustomerCollection = DUOSObjectsGet _
                ("CustomerCollection")
            'Return the correct customer object
            Set CustomerObject = CustomerCollection.Item(CustomerID)
            'Save the values from the window fields
            CustomerObject.Properties("URL Address") = URLAddress
            CustomerObject.Properties("Contact E-Mail Address") = ContactEMailAddress
            End If
        End If
    End If
End Sub

You’ll also use the window’s AfterModalDialog event to respond to the Microsoft Dynamics GP delete dialog. See Deleting a DUOS data object on page 78 for more information.

Retrieving a DUOS data object
After you’ve saved a DUOS data object, you’ll need to display it when the user displays its associated record. To do this, add a Changed event procedure for the window’s control field (the field that controls the display of the record in the window). This field also provides the record with a unique value, such as an item or customer ID, or a document number.

In the following example, the Customer Maintenance window’s control field is the Customer ID field. Whenever this field changes (such as when the user browses records in the Customer Maintenance window), the Changed event procedure runs. It uses the collection’s Item property to return the data object matching the new value in the Customer ID field. The procedure then updates window fields using the customer object’s Properties property:

Private Sub CustomerID_Changed()
    Dim CustomerCollection As DUOSObjects
    Dim CustomerObject As DUOSObject
    Set CustomerCollection = DUOSObjectsGet("CustomerCollection")
    If CustomerID.Empty = False Then
        'There’s a new value in the field. Retrieve the customer
        'object from the collection using the value in the
        'Customer ID field
        Set CustomerObject = CustomerCollection.Item(CustomerID)
        'Update the window fields
        URLAddress = CustomerObject.Properties("URL Address")
        ContactEMailAddress = CustomerObject.Properties _
            ("Contact E-Mail Address")
    End If
End Sub
**Deleting a DUOS data object**
You’ll likely want to delete a DUOS data object from the collection when the user deletes its corresponding record in the accounting system. This ensures that you don’t have an “orphaned” DUOS data object after the user deletes the corresponding record.

Instead of writing VBA code that deletes a DUOS data object when the user clicks the Delete button, you delete a DUOS data object based on how the user responds to the delete dialog. This modal dialog appears after the user clicks the Delete button. This example is intended to work in conjunction with the example for the Microsoft Dynamics GP message dialog box, which follows this example.

The user can either click Delete to delete the record, or click Cancel and cancel the operation. You’ll want to delete a DUOS object only if the user clicks Delete in response to the dialog. Similar to saving DUOS objects, you can accomplish this by using the `AfterModalDialog` event. The code that is added to this event will then track whether the user selected the delete button.

In some situations, there’s one other condition you’ll need to address when deleting a DUOS object. A second message dialog may appear if the accounting system was unable to delete the record. For example, an additional message will be displayed if a customer record has any outstanding transactions corresponding to it.

The second message dialog needs to be handled differently than the delete dialog. This is because there’s no `AfterModalDialog` event available for the secondary message. Instead, you’ll need to handle the message dialog as a separate Microsoft Dynamics GP window and add it to your VBA project. Once added, you use VBA code in the window’s `AfterOpen` event to ascertain whether the message prevented the record from being deleted.

The following series of examples show how to do this for DUOS data objects stored with customer information.

**Example: Customer Maintenance window**
The following example shows the VBA code for the Customer Maintenance window that’s necessary to correctly delete a DUOS data object. This example is intended to work in conjunction with the example for the Microsoft Dynamics GP message dialog box, which follows this example. Together, these two examples allow you to correctly handle DUOS delete operations for the Customer Maintenance window.

Notice that there are three different variables (`WantToDelete`, `OkToDelete` and `CustIDToDelete`) used throughout these examples. These are declared as public variables and then initialized and set in the appropriate events:
'Declare module-level variables in the General section of the `Customer Maintenance window object
Public WanttoDelete As Boolean
Public OktoDelete As Boolean
Dim CustIDtoDelete As String

---------------------------------------------------------------------
Private Sub CustomerID_Changed()
    If CustomerID.Empty = False Then
        'Initialize variables
        CustIDtoDelete = CustomerID
        OktoDelete = True
        WanttoDelete = False
    End If
End Sub

---------------------------------------------------------------------
Private Sub Window_AfterModalDialog(ByVal DlgType As Long, _
    PromptString As String, Control1String As String, Control2String As _
    String, Control3String As String, Answer As Long)
    If PromptString = "Are you sure you want to delete this + _
    "customer record?" Then
        If Answer = dcButton1 Then
            'The user clicked Delete on the delete dialog message
            WanttoDelete = True
        End If
    End If
End Sub

---------------------------------------------------------------------
Private Sub Delete_AfterUserChanged()
    Dim CustomerCollection As DUOSObject
    Dim CustomerObject As DUOSObject
    'The user clicks the Delete button and no transactions exist
    If WanttoDelete And OktoDelete Then
        'Create the collection
        Set CustomerCollection = DUOSObjectGet("Customer _
        Information")
        'Create the object using the CustomerID that was on the
        'window
        Set CustomerObject = CustomerCollection.Item(CustIDtoDelete)
        'Remove the object
        CustomerCollection.Remove (CustIDtoDelete)
    End If
End Sub

Example: Dynamics1 message dialog window
The following example is attached to the Dynamics1 window object (the second
message dialog). This example works with the example for the Customer
Maintenance window (shown in the previous example). In this case, the public
variable declared in the General section of the Customer Maintenance window code
(OktoDelete) is set to False, since Microsoft Dynamics GP cannot delete the customer
record:

Private Sub Window_AfterOpen()
    CustomerMaintenance.OktoDelete = False
End Sub
Printing data from the DUOS

After you’ve used the DUOS to display additional information in a window, you can use VBA to print DUOS information on a report. Printing data from the DUOS involves adding calculated fields to the report, adding the report and the calculated fields from the report to your project, then using VBA to set the calculated fields from the DUOS.

In the following illustration, the report layout for the RM Customer Report includes two new calculated string fields. The VBA example later in this section will use these fields to display values from a DUOS data object.

In the following example, the event procedure runs when the user prints the RM Customer Report. It updates two calculated string fields on the report (DUOSCustomerWebAddress and DUOSCustomerEMailAddress) with the data object in the customer collection. Since the new calculated fields are in the report’s additional header, this procedure uses the BeforeAH event to set the field values before data in the additional header prints.

Note that the RM Customer Report uses a field named Customer Number. This is the same field as the Customer ID field referenced in the Customer Maintenance window:

```vba
Private Sub Report_BeforeAH(ByVal Level As Integer, SuppressBand As Boolean)
    Dim CustomerCollection As DUOSObjects
    Dim CustomerObject As DUOSObject

    'Return the customer collection
    Set CustomerCollection = DUOSObjectsGet("CustomerCollection")
    Set CustomerObject = CustomerCollection(CustomerNumber)

    'Update the report fields
    DUOSCustomerWebAddress = CustomerObject.Properties("URL Address")
    DUOSCustomerEMailAddress = CustomerObject.Properties(_
    ("Contact E-Mail Address")
End Sub
```
Chapter 7: Distributing a Project

After you’ve developed and tested your project, you can distribute it to individual workstations. This portion of the documentation describes issues you need to be aware of when packaging and distributing VBA projects. It contains the following sections:

- Enabling VBA
- References to external components
- Distributing project files
- Package files
- Package file import/export issues

Enabling VBA

To use VBA, an installation of Microsoft Dynamics GP must have the appropriate registration keys. These registration keys must provide access to one of the following:

- The Modifier with VBA
- The Customization Site Enabler

Refer to the System documentation for Microsoft Dynamics GP to learn more about entering registration keys. You can obtain registration keys for the Customization Site Enabler module from Microsoft.

References to external components

You can reference external components, such as ActiveX controls, from within your VBA project. When you distribute the VBA customization to other workstations, be sure that any additional components you have used are also installed.

If you will be distributing other components with your VBA customization, be sure you aren’t violating the license agreement by distributing those components.

To avoid problems referencing external components from your VBA project, do the following:

- Refer only to components you need. Don’t make references to components that aren’t being used in your project.
- Watch for “hidden” references, such as those created when you add a control to the Visual Basic toolbox.
- Be sure you have installed and registered needed components before running the VBA project.
- Keep external components in known locations, such as next to the VBA project or in the Windows “system” directory.
Distributing project files

One method of distributing a finished VBA customization is to copy the VBA project file to the destination machine. The primary reason to use this method is when you have created an extensive VBA project that contains proprietary code you don’t want others to see. You can lock the VBA project before you distribute it, preventing others from examining your code. Do this by viewing the properties for the project and displaying the Protection tab.

There are disadvantages to locking a VBA project:

- Other VBA developer’s customizations can’t be added to the VBA project.
- End-users can’t add any of their own functionality with VBA.

Also, distributing an entire VBA project doesn’t allow you to package any modified forms or modified reports that may be part of your customization. For all these reasons, we recommend that you use package files to distribute your VBA customizations.

Package files

Package files are special text files used to deliver customizations made with the Modifier, VBA, and the Report Writer. A developer can create a package file containing their customizations, move the package file to the destination workstation, then import the customizations into the installation.

Package file contents
A package file can contain the following items:

**Modified forms**  Forms that have been customized with the Modifier.

**New or Modified reports**  Reports that have been created or customized with the Report Writer.

**VBA forms**  Forms that have VBA code attached.

**VBA reports**  Reports that have VBA code attached.

**VBA components**  User forms, code modules or class modules created with the VBA development environment.

**References**  References to external components used by VBA. A package file cannot contain the actual components referenced by the VBA customization. Those components must be delivered separately.
Exporting a package file

Package files are created using the Customization Maintenance window. Complete the following procedure to create a package file.

1. **Open the Customization Maintenance window.**
   
   Open this window by pointing to Customize in the Tools menu and choosing Customization Maintenance.

2. **Select the components needed for the customization.**
   
   The Customization Maintenance window lists all of the components that have been customized with the Modifier, Report Writer or VBA. Select all of the components that are required for your customization.

   To select non-contiguous items in the list, hold down the CTRL key and click the items.

   Click Export. A file dialog will appear, allowing you to specify the name of the package file. Be sure the file has the .package extension. The results of the export operation will be displayed in the status area at the bottom of the window.

   **Note that some global changes you make with the Modifier and Report Writer can’t be included in a package file. For instance, changes to picture resources or global data types won’t be included in package files.**
**Importing a package file**

To import the contents of a package file, complete the following procedure.

1. **Open the Customization Maintenance window.**
   
   Open this window by pointing to Customize in the Tools menu and choosing Customization Maintenance.

2. **Open the Import Package File window.**
   
   Click Import in the Customization Maintenance window to display the Import Package File window.

3. **Select the package to import.**
   
   Click Browse to open a file dialog that allows you to select the package file you want to import. The contents of the package will be displayed in the Import Package File window.

   ![Import Package File](image)

   An “X” in the Overwrite column indicates that a customized version of a component already exists for the system, and will be overwritten by the contents of the package file. Refer to Package file import/export issues on page 91 for details about overwriting existing customizations.

   If any of the package file items contain VBA code, the user importing them must have appropriate user privileges to update the .vba files in the Microsoft Dynamics GP installation. Typically, this means being part of the Administrators group or the Power Users group. If User Account Control (UAC) is active, it means launching Microsoft Dynamics GP with Administrative privileges.

   Click OK to start the import process. If any errors occur during the import process, the Errors window will be displayed. The results of the import operation will also be displayed in the status bar of the Customization Maintenance window. To view the list of the last errors that occurred, click the Errors button in the status bar.

4. **Set access to any modified forms or reports that are part of the customization.**
   
   If necessary, use the security features in Microsoft Dynamics GP to grant access to any modified forms or modified reports that are part of your customization.
Package file import/export issues

There are some additional issues you need to be aware of when importing and exporting package files, concerning how customized resources are stored by Microsoft Dynamics GP.

Microsoft Dynamics GP contains forms, which are groups of windows, menus and other resources that work together for a common purpose. A form can have several windows, but you can make customizations to individual windows with the Modifier and VBA. When you export a form to a package file, you are exporting all of the windows in that form, not just the windows you modified or applied VBA code to.

This fact is important to keep in mind when you import a package file that contains customized forms. If a customized version of the form already exists in the system and you import another set of customizations for that same form, the original customizations will be overwritten. This occurs even if the customizations are made for different windows in the form.

A similar issue occurs for reports. If you’ve made customizations to a report, then import a package that contains customizations for that same report, the existing customizations will be overwritten.

A warning message is automatically displayed allowing you to cancel an import operation that will overwrite existing customizations.
Part 2: VBA Object Library

This part describes the objects that Microsoft Dynamics GP makes available to VBA. It provides detailed descriptions of the events, properties and methods for each object. The following topics are discussed:

- Chapter 8, “Window Object,” describes the events, properties and methods for window objects.
- Chapter 9, “Field Object,” describes the events, properties and methods for window and report fields.
- Chapter 10, “Grid Object,” describes the events and properties for grids (scrolling windows).
- Chapter 11, “Report Object,” describes the events, properties and methods for reports.
- Chapter 12, “VBAGlobal Object,” describes the global methods available for the Dynamic User Object Store (DUOS).
- Chapter 13, “DUOSObjects Collection,” describes the properties and methods for a collection in the DUOS.
- Chapter 14, “DUOSObject Object,” describes the properties for a data object in the DUOS.
- Chapter 15, “DUOSPProperties Collection,” describes the properties and methods for a property collection in the DUOS.
- Chapter 16, “DUOSPProperty Object,” describes the properties for a property object in the DUOS.
- Chapter 17, “UserInfo Object,” describes the properties and method for retrieving information about the current user.

The syntax descriptions and programming style used in this part are the same as those used in VBA’s documentation.
Chapter 8: Window Object

Your project uses the window object to perform several tasks related to windows. The events, methods and properties that apply to the window object are listed below. A detailed explanation of each follows.

- **Activate method**
- **AfterActivate event**
- **AfterClose event**
- **AfterModalDialog event**
- **AfterOpen event**
- **BeforeActivate event**
- **BeforeClose event**
- **BeforeModalDialog event**
- **BeforeOpen event**
- **Caption property**
- **Changed property**
- **Close method**
- **EventMode property**
- **Height property**
- **Hide method**
- **IsLoaded property**
- **Left property**
- **Move method**
- **Name property**
- **Open method**
- **PullFocus method**
- **Required property**
- **Show method**
- **Top property**
- **Visible property**
- **Width property**
### Activate method

<table>
<thead>
<tr>
<th>Description</th>
<th>The <strong>Activate method</strong> activates a window, making it the frontmost.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax</td>
<td><code>window.Activate</code></td>
</tr>
<tr>
<td>Parameters</td>
<td>• <code>window</code> – A window object.</td>
</tr>
<tr>
<td>Comments</td>
<td>If the window is minimized, the <strong>Activate method</strong> expands the window.</td>
</tr>
<tr>
<td></td>
<td>The <strong>BeforeActivate</strong> and <strong>AfterActivate</strong> events occur when you activate a window.</td>
</tr>
<tr>
<td>Examples</td>
<td>This example uses the Activate method to activate the Customer Maintenance window:</td>
</tr>
<tr>
<td></td>
<td><code>CustomerMaintenance.Activate</code></td>
</tr>
<tr>
<td>Related items</td>
<td><strong>Events</strong></td>
</tr>
<tr>
<td></td>
<td>The window object's <strong>BeforeActivate event</strong> and <strong>AfterActivate event</strong>.</td>
</tr>
</tbody>
</table>
# AfterActivate event

**Description**
The **AfterActivate** event occurs when a window is activated. It occurs *after* the Microsoft Dynamics GP window activate event.

**Syntax**
```
window_AfterActivate()
```

**Parameters**
- `window` – A window object.

**Comments**
A window is activated when it’s moved from the background to the foreground in an application, becoming the frontmost window.

*Microsoft Dynamics GP rarely uses its window activate event. Whether you choose to use the **AfterActivate** event or **BeforeActivate** event is of little consequence.*

**Examples**
The following **AfterActivate** event procedure runs when the Customers And Prospects lookup window is activated. As the user browses through records in the Customer Maintenance window, the lookup window remains open, but deactivated. When the user activates the lookup window, this procedure places the focus on the same record in the lookup window as was displayed in the Customer Maintenance window:

```vba
Private Sub Window_BeforeActivate()
    'Set the lookup sort list to item 2, "by Name"
    CustomerSortBy = 2

    'Open the find dialog and make it invisible
    FindDialog.Open
    Dynamics.FindDialog.Visible = False

    'Set the sort by list and locate the customer record displayed in
    'the Customer Maintenance window
    FindDialog.CustomerSortBy = 2
    FindDialog.SearchText = CustomerMaintenance.Name1

    'Press the find button
    FindDialog.Find = 1
End Sub
```

**Related items**
- **Methods and properties**
The window object's **Activate** method.
**AfterClose event**

**Description**  The AfterClose event occurs when the window closes. It occurs after the Microsoft Dynamics GP window close event.

**Syntax**  
```
window_AfterClose()
```

**Parameters**  
- `window` – A window object.

**Comments**  Microsoft Dynamics GP typically uses the window close event to check for any unsaved changes, or whether the user entered all required fields in the window, and display the appropriate dialog.

**Examples**  In this example, the AfterClose event closes a lookup window after the Customer Maintenance window closes:

```vbnet
Private Sub CustomerMaintenance_AfterClose()
    If CustomersAndProspects.IsLoaded = True Then
        CustomersAndProspects.Close
    End If
End Sub
```

**Related items**  **Methods and properties**  
The window object's Close method.
AfterModalDialog event

Description
The AfterModalDialog event occurs after the user dismisses a modal dialog.

Syntax

```
window.AfterModalDialog((ByVal DlgType As Long, PromptString As String, Control1String As String, Control2String As String, Control3String As String, Answer As Long)
```

Parameters

- `window` – A window object.
- `ByVal DlgType As Long` – Returns the type of dialog displayed. The following constants specify the dialog type:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dtModalAskDialog</td>
<td>An ask dialog. It contains one or more buttons, and displays message text.</td>
</tr>
<tr>
<td>dtModalGetStringDialog</td>
<td>A getstring dialog. It contains a data entry field and OK and Cancel buttons.</td>
</tr>
</tbody>
</table>

- `PromptString As String` – The message text. Since multiple dialogs can appear for a single window, use the `PromptString` parameter to ascertain which modal dialog appeared.

- `Control1String As String...Control3String As String` – Returns the corresponding button text in the modal dialog. For getstring dialogs, `Control3String` returns the text the user entered in the data entry field.

- `Answer As Long` – A constant specifying the button the user clicked:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dcButton1</td>
<td>The first button in an ask dialog.</td>
</tr>
<tr>
<td>dcButton2</td>
<td>The second button in an ask dialog.</td>
</tr>
<tr>
<td>dcButton3</td>
<td>The third button in an ask dialog.</td>
</tr>
<tr>
<td>dcCancel</td>
<td>The Cancel button in a getstring dialog.</td>
</tr>
<tr>
<td>dcOK</td>
<td>The OK button in a getstring dialog.</td>
</tr>
</tbody>
</table>

Comments

The most common use of the AfterModalDialog event is when saving or deleting data objects from the Dynamic User Object Store (DUOS). For instance, if the user clicks the Delete button in the Customer Maintenance window, a modal dialog will appear, asking if the user wants to delete the customer record. Using the AfterModalDialog event, you can find out whether the user clicked the dialog’s Delete button. If so, you can delete the corresponding DUOS data object for that customer record.

The AfterModalDialog event won’t occur for all dialogs displayed in Microsoft Dynamics GP. Dialogs that display a help button and whose title is “Microsoft Dynamics GP” will function like standard window objects in your project, and will not invoke modal dialog events when displayed.

Ask dialogs

The dtModalAskDialog constant of the DlgType parameter specifies an ask dialog. An ask dialog displays one or more buttons and message text. The following illustration shows an ask dialog and the corresponding parameters in the AfterModalDialog event:
The `Answer` parameter of the `AfterModalDialog` event returns which button the user clicked: the first, second or third button in the dialog (represented by the constants `dcButton1`, `dcButton2` and `dcButton3`).

### Getstring dialogs
The `dtModalGetStringDialog` constant of the `DlgType` parameter specifies a getstring dialog. Getstring dialogs typically contain a message, a data entry field, and OK and Cancel buttons:

The `Answer` parameter of the `AfterModalDialog` event returns which button the user clicked: OK or Cancel (represented by the constants `dcCancel` and `dcOK`).

### Examples
This procedure runs when the accounting system displays an ask dialog that prompts the user to save changes to a customer record in the Customer Maintenance window. If the user clicks the Save button in the save dialog, this procedure saves the corresponding DUOS object:

```vba
Private Sub Window_AfterModalDialog(ByVal DlgType As Long, _
    PromptString As String, Control1String As String, _
    Control2String As String, Control3String As String, Answer As Long)
Dim Customers As DUOSObject
Dim Customer As DUOSObject

If PromptString = "Do you want to save changes to this " + _
    "customer?" Then
    'The user is trying to save the record using the save dialog
    If Answer = dcButton1 Then 'The user clicked Save
        Set Customers = DUOSObjectsGet("Customer Info")
        Set Customer = Customers(CustomerID)
        Customer.Properties("URL Address") = URLAddress
        Customer.Properties("Contact E-Mail Address") = _
            ContactEMailAddress
    End If
End If
End Sub
```
## AfterOpen event

**Description**  
The **AfterOpen** event occurs when a window opens. It occurs after the Microsoft Dynamics GP window open event.

**Syntax**  
`window_AfterOpen()`

**Parameters**  
- `window` – A window object.

**Comments**  
Microsoft Dynamics GP uses the window open event to set default field values in the window, such as a default document date or a sort list selection.

Since the **AfterOpen** event occurs after the Microsoft Dynamics GP open event, use this event to override any default values that the accounting system sets for window fields.

**Examples**  
In this example, the **AfterOpen** event procedure for the Sales Territory Maintenance window marks two check box controls when the window opens, overriding the default values that the accounting system sets in its open event:

```vba
Private Sub Window_AfterOpen()
    'Set both check boxes when the window opens
    MaintainHistory = 1
    MaintainHistory1 = 1
End Sub
```

**Related items**  
**Methods and properties**  
The window object's **Open method**.
**BeforeActivate event**

**Description**  
The **BeforeActivate** event occurs when a window is activated. It occurs *before* the Microsoft Dynamics GP window activate event.

**Syntax**  
`window_BeforeActivate()`

**Parameters**  
- `window` – A window object.

**Comments**  
A window is activated when it’s moved from the background to the foreground in an application, becoming the frontmost window.

> Microsoft Dynamics GP rarely uses the window activate event. Whether you choose to use the **AfterActivate** event or **BeforeActivate** event is of little consequence.

**Examples**  
See the example for the **AfterActivate** event.

**Related items**  
**Methods and properties**  
The window object's **Activate method**.
**BeforeClose event**

**Description**
The **BeforeClose** event occurs when the window closes. It occurs *before* the Microsoft Dynamics GP window close event.

