Please refer to the following table for more information on other notable objects in the Access object model.
| DependencyInfo | SmartTagActions | \_M |
| DependencyObjects | SmartTagProperties | \_S\_ |
| AccessObject | SmartTagProperty | Proj |
| AccessObjectProperties | | Rep |
| AccessObjectProperty | | |
| SmartTags | | C\_ |

Legend

Object and collection
Object only
New Objects

For the latest information about programming with Microsoft Access, including product news, technical articles, downloads, and samples, visit the Microsoft Office Developer Center on the Microsoft Developer Network (MSDN) Web site.

The following table lists objects added to the Microsoft Office Access 2003 object model.

<table>
<thead>
<tr>
<th>Object</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AdditionalData</strong></td>
<td>Represents the collection of tables and queries that will be included with the parent table that is exported by the ExportXML method.</td>
</tr>
<tr>
<td><strong>AutoCorrect</strong></td>
<td>Represents the AutoCorrect functionality of TextBox and ComboBox objects.</td>
</tr>
<tr>
<td><strong>DependencyInfo</strong></td>
<td>Represents the dependency information for an AccessObject object.</td>
</tr>
<tr>
<td><strong>DependencyObjects</strong></td>
<td>Represents the dependency information for an AccessObject object.</td>
</tr>
<tr>
<td><strong>SmartTag</strong></td>
<td>Represents a smart tag that has been added to a control on a form, report, or data access page.</td>
</tr>
<tr>
<td><strong>Smart_Tag_Action</strong></td>
<td>Represents a single action for a smart tag.</td>
</tr>
<tr>
<td><strong>Smart_Tag_Actions</strong></td>
<td>Represents a collection of actions for an individual smart tag or a type of smart tag.</td>
</tr>
<tr>
<td><strong>Smart_Tag_Properties</strong></td>
<td>A collection of SmartTagProperty objects that represents the properties related to a smart tag.</td>
</tr>
<tr>
<td><strong>SmartTagProperty</strong></td>
<td>Represents a single instance of a custom property for a smart tag.</td>
</tr>
<tr>
<td><strong>SmartTags</strong></td>
<td>Represents the collection of smart tags for a control on a form, report, or data access page.</td>
</tr>
</tbody>
</table>
New Properties (Alphabetical List)

For the latest information about programming with Microsoft Access, including product news, technical articles, downloads, and samples, visit the Microsoft Office Developer Center on the [Microsoft Developer Network (MSDN)](https://developer.microsoft.com) Web site.

The following table lists properties added to the Microsoft Office Access 2003 object model (sorted alphabetically).

<table>
<thead>
<tr>
<th>New Property</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AccessConnection</strong></td>
<td><strong>CodeProject, CurrentProject</strong></td>
</tr>
<tr>
<td><strong>AutoCorrect</strong></td>
<td><strong>Application</strong></td>
</tr>
<tr>
<td><strong>AutomationSecurity</strong></td>
<td><strong>Application</strong></td>
</tr>
<tr>
<td><strong>Dependants</strong></td>
<td><strong>DependencyInfo</strong></td>
</tr>
<tr>
<td><strong>Dependencies</strong></td>
<td><strong>DependencyInfo</strong></td>
</tr>
<tr>
<td><strong>DisplayAutoCorrectOptions</strong></td>
<td><strong>AutoCorrect</strong></td>
</tr>
<tr>
<td><strong>FetchDefaults</strong></td>
<td><strong>Form</strong></td>
</tr>
<tr>
<td><strong>InsufficientPermissions</strong></td>
<td><strong>DependencyInfo</strong></td>
</tr>
<tr>
<td><strong>IsMissing</strong></td>
<td><strong>SmartTag</strong></td>
</tr>
<tr>
<td><strong>NewFileTaskPane</strong></td>
<td><strong>Application</strong></td>
</tr>
<tr>
<td><strong>OutOfDateObjects</strong></td>
<td><strong>DependencyInfo</strong></td>
</tr>
<tr>
<td><strong>SmartTagActions</strong></td>
<td><strong>SmartTag</strong></td>
</tr>
<tr>
<td><strong>SmartTags</strong></td>
<td><strong>ComboBox, Control, Label, ListBox, TextBox</strong></td>
</tr>
<tr>
<td><strong>UnsupportedObjects</strong></td>
<td><strong>DependencyInfo</strong></td>
</tr>
<tr>
<td><strong>XML</strong></td>
<td><strong>SmartTag</strong></td>
</tr>
</tbody>
</table>
New Properties (by Object)

For the latest information about programming with Microsoft Access, including product news, technical articles, downloads, and samples, visit the Microsoft Office Developer Center on the Microsoft Developer Network (MSDN) Web site.

The following table lists properties added to the Microsoft Office Access 2003 object model (sorted by object name).

<table>
<thead>
<tr>
<th>Object</th>
<th>New Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>AutoCorrect, AutomationSecurity, NewFileTaskPane</td>
</tr>
<tr>
<td>AutoCorrect</td>
<td>DisplayAutoCorrectOptions</td>
</tr>
<tr>
<td>CodeProject</td>
<td>AccessConnection</td>
</tr>
<tr>
<td>ComboBox</td>
<td>SmartTags</td>
</tr>
<tr>
<td>Control</td>
<td>SmartTags</td>
</tr>
<tr>
<td>CurrentProject</td>
<td>AccessConnection</td>
</tr>
<tr>
<td>DependencyInfo</td>
<td>Dependants, Dependencies, InsufficientPermissions, OutOfDateObjects, UnsupportedObjects</td>
</tr>
<tr>
<td>Form</td>
<td>FetchDefaults</td>
</tr>
<tr>
<td>Label</td>
<td>SmartTags</td>
</tr>
<tr>
<td>ListBox</td>
<td>SmartTags</td>
</tr>
<tr>
<td>SmartTag</td>
<td>IsMissing, SmartTagActions, XML</td>
</tr>
<tr>
<td>TextBox</td>
<td>SmartTags</td>
</tr>
</tbody>
</table>
New Methods (Alphabetical List)

For the latest information about programming with Microsoft Access, including product news, technical articles, downloads, and samples, visit the Microsoft Office Developer Center on the Microsoft Developer Network (MSDN) Web site.

The following table lists methods added to the Microsoft Office Access 2003 object model (sorted alphabetically).

<table>
<thead>
<tr>
<th>New Method</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>CreateAdditionalData</td>
<td>Application</td>
</tr>
<tr>
<td>Execute</td>
<td>SmartTagAction</td>
</tr>
<tr>
<td>GetDependencyInfo</td>
<td>AccessObject</td>
</tr>
<tr>
<td>IsDependentUpon</td>
<td>AccessObject</td>
</tr>
<tr>
<td>TransformXML</td>
<td>Application</td>
</tr>
<tr>
<td>UpdateDependencyInfo</td>
<td>CodeProject, CurrentProject</td>
</tr>
</tbody>
</table>
New Methods (by Object)

For the latest information about programming with Microsoft Access, including product news, technical articles, downloads, and samples, visit the Microsoft Office Developer Center on the [Microsoft Developer Network (MSDN)](https://developer.microsoft.com) website.

The following table lists methods added to the Microsoft Office Access 2003 object model (sorted by object name).

<table>
<thead>
<tr>
<th>Object</th>
<th>New Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>AccessObject</td>
<td>GetDependencyInfo, IsDependentUpon</td>
</tr>
<tr>
<td>Application</td>
<td>CreateAdditionalData, TransformXML</td>
</tr>
<tr>
<td>CodeProject</td>
<td>UpdateDependencyInfo</td>
</tr>
<tr>
<td>CurrentProject</td>
<td>UpdateDependencyInfo</td>
</tr>
<tr>
<td>SmartTagAction</td>
<td>Execute</td>
</tr>
</tbody>
</table>
AccessObjectProperties Collection

Multiple objects \[\text{AccessObjectProperties}\]
\[\text{AccessObjectProperty}\]

The \text{AccessObjectProperties} collection contains all of the custom \text{AccessObjectProperty} objects of a specific instance of an object. These \text{AccessObjectProperty} objects (which are often just called properties) uniquely characterize that instance of the object.
Using the AccessObjectProperties Collection

Use the **AccessObjectProperties** collection in **Visual Basic** or in an **expression** to refer to properties of the **CurrentProject**, **CodeProject**, or **AccessObject** object. For example, you can enumerate the **AccessObjectProperties** collection to set or return the values of properties of an individual report.

**Note**  The **AccessObjectProperties** collection isn't accessible for objects derived from the **CurrentData** object (for example, CurrentData.AllTables!Table1). For objects derived in this manner, you can only access their built-in properties by direct calls to the desired property (for example, CurrentData.AllTables!Table1.Name).

To add a user-defined property to an existing instance of an object, first define its characteristics and add it to the collection with the **Add** method. Referencing a user-defined **AccessObjectProperty** object that has not yet been appended to an **AccessObjectProperties** collection will cause an error, as will appending a user-defined **AccessObjectProperty** object to an **AccessObjectProperties** collection containing an **AccessObjectProperty** object of the same name.

You can use the **Remove** method to remove user-defined properties from the **AccessObjectProperties** collection.

**Note**  A built-in or user-defined **AccessObjectProperty** object is associated only with the specific instance of an object. The property isn't defined for all instances of objects of the selected type.

To refer to a built-in or user-defined **AccessObjectProperty** object in a collection by its ordinal number or by its **Name** property setting, use any of the following syntax forms:

```plaintext
CurrentProject.AllForms("Form1").Properties(0)
CurrentProject.AllForms("Form1").Properties("name")
CurrentProject.AllForms("Form1").Properties!"name"
```

With the same syntax forms, you can also refer to the **Value** property of a **AccessObjectProperty** object. The context of the reference will determine whether you are referring to the **AccessObjectProperty** object itself or the
Value property of the AccessObjectProperty object.

Note Properties in the AccessObjectProperties collection are not stored and can be lost when the object they are associated with is checked in or out using the Source Code Control add-in.
AllDataAccessPages Collection

Multiple objects - AllDataAccessPages
  - AccessObject
  - AccessObjectProperties

The AllDataAccessPages collection contains an AccessObject object for each data access page in the CurrentProject or CodeProject object.

Note  Although a Microsoft Access project (.adp) or Microsoft Access database (.mdb) can appear to contain data access pages, these pages are actually stored in files that are external to the project or database.
Using the AllDataAccessPages Collection

The `CurrentProject` or `CodeProject` object has an `AllDataAccessPages` collection containing `AccessObject` objects that describe instances of all data access pages. For example, you can enumerate the `AllDataAccessPages` collection in Visual Basic to set or return the values of properties of individual `AccessObject` objects in the collection.

**Tip**

The `For Each...Next` statement is useful for enumerating a collection.

You can refer to an individual `AccessObject` object in the `AllDataAccessPages` collection either by referring to the item by name, or by referring to its index within the collection. If you want to refer to a specific data access page in the `AllDataAccessPages` collection, it's better to refer to the item by name because the index may change.

The `AllDataAccessPages` collection is indexed beginning with zero. If you refer to a data access page by its index, the first data access page is `AllDataAccessPages(0)`, the second data access page is `AllDataAccessPages(1)`, and so on.

**Note** To list all open data access pages in the database, use the `IsLoaded` property of each `AccessObject` object in the `AllDataAccessPages` collection. You can then use the `Name` property of each individual `AccessObject` object to return the name of a data access page.

You can't add or delete an `AccessObject` object from the `AllDataAccessPages` collection.

The following example prints the name of each open `AccessObject` object in the `AllDataAccessPages` collection.

```vba
Sub AllDataAccessPages()
    Dim obj As AccessObject, dbs As Object
    Set dbs = Application.CurrentProject
    ' Search for open AccessObject objects in
    ' AllDataAccessPages collection.
```
For Each obj In dbs.AllDataAccessPages
    If obj.IsLoaded = True Then
        ' Print name of obj.
        Debug.Print obj.Name
    End If
Next obj
End Sub
AllDatabaseDiagrams Collection

Multiple objects  
- AllDatabaseDiagrams
- AccessObject
- AccessObjectProperties

The AllDatabaseDiagrams collection contains an AccessObject for each database diagram in the CurrentData or CodeData object.
Using the AllDatabaseDiagrams Collection

The CurrentData or CodeData object has an AllDatabaseDiagrams collection containing AccessObject objects that describe instances of all database diagrams specified by CurrentData or CodeData. For example, you can enumerate the AllDatabaseDiagrams collection in Visual Basic to set or return the values of properties of individual AccessObject objects in the collection.

Tip

The For Each...Next statement is useful for enumerating a collection.

You can refer to an individual AccessObject object in the AllDatabaseDiagrams collection either by referring to the object by name, or by referring to its index within the collection. If you want to refer to a specific object in the AllDatabaseDiagrams collection, it's better to refer to the database diagram by name because a database diagram's collection index may change.

The AllDatabaseDiagrams collection is indexed beginning with zero. If you refer to a database diagram by its index, the first database diagram is AllDatabaseDiagrams(0), the second database diagram is AllDatabaseDiagrams(1), and so on.

Notes

- The AllDatabaseDiagrams collection only contains AccessObject objects within a Microsoft Access project (.adp). A Microsoft Access database (.mdb) does not contain any database diagrams.
- To list all open database diagrams in the project, use the IsLoaded property of each AccessObject object in the AllDatabaseDiagrams collection. You can then use the Name property of each individual AccessObject object to return the name of a database diagram.

You can't add or delete an AccessObject object from the AllDatabaseDiagrams collection.

The following example prints the name of each open AccessObject object in the AllDatabaseDiagrams collection.
Sub AllDatabaseDiagrams()
    Dim obj As AccessObject, dbs As Object
    Set dbs = Application.CurrentData
    ' Search for open AccessObject objects in
    ' AllDatabaseDiagrams collection.
    For Each obj In dbs.AllDatabaseDiagrams
        If obj.IsLoaded = True Then
            ' Print name of obj.
            Debug.Print obj.Name
        End If
    Next obj
End Sub
AllForms Collection

Multiple objects AllForms
  AccessObject
  AccessObjectProperties

The AllForms collection contains an AccessObject object for each form in the CurrentProject or CodeProject object.
Using the AllForms Collection

The CurrentProject and CodeProject object has an AllForms collection containing AccessObject objects that describe instances of all the forms in the database. For example, you can enumerate the AllForms collection in Visual Basic to set or return the values of properties of individual AccessObject objects in the collection.

Tip

The For Each...Next statement is useful for enumerating a collection.

You can refer to an individual AccessObject object in the AllForms collection either by referring to the object by name, or by referring to its index within the collection. If you want to refer to a specific object in the AllForms collection, it's better to refer to the form by name because a form's collection index may change.

The AllForms collection is indexed beginning with zero. If you refer to a form by its index, the first form is AllForms(0), the second form is AllForms(1), and so on.

Note To list all open forms in the database, use the IsLoaded property of each AccessObject object in the AllForms collection. You can then use the Name property of each individual AccessObject object to return the name of a form.

You can't add or delete an AccessObject object from the AllForms collection.

The following example prints the name of each open AccessObject object in the AllForms collection.

Sub AllForms()
  Dim obj As AccessObject, dbs As Object
  Set dbs = Application.CurrentProject
  ' Search for open AccessObject objects in AllForms collection.
  For Each obj In dbs.AllForms
    If obj.IsLoaded = True Then
      ' Print name of obj.
      Debug.Print obj.Name
    End If
  Next
End Sub
Next obj
End Sub
AllFunctions Collection

Multiple objects  
  AllFunctions  
  AccessObject  
  AccessObjectProperties

The AllFunctions collection contains an AccessObject object for each function in the CurrentData or CodeData object.
Using the AllFunctions collection

The CurrentData or CodeData object has an AllFunctions collection containing AccessObject objects that describe instances of all functions specified by the CurrentData or CodeData objects. For example, you can enumerate the AllFunctions collection in Visual Basic to set or return the values of properties of individual AccessObject objects in the collection.

You can refer to an individual AccessObject object in the AllFunctions collection either by referring to the object by name, or by referring to its index within the collection. If you want to refer to a specific object in the AllFunctions collection, it's better to refer to the function by name because a function's collection index may change.

The AllFunctions collection is indexed beginning with zero. If you refer to a function by its index, the first function is AllFunctions(0), the second table is AllFunctions(1), and so on.

To list all open functions in the database, use the IsLoaded property of each AccessObject object in the AllFunctions collection. You can then use the Name property of each individual AccessObject object to return the name of a function.

You can't add or delete an AccessObject object from the AllFunctions collection.
AllMacros Collection

Multiple objects \texttt{AllMacros} \texttt{AccessObject} \texttt{AccessObjectProperties}

The \texttt{AllMacros} collection contains an \texttt{AccessObject} for each \texttt{macro} in the \texttt{CurrentProject} or \texttt{CodeProject} object.
Using the AllMacros Collection

The **CurrentProject** or **CodeProject** object has an **AllMacros** collection containing **AccessObject** objects that describe instances of all the macros specified by **CurrentProject** or **CodeProject**. For example, you can enumerate the AllMacros collection in Visual Basic to set or return the values of properties of individual **AccessObject** objects in the collection.

**Tip**

The **For Each...Next** statement is useful for enumerating a collection.

You can refer to an individual **AccessObject** object in the **AllMacros** collection either by referring to the object by name, or by referring to its index within the collection. If you want to refer to a specific object in the **AllMacros** collection, it's better to refer to the macro by name because a macro's collection index may change.

The **AllMacros** collection is indexed beginning with zero. If you refer to a macro by its index, the first macro is AllMacros(0), the second macro is AllMacros(1), and so on.

**Note** To list all open macros in the database, use the **IsLoaded** property of each **AccessObject** object in the **AllMacros** collection. You can then use the **Name** property of each individual **AccessObject** object to return the name of a macro.

You can't add or delete an **AccessObject** object from the **AllMacros** collection.

The following example prints the name of each open **AccessObject** object in the **AllMacros** collection.

```vbad
Sub AllMacros()
    Dim obj As AccessObject, dbs As Object
    Set dbs = Application.CurrentProject
    ' Search for open AccessObject objects in AllMacros collection.
    For Each obj In dbs.AllMacros
        If obj.IsLoaded = True Then
            ' Print name of obj.
            Debug.Print obj.Name
        End If
    End If
End Sub
```
Next obj
End Sub
AllModules Collection

Multiple objects

  AllModules
  AccessObject
  AccessObjectProperties

The AllModules collection contains an AccessObject of each module in the CurrentProject or CodeProject object.
Using the AllModules Collection

The `CurrentProject` or `CodeProject` object has an `AllModules` collection containing `AccessObject` objects that describe instances of all the `Module` objects specified by `CurrentProject` or `CodeProject`. For example, you can enumerate the `AllModules` collection in Visual Basic to set or return the values of properties of individual `AccessObject` objects in the collection.

**Tip**

The **For Each...Next** statement is useful for enumerating a collection.

You can refer to an individual `AccessObject` object in the `AllModules` collection either by referring to the object by name, or by referring to its index within the collection. If you want to refer to a specific object in the `AllModules` collection, it's better to refer to the module by name because a module's collection index may change.

The `AllModules` collection is indexed beginning with zero. If you refer to a module by its index, the first module is `AllModules(0)`, the second module is `AllModules(1)`, and so on.

**Note** To list all open modules in the database, use the `IsLoaded` property of each `AccessObject` object in the `AllModules` collection. You can then use the `Name` property of each individual `AccessObject` object to return the name of a module.

You can't add or delete an `AccessObject` object from the `AllModules` collection.

The following example prints the name of each open `AccessObject` object in the `AllModules` collection.

```vba
Sub AllModules()
    Dim obj As AccessObject, dbs As Object
    Set dbs = Application.CurrentProject
    ' Search for open AccessObject objects in AllModules collection.
    For Each obj In dbs.AllModules
        If obj.IsLoaded = True Then
            ' Print name of obj.
            Debug.Print obj.Name
```
End If
Next obj
End Sub
The `AllQueries` collection contains an `AccessObject` for each query in the `CurrentData` or `CodeData` object.
Using the AllQueries Collection

The CurrentData or CodeData object has an AllQueries collection containing AccessObject objects that describe instances of all queries specified by CurrentData or CodeData. For example, you can enumerate the AllQueries collection in Visual Basic to set or return the values of properties of individual AccessObject objects in the collection.

Tip

The For Each...Next statement is useful for enumerating a collection.

You can refer to an individual AccessObject object in the AllQueries collection either by referring to the object by name, or by referring to its index within the collection. If you want to refer to a specific object in the AllQueries collection, it's better to refer to the query by name because a query's collection index may change.

The AllQueries collection is indexed beginning with zero. If you refer to a query by its index, the first query is AllQueries(0), the second query is AllQueries(1), and so on.

Notes

- The AllQueries collection only contains AccessObject objects within a Microsoft Access database (.mdb). A Microsoft Access project (.adp) does not contain any macros, see the AllViews collection.

- To list all open queries in the database, use the IsLoaded property of each AccessObject object in the AllQueries collection. You can then use the Name property of each individual AccessObject object to return the name of a query.

- You can't add or delete an AccessObject object from the AllQueries collection.

The following example prints the name of each open AccessObject object in the AllQueries collection.
Sub AllQueries()
    Dim obj As AccessObject, dbs As Object
    Set dbs = Application.CurrentData
    ' Search for open AccessObject objects in AllQueries collection.
    For Each obj In dbs.AllQueries
        If obj.IsLoaded = True Then
            ' Print name of obj.
            Debug.Print obj.Name
        End If
    Next obj
End Sub
AllReports Collection

Multiple objects

- AllReports
- AccessObject
- AccessObjectProperties

The AllReports collection contains an AccessObject for each report in the CurrentProject or CodeProject object.
Using the AllReports Collection

The CurrentProject or CodeProject object has an AllReports collection containing AccessObject objects that describe instances of all the reports in the database. For example, you can enumerate the AllReports collection in Visual Basic to set or return the values of properties of individual AccessObject objects in the collection.

Tip

The For Each...Next statement is useful for enumerating a collection.

You can refer to an individual AccessObject object in the AllReports collection either by referring to the item by name, or by referring to its index within the collection. If you want to refer to a specific report in the AllReports collection, it's better to refer to the item by name because the index may change.

The AllReports collection is indexed beginning with zero. If you refer to a report by its index, the first report is AllReports(0), the second report is AllReports(1), and so on.

Note  To list all open reports in the database, use the IsLoaded property of each AccessObject object in the AllReports collection. You can then use the Name property of each individual AccessObject object to return the name of a report.

You can't add or delete an AccessObject object from the AllReports collection.

The following example prints the name of each open AccessObject object in the AllReports collection.

Sub AllReports()
    Dim obj As AccessObject, dbs As Object
    Set dbs = Application.CurrentProject
    ' Search for open AccessObject objects in AllReports collection.
    For Each obj In dbs.AllReports
        If obj.IsLoaded = True Then
            ' Print name of obj.
            Debug.Print obj.Name
        End If
    Next obj
End Sub
AllStoredProcedures Collection

Multiple objects  
  AllStoredProcedures
  AccessObject
  AccessObjectProperties

The AllStoredProcedures collection contains an AccessObject for each stored procedure in the CurrentData or CodeData object.
Using the AllStoredProcedures Collection

The CurrentData or CodeData object has an AllStoredProcedures collection containing AccessObject objects that describe instances of all stored procedures specified by CurrentData or CodeData. For example, you can enumerate the AllStoredProcedures collection in Visual Basic to set or return the values of properties of individual AccessObject objects in the collection.

Tip

The For Each...Next statement is useful for enumerating a collection.

You can refer to an individual AccessObject object in the AllStoredProcedures collection either by referring to the object by name, or by referring to its index within the collection. If you want to refer to a specific object in the AllStoredProcedures collection, it's better to refer to the stored procedures by name because a stored procedure's collection index may change.

The AllStoredProcedures collection is indexed beginning with zero. If you refer to a stored procedure by its index, the first stored procedure is AllStoredProcedures(0), the second stored procedure is AllStoredProcedures(1), and so on.

Notes

- The AllStoredProcedures collection only contains AccessObject objects within a Microsoft Access project (.adp). A Microsoft Access database (.mdb) does not contain any stored procedures, see the AllMacros collection.

- To list all open stored procedures in the project, use the IsLoaded property of each AccessObject object in the AllStoredProcedures collection. You can then use the Name property of each individual AccessObject object to return the name of a stored procedure.

- You can't add or delete an AccessObject object from the AllStoredProcedures collection.
The following example prints the name of each open AccessObject object in the AllProcedures collection.

Sub AllStoredProcedures()
    Dim obj As AccessObject, dbs As Object
    Set dbs = Application.CurrentData
    ' Search for open AccessObject objects in AllStoredProcedures collection.
    For Each obj In dbs.AllStoredProcedures
        If obj.IsLoaded = True Then
            ' Print name of obj.
            Debug.Print obj.Name
        End If
    Next obj
End Sub
Show All
AllTables Collection

Multiple objects
  - AllTables
    - AccessObject
    - AccessObjectProperties

The AllTables collection contains an AccessObject for each table in the CurrentData or CodeData object.
Using the AllTables Collection

The **CurrentData** or **CodeData** object has an **AllTables** collection containing **AccessObject** objects that describe instances of all tables specified by **CurrentData** or **CodeData**. For example, you can enumerate the **AllTables** collection in Visual Basic to set or return the values of properties of individual **AccessObject** objects in the collection.

**Tip**

The **For Each...Next** statement is useful for enumerating a collection.

You can refer to an individual **AccessObject** object in the **AllTables** collection either by referring to the object by name, or by referring to its index within the collection. If you want to refer to a specific object in the **AllTables** collection, it's better to refer to the table by name because a table's collection index may change.

The **AllTables** collection is indexed beginning with zero. If you refer to a table by its index, the first table is AllTables(0), the second table is AllTables(1), and so on.

**Note** To list all open tables in the database, use the **IsLoaded** property of each **AccessObject** object in the **AllTables** collection. You can then use the **Name** property of each individual **AccessObject** object to return the name of a table.

You can't add or delete an **AccessObject** object from the **AllTables** collection.

The following example prints the name of each open **AccessObject** object in the **AllTables** collection.

```vba
Sub AllTables()
    Dim obj As AccessObject, dbs As Object
    Set dbs = Application.CurrentData
    ' Search for open AccessObject objects in AllTables collection.
    For Each obj In dbs.AllTables
        If obj.IsLoaded = True Then
            ' Print name of obj.
            Debug.Print obj.Name
        End If
    Next obj
End Sub
```
Next obj
End Sub
AllViews Collection

Multiple objects ≤ AllViews
   ≤ AccessObject
   ≤ AccessObjectProperties

The AllViews collection contains an AccessObject for each view in the CurrentData or CodeData object.
Using the AllViews Collection

The CurrentData or CodeData object has an AllViews collection containing AccessObject objects that describe instances of all views specified by CurrentData or CodeData. For example, you can enumerate the AllViews collection in Visual Basic to set or return the values of properties of individual AccessObject objects in the collection.

Tip

The For Each...Next statement is useful for enumerating a collection.

You can refer to an individual AccessObject object in the AllViews collection either by referring to the object by name, or by referring to its index within the collection. If you want to refer to a specific object in the AllViews collection, it's better to refer to the view by name because a view's collection index may change.

The AllViews collection is indexed beginning with zero. If you refer to a view by its index, the first view is AllViews(0), the second table is AllViews(1), and so on.

Notes

- The AllViews collection only contains AccessObject objects within a Microsoft Access project (.adp). A Microsoft Access database (.mdb) does not contain any views, see the AllQueries collection.

- To list all open views in the project, use the IsLoaded property of each AccessObject object in the AllViews collection. You can then use the Name property of each individual AccessObject object to return the name of a view.

- You can't add or delete an AccessObject object from the AllViews collection.

The following example prints the name of each open AccessObject object in the AllViews collection.
Sub AllViews()
    Dim obj As AccessObject, dbs As Object
    Set dbs = Application.CurrentData
    ' Search for open AccessObject objects in AllViews collection.
    For Each obj In dbs.AllViews
        If obj.IsLoaded = True Then
            ' Print name of obj.
            Debug.Print obj.Name
        End If
    Next obj
End Sub
Controls Collection

Multiple objects

- **Controls**
- **Control**
- Multiple objects

The **Controls** collection contains all of the **controls** on a form, report, or subform, within another control, or attached to another control. The **Controls** collection is a member of a **Form**, **Report**, and **SubForm** objects.
Using the Controls Collection

You can enumerate individual controls, count them, and set their properties in the Controls collection. For example, you can enumerate the Controls collection of a particular form and set the Height property of each control to a specified value.

Tip

The For Each...Next statement is useful for enumerating a collection.

It is faster to refer to the Controls collection implicitly, as in the following examples, which refer to a control called NewData on a form named OrderForm. Of the following syntax examples, Me!NewData is the fastest way to refer to the control.

Me!NewData       ' Or Forms!OrderForm!NewData.

Me![New Data]     ' Use if control name contains space.

Me("NewData")    ' Performance is slightly slower.

You can also refer to an individual control by referring explicitly to the Controls collection.

Me.Controls!NewData   ' Or Forms!OrderForm.Controls!NewData.

Me.Controls![New Data]

Me.Controls("NewData")

Additionally, you can refer to a control by its index in the collection. The Controls collection is indexed beginning with zero.

Me(0)                ' Refer to first item in collection.
Me.Controls(0)

**Note** You can use the `Me` keyword to represent a form or report within code only if you're referring to the form or report from code within the form module or report module. If you're referring to a form or report from a standard module or a different form's or report's module, you must use the full reference to the form or report.

To work with the controls on a section of a form or report, use the `Section` property to return a reference to a `Section` object. Then refer to the `Controls` collection of the `Section` object.

Two types of `Control` objects, the tab control and option group control, have `Controls` collections that can contain multiple controls. The `Controls` collection belonging to the option group control contains any option button, check box, toggle button, or label controls in the option group.