**Syntax**

```
window_BeforeClose(AbortClose As Boolean)
```

**Parameters**

- `window` – A window object.
- `AbortClose As Boolean` – If True, this parameter stops the window from closing, and stops the Microsoft Dynamics GP window close event from occurring.

**Comments**
Use this event is to cancel the window close if the contents in the window don’t meet certain criteria you’ve defined.

**Examples**
In the following example, the **BeforeClose** event procedure for the Invoice Entry window checks whether the user entered a comment ID. If not, it displays a message dialog using VBA’s `MsgBox()` function. If the user clicks Yes, the procedure cancels the close using the `AbortClose` parameter, and the user can enter a comment ID:

```vba
Private Sub Window_BeforeClose(AbortClose As Boolean)
    Dim Response As Integer
    If CommentID.Empty = True Then
        Response = MsgBox("Do you want to enter a Comment?", vbYesNo)
        If Response = vbYes Then
            AbortClose = True
            CommentID.Focus
        End If
    End If
End Sub
```

**Related items**

**Methods and properties**
The window object's **Close method**.
BeforeModalDialog event

Description
The BeforeModalDialog event occurs when a modal dialog opens, but before it’s displayed.

Syntax
```
window_BeforeModalDialog(ByVal DlgType As Long, PromptString As String,
Control1String As String, Control2String As String, Control3String As String, Answer As Long)
```

Parameters
- `window` – A window object.
- `ByVal DlgType As Long` – Returns the dialog type. The following constants specify the dialog type:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dtModalAskDialog</td>
<td>An ask dialog. It contains one or more buttons, and displays message text.</td>
</tr>
<tr>
<td>dtModalGetStringDialog</td>
<td>A getstring dialog. It contains a data entry field and OK and Cancel buttons.</td>
</tr>
</tbody>
</table>

- `PromptString As String` – The message’s text. Since multiple dialogs can appear for a single window, use the `PromptString` parameter to ascertain which modal dialog appeared. You can also change the `PromptString` text before Microsoft Dynamics GP displays the dialog.

- `Control1String As String...Control3String As String` – Returns the corresponding button text in the modal dialog. You can also change the button’s text before Microsoft Dynamics GP displays the dialog. For getstring dialogs, use the `Control3String` to programmatically enter the text in the data entry field.

- `Answer As Long` – A constant that specifies the buttons in the dialog:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dcButton1</td>
<td>The first button in an ask dialog.</td>
</tr>
<tr>
<td>dcButton2</td>
<td>The second button in an ask dialog.</td>
</tr>
<tr>
<td>dcButton3</td>
<td>The third button in an ask dialog.</td>
</tr>
<tr>
<td>dcCancel</td>
<td>The Cancel button in a getstring dialog.</td>
</tr>
<tr>
<td>dcOK</td>
<td>The OK button in a getstring dialog.</td>
</tr>
</tbody>
</table>

Comments
Since the BeforeModalDialog event occurs before Microsoft Dynamics GP displays the modal dialog, you can change the contents of the modal dialog. To change the message text, set the `PromptString` parameter. To change the button text, set the `Control1String, Control2String` and `Control3String` parameters.

⚠️ The AfterModalDialog event won’t occur for all dialogs displayed in Microsoft Dynamics GP. Dialogs that display a help button and whose title is “Microsoft Dynamics GP” will function like standard window objects in your project, and will not invoke modal dialog events when displayed.

Ask dialogs
The `dtModalAskDialog` constant of the `DlgType` parameter specifies an ask dialog. Ask dialogs display one or more buttons and message text. The following illustration shows an ask dialog and the corresponding parameters in the BeforeModalDialog event:
The Answer parameter of the BeforeModalDialog event lets you programmatically “click” a button, and dismiss the dialog automatically. To answer the ask dialog, set the Answer parameter to dcButton1 (clicks the first button), dcButton2 (the second button) or dcButton3 (the third button).

Getstring dialogs
The dtModalGetStringDialog constant of the DlgType parameter specifies a getstring dialog. Getstring dialogs typically contain a message, a data entry field, and OK and Cancel buttons:

The Answer parameter of the BeforeModalDialog event lets you programmatically “click” a button and dismiss the dialog automatically. To answer the getstring dialog, set the Answer parameter to dcCancel or dcOK. Prior to answering the dialog, you can also set the Control3String data entry field.

Examples
In this example, the BeforeModalDialog event procedure programmatically answers a dialog for the user. The dialog opens when the user enters a nonexistent shipping method. Before the accounting system can display this dialog, the procedure automatically “clicks” the Add button, allowing the user to add the new shipping method:

Private Sub Window_BeforeModalDialog(ByVal DlgType As Long, _ PromptString As String, Control1String As String, Control2String As String, Control3String As String, Answer As Long)
    If PromptString = "Do you want add this Shipping Method?" Then
        'Click the first button, the Add button
        Answer = dcButton1
    End If
End Sub
BeforeOpen event

**Description**

The **BeforeOpen** event occurs when a window opens. It occurs *before* the Microsoft Dynamics GP window open event.

**Syntax**

```
window_BeforeOpen(OpenVisible As Boolean)
```

**Parameters**

- **window** – A window object.
- **OpenVisible As Boolean** – If True, the window opens invisibly.

**Comments**

Microsoft Dynamics GP typically uses the window open event to set default field values in the window, such as a default document date or a sort list selection. You can use this event to set default field values in the window, while still allowing default values set by the accounting system to override your defaults.

**Examples**

The following event procedure for the Customer Maintenance window checks the **IsLoaded property** to ascertain whether the Reminders window is open. If it is, the procedure sets the opening position of the Customer Maintenance window so it appears to the right of the Reminders window:

```vba
Private Sub Window_BeforeOpen(OpenVisible As Boolean)
    If Reminders.IsLoaded = True Then
        'The Reminders window is open
        CustomerMaintenance.Left = Reminders.Left + 50
        CustomerMaintenance.Top = Reminders.Top + 50
    End If
End Sub
```

**Related items**

**Methods and properties**

The window object's **Open method**.

106  VBA DEVELOPER’S GUIDE
Caption property

<table>
<thead>
<tr>
<th>Description</th>
<th>The Caption property specifies the window’s title that’s displayed to the user.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax</td>
<td><code>window.Caption [= string]</code></td>
</tr>
<tr>
<td>Parameters</td>
<td>• <code>window</code> – A window object.</td>
</tr>
<tr>
<td></td>
<td>• <code>string</code> – The window’s caption.</td>
</tr>
<tr>
<td>Comments</td>
<td>Captions for data entry windows cannot exceed 80 characters. You cannot change the caption for modal dialogs. Windows modified using the Modifier include a period (.) at the beginning of the window’s caption. Windows that invoke VBA events include a period at the end of the window’s caption. These are visual indicators that help technical support ascertain which windows you’ve modified. We recommend you leave the period in the window’s title.</td>
</tr>
<tr>
<td>Examples</td>
<td>The following example sets the title when the window opens:</td>
</tr>
<tr>
<td></td>
<td><code>Private Sub Window_BeforeOpen()</code></td>
</tr>
<tr>
<td></td>
<td>  <code>CustomerMaintenance.Caption = &quot;Clients&quot;</code></td>
</tr>
<tr>
<td></td>
<td><code>End Sub</code></td>
</tr>
</tbody>
</table>
| Related items | Methods and properties  
|              | The field object's Caption property.                                            |
**Changed property**

**Description**
The Changed property indicates whether any non-button field values in the window have changed.

**Syntax**

`window.Changed [= boolean]`

**Parameters**

- `window` – A window object.
- `boolean` – If True, a non-button field value in the window changed.

**Comments**

A field’s value changes when the user makes an entry in a field, or when the field’s value changes using VBA. In both of these instances, the Changed property will return True for the current window.

The Changed property is useful for determining whether the contents of a window changed, and performing an action based on those changes.

The Changed event for a field always runs whenever a new record is displayed in a window (such as when the user is browsing records using browse buttons). However, only the window’s Changed property ascertains whether the user actually modified the contents displayed in the window.

**Examples**
The following example displays a message containing the date and time when a customer record was changed. This information could also be added to the DUOS or to another database (such as Microsoft Access) and stored with the customer record.

```vba
Private Sub CustomerID_Changed()
    If CustomerMaintenance.Changed Then
        'The user made a change to the current record.
        MsgBox "This user record was last changed on " & Date & " at " & Time & "."
    End If
End Sub
```

**Related items**

**Methods and properties**
The grid object’s Changed property.
Close method

Description
The Close method closes an open window.

Syntax
window.Close

Parameters
- window – A window object.

Comments
Use the Close method to close any open window in Microsoft Dynamics GP (a window whose IsLoaded property is True). An open window may not necessarily be visible.

The Close method invokes the VBA BeforeClose event and AfterClose event, and the Microsoft Dynamics GP close event for the window.

Examples
The following example uses the BeforeClose event to close the Customers and Prospects lookup window when the Customer Maintenance window closes:

Private Sub Window_BeforeClose()
    If CustomersandProspects.IsLoaded = True Then
        CustomersandProspects.Close
    End If
End Sub

Related items
Methods and properties
The window object's Open method, Activate method, BeforeClose event and AfterClose event.
**EventMode property**

**Description**
The EventMode property specifies whether VBA window events occur for the original or modified version of the window.

**Syntax**

```
window.EventMode [= mode]
```

**Parameters**

- `window` – A window object.
- `mode` – A constant specifying when events occur:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>emOriginalOnly</td>
<td>Window events occur only for the original version of the window.</td>
</tr>
<tr>
<td>emModifiedOnly</td>
<td>Window events occur only for the modified version of the window.</td>
</tr>
<tr>
<td>emNever</td>
<td>Window events don’t occur for the window.</td>
</tr>
</tbody>
</table>

**Comments**

If you modify a window using the Modifier, VBA window events will occur for the modified window only if you set the EventMode property to `emModified`. To change the EventMode property for the window object, use the Visual Basic Properties window.

If you modify a window containing a grid (such as a lookup window), you must change the EventMode property to `emModifiedOnly` for both the window and grid objects. Window and grid events can then occur for both objects.

Windows modified using the Modifier include a period (.) at the beginning of the window’s caption. Windows that invoke VBA events include a period at the end of the window’s caption.

**Examples**

Typically, you set the window’s EventMode property using the Visual Basic Properties window. You can also set it through an event procedure, as shown in the following example:

```vba
Private Sub Window_BeforeOpen(OpenVisible As Boolean)
    If CustomerMaintenance.Caption = ".CustomerMaintenance" Then
        'This is a modified window. Shut off VBA events for this user
        CustomerMaintenance.EventMode = emOriginalOnly
    End If
End Sub
```

**Related items**

Methods and properties

- The grid object’s EventMode property.
- The report object’s EventMode property.
**Height property**

**Description**
The **Height property** specifies the height of the window (in pixels).

**Syntax**

```
window.Height [= integer]
```

**Parameters**

- `window` – A window object.
- `integer` – The height of the window’s client area, measured in pixels.

**Comments**

The **Height property** specifies only the space within the window’s client area. The client area is the window less the window’s title bar.

**Examples**

This example uses the **Height property** to set the height of the Invoice Entry window:

```
Private Sub Window_BeforeOpen
    'Set the height to 500 pixels
    InvoiceEntry.Height = 500
End Sub
```

**Related items**

Methods and properties

The window object’s **Top property**, **Left property** and **Width property**.

---

**VBA DEVELOPER’S GUIDE 111**
**Hide method**

**Description**

The **Hide method** makes an open window invisible.

**Syntax**

`window.Hide`

**Parameters**

- `window` – A window object.

**Comments**

Use the **Hide method** to make any open window invisible. While invisible, the window is open (its **IsLoaded property** is True) and data in the window is still accessible. Use this method when you want to reference data displayed in a window without making the window visible. You can also make a window invisible using its **Visible property**, or by using the **OpenVisible** parameter of the window’s **BeforeOpen** event.

**Examples**

The following example compares a transaction amount a user enters in the Receivables Transaction Entry window with the batch limit displayed in an invisible Receivables Batch Entry window:

```vba
Private Sub SalesAmount_BeforeUserChanged(KeepFocus As Boolean, _
CancelLogic As Boolean)
    If BatchID.Empty = False Then
        'Open the window and make it invisible
        ExpansionButtons = 1
        ReceivablesBatchEntry.Hide
        'Compare the batch total to the transaction amount entered
        If CCur(SalesAmount) > CCur(ReceivablesBatchEntry .BatchTotal) Then
            MsgBox "Amount exceeds batch limit."
            'Clear the batch ID field and move the focus there
            BatchID.Empty = True
            BatchID.Focus
        End If
        'Close the window
        ReceivablesBatchEntry.Close
    End If
End Sub
```

**Related items**

Methods and properties

The window object’s **Visible property**, **IsLoaded property**, **Show method** and **BeforeOpen event**.
IsLoaded property

Description
The IsLoaded property returns whether the window is loaded (opened). A window can be loaded, but not visible.

Syntax
```
window.IsLoaded[= boolean]
```

Parameters
- `window` – A window object.
- `boolean` – If True, the window is loaded.

Comments
Use the IsLoaded property when you want to be sure a window is available, either visibly or invisibly. If you want to check only for the visible version of a window, use the window's Visible property.

When you open a window, Microsoft Dynamics GP implicitly open all the windows in the form invisibly, then displays the first window in the form (the parent window). The child windows in the form remain invisible, but loaded. You can use the Show method to selectively display these child windows without explicitly opening them.

Examples
The following event procedure runs when the Customer Maintenance window closes. It closes the Customers And Prospects lookup window if it’s loaded:

```
Private Sub CustomerMaintenance_AfterClose()
    If CustomersAndProspects.IsLoaded = True Then
        CustomersAndProspects.Close
    End If
End Sub
```

Related items
Methods and properties
The window object's Open method and Visible property.
Left property

Description
The Left property specifies the horizontal position of the window (in pixels).

Syntax
window.Left [= integer]

Parameters
• window – A window object.

• integer – The distance the window’s left border is from the left border of the main Microsoft Dynamics GP window, measured in pixels.

Examples
In the following example, the event procedure uses the Left property to open the Customer Maintenance window at a specified position:

Private Sub Window_BeforeOpen(OpenVisible As Boolean)
    CustomerMaintenance.Left = 100
    CustomerMaintenance.Top = 100
End Sub

Related items
Methods and properties
The window object’s Height property, Top property and Width property.
Move method

**Description**

The **Move method** changes the position of a window.

**Syntax**

\[\text{window}\.\text{Move}([\text{left}, \text{top}, \text{width}, \text{height}])\]

**Parameters**

- **window** – A window object.
- **left** – An integer specifying the distance (in pixels) from the left edge of the window to the left edge of the application border.
- **top** – An integer specifying the distance (in pixels) from the top edge of the main Dynamics GP window’s client area. The client area is the window area less the window’s title bar.
- **width** – An integer specifying the width (in pixels) of the window.
- **height** – An integer specifying the height (in pixels) of the window’s client area.

**Comments**

You can use named arguments for the **Move method**, or specify arguments by position. If you use positional arguments, enter each in the order shown, using commas and empty string values (“””) to indicate the relative position of arguments you don’t specify.

**Examples**

The following example uses named arguments to move the left position of and the top position of the window:

```vba
Private Sub Window_BeforeOpen(OpenVisible As Boolean)
    InvoiceEntry.Move Left:=200, Top:=150
End Sub
```

The following example positions the Sales Prospects Maintenance window using positional arguments. It sets only the first two parameters:

```vba
Private Sub Window_BeforeOpen(OpenVisible As Boolean)
    CustomersandProspects.Move 200,200,"","
End Sub
```

**Related items**

**Methods and properties**

The window object’s **Left property**, **Height property**, **Top property** and **Width property**. The field object’s **Move method**.
Name property

**Description**  
The Name property specifies the internal name VBA uses to reference a window.

**Syntax**  
`window.Name`

**Parameters**  
- `window` – A window object.

**Comments**  
You cannot set the Name property at runtime. We recommend that you change a window’s Name property using the Visual Basic Properties window. Be sure to replace all referenced occurrences of the window with the new name.

Use the Name property to change the window name when you need to resolve naming conflicts between windows and other objects in your application. Naming conflicts typically arise between names in your project and reserved words in VBA.

You can also use the Name property when you want to make an window’s name more readable in VBA code.

**Examples**  
To change a window’s Name property, use the Visual Basic Properties window.

**Related items**  
Methods and properties

- The field object’s Name property.
- The grid object’s Name property.
- The report object’s Name property.
## Open method

**Description**  
The Open method opens a window.

**Syntax**  
`window.Open`

**Parameters**  
- `window` – A window object.

**Comments**  
There are two ways of opening Microsoft Dynamics GP windows in VBA:

- Use the **Open method** to open windows that initially appear empty, with no existing record displayed.

- Programatically “click” a navigational control to open windows that display existing information, such as lookup windows. The application code attached to a navigational control (such as the lookup button) prepares the window to display existing data correctly. “Clicking” these controls ensures that this processing occurs.

If you use the **Open method** in your VBA code to open a window, but the window cannot be opened, a VBA exception will be thrown. You can handle this exception using the **On Error** statement in VBA.

**Examples**  
The following procedure opens the Receivables Transaction Entry window when using the **Open method** when the user initially logs into the accounting system.

```vba
Private Sub Window_AfterClose()
    On Error GoTo ErrorHandler
    If UserInfoGet.UserID = "LESSONUSER2" Then
        ReceivablesTransactionEntry.Open
    End If
    Exit Sub
ErrorHandler:
    MsgBox "Receivables Transaction Entry could not be opened."
    End Sub
```

The following procedure “clicks” the lookup button to display the Shipping Methods lookup window:

```vba
Private Sub ShippingMethod_AfterGotFocus()
    If ShippingMethod.Empty = True Then
        'No shipping method specified. Click the lookup button
        LookupButtons = 1
    End If
End Sub
```

**Related items**

**Methods and properties**  
The window object's **Close method, IsLoaded property, BeforeOpen event** and **AfterOpen event**.
PullFocus method

Description
The PullFocus method removes the focus from a window and specifies whether data for the currently-focused field is valid.

Syntax
window.PullFocus

Parameters
- window – A window object.

Return value
A boolean indicating whether the method succeeded in pulling the focus from the window:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>The focus was pulled from the window. Microsoft Dynamics GP field events (user changed and lost focus) verified that the focused field contained valid data.</td>
</tr>
<tr>
<td>False</td>
<td>The focus was not pulled from the window. Microsoft Dynamics GP field events determined that the focused field contained invalid data, then displayed a message dialog.</td>
</tr>
</tbody>
</table>

Comments
Use the PullFocus method with windows you’re programmatically controlling through VBA. This includes using VBA to open the window, set window fields, then close the window.

You cannot use the PullFocus method with a window where VBA window events or window field events are occurring. If you do, a VBA error will occur.

The primary use of the PullFocus method is to check whether the currently-focused field contains valid data. The accounting system verifies data in each field when the focus moves out of the field. However, when you’re controlling a window “remotely,” the focus remains in the last field you’ve set using VBA, and the accounting system cannot verify its contents. To manually verify the contents of this field, use the PullFocus method.

The PullFocus method moves the focus off the window, which runs the Microsoft Dynamics GP user changed and lost focus events for the focused field. The accounting system uses these field events to verify the contents of the field. If the field contains invalid data, a message dialog will appear and the PullFocus method will return False. If the focused field contains valid data, the PullFocus method will return True.

We recommend that you use the PullFocus method just prior to programmatically “clicking” the window’s Save button. This allows you to conditionally save the contents of the window if the focused field contains valid data.

Examples
The following procedure runs when the user clicks a button in a VBA user form. It saves the contents of the Receivables Transaction Entry window by programmatically “clicking” the Save button. In this case, the save occurs even though the user entered a non-existent shipping method in the currently focused field in the window (Shipping Method). No warning dialog appeared, and the partial shipping method wasn’t saved:

Private Sub CommandButton1_Click()
    ReceivablesTransactionEntry.Save = 1
    ReceivablesTransactionEntry.Close
End Sub
Using the **PullFocus method**, the same example can conditionally save the contents of the window only if data in the focused field is valid. In this case, when the **PullFocus method** moves the focus out of the Receivables Transaction Entry window, the Microsoft Dynamics GP user changed and lost focus events verify the currently-focused field (Shipping Method). If the field contains invalid data, the **PullFocus method** returns False, and the save isn’t performed:

```vba
Private Sub CommandButton1_Click()
    If ReceivablesTransactionEntry.PullFocus = True Then
        ReceivablesTransactionEntry.Save = 1
        ReceivablesTransactionEntry.Close
    End If
End Sub
```
## Required property

**Description**  
The Required property ascertains whether the user entered data in all fields whose Required property is True.

**Syntax**  
`window.Required [ = boolean ]`

**Parameters**  
- `window` – A window object.
- `boolean` – If True, the user entered data in all required fields.

**Comments**  
Required fields are those in which an entry is necessary to store the record properly. You can make additional fields required using the field object’s Required property.

Use the window object’s Required property when saving data objects to the Dynamic User Object Store (DUOS). Since the accounting system won’t permit the user to save a record in a window without entering all required fields, checking the Required property from VBA lets you save the DUOS data object only when the accounting system permits the save.

**Examples**  
In the following example, an event procedure for the save button saves DUOS data objects in the Customer Maintenance window only if the user entered all required fields (the Required property is True):

```vba
Private Sub Save_BeforeUserChanged(KeepFocus As Boolean, _
CancelLogic As Boolean)
    Dim CustomerCollection As DUOSObjects
    Dim CustomerObject As DUOSObject

    If CustomerMaintenance.Required = True Then
        'The user entered all required fields. Create/get the collection
        Set CustomerCollection = DUOSObjectsGet("CustomerCollection")
        'Create the object, using the customer ID as the object ID
        Set CustomerObject = CustomerCollection.Item(CustomerID)
        'Set the properties in the collection
        CustomerObject.Properties("URL Address") = URLAddress
        CustomerObject.Properties("Contact E-Mail Address") = ContactEMailAddress
    End If
End Sub
```
# Show method

**Description**

The **Show** method displays a window that’s open, but invisible.

**Syntax**

`window.Show`

**Parameters**

- `window` – A window object.

**Comments**

A window can be open, but still invisible. To check whether a window is open, but not necessarily visible, use the window’s `IsLoaded` property.

You can make a window invisible using the window’s **Visible property**, the window’s **Hide method**, or the `OpenVisible` parameter of the window’s **BeforeOpen** event.

When you open a window, the accounting system implicitly opens all the windows in the form invisibly, then displays the first window in the form (the parent window). The child windows in the form remain invisible, but loaded. You can use the **Show method** to selectively display these child windows without explicitly opening them.