The tab control contains a `Pages` collection, which is a special type of `Controls` collection. The `Pages` collection contains `Page` objects. `Page` objects are also controls. The `ControlType` property constant for a `Page` control is `acPage`. A `Page` object, in turn, has its own `Controls` collection, which contains all the controls on an individual page.

Other `Control` objects have a `Controls` collection that can contain an attached label. These controls include the text box, option group, option button, toggle button, check box, combo box, list box, command button, bound object frame, and unbound object frame controls.
The **DataAccessPages** collection contains all of the data access pages that are currently open in a **Microsoft Access project** (.adp) or Access database (.mdb).
Using the DataAccessPages Collection

Use the **DataAccessPages** collection in Visual Basic or in an expression to refer to data access pages that are currently open. For example, you can enumerate the **DataAccessPages** collection to set or return the values of properties of individual data access pages in the collection.

**Tip**

The **For Each...Next** statement is useful for enumerating a collection.

You can refer to an individual **DataAccessPage** object in the **DataAccessPages** collection either by referring to the data access page by name, or by referring to its index within the collection. If you want to refer to a specific data access page in the **DataAccessPages** collection, it's better to refer to the data access page by name because a data access page's collection index may change.

The **DataAccessPages** collection is indexed beginning with zero. If you refer to a data access page by its index, the first data access page opened is DataAccessPages(0), the second form opened is DataAccessPages(1), and so on. If you opened Page1 and then opened Page2, Page2 would be referenced in the **DataAccessPages** collection by its index as DataAccessPages(1). If you then closed Page1, Page2 would be referenced in the **DataAccessPages** collection by its index as DataAccessPages(0).

**Note** To list all data access pages in the database, whether open or closed, enumerate the **AllDataAccessPages** collection of the **CurrentProject** object. You can then use the **Name** property of each individual **AccessObject** object to return the name of a data access page.

You can't add or delete a **DataAccessPage** object from the **DataAccessPages** collection.

The following example creates a new data access page and sets certain properties:

```vba
Sub NewDataAccessPage()
    Dim dap As AccessObject
```
' Create new data access page.
Set dap = CreateDataAccessPage("c:\My Documents\Sales Entry", _
    True)
' Set data access page Tag property.
dap.Tag = "Sales Entry Data Access Page"
' Restore data access page.
DoCmd.Restore
End Sub

The next example enumerates the **DataAccessPages** collection and prints the name of each data access page in the **DataAccessPages** collection.

Sub AllOpenDataAccessPages()
    Dim dap As AccessObject
    Set dbs = Application.CurrentProject
    ' Search for open objects in DataAccessPages collection.
    For Each dap In dbs.AllDataAccessPages
        If dap.IsLoaded = TRUE then
            ' Print name of form.
            Debug.Print dap.Name
        End If
    Next dap
End Sub
DependencyObjects Collection

DependencyObjects ▲ AccessObject
    ▲ AccessObjectProperties

Represents the collection of AccessObject objects that a DependencyInfo object contains.
Using the DependencyObjects Collection

To return a **DependencyObjects** collection, use the **Dependants** or **Dependencies** property of the **DependencyInfo** object.

To return a single **AccessObject** object, use the **Item** method of the **DependencyObjects** collection.
The **FormatConditions** collection represents the collection of **conditional formats** for a combo box or text box control. Each format is represented by a **FormatCondition** object.
Using the FormatConditions Collection

Use the **FormatConditions** property of a combo box or text box in Visual Basic or in an **expression** to return a **FormatConditions** collection. Use the **Add** method to create a new conditional format, and use the **Modify** method to change an existing conditional format.

You can use the **Modify** method to change one of the formats, or the **Delete** method to delete a format.

Conditional formatting can also be set on a combo box or text box from the **Conditional Formatting** dialog box. The **Conditional Formatting** dialog box is available by clicking **Conditional Formatting** on the **Format** menu when a form is in **Design view**.
The **Forms** collection contains all of the currently open **forms** in a Microsoft Access **database**.
Using the Forms Collection

Use the **Forms** collection in Visual Basic or in an expression to refer to forms that are currently open. For example, you can enumerate the **Forms** collection to set or return the values of properties of individual forms in the collection.

**Tip**

The **For Each...Next** statement is useful for enumerating a collection.

You can refer to an individual **Form** object in the **Forms** collection either by referring to the form by name, or by referring to its index within the collection. If you want to refer to a specific form in the **Forms** collection, it's better to refer to the form by name because a form's collection index may change.

The **Forms** collection is indexed beginning with zero. If you refer to a form by its index, the first form opened is Forms(0), the second form opened is Forms(1), and so on. If you opened Form1 and then opened Form2, Form2 would be referenced in the **Forms** collection by its index as Forms(1). If you then closed Form1, Form2 would be referenced in the **Forms** collection by its index as Forms(0).

**Note** To list all forms in the database, whether open or closed, enumerate the **AllForms** collection of the **CurrentProject** object. You can then use the **Name** property of each individual **AccessObject** object to return the name of a form.

You can't add or delete a **Form** object from the **Forms** collection.
The Modules collection contains all open standard modules and class modules in a Microsoft Access database.
Using the Modules Collection

All open modules are included in the Modules collection, whether they are uncompiled, are compiled, are in break mode, or contain the code that's running.

To determine whether an individual Module object represents a standard module or a class module, check the Module object's Type property.

The Modules collection belongs to the Microsoft Access Application object.

Individual Module objects in the Modules collection are indexed beginning with zero.

The following example illustrates how to use the Modules collection to loop through the open modules. The example prints the name if each open module in the Immediate window.

Sub PrintOpenModuleName()  
    Dim i As Integer  
    Dim modOpenModules As Modules  
    Set modOpenModules = Application.Modules  
    For i = 0 To modOpenModules.Count - 1  
        Debug.Print modOpenModules(i).Name  
    Next  
End Sub
Pages Collection

Multiple objects

- Pages
  - Page
  - Properties

The Pages collection contains all Page objects in a tab control.
Using the Pages Collection

The Pages collection is a special kind of Controls collection belonging to the tab control. It contains Page objects, which are controls. The Pages collection differs from a typical Controls collection in that you can add and remove Page objects by using methods of the Pages collection.

To add a new Page object to the Pages collection from Visual Basic, use the Add method of the Pages collection. To remove an existing Page object, use the Remove method of the Pages collection. To count the number of Page objects in the Pages collection, use the Count property of the Pages collection.

You can also use the CreateControl method to add a Page object to the Pages collection of a tab control. To do this, you must specify the name of the tab control for the Parent argument of the CreateControl function. The ControlType property constant for a Page object is acPage.

You can enumerate through the Pages collection by using the For Each...Next statement.

Individual Page objects in the Pages collection are indexed beginning with zero.
Printers Collection

The **Printers** collection contains **Printer** objects representing all the printers available on the current system.
Using the Printers collection

Use the Printers property of the Application object to return the Printers collection. You can enumerate through the Printers collection by using the For Each...Next statement. The following example displays information about all the printers available to the system.

Dim prtLoop As Printer

For Each prtLoop In Application.Printers
    With prtLoop
        MsgBox "Device name: " & .DeviceName & vbCrLf & "Driver name: " & .DriverName & vbCrLf & "Port: " & .Port
    End With
Next prtLoop

You can refer to an individual Printer object in the Printers collection either by referring to the printer by name, or by referring to its index within the collection.

The Printers collection is indexed beginning with zero. If you refer to a printer by its index, the first printer is Printers(0), the second printer is Printers(1), and so on.

You can't add or delete a Printer object from the Printers collection.
Properties Collection

Multiple objects

The Properties collection contains all of the built-in properties in an instance of an open form, report, or control. These properties uniquely characterize that instance of the object.
Using the Properties Collection

Use the Properties collection in Visual Basic or in an expression to refer to form, report, or control properties on forms or reports that are currently open.

Tip

The For Each...Next statement is useful for enumerating a collection.

You can use the Properties collection of an object to enumerate the object's built-in properties. You don't need to know beforehand exactly which properties exist or what their characteristics (Name and Value properties) are to manipulate them.

Note In addition to the built-in properties, you can also create and add your own user-defined properties. To add a user-defined property to an existing instance of an object, see the AccessObjectProperties collection and Add method topics.

The following example enumerates the Forms collection and prints the name of each open form in the Forms collection. It then enumerates the Properties collection of each form and prints the name of each property and value.

Sub AllOpenForms()
    Dim frm As Form, prp As Property

    ' Enumerate Forms collection.
    For Each frm In Forms
        ' Print name of form.
        Debug.Print frm.Name
        ' Enumerate Properties.Name collection of each form.
        For Each prp In frm.Properties
            ' Print name of each property.
            Debug.Print prp.Name; " = "; prp.Value
        Next prp
    Next frm
End Sub
References Collection

Multiple objects

- References

- Reference

- References

The References collection contains Reference objects representing each reference that's currently set.
Using the References Collection

The Reference objects in the References collection correspond to the list of references in the References dialog box, available by clicking References on the Tools menu. Each Reference object represents one selected reference in the list. References that appear in the References dialog box but haven't been selected aren't in the References collection.

You can enumerate through the References collection by using the For Each...Next statement.

The References collection belongs to the Microsoft Access Application object.

Individual Reference objects in the References collection are indexed beginning with 1.
The *Reports* collection contains all of the currently open reports in a Microsoft Access database.
Using the Reports Collection

You can use the **Reports** collection in Visual Basic or in an **expression** to refer to reports that are currently open. For example, you can enumerate the **Reports** collection to set or return the values of properties of individual reports in the **collection**.

**Tip**

The **For Each...Next** statement is useful for enumerating a collection.

You can refer to an individual **Report** object in the **Reports** collection either by referring to the report by name, or by referring to its index within the collection.

The **Reports** collection is indexed beginning with zero. If you refer to a report by its index, the first report is Reports(0), the second report is Reports(1), and so on. If you opened Report1 and then opened Report2, Report2 would be referenced in the **Reports** collection by its index as Reports(1). If you then closed Report1, Report2 would be referenced in the **Reports** collection by its index as Reports(0).

**Note**  To list all reports in the database, whether open or closed, enumerate the **AllReports** collection of the **CurrentProject** object. You can then use the **Name** property of each individual **AccessObject** object to return the name of a report.

You can't add or delete a **Report** object from the **Reports** collection.
SmartTagActions Collection

SmartTagActions ![SmartTagAction]

Represents a collection of actions for an individual smart tag or a type of smart tag.

Smart tag actions are processes that are programmed into smart tags; they allow users to perform certain functions related to the smart tag. For example, one action for a smart tag might be to access a Web site, while another action inserts contact information from Microsoft Outlook, while yet another action displays a map and driving directions.
Using the SmartTagActions Collection

To return the collection of actions related to a smart tag, use the `SmartTagActions` property of the `SmartTag` object.
SmartTagProperties Collection

SmartTagProperties SmartTagProperty

A collection of SmartTagProperty objects that represents the properties related to a smart tag.
Using the SmartTagProperties Collection

To return the SmartTagProperties collection for a smart tag, use the Properties property of the SmartTag object. To create a custom property from within a Visual Basic for Applications (VBA) project, use the Add method of the SmartTagProperties collection.
SmartTags Collection

SmartTags SmartTag

Represents the collection of smart tags for a control on a form, report, or data access page.
Using the SmartTags Collection

To return a single SmartTag object, use the Item method or use SmartTags(Index), where Index represents the number of the smart tag.

**Note** Unlike the SmartTags collections in Microsoft Excel and Microsoft Word, the SmartTags collection in Microsoft Access is zero-based. Therefore, the code control.SmartTags(0) returns the first smart tag for the specified control.
AccessObject Object

Multiple objects  
  AccessObject  
    AccessObjectProperties

An AccessObject object refers to a particular Microsoft Access object within the following collections.

AllDataAccessPages
AllDatabaseDiagrams
AllForms
AllFunctions
AllMacros
AllModules
AllQueries
AllReports
AllStoredProcedures
AllTables
AllViews
## Using the AccessObject Object

An **AccessObject** object includes information about one instance of an object. The following table lists the types of objects each **AccessObject** describes, the name of its collection, and what type of information **AccessObject** contains.

<table>
<thead>
<tr>
<th>AccessObject</th>
<th>Collection</th>
<th>Contains information about</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data access page</td>
<td>AllDataAccessPages</td>
<td>Saved data access pages</td>
</tr>
<tr>
<td>Database diagram</td>
<td>AllDatabaseDiagrams</td>
<td>Saved database diagrams</td>
</tr>
<tr>
<td>Form</td>
<td>AllForms</td>
<td>Saved forms</td>
</tr>
<tr>
<td>Function</td>
<td>AllFunctions</td>
<td>Saved functions</td>
</tr>
<tr>
<td>Macro</td>
<td>AllMacros</td>
<td>Saved macros</td>
</tr>
<tr>
<td>Module</td>
<td>AllModules</td>
<td>Saved modules</td>
</tr>
<tr>
<td>Query</td>
<td>AllQueries</td>
<td>Saved queries</td>
</tr>
<tr>
<td>Report</td>
<td>AllReports</td>
<td>Saved reports</td>
</tr>
<tr>
<td>Stored procedure</td>
<td>AllStoredProcedures</td>
<td>Saved stored procedures</td>
</tr>
<tr>
<td>Table</td>
<td>AllTables</td>
<td>Saved tables</td>
</tr>
<tr>
<td>View</td>
<td>AllViews</td>
<td>Saved views</td>
</tr>
</tbody>
</table>

Because an **AccessObject** object corresponds to an existing object, you can't create new **AccessObject** objects or delete existing ones. To refer to an **AccessObject** object in a collection by its ordinal number or by its **Name** property setting, use any of the following syntax forms:

- `AllForms(0)`
- `AllForms("name")`
- `AllForms![name]`
AccessObjectProperty Object

Multiple objects  AccessObjectProperties
    AccessObjectProperty

An AccessObjectProperty object represents a built-in or user-defined characteristic of an AccessObject object.
Using the AccessObjectProperty Object

Every AccessObject object contains an AccessObjectProperties collection that has AccessObjectProperty objects corresponding to the properties of that AccessObject object. The user can also define AccessObjectProperty objects and append them to the AccessObjectProperties collection of some AccessObject objects.

You can create user-defined properties for the following objects:

- **CodeData, CodeProject, CurrentProject, and CurrentData objects**
- **AccessObject objects** in the following collections:
  - CurrentProject and CodeProject object collections
  - CodeData and CodeProject object collections

  - AllForms
  - AllReports
  - AllDataAccessPages
  - AllMacros
  - AllModules
  - AllTables
  - AllQueries
  - AllViews
  - AllStoredProcedures
  - AllDatabaseDiagrams

**Note** The AccessObjectProperties collection isn't accessible for objects derived from the CurrentData object (for example, CurrentData.AllTables!Table1). For objects derived in this manner, you can only access their built-in properties by direct calls to the desired property (for example, CurrentData.AllTables!Table1.Name).

To add a user-defined property, use the Add method to create and add an AccessObjectProperty object with a unique Name property setting and Value property of the new AccessObjectProperty object to the AccessObjectProperties collection of the appropriate object. The object to which you are adding the user-defined property must already be appended to a collection. Referencing a user-defined AccessObjectProperty object that has not yet been appended to an AccessObjectProperties collection will cause an error, as will appending a user-defined AccessObjectProperty object to an AccessObjectProperties collection containing an AccessObjectProperty object.
of the same name.

You can delete user-defined properties from the `AccessObjectProperties` collection.

**Note** A user-defined `AccessObjectProperty` object is associated only with the specific instance of an object. The property isn't defined for all instances of objects of the selected type.

The `AccessObjectProperty` object has two built-in properties:

- The *Name* property, a `String` that uniquely identifies the property.

- The *Value* property, a `Variant` that contains the property setting.

To refer to a built-in or user-defined `AccessObjectProperty` object in a collection by its ordinal number or by its *Name* property setting, use any of the following syntax forms:

```vba
CurrentProject.AllForms("Form1").Properties(0)
CurrentProject.AllForms("Form1").Properties("name")
CurrentProject.AllForms("Form1").Properties![name]
```

With the same syntax forms, you can also refer to the *Value* property of a `AccessObjectProperty` object. The context of the reference will determine whether you are referring to the `AccessObjectProperty` object itself or the *Value* property of the `AccessObjectProperty` object.

**Note** Properties in the `AccessObjectProperties` collection are not stored and can be lost when when the object they are associated with is checked in or out using the Source Code Control add-in.
AdditionalData Object

AdditionalData

Represents the collection of tables and queries that will be included with the parent table that is exported by the ExportXML method.
Using the AdditionalData Object

To create an AdditionalData object, use the CreateAdditionalData method of the Application object.

To add a table to an existing AdditionalData object, use the Add method.

The following example exports the contents of the Customers table in the Northwind Traders sample database, along with the contents of the Orders and Orders Details tables, to an XML data file named Customer Orders.xml.

Sub ExportCustomerOrderData()
    Dim objOrderInfo As AdditionalData
    Set objOrderInfo = Application.CreateAdditionalData

    ' Add the Orders and Order Details tables to the data to be expo
    objOrderInfo.Add "Orders"
    objOrderInfo.Add "Order Details"

    ' Export the contents of the Customers table. The Orders and Ord
    ' Details tables will be included in the XML file.
    Application.ExportXML ObjectType:=acExportTable, DataSource:="Cu
    DataTarget:="Customer Orders.xml", _
    AdditionalData:=objOrderInfo
End Sub
Show All
Application Object

The **Application** object refers to the active Microsoft Access application.
Using the Application Object

The **Application** object contains all **Microsoft Access objects** and **collections**.

You can use the **Application** object to apply methods or property settings to the entire Microsoft Access application. For example, you can use the **SetOption** method of the **Application** object to set database options from Visual Basic. The following example shows how you can set the **Status Bar** check box under **Show** on the **View** tab of the **Options** dialog box.

```vba
Application.SetOption "Show Status Bar", True
```

Microsoft Access is a COM component that supports **Automation**, formerly called OLE Automation. You can manipulate Microsoft Access objects from another application that also supports Automation. To do this, you use the **Application** object.

For example, Microsoft Visual Basic is a COM component. You can open a Microsoft Access database from Visual Basic and work with its objects. From Visual Basic, first create a reference to the Microsoft Office Access 2003 object library. Then create a new **instance** of the **Application class** and point an object variable to it, as in the following example:

```vba
Dim appAccess As New Access.Application
```

From applications that don't support the **New** keyword, you can create a new instance of the **Application** class by using the **CreateObject** function:

```vba
Dim appAccess As Object
Set appAccess = CreateObject("Access.Application")
```

Once you've created a new instance of the **Application** class, you can open a database or create a new database, by using either the **OpenCurrentDatabase** method or the **NewCurrentDatabase** method. You can then set the properties of the **Application** object and call its methods. When you return a reference to the **CommandBars** object by using the **CommandBars** property of the **Application** object, you can access all Office 2003 **command bar** objects and
collections by using this reference.

You can also manipulate other Microsoft Access objects through the Application object. For example, by using the OpenForm method of the Microsoft Access DoCmd object, you can open a Microsoft Access form from Microsoft Excel:

```vba
appAccess.DoCmd.OpenForm "Orders"
```

For more information on creating a reference and controlling objects by using Automation, see the documentation for the application that's acting as the COM component.
AutoCorrect Object

AutoCorrect

The AutoCorrect object represents the AutoCorrect functionality of TextBox and ComboBox objects.
Using the AutoCorrect Object

To return an AutoCorrect object, use the AutoCorrect property.

To enable or disable the display of the AutoCorrect Options button, use the DisplayAutoCorrectOptions property.

To enable or disable AutoCorrect functionality for a particular TextBox or ComboBox object, use the AllowAutoCorrect property.
BoundObjectFrame Object

**BoundObjectFrame**  |  **Properties**

A bound object frame object displays a picture, chart, or any **OLE object** stored in a table in a Microsoft Access database. For example, if you store pictures of your employees in a table in Microsoft Access, you can use a bound object frame to display these pictures on a form or report.
Using the BoundObjectFrame Object

This object type allows you to create or edit the object from within the form or report by using the OLE server.

A bound object frame is bound to a field in an underlying table.

The field in the underlying table to which the bound object frame is bound must be of the OLE Object data type.

The object in a bound object frame is different for each record. The bound object frame can display linked or embedded objects. If you want to display objects not stored in an underlying table, use an unbound object frame or an image control.
**CheckBox Object**

**CheckBox**  

This object corresponds to a check box on a form or report. This check box is a stand-alone control that displays a Yes/No value from an underlying record source.

**Control:** Address Change  
**Tool:**

![CheckBox](Image)

**Note** This control should not be confused with the Dynamic HTML check box control used on a data access page. For information about a check box control on a data access page, see [Check Box Control (Data Access Pages)](Check Box Control (Data Access Pages)).

When you select or clear a check box that's bound to a Yes/No field, Microsoft Access displays the value in the underlying table according to the field's **Format** property (Yes/No, **True/False**, or On/Off).

You can also use check boxes in an **option group** to display values to choose from.

It's also possible to use an **unbound** check box in a **custom dialog box** to accept user input.
The CodeData object refers to objects stored within the code database by the source (server) application (Jet or SQL).
**Remarks**

The **CodeData** object has several **collections** that contain specific object types within the code database. The following table lists the name of each collection defined by the database and the types of objects it contains.

<table>
<thead>
<tr>
<th>Collections</th>
<th>Object type</th>
</tr>
</thead>
<tbody>
<tr>
<td>AllTables</td>
<td>All tables</td>
</tr>
<tr>
<td>AllFunctions</td>
<td>All functions</td>
</tr>
<tr>
<td>AllQueries</td>
<td>All queries (the count of queries in a Microsoft Access project (.adp) will be zero).</td>
</tr>
<tr>
<td>AllViews</td>
<td>All views (the count of views in an Access database (.mdb) database will be zero).</td>
</tr>
<tr>
<td>AllStoredProcedures</td>
<td>All stored procedures (the count of stored procedures in a .mdb database will be zero).</td>
</tr>
<tr>
<td>AllDatabaseDiagrams</td>
<td>All database diagrams (the count of database diagrams in a .mdb database will be zero).</td>
</tr>
</tbody>
</table>

**Note**  The collections in the preceding table contain all of the respective objects in the database regardless if they are opened or closed.

For example, an **AccessObject** representing a table is a member of the **AllTables** collection, which is a collection of **AccessObject** objects within the current database. Within the **AllTables** collection, individual tables are indexed beginning with zero. You can refer to an individual **AccessObject** object in the **AllTables** collection either by referring to the table by name, or by referring to its index within the collection. If you want to refer to a specific item in the **AllTables** collection, it's better to refer to it by name because the item's index may change. If the object name includes a space, the name must be surrounded by brackets ([ ]).

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>AllTables!<code>tablename</code></td>
<td>AllTables!OrderTable</td>
</tr>
<tr>
<td>AllTables![<code>table name</code>]</td>
<td>AllTables![Order Table]</td>
</tr>
<tr>
<td>AllTables(&quot;tablename&quot;)</td>
<td>AllTables(&quot;OrderTable&quot;)</td>
</tr>
<tr>
<td>AllTables(index)</td>
<td>AllTables(0)</td>
</tr>
</tbody>
</table>
The **CodeProject** object refers to the **project** for the code database of a Microsoft **Access project** (.adp) or **Access database** (.mdb).
Using the CodeProject Object

The **CodeProject** object has several **collections** that contain specific **AccessObject** objects within the code database. The following table lists the name of each collection defined by Access project and the types of objects it contains.

<table>
<thead>
<tr>
<th>Collections</th>
<th>Object type</th>
</tr>
</thead>
<tbody>
<tr>
<td>AllForms</td>
<td>All forms</td>
</tr>
<tr>
<td>AllReports</td>
<td>All reports</td>
</tr>
<tr>
<td>AllMacros</td>
<td>All macros</td>
</tr>
<tr>
<td>AllModules</td>
<td>All modules</td>
</tr>
<tr>
<td>AllDataAccessPages</td>
<td>All data access pages</td>
</tr>
</tbody>
</table>

**Note**  The collections in the preceding table contain all of the respective objects in the database regardless if they are opened or closed.

For example, an **AccessObject** object representing a form is a member of the **AllForms** collection, which is a collection of **AccessObject** objects within the current database. Within the **AllForms** collection, individual members of the collection are indexed beginning with zero. You can refer to an individual **AccessObject** object in the **AllForms** collection either by referring to the form by name, or by referring to its index within the collection. If you want to refer to a specific object in the **AllForms** collection, it's better to refer to it by name because a item's collection index may change. If the object name includes a space, the name must be surrounded by brackets ([ ]).

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>AllForms!formname</td>
<td>AllForms!OrderForm</td>
</tr>
<tr>
<td>AllForms![form name]</td>
<td>AllForms![Order Form]</td>
</tr>
<tr>
<td>AllForms(&quot;formname&quot;)</td>
<td>AllForms(&quot;OrderForm&quot;)</td>
</tr>
<tr>
<td>AllForms(index)</td>
<td>AllForms(0)</td>
</tr>
</tbody>
</table>
ComboBox Object

This object corresponds to a combo box control. The combo box control combines the features of a text box and a list box. Use a combo box when you want the option of either typing a value or selecting a value from a predefined list.

Control: Tool:

Note This control should not be confused with the Dynamic HTML drop-down list box control used on a data access page. For information about a drop-down list box control on a data access page, see Drop-down List Box Control (Data Access Page).

In Form view, Microsoft Access doesn't display the list until you click the combo box's arrow.

If you have Control Wizards on before you select the combo box tool, you can create a combo box with a wizard. To turn Control Wizards on or off, click the Control Wizards tool in the toolbox.

The setting of the LimitToList property determines whether you can enter values that aren't in the list.

The list can be single- or multiple-column, and the columns can appear with or without headings.
CommandButton Object

This object corresponds to a command button. A command button on a form can start an action or a set of actions. For example, you could create a command button that opens another form. To make a command button do something, you write a macro or event procedure and attach it to the button's OnClick property.

Control: Tool:

Note This control should not be confused with the Dynamic HTML command button control used on a data access page. For information about a command button control on a data access page, see Command Button Control (Data Access)

You can display text on a command button by setting its Caption property, or you can display a picture by setting its Picture property.

Tip

You can create over 30 different types of command buttons with the Command Button Wizard. When you use the Command Button Wizard, Microsoft Access creates the button and the event procedure for you.
Control Object

Multiple objects \textit{Control}

The \textbf{Control} object represents a \textit{control} on a \textit{form}, \textit{report}, or section, within another control, or attached to another control.
**Using the Control Object**

All controls on a form or report belong to the **Controls** collection for that **Form** or **Report** object. Controls within a particular section belong to the **Controls** collection for that section. Controls within a tab control or option group control belong to the **Controls** collection for that control. A label control that is attached to another control belongs to the **Controls** collection for that control.

When you refer to an individual **Control** object in the **Controls** collection, you can refer to the **Controls** collection either implicitly or explicitly.

```vba
' Implicitly refer to NewData control in Controls collection.
Me!NewData

' Use if control name contains space.
Me![New Data]

' Performance slightly slower.
Me("NewData")

' Refer to a control by its index in the controls collection.
Me(0)

' Refer to a NewData control by using the subform Controls collection.
Me.ctlSubForm.Controls!NewData

' Explicitly refer to the NewData control in the Controls collection.
Me.Controls!NewData

Me.Controls("NewData")

Me.Controls(0)
```
**Note** You can use the **Me** keyword to represent a **Form** or **Report** object within code only if you're referring to the form or report from code within the **class module**. If you're referring to a form or report from a **standard module** or a different form's or report's module, you must use the full reference to the form or report.

Each **Control** object is denoted by a particular **intrinsic constant**. For example, the intrinsic constant **acTextBox** is associated with a **textBox** control, and **acCommandButton** is associated with a **command button**. The constants for the various Microsoft Access controls are set forth in the control's **ControlType** property.

To determine the type of an existing control, you can use the **ControlType** property. However, you don't need to know the specific type of a control in order to use it in code. You can simply represent it with a variable of data type **Control**.

If you do know the data type of the control to which you are referring, and the control is a built-in Microsoft Access control, you should represent it with a variable of a specific type. For example, if you know that a particular control is a text box, declare a variable of type **TextBox** to represent it, as shown in the following code.

```vba
Dim txt As TextBox
Set txt = Forms!Employees!LastName
```

**Note** If a control is an **ActiveX control**, then you must declare a variable of type **Control** to represent it; you cannot use a specific type. If you're not certain what type of control a variable will point to, declare the variable as type **Control**.

The option group control can contain other controls within its **Controls** collection, including **option button**, **check box**, **toggle button**, and label controls.

The tab control contains a **Pages** collection, which is a special type of **Controls** collection. The **Pages** collection contains **Page** objects, which are controls. Each **Page** object in turn contains a **Controls** collection, which contains all of the controls on that page.

Other **Control** objects have a **Controls** collection that can contain an attached
label. These controls include the text box, option group, option button, toggle button, check box, combo box, list box, command button, bound object frame, and unbound object frame controls.
CurrentData Object

The **CurrentData** object refers to the objects stored in the current database by the source (server) application (Jet or SQL).
Using the CurrentData Object

The CurrentData object has several collections that contain specific AccessObject objects within the current database. The following table lists the name of each collection defined by the database and the types of objects it contains.

<table>
<thead>
<tr>
<th>Collections</th>
<th>Object type</th>
</tr>
</thead>
<tbody>
<tr>
<td>AllTables</td>
<td>All tables</td>
</tr>
<tr>
<td>AllFunctions</td>
<td>All functions</td>
</tr>
<tr>
<td>AllQueries</td>
<td>All queries (the count of queries in a Microsoft Access project (.adp) will be zero).</td>
</tr>
<tr>
<td>AllViews</td>
<td>All views (the count of views in an Access database (.mdb) database will be zero).</td>
</tr>
<tr>
<td>AllStoredProcedures</td>
<td>All stored procedures (the count of stored procedures in a .mdb database will be zero).</td>
</tr>
<tr>
<td>AllDatabaseDiagrams</td>
<td>All database diagrams (the count of database diagrams in a .mdb database will be zero).</td>
</tr>
</tbody>
</table>

Note The collections in the preceding table contain all of the respective objects in the database regardless if they are opened or closed.