**Examples**

The following example shows a window that’s open, but invisible. In this case, the invisible window (Customer Maintenance Options) is from the same form as the parent window (Customer Maintenance) from which this event procedure is run:

```vba
Private Sub Country_AfterUserChanged()
    If Country <> USA Then
        MsgBox "Please choose a currency ID."
        CustomerMaintenanceOptions.Show
        CustomerMaintenanceOptions.CurrencyID.Focus
    End If
End Sub
```

**Related items**

**Methods and properties**

The window object’s **Hide method**, **Visible property** and **BeforeOpen event**.
### Top property

**Description**  
The Top property specifies the vertical position of the window.

**Syntax**  
`window.Top [= integer]`

**Parameters**

- `window` – A window object.
- `integer` – The distance from the top of the main Microsoft Dynamics GP window’s client area, measured in pixels.

**Examples**

In the following example, the event procedure ascertains the position of the Reminders window, then uses the Top property to align the top border of the Customer Maintenance window with the top border of the Reminders window:

```vba
Private Sub Window_BeforeOpen(OpenVisible As Boolean)
    If Reminders.Visible = True Then
        'The Reminders window is open and visible
        CustomerMaintenance.Top = Reminders.Top
    End If
End Sub
```

**Related items**

Methods and properties

The window object’s **Left property**, **Height property** and **Width property**.
Visible property

**Description**
The Visible property specifies whether an open window is visible.

**Syntax**

\[\text{window}.\text{Visible} \[= \text{boolean}\]\]

**Parameters**

- `window` – A window object.
- `boolean` – If True, the window is visible. If False, the window is invisible.

**Comments**
You can make any window invisible, provided it’s open (its IsLoaded property is True). While invisible, data in the window is still accessible. Use this property when you want to reference data displayed in a window without making the window visible.

You can also make a window invisible using the Hide method or the OpenVisible parameter of the window’s BeforeOpen event.

When you open a window, the accounting system implicitly opens all the windows in the form invisibly, then displays the first window in the form (the parent window). The child windows in the form remain invisible, but loaded. You can use the Visible property to selectively display invisible child windows without explicitly opening them.

**Examples**
When the user enters a transaction amount in the Receivables Transaction Entry window, the following event procedure compares the amount with the maximum batch total in an invisible Receivables Batch Entry window:

```vba
Private Sub SalesAmount_BeforeUserChanged(KeepFocus As Boolean, _
CancelLogic As Boolean)
    If BatchID.Empty = False Then
        'Open the Batch Entry window invisibly
        ExpansionButtons = 1
        ReceivablesBatchEntry.Visible = False
        'Compare the batch total to the transaction amount entered
        If CCur(SalesAmount) > CCur(ReceivablesBatchEntry . .
            .BatchTotal) Then
            MsgBox "Amount exceeds batch limit."
            'Clear the batch ID field and move the focus there
            BatchID.Empty = True
            BatchID.Focus
        End If
    End If
    'Close the window
    ReceivablesBatchEntry.Close
End Sub
```

**Related items**
Methods and properties

The window object’s Show method, Hide method and BeforeOpen event.
## Width property

**Description**
The **Width** property specifies the width of a window (in pixels).

**Syntax**

```
window.Width [= integer]
```

**Parameters**

- `window` – A window object.
- `integer` – The width of the window, measured in pixels.

**Examples**
The following event procedure runs when the Customer Maintenance window opens. The event procedure checks if the Reminders window is open, then uses the **Width** and **Left** properties for the Reminders window to position the Customer Maintenance window to the right and below:

```vba
Private Sub Window_BeforeOpen(OpenVisible As Boolean)
    If Reminders.Visible = True Then
        'The Reminders window is open
        CustomerMaintenance.Left = Reminders.Left + 50
        CustomerMaintenance.Top = Reminders.Top + 50
    End If
End Sub
```

**Related items**
Methods and properties

The window object’s **Show method**, **Hide method** and **BeforeOpen event**.
Chapter 9: Field Object

Your project uses field objects to perform several tasks related to working with fields in windows, grids and reports. The events, methods and properties that apply to the field object are listed below. A detailed explanation of each follows:

- AfterGotFocus event
- AfterLostFocus event
- AfterUserChanged event
- BeforeGotFocus event
- BeforeLostFocus event
- BeforeUserChanged event
- Caption property
- Changed event
- Empty property
- Enabled property
- Focus method
- FocusSeg method
- Height property
- Left property
- Locked property
- Move method
- Name property
- Object property
- Parent property
- Required property
- TabStop property
- Top property
- Value property
- ValueSeg property
- Visible property
- Width property
AfterGotFocus event

Description

The AfterGotFocus event occurs when the user moves the focus into a field, using the TAB key or the mouse. It occurs after the Microsoft Dynamics GP got focus event.

Syntax

`field_AfterGotFocus()`

Parameters

- `field` – A window field object.

Comments

Use the AfterGotFocus event to set or evaluate the contents of a field gaining focus. Microsoft Dynamics GP rarely uses the got focus event. When it does, the accounting system typically checks the value of the field that’s gaining focus, and performs calculations or sets other field values based on that value.

Examples

In the following example, the AfterGotFocus event for the Batch ID field checks whether the field is empty when the user moves to it. If it is, the event procedure opens the batch lookup window:

```vba
Private Sub BatchID_AfterGotFocus()
    If BatchID.Empty = True Then
        'The field is empty. Press the lookup button
        LookupButton3 = 1
    End If
End Sub
```
AfterLostFocus event

Description
The AfterLostFocus event occurs when the user moves the focus out of a field, using the TAB key or the mouse. It occurs after the Microsoft Dynamics GP lost focus event.

Syntax
`field_AfterLostFocus()`

Parameters
- `field` – A window field object.

Comments
Use the AfterLostFocus event to evaluate the value of the field losing focus, and perform any additional operations in the window. You cannot set the value of the field losing focus using the AfterLostFocus event. This event follows the Microsoft Dynamics GP user changed event, which is typically used to verify the contents of the field.

The AfterLostFocus event occurs immediately after the Microsoft Dynamics GP lost focus event. Microsoft Dynamics GP rarely uses the lost focus event. When they do, they typically lock control fields (fields that control the display of a record).

Examples
In the following example, the AfterLostFocus event checks whether the Payment Terms field is empty. If it is, it displays a VBA dialog that asks the user if they want to enter one. If the user clicks Yes, the procedure opens the Payment Terms Lookup window:

```vba
Private Sub PaymentTerms_AfterLostFocus()
    Dim Response As Integer
    If PaymentTerms.Empty = True Then
        'Set a default payment term
        Response = MsgBox("Do you want to enter a payment term?", _
                         vbYesNo)
        If Response = vbYes Then
            'Click the payment term lookup button
            LookupButtons = 1
        End If
    End If
End Sub
```
AfterUserChanged event

Description
The AfterUserChanged event occurs when the user changes the contents of a field, then moves the focus out of the field, using the TAB key or the mouse. In fields that toggle, such as push buttons and visual switches, this event occurs when the user clicks the field. For list fields, this event occurs when the user selects an item in the list. In all instances, the AfterUserChanged event occurs after the Microsoft Dynamics GP user changed event.

Syntax
field_AfterUserChanged()

Parameters
- field – A window field object.

Comments
Use the AfterUserChanged event to evaluate the value in the field losing focus. You cannot set the value of the field losing focus using the AfterUserChanged event. This event occurs after the Microsoft Dynamics GP user changed event, which is used to verify the contents of the field.

Examples
The following example uses the AfterUserChanged event to check the value of the Customer ID field in the Invoice Entry window. It disables the Trade Discount field:

```vba
Private Sub CustomerID_AfterUserChanged()
    If CustomerID = "ADVANCED0002" Then
        'Don't offer a trade discount
        TradeDiscount.Enabled = False
    End If
End Sub
```
BeforeGotFocus event

**Description**  
The BeforeGotFocus event occurs when the user moves the focus to a field, using the TAB key or the mouse. It occurs before the Microsoft Dynamics GP got focus event.

**Syntax**  
`field_BeforeGotFocus(CancelLogic As Boolean)`

- `field` – A window field object.
- `CancelLogic As Boolean` – If True, the event cancels the Microsoft Dynamics GP got focus event, and the VBA AfterGotFocus event.

**Comments**  
Use the BeforeGotFocus event to set or evaluate the contents of the field gaining focus, and cancel any further got focus events for the field from running (the Microsoft Dynamics GP got focus event and the VBA AfterGotFocus event).

If Microsoft Dynamics GP uses its got focus event, it typically checks the value of the field that’s gaining focus, and performs calculations or sets other field values based on that value. However, the accounting system uses this event infrequently to do such operations.

**Examples**  
In the following example, the procedure checks whether the document type is a return. If so, this procedure cancels the VBA AfterGotFocus event procedure, which automatically opens a lookup window for the Batch ID field:

```vbnet
Private Sub BatchID_BeforeGotFocus(CancelLogic As Boolean)
    If DocumentType = 7 Then
        'The document type is a return. Don’t use a batch
        BatchID.Empty = True
        'Cancel the VBA AfterGotFocus event, which opens a lookup
        CancelLogic = True
        MsgBox "Post returns individually, not in a batch."
        DocumentDate.Focus
    End If
End Sub
```
BeforeLostFocus event

Description

The BeforeLostFocus event occurs when the user moves the focus out of a field, using the TAB key or the mouse. It occurs before the Microsoft Dynamics GP lost focus event.

Syntax

`field_BeforeLostFocus(KeepFocus As Boolean, CancelLogic As Boolean)`

Parameters

- `field` – A window field object.
- `KeepFocus As Boolean` – If True, the focus returns to the current field.
- `CancelLogic As Boolean` – If True, the event cancels the Microsoft Dynamics GP lost focus event, and the VBA AfterLostFocus event.

Comments

Use the BeforeLostFocus event to evaluate the value of the field losing focus, and perform any additional operations in the window. You cannot set the value of the field losing focus using the BeforeLostFocus event. This event follows the Microsoft Dynamics GP user changed event, which is used to verify the contents of the field losing focus.

The BeforeLostFocus event occurs immediately before the Microsoft Dynamics GP lost focus event. The accounting system rarely uses its lost focus event. When it does use this event, it typically locks control fields (fields that control the display of a record).

Examples

In the following example, the BeforeLostFocus event procedure runs for the Number field (document number) in the Receivables Transaction Entry window. Normally, the Microsoft Dynamics GP lost focus event locks this field. In this case, the procedure cancels subsequent lost focus events, and the Number field remains unlocked and editable:

```vba
Private Sub Number_BeforeLostFocus(KeepFocus As Boolean, _
    CancelLogic As Boolean)
    CancelLogic = True
    'Move the focus to the next field
    Description.Focus = True
End Sub
```
BeforeUserChanged event

Description
The BeforeUserChanged event occurs when the user changes the contents of a field, then moves the focus out of the field, using the TAB key or the mouse. In fields that toggle, such as push buttons and visual switches, this event occurs when the user clicks the field. For list fields, this event occurs when the user selects an item in the list. In all instances, the BeforeUserChanged event occurs before the Microsoft Dynamics GP user changed event.

Syntax
field_BeforeUserChanged(KeepFocus As Boolean, CancelLogic As Boolean)

Parameters
- field – A window field object.
- KeepFocus As Boolean – If True, the focus returns to the current field.
- CancelLogic As Boolean – If True, the event cancels the Microsoft Dynamics GP user changed event, and the VBA AfterUserChanged event.

Comments
Use the BeforeUserChanged event to set or evaluate the value of field that’s losing focus. You can also use this event to bypass, or cancel, the Microsoft Dynamics GP user changed event that occurs for the field.

For many data entry fields, the Microsoft Dynamics GP user changed event verifies the value entered in the field. For push buttons, such as a Save or Delete button, the user changed event saves or deletes the record displayed in the window.

Use caution when canceling a data entry field’s user changed events using the BeforeUserChanged event. Canceling the Microsoft Dynamics GP user changed event for these fields inhibits the accounting system’s ability to verify the contents of the field.

Examples
In the following example, the BeforeUserChanged event procedure runs for the Invoice Entry window’s Trade Discount field. Using the CancelLogic parameter, it cancels processing for the Trade Discount field if the user enters an amount greater than 4% of the invoice subtotal. It also uses the KeepFocus parameter to place the focus in the same field, allowing the user to enter a lower amount:

```vba
Private Sub TradeDiscount_BeforeUserChanged(KeepFocus As Boolean, _
    CancelLogic As Boolean)
    If TradeDiscount > Subtotal * 0.04 Then
        'The trade discount is greater than 4% of the subtotal
        MsgBox "You cannot enter a discount greater than 4% of " + _
            "the subtotal."
        'Cancel the invoice calculation
        CancelLogic = True
        TradeDiscount. Empty = True
        KeepFocus = True
    End If
End Sub
```
Caption property

Description
The Caption property specifies the caption for a window field.

Syntax
field.Caption [= string]

Parameters
- field – A window field object.
- string – The caption for the field.

Comments
The following list explains the caption for each field type:

<table>
<thead>
<tr>
<th>Field type</th>
<th>Caption</th>
</tr>
</thead>
<tbody>
<tr>
<td>String, composite, currency, integer, long integer, date, time, text, list box, drop-down list, multi-select list, combo box</td>
<td>The text that’s linked to the field.</td>
</tr>
<tr>
<td>Check box</td>
<td>The text that’s linked to the check box group. If the check box does not appear in a check box group, or if there’s no caption for the check box group, the caption is the text for the individual check box.</td>
</tr>
<tr>
<td>Push button, button drop list, visual switch</td>
<td>The text appearing on the button face or switch. The Caption property has no effect on a button or visual switch with a picture.</td>
</tr>
<tr>
<td>Radio button</td>
<td>The text that’s linked to the radio button group. If there’s no caption for the radio button group, the caption is the prompt for the individual radio button.</td>
</tr>
</tbody>
</table>

If you make a caption longer than the existing caption, keep in mind that you may need to resize or rearrange fields to view the entire caption.

Examples
In the following example, the event procedure changes the caption for the Customer ID field to “Patient ID”, then changes the window’s caption:

Private Sub Window_BeforeOpen()
    'Set the field caption
    CustomerID.Caption = "Patient ID"
    'Set the window caption
    CustomerMaintenance.Caption = "Patient Maintenance"
End Sub
Changed event

**Description**
The Changed event *always* occurs when a field’s value changes.

**Syntax**
```
field_Changed()
```

**Parameters**
- `field` – A window field object.

**Comments**
The Changed event will always occur in the following instances:

- When the user changes the field directly. This also activates the field’s BeforeUserChanged and AfterUserChanged events, as well as the Microsoft Dynamics GP user changed event.

- When Microsoft Dynamics GP application code updates the window field.

- When a field’s value changes using an external tool, such as VBA.

Use the Changed event to execute VBA code for window fields that Microsoft Dynamics GP updates. This occurs most often when the accounting system retrieves a record from a table, then displays the record’s contents in the window. The AfterUserChanged and BeforeUserChanged events won’t occur for a field in this case, since the accounting system, not the user, changes the contents of the field.

*Exercise caution when using the Changed event. In many cases, Microsoft Dynamics GP causes the Changed event to occur numerous times for the same field even though the value of the field may not visibly change. This will execute the associated VBA event procedure each time the event occurs.*

**Examples**
In the following example, the Changed event procedure runs for the Document Date field. As the user browses records in the Receivables Transaction Entry window, the accounting system sets window field values for each record browsed. This procedure checks the value of the Document Date field, and displays a message if the document is older than 90 days:

```vba
Private Sub DocumentDate_Changed()
    Dim DaysOld As Integer
    If CDate(DocumentDate) < Date - 90 Then
        'The document is at least 90 days old
        DaysOld = Date - CDate(DocumentDate)
        MsgBox "This document is " + Str(DaysOld) + " days old." + _
        'Please post."
    End If
End Sub
```
Empty property

<table>
<thead>
<tr>
<th>Description</th>
<th>The Empty property specifies whether a field contains a value.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax</td>
<td>field.Empty [= boolean]</td>
</tr>
<tr>
<td>Parameters</td>
<td>• field – A window field object.</td>
</tr>
<tr>
<td></td>
<td>• boolean – If True, the field is empty. If False, the field contains a non-zero value.</td>
</tr>
<tr>
<td>Comments</td>
<td>To clear a field’s value, set the field’s Empty property to True. You cannot clear a field that the accounting system locked or disabled, but you can clear fields you’ve locked or disabled using VBA.</td>
</tr>
<tr>
<td></td>
<td>VBA considers numeric fields empty if they contain a value of 0. String fields are empty if they contain nothing or only spaces. Date fields are empty when the date is 00/00/00. Time fields are empty when the time is 12:00:00 AM.</td>
</tr>
<tr>
<td>Examples</td>
<td>The following example uses the Empty property to automatically open a lookup window if the Salesperson ID field is empty:</td>
</tr>
<tr>
<td></td>
<td>Private Sub SalespersonID_AfterGotFocus()</td>
</tr>
<tr>
<td></td>
<td>If SalespersonID.Empty = True Then</td>
</tr>
<tr>
<td></td>
<td>'Click the lookup button to display the lookup window</td>
</tr>
<tr>
<td></td>
<td>LookupButton8 = 1</td>
</tr>
<tr>
<td></td>
<td>End If</td>
</tr>
<tr>
<td></td>
<td>End Sub</td>
</tr>
</tbody>
</table>

|
### Enabled property

**Description**  
The **Enabled property** specifies whether a window field is enabled. The user can’t change the contents of a disabled field.

**Syntax**  
```vba
field.Enabled [= boolean]
```

**Parameters**  
- `field` – A window field object.
- `boolean` – If True, the field is enabled and editable. If False, the field is disabled.

**Comments**  
A disabled field appears dimmed, along with its caption, the field’s value, and any related controls.

You can set the value of a field you’ve disabled using VBA. You cannot set the value of a field disabled by the accounting system, nor can you enable the field. For most windows, Microsoft Dynamics GP application code disables and enables fields based on entries made by the user. You can always return a disabled field’s value.

Keep in mind that a disabled field is not the same as a locked field (a field whose **Locked property** is True). If True, the **Locked property** makes the field non-editable, but doesn’t dim the field’s value, prompt and related controls.

**Examples**  
In the following example, the event procedure disables a field in the Salesperson Maintenance window for a specific user:

```vba
Private Sub Window_BeforeOpen()
    If UserInfoGet.UserName = "LESSONUSER2" Then
        'Disable the Commission Percent field for this user
        Percent.Enabled = False
    End If
End Sub
```

**Related items**  
- **Methods and properties**  
The field object’s **Locked property**.
Focus method

**Description**  
The **Focus method** moves the focus to a field.

**Syntax**  
`field.Focus([setvalue])`

**Parameters**  
- **field** – A window field object.
- **setvalue** – The value you can set for the field gaining focus.

**Comments**  
Use the **Focus** method to skip unnecessary fields in a window, or place the focus in a specific field based upon a user entry. The Microsoft Dynamics GP got focus event and the VBA **BeforeGotFocus** and **AfterGotFocus** events occur when you move the focus to a field using the **Focus method**.

You cannot place the focus using a window’s **BeforeOpen** or **AfterOpen** event. By default, Microsoft Dynamics GP places the focus in the first field in the tab sequence after these events occur.

**Examples**  
In the following example, the **Focus method** redirects the focus to the Shipping Method field when the user moves out of the Customer ID field:

```vba
Private Sub CustomerID_AfterUserChanged()
    'Move the focus to the field.
    ShippingMethod.Focus
End Sub
```

Using the **setvalue** parameter, the **Focus method** redirects the focus and sets the value of the Shipping Method field:

```vba
Private Sub CustomerID_AfterUserChanged()
    'Move the focus to the Shipping method field, then set the value
    ShippingMethod.Focus("UPS BLUE")
End Sub
```

**Related items**  
**Methods and properties**  
The field object's **FocusSeg method** and **ValueSeg property**.
FocusSeg method

Description
The FocusSeg method moves the focus to a segment in a composite field. Microsoft Dynamics GP account number fields are composite fields.

Syntax
field.FocusSeg(index), [setvalue]

Parameters
- field – A composite window field object.
- index – An integer specifying the segment you’re moving the focus to.
- setvalue – The value you can set for the segment gaining focus.

Comments
Use the FocusSeg method to move the focus to a composite segment from another field in the window, or from one segment to the next within the same composite.

You can use the setvalue of the FocusSeg method to set the value of a segment gaining focus. Based on whether you’re setting the value from “inside” or “outside” the field, there are two primary ways to set segment values:

- If you set the segment’s value from “outside” the composite field (from another field’s event procedure), use the setvalue parameter of the FocusSeg method to set the segment’s value. In this case, you can set only one segment’s value.
- If you set the segment’s value from “inside” the composite field (from the composite field’s BeforeGotFocus, AfterGotFocus or BeforeUserChanged events), you can use the ValueSeg method to set values for each segment of the composite. Use the FocusSeg method to move the focus to a segment, but not set the segment’s value.

Examples
The following example sets an account number segment from “outside” the Account field (using the Checkbook ID field’s AfterUserChanged event). The procedure moves the focus to the Account field, then sets the first segment using the FocusSeg method:

Private Sub CheckbookID_AfterUserChanged()
    'Move to the Account field and set the first segment
    Account.FocusSeg(1), "100"
End Sub

The following example sets account segments from “inside” the Account field. In this case, the Account field already has the focus. The ValueSeg property sets the first two segment values. The FocusSeg method then moves the focus to the third segment:

Private Sub CashAccount_AfterGotFocus()
    If CheckbookID = "PAYROLL" Then
        'Set default payroll account segments
        CashAccount.ValueSeg(1) = "100"
        CashAccount.ValueSeg(2) = "1500"
        'Move the focus to the third segment
        CashAccount.FocusSeg(3)
    End If
End Sub

Related items
Methods and properties
The field object’s Focus method and ValueSeg property.
### Height property

**Description**

The *Height property* specifies the height of a window field (in pixels).

**Syntax**

```
field.Height [= integer]
```

**Parameters**

- `field` – A window field object.
- `integer` – The height of the field, measured in pixels.

**Comments**

For data entry fields, the default height is 20 pixels. For push buttons, such as Save and Delete, the default height is 24 pixels.

List box and multi-select lists require 20 pixels per item to display the item properly. Changing the height of drop lists and combo boxes does not change the height of the list in the “undropped” position. Instead, changing the height of these lists determines the height of the list in the “dropped” position.

**Examples**

In the following example, the event procedure changes the height of a list box field in the Account Maintenance window so that it shows two items rather than four:

```
Private Sub Window_BeforeOpen()
    'Change the list to display only two items
    IncludeinLookup.Height = 36
End Sub
```

**Related items**

*Methods and properties*

The field object's *Left property*, *Top property*, *Width property* and *Move method*. 
Left property

Description
The Left property specifies the horizontal position of a field (in pixels).

Syntax
field.Left [= integer]

Parameters
- field – A window field object.
- integer – The distance from the field’s left border to the window’s left border, measured in pixels. Your operating system’s settings for the active window border specifies the width of the border.

Comments
The Left property moves the field and the field’s caption. It does not move any buttons associated with the field, such as a lookup button. You must move those fields separately.

Setting the Left property to 0 will align the field with the left edge of the window. However, the field’s caption will move off the window area.