For example, an AccessObject representing a table is a member of the AllTables collection, which is a collection of AccessObject objects within the current database. Within the AllTables collection, individual tables are indexed beginning with zero. You can refer to an individual AccessObject object in the AllTables collection either by referring to the table by name, or by referring to its index within the collection. If you want to refer to a specific item in the AllTables collection, it's better to refer to it by name because the item's index may change. If the object name includes a space, the name must be surrounded by brackets ([ ])..

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>AllTables!tablename</td>
<td>AllTables!OrderTable</td>
</tr>
<tr>
<td>AllTables![table name]</td>
<td>AllTables![Order Table]</td>
</tr>
<tr>
<td><strong>AllTables</strong>(&quot;tablename&quot;)</td>
<td>AllTables(&quot;OrderTable&quot;)</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-------------------------</td>
</tr>
<tr>
<td><strong>AllTables</strong>(<em>index</em>)</td>
<td>AllTables(5)</td>
</tr>
</tbody>
</table>
CurrentProject Object

The **CurrentProject** object refers to the project for the current Microsoft **Access project** (.adp) or **Access database** (.mdb).
Using the CurrentProject Object

The **CurrentProject** object has several **collections** that contain specific **AccessObject** objects within the current database. The following table lists the name of each collection and the types of objects it contains.

<table>
<thead>
<tr>
<th>Collections</th>
<th>Object type</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AllForms</strong></td>
<td>All forms</td>
</tr>
<tr>
<td><strong>AllReports</strong></td>
<td>All reports</td>
</tr>
<tr>
<td><strong>AllMacros</strong></td>
<td>All macros</td>
</tr>
<tr>
<td><strong>AllModules</strong></td>
<td>All modules</td>
</tr>
<tr>
<td><strong>AllDataAccessPages</strong></td>
<td>All data access pages</td>
</tr>
</tbody>
</table>

**Note**  The collections in the preceding table contain all of the respective objects in the database regardless if they are opened or closed.

For example, an **AccessObject** object representing a form is a member of the **AllForms** collection, which is a collection of **AccessObject** objects within the current database. Within the **AllForms** collection, individual members of the collection are indexed beginning with zero. You can refer to an individual **AccessObject** object in the **AllForms** collection either by referring to the form by name, or by referring to its index within the collection. If you want to refer to a specific object in the **AllForms** collection, it's better to refer to it by name because a item's collection index may change. If the object name includes a space, the name must be surrounded by brackets ([ ]).

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AllForms</strong>/formname</td>
<td>AllForms!OrderForm</td>
</tr>
<tr>
<td><strong>AllForms</strong>/[form name]</td>
<td>AllForms![Order Form]</td>
</tr>
<tr>
<td><strong>AllForms</strong>(&quot;formname&quot;)</td>
<td>AllForms(&quot;OrderForm&quot;)</td>
</tr>
<tr>
<td><strong>AllForms</strong>(index)</td>
<td>AllForms(0)</td>
</tr>
</tbody>
</table>

The following example prints some current property settings of the **CurrentProject** object and then sets an option to display hidden objects within the application:
Sub ApplicationInformation()
    ' Print name and type of current object.
    Debug.Print Application.CurrentProject.FullName
    Debug.Print Application.CurrentProject.ProjectType
    ' Set Hidden Objects option under Show on View Tab
    ' of the Options dialog box.
    Application.SetOption "Show Hidden Objects", True
End Sub

The next example shows how to use the CurrentProject object using Automation from another Microsoft Office application. First, from the other application, create a reference to Microsoft Access by clicking References on the Tools menu in the Module window. Select the check box next to Microsoft Access Object Library. Then enter the following code in a Visual Basic module within that application and call the GetAccessData procedure.

The example passes a database name and report name to a procedure that creates a new instance of the Application class, opens the database, and verifies that the specified report exists using the CurrentProject object and AllReports collection.

Sub GetAccessData()
    ' Declare object variable in declarations section of a module
    Dim appAccess As Access.Application
    Dim strDB As String
    Dim strReportName As String

    strDB = "C:\Program Files\Microsoft " & "Office\Office11\Samples\Northwind.mdb"
    strReportName = InputBox("Enter name of report to be verified", "Report Verification")
    VerifyAccessReport strDB, strReportName
End Sub

Sub VerifyAccessReport(strDB As String, _
    strReportName As String)
    ' Return reference to Microsoft Access
    ' Application object.
    Set appAccess = New Access.Application
    ' Open database in Microsoft Access.
    appAccess.OpenCurrentDatabase strDB
    ' Verify report exists.
    On Error Goto ErrorHandler
    appAccess.CurrentProject.AllReports(strReportName)
    MsgBox "Report " & strReportName & _
" verified within Northwind database."
appAccess.CloseCurrentDatabase
Set appAccess = Nothing
Exit Sub
ErrorHandler:
    MsgBox "Report " & strReportName & 
    " does not exist within Northwind database."
appAccess.CloseCurrentDatabase
Set appAccess = Nothing
End Sub
CustomControl Object

CustomControl Properties

Some of the content in this topic may not be applicable to some languages.

When setting the properties of an ActiveX control, you may need or prefer to use the control's custom properties dialog box. This custom properties dialog box provides an alternative to the list of properties in the Microsoft Access property sheet for setting ActiveX control properties in Design view.
Using the CustomControl Object

Note  This information only applies to ActiveX controls in a Microsoft Access database (.mdb) environment.

Two Ways to Set Properties

The reason for the custom properties dialog box is that not all applications that use ActiveX controls provide a property sheet like the one in Microsoft Access. The custom properties dialog box provides an interface for setting key control properties regardless of the interface supplied by the hosting application.

For some ActiveX control properties, you can choose either of these two locations to set the property:

- The Microsoft Access property sheet.

- The ActiveX control's custom properties dialog box.

In some cases, the custom properties dialog box is the only way to set a property in Design view. This is usually the situation when the interface needed to set a property doesn't work inside the Microsoft Access property sheet. For example, the GridFont property for the Calendar control has a number of arguments; you can't set more than one argument per property in the Microsoft Access property sheet.

Finding the Custom Properties Dialog Box

Not all ActiveX controls provide a custom properties dialog box. To see whether a control provides this custom properties dialog box, look for the Custom property in the Microsoft Access property sheet for this control. If the list of properties contains the name Custom, then the control provides the custom properties dialog box.

Using the Custom Properties Dialog Box

After you click the Custom property box in the Microsoft Access property sheet, click the Build button ▶️ to the right of the property box to display the control's
custom properties dialog box, often presented as a tabbed dialog box. Choose the tab that contains the interface for setting the properties that you want to set.

After you make changes on one tab, you can often apply those changes immediately by clicking the **Apply** button (if provided). You can click other tabs to set other properties as needed. To approve all changes made in the custom properties dialog box, click the **OK** button. To return to the Microsoft Access property sheet without changing any property settings, click the **Cancel** button.

You can also view the custom properties dialog box by clicking the **Properties** subcommand of the ActiveX control **Object** command (for example, **Calendar Control Object**) on the **Edit** menu, or by clicking this same subcommand on the **shortcut menu** for the ActiveX control. In addition, some properties in the Microsoft Access property sheet for the ActiveX control, like the **GridFontColor** property of the Calendar control, have a **Build** button to the right of the property box. When you click the **Build** button, the custom properties dialog box is displayed, with the appropriate tab selected (for example, **Colors**).
DataAccessPage Object

Multiple objects - AccessPage
- WebOptions

A **DataAccessPage** object refers to a particular Microsoft Access [data access page](#).
Using the **DataAccessPage** Object

A **DataAccessPage** object is a member of the **DataAccessPages** collection, which is a collection of all currently open data access pages. Within the **DataAccessPages** collection, individual data access pages are indexed beginning with zero. You can refer to an individual **DataAccessPage** object in the **DataAccessPages** collection either by referring to the data access page by name, or by referring to its index within the collection. If you want to refer to a specific data access page in the **DataAccessPages** collection, it's better to refer to the data access page by name because a data access page's collection index may change. If the data access name includes a space, the name must be surrounded by brackets ([ ]).

### Syntax

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DataAccessPages!pagename</strong></td>
<td>DataAccessPages!SalePage</td>
</tr>
<tr>
<td><strong>DataAccessPages![page name]</strong></td>
<td>DataAccessPages![Sale Page]</td>
</tr>
<tr>
<td><strong>DataAccessPages(&quot;pagename&quot;)</strong></td>
<td>DataAccessPages(&quot;Sale Page&quot;)</td>
</tr>
<tr>
<td><strong>DataAccessPages(index)</strong></td>
<td>DataAccessPages(0)</td>
</tr>
</tbody>
</table>

The following example creates a new data access page and sets certain properties:

```vba
Sub NewDataAccessPage()
    Dim dap As AccessObject
    ' Create new data access page.
    Set dap = CreateDataAccessPage("c:\My Documents\Sales Entry", _
        True)
    ' Set data access page Tag property.
    dap.Tag = "Sales Entry Data Access Page"
    ' Restore data access page.
    DoCmd.Restore
End Sub
```

The next example enumerates the **DataAccessPages** collection and prints the name of each data access page in the **DataAccessPages** collection.

```vba
Sub AllOpenDataAccessPages()
    Dim dap As AccessObject
```
Set dbs = Application.CurrentProject
' Search for open objects in DataAccessPages collection.
For Each dap In dbs.AllDataAccessPages
    If dap.IsLoaded = TRUE then
        ' Print name of form.
        Debug.Print dap.Name
    End If
Next dap
End Sub
DefaultWebOptions Object

The **DefaultWebOptions** object contains global application-level attributes used by Microsoft Access when you save a data access page as a Web page or open a Web page. You can return or set attributes either at the application (global) level or at the data access page level.
Using the DefaultWebOptions Object

Use the `DefaultWebOptions` property to return the `DefaultWebOptions` object.

Note that attribute values can be different from one data access page to another, depending on the attribute value at the time the data access page was saved. Data access page-level attributes override application-level attributes. Data access page attributes are contained in the `WebOptions` object.

The following example checks to see whether Microsoft Office Web components are downloaded when a saved data access page is displayed and sets the download flag accordingly.

```vba
Set objAppWebOptions = Application.DefaultWebOptions
With objAppWebOptions
    If .DownloadComponents = True Then
        strCompDownload = "Loaded"
    Else
        strCompDownload = "Not Loaded"
    End If
End If
```

End With
DependencyInfo Object

DependencyInfo

Represents the dependency information for an AccessObject object.
Using the DependencyInfo Object

To return the dependency information for an AccessObject object, use the GetDependencyInfo method of the AccessObject object.
DoCmd Object

You can use the methods of the DoCmd object to run Microsoft Access actions from Visual Basic. An action performs tasks such as closing windows, opening forms, and setting the value of controls.
Using the DoCmd Object

For example, you can use the OpenForm method of the DoCmd object to open a form, or use the Hourglass method to change the mouse pointer to an hourglass icon.

Most of the methods of the DoCmd object have arguments— some are required, while others are optional. If you omit optional arguments, the arguments assume the default values for the particular method. For example, the OpenForm method uses seven arguments, but only the first argument, FormName, is required. The following example shows how you can open the Employees form in the current database. Only employees with the title Sales Representative are included.

DoCmd.OpenForm "Employees", , ,"[Title] = 'Sales Representative'"

The DoCmd object doesn't support methods corresponding to the following actions:

- AddMenu.
- MsgBox. Use the MsgBox function.
- RunApp. Use the Shell function to run another application.
- RunCode. Run the function directly in Visual Basic.
- SendKeys. Use the SendKeys statement.
- SetValue. Set the value directly in Visual Basic.
- StopAllMacros.
- StopMacro.

For more information on the Microsoft Access action corresponding to a DoCmd method, search the Help index for the name of the action.

The following example opens a form in Form view and moves to a new record.

Sub ShowNewRecord()
    DoCmd.OpenForm "Employees", acNormal
    DoCmd.GoToRecord , , acNewRec
End Sub
Form Object

A **Form** object refers to a particular Microsoft Access **form**.
Using the Form Object

A **Form** object is a member of the **Forms** collection, which is a collection of all currently open forms. Within the **Forms** collection, individual forms are indexed beginning with zero. You can refer to an individual *Form* object in the **Forms** collection either by referring to the form by name, or by referring to its index within the **collection**. If you want to refer to a specific form in the **Forms** collection, it's better to refer to the form by name because a form's collection index may change. If the form name includes a space, the name must be surrounded by brackets ([ ]).

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Forms!formname</strong></td>
<td>Forms!OrderForm</td>
</tr>
<tr>
<td><strong>Forms![form name]</strong></td>
<td>Forms![Order Form]</td>
</tr>
<tr>
<td><strong>Forms(&quot;formname&quot;)</strong></td>
<td>Forms(&quot;OrderForm&quot;)</td>
</tr>
<tr>
<td><strong>Forms(index)</strong></td>
<td>Forms(0)</td>
</tr>
</tbody>
</table>

Each **Form** object has a **Controls** collection, which contains all **controls** on the form. You can refer to a control on a form either by implicitly or explicitly referring to the **Controls** collection. Your code will be faster if you refer to the **Controls** collection implicitly. The following examples show two of the ways you might refer to a control named NewData on the form called OrderForm:

```
' Implicit reference.
Forms!OrderForm!NewData

' Explicit reference.
Forms!OrderForm.Controls!NewData
```

The next two examples show how you might refer to a control named NewData on a subform **ctlSubForm** contained in the form called OrderForm:

```
Forms!OrderForm.ctlSubForm.Form!Controls.NewData

Forms!OrderForm.ctlSubForm!NewData
```
FormatCondition Object

The **FormatCondition** object represents a [conditional format](#) of a combo box or text box control and is a member of the **FormatConditions** collection.
Using the FormatCondition Object

You can use the `FormatConditions(index)`, where `index` is the index number of the conditional format, to return a `FormatCondition` object.

Use the `Add` method to create a new conditional format. You can use the `Modify` method to change one of the formats, or the `Delete` method to delete a format.

Conditional formatting can also be set on a combo box or text box from the `Conditional Formatting` dialog box. The `Conditional Formatting` dialog box is available by clicking `Conditional Formatting` on the `Format` menu when a form is in Design view.

Use the `BackColor`, `Enabled`, `FontBold`, `FontItalic`, `FontUnderline`, and `ForeColor` properties of the `FormatCondition` object to control the appearance of formatted combo box and text box controls.
GroupLevel Object

You can use the `GroupLevel` property in Visual Basic to refer to the group level you are grouping or sorting on in a report.
Using the GroupLevel Object

The GroupLevel property setting is an array in which each entry identifies a group level. To refer to a group level, use this syntax:

**GroupLevel(n)**

The number \( n \) is the group level, starting with 0. The first field or expression you group on is group level 0, the second is group level 1, and so on. You can have up to 10 group levels (0 to 9).

The following sample settings show how you use the GroupLevel property to refer to a group level.

<table>
<thead>
<tr>
<th>Group level</th>
<th>Refers to</th>
</tr>
</thead>
<tbody>
<tr>
<td>GroupLevel(0)</td>
<td>The first field or expression you sort or group on.</td>
</tr>
<tr>
<td>GroupLevel(1)</td>
<td>The second field or expression you sort or group on.</td>
</tr>
<tr>
<td>GroupLevel(2)</td>
<td>The third field or expression you sort or group on.</td>
</tr>
</tbody>
</table>

You can use this property only by using Visual Basic to set the SortOrder, GroupOn, GroupInterval, KeepTogether, and ControlSource properties. You set these properties in the Open event procedure of a report.

In reports, you can group or sort on more than one field or expression. Each field or expression you group or sort on is a group level.

You specify the fields and expressions to sort and group on by using the CreateGroupLevel method.

If a group is already defined for a report (the GroupLevel property is set to 0), then you can use the ControlSource property to change the group level in the report's Open event procedure. For example, the following code changes the ControlSource property to a value contained in the txtPromptYou text box on the open form named SortForm:

```vbnet
Private Sub Report_Open(Cancel As Integer)
    Me.GroupLevel(0).ControlSource _
```
= Forms!SortForm!txtPromptYou
End Sub
Hyperlink Object

Hyperlink

The Hyperlink object represents a hyperlink associated with a control on a form, report, or data access page.
Using the Hyperlink Object

Use the Hyperlink property to return a reference to a hyperlink object.
Image Object

This object corresponds to an image control. The image control can add a picture to a form or report. For example, you could include an image control for a logo on an Invoice report.

**Control: Tool:**

![Image Control Icon]

**Note**  This control should not be confused with the Dynamic HTML image control used on a data access page. For information about a image control on a data access page, see Image Control (Data Access Page).

You can use the image control or an unbound object frame for unbound pictures. The advantage of using the image control is that it's faster to display. The advantage of using the unbound object frame is that you can edit the object directly from the form or report.
Label Object

**Label**  
**Properties**

This object corresponds to a label control. Labels on a form or report display descriptive text such as titles, captions, or brief instructions.

Labels have certain characteristics:

- Labels don't display values from fields or expressions.
- Labels are always unbound.
- Labels don't change as you move from record to record.

**Control:**  
**Tool:**

**Note**  
This control should not be confused with the Dynamic HTML label control used on a data access page. For information about a label control on a data access page, see Label Control (Data Access Pages).

A label can be attached to another control. When you create a text box, for example, it has an attached label that displays a caption for that text box. This label appears as a column heading in the *Datasheet view* of a form.

When you create a label by using the **Label** tool, the label stands on its own—it isn't attached to any other control. You use stand-alone labels for information such as the title of a form or report, or for other descriptive text. Stand-alone labels don't appear in Datasheet view.
Line Object

The line control displays a horizontal, vertical, or diagonal line on a form or report.

**Note** This control should not be confused with the Dynamic HTML line control used on a data access page. For information about a line control on a data access page, see Line Control (Data Access Pages).
Using the Line object

You can use the **BorderWidth** property to change the line width. You can use the **BorderColor** property to change the color of the border or make it transparent. You can change the line style (dots, dashes, and so on) of the border by using the **BorderStyle** property.
Show All
ListBox Object

ListBox  Properties

This object corresponds to a list box control. The list box control displays a list of values or alternatives.

In many cases, it's quicker and easier to select a value from a list than to remember a value to type. A list of choices also helps ensure that the value that's entered in a field is correct.

Control:  Tool:

Note  The list in a list box consists of rows of data. Rows can have one or more columns, which can appear with or without headings.

Note  This control should not be confused with the Dynamic HTML list box control used on a data access page. For information about a list box control on a data access page, see List Box Control (Data Access Pages).

If a multiple-column list box is bound, Microsoft Access stores the values from one of the columns.

You can use an unbound list box to store a value that you can use with another control. For example, you could use an unbound list box to limit the values in another list box or in a custom dialog box. You could also use an unbound list box to find a record based on the value you select in the list box.

If you don't have room on your form to display a list box, or if you want to be able to type new values as well as select values from a list, use a combo box.
instead of a list box.
Module Object

Multiple objects - Module

A Module object refers to a standard module or a class module.
Using the Module Object

Microsoft Access includes class modules that are not associated with any object, and form modules and report modules, which are associated with a form or report.

To determine whether a Module object represents a standard module or a class module from code, check the Module object's Type property.

The Modules collection contains all open Module objects, regardless of their type. Modules in the Modules collection can be compiled or uncompiled.

To return a reference to a particular standard or class Module object in the Modules collection, use any of the following syntax forms.

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modules!modulename</td>
<td>The modulename argument is the name of the Module object.</td>
</tr>
<tr>
<td>Modules(&quot;modulename&quot;)</td>
<td>The modulename argument is the name of the Module object.</td>
</tr>
<tr>
<td>Modules(index)</td>
<td>The index argument is the numeric position of the object within the collection.</td>
</tr>
</tbody>
</table>

The following example returns a reference to a standard Module object and assigns it to an object variable:

```vba
Dim mdl As Module
Set mdl = Modules!"Utility Functions"
```

Note that the brackets enclosing the name of the Module object are necessary only if the name of the Module includes spaces.

The next example returns a reference to a form Module object and assigns it to an object variable:

```vba
Dim mdl As Module
Set mdl = Modules!Form_Employees
```
To refer to a specific form or report module, you can also use the **Form** or **Report** object's **Module** property:

`Forms!formname.Module`

The following example also returns a reference to the **Module** object associated with an Employees form and assigns it to an object variable:

```vbnet
Dim mdl As Module
Set mdl = Forms!Employees.Module
```

Once you've returned a reference to a **Module** object, you can set or read its properties and apply its methods.
ObjectFrame Object

**ObjectFrame**  \[\text{Properties}\]

This object corresponds to an unbound object frame. The unbound object frame control displays a picture, chart, or any [OLE object](#) not stored in a table.

For example, you can use an unbound object frame to display a chart that you created and stored in Microsoft Graph.

This control allows you to create or edit the object from within a Microsoft Access form or report by using the application in which the object was originally created.

To display objects that are stored in a Microsoft Access database, use a [bound object frame control](#).

The object in an unbound object frame is the same for every record.

The unbound object frame can display [linked](#) or [embedded](#) objects.

**Tip**

You can use the unbound object frame or an [image control](#) to display unbound pictures in a form or report. The advantage of using the unbound object frame is that you can edit the object directly from the form or report. The advantage of using the image control is that it's faster to display.
OptionButton Object

**OptionButton**

**Properties**

An option button on a form or report is a stand-alone control used to display a Yes/No value from an underlying record source.

**Note** This control should not be confused with the Dynamic HTML option button control used on a data access page. For information about an option button control on a data access page, see [Option Button Control (Data Access Pages)](#).
Using the OptionButton object

When you select or clear an option button that's bound to a Yes/No field, Microsoft Access displays the value in the underlying table according to the field's Format property (Yes/No, True/False, or On/Off).

You can also use option buttons in an option group to display values to choose from.

It's also possible to use an unbound option button in a custom dialog box to accept user input.
OptionGroup Object

OptionGroup Properties

An option group on a form or report displays a limited set of alternatives. An option group makes selecting a value easy since you can just click the value you want. Only one option in an option group can be selected at a time.

An option group consists of a group frame and a set of check boxes, toggle buttons, or option buttons.

Note This control should not be confused with the Dynamic HTML option group control used on a data access page. For information about a option group control on a data access page, see Option Group Control (Data Access Pages).
Using the OptionGroup object

If an option group is bound to a field, only the group frame itself is bound to the field, not the check boxes, toggle buttons, or option buttons inside the frame. Instead of setting the ControlSource property for each control in the option group, you set the OptionValue property of each check box, toggle button, or option button to a number that's meaningful for the field to which the group frame is bound. When you select an option in an option group, Microsoft Access sets the value of the field to which the option group is bound to the value of the selected option's OptionValue property.

**Note** The OptionValue property is set to a number because the value of an option group can only be a number, not text. Microsoft Access stores this number in the underlying table. In the preceding example, if you want to display the name of the shipper instead of a number in the Orders table, you can create a separate table called Shippers that stores shipper names, and then make the ShipVia field in the Orders table a Lookup field that looks up data in the Shippers table.

An option group can also be set to an expression, or it can be unbound. You can use an unbound option group in a custom dialog box to accept user input and then carry out an action based on that input.
Page Object

Multiple objects - Pages
  - Page
  - Properties

A Page object corresponds to an individual page on a tab control.
Using the Page Object

A Page object is a member of a tab control's Pages collection.

To return a reference to a particular Page object in the Pages collection, use any of the following syntax forms.

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pages!pagename</td>
<td>The pagename argument is the name of the Page object.</td>
</tr>
<tr>
<td>Pages(&quot;pagename&quot;)</td>
<td>The pagename argument is the name of the Page object.</td>
</tr>
<tr>
<td>Pages(index)</td>
<td>The index argument is the numeric position of the object within the collection.</td>
</tr>
</tbody>
</table>

You can create, move, or delete Page objects and set their properties either in Visual Basic or in form Design view. To create a new Page object in Visual Basic, use the Add method of the Pages collection. To delete a Page object, use the Remove method of the Pages collection.

To create a new Page object in form Design view, right-click the tab control and then click Insert Page on the shortcut menu. You can also copy an existing page and paste it. You can set the properties of the new Page object in form Design view by using the property sheet.

Each Page object has a PageIndex property that indicates its position within the Pages collection. The Value property of the tab control is equal to the PageIndex property of the current page. You can use these properties to determine which page is currently selected after the user has switched from one page to another, or to change the order in which the pages appear in the control.

A Page object is also a type of Control object. The ControlType property constant for a Page object is acPage. Although it is a control, a Page object belongs to a Pages collection, rather than a Controls collection. A tab control's Pages collection is a special type of Controls collection.

Each Page object can also contain one or more controls. Controls on a Page object belong to that Page object's Controls collection. In order to work with a
control on a Page object, you must refer to that control within the Page object's Controls collection.
PageBreak Object

**PageBreak**  **Properties**

This object corresponds to a page break control. The page break control marks the start of a new screen or printed page on a form or report.

**Control: Tool:**

In a form, a page break is active only when you set the form's `DefaultView` property to Single Form. Page breaks don't affect a form's datasheet.

In **Form view**, press the PAGE UP or PAGE DOWN key to move to the previous or next page break.

Position page breaks above or below other controls. Placing a page break on the same line as another control splits that control's data.
A **Printer** object corresponds to a printer available on your system.
Using the Printer object

A **Printer** object is a member of the **Printers** collection.

To return a reference to a particular **Printer** object in the **Printers** collection, use any of the following syntax forms.

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Printer</strong></td>
<td>The <em>devicename</em> argument is the name of the <strong>Printer</strong> object as returned by the <strong>DeviceName</strong> property.</td>
</tr>
<tr>
<td><strong>Printer</strong>! <em>devicename</em></td>
<td>The <em>devicename</em> argument is the name of the <strong>Printer</strong> object as returned by the <strong>DeviceName</strong> property.</td>
</tr>
<tr>
<td><strong>Printer</strong> <em>(&quot;devicename&quot;)</em></td>
<td>The <em>index</em> argument is the numeric position of the object within the collection. The valid range is from 0 to <strong>Printers.Count-1</strong>.</td>
</tr>
</tbody>
</table>

You can use the properties of the **Printer** object to set the printing characteristics for any of the printers available on your system.

Use the **ColorMode**, **Copies**, **Duplex**, **Orientation**, **PaperBin**, **PaperSize**, and **PrintQuality** properties to specify print settings for a particular printer.

Use the **LeftMargin**, **RightMargin**, **TopMargin**, **BottomMargin**, **ColumnSpacing**, **RowSpacing**, **DataOnly**, **DefaultSize**, **ItemLayout**, **ItemsAcross**, **ItemSizeHeight**, and **ItemSizeWidth** properties to specify how Microsoft Access should format the appearance of data on printed pages.

Use the **DeviceName**, **DriverName**, and **Port** properties to return system information about a particular printer.

The following example displays system information about the first printer in the **Printers** collection.

```vba
Dim prtFirst As Printer
Set prtFirst = Application.Printers(0)

With prtFirst
    MsgBox "Device name: " & .DeviceName & vbCrLf 
```


& "Driver name: " & .DriverName & vbCrLf _
& "Port: " & .Port
End With
Rectangle Object

This object corresponds to a rectangle control. The rectangle control displays a rectangle on a form or report.

**Control: Tool:**

This control should not be confused with the Dynamic HTML rectangle control used on a data access page. For information about a rectangle control on a data access page, see Rectangle Control (Data Access Page).

You can move a rectangle and the controls in it as a single unit by dragging the mouse pointer diagonally across the entire rectangle to select all of the controls. The entire selection can then be moved to a new position.
Reference Object

The Reference object refers to a reference set to another application's or project's type library.
Using the Reference Object

When you create a **Reference** object, you set a reference dynamically from Visual Basic.

The **Reference** object is a member of the **References** collection. To refer to a particular **Reference** object in the **References** collection, use any of the following syntax forms.

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>References!referencename</code></td>
<td>The <code>referencename</code> argument is the name of the <strong>Reference</strong> object.</td>
</tr>
<tr>
<td><code>References(&quot;referencename&quot;)</code></td>
<td>The <code>referencename</code> argument is the name of the <strong>Reference</strong> object.</td>
</tr>
<tr>
<td><code>References(index)</code></td>
<td>The <code>index</code> argument is the object's numerical position within the collection.</td>
</tr>
</tbody>
</table>

**Note** The following example refers to the **Reference** object that represents the reference to the Microsoft Access type library:

```vba
Dim ref As Reference
Set ref = References!Access
```
Report Object

Multiple objects — Report
— Multiple objects

A Report object refers to a particular Microsoft Access report.
Using the Report Object

A Report object is a member of the Reports collection, which is a collection of all currently open reports. Within the Reports collection, individual reports are indexed beginning with zero. You can refer to an individual Report object in the Reports collection either by referring to the report by name, or by referring to its index within the collection. If the report name includes a space, the name must be surrounded by brackets ([ ]).

### Syntax

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reports!reportname</td>
<td>Reports!OrderReport</td>
</tr>
<tr>
<td>Reports![report name]</td>
<td>Reports![Order Report]</td>
</tr>
<tr>
<td>Reports(&quot;reportname&quot;)</td>
<td>Reports(&quot;OrderReport&quot;)</td>
</tr>
<tr>
<td>Reports(index)</td>
<td>Reports(0)</td>
</tr>
</tbody>
</table>

### Note

Each Report object has a Controls collection, which contains all controls on the report. You can refer to a control on a report either by explicitly or implicitly referring to the Controls collection. Your code will be faster if you refer to the Controls collection implicitly. The following examples show two of the ways you might refer to a control named NewData on a report called OrderReport.

' Implicit reference.
Reports!OrderReport!NewData

' Explicit reference.
Reports!OrderReport.Controls!NewData
Screen Object

The **Screen** object refers to the particular form, report, or control that currently has the **focus**.
Using the Screen Object

You can use the **Screen** object together with its properties to refer to a particular form, report, or control that has the focus.