Examples
In the following example, the event procedure runs when the Account Maintenance window opens. It first hides the Budget button, then uses the Left property to move the Currency button to the same position as the Budget button:

Private Sub Window_BeforeOpen(OpenVisible As Boolean)
  'Hide the Budget button
  Budget.Visible = False
  'Move the Currency button to the left
  CurrencyButton.Left = Budget.Left
End Sub

Related items
Methods and properties
The field object’s Top property, Width property, Height property and Move method.
## Locked property

**Description**
The **Locked property** locks a field. The user can’t change the value of a locked field, but they can view data in the field.

**Syntax**
```vba
field.Locked [= boolean]
```

**Parameters**
- `field` – A window field object.
- `boolean` – If True, the field is locked and not editable. If False, the field is not locked.

**Comments**
A locked field appears with a light gray background, indicating that the user can’t change the contents of the field.

You can set the value of a field you’ve locked using VBA. You cannot set the value of a field locked by the accounting system, nor can you unlock a field locked by the accounting system. Typically, Microsoft Dynamics GP application code locks fields whose values are necessary to preserve the integrity of accounting data, such as document totals. You can always return a locked field’s value.

Keep in mind that a locked field is not the same as a disabled field (a field whose **Enabled property** is False). If False, the **Enabled property** makes the field non-editable, and also dims the field’s value, prompt and related controls.

**Examples**
The following event procedure runs when the Account Maintenance window opens. It locks the Account Number field for a specific user:

```vba
Private Sub Window_AfterOpen()
    If UserInfoGet.UserID = "LESSONUSER1" Then
        Account.Locked = True
    End If
End Sub
```

**Related items**
**Methods and properties**
The field object’s **Enabled property**.
## Move method

**Description**
The **Move method** moves and resizes a window field.

**Syntax**
```vba
field.Move ([Left [,Top [,Width [,Height]]]])
```

**Parameters**
- `field` – A window field object.
- `left` – An integer specifying the distance from the field’s left border to the window’s border, measured in pixels. Your operating system’s settings for the active window border determines the width of the border.
- `top` – An integer specifying the distance from the field’s top border to the top edge of the window’s client area, measured in pixels. The client area is the area of the window, less the window’s title bar.
- `width` – An integer specifying the width of the field (measured in pixels).
- `height` – An integer specifying the height of the field (measured in pixels).

**Comments**
You can use named arguments for the **Move method**, or enter arguments by position. If you use positional arguments, enter each in the order shown, using commas and empty string values (""”) to indicate the relative position of arguments you don’t specify. Any unspecified arguments remain unchanged.

**Examples**
The following example uses positional arguments to change the position of a Customer ID field when the Customer Maintenance window opens:

```vba
Private Sub Window_BeforeOpen(OpenVisible As Boolean)
    CustomerID.Move 10, 10, 50, ""
End Sub
```

The following example uses named arguments to move the left position of the field and change the width of the field:

```vba
Private Sub Window_BeforeOpen(OpenVisible As Boolean)
    'Set the left distance and the field width
    CustomerID.Move Left: = 10, Width: = 70
End Sub
```

**Related items**
- The field object’s **Left property, Top property, Width property** and **Height property**.
- The window object’s **Move method**.
Name property

Description
The Name property specifies the internal name VBA uses to reference a window field or report field object.

Syntax
field.Name

Parameters
• field – A window field or report field object.

Comments
You cannot set the Name property at runtime. We recommend that you change a field’s Name property using the Visual Basic Properties window. Be sure to replace all referenced occurrences of the field with the new name.

Use the Name property to change the field name when you need to resolve naming conflicts between fields and other objects in your application. Naming conflicts typically arise between names in your project and reserved words in VBA. For instance, if you add the Account Maintenance window’s Currency button to your project, you can’t compile VBA code referencing that field until you rename the field. “Currency” in VBA is a reserved word.

You can also use the Name property to make a field’s name more readable in VBA code.

Examples
Use the Visual Basic Properties window to change a field’s Name property.

Related items
Methods and properties
The window object’s Name property.
The report object’s Name property.
The grid object’s Name property.
Object property

Description
The Object property returns a field object without the standard properties and methods extended to it by Visual Basic.

⚠️ VBA requires host applications to define an Object property within an object model. For your applications, it’s unlikely that you’ll need to use this property.

Syntax
field.Object [property | method]

Parameters
• field – A window field or report field object.

• [property | method] – The Microsoft Dynamics GP “custom” property or method for the field object you’re referencing. This cannot be a “standard” property or method assigned to it by VBA.

Comments
Use the Object property to reference “custom” properties and methods assigned to the field object exclusively for Microsoft Dynamics GP. For field objects, custom properties and methods include:

<table>
<thead>
<tr>
<th>Method/property</th>
<th>Method/property</th>
</tr>
</thead>
<tbody>
<tr>
<td>Caption property</td>
<td>Locked property</td>
</tr>
<tr>
<td>Empty property</td>
<td>Required property</td>
</tr>
<tr>
<td>Enabled property</td>
<td>Value property (report and window fields)</td>
</tr>
<tr>
<td>FocusSeg method</td>
<td>Visible property (report and window fields)</td>
</tr>
</tbody>
</table>

In addition to custom properties and methods, VBA assigns each field object “standard” properties and methods. (VBA sometimes calls these “extender” properties and methods). In the field object library, these are the methods and properties not shown in the above table. You cannot reference VBA's standard properties and methods using the Object property.

The primary use of the Object property is to resolve naming conflicts between custom properties and methods supplied by a VBA host application for an object, and standard properties and methods supplied by VBA for the same object. If a custom property or method supplied by a VBA host application uses the same name as a standard property or method supplied by VBA, VBA automatically uses the standard property or method.

In these cases, the Object property allows you to bypass the VBA-supplied standard property or method and use the identically-named property or method defined by the host application.

⚠️ Since there are no inherent naming conflicts with VBA methods and properties for field objects, you don’t need to use this property when working with field objects.

Examples
The Object property is not relevant for field objects at this time. Refer to the Visual Basic online help for additional information and examples of the Object property.
Parent property

Description
The Parent property returns a window or report object containing a specified field object.

Syntax
field.Parent

Parameters
- field – A window field or report field object.

Comments
Use the Parent property to access the methods or properties of a window or report object containing a given field object. This is especially useful if you’re passing field objects to procedures as arguments. In these cases, you can use the Parent property to obtain information about the window or report containing the field object you passed to the procedure.

Examples
In the following example, two procedures illustrate how to use the Parent property. The first procedure uses the Call statement to call a sub procedure named FieldCaption after a Microsoft Dynamics GP window opens. It passes the window’s Customer ID field as an object to the procedure.

The second procedure is a new sub procedure defined for the module. It receives the Customer ID field object passed from the first procedure. The procedure then uses the Parent property to ascertain the name of the window where the field was passed from, and changes the field’s caption based on the window’s name:

```vba
Private Sub Window_AfterOpen()
    'Pass a window field to the FieldCaption procedure
    Call FieldCaption(CustomerID)
End Sub
```

```vba
Public Sub FieldCaption(FieldObj As Object)
    If FieldObj.Parent.Name = "CustomerMaintenance" Then
        CustomerMaintenance.CustomerID.Caption = "Customer No."
    ElseIf FieldObj.Parent.Name = "InvoiceEntry" Then
        InvoiceEntry.CustomerID.Caption = "Cus. No."
    End If
End Sub
```
Required property

**Description**
The Required property specifies whether the field must contain data before the user can save information in the window.

**Syntax**
`field.Required [= boolean]`

**Parameters**
- `field` – A window field object.
- `boolean` – If True, the field is required. If False, the field is not required.

**Comments**
Microsoft Dynamics GP will prompt the user to enter data in a required field if the field is empty when the user attempts to save the record. Fields marked as required by VBA are displayed with the same caption, font style and color as fields marked as required by the accounting system.

You cannot set the Required property to False for fields already marked as required by the accounting system. These are fields necessary for the system to store the record properly.

If you make a field required, be sure the field is enabled (its Enabled property is True) and unlocked (its Locked property is False).

**Examples**
The following example sets the Required property for the Phone 1 field in the Customer Maintenance window:

```vba
Private Sub UserDefined1_AfterUserChanged()
    If UserDefined1 = "RETAIL" Then
        Phone1.Required = True
    ElseIf UserDefined1 = "WHOLESALE" Then
        Phone1.Required = False
    End If
End Sub
```
TabStop property

Description
The TabStop property specifies whether a field is in the window’s tab sequence.

Syntax
field.TabStop [= boolean]

Parameters
- field – A field object.
- boolean – If True, the field is in the window’s tab sequence. If False, the field is not in the tab sequence.

Comments
The tab sequence is the order in which the focus moves through fields when the user presses the TAB key. Removing a field from the tab sequence bypasses the field when you press the TAB key. If you add the field back to the sequence, it maintains its original order in the sequence.

Removing a rarely-used field from the tab sequence allows users to move from field to field more efficiently. Once you remove a field from the tab sequence, the user can still place the focus in the field using the mouse. This setting doesn’t affect the user’s ability to enter data in the field.

Since you cannot place the focus in hidden, locked or disabled fields, the accounting system ignores these fields in the tab sequence.

Examples
The following example removes three fields from the tab sequence in the Invoice Entry window:

Private Sub Window_BeforeOpen(OpenVisible As Boolean)
    If UserInfoGet.UserID = "LESSONUSER2" Then
        Hold.TabStop = False
        BatchID.TabStop = False
        CustomerPONumber.TabStop = False
    End If
End Sub
# Top property

**Description**
The Top property specifies the vertical position of a field.

**Syntax**

\[ \text{field}.\text{Top} \[= \text{integer}] \]

**Parameters**

- `field` – A window field object.
- `integer` – The distance from the field’s top border to the top edge of the window’s client area, measured in pixels. The client area is the area of the window, less the window’s title bar.

**Examples**
The following example uses the Top property and Left property to move the Currency1 button field to the same window position as the invisible Budget button field:

```vba
Private Sub Window_BeforeOpen(OpenVisible As Boolean)
    Budget.Visible = False
    Currency1.Top = Budget.Top
    Currency1.Left = Budget.Left
End Sub
```

**Related items**

Methods and properties

The field object’s Left property, Width property, Height property and Move method.
Value property

Description
The **Value property** specifies the value of a window or report field. For window fields, the **Value property** is a read/write property. For report fields, the **Value property** is a read-only property unless the field is a user-defined calculated field.

Syntax
`field.Value [= string_value]`

Parameters
- `field` – A window field or report field object.
- `string_value` – A string specifying the value of the field.

Comments
If you omit the **Value property** from your code, VBA assumes you’re returning or setting the value of the field. The following example explicitly uses the **Value property**:

```vba
BatchID.Value = "DEBITMEMOS"
```

The same code can also omit the **Value property** and be written like this:

```vba
BatchID = "DEBITMEMOS"
```

The following table lists each field type for which you can set or return a value using the **Value property**, as well as a description of the string value used for the field type:

<table>
<thead>
<tr>
<th>Field type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Button drop list</td>
<td>A numeric value identifying the item selected in the list.</td>
</tr>
<tr>
<td>Check box</td>
<td>The value 0 indicating an unmarked check box. The value 1 indicating a marked check box.</td>
</tr>
<tr>
<td>Combo box</td>
<td>The text of the item selected in the combo box.</td>
</tr>
<tr>
<td>Composite</td>
<td>The value of the composite field, including any formatting.</td>
</tr>
<tr>
<td>Currency</td>
<td>A currency value for the field, including the decimal separator.</td>
</tr>
<tr>
<td>Date</td>
<td>The date value of the field, including any formatting.</td>
</tr>
<tr>
<td>Drop-down list</td>
<td>A numeric value identifying the item selected in the list.</td>
</tr>
<tr>
<td>Integer</td>
<td>The numeric value of the field, including any formatting.</td>
</tr>
<tr>
<td>List box</td>
<td>A numeric value identifying the item selected in the list.</td>
</tr>
<tr>
<td>Long integer</td>
<td>The numeric value of the field, including any formatting.</td>
</tr>
<tr>
<td>Multi-select list box</td>
<td>A 32-bit numeric value identifying the selected items in the list.</td>
</tr>
<tr>
<td>Push button</td>
<td>The value 0 indicating the “unpressed” state of the push button.</td>
</tr>
<tr>
<td></td>
<td>The value 1 indicating the “pressed” state.</td>
</tr>
<tr>
<td>Radio group</td>
<td>A numeric value identifying the radio button selected in the group.</td>
</tr>
<tr>
<td>String</td>
<td>The string value for the field, including any formatting.</td>
</tr>
<tr>
<td>Time</td>
<td>The time value for the time field, including any formatting.</td>
</tr>
<tr>
<td>Visual switch</td>
<td>A numeric value identifying the item selected in the visual switch.</td>
</tr>
</tbody>
</table>

Examples
The following example sets the Sort By list field in the Customer Maintenance window when the window opens:

```vba
Private Sub Window_AfterOpen()
    'Set the Sort By list to "By Salesperson ID"
    SortBy.Value = 5
End Sub
```
The following example sets the value of the Batch ID string field when the user chooses a document type from the Document Type drop-down list:

```vba
Private Sub DocumentType_AfterUserChanged()
    Select Case DocumentType
        'Set the batch ID to one of three existing batches
        Case 1
            BatchID.Value = "DAILYSLS"
        Case 7
            BatchID.Value = "DAILYRET"
        Case Else
            BatchID.Value = "DAILYMISC"
    End Select
End Sub
```

The following example checks the value of the Checkbook ID field, then sets the value of the Cash Account composite field to an existing account number. Note that the syntax omits the `Value property`:

```vba
Private Sub CheckbookID_Changed()
    If CheckbookID="FIRST NATIONAL" And CashAccount.Enabled=True Then
        CashAccount = "100-5100-00"
        CashAccount.Enabled = False
    ElseIf CheckbookID <> "FIRST NATIONAL" Then
        CashAccount.Enabled = True
    End If
End Sub
```

**Related items**

**Methods and properties**

The field object's `ValueSeg property` and `Empty property`.

**Additional information**

- `Working with field values` and `Window field type reference` in Chapter 3, “Programming Window Fields.”
## ValueSeg property

### Description
The **ValueSeg property** specifies the value of a segment in a composite field.

### Syntax

```
field.ValueSeg(index) = [string_value]
```

### Parameters

- **field** – A composite window field object.
- **index** – An integer specifying the segment.
- **string_value** – A string specifying the value of the segment.

### Comments
Use the **ValueSeg property** to set or return segment values after the composite gains focus (using the **BeforeGotFocus**, **AfterGotFocus** or **BeforeUserChanged** events). The composite gains focus when the user moves to the composite field, or when you programmatically move the focus using the **Focus method** or **FocusSeg method**.

### Examples
The following procedure moves the focus to the Cash Account field using the **FocusSeg method**:

```vbnet
Private Sub CheckbookID_BeforeUserChanged(KeepFocus As Boolean, CancelLogic As Boolean)
    CashAccount.FocusSeg(1)
End Sub
```

Once the Cash Account has focus, a **BeforeGotFocus** event procedure sets the value of the segments in the composite using the **ValueSeg property**:

```vbnet
Private Sub CashAccount_BeforeGotFocus(CancelLogic As Boolean)
    If CheckbookID = "CO. PAYROLL" Then
        CashAccount.ValueSeg(1) = "000"
        CashAccount.ValueSeg(2) = "21"
        CashAccount.FocusSeg(2)
    End If
End Sub
```

### Related items

**Methods and properties**
The field object's **FocusSeg method**.
### Visible property

**Description**  
The Visible property specifies whether the field is visible.

**Syntax**  
`field.Visible [= boolean]`

**Parameters**  
- `field` – A window or report field object.
- `boolean` – If True, the field is visible. If False, the field is invisible.

**Comments**  
If you hide a window field using the Visible property, VBA hides both the field and its caption. You can still return or set the values of fields you make invisible with VBA. However, you cannot set the value of fields hidden by the accounting system, nor can you show these fields.

**Examples**  
The following example hides sales fields when a user opens the Salesperson Maintenance window:

```vba
Private Sub Window_AfterOpen()
    If UserInfoGet.UserID = "LESSONUSER1" Then
        CommissionedSales.Visible = False
        CostofSales.Visible = False
        NonCommissionedSales.Visible = False
        TotalCommissions.Visible = False
    End If
End Sub
```

**Related items**  
Methods and properties  
The window object's Visible property and Hide method.

**Events**  
The window object's BeforeOpen event.
Width property

Description
The Width property specifies the width of a field (in pixels).

Syntax
field.Width [= integer]

Parameters
• field – A field object.
• integer – Specifies the width of the field (measured in pixels).

Comments
The Width property calculates the field’s width starting at the left edge of the field, and sizes the field to the specified width. The Width property doesn’t affect the field’s caption.

Examples
The following example changes the width of several fields in the Invoice Entry window:

Private Sub Window_BeforeOpen(OpenVisible As Boolean)
    TradeDiscount.Width = 140
    Freight.Width = 140
    Miscellaneous.Width = 140
    Tax.Width = 140
    Total.Width = 140
End Sub

Related items
Methods and properties
The field object’s Left property, Top property, Height property and Move method.
Chapter 10: Grid Object

Your project uses the grid object to perform several tasks related to working with grids (scrolling windows). The events, methods and properties that apply to the grid object are listed below. A detailed explanation of each follows:

- **AfterLineChange event**
- **AfterLineGotFocus event**
- **AfterLineLostFocus event**
- **BeforeLineChange event**
- **BeforeLineGotFocus event**
- **BeforeLineLostFocus event**
- **BeforeLinePopulate event**
- **Changed property**
- **EventMode property**
- **Hide method**
- **Left property**
- **Move method**
- **Name property**
- **Show method**
- **Top property**
- **Visible property**
# AfterLineChange event

## Description
The **AfterLineChange** event occurs when the contents of a line change in an editable or adds-allowed grid, and the focus moves out of the line. It occurs *after* the Microsoft Dynamics GP line change event.

## Syntax
```
grid_AfterLineChange()
```

## Parameters
- `grid` – A grid object.

## Comments
Use the **AfterLineChange** event to perform other actions in the window after the Microsoft Dynamics GP line change event. Microsoft Dynamics GP uses the line change event to save data in the line when the user leaves the line. An example of this is the Invoice Entry window, where each grid line represents a separate transaction record in the invoice. As the user moves to a new line, the line change event saves the invoice line item record to a table.

## Examples
The following **AfterLineChange** event procedure checks the Subtotal field in the Invoice Entry window when the user enters line item information in the Invoice Entry grid. If the invoice subtotal is greater than $1000, it assigns the transaction to an existing batch:

```vba
Private Sub Grid_AfterLineChange()
    'Check the Invoice subtotal
    If CCur(InvoiceEntry.Subtotal) > 1000 Then
        'Assign an existing batch
        InvoiceEntry.BatchID = "BIGSALES"
        InvoiceEntry.BatchID.Locked = True
    End If
End Sub
```
AfterLineGotFocus event

**Description**  
The AfterLineGotFocus event occurs when the focus moves to a new line in a grid. It occurs after the Microsoft Dynamics GP line got focus event.

**Syntax**  
```vba
grid_AfterLineGotFocus()
```

**Parameters**  
- `grid` – A grid object.

**Comments**  
Use the AfterLineGotFocus event to evaluate the contents of fields in the line gaining focus.

**Examples**  
In the following example, the AfterLineGotFocus event checks the value of the Item Number field in the Invoice Entry grid. If the field is empty, the procedure opens the Items lookup window:

```vba
Private Sub Grid_AfterLineGotFocus()
    If ItemNumber.Empty = True Then
        'Open the lookup window
        InvoiceEntry.LookupButton6 = 1
    End If
End Sub
```
# AfterLineLostFocus event

**Description**  
The `AfterLineLostFocus` event occurs when the focus moves out of a line. It occurs *after* the Microsoft Dynamics GP line lost focus event.

**Syntax**  
```vbnet
grid_AfterLineLostFocus()
```

**Parameters**  
- `grid` – A grid object.

**Comments**  
Microsoft Dynamics GP rarely uses the line lost focus event. Whether you choose the VBA `AfterLineLostFocus` or `BeforeLineLostFocus` event is of little consequence.

**Examples**  
In this example, the `AfterLineLostFocus` event procedure “shrinks” the grid in the Invoice Entry window when the user moves to the next line. To do this, it programmatically “clicks” the grid’s expand button:

```vbnet
Private Sub Grid_AfterLineLostFocus()
    InvoiceEntry.ScrollingWindowExpandButton = 1
End Sub
```
### BeforeLineChange event

**Description**  
The **BeforeLineChange** event occurs when the contents of a line change in an editable or adds-allowed grid, and the focus moves out of the line. It occurs **before** the Microsoft Dynamics GP line change event.

**Syntax**  
```vba
grid_BeforeLineChange(KeepFocus As Boolean)
```

**Parameters**  
- `grid` – A grid object.
- `KeepFocus As Boolean` – If True, this parameter prevents the Microsoft Dynamics GP line change event and **and** the VBA **AfterLineChange** event from occurring, and keeps the focus in the current line.

**Comments**  
Use the **BeforeLineChange** event to check the value of fields in a line, then cancel subsequent line change events if criteria you specify aren’t met.

**Examples**  
In this example, the **BeforeLineChange** event checks the value of the Markdown Amount field in the Invoice Entry grid. If the user attempts to enter a markdown amount greater than 20%, the `KeepFocus` parameter stops the line change event, and moves the focus back to the current line:

```vba
Private Sub Grid_BeforeLineChange(KeepFocus As Boolean)
    If CCurr(MarkdownAmount) > CCurr(UnitPrice) * 0.2 Then
        'The markdown was greater than 20%
        KeepFocus = True
        MsgBox "You cannot enter a markdown greater than 20% * + _
               "of the unit price."
    'End If
End Sub
```
BeforeLineGotFocus event

Description
The BeforeLineGotFocus event occurs when the focus moves to a new line in a grid. It occurs before the Microsoft Dynamics GP line got focus event.

Syntax
grid_BeforeLineGotFocus()

Parameters
• grid – A grid object.

Comments
Use the BeforeLineGotFocus event to evaluate the contents of fields in the line gaining focus.

Microsoft Dynamics GP rarely uses the line got focus event, so whether you choose the VBA AfterLineGotFocus or BeforeLineGotFocus event is of little consequence.

Examples
See the example for the AfterLineGotFocus event.
BeforeLineLostFocus event

Description
The BeforeLineLostFocus event occurs when the focus moves out of a line. It occurs before the Microsoft Dynamics GP line lost focus event.

Syntax
grid_BeforeLineLostFocus()

Parameters
• grid – A grid object.

Comments
Microsoft Dynamics GP rarely uses the line lost focus event. Whether you choose the VBA AfterLineLostFocus or BeforeLineLostFocus event is of little consequence.