For example, you can use the **Screen** object with the **ActiveForm** property to refer to the form in the active window without knowing the form's name. The following example displays the name of the form in the active window:

```vba
MsgBox Screen.ActiveForm.Name
```

Referring to the **Screen** object doesn't make a form, report, or control active. To make a form, report, or control active, you must use the **SelectObject** method of the **DoCmd** object.

If you refer to the **Screen** object when there's no active form, report, or control, Microsoft Access returns a **run-time error**. For example, if a **standard module** is in the active window, the code in the preceding example would return an error.

The following example uses the **Screen** object to print the name of the form in the active window and of the active control on that form:

```vba
Sub ActiveObjects()
  Dim frm As Form, ctl As Control

    ' Return Form object pointing to active form.
    Set frm = Screen.ActiveForm
    MsgBox frm.Name & " is the active form."
    ' Return Control object pointing to active control.
    Set ctl = Screen.ActiveControl
    MsgBox ctl.Name & " is the active control " & "on this form."
End Sub
```
Section Object

Multiple objects [Section

Properties

A form section is part of a form such as a header, footer, or detail section.
Using the Section Object

You can set section properties which are attributes of a form that affect the appearance or behavior of that section. For example, you can set the \texttt{CanGrow} property to specify whether the section will increase vertically to print all the data the section contains. Section properties are set in \texttt{form Design view}.
SmartTag Object

SmartTags  SmartTag

Represents a smart tag that has been added to a control on a form, report, or data access page. The SmartTag object is a member of the SmartTags collection.
Using the SmartTag Object

To return a single SmartTag object, use the Item method of the SmartTags collection, or use SmartTags (Index), where Index represents the number of the smart tag.

Note  Unlike the SmartTags collections in Microsoft Excel and Microsoft Word, the SmartTags collection in Microsoft Access is zero-based. Therefore, the code control.SmartTags(0) returns the first smart tag for the specified control.

To return the collection of actions available for the smart tag, use the SmartTagActions property. To perform a smart tag action, use the Execute method of the SmartTagAction object.
SmartTagAction Object

SmartTagActions ⊂ SmartTagAction

Represents a single action for a smart tag.

Smart tag actions are processes that are programmed into smart tags that allow users to perform certain functions related to the smart tag. For example, one action for a smart tag might be to access a Web site, while another action inserts contact information from Microsoft Outlook, while yet another action displays a map and driving directions.
Using the SmartTagAction Object

To perform an action represented by a SmartTagAction object, use the Execute method.

To return the parent control of the SmartTag object, use the Parent property.
SmartTagProperty Object

\texttt{SmartTagProperties} \leftarrow \texttt{SmartTagProperty}

Represents a single instance of a custom property for a \texttt{smart tag}. The \texttt{SmartTagProperty} object is a member of the \texttt{SmartTagProperties} collection.
Using the SmartTagProperty Object

To return a SmartTagProperty object, use the Item method of the SmartTagProperties collection, or use SmartTagProperties (Index), where Index is the number of the property. To return the information related to a custom property for a smart tag, use the Name and Value properties of the SmartTagProperty object.
Show All
SubForm Object

SubForm | Multiple objects

This object corresponds to a subform control. The subform control embeds a form in a form.

Control: | Tool:

<table>
<thead>
<tr>
<th>CategoryID: BEVR</th>
<th>Subform</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProductID</td>
<td>Quantity</td>
</tr>
<tr>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>Record:</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Note  For example, you can use a form with a subform to present one-to-many relationships, such as one product category with the items that fall into that category. In this case, the main form can display the category ID, name, and description; the subform can display the available products in that category.

Tip  Instead of creating the main form, and then adding the subform control to it, you can simultaneously create the main form and subform with a wizard. You can also create a subform by dragging an existing form or report from the Database window to the main form.
SubReport Object

SubReport — Multiple objects

This object corresponds to a subreport control. A subreport control embeds a report in a report.

Tip You can create a subreport by dragging an existing report from the Database window to the main report.
TabControl Collection

TabControl

A tab control contains multiple pages on which you can place other controls, such as text boxes or option buttons. When a user clicks the corresponding tab, that page becomes active.
Using the TabControl collection

With the tab control, you can construct a single form or dialog box that contains several different tabs, and you can group similar options or data on each tab's page. For example, you might use a tab control on an Employees form to separate general and personal information.
TextBox Object

This object corresponds to a text box. Text boxes on a form or report display data from a record source.

This type of text box is called a bound text box because it's bound to data in a field. Text boxes can also be unbound. For example, you can create an unbound text box to display the results of a calculation, or to accept input from a user. Data in an unbound text box isn't saved with the database.

**Control:**

**Tool:**

Note  This control should not be confused with the Dynamic HTML text box control used on a data access page. For information about a text box control on a data access page, see Text Box Control (Data Access Pages).
ToggleButton Object

ToggleButton

Properties

This object corresponds to a toggle button. A toggle button on a form is a stand-alone control used to display a Yes/No value from an underlying record source.

Control:  Tool:

[Discontinued]  [Discontinued]

Note  When you click a toggle button that's bound to a Yes/No field, Microsoft Access displays the value in the underlying table according to the field's Format property (Yes/No, True/False, or On/Off).

Toggle buttons are most useful when used in an option group with other buttons.

You can also use a toggle button in a custom dialog box to accept user input.
WebOptions Object

DataAccessPage → WebOptions

A WebOptions object refers to a specific Microsoft Access data access page's web option properties.
Using the WebOptions Object

Use the **WebOptions** property to return the **WebOptions** object.

Contains data access page attributes used by Microsoft Access when you save a data access page as a Web page or open a Web page. You can return or set attributes either at the application (global) level or at the data access page level. (Note that attribute values can be different from one data access page to another, depending on the attribute value at the time the data access page was saved.) Data access page-level attributes override application-level attributes. Application-level attributes are contained in the **DefaultWebOptions** object.

Remember that if you change the value of a data access page-level attribute, the corresponding application-level attribute is automatically set to the same value. Therefore, you should restore and save the application-level values for a given data access page during each session in that data access page.

The following example checks to see whether Microsoft Office Web components are downloaded when a saved data access page ("Inventory") is displayed and sets the download flag accordingly.

```vba
Set objAppWebOptions = DataAccessPages("Inventory").WebOptions
With objAppWebOptions
    If .DownloadComponents = True Then
        strCompDownload = "Loaded"
    Else
        strCompDownload = "Not Loaded"
    End If
End With
```
**AccessError Method**

You can use the `AccessError` method to return the descriptive string associated with a Microsoft Access or DAO error. **Variant.**

```
expression.AccessError(ErrorNumber)
```

*expression*  Required. An expression that returns one of the objects in the Applies To list.

*ErrorNumber*  Required **Variant.** The number of the error for which you wish to return a descriptive string.
Remarks

You can use the **AccessError** method to return the descriptive string associated with a Microsoft Access or DAO error when the error hasn't actually occurred, but you cannot use it for ADO errors.

You can use the Visual Basic **Raise** method to raise a Visual Basic error. Once you've raised the error, you can determine its associated descriptive string by reading the **Description** property of the **Err** object.

You can't use the **Raise** method to raise a Microsoft Access or DAO error. However, you can use the **AccessError** method to return the descriptive string associated with these errors, without having to generate the error.

You can use the **AccessError** method to return a descriptive string from within a form's Error event.

If the Microsoft Access error has occurred, you can return the descriptive string by using either the **AccessError** method or the **Description** property of the Visual Basic **Err** object.
Example

The following function returns an error string for any valid error number:

**Note** You must have your error trapping options set to Break on Unhandled Errors for the code to run in the VBA IDE. You can set this option on the General tab of the Options dialog found on the VBA Tools menu.

Function ErrorString(ByVal lngError As Long) As String

    Const conAppError = "Application-defined or " & _
"object-defined error"

    On Error Resume Next
    Err.Raise lngError

    If Err.Description = conAppError Then
        ErrorString = AccessError(lngError)
    ElseIf Err.Description = vbNullString Then
        MsgBox "No error string associated with this number."
    Else
        ErrorString = Err.Description
    End If

End Function
Add Method

Add method as it applies to the AccessObjectProperties collection object.

You can use the Add method to add a new property as an AccessObjectProperty object to the AccessObjectProperties collection of an AccessObject object.

expression>Add(PropertyName, Value)

expression  Required. An expression that returns an AccessObjectProperties collection object.

PropertyName  Required String. A string expression that's the name of the new property.

Value  Required Variant. A Variant value corresponding to the option setting. The setting of the value argument depends on the possible settings for a particular option. Can be a constant or a string value.
Remarks

You can use the **Remove** method of the **AccessObjectProperties** collection to delete an existing property.

\[ \text{Add method as it applies to the FormatConditions collection object.} \]

You can use the **Add** method to add a conditional format as a **FormatCondition** object to the **FormatConditions** collection of a combo box or text box control.

\[ \text{expression.Add}(\text{Type, Operator, Expression1, Expression2}) \]

\[ \text{expression} \quad \text{Required. An expression that returns a FormatConditions collection object.} \]

**Type**  Required **AcFormatConditionType**. The type of format condition to be added.

AcFormatConditionType can be one of these AcFormatConditionType constants.

- **acExpression**
- **acFieldHasFocus**
- **acFieldValue**

**Operator**  Optional **AcFormatConditionOperator**. If the **Type** argument is **acExpression**, the **Operator** argument is ignored. If you leave this argument blank, the default constant (**acBetween**) is assumed.

AcFormatConditionOperator can be one of these AcFormatConditionOperator constants.

- **acBetween**  default
- **acEqual**
- **acGreaterThan**
- **acGreaterThanOrEqual**
- **acLessThan**
- **acLessThanOrEqual**
acNotBetween
acNotEqual

*Expression1*  Optional **Variant**. A **Variant** value or expression associated with the first part of the conditional format. Can be a constant or a string value.

*Expression2*  Optional **Variant**. A **Variant** value or expression associated with the second part of the conditional format when the **Operator** argument is **acBetween** or **acNotBetween** (otherwise, this argument is ignored). Can be a constant or a string value.
Remarks

You can use the **Delete** method of the **FormatConditions** collection to delete an existing **FormatConditions** collection from a combo box or text box control.

**Add method as it applies to the Pages collection object.**

The **Add** method adds a new **Page** object to the **Pages** collection of a tab control.

```
expression.Add(Before)
```

**expression**  Required. An expression that returns a **Pages** collection object.

**Before**  Optional **Variant**. An **Integer** that specifies the index of the **Page** object before which the new **Page** object should be added. The index of the **Page** object corresponds to the value of the **PageIndex** property for that **Page** object. If you omit this argument, the new **Page** object is added to the end of the collection.
Remarks

The first Page object in the Pages collection corresponds to the leftmost page in the tab control and has an index of 0. The second Page object is immediately to the right of the first page and has an index of 1, and so on for all the Page objects in the tab control.

If you specify 0 for the Before argument, the new Page object is added before the first Page object in the Pages collection. The new Page object then becomes the first Page object in the collection, with an index of 0.

You can add a Page object to the Pages collection of a tab control only when the form is in Design view.

expression.Add(var)

expression  Required. An expression that returns an AdditionalData object.

var  Required String. The name of the table or query to add.

expression.Add(Name, Value)

expression  Required. An expression that returns a SmartTagProperties object.

Name  Required String. The name to be used for the custom property.

Value  Required Variant. The value of the custom property.
Adds a smart tag to a form, control, or data access page. Returns SmartTag.

`expression.Add(Name)`

`expression` Required. An expression that returns one a SmartTags collection.

`Name` Required String. The name of the smart tag to add.
Example

As it applies to the Pages collection object.

The following example adds a page to a tab control on a form that's in Design view. To try this example, create a new form named Form1 with a tab control named TabCtl0. Paste the following code into a standard module and run it:

```vba
Function AddPage() As Boolean
    Dim frm As Form
    Dim tbc As TabControl, pge As Page

    On Error GoTo Error_AddPage
    Set frm = Forms!Form1
    Set tbc = frm!TabCtl0
    tbc.Pages.Add
    AddPage = True

Exit_AddPage:
    Exit Function

Error_AddPage:
    MsgBox Err & "\: " & Err.Description
    AddPage = False
    Resume Exit_AddPage
End Function
```
AddFromFile Method

AddFromFile method as it applies to the References object.

The AddFromFile method creates a reference to a type library in a specified file. Reference object.

expression.AddFromFile(FileName)

expression Required. An expression that returns one of the above objects.

FileName Required String. A string expression that evaluates to the full path and file name of the file containing the type library to which you wish to set a reference.

AddFromFile method as it applies to the Module object.

The AddFromFile method adds the contents of a text file to a Module object. The Module object may represent a standard module or a class module.

expression.AddFromFile(FileName)

expression Required. An expression that returns one of the above objects.

FileName Required String. The name and full path of a text (.txt) file or another file that stores text in an ANSI format.
Remarks

As it applies to the References object.

The following table lists types of files that commonly contain type libraries.

<table>
<thead>
<tr>
<th>File extension</th>
<th>Type of file</th>
</tr>
</thead>
<tbody>
<tr>
<td>.olb, .tlb</td>
<td>Type library file</td>
</tr>
<tr>
<td>.mdb, .mda, .mde</td>
<td>Database</td>
</tr>
<tr>
<td>.exe, .dll</td>
<td>Executable file</td>
</tr>
<tr>
<td>.ocx</td>
<td>ActiveX control</td>
</tr>
</tbody>
</table>

As it applies to the Module object.

The AddFromFile method places the contents of the specified text file immediately after the Declarations section and before the first procedure in the module if it contains other procedures.

The AddFromFile method enables you to import code or comments stored in a text file.

In order to add the contents of a file to a form or report module, the form or report must be open in form Design view or report Design view. In order to add the contents of a file to a standard module or class module, the module must be open.
Example

As it applies to the References object.

The following example adds a reference to the Microsoft Scripting Runtime library.

References.AddFromFile "C:\WINNT\system32\scrrun.dll"

As it applies to the Module object.

The following example places the contents of the file "ShippingRoutines.bas" into the module "CalculateShipping" immediately after the Declarations section, but before the first procedure in the module.

 Modules("CalculateShipping").AddFromFile "C:\Shipping\ShippingRoutines.bas"
AddFromGuid Method

The AddFromGUID method creates a Reference object based on the GUID that identifies a type library. Reference object.

expression.AddFromGuid(Guid, Major, Minor)

expression Required. An expression that returns one of the objects in the Applies To list.

Guid Required String. A GUID that identifies a type library.

Major Required Long.

Minor Required Long.
Remarks

The **GUID** property returns the GUID for a specified **Reference** object. If you've stored the value of the **GUID** property, you can use it to re-create a reference that's been broken.
Example

The following example re-creates a reference to the **Microsoft Scripting Runtime** version 1.0, based on its GUID on the user's system.

References. AddFromGuid "{420B2830-E718-11CF-893D-00A0C9054228}", 1,
Show All
AddFromString Method

The AddFromString method adds a string to a Module object. The Module object may represent a standard module or a class module.

expression.AddFromString(String)

expression Required. An expression that returns one of the objects in the Applies To list.

String Required String.
Remarks

The **AddFromString** method places the contents of a string after the **Declarations section** and before the first existing procedure in the module if the module contains other procedures.

In order to add a string to a **form** or **report** module, the form or report must be open in **form Design view** or **report Design view**. In order to add a string to a standard module or class module, the module must be open.
Example

This example creates a new form and adds a string and the contents of the Functions.txt file to its module. Run the following procedure from a standard module:

Sub AddTextToFormModule()
    Dim frm As Form, mdl As Module

    Set frm = CreateForm
    Set mdl = frm.Module
    mdl.AddFromString "Public intY As Integer"
    mdl.AddFromFile "C:\My Documents\Functions.txt"
End Sub
AddItem Method

Adds a new item to the list of values displayed by the specified list box control or combo box control.

expression.AddItem(Item, Index)

expression  Required. An expression that returns one of the objects in the Applies To list.

Item  Required String. The display text for the new item.

Index  Optional Variant. The position of the item in the list. If this argument is omitted, the item is added to the end of the list.
Remarks

The **RowSourceType** property of the specified control must be set to "Value List".

This method is only valid for list box or combo box controls on forms.

List item numbers start from zero. If the value of the **Index** argument doesn't correspond to an existing item number, an error occurs.

For multiple-column lists, use semicolons to delimit the strings for each column (for example, "1010;red;large" for a three-column list). If the **Item** argument contains fewer strings than columns in the control, items will be added starting with the left-most column. If the **Item** argument contains more strings than columns in the control, the extra strings are ignored.

Use the **RemoveItem** method to remove items from the list of values.
Example

This example adds an item to the end of the list in a list box control. For the function to work, you must pass it a ListBox object representing a list box control on a form and a String value representing the text of the item to be added.

Function AddItemToEnd(ctrlListBox As ListBox, ByVal strItem As String)
    ctrlListBox.AddItem Item:=strItem
End Function

This example adds an item to the beginning of the list in a combo box control. For the function to work, you must pass it a ComboBox object representing a combo box control on a form and a String value representing the text of the item to be added.

Function AddItemToBeginning(ctrlComboBox As ComboBox, ByVal strItem As String)
    ctrlComboBox.AddItem Item:=strItem, Index:=0
End Function
AddMenu Method

The **AddMenu** method carries out the **AddMenu** action in Visual Basic.

expression.**AddMenu**(**MenuName**, **MenuMacroName**, **StatusBarText**)  

**expression**  Required. An expression that returns one of the objects in the Applies To list.

**MenuName**  Required **Variant**. A **string expression** that's the valid name of a drop-down menu to add to the custom menu bar or global menu bar. To create an **access key** so that you can use the keyboard to choose the menu, type an ampersand (&) before the letter you want to be the access key. This letter will be underlined in the menu name on the menu bar.

**MenuMacroName**  Required **Variant**. A string expression that's the valid name of the **macro group** that contains the macros for the menu's commands. This is a required argument.

**StatusBarText**  Required **Variant**. A string expression that's the text to display in the **status bar** when the menu is selected.
Remarks

You must include the `menuname` and `menumacroname` arguments in the `AddMenu` method for custom menu bars and global menu bars. The `menuname` argument is not required and will be ignored for custom shortcut menus and global shortcut menus.

The `statusbartext` argument is optional, this argument is ignored for custom shortcut menus and global shortcut menus.
AddToFavorites Method

The AddToFavorites method adds a hyperlink address to the Favorites folder.

`expression.AddToFavorites`

`expression` Required. An expression that returns one of the objects in the Applies To list.
Remarks

When applied to the Application object, the AddToFavorites method adds the name of the current database to the Favorites folder. For example, if you're working in the Northwind sample database, applying the AddToFavorites method to the Application object adds the hyperlink address of the Northwind database to the Favorites folder.

When applied to a Control object, the AddToFavorites method adds the hyperlink address contained in a control to the Favorites folder. The Favorites folder is installed in the Windows folder by default.

The AddToFavorites method has the same effect as clicking AddToFavorites on the Favorites menu on the Web toolbar when the document whose address you wish to add is open.
Example

The following example sets the **HyperlinkAddress** property of a command button. When the user clicks the command button, the address is added to the Favorites folder by using the **AddToFavorites** method.

To try this example, create a new form and add a command button named Command0. Paste the following code into the form's module. Switch to Form view and click the command button.

```vbnet
Private Sub Form_Load()
    Me!Command0.HyperlinkAddress = "http://www.microsoft.com/
End Sub

Private Sub Command0_Click()
    Me!Command0.Hyperlink.AddToFavorites
End Sub
```
ApplyFilter Method

The **ApplyFilter** method carries out the **ApplyFilter** action in Visual Basic.

*expression*.**ApplyFilter(FilterName, WhereCondition)*

*expression*  Required. An expression that returns one of the objects in the Applies To list.

**FilterName**  Optional **Variant**. A string expression that's the valid name of a filter or query in the current database. When using this method to apply a server filter, the **FilterName** argument must be blank.

**WhereCondition**  Optional **Variant**. A string expression that's a valid SQL **WHERE clause** without the word WHERE.
Remarks

For more information on how the action and its arguments work, see the action topic.

The filter and WHERE condition you apply become the setting of the form's **Filter** property or report's **ServerFilter** property.

You must include at least one of the two **ApplyFilter** method arguments. If you enter a value for both arguments, the **WhereCondition** argument is applied to the filter.

The maximum length of the **WhereCondition** argument is 32,768 characters (unlike the Where Condition action argument in the Macro window, whose maximum length is 256 characters).

If you specify the **WhereCondition** argument and leave the **FilterName** argument blank, you must include the **FilterName** argument's comma.
Example

The following example uses the `ApplyFilter` method to display only records that contain the name King in the LastName field:

```vba
DoCmd.ApplyFilter, "LastName = 'King'
```
ApplyTheme Method

You can use the `ApplyTheme` method to specify the Microsoft Office theme for a specified data access page.

`expression.ApplyTheme(ThemeName)`

- `expression`  Required. An expression that returns one of the objects in the Applies To list.

- `ThemeName`  Required `String`. The name of the Office theme.
Remarks

The **ApplyTheme** method provides a means of changing environment options from [Visual Basic](https://en.wikipedia.org/wiki/Visual_Basic) code.
Example

The following example applies the Artsy theme to the data access page named "Switchboard".

`DataAccessPages("Switchboard").ApplyTheme "Artsy"`
Beep Method

The **Beep** method carries out the **Beep** action in Visual Basic.

*expression*.**Beep**

*expression*  Required. An expression that returns one of the objects in the Applies To list.
Remarks

This method has no arguments and can called directly using the syntax `DoCmd.BEEP`.

You can also use the VBA command `Interaction.BEEP` to accomplish the same result.

You can also use the Visual Basic `Beep` statement to cause your computer to sound a tone through its speaker.
BuildCriteria Method

The **BuildCriteria** method returns a parsed criteria string as it would appear in the query design grid, in **Filter By Form** or **Server Filter By Form** mode. For example, you may want to set a form's **Filter** or **ServerFilter** property based on varying criteria from the user. You can use the **BuildCriteria** method to construct the string expression argument for the **Filter** or **ServerFilter** property.

**String.**

*expression*.BuildCriteria(*Field*, *FieldType*, *Expression*)

*expression*  Required. An expression that returns one of the objects in the Applies To list.

*Field*  Required **String**. A **string expression** identifying the field for which you wish to define criteria.

*FieldType*  Required **Integer**. An **intrinsic constant** denoting the data type of the field. Can be set to on of the DAO **DataTypeEnum** values.

*Expression*  Required **String**. A string expression identifying the criteria to be parsed.
Remarks

The **BuildCriteria** method returns a string.

The **BuildCriteria** method enables you to easily construct criteria for a filter based on user input. It parses the *expression* argument in the same way that the expression would be parsed had it been entered in the query design grid, in Filter By Form or Server Filter By Form mode.

For example, a user creating a query on an Orders table might restrict the **result set** to orders placed after January 1, 1995, by setting criteria on an OrderDate field. The user might enter an expression such as the following one in the **Criteria** row beneath the OrderDate field:

>1-1-95

Microsoft Access automatically parses this expression and returns the following expression:

>\#1/1/95\#

The **BuildCriteria** method provides the same parsing from Visual Basic code. For example, to return the preceding correctly parsed string, you can supply the following arguments to the **BuildCriteria** method:

```vbnet
Dim strCriteria As String
strCriteria = BuildCriteria("OrderDate", dbDate, ">1-1-95")
```

Since you need to supply criteria for the **Filter** property in correctly parsed form, you can use the **BuildCriteria** method to construct a correctly parsed string.

You can use the **BuildCriteria** method to construct a string with multiple criteria if those criteria refer to the same field. For example, you can use the **BuildCriteria** method with the following arguments to construct a string with multiple criteria relating to the OrderDate field:

```vbnet
strCriteria = BuildCriteria("OrderDate", dbDate, ">1-1-95 and <5-1-95")
```
This example returns the following criteria string:

OrderDate>#1/1/95# And OrderDate<#5/1/95#

However, if you wish to construct a criteria string that refers to multiple fields, you must create the strings and concatenate them yourself. For example, if you wish to construct criteria for a filter to show records for orders placed after 1-1-95 and for which freight is less than $50, you would need to use the **BuildCriteria** method twice and concatenate the resulting strings.
Example

The following example prompts the user to enter the first few letters of a product's name and then uses the BuildCriteria method to construct a criteria string based on the user's input. Next, the procedure provides this string as an argument to the Filter property of a Products form. Finally, the FilterOn property is set to apply the filter.

Sub SetFilter()
    Dim frm As Form, strMsg As String
    Dim strInput As String, strFilter As String

    ' Open Products form in Form view.
    DoCmd.OpenForm "Products"
    ' Return Form object variable pointing to Products form.
    Set frm = Forms!Products
    strMsg = "Enter one or more letters of product name " & "followed by an asterisk."
    ' Prompt user for input.
    strInput = InputBox(strMsg)
    ' Build criteria string.
    strFilter = BuildCriteria("ProductName", dbText, strInput)
    ' Set Filter property to apply filter.
    frm.Filter = strFilter
    ' Set FilterOn property; form now shows filtered records.
    frm.FilterOn = True
End Sub
CancelEvent Method

The **CancelEvent** method carries out the **CancelEvent** action in Visual Basic.

*expression*.**CancelEvent**

*expression*  Required. An expression that returns a **DoCmd** object.
Remarks

This method has no arguments and can be called directly using the syntax `DoCmd.CancelEvent`.

The `CancelEvent` method has an effect only when it's run as the result of an event. This method cancels the event.

All events that can be canceled in Visual Basic have a Cancel argument. You can use this argument instead of the `CancelEvent` method to cancel the event. The `KeyPress` event and `MouseDown` event (for right-clicking only) can be canceled only in `macros`, not `event procedures`, so you must use the `CancelEvent` action in a macro to cancel these events.
Circle Method

The **Circle** method draws a circle, an ellipse, or an arc on a **Report** object when the **Print** event occurs.

**expression**.Circle(**flags**, **X**, **Y**, **radius**, **color**, **start**, **end**, **aspect**)

**expression** Required. An expression that returns one of the objects in the Applies To list.

**flags** Required **Integer**.

**X** Required. **Single** value indicating the x-coordinate of the center point of the circle, ellipse, or arc. The Scale properties (**ScaleMode**, **ScaleLeft**, **ScaleTop**, **ScaleHeight**, and **ScaleWidth**) of the **Report** object specified by the **object** argument determine the unit of measure used.

**Y** Required. **Single** value indicating the y-coordinate of the center point of the circle, ellipse, or arc. The Scale properties (**ScaleMode**, **ScaleLeft**, **ScaleTop**, **ScaleHeight**, and **ScaleWidth**) of the **Report** object specified by the **object** argument determine the unit of measure used.

**radius** Required. A **Single** value indicating the radius of the circle, ellipse, or arc. The Scale properties (**ScaleMode**, **ScaleLeft**, **ScaleTop**, **ScaleHeight**, and **ScaleWidth**) of the **Report** object specified by the **object** argument determine the unit of measure used. By default, distances are measured in **twips**.

**color** Required **Long**. A **Long** value indicating the RGB (red-green-blue) color of the circle outline. If this argument is omitted, the value of the **ForeColor** property is used. You can also use the **RGB** function or **QBColor** function to specify the color.

**start** Required **Single**. When a partial circle or ellipse is drawn, the **start** argument specifies (in radians) the beginning position of the arc. The default value for the **start** argument is 0 radians. The range is –2 pi radians to 2 pi radians.
**end** Required **Single**. When a partial circle or ellipse is drawn, the **end** argument specifies (in radians) the end position of the arc. The default value for the **end** argument is 2 pi radians. The range is –2 pi radians to 2 pi radians.

**aspect** Required. A **Single** value indicating the aspect ratio of the circle. The default value is 1.0, which yields a perfect (nonelliptical) circle on any screen.
Remarks

You can use this method only in an **event procedure** or a **macro** specified by the **event properties** for a report section, or the **OnPage** event property for a report.

When drawing a partial circle or ellipse, if the **start** argument is negative, the **Circle** method draws a radius to the position specified by the **start** argument and treats the angle as positive. If the **end** argument is negative, the **Circle** method draws a radius to the position specified by the **end** argument and again treats the angle as positive. The **Circle** method always draws in a counterclockwise (positive) direction.

To fill a circle, set the **FillColor** and **FillStyle** properties of the report. Only a closed figure can be filled. Closed figures include circles, ellipses, and pie slices, which are arcs with radius lines drawn at both ends.

When drawing pie slices, if you need to draw a radius to angle 0 to form a horizontal line segment to the right, specify a very small negative value for the **start** argument rather than 0. For example, you might specify −.0000001 for the **start** argument.

You can omit an argument in the middle of the syntax, but you must include the argument's comma before including the next argument. If you omit a trailing argument, don't use any commas following the last argument you specify.

The width of the line used to draw the circle, ellipse, or arc depends on the **DrawWidth** property setting. The way the circle is drawn on the background depends on the settings of the **DrawMode** and **DrawStyle** properties.

When you apply the **Circle** method, the **CurrentX** and **CurrentY** properties are set to the center point specified by the **x** and **y** arguments.
Example

The following example uses the **Circle** method to draw a circle, and then create a pie slice within the circle and color it red.