Examples
See the example for the AfterLineLostFocus event.
## BeforeLinePopulate event

### Description
The **BeforeLinePopulate** event occurs each time the accounting system displays a new line in a grid.

### Syntax
```
grid_BeforeLinePopulate(RejectLine As Boolean)
```

### Parameters
- **grid** – A grid object.
- **RejectLine As Boolean** – If True, the accounting system won’t display the current line in the grid.

### Comments
When Microsoft Dynamics GP initially displays a grid, the **BeforeLinePopulate** event occurs repeatedly until the grid is filled. It also occurs for each line displayed when the user displays new lines in the grid, (typically, through scrolling up or down in the grid).

Use the **BeforeLinePopulate** event to selectively filter items from a grid. When set to True, the **RejectLine** parameter prevents the current line from appearing in the grid.

### Examples
The following event procedure displays only Illinois customers in the Customers and Prospects lookup window:
```
Private Sub Grid_BeforeLinePopulate(RejectLine As Boolean)
    If State <> "IL" Then
        RejectLine = True
    End If
End Sub
```
## Changed property

**Description**

The **Changed property** indicates whether any non-button field values in a grid’s line have changed.

**Syntax**

`grid.Changed [= boolean]`

**Parameters**

- `grid` – A grid object.
- `boolean` – If True, a non-button field value in a grid’s line changed.

**Comments**

A grid line changes when the value changes in one of the line’s fields. If this occurs, the **Changed property** will return True for the current line (the line with focus) in the grid.

The **Changed property** is useful for determining whether the contents of a line changed, and performing an action based on those changes.

**Examples**

The following example displays a message containing the date and time when a line in a sales transaction record was changed. This information could also be added to the DUOS or to another database (such as Microsoft Access) and stored with the transaction record:

```vba
Private Sub Grid_AfterLineChange()
    If SalesTransactionEntryDetail.Changed Then
        MsgBox "This item changed on " & Date & " at " & Time & "."
    End If
End Sub
```

**Related items**

**Properties and methods**

The window object’s [Changed property](#).
EventMode property

Description
The EventMode property specifies whether VBA grid events occur for the original or modified version of the grid.

Syntax
grid.EventMode [= mode]

Parameters
- grid – A grid object.
- mode – A constant specifying when events occur:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>emOriginalOnly</td>
<td>Grid events occur only for the original version of the grid.</td>
</tr>
<tr>
<td>emModifiedOnly</td>
<td>Grid events occur only for the modified version of the grid.</td>
</tr>
<tr>
<td>emNever</td>
<td>Grid events don’t occur for the grid.</td>
</tr>
</tbody>
</table>

Comments
If you use the Modifier to modify a window containing a grid (such as a lookup window), VBA grid events will occur only if you set both the window’s and the grid’s EventMode property to emModifiedOnly. Changing this property allows events to occur only for the modified version of the window and the grid. To change the EventMode property for the window object, use the Visual Basic Properties window.

Windows modified using the Modifier include a period (.) at the beginning of the window’s caption. Windows that invoke VBA events include a period at the end of the window’s caption. Grids have no caption.

Examples
Typically, you set the grid’s EventMode property using the Visual Basic Properties window. You can also set it through an event procedure, as shown in the following example:

Private Sub Window_BeforeOpen(OpenVisible As Boolean)
    If CustomersandProspects.Caption = "Customers and Prospects." Then
        'Shut off events for the window and the grid.
        CustomersandProspects.EventMode = emOriginalOnly
        CustomersandProspectsDetail.EventMode = emOriginalOnly
    End If
End Sub

Related items
Methods and properties
- The window object’s EventMode property.
- The report object’s EventMode property.
Hide method

**Description**
The Hide method makes a grid invisible.

**Syntax**
`grid.Hide`

**Parameters**
- `grid` – A grid object.

**Comments**
Use the Hide method to make any grid in an open window invisible. While invisible, data in the grid is accessible. Use this method when you want to reference data displayed in the grid without making the grid itself visible, or if you want to hide the contents of the grid from the user.

If you use the window object’s Hide method to hide a window containing a grid (such as a lookup window), the Hide method hides both the window and the grid.

**Examples**
The following example hides the grid on the Sales Transaction Entry window when the window opens. The rest of the window (the window’s header fields, and the window’s summary fields) are still visible:

```vba
Private Sub Window_BeforeOpen(OpenVisible As Boolean)
    SalesTransactionEntryDetail.Hide
End Sub
```

**Related items**
Methods and properties
- The grid object’s Visible property and Show method.
Left property

Description
The Left property specifies the horizontal position of the grid (in pixels) relative to the left edge of the window containing the grid.

Syntax
grid.Left [ = integer ]

Parameters
• grid – A grid object.
• integer – The distance (in pixels) from the left edge of the grid to the left edge of the window’s border.

Comments
The Left property will specify the distance to the left edge of the window for the grid and any of the fields, prompts and controls associated with the grid. These items are associated with the grid if they’re no further than one pixel from the grid’s border, or if they’re one pixel from the edge of another associated item.

Most grids in Microsoft Dynamics GP are positioned 8 pixels to the left of the window’s left edge.

Examples
The following example positions the left edge of the grid 20 pixels from the left edge of the window using the Left property:

Private Sub Window_BeforeOpen(OpenVisible As Boolean)
    CustomersandProspectsDetail.Left = 20
End Sub

Related items
Methods and properties
The grid object’s Top property and Move method.
Move method

Description
The Move method changes the position of a grid relative to the window containing the grid.

Syntax
`grid.Move ([left [, top [, width [, height]]]])`

Parameters
- `grid` – A grid object.
- `left` – An integer specifying the distance (in pixels) from the left edge of the grid to the left edge of the window’s border.
- `top` – An integer specifying the distance (in pixels) from the top edge of the grid to the bottom edge of the window’s title bar.
- `width` – An integer specifying the width (in pixels) of the grid.
- `height` – An integer specifying the height (in pixels) of the grid.

Comments
You can use named arguments for the Move method, or specify arguments by position. If you use positional arguments, enter each in the order shown, using commas and null string values (""") to indicate the relative position of arguments you don’t specify.

The Move method will move the grid and any of the fields, prompts and controls associated with the grid. These items are associated with the grid if they’re no further than one pixel from the grid’s border, or if they’re one pixel from the edge of another associated item.

Examples
The following example uses named arguments to change the height of the grid when the Customers and Prospects lookup window opens:

```vba
Private Sub Window_AfterOpen()
    CustomersandProspectsDetail.Move Height:= 300
End Sub
```

The following example positions the grid in the Customers and Prospects lookup window using positional arguments. It sets only the first two parameters:

```vba
Private Sub Window_AfterOpen()
    CustomersandProspects.Move 10,20,"","
End Sub
```

Related items
Methods and properties
- The grid object's **Top property** and **Left property**.
# Name property

## Description

The **Name property** specifies the internal name VBA uses to reference a grid object.

## Syntax

```vba
grid.Name
```

## Parameters

- `grid` – A grid object.

## Comments

The first portion of a grid’s name is the same as the name of the window containing the grid. However, grid names include the word “Detail” at the end of the name.

Use the **Name property** to change a grid name when you need to resolve naming conflicts between grids and other objects in your application. You can also use the **Name property** when you want to make an object’s name more readable in VBA code. Be sure to replace all referenced occurrences of the window with the new name.

## Examples

To change a grid’s **Name property**, use the Visual Basic Properties window.

## Related items

- **Methods and properties**
  - The field object’s **Name property**.
  - The window object’s **Name property**.
  - The report object’s **Name property**.
**Show method**

<table>
<thead>
<tr>
<th>Description</th>
<th>The Show method makes visible any invisible grid.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax</td>
<td>grid.Show</td>
</tr>
<tr>
<td>Parameters</td>
<td>• grid – A grid object.</td>
</tr>
<tr>
<td>Comments</td>
<td>Use the Show method to make visible any grid that’s currently invisible. The Hide method and Visible property make a grid invisible.</td>
</tr>
<tr>
<td>Examples</td>
<td>The following example shows a grid that’s been hidden using either the Hide method or the Visible property:</td>
</tr>
</tbody>
</table>

```vba
Private Sub Window_BeforeOpen(OpenVisible As Boolean)
    If UserInfoGet.UserID <> "LESSONUSER1" Then
        SalesTransactionEntryDetail.Show
    End If
End Sub
```

**Related items**

Methods and properties

The grid object’s Hide method and Visible property.
### Top property

**Description**  
The Top property specifies the vertical position of the grid relative to the bottom border of the window’s title bar.

**Syntax**  
`grid.Top [= integer]`

**Parameters**  
- `grid` – A grid object.
- `integer` – The distance from the top of the grid to the bottom of the window’s title bar, measured in pixels. The “top” of the grid includes any of the fields, prompts and controls associated with the grid.

**Comments**  
The Top property will move the grid and any of the fields, prompts and controls associated with the grid. These items are associated with the grid if they’re no further than one pixel from the grid’s border, or if they’re one pixel from the edge of another associated item.

**Examples**  
The following example changes the top position of the grid so it now appears below a new field, Sort By State, in the Customers and Prospects lookup window:

```vba
Private Sub Window_BeforeOpen(OpenVisible As Boolean)  
    CustomersandProspectsDetail.Top = SortByState.Top + _
    SortByState.Height + 6  
End Sub
```

**Related items**  
Methods and properties

The grid object’s [Left property](#) and [Move method](#).
**Visible property**

**Description**  
The Visible property specifies whether a grid is visible.

**Syntax**  
`grid.Visible [= boolean]`

**Parameters**  
- `grid` – A grid object.
- `boolean` – If True, the grid is visible. If False, the grid is invisible.

**Comments**  
While invisible, data in a grid is still accessible. Use this property when you want to reference data displayed in the grid without making the grid visible. You can also make a grid invisible using the Hide method.

**Examples**  
The following example hides the grid in the Sales Transaction Entry window based on the current user:

```vba
Private Sub Window_BeforeOpen(OpenVisible As Boolean)
    If.UserInfoGet.UserID = "LESSONUSER1" Then
        SalesTransactionEntryDetail.Visible = False
    End If
End Sub
```

**Related items**  
Methods and properties  
The grid object's Show method and Hide method.
Chapter 11: Report Object

Your project uses the report object to perform several tasks related to working with reports. The events, methods and properties that apply to the report object are listed below. A detailed explanation of each follows:

- BeforeAF event
- BeforeAH event
- BeforeBody event
- BeforePF event
- BeforePH event
- BeforeRF event
- BeforeRH event
- Cancel method
- End event
- EventMode property
- Legend property
- Name property
- Start event
## BeforeAF event

**Description**  
The BeforeAF event occurs before each instance of the specified additional footer prints.

**Syntax**  
```vba  
report_BeforeAF(ByVal Level As Integer, SuppressBand As Boolean)  
```

**Parameters**  
- `report` – A report object.
- `By Val Level As Integer` – The index of the additional footer the event will occur for. Since a report may have more than one additional footer (indicated by F1, F2 and so on), use the `Level` parameter to specify which additional footer you want the event to occur for.
- `SuppressBand As Boolean` – If True, the instance of the specified additional footer won’t print and no VBA events will occur for it. Other portions of the report won’t be affected.

**Comments**  
Use the BeforeAF event to reference fields an additional footer. A report may have none, one or several additional footers. Each prints when the data in the field it is based on changes. Microsoft Dynamics GP uses additional footers to display summary data, such as a total of all records in the report’s body.

If the report has only one additional footer, the BeforeAF event occurs only for that additional footer, and you don’t need to set a value for the `Level` parameter.

**Examples**  
In the following example, the BeforeAF event procedure sets a user-defined calculated field in the second additional footer:

```vba  
Private Sub Report_BeforeAF(ByVal Level As Integer, SuppressBand As Boolean)  
    If Level = 2 Then  
        If CustomerName = "Aaron Fitz Electric" Then  
            'Set a user-defined calculated field  
            Status = "Preferred"  
        End If  
    End If  
End Sub  
```
BeforeAH event

**Description**
The BeforeAH event occurs before each instance of the specified additional header prints.

**Syntax**
`report.BeforeAH(ByVal Level As Integer, SuppressBand As Boolean)`

**Parameters**
- `report` – A report object.
- `By Val Level As Integer` – The index of the additional header the event will occur for. Since a report may have more than one additional header (indicated on the report layout by H1, H2 and so on), use the `Level` parameter to specify which additional header you want the event to occur for.
- `SuppressBand As Boolean` – If True, the instance of the specified additional header won’t print and no VBA events will occur for it. Other portions of the report won’t be affected.

**Comments**
Use the **BeforeAH** event to reference fields in an additional header. A report may have none, one or several additional headers. Each prints when the data in the field it is based on changes. If the report has only one additional header, the **BeforeAH** event occurs only for that additional header, and you don’t need to set a value for the `Level` parameter.

**Examples**
In the following example, the **BeforeAH** event procedure sets the `Level` parameter to 2, indicating that the procedure runs for the second additional header on a report. The procedure then checks the value of the Checkbook ID field in the second additional header:

```vba
Private Sub Report_BeforeAH(ByVal Level As Integer, SuppressBand As Boolean)
    If Level = 2 Then
        If CheckbookID = "PAYROLL" Then
            BankDepositPostingJournal.Cancel
        End If
    End If
End Sub
```
BeforeBody event

Description
The BeforeBody event occurs before each instance of the report’s body prints.

Syntax
report_BeforeBody(SuppressBand As Boolean)

Parameters
- report – A report object.
- SuppressBand As Boolean – If True, the current body record won’t print, and no events will occur for it. Other portions of the report won’t be affected.

Comments
Use the BeforeBody event to reference fields in a report’s body. Microsoft Dynamics GP uses the report body for the bulk of a report, which typically is made up of table fields. A report prints the body repeatedly until all records on the report have printed.

Examples
The following example uses the BeforeBody event to return the value of the Commissioned Sales field in the current body record of the RM Salesperson Report. If a salesperson’s commissioned sales are less than $200,000, the SuppressBand parameter stops the current body record from printing. The result is that the report prints only records for salespeople who’ve exceeded $200,000 in commissioned sales:

    Private Sub Report_BeforeBody(SuppressBand As Boolean)
        If CCur(CommissionedSalesToDate) < 200000 Then
            SuppressBand = True
        End If
    End Sub
# BeforePF event

**Description**  
The **BeforePF** event occurs before the report’s page footer prints.

**Syntax**  
```vba  
report_BeforePF(SuppressBand As Boolean)  
```

**Parameters**  
- `report` – A report object.
- `SuppressBand As Boolean` – If True, the page footer won’t print and no VBA events will occur for it.

**Comments**  
Use the **BeforePF** event to reference fields in a report’s page footer. Items in the page footer are placed at the bottom of every report page. The page footer often includes administrative information, such as the name of the person running the report.

**Examples**  
In the following example, the **BeforePF** event sets a user-defined calculated string field in the report’s footer. It uses the VBA **Time** and **Date** functions to compose a comment on the report:

```vba  
Private Sub Report_BeforePF(SuppressBand As Boolean)  
    Comment = UserInfoGet.UserID + ' printed this at ' + str(Time) + ' on ' + str(Date)  
End Sub  
```
BeforePH event

Description
The BeforePH event occurs before the report’s page header prints.

Syntax
report_BeforePH(SuppressBand As Boolean)

Parameters
• report – A report object.
• SuppressBand As Boolean – If True, the page header won’t print and no events will occur for it.

Comments
Use the BeforePH event to reference fields in a report’s page header. Items in the page header are placed at the top of every report page. Page number, date and time fields are commonly placed in this section of a report.

Examples
The following BeforePH event procedure suppresses the page header based on the current user ID:

Private Sub Report_BeforePH(SuppressBand As Boolean)
    If UserInfoGet.UserID = "LESSONUSER1" Then
        SuppressBand = True
    End If
End Sub
BeforeRF event

Description
The BeforeRF event occurs before the report’s report footer prints.

Syntax
`report_BeforeRF(SuppressBand As Boolean)`

Parameters
- `report` – A report object.
- `SuppressBand As Boolean` – If True, the report footer won’t print and no VBA events will occur for it.

Comments
Use the BeforeRF event to reference fields in the report footer. Summary information is often included in the report footer. If a page footer is also included on the last page, the report footer will print before the page footer.

Examples
The following BeforeRF event procedure runs for the RM Transaction Inquiry report. If the current user is LESSONUSER1, the procedure hides two fields in the report footer:

```vba
Private Sub Report_BeforeRF(SuppressBand As Boolean)
    If UserInfoGet.UserID = "LESSONUSER1" Then
        OriginalAmountSUMRF.Visible = False
        CurrentAmountSUMRF.Visible = False
    Else
        OriginalAmountSUMRF.Visible = True
        CurrentAmountSUMRF.Visible = True
    End If
End Sub
```
**BeforeRH event**

**Description**

The **BeforeRH** event occurs before the report’s report header prints.

**Syntax**

`report_BeforeRH(SuppressBand As Boolean)`

**Parameters**

- `report` – A report object.
- `SuppressBand As Boolean` – If True, the report header won’t print and no VBA events will occur for it.

**Comments**

Use the **BeforeRH** event to reference fields in the report header. Items in the report header appear only on the first page of a report. If a page header is also included on the first page, the report header will appear after the page header.

Microsoft Dynamics GP typically uses fields in the report header that you can’t reference from the **BeforeRH** event; such as system variables that show the page number, user ID and company name, and legend fields that show the record ranges for the report. Microsoft Dynamics GP rarely uses table fields or calculated fields in the report header.

**Examples**

In the following example, the **BeforeRH** event sets a user-defined calculated string field in the report’s header. It uses the VBA **Time** and **Date** function to compose a comment on the report:

```vba
Private Sub Report_BeforeRH(SuppressBand As Boolean)
    Comment = UserInfoGet.UserID + " printed this at " + str(Time) + 
    " on " + str(Date)
End Sub
```
## Cancel method

<table>
<thead>
<tr>
<th>Description</th>
<th>The <strong>Cancel method</strong> cancels a report from a report event.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax</td>
<td><code>report.Cancel</code></td>
</tr>
<tr>
<td>Parameters</td>
<td>• <code>report</code> – A report object.</td>
</tr>
<tr>
<td>Comments</td>
<td>The <strong>Cancel method</strong> stops the report from printing. If canceled from a report band event, data from that band will print before the report stops. If the report prints to the screen, the <strong>Cancel method</strong> automatically closes the Screen Output window. The <strong>End event</strong> occurs after you cancel a report using the <strong>Cancel method</strong>.</td>
</tr>
<tr>
<td>Examples</td>
<td>The following example cancels a report from the report’s <strong>BeforeBody event</strong>:</td>
</tr>
<tr>
<td></td>
<td>Private Sub Report_BeforeBody(SuppressBand As Boolean)</td>
</tr>
<tr>
<td></td>
<td>If SalespersonID = “ERIN J.” Then</td>
</tr>
<tr>
<td></td>
<td>RMSalespersonReport.Cancel</td>
</tr>
<tr>
<td></td>
<td>End If</td>
</tr>
<tr>
<td></td>
<td>End Sub</td>
</tr>
<tr>
<td>Related items</td>
<td><strong>Events</strong></td>
</tr>
<tr>
<td></td>
<td>The report object’s <strong>End event</strong>.</td>
</tr>
</tbody>
</table>
End event

Description
The End event occurs when a report finishes printing.

Syntax
report.End()

Parameters
- report – A report object.

Comments
Use the End event to perform any clean-up of activities for the report, launch other applications, or open and close other windows. The End event will always run if a report’s Start event occurs, including when a report is canceled using the Cancel method.

Examples
In the following example, the End event procedure launches Microsoft Outlook after the RM Customer Report prints:

Private Sub Report_End()
    Dim Response As Integer
    Dim RetVal As Variant
    Response = MsgBox("Do you want to launch MS Outlook?", vbYesNo)
    If Response = vbYes Then
        RetVal = Shell("C:\Program Files\Microsoft Office\Office\Outlook.exe", vbNormalFocus)
    End If
End Sub

Related items
Methods and properties
The report object’s Cancel method.

Events
The report object’s Start event.
EventMode property

Description
The EventMode property specifies whether report events occur for the original or modified version of the report.

Syntax
report.EventMode [= mode]

Parameters
- report – A report object.
- mode – A constant specifying when events occur:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>emOriginalOnly</td>
<td>Report events occur only for the original version of the report.</td>
</tr>
<tr>
<td>emModifiedOnly</td>
<td>Report events occur only for the modified version of the report.</td>
</tr>
<tr>
<td>emNever</td>
<td>Report events don’t occur for the report.</td>
</tr>
</tbody>
</table>

Comments
If you modify a report using the Report Writer, you must set the EventMode property to emModified if you want VBA events to occur for the modified report. To change the EventMode property for the report object, use the Visual Basic Properties window.

Examples
Typically, you set the report’s EventMode property using the Visual Basic Properties window. You can also set it through an event procedure, as shown in the following example:

Private Sub Report_Start()
    If UserInfoGet.UserID = "LESSONUSER1" Then
        RMCustomerReport.EventMode = emOriginalOnly
    End If
End Sub

Related items
Methods and properties
The window object’s EventMode property.
The grid object’s EventMode property.
Legend property

Description
The Legend property specifies the value of a report legend.

Syntax
report.Legend(index) [= string_value]

Parameters
• report – A report object.
• index – An integer that corresponds to the legend field’s array index.
• string_value – The value of the legend field.

Comments
Legends are fields whose data is persistent throughout the report, and must be passed to the report as it starts. You must use the Legend property in the report’s Start event to set or return the value of a legend field in a report.

Legend fields typically include information about the report, such as the range of records the user chose for the report. The following illustration shows legend fields in the RM Customer Report used to show the range of customer records that print for the report.

You don’t add a legend field to your project; instead, you use the Legend property’s index parameter to refer to a specific legend field on the report.

Examples
The following Start event procedure returns the value of the Customer Class legend field using the field’s index:

Private Sub Report_Start()
    If RMCustomerReport.Legend(5) = "AARONFIT0001 - AARONFIT0001" Then
        RMCustomerReport.Cancel
    End If

Related items
Events
The report object’s Start event.
### Name property

**Description**
The **Name** property specifies the internal name VBA uses to reference a report object.

**Syntax**
```
report.Name
```

**Parameters**
- `report` – A report object.

**Comments**
You cannot set the **Name** property at runtime. We recommend that you change a report’s **Name** property using the Visual Basic Properties window. Be sure to replace all referenced occurrences of the report with the new name.

Use the **Name** property to change the report name when you need to resolve naming conflicts between reports and other objects in your application. These conflicts typically arise between names in your project and reserved words in VBA.

You can also use the **Name** property when you want to make an report’s name more readable in VBA code.

**Examples**
To change a report’s **Name** property, use the Visual Basic Properties window.

**Related items**
- **Methods and properties**
  - The window object’s **Name** property.
  - The field object’s **Name** property.
  - The grid object’s **Name** property.
Start event

**Description**
The **Start event** occurs as the report starts to print, but before any data actually prints.

**Syntax**

```
report_Start()
```

**Parameters**

- `report` – A report object.