To try this example in Microsoft Access, create a new report. Set the **OnPrint** property of the Detail section to [Event Procedure]. Enter the following code in the report's module, then switch to Print Preview.

```vba
Private Sub Detail_Print(Cancel As Integer, PrintCount As Integer)
    Const conPI = 3.14159265359
    Dim sngHCtr As Single, sngVCtr As Single
    Dim sngRadius As Single
    Dim sngStart As Single, sngEnd As Single

    sngHCtr = Me.ScaleWidth / 2 ' Horizontal center.
    sngVCtr = Me.ScaleHeight / 2 ' Vertical center.
    sngRadius = Me.ScaleHeight / 3 ' Circle radius.
    ' Draw circle.
    Me.Circle(sngHCtr, sngVCtr), sngRadius
    sngStart = -0.00000001 ' Start of pie slice.
    sngEnd = -2 * conPI / 3 ' End of pie slice.
    Me.FillColor = RGB(255,0,0) ' Color pie slice red.
    Me.FillStyle = 0 ' Fill pie slice.
    ' Draw pie slice within circle.
    Me.Circle(sngHCtr, sngVCtr), sngRadius, , sngStart, sngEnd
End Sub
```
Close Method

The Close method carries out the Close action in Visual Basic.

expression.Close(ObjectType, ObjectName, Save)

expression Required. An expression that returns one of the objects in the Applies To list.

ObjectType Optional AcObjectType.

AcObjectType can be one of these AcObjectType constants.

acDataAccessPage
acDefault default
acDiagram
acForm
acFunction
acMacro
acModule
acQuery
acReport
acServerView
acStoredProcedure
acTable

Note If closing a module in the Visual Basic Editor (VBE), you must use acModule in the objecttype argument.

ObjectName Optional Variant. A string expression that's the valid name of an object of the type selected by the objecttype argument.

Save Optional AcCloseSave.

AcCloseSave can be one of these AcCloseSave constants.
acSaveNo

acSavePrompt default This value is ignored if you are closing a Visual Basic module. The module will be closed, but changes to the module will not be saved.

acSaveYes

If you leave this argument blank, the default constant (acSavePrompt) is assumed.
Remarks

For more information on how the action and its arguments work, see the action topic.

If you leave the objecttype and objectname arguments blank (the default constant, acDefault, is assumed for objecttype), Microsoft Access closes the active window. If you specify the save argument and leave the objecttype and objectname arguments blank, you must include the objecttype and objectname arguments' commas.

**Note** If a form has a control bound to a field that has its Required property set to 'Yes,' and the form is closed using the **Close** method without entering any data for that field, an error message is not displayed. Any changes made to the record will be aborted. When the form is closed using the Windows **Close** button, the Close action in a macro, or selecting **Close** from the **File** menu, Microsoft Access displays an alert. The following code will display an error message when attempting to close a form with a Null field, using the **Close** method.

```vba
If IsNull(Me!"Field1") Then
    If MsgBox("'Field1' must contain a value." _ & Chr(13) & Chr(10) _ & "Press 'OK' to return and enter a value." _ & Chr(13) & Chr(10) _ & "Press 'Cancel' to abort the record.", vbOKCancel, "A Required field is Null") = vbCancel Then
        DoCmd.Close

    End If

End If
```

End If
Example

The following example uses the **Close** method to close the form Order Review, saving any changes to the form without prompting:

```vba
DoCmd.Close acForm, "Order Review", acSaveYes
```
CloseConnection Method

You can use the **CloseConnection** method to close the current connection between the **CurrentProject** or **CodeProject** object in a **Microsoft Access project** (.adp) or **Access database** (.mdb) and the database specified in the project's base connection string.

(expression).CloseConnection

expression  Required. An expression that returns one of the objects in the Applies To list.
Remarks

The CloseConnection method closes the current connection of the Access project, database, or data source control, frees the ADO Connection object, and sets the Connection property to Null. The BaseConnectionString property is left unchanged. Users are prevented from calling datasoucecontrol.Connection.Close and must use this method instead.

The CloseConnection method is useful when you have opened a Microsoft Access database from another application through Automation.
CloseCurrentDatabase Method

You can use the **CloseCurrentDatabase** method to close the current database (either a [Microsoft Access database](https://en.wikipedia.org/wiki/Microsoft_Access) (.mdb) or an [Access project](https://en.wikipedia.org/wiki/Microsoft_Access) (.adp) from another application that has opened a database through [Automation](https://en.wikipedia.org/wiki/Automation_(computer_programming)).

**expression.CloseCurrentDatabase**

**expression** Required. An expression that returns one of the objects in the Applies To list.
Remarks

For example, you might use this method from Microsoft Excel to close the database currently open in the Microsoft Access window before opening another database.

The `CloseCurrentDatabase` method is useful when you have opened a Microsoft Access database from another application through Automation. Once you have created an instance of Microsoft Access from another application, you must also create a new database or specify an existing database to open. This database opens in the Microsoft Access window.

If you use the `CloseCurrentDatabase` method to close the database that is open in the current instance of Microsoft Access, you can then open a different database without having to create another instance of Microsoft Access.
**Example**

The following example opens a Microsoft Access database from another application through Automation, creates a new form and saves it, then closes the database.

You can enter this code in a Visual Basic module in any application that can act as a COM component. For example, you might run the following code from Microsoft Excel or Microsoft Visual Basic.

When the variable pointing to the **Application** object goes out of scope, the instance of Microsoft Access that it represents closes as well. Therefore, you should declare this variable at the module level.

' Enter following in Declarations section of module.
Dim appAccess As Access.Application

Sub CreateForm()
  Const strConPathToSamples = "C:\Program Files\Microsoft Office\"

  Dim frm As Form, strDB As String

  ' Initialize string to database path.
  strDB = strConPathToSamples & "Northwind.mdb"
  ' Create new instance of Microsoft Access.
  Set appAccess = CreateObject("Access.Application")
  ' Open database in Microsoft Access window.
  appAccess.OpenCurrentDatabase strDB
  ' Create new form.
  Set frm = appAccess.CreateForm
  ' Save new form.
  appAccess.DoCmd.Save , "NewForm1"
  ' Close currently open database.
  appAccess.CloseCurrentDatabase
  Set AppAccess = Nothing

End Sub
CodeDb Method

You can use the CodeDb method in a code module to determine the name of the Database object that refers to the database in which code is currently running. Use the CodeDb method to access Data Access Objects (DAO) that are part of a library database.

`expression.CodeDb`

`expression` Required. An expression that returns one of the objects in the Applies To list.
Remarks

For example, you can use the CodeDb method in a module in a library database to create a Database object referring to the library database. You can then open a recordset based on a table in the library database.

Set database = CodeDb

The CodeDb method returns a Database object for which the Name property is the full path and name of the database from which it is called. This method can be useful when you need to manipulate the Data Access Objects in your library database.

When you call a method in a library database, the database from which you have called the method remains the current database, even while code is running in a module in the library database. In order to refer to the Data Access Objects in the library database, you need to know the name of the Database object that represents the library database.

For example, suppose you have a table in a library database that lists error messages. To manipulate data in the table from code, you could use the CodeDb method to determine the name of the Database object that refers to the library database that contains the table.

If the CodeDb method is run from the current database, it returns the name of the current database, which is the same value returned by the CurrentDb method.
Example

The following example uses the **CodeDb** method to return a **Database** object that refers to a library database. The library database contains both a table named Errors and the code that is currently running. After the **CodeDb** method determines this information, the **GetErrorString** function opens a table-type recordset based on the Errors table. It then extracts an error message from a field named ErrorData based on the **Integer** value passed to the function.

```vba
Function GetErrorString(ByVal intError As Integer) As String
    Dim dbs As Database, rst As RecordSet
    
    ' Variable refers to database where code is running.
    Set dbs = CodeDb
    ' Create table-type Recordset object.
    Set rst = dbs.OpenRecordSet("Errors", dbOpenTable)
    ' Set index to primary key (ErrorID field).
    rst.Index = "PrimaryKey"
    ' Find error number passed to GetErrorString function.
    rst.Seek "=", intError
    ' Return associated error message.
    GetErrorString = rst.Fields!ErrorData.Value
    rst.Close
End Function
```
CompactRepair Method

Compacts and repairs the specified database (.mdb) or Microsoft Access project (.adp) file. Returns a **Boolean; True** if the process was successful.

**expression.**CompactRepair(SourceFile, DestinationFile, LogFile)**

**expression**  Required. An expression that returns one of the objects in the Applies To list.

**SourceFile**  Required **String**. The full path and filename of the database or project file to compact and repair.

**DestinationFile**  Required **String**. The full path and filename for where the recovered file will be saved.

**LogFile**  Optional **Boolean**. **True** if a log file is created in the destination directory to record any corruption detected in the source file. A log file is only created if corruption is detected in the source file. If **LogFile** is **False** or omitted, no log file is created, even if corruption is detected in the source file.
Remarks

The source file must not be the current database or be open by any other user, since calling this method will open the file exclusively.
Example

The following example compacts and repairs a database, creates a log if there's any corruption in the source file, and returns a Boolean value based on whether the recovery was successful. For the example to work, you must pass it the paths and file names of the source and destination files.

Function RepairDatabase(strSource As String, _
    strDestination As String) As Boolean
    ' Input values: the paths and file names of
    ' the source and destination files.
    
    ' Trap for errors.
    On Error GoTo error_handler
    
    ' Compact and repair the database. Use the return value of
    ' the CompactRepair method to determine if the file was
    ' successfully compacted.
    RepairDatabase = _
        Application. CompactRepair( _
            LogFile:=True, _
            SourceFile:=strSource, _
            DestinationFile:=strDestination)
        
    ' Reset the error trap and exit the function.
    On Error GoTo 0
    Exit Function
    
    ' Return False if an error occurs.
    error_handler:
        RepairDatabase = False
    
End Function
ConvertAccessProject Method

Converts the specified Microsoft Access file from one version to another.

`expression.ConvertAccessProject(SourceFilename, DestinationFilename, DestinationFileFormat)`

`expression` Required. An expression that returns one of the objects in the Applies To list.

**SourceFilename** Required **String**. The name of the Access file to convert. If a path isn't specified, Access looks for the file in the current directory.

**DestinationFilename** Required **String**. The name of the file where Access saves the converted file. If a path isn't specified, Access saves the file in the current directory.

**DestinationFileFormat** Required **AcFileFormat**. The Access version of the converted file.

AcFileFormat can be one of these AcFileFormat constants.
- `acFileFormatAccess2`
- `acFileFormatAccess2000`
- `acFileFormatAccess2002`
- `acFileFormatAccess95`
- `acFileFormatAccess97`
Remarks

The file specified by *DestinationFilename* cannot already exist, or an error occurs.
Example

The following example converts the specified Access 97 file to an Access 2000 file in the same directory.

Application.ConvertAccessProject _
    SourceFilename:="C:\My Documents\Sales-Access97.mdb", _
CopyDatabaseFile Method

Copies the database connected to the current project to a Microsoft SQL Server database file for export.

\[\text{expression}.\text{CopyDatabaseFile( DatabaseFileName, OverwriteExistingFile, DisconnectAllUsers)}\]

\textit{expression} Required. An expression that returns a \texttt{DoCmd} object.

\textit{DatabaseFileName} Required \texttt{Variant}. The name of the file (and path) to which the current database is copied. If no path is specified, the current directory is used.

\textit{OverwriteExistingFile} Optional \texttt{Variant}. Determines whether Microsoft Access overwrites the file specified by \textit{DatabaseFileName}. \texttt{True} to overwrite the existing file. If the file doesn't already exist, this argument is ignored.

\textit{DisconnectAllUsers} Optional \texttt{Variant}. Determines whether Access disconnects any users connected to the current database in order to make the copy. \texttt{True} to disconnect other users before copying the database file.
Remarks

The file name of the copy must have an .mdf extension in order to be recognized as a SQL Server database file.

The method fails and an error occurs if any of the following occurs:

- **DisconnectAllUsers** is True but Access is unable to log off other users.
- The method cancels a save operation by any open design sessions.
- The destination file exists but **OverwriteExistingFile** was not set to True.
- The destination file exists, but is in use by another application.
- Access could not reconnect the original .mdf file.
- The current user for the Access project doesn’t have system administrator privileges for the database server.
Example

This example copies the database connected to the current project to a SQL Server database file. If the file exists already, Access overwrites it, and any other users connected to the database are disconnected before the copy is made.

```vba
DoCmd.CopySQLDatabaseFile
    DatabaseFileName:="C:\Export\Sales.mdf",
    OverwriteExistingFile:=True,
    DisconnectAllUsers:=True
```
CopyObject Method

The CopyObject method carries out the CopyObject action in Visual Basic.

expression.CopyObject(DestinationDatabase, NewName, SourceObjectType, SourceObjectName)

expression Required. An expression that returns one of the objects in the Applies To list.

DestinationDatabase Optional Variant. A string expression that's the valid path and file name for the database you want to copy the object into. To select the current database, leave this argument blank. Note: In a Microsoft Access project (.adp) you must leave the destinationdatabase argument blank. If you execute Visual Basic code containing the CopyObject method in a library database and leave this argument blank, Microsoft Access copies the object into the library database.

NewName Optional Variant. A string expression that's the new name for the object you want to copy. To use the same name if you are copying into another database, leave this argument blank.

SourceObjectType Optional AcObjectType. AcObjectType can be one of these AcObjectType constants.

acDataAccessPage
acDefault default
acDiagram
acForm
acFunction
acMacro
acModule
acQuery
acReport
acServerView
acStoredProcedure
acTable

**Note** When using the `CopyObject` method with a data access page, a copy of the HTML file for the data access page is created in the Default database folder and a link to it is created in the destination database.

**SourceObjectName** Optional **Variant.** A string expression that's the valid name of an object of the type selected by the `sourceobjecttype` argument. If you run Visual Basic code containing the `CopyObject` method in a library database, Microsoft Access looks for the object with this name first in the library database, then in the current database.
Remarks

For more information on how the action and its arguments work, see the action topic.

You must include either the destinationdatabase or newname argument or both for this method.

If you leave the sourceobjecttype and sourceobjectname arguments blank (the default constant, acDefault, is assumed for sourceobjecttype), Microsoft Access copies the object selected in the Database window. To select an object in the Database window, you can use the SelectObject action or SelectObject method with the In Database Window argument set to Yes (True).

If you specify the sourceobjecttype and sourceobjectname arguments but leave either the newname argument or the destinationdatabase argument blank, you must include the newname or destinationdatabase argument's comma. If you leave a trailing argument blank, don't use a comma following the last argument you specify.
Example

The following example uses the `CopyObject` method to copy the Employees table and give it a new name in the current database:

`DoCmd.CopyObject, "Employees Copy", acTable, "Employees"`
CreateAccessProject Method

You can use the CreateAccessProject method to create a new Microsoft Access project (.adp) on disk.

expression.CreateAccessProject(filepath, Connect)

**expression**  Required. An expression that returns one of the objects in the Applies To list.

**filepath**  Required **String**. A string expression that is the name of the new Access project, including the path name and the file name extension. If your network supports it, you can also specify a network path in the following form: `\Server\Share\Folder\Filename .adp`

**Connect**  Optional **Variant**. A string expression that's the valid connection string for the Access project. See the ADO **ConnectionString** property for details about this string.
Remarks

The **CreateAccessProject** method enables you to create a new Access project from within Microsoft Access or another application through Automation, formally called OLE Automation. For example, you can use the **CreateAccessProject** method from Microsoft Excel to create a new Access project on disk. Once you have created an instance of Microsoft Access from another application, you must also create a new Access project.

If the Access project identified by *projname* already exists, an error occurs.

To create and open a new Access project as the current Access project in the Access window, use the **NewAccessProject** method of the **Application** object.

To open an existing Access project as the current Access project in the Access window, use the **OpenAccessProject** method of the **Application** object.
**Example**

The following example creates a Microsoft Access project named "Order Entry.adp" on drive C.

```
Application.CreateAccessProject "C:\Order Entry.adp"
```
CreateAdditionalData Method

Creates an AdditionalData object that can be used to add additional tables and queries to the parent table that is being exported by the ExportXML method. Returns AdditionalData.

expression.CreateAdditionalData()

expression Required. An expression that returns an Application object.
Example

The following example exports the contents of the Customers table in the Northwind Traders sample database, along with the contents of the Orders and Orders Details tables, to an XML data file named Customer Orders.xml.

Sub ExportCustomerOrderData()
    Dim objOrderInfo As AdditionalData

    Set objOrderInfo = Application.CreateAdditionalData()

    ' Add the Orders and Order Details tables to the data to be expo
    objOrderInfo.Add "Orders"
    objOrderInfo.Add "Order Details"

    ' Export the contents of the Customers table. The Orders and Ord
    ' Details tables will be included in the XML file.
    Application.ExportXML ObjectType:=acExportTable, DataSource:="Cu
    DataTarget:="Customer Orders.xml", _
    AdditionalData:=objOrderInfo
End Sub
CreateControl Method

The CreateControl method creates a control on a specified open form. For example, suppose you are building a custom wizard that allows users to easily construct a particular form. You can use the CreateControl method in your wizard to add the appropriate controls to the form.

CreateControl(formname, controltype[, section[, parent[, columnname[, left[, top[, width[, height]]]]]]])

The CreateControl method has the following arguments.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>formname</td>
<td>A string expression identifying the name of the open form or report on which you want to create the control. One of the following intrinsic constants identifying the type of control you want to create. To view these constants and paste them into your code from the Object Browser, click Object Browser on the Visual Basic toolbar, then click Access in the Project/Library box, and click AcControlType in the Classes box.</td>
</tr>
<tr>
<td>controltype</td>
<td>Constant</td>
</tr>
<tr>
<td></td>
<td>Control</td>
</tr>
<tr>
<td>acBoundObjectFrame</td>
<td>Bound object frame</td>
</tr>
<tr>
<td>acCheckBox</td>
<td>Check box</td>
</tr>
<tr>
<td>acComboBox</td>
<td>ComboBox</td>
</tr>
<tr>
<td>acCommandButton</td>
<td>Command button</td>
</tr>
<tr>
<td>acCustomControl</td>
<td>ActiveX control</td>
</tr>
<tr>
<td>acImage</td>
<td>Image</td>
</tr>
</tbody>
</table>
One of the following intrinsic constants identifying the section that will contain the new control. To view these constants and paste them into your code from the Object Browser, click Object Browser on the Visual Basic toolbar, then click Access in the Project/Library box, and click AcSection in the Classes box.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>acDetail</td>
<td>Detail section</td>
</tr>
<tr>
<td>acHeader</td>
<td>Form or report header</td>
</tr>
<tr>
<td>acFooter</td>
<td>Form or report footer</td>
</tr>
</tbody>
</table>
If a report has additional group levels, the header/footer pairs are numbered consecutively, beginning with 9.

A string expression identifying the name of the parent control of an attached control. For controls that have no parent control, use a zero-length string for this argument, or omit it.

The name of the field to which the control will be bound, if it is to be a data-bound control.

If you are creating a control that won't be bound to a field, use a zero-length string for this argument.

Numeric expressions indicating the coordinates for the upper-left corner of the control in twips.
width,  height  Numeric expressions indicating the width and height of the control in twips.
Remarks

You can use the **CreateControl** and **CreateReportControl** methods in a custom wizard to create controls on a form or report. Both methods return a **Control** object.

You can use the **CreateControl** and **CreateReportControl** methods only in **form Design view** or **report Design view**, respectively.

You use the *parent* argument to identify the relationship between a main control and a subordinate control. For example, if a text box has an attached label, the text box is the main (or parent) control and the label is the subordinate (or child) control. When you create the label control, set its *parent* argument to a string identifying the name of the parent control. When you create the text box, set its *parent* argument to a zero-length string.

You also set the *parent* argument when you create check boxes, option buttons, or toggle buttons. An option group is the parent control of any check boxes, option buttons, or toggle buttons that it contains. The only controls that can have a parent control are a label, check box, option button, or toggle button. All of these controls can also be created independently, without a parent control.

Set the *columnname* argument according to the type of control you are creating and whether or not it will be **bound** to a field in a table. The controls that may be bound to a field include the text box, list box, combo box, option group, and bound object frame. Additionally, the toggle button, option button, and check box controls may be bound to a field if they are not contained in an option group.

If you specify the name of a field for the *columnname* argument, you create a control that is bound to that field. All of the control's properties are then automatically set to the settings of any corresponding field properties. For example, the value of the control's **ValidationRule** property will be the same as the value of that property for the field.

**Note** If your wizard creates controls on a new or existing form or report, it must first open the form or report in **Design view**.

To remove a control from a form or report, use the `DeleteControl` and `DeleteReportControl` statements.
Example

The following example first creates a new form based on an Orders table. It then uses the `CreateControl` method to create a text box control and an attached label control on the form.

```vba
Sub NewControls()
    Dim frm As Form
    Dim ctlLabel As Control, ctlText As Control
    Dim intDataX As Integer, intDataY As Integer
    Dim intLabelX As Integer, intLabelY As Integer

    ' Create new form with Orders table as its record source.
    Set frm = CreateForm
    frm.RecordSource = "Orders"
    ' Set positioning values for new controls.
    intLabelX = 100
    intLabelY = 100
    intDataX = 1000
    intDataY = 100
    ' Create unbound default-size text box in detail section.
    Set ctlText = CreateControl(frm.Name, acTextBox, , "", "", _
                               intDataX, intDataY)
    ' Create child label control for text box.
    Set ctlLabel = CreateControl(frm.Name, acLabel, , _
                                  ctlText.Name, "NewLabel", intLabelX, intLabelY)
    ' Restore form.
    DoCmd.Restore
End Sub
```
**CreateEventProc Method**

The `CreateEventProc` method creates an event procedure in a class module. It returns a `Long` value that indicates the line number of the first line of the event procedure. `Long`.

```
expression.CreateEventProc(EventName, ObjectName)
```

**expression**  Required. An expression that returns one of the objects in the Applies To list.

**EventName**  Required String. A string expression that evaluates to the name of an event.

**ObjectName**  Required String. An object that has the event specified by the `eventname` argument. It may be a Form, Report, or Control object, a form or report section, or a class module.
Remarks

The `CreateEventProc` method creates a code stub for an event procedure for the specified object. For example, you can use this method to create a Click event procedure for a command button on a form. Microsoft Access creates the Click event procedure in the module associated with the form that contains the command button.

Once you've created the event procedure code stub by using the `CreateEventProc` method, you can add lines of code to the procedure by using other methods of the `Module` object. For example, you can use the `InsertLines` method to insert a line of code.
Example

The following example creates a new form, adds a command button, and creates a Click event procedure for the command button:

Function ClickEventProc() As Boolean
    Dim frm As Form, ctl As Control, mdl As Module
    Dim lngReturn As Long

    On Error GoTo Error_ClickEventProc
    ' Create new form.
    Set frm = CreateForm
    ' Create command button on form.
    Set ctl = CreateControl(frm.Name, acCommandButton, , , , , _
                             1000, 1000)
    ctl.Caption = "Click here"
    ' Return reference to form module.
    Set mdl = frm.Module
    ' Add event procedure.
    lngReturn = mdl.CreateEventProc("Click", ctl.Name)
    ' Insert text into body of procedure.
    mdl.InsertLines lngReturn + 1, vbTab & "MsgBox ""Way cool!""
    ClickEventProc = True

Exit_ClickEventProc:
    Exit Function

Error_ClickEventProc:
    MsgBox Err & " : " & Err.Description
    ClickEventProc = False
    Resume Exit_ClickEventProc
End Function
CreateForm Method

The **CreateForm** method creates a form and returns a **Form** object.

**CreateForm([database[, formtemplate]])**

The **CreateForm** method has the following arguments.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>database</strong></td>
<td>A string expression identifying the name of the database that contains the form template you want to use to create a form. If you want the current database, omit this argument. If you want to use an open library database, specify the library database with this argument. A string expression identifying the name of the form you want to use as a template to create a new form. If you omit this argument, Microsoft Access bases the new form on the template specified by the Forms/Reports tab of the Options dialog box, available by clicking Options on the Tools menu.</td>
</tr>
<tr>
<td><strong>formtemplate</strong></td>
<td>Microsoft Access bases the new form on the template specified by the Forms/Reports tab of the Options dialog box, available by clicking Options on the Tools menu.</td>
</tr>
</tbody>
</table>
Remarks

You can use the `CreateForm` method when designing a wizard that creates a new form.

The `CreateForm` method opens a new, minimized form in `form Design view`.

If the name you use for the `formtemplate` argument isn't valid, Visual Basic uses the form template specified by the `Form Template` setting on the `Forms/Reports` tab of the `Options` dialog box.
Example

This example creates a new form in the Northwind sample database based on the Customers form, and sets its **RecordSource** property to the Customers table. Run this code from the Northwind sample database.

```vba
Sub NewForm()
    Dim frm As Form
    ' Create form based on Customers form.
    Set frm = CreateForm( , "Customers")
    DoCmd.Restore
    ' Set RecordSource property to Customers table.
    frm.RecordSource = "Customers"
End Sub
```
CreateGroupLevel Method

You can use the **CreateGroupLevel** method to specify a field or **expression** on which to group or sort data in a report. **Long**.

**expression.**CreateGroupLevel(ReportName, Expression, Header, Footer)

**expression** Required. An expression that returns one of the objects in the Applies To list.

**ReportName** Required **String**. A **string expression** identifying the name of the report that will contain the new group level.

**Expression** Required **String**. A string expression identifying the field or expression to sort or group on.

**Header** Required **Integer**. An **Integer** value that indicates a field or expression will have an associated group header. If the **header** argument is **True** (–1), the field or expression will have a group header. If the **header** argument is **False** (0), the field or expression won't. You can create a header by setting the argument to **True**.

**Footer** Required **Integer**. An **Integer** value that indicates a field or expression will have an associated group footer. If the **footer** argument is **True** (–1), the field or expression will have a group footer. If the **footer** argument is **False** (0), the field or expression won't. You can create a footer by setting the argument to **True**.
Remarks

For example, suppose you are building a custom wizard that provides the user with a choice of fields on which to group data when designing a report. Call the `CreateGroupLevel` method from your wizard to create the appropriate groups according to the user's choice.

You can use the `CreateGroupLevel` method when designing a wizard that creates a report with groups or totals. The `CreateGroupLevel` method groups or sorts data on the specified field or expression and creates a header and/or footer for the group level.

The `CreateGroupLevel` method is available only in `report Design view`.

Microsoft Access uses an `array`, the `GroupLevel` property array, to keep track of the group levels created for a report. The `CreateGroupLevel` method adds a new group level to the array, based on the `expression` argument. The `CreateGroupLevel` method then returns an index value that represents the new group level's position in the array. The first field or expression you sort or group on is level 0, the second is level 1, and so on. You can have up to ten group levels in a report (0 to 9).

When you specify that either the `header` or `footer` argument, or both, is `True`, the `GroupHeader` and `GroupFooter` properties in a report are set to `Yes`, and a header and/or footer is created for the group level.

Once a header or footer is created, you can set other `GroupLevel` properties: `GroupOn`, `GroupInterval`, and `KeepTogether`. You can set these properties in Visual Basic or in the report's `Sorting And Grouping` box, available by clicking `Sorting And Grouping` on the `Report Design toolbar`.

**Note** If your wizard creates group levels in a new or existing report, it must open the report in `Design view`.
Example

The following example creates a group level on an OrderDate field on a report called OrderReport. The report on which the group level is to be created must be open in Design view. Since the header and footer arguments are set to True (–1), the method creates both the header and footer for the group level. The header and footer are then sized.

Sub CreateGL()
    Dim varGroupLevel As Variant

    ' Create new group level on OrderDate field.
    varGroupLevel = CreateGroupLevel("OrderReport", "OrderDate", _
        True, True)

    ' Set height of header/footer sections.
    Reports!OrderReport.Section(acGroupLevel1Header).Height = 400
    Reports!OrderReport.Section(acGroupLevel1Footer).Height = 400

End Sub
CreateNewDocument Method

You can use the CreateNewDocument method to create a new document associated with a specified hyperlink.

expression.CreateNewDocument(FileName, EditNow, Overwrite)

expression Required. An expression that returns one of the objects in the Applies To list.

**FileName** Required **String**. A string expression identifying the name and path of the document. The type of document format you want used can be determined by the extension used with the filename to output the data. You can create HTML (*.htm), Microsoft Active Server Pages (*.asp), Microsoft Excel (*.xls), Microsoft IIS (*.htx, *.idc), MS-DOS Text (*.txt), Rich Text Format (*.rtf), or Microsoft Data Access Pages (*.html). Modules can be output only to MS-DOS text format. Data access pages can only be output in HTML format. Microsoft Internet Information Server and Microsoft Active Server formats are available only for tables, queries, and forms. **Note:** If an extension is not provided then the data access page format (.html) is assumed. If a directory is not specified, the default database directory is used. This directory is determined by the setting in Options dialog box.

**EditNow** Required **Boolean**. A Boolean value where True opens the document in design view and False stores the new document in the specified database directory. The default is True.

**Overwrite** Required **Boolean**. A Boolean value where True overwrites an existing document if the filename argument identifies an existing document and False requires that the filename argument specifies a new filename. The default is False.
Remarks

The **CreateNewDocument** method provides a way to programmatically create a document associated with a hyperlink within a control.
Example

The following example utilizes a hyperlink control's **Click** event. This event creates a new file named "Report.txt" when the user clicks the hyperlink control named "GenerateReport" on a form. The new file opened for editing. If a file named "Report.txt" already exists on drive C, it is replaced with this new file.

Private Sub GenerateReport_Click()
    ActiveControl.Hyperlink/CreateNewDocument_  
        "C:\Report.txt", EditNow:=True, Overwrite:=True
End Sub
CreateNewWorkgroupFile Method

Creates a new workgroup file so that a user can securely access a database.

(expression.\textit{CreateNewWorkgroupFile}(*Path*, *Name*, *Company*, *WorkgroupID*, Replace))

\textit{expression} Required. An expression that returns one of the objects in the Applies To list.

\textit{Path} Optional \textit{String}. The path of the new workgroup file. If the path is invalid, an error occurs. Default is an empty string (""").

\textit{Name} Optional \textit{String}. The name of the user creating the file. Default is an empty string (""").

\textit{Company} Optional \textit{String}. The company of the user creating the file. Default is an empty string (""").

\textit{WorkgroupID} Optional \textit{String}. The name of the workgroup. Default is an empty string (""").

\textit{Replace} Optional \textit{Boolean}. Specifies whether to replace the workgroup file in the directory specified by \textit{Path} if it exists already. Default is \textbf{False}. 
Remarks

If a workgroup file already exists in the directory specified by *Path*, and *Replace* is not *True*, an error occurs.
Example

This example creates a new workgroup file in the specified directory with the specified user information. If a workgroup file for this user already exists in the specified directory, Microsoft Access replaces it.

```vbnet
Application.CreateNewWorkgroupFile _
    Path:="C:\Documents and Settings\Wendy Vasse" _
    & "\Application Data\Microsoft\Access", _
    Name:="Wendy Vasse", _
    Company:="Microsoft", _
    Replace:=True
```
CreateReport Method

The CreateReport method creates a report and returns a Report object. For example, suppose you are building a custom wizard to create a sales report. You can use the CreateReport method in your wizard to create a new report based on a specified report template.

expression.CreateReport(Database, ReportTemplate)

expression Required. An expression that returns one of the objects in the Applies To list.

Database Optional Variant. A string expression identifying the name of the database that contains the report template you want to use to create a report. If you want the current database, omit this argument. If you want to use an open library database, specify the library database with this argument.

ReportTemplate Optional Variant. A string expression identifying the name of the report you want to use as a template to create a new report. If you omit this argument, Microsoft Access bases the new report on the template specified by the Forms/Reports tab of the Options dialog box, available by clicking Options on the Tools menu.
Remarks

You can use the `CreateReport` method when designing a wizard that creates a new report.


If the name you use for the `reporttemplate` argument isn't valid, Visual Basic uses the report template specified by the `Report Template` setting on the Forms/Reports tab of the Options dialog box.
Example

The following example creates a report in the current database by using the template specified by the Report Template setting on the Forms/Reports tab of the Options dialog box.

```vba
Sub NormalReport()
    Dim rpt As Report

    Set rpt = CreateReport  ' Create minimized report.
    DoCmd.Restore          ' Restore report.
End Sub
```
CreateReportControl Method

The CreateReportControl method creates a control on a specified open report. For more information, see the CreateControl method.

expression.CreateReportControl(ReportName, ControlType, Section, Parent, ColumnName, Left, Top, Width, Height)

expression Required. An expression that returns one of the objects in the Applies To list.

ReportName Required String. A string expression identifying the name of the open report on which you want to create the control.

ControlType Required AcControlType. The type of control you want to create. AcControlType can be one of these AcControlType constants.

acBoundObjectFrame
acCheckBox
acComboBox
acCommandButton
acCustomControl
acImage
acLabel
acLine
acListBox
acObjectFrame
acOptionButton
acOptionGroup
acPage
acPageBreak
acRectangle
acSubform
acTabCtl
**acTextBox**
**acToggleButton**

**Section**  Optional **AcSection**. The section that will contain the new control.

AcSection can be one of these AcSection constants.

**acDetail**  *default*
**acFooter**
**acGroupLevel1Footer**
**acGroupLevel1Header**
**acGroupLevel2Footer**
**acGroupLevel2Header**
**acHeader**
**acPageFooter**
**acPageHeader**

**Parent**  Optional **Variant**. A string expression identifying the name of the parent control of an attached control. For controls that have no parent control, use a zero-length string for this argument, or omit it.

**ColumnName**  Optional **Variant**. The name of the field to which the control will be bound, if it is to be a data-bound control.

**Left, Top**  Optional **Variant**. Numeric expressions indicating the coordinates for the upper-left corner of the control in twips.

**Width, Height**  Optional **Variant**. Numeric expressions indicating the width and height of the control in twips.
CurrentDb Method

The CurrentDb method returns an object variable of type Database that represents the database currently open in the Microsoft Access window.

expression.CurrentDb

expression Required. An expression that returns one of the objects in the Applies To list.
Remarks

Note In Microsoft Access the **CurrentDb** method establishes a hidden reference to the Microsoft DAO 3.6 Object Library in a Microsoft Access database (.mdb).

In order to manipulate the structure of your database and its data from Visual Basic, you must use **Data Access Objects (DAO)**. The **CurrentDb** method provides a way to access the current database from Visual Basic code without having to know the name of the database. Once you have a variable that points to the current database, you can also access and manipulate other objects and collections in the **DAO hierarchy**.

You can use the **CurrentDb** method to create multiple object variables that refer to the current database. In the following example, the variables `dbsA` and `dbsB` both refer to the current database:

```vba
Dim dbsA As Database, dbsB As Database
Set dbsA = CurrentDb
Set dbsB = CurrentDb
```

Note In previous versions of Microsoft Access, you may have used the syntax `DBEngine.Workspaces(0).Databases(0)` or `DBEngine(0)(0)` to return a pointer to the current database. In Microsoft Access 2000, you should use the **CurrentDb** method instead. The **CurrentDb** method creates another instance of the current database, while the `DBEngine(0)(0)` syntax refers to the open copy of the current database. The **CurrentDb** method enables you to create more than one variable of type **Database** that refers to the current database. Microsoft Access still supports the `DBEngine(0)(0)` syntax, but you should consider making this modification to your code in order to avoid possible conflicts in a **multiuser database**.

If you need to work with another database at the same time that the current database is open in the Microsoft Access window, use the **OpenDatabase** method of a **Workspace** object. The **OpenDatabase** method doesn't actually open the second database in the Microsoft Access window; it simply returns a **Database** variable representing the second database. The following example returns a pointer to the current database and to a database called `Contacts.mdb`:
Dim dbsCurrent As Database, dbsContacts As Database
Set dbsCurrent = CurrentDb
Set dbsContacts = DBEngine.Workspaces(0).OpenDatabase("Contacts.mdb")
CurrentUser Method

You can use the **CurrentUser** method to return the name of the current user of the database. *String.*

`expression.CurrentUser`

*expression* Required. An expression that returns one of the objects in the Applies To list.
Remarks

For example, use the `CurrentUser` method in a procedure that keeps track of the users who modify the database.

The `CurrentUser` method returns a `string` that contains the name of the current user account.

If you haven't established a `security-enabled workgroup`, the `CurrentUser` method returns the name of the default user account, Admin. The Admin user account gives the user full permissions to all `database objects`.

If you have enabled workgroup security, then the `CurrentUser` method returns the name of the current user account. For user accounts other than Admin, you can specify `permissions` that restrict the users' access to database objects.
Example

The following example obtains the name of the current user and displays it in a dialog box.

MsgBox("The current user is: " & CurrentUser)
DAvg Function

You can use the **DAvg** function to calculate the average of a set of values in a specified set of records (a **domain**). Use the **DAvg** function in Visual Basic code or in a **macro**, in a query expression, or in a **calculated control**.

For example, you could use the **DAvg** function in the criteria row of a select query on freight cost to restrict the results to those records where the freight cost exceeds the average. Or you could use an expression including the **DAvg** function in a calculated control and display the average value of previous orders next to the value of a new order.

**DAvg**(expr, domain, [criteria])

The **DAvg** function has the following arguments.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>expr</td>
<td>An expression that identifies the field containing the numeric data you want to average. It can be a string expression identifying a field in a table or query, or it can be an expression that performs a calculation on data in that field. In <em>expr</em>, you can include the name of a field in a table, a control on a form, a constant, or a function. If <em>expr</em> includes a function, it can be either built-in or user-defined, but not another domain aggregate or SQL aggregate function.</td>
</tr>
<tr>
<td>domain</td>
<td>A string expression identifying the set of records that constitutes the domain. It can be a table name or a query name for a query that does not require a parameter.</td>
</tr>
<tr>
<td>criteria</td>
<td>An optional string expression used to restrict the range of data on which the <strong>DAvg</strong> function is performed. For example, <em>criteria</em> is often equivalent to the WHERE clause in an SQL expression, without the word WHERE. If <em>criteria</em> is omitted, the <strong>DAvg</strong> function evaluates <em>expr</em> against the entire domain. Any field that is included in <em>criteria</em> must also be a field in <em>domain</em>; otherwise the <strong>DAvg</strong> function returns a <strong>Null</strong>.</td>
</tr>
</tbody>
</table>
Remarks

Records containing Null values aren't included in the calculation of the average.

Whether you use the DAvg function in a macro or module, in a query expression, or in a calculated control, you must construct the criteria argument carefully to ensure that it will be evaluated correctly.

You can use the DAvg function to specify criteria in the Criteria row of a query. For example, suppose you want to view a list of all products ordered in quantities above the average order quantity. You could create a query on the Orders, Order Details, and Products tables, and include the Product Name field and the Quantity field, with the following expression in the Criteria row beneath the Quantity field:

>DAvg("[Quantity]", "Orders")

You can also use the DAvg function within a calculated field expression in a query, or in the Update To row of an update query.

Note  You can use either the DAvg or Avg function in a calculated field expression in a totals query. If you use the DAvg function, values are averaged before the data is grouped. If you use the Avg function, the data is grouped before values in the field expression are averaged.

Use the DAvg function in a calculated control when you need to specify criteria to restrict the range of data on which the DAvg function is performed. For example, to display the average cost of freight for shipments sent to California, set the ControlSource property of a text box to the following expression:

=DAvg("[Freight]", "Orders", ";ShipRegion" = 'CA'")

If you simply want to average all records in domain, use the Avg function.

You can use the DAvg function in a module or macro or in a calculated control on a form if a field that you need to display isn't in the record source on which your form is based. For example, suppose you have a form based on the Orders
table, and you want to include the Quantity field from the Order Details table in order to display the average number of items ordered by a particular customer. You can use the **DAvg** function to perform this calculation and display the data on your form.

**Tips**

- If you use the **DAvg** function in a calculated control, you may want to place the control on the form header or footer so that the value for this control is not recalculated each time you move to a new record.
- If the data type of the field from which `expr` is derived is a number, the **DAvg** function returns a **Double** data type. If you use the **DAvg** function in a calculated control, include a data type conversion function in the expression to improve performance.
- Although you can use the **DAvg** function to determine the average of values in a field in a **foreign table**, it may be more efficient to create a query that contains all of the fields that you need and then base your form or report on that query.

**Note**  Unsaved changes to records in **domain** aren't included when you use this function. If you want the **DAvg** function to be based on the changed values, you must first save the changes by clicking **Save Record** on the **Records** menu, moving the focus to another record, or by using the **Update** method.
Example
The following function returns the average freight cost for orders shipped on or
after a given date. The domain is an Orders table. The criteria argument restricts
the resulting set of records based on the given country and ship date. Note that
the keyword AND is included in the string to separate the multiple fields in the
criteria argument. All records included in the DAvg function calculation will
have both of these criteria.
Public Function AvgFreightCost(ByVal strCountry As String, _
ByVal dteShipDate As Date) As Double
AvgFreightCost = DAvg("[Freight]", "Orders", _
"[ShipCountry] = '" & strCountry & _
"'AND [ShippedDate] >= #" & dteShipDate & "#")
End Function

To call the function, use the following line of code in the Immediate window:
:AvgFreightCost "UK", #1/1/96#


DCount Function

You can use the DCount function to determine the number of records that are in a specified set of records (a domain). Use the DCount function in Visual Basic, a macro, a query expression, or a calculated control.

For example, you could use the DCount function in a module to return the number of records in an Orders table that correspond to orders placed on a particular date.

DCount(expr, domain, [criteria])

The DCount function has the following arguments.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>expr</td>
<td>An expression that identifies the field for which you want to count records. It can be a string expression identifying a field in a table or query, or it can be an expression that performs a calculation on data in that field. In expr, you can include the name of a field in a table, a control on a form, a constant, or a function. If expr includes a function, it can be either built-in or user-defined, but not another domain aggregate or SQL aggregate function.</td>
</tr>
<tr>
<td>domain</td>
<td>A string expression identifying the set of records that constitutes the domain. It can be a table name or a query name for a query that does not require a parameter. An optional string expression used to restrict the range of data on which the DCount function is performed. For example, criteria is often equivalent to the WHERE clause in an SQL expression, without the word WHERE. If criteria is omitted, the DCount function evaluates expr against the entire domain. Any field that is included in criteria must also be a field in domain; otherwise the DCount function returns a Null.</td>
</tr>
<tr>
<td>criteria</td>
<td></td>
</tr>
</tbody>
</table>


Remarks

Use the **DCount** function to count the number of records in a domain when you don't need to know their particular values. Although the *expr* argument can perform a calculation on a field, the **DCount** function simply tallies the number of records. The value of any calculation performed by *expr* is unavailable.

Use the **DCount** function in a calculated control when you need to specify criteria to restrict the range of data on which the function is performed. For example, to display the number of orders to be shipped to California, set the **ControlSource** property of a text box to the following expression:

=**DCount**("[OrderID]", "Orders", ";ShipRegion; = 'CA'")

If you simply want to count all records in *domain* without specifying any restrictions, use the **Count** function.

**Tip**

The **Count** function has been optimized to speed counting of records in queries. Use the **Count** function in a query expression instead of the **DCount** function, and set optional criteria to enforce any restrictions on the results. Use the **DCount** function when you must count records in a domain from within a code module or macro, or in a calculated control.

You can use the **DCount** function to count the number of records containing a particular field that isn't in the record source on which your form or report is based. For example, you could display the number of orders in the Orders table in a calculated control on a form based on the Products table.

The **DCount** function doesn't count records that contain **Null** values in the field referenced by *expr* unless *expr* is the asterisk (*) wildcard character. If you use an asterisk, the **DCount** function calculates the total number of records, including those that contain **Null** fields. The following example calculates the number of records in an Orders table.

```
intX = DCount("*", "Orders")
```
If *domain* is a table with a **primary key**, you can also count the total number of records by setting *expr* to the primary key field, since there will never be a **Null** in the primary key field.

If *expr* identifies multiple fields, separate the field names with a concatenation operator, either an ampersand (&) or the addition operator (+). If you use an ampersand to separate the fields, the **DCount** function returns the number of records containing data in any of the listed fields. If you use the addition operator, the **DCount** function returns only the number of records containing data in all of the listed fields. The following example demonstrates the effects of each operator when used with a field that contains data in all records (ShipName) and a field that contains no data (ShipRegion).

```plaintext
intValueW = DCount("[ShipName]", "Orders")
intValueX = DCount("[ShipRegion]", "Orders")
intValueY = DCount("[ShipName] + [ShipRegion]", "Orders")
intValueZ = DCount("[ShipName] & [ShipRegion]", "Orders")
```

**Note**  The ampersand is the preferred operator for performing string concatenation. You should avoid using the addition operator for anything other than numeric addition, unless you specifically wish to propagate **Nulls** through an expression.

Unsaved changes to records in *domain* aren't included when you use this function. If you want the **DCount** function to be based on the changed values, you must first save the changes by clicking **Save Record** on the **Records** menu, moving the focus to another record, or by using the **Update** method.
Example

The following function returns the number of orders shipped to a specified country after a specified ship date. The domain is an Orders table.

Public Function OrdersCount(ByVal strCountry As String, ByVal dteShipDate As Date) As Integer

    OrdersCount = DCount("[ShippedDate]", "Orders", "[ShipCountry] = "'" & strCountry & "' AND [ShippedDate] > #" & dteShipDate & ":#")

End Function

To call the function, use the following line of code in the Immediate window:

:OrdersCount "UK", #1/1/96#
DDEExecute Method

You can use the **DDEExecute** statement to send a command from a client application to a server application over an open **dynamic data exchange (DDE)** channel.

\[\text{expression}.\text{DDEExecute}(\text{ChanNum}, \text{Command})\]

**expression** Required. An expression that returns one of the objects in the Applies To list.

**ChanNum** Required **Variant**. A **channel number**, the long integer returned by the **DDEInitiate** function.

**Command** Required **String**. A **string expression** specifying a command recognized by the server application. Check the server application's documentation for a list of these commands.
Remarks

For example, suppose you've opened a DDE channel in Microsoft Access to transfer text data from a Microsoft Excel spreadsheet into a Microsoft Access database. Use the `DDEExecute` statement to send the New command to Microsoft Excel to specify that you wish to open a new spreadsheet. In this example, Microsoft Access acts as the client application, and Microsoft Excel acts as the server application.

The value of the `command` argument depends on the application and topic specified when the channel indicated by the `channum` argument is opened. An error occurs if the `channum` argument isn't an integer corresponding to an open channel or if the other application can't carry out the specified command.

From Visual Basic, you can use the `DDEExecute` statement only to send commands to another application. For information on sending commands to Microsoft Access from another application, see Use Microsoft Access as a DDE Server.

**Tip**

If you need to manipulate another application's objects from Microsoft Access, you may want to consider using Automation.
**DDEInitiate Function**

You can use the **DDEInitiate** function to begin a dynamic data exchange (DDE) conversation with another application. The **DDEInitiate** function opens a DDE channel for transfer of data between a DDE server and client application.

For example, if you wish to transfer data from a Microsoft Excel spreadsheet to a Microsoft Access database, you can use the **DDEInitiate** function to open a channel between the two applications. In this example, Microsoft Access acts as the client application and Microsoft Excel acts as the server application.

**DDEInitiate(application, topic)**

The **DDEInitiate** function has the following arguments.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>application</td>
<td>A string expression identifying an application that can participate in a DDE conversation. Usually, the <em>application</em> argument is the name of an .exe file (without the .exe extension) for a Microsoft Windows–based application, such as Microsoft Excel.</td>
</tr>
<tr>
<td>topic</td>
<td>A string expression that is the name of a <em>topic</em> recognized by the <em>application</em> argument. Check the application's documentation for a list of topics.</td>
</tr>
</tbody>
</table>
Remarks

If successful, the **DDEInitiate** function begins a DDE conversation with the application and topic specified by the *application* and *topic* arguments, and then returns a **Long** integer value. This return value represents a unique **channel number** identifying a channel through which data transfer can take place. This channel number is subsequently used with other DDE functions and statements.

If the application isn't already running or if it's running but doesn't recognize the *topic* argument or doesn't support DDE, the **DDEInitiate** function returns a **run-time error**.

The value of the *topic* argument depends on the application specified by the *application* argument. For applications that use documents or data files, valid topic names often include the names of those files.

**Note** The maximum number of channels that can be open simultaneously is determined by Microsoft Windows and your computer's memory and resources. If you aren't using a channel, you should conserve resources by terminating it with a **DDETerminate** or **DDETerminateAll** statement.

**Tip**

If you need to manipulate another application's objects from Microsoft Access, you may want to consider using Automation.
DDEPoke Method

You can use the **DDEPoke** statement to supply text data from a client application to a server application over an open **dynamic data exchange (DDE)** channel.

\[ \text{expression.DDEPoke(ChanNum, Item, Data)} \]

*expression* Required. An expression that returns one of the objects in the Applies To list.

**ChanNum** Required **Variant**. A channel number, an integer returned by the **DDEInitiate** function.

**Item** Required **String**. A string expression that's the name of a data item recognized by the application specified by the **DDEInitiate** function. Check the application's documentation for a list of possible items.

**Data** Required **String**. A string containing the data to be supplied to the other application.
Remarks

For example, if you have an open DDE channel between Microsoft Access and Microsoft Excel, you can use the **DDEpok**e statement to transfer text from a Microsoft Access database to a Microsoft Excel spreadsheet. In this example, Microsoft Access acts as the client application, and Microsoft Excel acts as the server application.

The value of the *item* argument depends on the application and *topic* specified when the channel indicated by the *channum* argument is opened. For example, the *item* argument may be a range of cells in a Microsoft Excel spreadsheet.

The string contained in the *data* argument must be an alphanumeric text string. No other formats are supported. For example, the *data* argument could be a number to fill a cell in a specified range in an Excel worksheet.

If the *channum* argument isn't an integer corresponding to an open channel or if the other application doesn't recognize or accept the specified data, a run-time error occurs.

**Tip**

If you need to manipulate another application's objects from Microsoft Access, you may want to consider using Automation.
You can use the **DDERequest** function over an open **dynamic data exchange (DDE)** channel to request an item of information from a DDE server application.

For example, if you have an open **DDE channel** between Microsoft Access and Microsoft Excel, you can use the **DDERequest** function to transfer text from a Microsoft Excel spreadsheet to a Microsoft Access database.

**DDERequest**(channum, item)

The **DDERequest** function has the following arguments.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>channum</td>
<td>A <strong>channel number</strong>, the integer returned by the <strong>DDEInitiate</strong> function. A <strong>string expression</strong> that's the name of a <strong>data item</strong> recognized by the application specified by the <strong>DDEInitiate</strong> function. Check the application's documentation for a list of possible items.</td>
</tr>
</tbody>
</table>
Remarks

The `channum` argument specifies the channel number of the desired DDE conversation, and the `item` argument identifies which data should be retrieved from the server application. The value of the `item` argument depends on the application and `topic` specified when the channel indicated by the `channum` argument is opened. For example, the `item` argument may be a range of cells in a Microsoft Excel spreadsheet.

The **DDERequest** function returns a **Variant** as a **string** containing the requested information if the request was successful.

The data is requested in alphanumeric text format. Graphics or text in any other format can't be transferred.

If the `channum` argument isn't an integer corresponding to an open channel, or if the data requested can't be transferred, a **run-time error** occurs.

**Tip**

If you need to manipulate another application's objects from Microsoft Access, you may want to consider using **Automation**.
DDETerminate Method

You can use the DDETerminate statement to close a specified dynamic data exchange (DDE) channel.

expression.DDETerminate(ChanNum)

expression Required. An expression that returns one of the objects in the Applies To list.

ChanNum Required Variant. A channel number to close, refers to a channel opened by the DDEInitiate function.
Remarks

For example, if you've opened a DDE channel to transfer data between Microsoft Excel and Microsoft Access, you can use the **DDETerminate** statement to close that channel once the transfer is complete.

If the *channum* argument isn't an integer corresponding to an open channel, a run-time error occurs.

Once a channel is closed, any subsequent DDE functions or statements performed on that channel cause a run-time error.

The **DDETerminate** statement has no effect on active DDE link expressions in fields on forms or reports.

**Tip**

If you need to manipulate another application's objects from Microsoft Access, you may want to consider using Automation.
DDETerminateAll Method

You can use the `DDETerminateAll` statement to close all open dynamic data exchange (DDE) channels.

`expression.DDETerminateAll`

`expression`   Required. An expression that returns one of the objects in the Applies To list.
Remarks

For example, suppose you've opened two DDE channels between Microsoft Excel and Microsoft Access, one to retrieve system information about Microsoft Excel and one to transfer data. You can use the DDETerminateAll statement to close both channels simultaneously.

Using the DDETerminateAll statement is equivalent to executing a DDETerminate statement for each open channel number. Like the DDETerminate statement, the DDETerminateAll statement has no effect on active DDE link expressions in fields on forms or reports.

If there are no DDE channels open, the DDETerminateAll statement runs without causing a run-time error.

Tips

- If you interrupt a procedure that performs DDE, you may inadvertently leave channels open. To avoid exhausting system resources, use the DDETerminateAll statement in your code or from the Immediate (lower) pane of the Debug window while debugging code that performs DDE.

- If you need to manipulate another application's objects from Microsoft Access, you may want to consider using Automation.
DefaultWorkspaceClone Method

You can use the `DefaultWorkspaceClone` method to create a new `Workspace` object without requiring the user to log on again. For example, if you need to conduct two sets of transactions simultaneously in separate workspaces, you can use the `DefaultWorkspaceClone` method to create a second `Workspace` object with the same user name and password without prompting the user for this information again.

```
expression.DefaultWorkspaceClone
```

`expression`  Required. An expression that returns one of the objects in the Applies To list.
Remarks

Note  In Microsoft Access, the DefaultWorkspaceClone method is included in this version of Microsoft Access only for compatibility with previous versions using Data Access Object (DAO) language.

The DefaultWorkspaceClone method creates a clone of the default Workspace object in Microsoft Access. The properties of the Workspace object clone have settings identical to those of the default Workspace object, except for the Name property setting. For the default Workspace object, the value of the Name property is always #Default Workspace#. For the cloned Workspace object, it is #CloneAccess#.

The UserName property of the default Workspace object indicates the name under which the current user logged on. The Workspace object clone is equivalent to the Workspace object that would be created if the same user logged on again with the same password.
Delete Method

Deletes the specified object.

expression.Delete

expression Required. An expression that returns one of the objects in the Applies To list.
DeleteControl Method

The **DeleteControl** method deletes a specified control from a form.

`expression.DeleteControl(FormName, ControlName)`

*expression*  Required. An expression that returns one of the objects in the Applies To list.

*FormName*  Required **String**. A string expression identifying the name of the form or report containing the control you want to delete.

*ControlName*  Required **String**. A string expression identifying the name of the control you want to delete.
Remarks

For example, suppose you have a procedure that must be run the first time each user logs onto your database. You can set the OnClick property of a button on the form to this procedure. Once the user has logged on and run the procedure, you can use the DeleteControl method to dynamically remove the command button from the form.

The DeleteControl method is available only in form Design view or report Design view, respectively.

Note If you are building a wizard that deletes a control from a form or report, your wizard must open the form or report in Design view before it can delete the control.
Example

The following example creates a form with a command button and displays a message that asks if the user wants to delete the command button. If the user clicks Yes, the command button is deleted.

Sub DeleteCommandButton()
    Dim frm As Form, ctlNew As Control
    Dim strMsg As String, intResponse As Integer, 
        intDialog As Integer

    ' Create new form and get pointer to it.
    Set frm = CreateForm
    ' Create new command button.
    Set ctlNew = CreateControl(frm.Name, acCommandButton)
    ' Restore form.
    DoCmd.Restore
    ' Set caption.
    ctlNew.Caption = "New Command Button"
    ' Size control.
    ctlNew.SizeToFit
    ' Prompt user to delete control.
    strMsg = "About to delete " & ctlNew.Name & ". Continue?"
    ' Define buttons to be displayed in dialog box.
    intDialog = vbYesNo + vbCritical + vbDefaultButton2
    intResponse = MsgBox(strMsg, intDialog)
    If intResponse = vbYes Then
        ' Delete control.
        DeleteControl frm.Name, ctlNew.Name
    End If
End Sub
DeleteLines Method

The **DeleteLines** method deletes lines from a [standard module](#) or a [class module](#).

`expression.DeleteLines(StartLine, Count)`

*expression* Required. An expression that returns one of the objects in the Applies To list.

*StartLine* Required [Long](#). A [Long](#) value that specifies the number of the line from which to begin deleting.

*Count* Required [Long](#). A [Long](#) value that specifies the number of lines to delete.
Remarks

Lines in a module are numbered beginning with one. To determine the number of lines in a module, use the `CountOfLines` property.

To replace one line with another line, use the `ReplaceLine` method.
Example

The following example deletes a specified line from a module.

Function DeleteWholeLine(strModuleName, strText As String) _
    As Boolean
    Dim mdl As Module, lngNumLines As Long
    Dim lngSLine As Long, lngSCol As Long
    Dim lngELine As Long, lngECol As Long
    Dim strTemp As String
    On Error GoTo Error_DeleteWholeLine
    DoCmd.OpenModule strModuleName
    Set mdl = Modules(strModuleName)
    If mdl.Find(strText, lngSLine, lngSCol, lngELine, lngECol) Then
        lngNumLines = Abs(lngELine - lngSLine) + 1
        strTemp = LTrim$(mdl.Lines(lngSLine, lngNumLines))
        strTemp = RTrim$(strTemp)
        If strTemp = strText Then
            mdl.DeleteLines lngSLine, lngNumLines
        Else
            MsgBox "Line contains text in addition to " _
                & strText & "."
    End If
    Else
        MsgBox "Text " & strText & " not found."
    End If
    DeleteWholeLine = True
End Function

Exit_DeleteWholeLine:
    Exit Function
Error_DeleteWholeLine:
    MsgBox Err & ":" & Err.Description
    DeleteWholeLine = False
    Resume Exit_DeleteWholeLine
End Function

You could call this function from a procedure such as the following, which searches the module Module1 for a constant declaration and deletes it.

Sub DeletePiConst()
    If DeleteWholeLine("Module1", "Const conPi = 3.14") Then
Debug.Print "Constant declaration deleted successfully."
Else
    Debug.Print "Constant declaration not deleted."
End If
End Sub
DeleteObject Method

The **DeleteObject** method carries out the **DeleteObject** action in Visual Basic.

*expression*.DeleteObject(*ObjectType, ObjectName*)

*expression*  Required. An expression that returns one of the objects in the Applies To list.

*ObjectType*  Optional **AcObjectType**.

AcObjectType can be one of these AcObjectType constants.

acDataAccessPage
acDefault  *default*
acDiagram
acForm
acFunction
acMacro
acModule
acQuery
acReport
acServerView
acStoredProcedure
acTable

*ObjectName*  Optional Variant. A **string expression** that's the valid name of an object of the type selected by the *objecttype* argument. If you run Visual Basic code containing the **DeleteObject** method in a library database, Microsoft Access looks for the object with this name first in the library database, then in the current database.
Remarks

For more information on how the action and its arguments work, see the action topic.

If you leave the `objectype` and `objectname` arguments blank (the default constant, `acDefault`, is assumed for `objectype`), Microsoft Access deletes the object selected in the Database window. To select an object in the Database window, you can use the SelectObject action or `SelectObject` method with the In Database Window argument set to Yes (True).
Example

The following example deletes the specified table:

DoCmd.DeleteObject acTable, "Former Employees Table"
DeleteReportControl Method

The **DeleteReportControl** method deletes a specified control from a report.

*expression*.DeleteReportControl(*ReportName*, *ControlName*)

*expression*    Required. An expression that returns one of the objects in the Applies To list.

*ReportName*    Required **String**. A **string expression** identifying the name of the form or report containing the control you want to delete.

*ControlName*    Required **String**. A string expression identifying the name of the control you want to delete.
Remarks

The **DeleteReportControl** method is available only in form **Design view** or report **Design view**, respectively.

**Note** If you are building a wizard that deletes a control from a form or report, your wizard must open the form or report in **Design view** before it can delete the control.
Example

The following example creates a form with a command button and displays a message that asks if the user wants to delete the command button. If the user clicks Yes, the command button is deleted.