**Comments**
Use the **Start event** to set any report legends using the **Legend property**. Legends are fields whose data is persistent throughout the report, and must be passed to the report as it starts.

**Examples**
The following example sets the value of a legend field from the **Start event**:

```vbnet
Private Sub Report_Start()
    RMCustomerReport.Legend(2) = "Aging Period Amount"
End Sub
```

**Related items**

Properties and methods

The report object's **Cancel method**.

Events

The report object's **End event**.
Chapter 12: VBAGlobal Object

Your project uses the VBAGlobal object to perform tasks related to the Dynamic User Object Store (DUOS). It also provides a method to retrieve the UserInfo object used to access information about the current user. The methods that apply to the VBAGlobal object are listed below. A detailed explanation of each follows:

- DUOSObjectCombineID method
- DUOSObjectExtractID method
- DUOSObjectsGet method
- UserInfoGet method
DUOSObjectCombineID method

**Description**
The **DUOSObjectCombineID method** constructs a DUOS data object ID using two string values.

**Syntax**
```
DUOSObjectsCombineID(string1, string2)
```

**Parameters**
- *string1* – The string you’re combining with *string2*.
- *string2* – The string you’re combining with *string1*.

**Return value**
A combined string composed of *string1* and *string2*.

**Comments**
Data objects you define in the DUOS must have a unique object ID within a collection. If you create a data object with an object ID that already exists in the collection, the new data object will overwrite the existing object. To avoid duplicate object IDs, create a unique ID using this method.

In most cases, you can assign the value of a Microsoft Dynamics GP control field as the object’s ID. A control field, such as a customer ID or a document number, controls the display of a record in a window, and is a unique value in the accounting system.

**Examples**
The following procedure runs when the user clicks the Save button in the Item Vendors Maintenance window. It uses the **DUOSObjectsCombineID method** to combine the values of two window fields (Vendor ID and Item Number) to construct a unique object ID:

```vba
Private Sub Save_BeforeUserChanged(KeepFocus As Boolean, CancelLogic As Boolean)
    Dim VendorCollection As DUOSObjects
    Dim VendorObject As DUOSObject
    Dim ObjectID As String

    'Specify a vendors collection
    Set VendorCollection = DUOSObjectsGet("Vendors")

    'Combine two field values to create a unique object ID
    ObjectID = DUOSObjectCombineID(VendorID, ItemNumber)

    'Create the object, using the combined ID
    Set VendorObject = Vendors.Item(ObjectID)

    'Set the object’s properties using window fields
    VendorObject.Properties("Discount Quantity") = DiscountQuantity
    VendorObject.Properties("Discount Percent") = DiscountPercent

End Sub
```

**Related items**
**Properties and methods**
The VBAGlobal object’s **DUOSObjectExtractID method**.
**DUOSObjectExtractID method**

**Description**
The `DUOSObjectExtractID` method extracts two strings combined using the `DUOSObjectCombineID` method.

**Syntax**
`DUOSObjectCombineID(objectID, string1, string2)`

**Parameters**
- `objectID` – A string constructed using the `DUOSObjectCombineID` method.
- `string1` – The string combined with `string2` to construct the `objectID`.
- `string2` – The string combined with `string1` to construct the `objectID`.

**Return value**
A boolean indicating whether the `objectID` was combined using the `DUOSObjectCombineID` method. If True, `objectID` was combined.

**Comments**
Use the `DUOSObjectExtractID` method to return the original string values combined using the `DUOSObjectCombineID` method. Extracting an object ID returns the original combined string values. This is useful if you want to update DUOS data objects based on one of the original string values.

**Examples**
The following example uses a button on a VBA user form to loop through a collection named Vendors. It uses the `DUOSObjectExtractID` method to check whether each data object's `objectID` is combined. Any object with a combined `objectID` uses an ID composed of the vendor’s ID and the vendor type. If the vendor type (`ventype`) is “PREFERRED,” the procedure updates the properties for the data object:

```vba
Private Sub CommandButton1_Click()
    Dim VendorCollection As DUOSObjects
    Dim VendorObject As DUOSObject
    Dim ObjectID As String
    Dim venID As String
    Dim ventype As String
    'Return a vendors collection
    Set VendorCollection = DUOSObjectsGet("Vendors")
    For Each VendorObject In VendorCollection
        If DUOSObjectExtractID(VendorObject.ID, venID, ventype) = True Then
            'This is a combined ID. Update with new discount info.
            If ventype = "PREFERRED" Then
                'Update the properties for this vendor
                VendorObject.Properties("Discount Quantity") = "200"
                VendorObject.Properties("Discount Percent") = "6.50"
            End If
        End If
    Next
End Sub
```

**Related items**
- Properties and methods
  - The VBAGlobal object's `DUOSObjectCombineID` method.
DUOSSObjectsGet method

Description
The DUOSSObjectsGet method returns a DUOS collection for the current company.

Syntax
DUOSSObjectsGet(collection_name)

Parameters
- collection_name – A string specifying the collection you’re working with. If the specified collection doesn’t exist for the current company, this method creates it.

Return value
A collection specified by collection_name.

Comments
Use the DUOSSObjectsGet method to specify a collection in the DUOS and assign it to a DUOSSObjects variable. For instance, the following example returns a collection named “Customers”:

Dim CustomerCollection As DUOSSObjects
Set CustomerCollection = DUOSSObjectsGet("Customers")

You can use the DUOSSObjectsGet method to retrieve an existing collection or create a new collection. In the following illustration, the DUOSSObjectsGet method returns an existing collection named Items.

Data stored in the DUOS is company-specific, so collections defined in company A don’t exist when you’re accessing the DUOS in company B.

Examples
The following example returns an existing collection named “CustomerCollection” using the DUOSSObjectsGet method. Once the collection is created, the procedure can add a data object using the collection’s Item property, then assign properties and property values for the data object:

Dim CustomerCollection As DUOSSObjects
Dim CustomerObject As DUOSSObject

'Return the collection
Set CustomerCollection = DUOSSObjectsGet("CustomerCollection")
'Create a data object in the collection with a unique object ID
Set CustomerObject = CustomerCollection.Item("AARONFIT001")
'Create properties for this object
CustomerObject.Properties.Item("URL Address") = "www.fitzelectric.com"
CustomerObject.Properties.Item("Contact E-Mail Address") = "afitz@contoso.com"
UserInfoGet method

**Description**
The `UserInfoGet` method returns a `UserInfo` object containing information about the current user.

**Syntax**
`UserInfoGet(collection_name)`

**Parameters**
- None

**Return value**
The `UserInfo` object for the current user.

**Examples**
The following example retrieves the `UserInfo` object, and then displays the current company’s name in a message box.

```vba
Dim UserInfoObj As UserInfo
Dim CompanyName As String

'Get the UserInfo object
Set UserInfoObj = VbaGlobal.UserInfoGet()

'Retrieve and display the company name
CompanyName = UserInfoObj.CompanyName
MsgBox CompanyName
```

**Related items**
Methods and properties
- The `UserInfo` object's `CompanyName property`, `IntercompanyID property`, `UserDate property`, `UserID property`, `UserName property`, and `CreateADOConnection method`. 
Chapter 13: DUOSObjects Collection

Your project uses the DUOSObjects collection to perform several tasks related to creating and maintaining a collection of objects in the Dynamic User Object Store (DUOS). The methods and properties that apply to a DUOSObjects collection are listed below. A detailed explanation of each follows:

- Exists property
- Item property
- Name property
- Remove method
- SortByExisting property
** Exists property **

**Description**

The `Exists` property returns True if specified data object exists in a collection.

**Syntax**

`DUOSObjects.Exists(objectID)`

**Parameters**

- `DUOSObjects` – A collection of data objects in the DUOS.
- `objectID` – The unique identifier for the data object in the collection.

**Comments**

The `Exists` property is a read-only property. If the specified data object exists, it will return True. Otherwise, it will return False.

**Examples**

In the following example, the `Changed` event procedure runs for the Total field when the user changes an invoice amount in the Invoice Entry window. The procedure uses the `Exists` property to ascertain whether a corresponding invoice object already exists in the collection. If it does exist, the procedure updates it with the current system date using the VBA `Date()` function:

```vba
Private Sub Total_Changed()
    Dim InvoiceCollection As DUOSObjects
    Dim InvoiceObject As DUOSObject

    Set InvoiceCollection = DUOSObjectsGet("Invoices")
    If InvoiceCollection.Exists(DocumentNumber) Then
        ' A data object for this invoice exists
        ' Set a property to indicate the user changed the total
        InvoiceObject.Properties("Invoice Adjustment Date") = Date
    End If
End Sub
```
Item property

Description
The Item property returns a data object from a collection.

Syntax
`DUOSObjects.Item(objectID)`

Parameters
- `DUOSObjects` – A collection of data objects in the DUOS.
- `objectID` – The unique identifier for the data object in the collection.

Comments
The Item property is the default member of the DUOSObjects collection. If you omit the Item property from your VBA code, the collection assumes you’re returning the data object for the specified collection. The following example includes the Item property:

```
ItemCollection.Item("ITM001")
```

This example omits the Item property:

```
ItemCollection("ITM001")
```

Examples
The following procedure runs when the user clicks a button in a VBA user form. It returns a collection named Item Info, then uses the Item property to return a data object with an objectID matching the Item Number in the Item Maintenance window. The Set statement assigns the returned data object to an object variable named ItemObject. The Remove method then deletes the object returned by the Item property:

```
Private Sub DeleteItem_Click()
    Dim ItemCollection As DUOSObjects
    Dim ItemObject As DUOSObject

    Set ItemCollection = DUOSObjectsGet("ItemInfo")
    'Specify a data object using a window field value
    Set ItemObject = ItemCollection.Item(ItemMaintenance.ItemNumber)
    ItemCollection.Remove(ItemMaintenance.ItemNumber)
End Sub
```
## Name property

**Description**  
The Name property returns the name of a collection.

**Syntax**  
`DUOSObjects.Name`

**Parameters**  
- `DUOSObjects` – A collection of data objects in the DUOS.

**Comments**  
Use the Name property to identify the name of a collection object returned by the `DUOSObjectsGet` method. You can use the Name property to distinguish between multiple collections in the same procedure.

**Examples**  
The following procedure runs when the user clicks a button in a VBA user form. The Name property returns the name of the collection object returned by the `DUOSObjectsGet` method:

```vba
Private Sub CommandButton_Click()
    Dim Collection As DUOSObjects
    Dim ItemObject As DUOSObject

    If CustomerMaintenance.IsLoaded Then
        Set Collection = DUOSObjectsGet("Customers")
    Else
        Set Collection = DUOSObjectsGet("Items")
    End If

    'Display either "Customers" or "Items"
    MsgBox "The current collection is " + Collection.Name
End Sub
```

**Related items**  
Properties and methods

The VBAGlobal object's `DUOSObjectsGet` method.
Remove method

Description

The **Remove method** deletes a data object from a collection.

Syntax

```
DUOSObjects.Remove(objectID)
```

Parameters

- **DUOSObjects** – A collection of data objects in the DUOS.
- **objectID** – The unique identifier for the data object in the collection.

Examples

The following procedure runs when the user clicks a button in a VBA user form. The procedure loops through a collection named Item Info and checks the Item Color property for each item. If the value is Red, the collection’s **Remove method** deletes the data object:

```vbnet
Private Sub CommandButton_Click()
    Dim ItemCollection As DUOSObjects
    Dim ItemObject As DUOSObject

    Set ItemCollection = DUOSObjectsGet("Item Info")
    For Each ItemObject In ItemCollection
        If ItemObject.Properties.Item("Item Color") = "Red" Then
            'Delete the object
            ItemCollection.Remove(ItemObject.ID)
        End If
    Next
End Sub
```
SortByExisting property

Description
The SortByExisting property specifies the property by which the collection sorts objects.

Syntax
DUOSObjects.SortByExisting [= property_name]

Parameters
- DUOSObjects – A collection of data objects in the DUOS.
- property_name – The property used for the sort operation.

Comments
Use the SortByExisting property to sort the objects in a collection by the value of a given property. In the following example, the SortByExisting property sorts the data objects in the Customers collection by the Date Added property:

<table>
<thead>
<tr>
<th>Collection Name</th>
<th>Object ID</th>
<th>Property name</th>
<th>Property value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customers</td>
<td>C001</td>
<td>Date Added</td>
<td>03/22/96</td>
</tr>
<tr>
<td></td>
<td>C005</td>
<td>Date Added</td>
<td>01/04/96</td>
</tr>
<tr>
<td></td>
<td></td>
<td>URL Address</td>
<td><a href="http://www.afitz.com">www.afitz.com</a></td>
</tr>
<tr>
<td></td>
<td>C002</td>
<td>Date Added</td>
<td>12/18/95</td>
</tr>
<tr>
<td></td>
<td></td>
<td>URL Address</td>
<td><a href="http://www.altonmfg.com">www.altonmfg.com</a></td>
</tr>
<tr>
<td></td>
<td>C001</td>
<td>Date Added</td>
<td>04/13/95</td>
</tr>
</tbody>
</table>

In this case, the items are sorted in ascending order starting with the most recent date. If the property_name exists only for a subset of objects in the collection, the collection will be restricted to that subset of objects. In the following example, the SortByExisting method sorts by the URL Address property:

<table>
<thead>
<tr>
<th>Collection Name</th>
<th>Object ID</th>
<th>Property name</th>
<th>Property value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customers</td>
<td>C005</td>
<td>Date Added</td>
<td>01/04/96</td>
</tr>
<tr>
<td></td>
<td></td>
<td>URL Address</td>
<td><a href="http://www.afitz.com">www.afitz.com</a></td>
</tr>
<tr>
<td></td>
<td>C002</td>
<td>Date Added</td>
<td>12/18/95</td>
</tr>
<tr>
<td></td>
<td></td>
<td>URL Address</td>
<td><a href="http://www.altonmfg.com">www.altonmfg.com</a></td>
</tr>
</tbody>
</table>
Examples

The following procedure runs when the user clicks a button added to the Customer Maintenance window using the Modifier. It uses the `SortByExisting` method to sort the CustomerCollection by the collection’s Date Added property. It then creates a report file using the VBA `Open` statement, and prints the object’s ID, property names and property values to the file using the VBA `Print#` statement. Only data objects that include the Date Added property will be printed, and will be sorted by date:

```vba
Private Sub PrintByDateButton_AfterUserChanged()
    Dim CustomerCollection As DUOSObjects
    Dim CustomerObject As DUOSObject
    Dim CustomerProperty As DUOSProperty
    Dim CustomerProperties As DUOSProperties

    Set CustomerCollection = DUOSObjectsGet("CustomerCollection")

    'Sort the collection by the date the customer was added
    CustomerCollection.SortByExisting = "Date Added"
    'Create a text file for the report
    Open "CustomerReport.txt" For Output As #1
    For Each CustomerObject In CustomerCollection
        Print #1, CustomerObject.ID
        Set CustomerProperties = CustomerObject.Properties
        For Each CustomerProperty In CustomerObject.Properties
            Print #1, " " + CustomerProperty.Name + " - " + _
            CustomerProperty.Value
        Next
        Next
    End Sub
```
Chapter 14: DUOSObject Object

Your project uses the DUOSObject object to perform several tasks related to creating and maintaining a data object in the Dynamic User Object Store (DUOS). The properties that apply to the DUOSObject object are listed below. A detailed explanation of each follows:

- ID property
- Properties property
**ID property**

**Description**  
The **ID property** returns an object ID for a DUOS data object.

**Syntax**  
`DUOSObject.ID`

**Parameters**  
- `DUOSObject` – A DUOS data object.

**Comments**  
The **ID property** is a read-only property. To set an object ID for a DUOS data object, use the **Item property** for the DUOSObjects collection.

**Examples**  
The following procedure runs when the user clicks a button in a VBA user form. The procedure loops through the collection named Item Info and checks the Item Color property for each item. If the value is Red, the collection’s **Remove method** deletes the object indicated by the **ID property**:

```vba
Private Sub CommandButton_Click()
    Dim ItemCollection As DUOSObjects
    Dim ItemObject As DUOSObject

    Set ItemCollection = DUOSObjectsGet('Item Info')
    For Each ItemObject In ItemCollection
        If ItemObject.Properties.Item('Item Color') = 'Red' Then
            'Delete the object
            ItemCollection.Remove (ItemObject.ID)
        End If
    Next
End Sub
```
# Properties property

**Description**  
The `Properties` property returns a collection of properties for a DUOS data object.

**Syntax**  
`DUOSObject.Properties`

**Parameters**  
- `DUOSObject` – A DUOS data object.

**Comments**  
You can use the `Properties` property in combination with the `Item` property to specify a named property for the object. In most cases, this is the preferred way to set or return a property’s value.

**Examples**  
The following procedure runs when the user clicks a button in a VBA user form. This example returns a collection of properties for the current data object:

```vba
Private Sub CommandButton_Click()
    Dim ItemCollection As DUOSObjects
    Dim ItemObject As DUOSObject
    Dim ItemProperties As DUOSProperties

    'Specify the collection
    Set ItemCollection = DUOSObjectsGet("Additional Item Info")
    'Specify an object in the collection with a unique object ID
    Set ItemObject = ItemCollection.Item("ITM002")
    'Get properties for the object
    ItemProperties = ItemObject.Properties
End Sub
```
Chapter 15: DUOSProperties Collection

Your project uses the DUOSProperties collection to perform several tasks related to creating and maintaining properties for data objects in the Dynamic User Object Store (DUOS). The methods and properties that apply to the DUOSProperties collection are listed below. A detailed explanation of each follows:

- Count property
- Exists property
- Item property
- Remove method
Count property

**Description**  
The Count property returns the number of property objects in a properties collection.

**Syntax**  
`DUOSProperties.Count`

**Parameters**  
- `DUOSProperties` – A collection of properties for a data object.

**Comments**  
The Count property is useful when you want to perform any DUOS maintenance activities, such as removing unwanted properties, or adding properties to a data object if it doesn’t have the correct number of properties.

**Examples**  
The following example uses the Count property to return the number of properties for a given data object. If the number of properties is greater than 2, the procedure loops through the collection, and uses the Remove method to delete properties other than Item Weight and Item Color:

```vba
Private Sub DeleteExtraProperties_Click()
    Dim ItemCollection As DUOSObjects
    Dim ItemObject As DUOSObject
    Dim ItemProperties As DUOSProperties
    Dim ItemProperty As DUOSProperty

    Set ItemCollection = DUOSObjectsGet("ItemInfo")
    Set ItemObject = ItemCollection(ItemNumber)
    Set ItemProperties = ItemObject.Properties

    If ItemProperties.Count > 2 Then
        'There are too many properties for this object
        'Remove any other than Item Weight and Item Color
        For Each ItemProperty In ItemProperties
            If Not ((ItemProperty.Name = "Item Weight") Or _
                (ItemProperty.Name = "Item Color")) Then
                ItemProperties.Remove (ItemProperty)
            End If
        Next
    End If
End Sub
```
**Exists property**

**Description**

The **Exists property** returns True if a property object exists in a properties collection.

**Syntax**

`DUOSProperties.Exists(property_name)`

**Parameters**

- `DUOSProperties` – A collection of properties for a data object.
- `property_name` – The property object’s name.

**Examples**

The following procedure runs for a button in a VBA user form. It uses the **Exists property** to check whether the Item Color property exists for a given data object. If the property doesn’t exist, the procedure creates it using the collection’s **Item property**:

```vba
Private Sub CheckItemColor_Click()
    Dim ItemCollection As DUOSObjects
    Dim ItemObject As DUOSObject
    Dim ItemProperties As DUOSProperties
    Dim ItemColor As String

    Set ItemCollection = DUOSObjectsGet("ItemCollection")
    Set ItemObject = ItemCollection(ItemNumber)
    Set ItemProperties = ItemObject.Properties

    If ItemProperties.Exists("Item Color") = False Then
       'This item has no color property
       ItemProperties.Item("Item Color") = InputBox("Enter a " + _
       "color for this item.")
       'Set the window field
       ItemColor = ItemProperties.Item("Item Color")
    End If
End Sub
```
Item property

Description

The Item property returns a property object from a properties collection.

Syntax

DUOSProperties.Item(index)

Parameters

- DUOSProperties – A collection of properties for a data object.
- index – The property object’s name (a string) or the position (an integer) of the property object in the collection.

Comments

If you reference the property position in the index, the position starts at 1 for the first property in the data object. The order you added the property to the data object determines the property object’s position.

If you reference the property name in the index, and the name does not exist, the Item property creates a property in the collection with that name.

The Item property is the default member of the DUOSProperties collection. If you omit the Item property from your VBA code, the collection assumes you’re returning the property for the specified properties collection. The following example includes the Item property:

ItemProperties.Item("Item Color")

This example omits the Item property:

ItemProperties("Item Color")

Examples

The following procedure runs when the user clicks a button in a VBA user form. It loops through a properties collection and uses the Item property to check the property named Item Weight:

Private Sub UpdateItemLocation_Click()
    Dim ItemCollection As DUOSObjects
    Dim ItemObject As DUOSObject
    Dim ItemProperties As DUOSProperties
    Dim ItemProperty As DUOSProperty

    Set ItemCollection = DUOSObjectsGet("ItemCollection")
    For Each ItemObject In ItemCollection
        For Each ItemProperty In ItemProperties
            If ItemProperties.Item("Item Weight") > "100.00" Then
                'The item is greater than 100 pounds
                ItemProperties.Item("Storage Location") = _
                "Bulk Item Area"
            End If
            Next
        Next
    End Sub
In this example, the **Item property** references the position of the property rather than the name:

```vba
Private Sub CommandButton_Click()
    Dim ItemCollection As DUOSObjects
    Dim ItemObject As DUOSObject
    Dim ItemProperties As DUOSProperties

    'Specify the collection
    Set ItemCollection = DUOSObjectsGet("Additional Item Info")
    'Specify an object in the collection with a unique object ID
    Set ItemObject = ItemCollection.Item("ITM002")
    'Get properties for the object
    ItemObject.Properties(3) = "3.5"

    End Sub
```
**Remove method**

**Description**

The **Remove method** deletes a specified property object from a properties collection.

**Syntax**

```vba
DUOSProperties.Remove(index)
```

**Parameters**

- **DUOSProperties** – A collection of properties for a data object.
- **index** – The property object’s name (a string) or the position (an integer) of the property object in the collection.