Sub DeleteCommandButton()
    Dim frm As Form, ctlNew As Control
    Dim strMsg As String, intResponse As Integer,
        intDialog As Integer

    ' Create new form and get pointer to it.
    Set frm = CreateForm
    ' Create new command button.
    Set ctlNew = CreateControl(frm.Name, acCommandButton)
    ' Restore form.
    DoCmd.Restore
    ' Set caption.
    ctlNew.Caption = "New Command Button"
    ' Size control.
    ctlNew.SizeToFit
    ' Prompt user to delete control.
    strMsg = "About to delete " & ctlNew.Name & ". Continue?"
    ' Define buttons to be displayed in dialog box.
    intDialog = vbYesNo + vbCritical + vbDefaultButton2
    intResponse = MsgBox(strMsg, intDialog)
    If intResponse = vbYes Then
        ' Delete control.
        DeleteControl frm.Name, ctlNew.Name
    End If
End Sub
DFirst, DLast Functions

You can use the **DFirst** and **DLast** functions to return a random record from a particular field in a table or query when you simply need any value from that field. Use the **DFirst** and **DLast** functions in a **macro**, module, query expression, or **calculated control** on a form or report.

**DFirst**(expr, domain, [criteria])

**DLast**(expr, domain, [criteria])

The **DFirst** and **DLast** functions have the following arguments:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>expr</td>
<td>An expression that identifies the field from which you want to find the first or last value. It can be either a string expression identifying a field in a table or query, or an expression that performs a calculation on data in that field. In expr, you can include the name of a field in a table, a control on a form, a constant, or a function. If expr includes a function, it can be either built-in or user-defined, but not another domain aggregate or SQL aggregate function.</td>
</tr>
<tr>
<td>domain</td>
<td>A string expression identifying the set of records that constitutes the domain.</td>
</tr>
<tr>
<td>criteria</td>
<td>An optional string expression used to restrict the range of data on which the <strong>DFirst</strong> or <strong>DLast</strong> function is performed. For example, <strong>criteria</strong> is often equivalent to the WHERE clause in an SQL expression, without the word WHERE. If <strong>criteria</strong> is omitted, the <strong>DFirst</strong> and <strong>DLast</strong> functions evaluate <em>expr</em> against the entire domain. Any field that is included in <strong>criteria</strong> must also be a field in <strong>domain</strong>; otherwise, the <strong>DFirst</strong> and <strong>DLast</strong> functions return a <strong>Null</strong>.</td>
</tr>
</tbody>
</table>
Remarks

**Note**  If you want to return the first or last record in a set of records (a **domain**), you should create a query sorted as either ascending or descending and set the **TopValues** property to 1. For more information, see the **TopValues** property topic. From Visual Basic, you can also create an ADO **Recordset** object and use the **MoveFirst** or **MoveLast** method to return the first or last record in a set of records.
DLookup Function

You can use the DLookup function to get the value of a particular field from a specified set of records (a domain). Use the DLookup function in Visual Basic, a macro, a query expression, or a calculated control on a form or report.

You can use the DLookup function to display the value of a field that isn't in the record source for your form or report. For example, suppose you have a form based on an Order Details table. The form displays the OrderID, ProductID, UnitPrice, Quantity, and Discount fields. However, the ProductName field is in another table, the Products table. You could use the DLookup function in a calculated control to display the ProductName on the same form.

DLookup(expr, domain, [criteria])

The DLookup function has the following arguments.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>expr</td>
<td>An expression that identifies the field whose value you want to return. It can be a string expression identifying a field in a table or query, or it can be an expression that performs a calculation on data in that field. In expr, you can include the name of a field in a table, a control on a form, a constant, or a function. If expr includes a function, it can be either built-in or user-defined, but not another domain aggregate or SQL aggregate function.</td>
</tr>
<tr>
<td>domain</td>
<td>A string expression identifying the set of records that constitutes the domain. It can be a table name or a query name for a query that does not require a parameter. An optional string expression used to restrict the range of data on which the DLookup function is performed. For example, criteria is often equivalent to the WHERE clause in an SQL expression, without the word WHERE. If criteria is omitted, the DLookup function evaluates expr against the entire domain. Any field that is included in criteria must also be a field in domain; otherwise, the DLookup function returns a Null.</td>
</tr>
<tr>
<td>criteria</td>
<td></td>
</tr>
</tbody>
</table>

Null
Remarks

The **DLookup** function returns a single field value based on the information specified in **criteria**. Although **criteria** is an optional argument, if you don't supply a value for **criteria**, the **DLookup** function returns a random value in the domain.

If no record satisfies **criteria** or if **domain** contains no records, the **DLookup** function returns a **Null**.

If more than one field meets **criteria**, the **DLookup** function returns the first occurrence. You should specify criteria that will ensure that the field value returned by the **DLookup** function is unique. You may want to use a **primary key** value for your criteria, such as **[EmployeeID]** in the following example, to ensure that the **DLookup** function returns a unique value:

```vba
Dim varX As Variant
varX = DLookup("[LastName]", "Employees", "[EmployeeID] = 1")
```

Whether you use the **DLookup** function in a macro or module, a query expression, or a calculated control, you must construct the **criteria** argument carefully to ensure that it will be evaluated correctly.

You can use the **DLookup** function to specify criteria in the Criteria row of a query, within a calculated field expression in a query, or in the Update To row in an **update query**.

You can also use the **DLookup** function in an expression in a calculated control on a form or report if the field that you need to display isn't in the record source on which your form or report is based. For example, suppose you have an Order Details form based on an Order Details table with a text box called **ProductID** that displays the **ProductID** field. To look up **ProductName** from a Products table based on the value in the text box, you could create another text box and set its **ControlSource** property to the following expression:

```vba
=VLookup("[ProductName]", "Products", "[ProductID] =" _ 
& Forms![Order Details]!ProductID)
```
Tips

- Although you can use the `DLookup` function to display a value from a field in a foreign table, it may be more efficient to create a query that contains the fields that you need from both tables and then to base your form or report on that query.
- You can also use the Lookup Wizard to find values in a foreign table.

Note  Unsaved changes to records in domain aren't included when you use this function. If you want the `DLookup` function to be based on the changed values, you must first save the changes by clicking `Save Record` on the `Records` menu, moving the focus to another record, or by using the `Update` method.
Example

The following example returns name information from the CompanyName field of the record satisfying criteria. The domain is a Shippers table. The criteria argument restricts the resulting set of records to those for which ShipperID equals 1.

Dim varX As Variant
varX = DLookup("[CompanyName]", "Shippers", "[ShipperID] = 1")

The next example from the Shippers table uses the form control ShipperID to provide criteria for the DLookup function. Note that the reference to the control isn't included in the quotation marks that denote the strings. This ensures that each time the DLookup function is called, Microsoft Access will obtain the current value from the control.

Dim varX As Variant
varX = DLookup("[CompanyName]", "Shippers", 
"[ShipperID] = " _ & Forms!Shippers!ShipperID)

The next example uses a variable, intSearch, to get the value.

Dim intSearch As Integer
Dim varX As Variant
intSearch = 1
varX = DLookup("[CompanyName]", "Shippers", _
"[ShipperID] = " & intSearch)
DMin, DMax Functions

You can use the **DMin** and **DMax** functions to determine the minimum and maximum values in a specified set of records (a domain). Use the **DMin** and **DMax** functions in Visual Basic, a **macro**, a query expression, or a **calculated control**.

For example, you could use the **DMin** and **DMax** functions in calculated controls on a report to display the smallest and largest order amounts for a particular customer. Or you could use the **DMin** function in a query expression to display all orders with a discount greater than the minimum possible discount.

**DMin**(expr, domain, [criteria])

**DMax**(expr, domain, [criteria])

The **DMin** and **DMax** functions have the following arguments.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>expr</td>
<td>An expression that identifies the field for which you want to find the minimum or maximum value. It can be a <strong>string expression</strong> identifying a field in a table or query, or it can be an expression that performs a <strong>calculation on data in that field</strong>. In expr, you can include the name of a field in a table, a control on a form, a constant, or a function. If expr includes a function, it can be either built-in or user-defined, but not another domain aggregate or SQL aggregate function.</td>
</tr>
<tr>
<td>domain</td>
<td>A string expression identifying the set of records that constitutes the domain. It can be a table name or a query name for a query that does not require a parameter. An optional string expression used to restrict the range of data on which the <strong>DMin</strong> or <strong>DMax</strong> function is performed. For example, <strong>criteria</strong> is often equivalent to the WHERE clause in an SQL expression, without the word WHERE. If <strong>criteria</strong> is omitted, the <strong>DMin</strong> and <strong>DMax</strong> functions evaluate <strong>expr</strong> against the entire domain. Any field that is included in <strong>criteria</strong> must also be a field in <strong>domain</strong>,</td>
</tr>
<tr>
<td>criteria</td>
<td></td>
</tr>
</tbody>
</table>

Any field that is included in **criteria** must also be a field in **domain**,
otherwise the \texttt{DMin} and \texttt{DMax} functions returns a \texttt{Null}. 
Remarks

The **DMin** and **DMax** functions return the minimum and maximum values that satisfy *criteria*. If *expr* identifies numeric data, the **DMin** and **DMax** functions return numeric values. If *expr* identifies string data, they return the string that is first or last alphabetically.

The **DMin** and **DMax** functions ignore **Null** values in the field referenced by *expr*. However, if no record satisfies *criteria* or if *domain* contains no records, the **DMin** and **DMax** functions return a **Null**.

Whether you use the **DMin** or **DMax** function in a macro, module, query expression, or calculated control, you must construct the *criteria* argument carefully to ensure that it will be evaluated correctly.

You can use the **DMin** and **DMax** function to specify criteria in the Criteria row of a query, in a calculated field expression in a query, or in the Update To row of an *update query*.

**Note**  You can use the **DMin** and **DMax** functions or the **Min** and **Max** functions in a calculated field expression of a *totals query*. If you use the **DMin** or **DMax** function, values are evaluated before the data is grouped. If you use the **Min** or **Max** function, the data is grouped before values in the field expression are evaluated.

Use the **DMin** or **DMax** function in a calculated control when you need to specify criteria to restrict the range of data on which the function is performed. For example, to display the maximum freight charged for an order shipped to California, set the **ControlSource** property of a text box to the following expression:

\[
=\text{DMax("[Freight]", "Orders", "[ShipRegion] = 'CA'"})
\]

If you simply want to find the minimum or maximum value of all records in *domain*, use the **Min** or **Max** function.

You can use the **DMin** or **DMax** function in a module or macro or in a calculated control on a form if the field that you need to display is not in the
record source on which your form is based.

Tip

Although you can use the $DMin$ or $DMax$ function to find the minimum or maximum value from a field in a foreign table, it may be more efficient to create a query that contains the fields that you need from both tables and base your form or report on that query.

Note  Unsaved changes to records in domain aren't included when you use these functions. If you want the $DMax$ or $DMin$ function to be based on the changed values, you must first save the changes by clicking Save Record on the Records menu, moving the focus to another record, or by using the Update method.
Example

The following example returns the lowest and highest values from the Freight field for orders shipped to the United Kingdom. The domain is an Orders table. The criteria argument restricts the resulting set of records to those for which ShipCountry equals UK.

```vba
Dim curX As Currency
Dim curY As Currency

curX = DMin("[Freight]", "Orders", "[ShipCountry] = 'UK'")
curY = DMax("[Freight]", "Orders", "[ShipCountry] = 'UK'")
```

In the next example, the criteria argument includes the current value of a text box called OrderDate. The text box is bound to an OrderDate field in an Orders table. Note that the reference to the control isn't included in the double quotation marks (") that denote the strings. This ensures that each time the DMax function is called, Microsoft Access obtains the current value from the control.

```vba
Dim curX As Currency

curX = DMax("[Freight]", "Orders", "[OrderDate] = #" & Forms!Orders!OrderDate & "#")
```

In the next example, the criteria expression includes a variable, dteOrderDate. Note that number signs (#) are included in the string expression, so that when the strings are concatenated, they will enclose the date.

```vba
Dim dteOrderDate As Date
Dim curX As Currency

dteOrderDate = #03/30/2000#
curX = DMin("[Freight]", "Orders", "[OrderDate] = #" & dteOrderDate & "#")
```
DoMenuItem Method

Displays the appropriate menu or toolbar command for Microsoft Access.

expression. DoMenuItem( MenuBar, MenuName, Command, Subcommand, Version )

expression Required. An expression that returns one of the objects in the Applies To list.

MenuBar Required Variant. Use the intrinsic constant acFormBar for the menu bar in Form view. For other views, use the number of the view in the menu bar argument list, as shown in the Macro window in previous versions of Microsoft Access (count down the list, starting from 0).

MenuName Required Variant. You can use one of the following intrinsic constants.

Intrinsic constants:
acFile
acEditMenu
acRecordsMenu

You can use acRecordsMenu only for the Form view menu bar in Microsoft Access version 2.0 and Microsoft Access 95 databases. For other menus, use the number of the menu in the menu name argument list, as shown in the Macro window in previous versions of Microsoft Access (count down the list, starting from 0).

Command Required Variant. You can use one of the following intrinsic constants.

Intrinsic constants:
acNew
For other commands, use the number of the command in the command argument list, as shown in the Macro window in previous versions of Microsoft Access (count down the list, starting from 0).

Subcommand  Optional Variant. You can use one of the following intrinsic constants.

Intrinsic constants:
acObjectVerb acObjectUpdate

The acObjectVerb constant represents the first command on the submenu of the Object command on the Edit menu. The type of object determines the first command on the submenu. For example, this command is Edit for a Paintbrush object that can be edited.
For other commands on submenus, use the number of the subcommand in the subcommand argument list, as shown in the Macro window in previous versions of Microsoft Access (count down the list, starting from 0).

**Version**  Optional **Variant**. Use the intrinsic constant `acMenuVer70` for code written for Microsoft Access 95 databases, the intrinsic constant `acMenuVer20` for code written for Microsoft Access version 2.0 databases, and the intrinsic constant `acMenuVer1X` for code written for Microsoft Access version 1.x databases. This argument is available only in Visual Basic.

**Note** The default for this argument is `acMenuVer1X`, so that any code written for Microsoft Access version 1.x databases will run unchanged. If you're writing code for a Microsoft Access 95 or version 2.0 database and want to use the Microsoft Access 95 or version 2.0 menu commands with the `DoMenuItem` method, you must set this argument to `acMenuVer70` or `acMenuVer20`.

Also, when you are counting down the lists for the Menu Bar, Menu Name, Command, and Subcommand action arguments in the Macro window to get the numbers to use for the arguments in the `DoMenuItem` method, you must use the Microsoft Access 95 lists if the **Version** argument is `acMenuVer70`, the Microsoft Access version 2.0 lists if the **Version** argument is `acMenuVer20`, and the Microsoft Access version 1.x lists if **Version** is `acMenuVer1X` (or blank).

**Note** There is no `acMenuVer80` setting for this argument. You can't use the `DoMenuItem` method to display Microsoft Access 97 or Microsoft Access 2000 commands (although existing `DoMenuItem` methods in Visual Basic code will still work). Use the `RunCommand` method instead.
Remarks

Note  In Microsoft Access 97, the **DoMenuItem** method was replaced by the **RunCommand** method. The **DoMenuItem** method is included in this version of Microsoft Access only for compatibility with previous versions. When you run existing Visual Basic code containing a **DoMenuItem** method, Microsoft Access will display the appropriate *menu* or *toolbar* command for Microsoft Access 2000. However, unlike the DoMenuItem action in a *macro*, a **DoMenuItem** method in Visual Basic code isn't converted to a **RunCommand** method when you convert a database created in a previous version of Microsoft Access.

Some commands from previous versions of Microsoft Access aren't available in Microsoft Access 2000, and **DoMenuItem** methods that run these commands will cause an error when they're executed in Visual Basic. You must edit your Visual Basic code to replace or delete occurrences of such **DoMenuItem** methods.

The selections in the lists for the menu name, command, and subcommand action arguments in the Macro window depend on what you've selected for the previous arguments. You must use numbers or intrinsic constants that are appropriate for each **MenuBar**, **MenuName**, **Command**, and **Subcommand** argument.

If you leave the **Subcommand** argument blank but specify the **Version** argument, you must include the **Subcommand** argument's comma. If you leave the **Subcommand** and **Version** arguments blank, don't use a comma following the **Command** argument.
Example

The following example uses the **DoMenuItem** method to carry out the **Paste** command on the **Edit** menu in Form view in a Microsoft Access 95 database:

```
DoCmd.DoMenuItem acFormBar, acEditMenu, acPaste, , acMenuVer70
```

The next example carries out the **Tile** command on the **Window** menu in Form view in a Microsoft Access version 2.0 database:

```
DoCmd.DoMenuItem acFormBar, 4, 0, , acMenuVer20
```
Dropdown Method

You can use the **Dropdown** method to force the list in the specified combo box to drop down.

`expression.Dropdown`

*expression* Required. An expression that returns one of the objects in the Applies To list.
Remarks

For example, you can use this method to cause a combo box listing vendor codes to drop down when the vendor code control receives the focus during data entry.

If the specified combo box control doesn't have the focus, an error occurs. The use of this method is identical to pressing the F4 key when the control has the focus.
Example

The following example shows how you can use the **Dropdown** method within the **GotFocus** event procedure to force a combo box named SupplierID to drop down when it receives the focus.

```vbnet
Private Sub SupplierID_GotFocus()
   Me!SupplierID.Dropdown
End Sub
```
Show All
**DStDev, DStDevP Functions**

You can use the **DStDev** and **DStDevP** functions to estimate the standard deviation across a set of values in a specified set of records (a **domain**). Use the **DStDev** and **DStDevP** functions in Visual Basic, a **macro**, a query expression, or a **calculated control** on a form or report.

Use the **DStDevP** function to evaluate a population and the **DStDev** function to evaluate a population sample.

For example, you could use the **DStDev** function in a module to calculate the standard deviation across a set of students' test scores.

**DStDev(expr, domain, [criteria])**

**DStDevP(expr, domain, [criteria])**

The **DStDev** and **DStDevP** functions have the following arguments.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>expr</td>
<td>An expression that identifies the numeric field on which you want to find the standard deviation. It can be a <strong>string expression</strong> identifying a field from a table or query, or it can be an expression that performs a <strong>calculation on data in that field</strong>. In <em>expr</em>, you can include the name of a field in a table, a control on a form, a constant, or a function. If <em>expr</em> includes a function, it can be either built-in or user-defined, but not another domain aggregate or SQL aggregate function.</td>
</tr>
<tr>
<td>domain</td>
<td>A string expression identifying the set of records that constitutes the domain. It can be a table name or a query name for a query that does not require a parameter.</td>
</tr>
<tr>
<td>criteria</td>
<td>An optional string expression used to restrict the range of data on which the <strong>DStDev</strong> or <strong>DStDevP</strong> function is performed. For example, <em>criteria</em> is often equivalent to the WHERE clause in an SQL expression, without the word WHERE. If <em>criteria</em> is omitted, the <strong>DStDev</strong> and <strong>DStDevP</strong> functions evaluate <em>expr</em> against the entire domain. Any field that is included in <em>criteria</em> must also be a field in...</td>
</tr>
</tbody>
</table>
domain; otherwise, the \texttt{DStDev} and \texttt{DStDevP} functions will return a \texttt{Null}. 
Remarks

If *domain* refers to fewer than two records or if fewer than two records satisfy *criteria*, the **DStDev** and **DStDevP** functions return a **Null**, indicating that a standard deviation can't be calculated.

Whether you use the **DStDev** or **DStDevP** function in a macro, module, query expression, or calculated control, you must construct the *criteria* argument carefully to ensure that it will be evaluated correctly.

You can use the **DStDev** and **DStDevP** functions to specify criteria in the Criteria row of a select query. For example, you could create a query on an Orders table and a Products table to display all products for which the freight cost fell above the mean plus the standard deviation for freight cost. The Criteria row beneath the Freight field would contain the following expression:

\[
>(\text{DStDev}([\text{Freight}],[\text{Orders}]) + \text{DAvg}([\text{Freight}],[\text{Orders}]))
\]

You can use the **DStDev** and **DStDevP** functions in a calculated field expression of a query, or in the Update To row of an update query.

**Note** You can use the **DStDev** and **DStDevP** functions or the **StDev** and **StDevP** functions in a calculated field expression of a totals query. If you use the **DStDev** or **DStDevP** function, values are calculated before data is grouped. If you use the **StDev** or **StDevP** function, the data is grouped before values in the field expression are evaluated.

Use the **DStDev** and **DStDevP** function in a calculated control when you need to specify criteria to restrict the range of data on which the function is performed. For example, to display standard deviation for orders to be shipped to California, set the **ControlSource** property of a text box to the following expression:

\[
=\text{DStDev}([\text{Freight}],[\text{Orders},[\text{ShipRegion}]=\text{CA}])
\]

If you simply want to find the standard deviation across all records in *domain*, use the **StDev** or **StDevP** function.
If the data type of the field from which `expr` is derived is a number, the `DStDev` and `DStDevP` functions return a `Double` data type. If you use the `DStDev` or `DStDevP` function in a calculated control, include a data type conversion function in the expression to improve performance.

**Note**  Unsaved changes to records in *domain* are not included when you use these functions. If you want the `DStDev` or `DStDevP` function to be based on the changed values, you must first save the changes by clicking *Save Record* on the *Records* menu, moving the focus to another record, or by using the *Update* method.
Example

The following example returns estimates of the standard deviation for a population and a population sample for orders shipped to the United Kingdom. The domain is an Orders table. The criteria argument restricts the resulting set of records to those for which the ShipCountry value is UK.

Dim dblX As Double
Dim dblY As Double

' Sample estimate.
dblX = DStDev("[Freight]", "Orders", "[ShipCountry] = 'UK'")

' Population estimate.
dblY = DStDevP("[Freight]", "Orders", "[ShipCountry] = 'UK'")

The next example calculates the same estimates by using a variable, strCountry, in the criteria argument. Note that single quotation marks ('') are included in the string expression, so that when the strings are concatenated, the string literal UK will be enclosed in single quotation marks.

Dim strCountry As String
Dim dblX As Double
Dim dblY As Double

strCountry = "UK"

dblX = DStDev("[Freight]", "Orders", _
            "[ShipCountry] = '" & strCountry & "'")

dblY = DStDevP("[Freight]", "Orders", _
              "[ShipCountry] = '" & strCountry & "'")
**DSum Function**

You can use the **DSum** functions to calculate the sum of a set of values in a specified set of records (a **domain**). Use the **DSum** function in Visual Basic, a **macro**, a query expression, or a **calculated control**.

For example, you could use the **DSum** function in a calculated field expression in a query to calculate the total sales made by a particular employee over a period of time. Or you could use the **DSum** function in a calculated control to display a running sum of sales for a particular product.

**DSum(expr, domain, [criteria])**

The **DSum** function has the following arguments.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>expr</td>
<td>An expression that identifies the numeric field whose values you want to total. It can be a <strong>string expression</strong> identifying a field in a table or query, or it can be an expression that performs a <strong>calculation</strong> on data in that field. In expr, you can include the name of a field in a table, a control on a form, a constant, or a function. If expr includes a function, it can be either built-in or user-defined, but not another domain aggregate or SQL aggregate function.</td>
</tr>
<tr>
<td>domain</td>
<td>A string expression identifying the set of records that constitutes the domain. It can be a table name or a query name for a query that does not require a parameter. An optional string expression used to restrict the range of data on which the <strong>DSum</strong> function is performed. For example, <strong>criteria</strong> is often equivalent to the <strong>WHERE</strong> clause in an SQL expression, without the word <strong>WHERE</strong>. If <strong>criteria</strong> is omitted, the <strong>DSum</strong> function evaluates <strong>expr</strong> against the entire domain. Any field that is included in <strong>criteria</strong> must also be a field in <strong>domain</strong>; otherwise, the <strong>DSum</strong> function returns a <strong>Null</strong>.</td>
</tr>
<tr>
<td>criteria</td>
<td></td>
</tr>
</tbody>
</table>
Remarks

If no record satisfies the criteria argument or if domain contains no records, the DSum function returns a Null.

Whether you use the DSum function in a macro, module, query expression, or calculated control, you must construct the criteria argument carefully to ensure that it will be evaluated correctly.

You can use the DSum function to specify criteria in the Criteria row of a query, in a calculated field in a query expression, or in the Update To row of an update query.

**Note** You can use either the DSum or Sum function in a calculated field expression in a totals query. If you use the DSum function, values are calculated before data is grouped. If you use the Sum function, the data is grouped before values in the field expression are evaluated.

You may want to use the DSum function when you need to display the sum of a set of values from a field that is not in the record source for your form or report. For example, suppose you have a form that displays information about a particular product. You could use the DSum function to maintain a running total of sales of that product in a calculated control.

**Tip** If you need to maintain a running total in a control on a report, you can use the RunningSum property of that control if the field on which it is based is included in the record source for the report. Use the DSum function to maintain a running sum on a form.

**Note** Unsaved changes to records in domain aren't included when you use this function. If you want the DSum function to be based on the changed values, you must first save the changes by clicking Save Record on the Records menu, moving the focus to another record, or by using the Update method.
Example

The following example totals the values from the Freight field for orders shipped to the United Kingdom. The domain is an Orders table. The *criteria* argument restricts the resulting set of records to those for which ShipCountry equals UK.

```vba
Dim curX As Currency
curX = DSum("[Freight]", "Orders", "[ShipCountry] = 'UK'")
```

The next example calculates a total by using two separate criteria. Note that single quotation marks ('') and number signs (#) are included in the string expression, so that when the strings are concatenated, the string literal will be enclosed in single quotation marks, and the date will be enclosed in number signs.

```vba
Dim curX As Currency
curX = DSum("[Freight]", "Orders", _
    "[ShipCountry] = 'UK' AND [ShippedDate] > #1-1-95#")
```

You can use a domain function in the Update To row of an update query. For example, suppose you want to track current sales by product in a Products table. You could add a new field called SalesSoFar to the Products table, and run an update query to calculate the correct values and update the records. Create a new query based on the Products table, and click **Update** on the **Query** menu. Add the SalesSoFar field to the query grid, and enter the following in the Update To row:

```vba
DSum("[Quantity]*[UnitPrice]", "Order Details", "[ProductID] = " _
    & [ProductID])
```

When the query is run, Microsoft Access calculates the total amount of sales for each product, based on information from an Order Details table. The sum of sales for each product is added to the Products table.
**DVar, DVarP Functions**

You can use the **DVar** and **DVarP** functions to estimate variance across a set of values in a specified set of records (a domain). Use the **DVar** and **DVarP** functions in Visual Basic, a macro, a query expression, or a calculated control on a form or report.

Use the **DVarP** function to evaluate variance across a population and the **DVar** function to evaluate variance across a population sample.

For example, you could use the **DVar** function to calculate the variance across a set of students' test scores.

**DVar**(*expr, domain, [criteria]*)

**DVarP**(*expr, domain, [criteria]*)

The **DVar** and **DVarP** functions have the following arguments.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>expr</em></td>
<td>An expression that identifies the numeric field on which you want to find the variance. It can be a string expression identifying a field from a table or query, or it can be an expression that performs a calculation on data in that field. In <em>expr</em>, you can include the name field in a table, a control on a form, a constant, or a function. If <em>expr</em> includes a function, it can be either built-in or user-defined, but not another domain aggregate or SQL aggregate function. Any field included in <em>expr</em> must be a numeric field.</td>
</tr>
<tr>
<td><em>domain</em></td>
<td>A string expression identifying the set of records that constitutes the domain. It can be a table name or a query name for a query that does not require a parameter.</td>
</tr>
<tr>
<td><em>criteria</em></td>
<td>An optional string expression used to restrict the range of data on which the <strong>DVar</strong> or <strong>DVarP</strong> function is performed. For example, <em>criteria</em> is often equivalent to the WHERE clause in an SQL expression, without the word WHERE. If <em>criteria</em> is omitted, the <strong>DVar</strong> and <strong>DVarP</strong> functions evaluate <em>expr</em> against the entire domain.</td>
</tr>
</tbody>
</table>
Any field that is included in *criteria* must also be a field in *domain*; otherwise the **DVar** and **DVarP** functions return a **Null**.
Remarks

If domain refers to fewer than two records or if fewer than two records satisfy criteria, the DVar and DVarP functions return a Null, indicating that a variance can't be calculated.

Whether you use the DVar or DVarP function in a macro, module, query expression, or calculated control, you must construct the criteria argument carefully to ensure that it will be evaluated correctly.

You can use the DVar and DVarP function to specify criteria in the Criteria row of a select query, in a calculated field expression in a query, or in the Update To row of an update query.

Note You can use the DVar and DVarP functions or the Var and VarP functions in a calculated field expression in a totals query. If you use the DVar or DVarP function, values are calculated before data is grouped. If you use the Var or VarP function, the data is grouped before values in the field expression are evaluated.

Use the DVar and DVarP functions in a calculated control when you need to specify criteria to restrict the range of data on which the function is performed. For example, to display a variance for orders to be shipped to California, set the ControlSource property of a text box to the following expression:

= DVar("[Freight]", "Orders", "[ShipRegion] = 'CA'")

If you simply want to find the standard deviation across all records in domain, use the Var or VarP function.

Note Unsaved changes to records in domain are not included when you use these functions. If you want the DVar or DVarP function to be based on the changed values, you must first save the changes by clicking Save Record on the Records menu, moving the focus to another record, or by using the Update method.
Example

The following example returns estimates of the variance for a population and a population sample for orders shipped to the United Kingdom. The domain is an Orders table. The criteria argument restricts the resulting set of records to those for which ShipCountry equals UK.

Dim dblX As Double
Dim dblY As Double

' Sample estimate.
dblX = DVar("[Freight]", "Orders", ":ShipCountry":UK")

' Population estimate.
dblY = DVarP("[Freight]", "Orders", ":ShipCountry":UK")

The next example returns estimates by using a variable, strCountry, in the criteria argument. Note that single quotation marks ('') are included in the string expression, so that when the strings are concatenated, the string literal UK will be enclosed in single quotation marks.