**Examples**

The following procedure runs when the user clicks a button in a VBA user form. It removes a property object from a properties collection using the **Remove method**:

```vba
Private Sub RemoveColorProperty_Click()
    Dim ItemCollection As DUOSObjects
    Dim ItemObject As DUOSObject
    Dim ItemProperties As DUOSProperties

    Set ItemCollection = DUOSObjectsGet("ItemInfo")
    Set ItemObject = ItemCollection(ItemNumber)
    Set ItemProperties = ItemObject.Properties

    If ItemProperties.Exists("Item Color") Then
        ItemProperties.Remove("Item Color")
    End If

End Sub
```
Chapter 16: DUOSProperty Object

Your project uses the DUOSProperty object to perform tasks related to a property for an object in the Dynamic User Object Store (DUOS). The properties that apply to the DUOSProperty object are listed below. A detailed explanation of each follows:

- Name property
- Value property
Name property

**Description**
The Name property specifies the name of a property for a data object.

**Syntax**

```
DUOSProperty.Name
```

**Parameters**
- `DUOSProperty` – A property object within a properties collection.

**Examples**
The following procedure runs when the user clicks a button in a VBA user form. It uses the Name property with the DUOSProperties’ Remove method to delete properties for a data object other than Item Weight and Item Color:

```vba
Private Sub DeleteExtraProperties_Click()
    Dim ItemCollection As DUOSObjects
    Dim ItemObject As DUOSObject
    Dim ItemProperties As DUOSProperties
    Dim ItemProperty As DUOSProperty

    Set ItemCollection = DUOSObjectsGet('ItemInfo')
    Set ItemObject = ItemCollection(ItemNumber)
    Set ItemProperties = ItemObject.Properties

    'Remove properties other than Item Weight and Item Color
    For Each ItemProperty In ItemProperties
        If Not ((ItemProperty.Name = "Item Weight") Or _
            (ItemProperty.Name = "Item Color")) Then
            ItemProperties.Remove (ItemProperty)
        End If
    Next

End Sub
```
**Value property**

**Description**
The Value property specifies the value of a property object.

**Syntax**
```
DUOSProperty.Value [= string_value]
```

**Parameters**
- `DUOSProperty` – A property object within a properties collection.
- `string_value` – A string that specifies the value of a property object.

**Comments**
When you set the value of a property object using the Value property, you cannot exceed 132 characters. The DUOS stores and returns property values as strings.

The Value property is the default member of the DUOSProperty object. If you omit the Value property from your VBA code, the object assumes you’re returning the value of the specified property object. The following example includes the Value property:

```
ItemProperty.Value = "Red"
```

This example omits the Value property:

```
ItemProperty = "Red"
```

**Examples**
The following example runs when the user clicks a button in a VBA user form. The procedure loops through a collection named Item Info and uses the Value property to set the value of the Item Color property for all data objects:

```vba
Private Sub UpdateItemColor_Click()
    Dim ItemCollection As DUOSObjects
    Dim ItemObject As DUOSObject
    Dim ItemProperties As DUOSProperties
    Dim ItemProperty As DUOSProperty

    Set ItemCollection = DUOSObjectsGet("Item Info")
    'Update the item color for all objects in the collection
    For Each ItemObject In ItemCollection
        For Each ItemProperty In ItemObject.Properties
            If ItemProperty.Name = "Item Color" Then
                ItemProperty.Value = "Red"
            End If
        Next
    Next
End Sub
```

**Related items**
**Methods and properties**
The field object’s Value property.
Chapter 17: User Info Object

The User Info object contains information about the user currently logged into Microsoft Dynamics GP. The properties and method that apply to the User Info object are listed below. A detailed explanation of each follows:

- CompanyName property
- CreateADOConnection method
- IntercompanyID property
- SystemDatabaseName property
- UserDate property
- UserID property
- UserName property
## CompanyName property

<table>
<thead>
<tr>
<th><strong>Description</strong></th>
<th>The <strong>CompanyName property</strong> contains the name of the company that the user is currently logged into in Microsoft Dynamics GP.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Syntax</strong></td>
<td><code>UserInfo.CompanyName</code></td>
</tr>
<tr>
<td><strong>Parameters</strong></td>
<td>• <code>UserInfo</code> – A UserInfo object.</td>
</tr>
<tr>
<td><strong>Examples</strong></td>
<td>The following example retrieves and displays the name of the company the user is currently logged into.</td>
</tr>
</tbody>
</table>

```vba
Private Sub CompanyName_Click()
    Dim UserInfoObj As UserInfo
    Dim CompanyName As String

    'Get the UserInfo object
    Set UserInfoObj = VbaGlobal.UserInfoGet()

    'Retrieve and display the company name
    CompanyName = UserInfoObj.CompanyName
    MsgBox CompanyName
End Sub
```
CreateADOConnection method

Description
The CreateADOConnection method creates a new ADO connection to the database.

Syntax
UserInfo.CreateADOConnection()

Parameters
- UserInfo – A UserInfo object.

Examples
The following example creates an ADO connection and issues a command to retrieve all of the customer records from the RM00101 table in the current company’s database. Note how the IntercompanyID property of the UserInfo object is used to set the default database for the ADO connection.

Private Sub CreateADOConnection_Click()
    Dim cn As New ADODB.Connection
    Dim rst As New ADODB.Recordset
    Dim cmd As New ADODB.Command

    On Error Resume Next

    'Retrieve an ADO connection for the current user
    Set cn = UserInfoGet.CreateADOConnection()

    'Set the connection properties
    cn.CursorLocation = adUseClient

    'Set the current database, using the IntercompanyID property
    cn.DefaultDatabase = UserInfoGet.IntercompanyID

    'Create a command to select all customers
    cmd.ActiveConnection = cn
    cmd.CommandType = adCmdText
    cmd.CommandText = "Select * from RM00101"
    Set rst = cmd.Execute

    'Display the number of rows retrieved
    MsgBox rst.RecordCount

    'Close the connection
    cn.Close
End Sub
IntercompanyID property

**Description**  
The IntercompanyID property contains the internal ID of the company that the user is currently logged into in Microsoft Dynamics GP.

**Syntax**  
`UserInfo.IntercompanyID`

**Parameters**  
- `UserInfo` – A UserInfo object.

**Comments**  
The value of the IntercompanyID property is the same as the name of the SQL database used for that company.

**Examples**  
The following example retrieves and displays the intercompany ID for the company the user is currently logged into.

```vba
Private Sub IntercompanyID_Click()
    Dim UserInfoObj As UserInfo
    Dim IntercompanyID As String
    'Get the UserInfo object
    Set UserInfoObj = VbaGlobal.UserInfoGet()
    'Retrieve and display the intercompany ID
    IntercompanyID = UserInfoObj.IntercompanyID
    MsgBox IntercompanyID
End Sub
```
SystemDatabaseName property

**Description**
The **SystemDatabaseName** property contains the name of the system database for the Microsoft Dynamics GP installation that the user is currently logged into.

**Syntax**
```vba
UserInfo.SystemDatabaseName
```

**Parameters**
- **UserInfo** – A UserInfo object.

**Comments**
The value of the SystemDatabaseName property is the same as the name of the SQL database used to store the system information for the Microsoft Dynamics GP installation.

**Examples**
The following example retrieves and displays the name of the system database for the Microsoft Dynamics GP installation that the user is currently logged into.

```vba
Private Sub SystemDatabase_Click()
    Dim UserInfoObj As UserInfo
    Dim SystemDatabase As String

    'Get the UserInfo object
    Set UserInfoObj = VbaGlobal.UserInfoGet()

    'Retrieve and display the system database name
    SystemDatabase = UserInfoObj.SystemDatabaseName
    MsgBox SystemDatabase

End Sub
```
**UserDate property**

**Description**  
The UserDate property contains the user date currently set in Microsoft Dynamics GP.

**Syntax**  
`UserInfo.UserDate`

**Parameters**  
- *UserInfo* – A UserInfo object.

**Examples**  
The following example retrieves and displays the user date currently set in Microsoft Dynamics GP.

```vba
Private Sub UserDate_Click()
    Dim UserInfoObj As UserInfo
    Dim UserDate As Date

    'Get the UserInfo object
    Set UserInfoObj = VbaGlobal.UserInfoGet()

    'Retrieve and display the user date
    UserDate = UserInfoObj.UserDate
    MsgBox "Current date is: " + Str(UserDate)

End Sub
```
**UserID property**

**Description**  
The **UserID property** contains the user ID value for the user currently logged into Microsoft Dynamics GP.

**Syntax**  
`UserInfo.UserID`

**Parameters**  
- `UserInfo` – A `UserInfo` object.

**Comments**  
The value of the UserID property is the same as the user’s SQL login.

**Examples**  
The following example retrieves and displays the user ID for the user currently logged into Microsoft Dynamics GP.

```vba
Private Sub UserID_Click()
    Dim UserInfoObj As UserInfo
    Dim UserID As String

    'Get the UserInfo object
    Set UserInfoObj = VbaGlobal.UserInfoGet()

    'Retrieve and display the user ID
    UserID = UserInfoObj.UserID
    MsgBox UserID

End Sub
```
**UserName property**

**Description**  
The **UserName** property contains the display name for the user currently logged into Microsoft Dynamics GP.

**Syntax**  
`UserInfo.UserName`

**Parameters**  
- `UserInfo` – A `UserInfo` object.

**Examples**  
The following example retrieves and displays the name of the user currently logged into Microsoft Dynamics GP.

```vba
Private Sub UserName_Click()
    Dim UserInfoObj As UserInfo
    Dim UserName As String

    'Get the UserInfo object
    Set UserInfoObj = VbaGlobal.UserInfoGet()

    'Retrieve and display the user name
    UserName = UserInfoObj.UserName
    MsgBox UserName
End Sub
```
Appendix

This manual has the following appendix:

- Appendix A, “VBA Errors,” explains how to handle specific types of errors in your Microsoft Dynamics GP VBA projects.
Appendix A: VBA Errors

The information presented here explains the runtime errors that you can encounter using VBA with Microsoft Dynamics GP. It is divided into the following sections:

- Handling VBA errors
- VBA error reference

Handling VBA errors

VBA errors display a VBA dialog with an error number and message text. VBA halts the execution of VBA code at the point the error occurred. You can handle Microsoft Dynamics GP VBA errors in the same manner as other errors that appear in VBA, using VBA’s On Error statement to respond to the error condition. The On Error statement has three clauses that allow you to handle errors:

<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>On Error Go To line</td>
<td>Enables an error-handling routine that starts at line. The specified line must be in the same procedure as the On Error statement.</td>
</tr>
<tr>
<td>On Error Resume Next</td>
<td>Passes control to the statement that immediately follows the statement that generated the error.</td>
</tr>
<tr>
<td>On Error Go To 0</td>
<td>Disables any enabled error handler in the current procedure.</td>
</tr>
</tbody>
</table>

In the following example, the Resume Next clause of the On Error statement allows the procedure to skip the statement that generates an error. In this case, setting the Shipping Method field will generate an error (VBA error 1007) since “NEW” isn’t an existing shipping method. However, the procedure continues to run, and the final statement sets the Trade Discount field:

```vba
Private Sub CustomerID_AfterUserChanged()
    'Bypass any line that encounters an error
    On Error Resume Next
    'Set a window field that generates an error
    ShippingMethod = "NEW"
    'Set a window field that doesn’t generate an error
    TradeDiscount = "10.00"
End Sub
```

VBA error reference

The following table provides a short description of the errors specific to VBA for Microsoft Dynamics GP. Detailed explanations of the errors follow.

<table>
<thead>
<tr>
<th>Error code</th>
<th>Message text</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>Unexpected error.</td>
</tr>
<tr>
<td>1002</td>
<td>Object not available.</td>
</tr>
<tr>
<td>1003</td>
<td>Unsafe operation. This operation could compromise the integrity of the application.</td>
</tr>
<tr>
<td>1004</td>
<td>This property is Read Only.</td>
</tr>
<tr>
<td>1005</td>
<td>Unsafe operation. This operation can only be performed in the BeforeGotFocus or AfterGotFocus events of the target field. The operation could compromise the integrity of the application.</td>
</tr>
<tr>
<td>1006</td>
<td>Unsafe operation. This operation cannot be performed in the target field’s AfterUserChanged, BeforeLostFocus or AfterLostFocus events. This operation could compromise the integrity of the application.</td>
</tr>
</tbody>
</table>
To access online help for these VBA errors, press the Help button in the error dialog when the error occurs.

**Error 1001  Unexpected error.**

**Situation:**
This error occurs in the following instances:

- You set the caption for a field that didn’t previously have a caption.

- OLE automation failed within Microsoft Dynamics GP. Microsoft Dynamics GP uses OLE automation to communicate with VBA.

**Solution:**
If you’re using the **Caption property** to set a field’s caption, be sure the field has an existing caption.

If you’re performing other tasks and receive this error, Microsoft Dynamics GP may be running out of memory. Shut down applications that aren’t necessary, then restart the accounting system. If the error persists, contact Technical Support.

**Error 1002  Object not available.**

**Situation:**
You’ve referenced a window field object, but the window containing the field currently isn’t open.

**Solution:**
To reference a window field in VBA code, the window must be open (its **IsLoaded property** is True). The window does not need to be visible to reference the window field.

When you reference a window field from outside the current window, it’s good practice to check the window object’s **IsLoaded** property to ascertain whether the window containing the field is open. The following procedure checks if the window is open using the **IsLoaded** property. If it is, the procedure sets value of a window field:

```vba
Private Sub
    If CustomerMaintenance.IsLoaded Then
        'Reference a window field
        CustomerMaintenance.ShippingMethod = “UPS BLUE”
    End If
End Sub
```
Error 1003  Unsafe operation. This operation could compromise the integrity of the application.

**Situation:**
You attempted to enable a field disabled by the accounting system, unlock a field locked by the accounting system, make visible a field made invisible by the accounting system, or make unrequired a field marked as required by the accounting system. To preserve data integrity, Microsoft Dynamics GP application code overrides any attempt you make through VBA to change properties for disabled, locked, hidden or required fields.

**Solution:**
If Microsoft Dynamics GP application code disables, locks, hides or requires a field, it may impact how your VBA code works with the field.

The following table explains some general guidelines for these properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enabled</td>
<td>If the accounting system disables a field (its Enabled property is False), you cannot enable it or set its value using VBA.</td>
</tr>
<tr>
<td>Locked</td>
<td>If the accounting system locks a field, you cannot unlock it (set its Locked property to False) or set its value using VBA.</td>
</tr>
<tr>
<td>Visible</td>
<td>If the accounting system hides a field, you cannot make the field visible (set its Visible property to True) or set its value using VBA.</td>
</tr>
<tr>
<td>Required</td>
<td>If the accounting system requires a field, you cannot make the field unrequired (set its Required property to False).</td>
</tr>
</tbody>
</table>

Error 1004  This property is Read Only.

**Situation:**
You attempted to set the value of a read-only property.

**Solution:**
You cannot set the value of a read-only property.

The following table shows the read only properties in VBA:

<table>
<thead>
<tr>
<th>Object/Collection</th>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Window</td>
<td>IsLoaded</td>
<td>Specifies whether the window is open.</td>
</tr>
<tr>
<td></td>
<td>Required</td>
<td>Specifies whether the user entered data in all fields whose Required property is True.</td>
</tr>
<tr>
<td>Field</td>
<td>Parent</td>
<td>Returns the window or report object that contains the specified field.</td>
</tr>
<tr>
<td></td>
<td>Object</td>
<td>Returns a field object without the standard properties and methods extended to it by Visual Basic.</td>
</tr>
<tr>
<td>DUOSObject</td>
<td>ID</td>
<td>Returns an object ID for a data object.</td>
</tr>
<tr>
<td>DUOSObjects</td>
<td>Item</td>
<td>Returns a data object from a collection of data objects.</td>
</tr>
<tr>
<td>DUOSPProperties</td>
<td>Item</td>
<td>Returns a property object from a collection of properties.</td>
</tr>
</tbody>
</table>
Error 1005  Unsafe operation. This operation can only be performed in the
BeforeGotFocus or AfterGotFocus events of the target field. The operation could
compromise the integrity of the application.

**Situation:**
You attempted to set the value of a composite field using the ValueSeg method
from an event other than the composite field’s BeforeGotFocus or AfterGotFocus
events.

**Solution:**
When setting a composite field using the ValueSeg property, set the composite from
only the BeforeGotFocus or AfterGotFocus events for the field. When a composite
gains focus, Microsoft Dynamics GP application code validates the value of the
composite, segment by segment. Since the accounting system can perform this
validation only after segments’ values have been set, you must set segment values
using the ValueSeg property in events that precede the validation (the
BeforeGotFocus and AfterGotFocus events).

Error 1006  Unsafe operation. This operation cannot be performed in the target field’s
AfterUserChanged, BeforeLostFocus or AfterLostFocus events. This operation could
compromise the integrity of the application.

**Situation:**
You attempted to set a field’s value using the field’s AfterUserChanged,
BeforeUserChanged or AfterLostFocus events. These events follow the Microsoft
Dynamics GP user changed event. Microsoft Dynamics GP uses the user changed
event to verify the value in a field; setting the value of the field after the Microsoft
Dynamics GP user changed event does not allow the accounting system to verify
the field.

**Solution:**
Set the field’s value using the BeforeGotFocus, AfterGotFocus or
BeforeUserChanged events. These events precede the Microsoft Dynamics GP user
changed event, and allow the accounting system to verify the contents of the field.

Error 1007  Unsafe operation. An attempt was made to set a value that violates the
application’s business logic. This operation could compromise the integrity of the
application.

**Situation:**
You attempted to set a field’s value from “outside” the field (before the field gains
focus), such as by using the window’s BeforeOpen or AfterOpen event, or another
field’s BeforeUserChanged or AfterUserChanged event.

When you set the value of a field before the field gains focus, VBA automatically
runs the Microsoft Dynamics GP user changed event for the field you’re setting.
This is necessary so any accounting system application code associated with these
events can verify the field’s value. Although Microsoft Dynamics GP doesn’t
perform this verification for all fields, they will perform verification for fields that
affect business logic (such as an invoice discount percent, or a tax amount) or for
add-on-the-fly fields. If the accounting system determines that the field’s value is
invalid, it will first display its error dialog, followed by the VBA error dialog.
**Solution:**
There are three ways to avoid this type of error:

- Whenever possible, set a field’s value from “inside” the field, after it has gained focus, using the field’s `AfterGotFocus` or `BeforeGotFocus` field events. If your event procedure sets an invalid value, the accounting system will still display a dialog, but VBA won’t generate the error:

```vba
Private Sub ShippingMethod_AfterGotFocus()
    'Set the Shipping Method field to a value that doesn't exist.
    ShippingMethod = "NEW"
End Sub
```

- If you set the value from “outside” the field, before the field has gained focus, use the `Focus` method with the `setValue` parameter in your event procedure. The `Focus` method moves the focus to the field, then sets it to the value of the `setValue` parameter:

```vba
Private Sub CustomerID_AfterUserChanged()
    'Move the focus to the field, then set the value
    ShippingMethod.Focus("NEW")
End Sub
```

- Only set fields from “outside” the field if you know the accounting system isn’t performing field verification. Microsoft Dynamics GP performs verification for fields that affect business logic (such as an invoice discount percent, or a tax amount) or for add-on-the-fly fields. You can set the value of add-on-the-fly fields only if the add-on-the-fly value already exists as a record (such as an existing shipping method).

**Error 1008  Unsafe Operation. An attempt was made to set a value into an application field that is disabled. This operation could compromise the integrity of the application.**

**Situation:**
You attempted to set the value of a field that was locked or disabled by the accounting system.

**Solution:**
If Microsoft Dynamics GP application code disables or locks a field (the field’s `Enabled` property is False, or its `Locked` property is True), you cannot set its value. The accounting system disables fields when they don’t apply in a given situation, and locks fields to preserve the integrity of accounting data (such as document totals). However, you can set the value of fields you’ve disabled or locked through VBA.

**Error 80040208  This operation cannot be performed while the window is firing events.**

**Situation:**
You attempted to use the `PullFocus` method with a window while VBA window events or window field events are occurring.

**Solution:**
The `PullFocus` method forces a window to “lose” the focus and should only be used in windows where no pending VBA window events or field events are occurring. VBA events rely on the focus being present to run properly.
Glossary

**Additional headers and footers**
Report bands that are printed when a specified field changes.

**Adds-allowed grid**
A grid that allows the user to enter and save new information using the grid.

**Ask dialog**
A modal dialog box that contains one or more buttons, allowing the user to make a selection, and displays message text.

**Band**
A section of a report, such as the body, report header, report footer or a page header.

**Band events**
A VBA event that occurs just before a report band prints.

**Browse-only grid**
A grid that allows the user to browse through items in the window, but not make any changes.

**Button drop list field**
A button field that “drops” to allow a user to select one item from a list of values. VBA uses a numeric value to identify the selected item in the list.

**Calculated field**
A report field containing an expression that combines fields in a report’s table, report fields, constants, functions and operators. You can set or return the value of user-defined calculated fields. You can only return the value of Microsoft Dynamics GP calculated fields.

**Caption**
A field caption is a text label that shows the user the information that’s displayed or that can be entered in the corresponding window field. A window caption is the title of the window.

**Check box field**
A field that allows a user to mark or unmark a selection. In VBA, the value 1 indicates a marked item; a 0 indicates an unmarked item.

**Combo box field**
A list field that allows a user to enter a text value or choose that value from a list. In VBA, a string value specifies an item in a combo box.

**Composite field**
A field made up of multiple segments. Microsoft Dynamics GP account numbers are composite fields.

**Concatenate**
To connect two or more strings to form a single string.

**Control field**
A unique field in a window that’s used to control access to a record. In Microsoft Dynamics GP, control fields are unique values, such as document numbers or customer IDs.

**Currency field**
A field that allows the user to enter a currency amount.

**Disabled field**
A field that is unavailable to or cannot be chosen by a user. A disabled button or field appears dimmed.

**Drop-down list field**
A field that “drops” to allow users to select one item from a list. VBA uses a numeric value to identify the selected item in the list.

**DUOS**
An acronym for Dynamic User Object Store. The DUOS lets you use VBA to create, store and retrieve user-definable data objects.

**Editable field**
A field whose contents can be changed by the user.

**Editable grid**
A scrolling window that allows the user to edit items in the window.

**Event**
An action for a given object where associated application code (an event procedure) runs.

**Event procedure**
VBA code that executes when a given event occurs.

**Field**
A field contains a single piece of information. A field can be displayed in a window or on a report. The type of information displayed depends on the field’s type.

**Focus**
The indicator that shows the object being controlled in the current window.

**Format**
The extra characters, spacing and attributes that can be applied to a field when data is entered or displayed.

**Forms dictionary**
The dictionary that stores user-modified resources for Microsoft Dynamics GP. This dictionary is created when the Modifier is accessed for the first time.

**Getstring dialog**
A modal dialog that contains a data entry field and OK and Cancel buttons.

**Grid**
A window used to display information from a table (also known as a scrolling window). Grids allow the user to scroll through records in the table. There are three types of grids: browse-only, editable and addsa-allowed.

**Grid event**
A VBA event that occurs when a grid fills, or a grid’s line changes, gains focus or loses focus.

**Group box**
A box drawn around a group of check boxes or other fields to visually group the items.

**Integer field**
A field that allows the user to enter a number between -32,768 and 32,767.

**Keyable length**
The number of characters that can be typed in a field.

**Layout window**
A window in the Modifier and Report Writer that allows users to design the layout of a window, grid (scrolling window) or report.

**List box field**
A field that allows users to select one static text value from a list. VBA uses a numeric value to identify the selected item in the list.

**List field**
Any list box, drop-down list, multi-select list, button drop list, combo box or visual switch field.

**Method**
An action in VBA that you can perform for a given object. Methods include opening and closing a window or moving a field in a window.

**Modal dialog**
A window that contains no operating system controls and can’t be resized. Modal windows are used when you require the user to perform an action before continuing.

**Modal dialog event**
A VBA event that occurs when a modal dialog appears.

**Modifier**
A tool that allows the user to change the user interface of Microsoft Dynamics GP windows. A forms dictionary stores these modifications.

**Multi-select list box field**
A field from which the user can select one or more items. VBA uses a 32-bit numeric value to identify the selected item in the list.
Object model
The relationship objects have to each other within a given application.

Package files
Special text files that are used to deliver customizations made with the Modifier, VBA, and the Report Writer.

Page footer
A report band placed at the bottom of every report page. The page footer often includes administrative information, such as the name of the person running the report.

Page header
A report band placed at the top of every report page. The page header typically includes fields for the page number, date and time.

Palette
A type of window used for navigation.

Parameter
A value passed to or returned from a VBA statement, function or procedure.

Pixel
The smallest graphical element displayed on a monitor. The pixel is the smallest unit of measurement in windows.

Procedure
A named sequence of statements executed as a single unit. An event procedure is the most common type of procedure used in a VBA project.

Project file
A file containing VBA code, user forms and objects. The project is named based on the dictionary. For instance, the project for Microsoft Dynamics GP is named DYNAMICS.VBA. For third-party applications integrating with the accounting system, this file is name.VBA, where name is the name of the integrating application’s dictionary.

Prompt
See Caption.

Property
A characteristic of an object in VBA that you can retrieve or set. Properties include the title of a window, or the value of a field.

Push button field
A field the user can click to perform an action.

Radio button field
A field that allows a single selection to be made from a group of two or more options. Radio buttons must be part of a radio group. VBA uses a numeric value to identify the selected item in the radio group.

Read only
A property access mode that indicates the property’s value can only be returned.

Read/write
A property access mode that indicates the property’s value can be returned or set.

Report event
A VBA event that occurs when you print a report. Report events occur when the report starts, just before a given band within the report prints, and when the report ends.

Report field
Any field that appears on a report. You can use VBA to return the value of report fields if they’re table fields, accounting system calculated fields, or legends. You can set the value of report fields only if they’re user-defined calculated fields or legend fields.

Report footer
A report band that prints at the bottom of the last report page. Reports can include report footers, page footers and additional footers.

Report legend
A report field that contains data passed to the report at runtime, before it prints. Legend fields typically include information about the report, such as the range of records the user chose for the report.

Report Writer
A tool that allows you to modify reports, or design new reports. A reports dictionary stores these changes or additions.

Required field
A field in which a value must be entered to process information, such as an ID or customer name. You can use VBA to mark fields as required.

Required fields are displayed in a different color or type style when a user chooses the Show Required Fields menu item.

Resource Descriptions tool
A tool that displays information about the accounting system’s fields, windows and tables. To access this tool from within the accounting system, point to Resource Descriptions in the Tools menu, then choose Tables, Fields or Windows.

Scrolling window
See Grid.

Segment
One portion of a composite field. Microsoft Dynamics GP uses composite fields for account numbers.

String field
A field that allows the user to enter up to 255 ASCII characters. VBA include any formatting when setting or returning the value of a string field.

Tab sequence
The order in which the focus moves from one field to the next in a window when a user presses the TAB key.

Text field
A field that allows the user to enter up to 32,000 ASCII characters. VBA references the string value of a text field.

VBA
Visual Basic for Applications. A development environment used to interact with and control objects within host applications.

Visual switch field
A field that displays a series of text or picture items. Clicking the field displays the next item in the series. VBA uses a numeric value to identify the item selected in a visual switch.

Window
The work area used to enter and display information in an application.

Window event
A VBA event that occurs when a window opens, activates, closes or when a modal dialog appears.

Window field
A field within a Microsoft Dynamics GP window.

Window field event
A VBA event that occurs when a field gains focus, loses focus, or when its value changes.
Index

A
account numbers
  setting account segments 137
  using 43
accumulator fields, in VBA 70
Activate method
  described 96
  using 21
activating windows 21
additional footer
  defined 229
  using the BeforeAF event 68
additional header
  defined 229
  using the BeforeAH event 68
adds-allowed grids
  defined 229
  described 58
ADO connection, creating for database 215
AfterActivate event
  described 97
  using 17
AfterClose event
  described 98
  using 17
AfterGotFocus event
  avoiding errors 38
  button drop list fields 40
  described 126
  using 27
AfterLineChange event
  described 154
  using 60
AfterLineGotFocus event
  described 155
  using 59
AfterLineLostFocus event
  described 156
  using 61
AfterLostFocus event
  button drop list fields 40
  described 127
  related errors 226
  using 30
AfterModalDialog event
  described 99
  using 18
AfterOpen event
  described 101
  using 16
AfterUserChanged event
  button drop list fields 40
  combo box fields 42
  composite fields 43
  described 128
  drop-down list fields 46
  list box fields 48
  AfterUserChanged event (continued)
  multi-select list box fields 49
  push button fields 50
  radio button fields 51
  related errors 226
  using 28
  visual switch fields 54
alert messages, see modal dialogs
ask dialogs
  defined 229
  described 17, 99
  responding to 17
B
band, defined 229
band events
  defined 229
  returning data from multiple bands 71
  suppressing report data 67
  using 66
BeforeActivate event
  described 102
  using 17
BeforeAF event
  described 172
  using 68
BeforeAH event
  described 173
  using 68
BeforeBody event
  described 174
  using 68
BeforeClose event
  described 103
  using 16
BeforeGotFocus event
  avoiding errors 38
  button drop list fields 40
  described 129
  using 26
BeforeLineChange event
  described 157
  using 60
BeforeLineGotFocus event
  described 158
  using 59
BeforeLineLostFocus event
  described 159
  using 61
BeforeLinePopulate event
  described 160
  using 61
BeforeLostFocus event
  button drop list fields 40
  described 130
  using 30
BeforeModalDialog event
  described 104
  using 18
BeforeOpen event
  described 106
  using 15
BeforePF event
  described 175
  using 68
BeforePH event
  described 176
  using 67
BeforeRF event
  described 177
  using 68
BeforeRH event
  described 178
  using 67
BeforeUserChanged event
  button drop list fields 40
  combo box fields 42
  composite fields 43
  described 131
  drop-down list fields 46
  list box fields 48
  multi-select list box fields 49
  push button fields 50
  radio button fields 51
  related errors 226
  using 28
  visual switch fields 54
body, using the BeforeBody event 68
browse-only grids
  defined 229
  described 58
button drop list field
  defined 229
  programming 40
button fields, see push button field
C
calculated fields
  defined 229
  referencing in reports 70
Cancel button, pushing from VBA 50
Cancel method
  described 179
  using with reports 69
canceling
  field processing
    got focus events 26
    lost focus events 30
    user changed events 28
  grid processing 60
  report processing 69
  window processing 16
caption
  applied to Microsoft Dynamics GP fields 132
  defined 229
Caption property
  check box fields 41
  described for fields 132
  described for windows 107
Caption property (continued)
  push button fields 50
  radio button fields 51
  related errors 224
  windows 22

Changed event
  button drop list fields 40
  described for fields 133
  push button fields 50
  using 29

Changed property
  described for grids 161
  described for windows 108

title field
  defined 229
  programming 41

circular references, avoiding 12

Clear button, pushing from VBA 50

Close method
  described 109
  using with windows 20

collections
  creating a DUOS collection 77
  DUOS object collection 10
  DUOS property collection 10

combo box field
  defined 229
  programming 42

copying fields
  for numeric calculations 35

guidelines for setting 137

related errors 225

using 31

distributing a VBA project 87

documentation
  other VBA resources 2
  symbols and conventions 3

drop-down list field
  defined 229
  programming 46

DUOS
  chapter 75-85
  creating a collection 77
  creating data objects 77
  data objects
  creating 77
  deleting 78, 83
  described 75
  printing 85
  retrieving 78, 82
  saving 80
  tables for storing 76
  updating 78
  defined 229
  deleting a data object 78, 83
  described 75
  in the object model 9, 10

DUOS (continued)
  overview of the object model 75
  printing a data object 85
  programming 75
  property and method list 79
  retrieving data objects 78
  saving a data object 80
  structure 76
  updating a data objects 78
  using 80
  using modal dialogs to synchronize 18

DUOSObject object library 199-201
DUOSObjectCombineID method, described 186
DUOSObjectExtractID method, described 187
DUOSObjects collection library 191-197
DUOSObjectsGet method
  described 188
  using 77

DUOSProperties collection library 203-208
DUOSProperty object library 209-211
Dynamic User Object Store, see DUOS

editable fields, defined 229

editable grids
  defined 229
  described 58

Empty property
  described 134
  using 34

Enabled property
  described 135
  related errors 225, 227
  using 31

End event
  described 180
  using 66

errors
  appendix 223-227
  error 1001 224
  error 1002 224
  error 1003 225
  error 1004 225
  error 1005 226
  error 1006 226
  error 1007 226
  error 1008 227
  error 80040208 227
  handling VBA errors 223
  VBA error list 223
  when setting fields 37

event procedures
  see also events
  defined 229

EventMode property
  described for grids 162
  described for reports 181
  described for windows 110

232  VBA DEVELOPER’S GUIDE
EventMode property (continued)
  using with grids 62
  using with reports 65
  using with windows 22

events
  defined 229
  described 11
  for grids 59
  for reports 64
  for window fields 25
  for windows 15

Exists property, described 192, 205
expand button, pushing from VBA 50, 54
exporting data to other applications 72
exporting package files 89
extender properties, described 143
external components, referencing 87

F
  field object library 125-152
  fields
    see also window fields
    defined 229
    grid fields 56
    report fields 70
    window fields 33
    working with report fields 66
    working with report legends 65
  focus, defined 229
  Focus method
    avoiding errors 38
    described 136
    using 32
  FocusSeg method
    composite fields 43
    described 137
    using 32
  Format function, use with report fields 70
  formats
    composite format 43
    currency format 44
    date format 45
    defined 229
    integer format 47
    report field format 70
    string format 52
  forms
    defined 229
    using with VBA 20
  forms dictionary, defined 229

G
  getstring dialogs
    defined 229
    responding to 100
  got focus events, using 26
  grid events
    defined 229
    line change events 60
    line got focus events 59
    line lost focus events 61
  grids
    see also grid events
    adding to a project 56
    adding window fields from 56
    canceling processing 60
    changing a grid’s name 62
    chapter 55-62
    defined 229
    filtering grid content 61
    in the object model 9, 10
    programming 55, 57
    property and method list 62
    removing from a project 56
    types of 58
    using grid events 59
    using VBA with modified grids 59
  group box, defined 229

H
  Height property
    described for fields 138
    described for windows 111
    using 22
  Hide method
    described for grids 163
    described for windows 112
    using 21
  hiding
    report fields 71
    window fields 32
    windows 21
  ID property, described 200
  IDs, retrieving user ID 219
  Import Package File window 90
  ImportPackageFile method, use with report fields 70
  Importing package files 90
  integer field
    defined 229
    programming 47
  IntercompanyID property, described for
    UserInfo object 216
  IsLoaded property, described 113
  Item property
    described for a DUOSObjects
      collection 193
    described for a DUOSProperties
      collection 206
  lightweight symbol 3
  line change events, using 60
  line got focus events, using 59
  line lost focus events, using 61
  line populate events, using 61
  list box field
    defined 229
    programming 48
  list fields
    button drop list 40
    defined 229
    drop-down list 46
    list box 48
    multi-select list box 49
  Locked property
    described 140
    related errors 225, 227
    using 31
  locking window fields 31
  lookup button, pushing from VBA 50
  lost focus events, using 29

M
  margin notes 3
  messages, see modal dialogs
  method list
    DUOS 79
    grids 62
    reports 69
    window fields 31
    windows 19
  methods
    defined 229
    described 11
  Microsoft Office, exporting data to Office
    applications 72
  Missing File dialog 87
  modal dialog events
    defined 229
    using 17
  modal dialogs
    changing the contents of 18
    defined 229
    delete dialog and the DUOS 83
    programmatically answering 18
    responding to 17, 18
    save dialog and the DUOS 81
    types 99
    using with the DUOS 18
  Modifier
    defined 229
    using VBA with grids 59
    using VBA with modified windows 15
    using with VBA 12
  Move method
    described for fields 141
    described for grids 165
Move method (continued)
described for windows 115
moving windows 22
multi-select list box field
defined 229
programming 49

N
Name property
described for a DUOSObjects
collection 194
described for a DUOSProperty object
210
described for fields 142
described for grids 166
described for reports 183
described for windows 116
using with fields 33
using with grids 62
using with reports 69
using with windows 22
names
retrieving company name 214
retrieving user name 220
note button, pushing from VBA 50
numeric fields, converting for numeric
calculations 35

O
object model
defined 230
using the object model 9
Object property, described 143
objects
DUOS data object 10
DUOS property object 10
field object, described 10
grid object, described 10
in the object model 9
report object, described 10
user information object, described 10
using with events 11
using with methods 11
using with properties 11
window field object, described 23
window object, described 10, 13
Open method
described 117
using with windows 20
opening windows, guidelines 117
overwriting customizations with package
files 91

P
package files
contents 88
defined 230
described 88
exporting 89
import/export issues 91
importing 90
overwriting customizations 91
page footer
defined 230
using the BeforePF event 68
page header
defined 230
using the BeforePH event 67
palettes
defined 230
pushing a stick pin from VBA 54
parameters, defined 230
Parent property, described 144
password protecting a project 88
pixel, defined 230
prerequisites for VBA 2
printing
DUOS data objects 85
reports, and VBA events that occur 64
privileges, required to create or edit VBA
code 7
procedure, defined 230
product support for VBA 2
programming
grids 57
reports 64
the Dynamic User Object Store
(DUOS) 77
window fields 24
windows 14
project
see also project file
adding grids 56
adding report fields 64
adding reports 63
adding window fields 23
adding windows 13
locking 88
removing grids 56
removing reports 63
removing window fields 24
removing windows 14
renaming grids 62
renaming window fields 33
renaming windows 22
Project Explorer
described 8
viewing grids 56
viewing reports 63
viewing window fields 23
viewing windows 13
project file
defined 230
described 8
for third-party applications 8
prompt, see caption
properties
custom, described 143
defined 230
described 11
related errors 225
standard, described 143
Properties property, described 201
property list
DUOS 79
grids 62
reports 69
window fields 31
windows 19
PullFocus method
described 118
related errors 227
push button field
defined 230
programming 50
radio button field
defined 230
programming 51
radio button group 51
read only, defined 230
read/write, defined 230
references
to external components 87
to third-party dictionaries 12
registration keys, for VBA 87
Remove method
described for a DUOSObject
collection 195
described for DUOSProperties
collection 208
report body, using the BeforeBody event
68
report events
band events 66
defined 230
Start/End events 65
report fields
accumulator fields 70
adding to a project 64
calculated fields 70
defined 230
format of returned values 70
guidelines for use 70
hiding and showing 71
in the object model 9
legend fields 65
returning 66
setting 66
suppressing 67
table fields 70
report footer
defined 230
using the BeforeRF event 68
report header, using the BeforeRH event
67
report legends
defined 230
setting from a Start event 65
report object library 171-184
report sections, see band events
Report Writer
defined 230

INDEX

234  VBA DEVELOPER’S GUIDE
Report Writer (continued)
- using VBA with modified reports 65

- see also: report events
- adding to a project 63
- canceling 69
- chapter 63-73
- exporting data to other applications 72
- in the object model 9, 10
- programming 63
- property and method list 69
- removing a report 63
- returning data from an entire report 71
- returning report fields 66
- setting report fields 66
- using legend fields 65
- using report events 64
- using VBA with modified reports 65

- required fields
  - defined 230
  - using the Required property 33

- Required property
  - described for fields 145
  - described for windows 120
  - related errors 225
  - using with DUOS 80
  - using with window fields 33

- resizing windows 22

- Resource Descriptions tool
  - defined 230
  - described 39

- retrieving DUOS data objects 82
- returning report fields
  - from a Start event 65
  - from band events 66

- returning window fields, guidelines 35
- runtime errors, see errors

**S**

- Save button
  - pushing from VBA 50
  - using with DUOS 80

- saving DUOS data objects 80

- scrolling windows
  - see also: grids
  - pushing expand and shrink buttons 54

- security, setting access to customized resources 90

- segments
  - controlling the focus within a composite 32
  - defined 230
  - guidelines for setting 137
  - using with a composite 43

- setting report fields
  - using a Start event 65
  - using band events 66

- setting window fields
  - guidelines 36
  - related errors 226
  - using a BeforeGotFocus event 26
  - using a BeforeUserChanged event 28
  - using an AfterGotFocus event 27

- Show method
  - described for grids 167
  - described for windows 121

- showing
  - report fields 71
  - window fields 32
  - windows 21

- shrink button, pushing from VBA 54

- SortByExisting property, described 196

- Start event
  - described 184
  - using 65

- stick pin button, pushing from VBA 54

- storing additional data, chart 75-85

- string field
  - defined 230
  - programming 52

- support available for VBA 2

- SY_User_Object_Store table, described 76

- symbols in documentation 3

- system database name, retrieving 217

- SystemDatabaseName property
  - described for UserInfo object 217

**T**

- tab sequence, defined 230

- table fields, for reports, using with VBA 70

- TabStop property, described 146

- technical support for VBA 2

- text field
  - defined 230
  - programming 53

- third-party dictionaries
  - project file 8
  - using VBA with 12

- title, see caption

- Top property
  - described for fields 147
  - described for grids 168
  - described for windows 122
  - using 22

**U**

- user changed events, using 27

- user information, in the object model 10

- UserDate property, described for UserInfo object 218

- UserID property, described for UserInfo object 219

- UserInfo object library 213-220

- UserInfoGet method, described 189

- UserName property, described for
  - UserInfo object 220

- Using VBA, part 6-91

**V**

- Value property
  - button drop list fields 40
  - check box fields 41
  - combo box fields 42
  - composite fields 43
  - currency fields 44
  - date fields 45
  - described for a DUOSProperty object 211
  - described for fields 148
  - drop-down list fields 46
  - guidelines for returning window fields 35
  - guidelines for setting fields 36
  - integer fields 47
  - list box fields 48
  - multi-select list box fields 49
  - push button fields 50
  - radio button fields 51
  - related errors 226
  - string fields 52
  - text fields 53
  - using with report fields 66
  - using with window fields 33
  - visual switch fields 54

- ValueSeg property
  - composite fields 43
  - described 150
  - related errors 226
  - using 34
  - using with the FocusSeg method 137

**VBA**

- creating the project file 8
  - defined 230
  - described 2
  - documentation 2
  - enabling 87
  - packaging a project 88
  - prerequisites 2
  - registration keys 3, 87
  - using with fields 24
  - using with grids 57
  - using with reports 63
  - using with the Dynamic User Object Store (DUOS) 75
  - using with the Modifier 12
  - using with windows 14

- Visual Basic Editor 7

- VBA Object Library, part 94-220

- VBA Overview, chapter 7-12

- VBAGlobal object library 185-189

- Visible property
  - described for fields 151
  - described for grids 169
  - described for windows 123
  - related errors 225
  - using with report fields 71
  - using with window fields 32
  - using with windows 21

- Visual Basic Editor, described 7
Visual Basic for Applications, see VBA
visual switch field
defined 230
programming 54

W
warning messages, see modal dialogs
warning symbol 3
Width property
described for fields 152
described for windows 124
using 22
window activate events, using 17
window close events, using 16
window events
activate events 17
close events 16
defined 230
modal dialog events 17
open events 15
window field events
Changed event 29
defined 230
got focus events 26
lost focus events 29
user changed events 27
window fields
see also window field events
accessing fields in child windows 20
adding to a project 23
button drop list 40
canceling field processing
using got focus events 26
using lost focus events 30
using user changed events 28
chapter 23-54
check box 41
combo box 42
composite field 43
controlling the focus 32
currency field 44
date field 45
defined 230
setting
dealing with VBA errors 37
during window open event 15
using a BeforeGotFocus event 26
using a BeforeUserChanged
event 28
using an AfterGotFocus event 27
showing and hiding 32
string field 52
text field 53
using VBA with new fields 12
using window field events 25
using with grids 56
Value property 33
ValueSeg property 34
verifying field data 36
visual switch field 54
window object library 95-124
window open events, using 15
windows
see also window events
activating 21
adding to a project 13
as part of a form 20
canceling a window close 16
changing a window’s caption 22
chapter 13-22
closing other windows 17
defined 230
determining which invoke VBA
events 107
hiding 21
in the object model 9, 10
moving and resizing 22
opening and closing 20
opening invisibly 15
programming 13, 14
property and method list 19
removing window objects 14
renaming windows 22
setting the event mode 22, 62
using VBA with modified windows
15
using window events 15

window fields (continued)
progression of events 37
property and method list 31
push button field 50
radio button field 51
removing from a project 24
renaming window fields 33
setting
dealing with VBA errors 37
during window open event 15
using a BeforeGotFocus event 26
using a BeforeUserChanged
event 28
using an AfterGotFocus event 27
showing and hiding 32
string field 52
text field 53
using VBA with new fields 12
using window field events 25
using with grids 56
Value property 33
ValueSeg property 34
verifying field data 36
visual switch field 54
window object library 95-124
window open events, using 15
windows
see also window events
activating 21
adding to a project 13
as part of a form 20
canceling a window close 16
changing a window’s caption 22
chapter 13-22
closing other windows 17
defined 230
determining which invoke VBA
events 107
hiding 21
in the object model 9, 10
moving and resizing 22
opening and closing 20
opening invisibly 15
programming 13, 14
property and method list 19
removing window objects 14
renaming windows 22
setting the event mode 22, 62
using VBA with modified windows
15
using window events 15

window fields (continued)
progression of events 37
property and method list 31
push button field 50
radio button field 51
removing from a project 24
renaming window fields 33
setting
dealing with VBA errors 37
during window open event 15
using a BeforeGotFocus event 26
using a BeforeUserChanged
event 28
using an AfterGotFocus event 27
showing and hiding 32
string field 52
text field 53
using VBA with new fields 12
using window field events 25
using with grids 56
Value property 33
ValueSeg property 34
verifying field data 36
visual switch field 54
window object library 95-124
window open events, using 15
windows
see also window events
activating 21
adding to a project 13
as part of a form 20
canceling a window close 16
changing a window’s caption 22
chapter 13-22
closing other windows 17
defined 230
determining which invoke VBA
events 107
hiding 21
in the object model 9, 10
moving and resizing 22
opening and closing 20
opening invisibly 15
programming 13, 14
property and method list 19
removing window objects 14
renaming windows 22
setting the event mode 22, 62
using VBA with modified windows
15
using window events 15