Dim strCountry As String
Dim dblX As Double

strCountry = "UK"

dblX = DVar("[Freight]", "Orders", ":ShipCountry":":" _
& strCountry & ":")
Echo Method

Echo method as it applies to the Application object.

The Echo method specifies whether Microsoft Access repaints the display screen.

expression.Echo(EchoOn, bstrStatusBarText)

expression Required. An expression that returns an Application object.

EchoOn Required Integer. True (default) indicates that the screen is repainted.

bstrStatusBarText Optional String. A string expression that specifies the text to display in the status bar when repainting is turned on or off.
Remarks

If you are running Visual Basic code that makes a number of changes to objects displayed on the screen, your code may work faster if you turn off screen repaintng until the procedure has finished running. You may also want to turn repaintning off if your code makes changes that the user shouldn't or doesn't need to see.

The **Echo** method doesn't suppress the display of **modal** dialog boxes, such as error messages, or **pop-up forms**, such as property sheets.

If you turn screen repaintning off, the screen won't show any changes, even if the user presses CTRL+BREAK or Visual Basic encounters a **breakpoint**. You may want to create a **macro** that turns repaintning on and then assign the macro to a key or custom menu command. You can then use the key combination or menu command to turn repaintning on if it has been turned off in Visual Basic.

If you turn screen repaintning off and then try to step through the code, you won't be able to see progress through the code or any other visual cues until repaintning is turned back on. However, your code will continue to execute.

**Note**  Don't confuse the **Echo** method with the **Repaint** method. The **Echo** method turns screen repaintning on or off. The **Repaint** method forces an immediate screen repaintning.

---

**Echo method as it applies to the DoCmd object.**

The **Echo** method of the **DoCmd** object carries out the **Echo** action in Visual Basic.

```
expression.Echo(EchoOn, StatusBarText)
```

*expression*  Required. An expression that returns a **DoCmd** object.

*EchoOn*  Required **Variant**. Use **True** to turn **echo** on and **False** to turn it off.

*StatusBarText*  Optional **Variant**. A **string expression** indicating the text that appears in the status bar.
Remarks

If you leave the StatusBarText argument blank, don't use a comma following the echoon argument.

If you turn echo off in Visual Basic, you must turn it back on or it will remain off, even if the user presses CTRL+BREAK or if Visual Basic encounters a breakpoint. You may want to create a macro that turns echo on and then assign that macro to a key combination or a custom menu command. You could then use the key combination or menu command to turn echo on if it has been turned off in Visual Basic.

The Echo method of the DoCmd object was added to provide backward compatibility for running the Echo action in Visual Basic code in Microsoft Access for Windows 95. It's recommended that you use the existing Echo method of the Application object instead.
Example

As it applies to the **Application** object.

The following example uses the **Echo** method to prevent the screen from being repainted while certain operations are underway. While the procedure opens a form and minimizes it, the user only sees an hourglass icon indicating that processing is taking place, and the screen isn't repainted. When this task is completed, the hourglass changes back to a pointer and screen repainting is turned back on.

Public Sub EchoOff()

    ' Open the Employees form minimized.
    Application.Echo False
    DoCmd.Hourglass True
    DoCmd.OpenForm "Employees", acNormal
    DoCmd.Minimize
    Application.Echo True
    DoCmd.Hourglass False

End Sub

As it applies to the **DoCmd** object.

The following example uses the **Echo** method to turn echo off and display the specified text in the status bar while Visual Basic code is executing:

DoCmd.Echo False, "Visual Basic code is executing."
EuroConvert Function

You can use the **EuroConvert** function to convert a number to euro or from euro to a participating currency. You can also use it to convert a number from one participating currency to another by using the euro as an intermediary (triangulation). The **EuroConvert** function uses fixed conversion rates established by the European Union.

**EuroConvert**(number, sourcecurrency, targetcurrency, [fullprecision, triangulationprecision])

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>The number you want to convert, or a reference to a field containing the number.</td>
</tr>
<tr>
<td>sourcecurrency</td>
<td>A string expression, or reference to a field containing the string, corresponding to the International Standards Organization (ISO) acronym for the currency you want to convert. Can be one of the ISO codes listed in the following table.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Currency</th>
<th>ISO Code</th>
<th>Calculation Precision</th>
<th>Display Precision</th>
</tr>
</thead>
<tbody>
<tr>
<td>Belgian franc</td>
<td>BEF</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Luxembourg franc</td>
<td>LUF</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Deutsche mark</td>
<td>DEM</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Spanish peseta</td>
<td>ESP</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>French franc</td>
<td>FRF</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Irish punt</td>
<td>IEP</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Italian lira</td>
<td>ITL</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
In the preceding table, the calculation precision determines what currency unit to round the result to based on the conversion currency. For example, when converting to Deutsche marks, the calculation precision is 2, and the result is rounded to the nearest pfennig, 100 pfennigs to a mark. The display precision determines how many decimal places appear in the field containing the result.

Later versions of the *EuroConvert* function may support additional currencies. For information about new participating currencies and updates to the *EuroConvert* function, see the Microsoft Office Euro Currency Web site.

<table>
<thead>
<tr>
<th>Currency</th>
<th>ISO Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Danish Krone</td>
<td>DKK</td>
</tr>
<tr>
<td>Drachma</td>
<td>GRD</td>
</tr>
<tr>
<td>Swedish Krona</td>
<td>SEK</td>
</tr>
</tbody>
</table>
targetcurrency

A string expression, or reference to a field containing the string, corresponding to the ISO code of the currency to which you want to convert the number. For a list of ISO codes, see the sourcecurrency argument description.

Optional. A **Boolean** value where **True** (1) ignores the currency-specific rounding rules (called display precision in sourcecurrency argument description) and uses the 6-significant-digit conversion factor with no follow-up rounding. **False** (0) uses the currency-specific rounding rules to display the result. If the parameter is omitted, the default value is **False**.

fullprecision

Optional. An **Integer** value greater than or equal to 3 that specifies the number of significant digits in the triangulationprecision calculation precision used for the intermediate euro value when converting between two national currencies.
Remarks

Any trailing zeros are truncated and invalid parameters return #Error.

If the source ISO code is the same as the target ISO code, the original value of the number is active.

This function does not apply a format.

The EuroConvert function uses the current rates established by the European Union. If the rates change, Microsoft will update the function. To get full information about the rules and the rates currently in effect, see the European Commission publications about the euro. For information about obtaining these publications, see the Microsoft Office Euro Currency Web site.
Example

The first example converts 1.20 Deutsche marks to a euro dollar value (answer = 0.61). The second example converts 1.47 French francs to Deutsche marks (answer = 0.44 DM). They assume conversion rates of 1 euro = 6.55858 French francs and 1.92974 Deutsche marks.

`EuroConvert(1.20,"DEM","EUR")`
`EuroConvert(1.47,"FRF","DEM",TRUE,3)`
Eval Function

You can use the **Eval** function to evaluate an **expression** that results in a text string or a numeric value.

You can construct a string and then pass it to the **Eval** function as if the string were an actual expression. The **Eval** function evaluates the **string expression** and returns its value. For example, `Eval("1 + 1")` returns 2.

If you pass to the **Eval** function a string that contains the name of a function, the **Eval** function returns the return value of the function. For example, `Eval("Chr$(65)")` returns "A".

**Eval**(stringexpr)

The **stringexpr** argument is an expression that evaluates to an alphanumeric text string. For example, **stringexpr** can be a function that returns a string or a numeric value. Or it can be a reference to a **control** on a form. The **stringexpr** argument must evaluate to a string or numeric value; it can't evaluate to a **Microsoft Access object**.

**Note** If you are passing the name of a function to the **Eval** function, you must include parentheses after the name of the function in the **stringexpr** argument. For example:

`' ShowNames is user-defined function. Debug.Print Eval("ShowNames()")`

`Debug.Print Eval("StrComp("'"Joe"','"joe"', 1)")`

`Debug.Print Eval("Date()")`
Remarks

You can use the **Eval** function in a calculated control on a form or report, or in a macro or module. The **Eval** function returns a **Variant** that is either a string or a numeric type.

The argument *stringexpr* must be an expression that is stored in a string. If you pass to the **Eval** function a string that doesn't contain a numeric expression or a function name but only a simple text string, a run-time error occurs. For example, **Eval("Smith")** results in an error.

You can use the **Eval** function to determine the value stored in the **Value** property of a control. The following example passes a string containing a full reference to a control to the **Eval** function. It then displays the current value of the control in a dialog box.

```vbnet
Dim ctl As Control
Dim strCtl As String

Set ctl = Forms!Employees!LastName
strCtl = "Forms!Employees!LastName"
MsgBox ("The current value of " & ctl.Name & " is " & Eval(strCtl))
```

You can use the **Eval** function to access expression operators that aren't ordinarily available in Visual Basic. For example, you can't use the SQL operators *Between...And* or *In* directly in your code, but you can use them in an expression passed to the **Eval** function.

The next example determines whether the value of a ShipRegion control on an Orders form is one of several specified state abbreviations. If the field contains one of the abbreviations, **intState** will be **True** (−1). Note that you use single quotation marks (') to include a string within another string.

```vbnet
Dim intState As Integer
intState = Eval("Forms!Orders!ShipRegion In " & "('AK', 'CA', 'ID', 'WA', 'MT', 'NM', 'OR')")
```
Example

The following example assumes that you have a series of 50 functions defined as A1, A2, and so on. This example uses the **Eval** function to call each function in the series.

```vba
Sub CallSeries()
    Dim intI As Integer
    For intI = 1 To 50
        Eval("A" & intI & ")")
    Next intI
End Sub
```

The next example triggers a Click event as if the user had clicked a button on a form. If the value of the button's **OnClick** property begins with an equal sign (=), signifying that it is the name of a function, the **Eval** function calls the function, which is equivalent to triggering the **Click** event. If the value doesn't begin with an equal sign, then the value must name a macro. The **RunMacro** method of the **DoCmd** object runs the named macro.

```vba
Dim ctl As Control
Dim varTemp As Variant
Set ctl = Forms!Contacts!HelpButton
If (Left(ctl.OnClick, 1) = ")") Then
    varTemp = Eval(Mid(ctl.OnClick, 2))
Else
    DoCmd.RunMacro ctl.OnClick
End If
```
Execute Method

The **Execute** method performs the specified **smart tag** action.

Smart tag actions are processes that are programmed into smart tags that allow users to perform certain functions related to the smart tag. For example, one action for a smart tag might be to access a Web site, while another action inserts contact information from Microsoft Outlook, while yet another action displays a map and driving directions.

\[expression.\text{Execute}()\]

- **expression** Required. An expression that returns a **SmartTagAction** object.
Remarks

Calling the **Execute** method will result in a run-time error if the smart tag is:

- Located on a report.
- Located on a form in **Design view**, **PivotTable view**, or **PivotChart view**.
ExportXML Method

The ExportXML method allows developers to export XML data, schemas, and presentation information from Microsoft SQL Server 2000 Desktop Engine (MSDE 2000), Microsoft SQL Server 6.5 or later, or the Microsoft Jet database engine.

expression.ExportXML(ObjectType, DataSource, DataTarget, SchemaTarget, PresentationTarget, ImageTarget, Encoding, OtherFlags, FilterCriteria, AdditionalData)

expression  Required. An expression that returns an Application object.

ObjectType  Required AcExportXMLObjectType. The type of AccessObject object to export.

AcExportXMLObjectType can be one of these AcExportXMLObjectType constants:
acExportForm
acExportFunction
acExportQuery
acExportReport
acExportServerView
acExportStoredProcedure
acExportTable

DataSource  Required String. The name of the AccessObject object to export. The default is the currently open object of the type specified by the ObjectType argument.

DataTarget  Optional String. The file name and path for the exported data. If this argument is omitted, data is not exported.

SchemaTarget  Optional String. The file name and path for the exported schema information. If this argument is omitted, schema information is not exported to a separate XML file.
**PresentationTarget**  Optional **String**. The file name and path for the exported presentation information. If this argument is omitted, presentation information is not exported.

**ImageTarget**  Optional **String**. The path for exported images. If this argument is omitted, images are not exported.

**Encoding**  Optional **AcExportXMLEncoding**. The text encoding to use for the exported XML.

**AcExportXMLEncoding** can be one of these **AcExportXMLEncoding** constants:
- **acUTF16**
- **acUTF8**  **default**

**OtherFlags**  Optional **AcExportXMLOtherFlags**. A **bit mask** that specifies other behaviors associated with exporting to XML. The following table describes the behavior that results from specific values; values can be added to specify a combination of behaviors.

**AcExportXMLOtherFlags** can be one of these **AcExportXMLOtherFlags** constants:
- **acEmbedSchema** Writes schema information into the document specified by the **DataTarget** argument; this value takes precedence over the **SchemaTarget** argument.
- **acExcludePrimaryKeyAndIndexes** Does not export primary key and index schema properties.
- **acLiveReportSource** Creates a live link to a remote Microsoft SQL Server 2000 database. Valid only when you are exporting reports that are bound to a Microsoft SQL Server 2000 database.
- **acPersistReportML** Persists the exported object's ReportML information.
- **acRunFromServer** Creates an Active Server Pages (ASP) wrapper; otherwise, default is an HTML wrapper. Applies only when you are exporting reports.

**FilterCriteria**  Optional **String**. Specifies a subset of records to be exported.

**AdditionalData**  Optional **AdditionalData**. Specifies additional tables to export. This argument is ignored if the **OtherFlags** argument is set to
acLiveReportSource.
Remarks

Although the **DataTarget**, **SchemaTarget**, and **PresentationTarget** arguments are all optional, at least one must be specified when you are using this method. When the **ExportXML** method is called from within an **AccessObject** object, the default behavior is to overwrite any existing files specified in any of the arguments.
Example

The following example exports the contents of the Customers table in the Northwind Traders sample database, along with the contents of the Orders and Orders Details tables, to an XML data file named Customer Orders.xml.

Sub ExportCustomerOrderData()
    Dim objOrderInfo As AdditionalData
    Dim objOrderDetailsInfo As AdditionalData

    Set objOrderInfo = Application.CreateAdditionalData

    ' Add the Orders and Order Details tables to the data to be expo
    Set objOrderDetailsInfo = objOrderInfo.Add("Orders")
    objOrderDetailsInfo.Add "Order Details"

    ' Export the contents of the Customers table. The Orders and Ord
    ' Details tables will be included in the XML file.
    Application.ExportXML ObjectType:=acExportTable,
    DataSource:="Customers", DataTarget:="Customer Orders.xml",
    AdditionalData:=objOrderInfo
End Sub

The following example exports the table named Customers in the current
database to an XML file; the data and schema are exported as separate files.

Application.ExportXML _
    ObjectType:=acExportTable, _
    DataSource:="Customers", _
    DataTarget:="Customers.xml", _
    SchemaTarget:="CustomersSchema.xml"

The following example exports the report called Fall2000 in the current database
to an XML file. Presentation information is also exported, and images are placed
in the specified directory. The report is exported with an ASP wrapper rather
than the default HTML wrapper.

Application.ExportXML _
    ObjectType:=acExportReport, _
    DataSource:="Fall2000", _
    DataTarget:="Fall2000.xml", _
    PresentationTarget:="Fall2000Report.xsl", _
    ImageTarget:="Images", _
OtherFlags:=acRunFromServer
Find Method

Finds specified text in a standard module or class module.

expression.Find(Target, StartLine, StartColumn, EndLine, EndColumn, WholeWord, MatchCase, PatternSearch)

expression Required. An expression that returns one of the objects in the Applies To list.

Target Required String. A string expression that evaluates to the text that you want to find.

StartLine Required Long. The line on which to begin searching. If a match is found, the value of the StartLine argument is set to the line on which the beginning character of the matching text is found.

StartColumn Required Long. The column on which to begin searching. Each character in a line is in a separate column, beginning with zero on the left side of the module. If a match is found, the value of the StartColumn argument is set to the column on which the beginning character of the matching text is found.

EndLine Required Long. The line on which to stop searching. If a match is found, the value of the EndLine argument is set to the line on which the ending character of the matching text is found.

EndColumn Required Long. The column on which to stop searching. If a match is found, the value of the EndColumn argument is set to the column on which the beginning character of the matching text is found.

WholeWord Optional Boolean. True results in a search for whole words only. The default is False.

MatchCase Optional Boolean. True results in a search for words with case matching the Target argument. The default is False.

PatternSearch Optional Boolean. True results in a search in which the Target argument may contain wildcard characters such as an asterisk (*) or a question
mark (?). The default is **False**.
Remarks

The **Find** method searches for the specified text `string` in a **Module** object. If the string is found, the **Find** method returns **True**.

To determine the position in the module at which the search text was found, pass empty variables to the **Find** method for the **StartLine**, **StartColumn**, **EndLine**, and **EndColumn** arguments. If a match is found, these arguments will contain the line number and column position at which the search text begins (**StartLine**, **StartColumn**) and ends (**EndLine**, **EndColumn**).

For example, if the search text is found on line 5, begins at column 10, and ends at column 20, the values of these arguments will be: **StartLine** = 5, **StartColumn** = 10, **EndLine** = 5, **EndColumn** = 20.
**Example**

The following function finds a specified string in a module and replaces the line that contains that string with a new specified line.

```vba
Function FindAndReplace(strModuleName As String, _
    strSearchText As String, _
    strNewText As String) As Boolean
    Dim mdl As Module
    Dim lngSLine As Long, lngSCol As Long
    Dim lngELine As Long, lngECol As Long
    Dim strLine As String, strNewLine As String
    Dim intChr As Integer, intBefore As Integer, _
        intAfter As Integer
    Dim strLeft As String, strRight As String
    ' Open module.
    DoCmd.OpenModule strModuleName
    ' Return reference to Module object.
    Set mdl = Modules(strModuleName)
    ' Search for string.
    If mdl.Find(strSearchText, lngSLine, lngSCol, lngELine, _
        lngECol) Then
        ' Store text of line containing string.
        strLine = mdl.Lines(lngSLine, Abs(lngELine - lngSLine) + 1)
        ' Determine length of line.
        intChr = Len(strLine)
        ' Determine number of characters preceding search text.
        intBefore = lngSCol - 1
        ' Determine number of characters following search text.
        intAfter = intChr - CInt(lngECol - 1)
        ' Store characters to left of search text.
        strLeft = Left$(strLine, intBefore)
        ' Store characters to right of search text.
        strRight = Right$(strLine, intAfter)
        ' Construct string with replacement text.
        strNewLine = strLeft & strNewText & strRight
        ' Replace original line.
        mdl.ReplaceLine lngSLine, strNewLine
        FindAndReplace = True
    Else
        MsgBox "Text not found."
        FindAndReplace = False
    End If
```
Exit_FindAndReplace:
   Exit Function

Error_FindAndReplace:

MsgBox Err & ": " & Err.Description
   FindAndReplace = False
   Resume Exit_FindAndReplace
End Function
FindNext Method

The **FindNext** method carries out the **FindNext** action in Visual Basic.

*expression*.**FindNext**

*expression*  Required. An expression that returns a **DoCmd** object.
Remarks

This method has no arguments and can be called using the syntax `DoCmd.FindNext`.

You can use the `FindNext` method to find the next record that meets the criteria specified by the previous `FindRecord` method or the `Find In Field` dialog box, available by clicking `Find` on the `Edit` menu. You can use the `FindNext` method to search repeatedly for records. For example, you can move successively through all the records for a specific customer.
FindRecord Method

The **FindRecord** method carries out the **FindRecord** action in Visual Basic.

```
expression. FindRecord(FindWhat, Match, MatchCase, Search, 
SearchAsFormatted, OnlyCurrentField, FindFirst)
```

*expression* Required. An expression that returns one of the objects in the Applies To list.

*FindWhat* Required **Variant**. An *expression* that evaluates to text, a number, or a date. The expression contains the data to search for.

*Match* Optional **AcFindMatch**.

AcFindMatch can be one of these AcFindMatch constants.

- **acAnywhere**
- **acEntire** *default*
- **acStart**

If you leave this argument blank, the default constant (**acEntire**) is assumed.

*MatchCase* Optional **Variant**. Use **True** for a case-sensitive search and **False** for a search that's not case-sensitive. If you leave this argument blank, the default (**False**) is assumed.

*Search* Optional **AcSearchDirection**.

AcSearchDirection can be one of these AcSearchDirection constants.

- **acDown**
- **acSearchAll** *default*
- **acUp**

If you leave this argument blank, the default constant (**acSearchAll**) is assumed.
**SearchAsFormatted** Optional Variant. Use **True** to search for data as it's formatted and **False** to search for data as it's stored in the database. If you leave this argument blank, the default (**False**) is assumed.

**OnlyCurrentField** Optional **AcFindField**.

*AcFindField* can be one of these *AcFindField* constants.

**acAll**

**acCurrent** default

If you leave this argument blank, the default constant (**acCurrent**) is assumed.

**FindFirst** Optional Variant. Use **True** to start the search at the first record. Use **False** to start the search at the record following the current record. If you leave this argument blank, the default (**True**) is assumed.
Remarks

For more information on how the action and its arguments work, see the action topic.

You can leave an optional argument blank in the middle of the syntax, but you must include the argument's comma. If you leave one or more trailing arguments blank, don't use a comma following the last argument you specify.
Example

The following example finds the first occurrence in the records of the name Smith in the current field. It doesn't find occurrences of smith or Smithson.

DoCmd.**FindRecord** "Smith", , True,, True
Follow Method

The **Follow** method opens the document or Web page specified by a hyperlink address associated with a control on a form or report.

`expression.Follow(NewWindow, AddHistory, ExtraInfo, Method, HeaderInfo)`

*expression* Required. An expression that returns one of the objects in the Applies To list.

**NewWindow** Optional Boolean. A `Boolean` value where `True` (–1) opens the document in a new window and `False` (0) opens the document in the current window. The default is `False`.

**AddHistory** Optional Boolean. A `Boolean` value where `True` adds the hyperlink to the History folder and `False` doesn't add the hyperlink to the History folder. The default is `True`.

**ExtraInfo** Optional Variant. A `string` or an `array` of `Byte` data that specifies additional information for navigating to a hyperlink. For example, this argument may be used to specify a search parameter for an .ASP or .IDC file. In your Web browser, the `extrainfo` argument may appear after the hyperlink address, separated from the address by a question mark (?). You don't need to include the question mark when you specify the `extrainfo` argument.

**Method** Optional `MsoExtraInfoMethod`. An `Integer` value that specifies how the `extrainfo` argument is attached. The `method` argument may be one of the following `intrinsic constants`.

MsoExtraInfoMethod can be one of these MsoExtraInfoMethod constants.

**msoMethodGet default**. The `extrainfo` argument is appended to the hyperlink address and can only be a string. This value is passed by default.

**msoMethodPost**. The `extrainfo` argument is posted, either as a string or an array of type Byte.

**HeaderInfo** Optional `String`. A string that specifies header information. By default the `headerinfo` argument is a `zero-length string` (" ").
Remarks

The **Follow** method has the same effect as clicking a hyperlink.

You can include the **Follow** method in an event procedure if you want to open a hyperlink in response to a user action. For example, you may want to open a web page with reference information when a user opens a particular form.

When you use the **Follow** method, you don't need to know the address specified by a control's **HyperlinkAddress** property. You only need to know the name of the control that contains the hyperlink. Conversely, when you use the **FollowHyperlink** method, you need to specify the address for the particular hyperlink you wish to follow.
Example

The following example sets the HyperlinkAddress property of a command button and then opens the hyperlink when the form is loaded.

To try this example, create a form and add a command button named Command0. Paste the following code into the form's module and switch to Form view:

Private Sub Form_Load()
    Dim ctl As CommandButton
    Set ctl = Me!Command0
    With ctl
        .Visible = False
        .HyperlinkAddress = "http://www.microsoft.com/
        .Hyperlink.Follow
    End With
End Sub
FollowHyperlink Method

The **FollowHyperlink** method opens the document or Web page specified by a hyperlink address.

\[ \text{expression}.\text{FollowHyperlink}(\text{Address, SubAddress, NewWindow, AddHistory, ExtraInfo, Method, HeaderInfo}) \]

*expression*  Required. An expression that returns one of the objects in the Applies To list.

**Address**  Required **String**. A **string expression** that evaluates to a valid hyperlink address.

**SubAddress**  Optional **String**. A string expression that evaluates to a named location in the document specified by the **address** argument. The default is a zero-length string (" ").

**NewWindow**  Optional **Boolean**. A **Boolean** value where **True** (–1) opens the document in a new window and **False** (0) opens the document in the current window. The default is **False**.

**AddHistory**  Optional **Boolean**. A **Boolean** value where **True** adds the hyperlink to the History folder and **False** doesn't add the hyperlink to the History folder. The default is **True**.

**ExtraInfo**  Optional **Variant**. A **string** or an **array** of **Byte** data that specifies additional information for navigating to a hyperlink. For example, this argument may be used to specify a search parameter for an .asp or .idc file. In your Web browser, the **extrainfo** argument may appear after the hyperlink address, separated from the address by a question mark (?). You don't need to include the question mark when you specify the **extrainfo** argument.

**Method**  Optional **MsoExtraInfoMethod**. An **Integer** value that specifies how the **extrainfo** argument is attached. The **method** argument may be one of the following **intrinsic constants**.
MsoExtraInfoMethod can be one of these MsoExtraInfoMethod constants. **msoMethodGet default.** The extrainfo argument is appended to the hyperlink address and can only be a string. This value is passed by default. **msoMethodPost.** The extrainfo argument is posted, either as a string or as an array of type `Byte`.

**HeaderInfo**  Optional **String.** A string that specifies header information. By default the `headerinfo` argument is a zero-length string.
Remarks

By using the **FollowHyperlink** method, you can follow a hyperlink that doesn't exist in a control. This hyperlink may be supplied by you or by the user. For example, you can prompt a user to enter a hyperlink address in a dialog box, then use the **FollowHyperlink** method to follow that hyperlink.

You can use the `extrainfo` and `method` arguments to supply additional information when navigating to a hyperlink. For example, you can supply parameters to a search engine.

You can use the **Follow** method to follow a hyperlink associated with a control.
Example

The following function prompts a user for a hyperlink address and then follows the hyperlink:

Function GetUserAddress() As Boolean
    Dim strInput As String
    On Error GoTo Error_GetUserAddress
    strInput = InputBox("Enter a valid address")
    Application.FollowHyperlink strInput, , True
    GetUserAddress = True
Exit_GetUserAddress:
    Exit Function
Error_GetUserAddress:
    MsgBox Err & ": " & Err.Description
    GetUserAddress = False
    Resume Exit_GetUserAddress
End Function

You could call this function with a procedure such as the following:

Sub CallGetUserAddress()
    If GetUserAddress = True Then
        MsgBox "Successfully followed hyperlink."
    Else
        MsgBox "Could not follow hyperlink."
    End If
End Sub
GetDependencyInfo Method

Returns a **DependencyInfo** object that represents the database objects that are dependent upon the specified object.

*expression*.GetDependencyInfo()

*expression*   Required. An expression that returns an **AccessObject** object.
Remarks

Access does not search Visual Basic for Applications (VBA) code, macros, or data access pages for dependencies.

This method will return a run-time error if any of the following conditions are true:

- The **Track name AutoCorrect info** setting (Tools menu, Options dialog box, General tab) is disabled. You can use the following code to enable the **Track name AutoCorrect info** setting and update the dependency information for all of the objects in the database:

  Application.SetOption "Track Name AutoCorrect Info", 1

- You have insufficient permissions to check the dependency information for the specified **AccessObject** object.
- This method is being called from an **Access project (.adp)**.
GetHiddenAttribute Method

The GetHiddenAttribute method returns the value of hidden attribute of a Microsoft Access object in the object's Properties dialog box, available by selecting the object in the Database window and clicking Properties on the View menu.

`object.GetHiddenAttribute(objecttype, objectname)`

The GetHiddenAttribute method has the following arguments.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>Optional. The Application object. One of the following intrinsic constants:</td>
</tr>
<tr>
<td></td>
<td>acDataAccessPage</td>
</tr>
<tr>
<td></td>
<td>acDiagram</td>
</tr>
<tr>
<td></td>
<td>acForm</td>
</tr>
<tr>
<td></td>
<td>acMacro</td>
</tr>
<tr>
<td>objecttype</td>
<td>acModule</td>
</tr>
<tr>
<td></td>
<td>acQuery</td>
</tr>
<tr>
<td></td>
<td>acReport</td>
</tr>
<tr>
<td></td>
<td>acServerView</td>
</tr>
<tr>
<td></td>
<td>acStoredProcedure</td>
</tr>
<tr>
<td></td>
<td>acTable</td>
</tr>
</tbody>
</table>

You must enter a constant for the `objecttype` argument, acDefault is not a valid entry.

`objectname` A string expression identifying the name of the Access object.
Remarks

The `GetHiddenAttribute` method (along with the `SetHiddenAttribute` method) provide a means of changing an object's hidden attribute from Visual Basic code. With these methods, you can set or read the hidden option available in the object's `Properties` dialog box.

Since the hidden attributes that the user can set by selecting or clearing a check box, the `GetHiddenAttribute` method returns `True` if the option setting is Yes (the check box is selected) or `False` if the option setting is No (the check box is cleared). For example, to set an option of this kind by using the `SetHiddenAttribute` method, specify `True` or `False` for the setting argument, as in the following:

```
Application.SetHiddenAttribute acTable,"Customers", True
```
GetOption Method

The **GetOption** method returns the current value of an option in the **Options** dialog box, available by clicking **Options** on the **Tools** menu. **Variant.**

```
expression.GetOption(OptionName)
```

- **expression** Required. An expression that returns one of the objects in the Applies To list.
- **OptionName** Required **String.** The name of the option. For a list of **optionname** argument strings, see **Set Options from Visual Basic.**
Remarks

The **GetOption** and **SetOption** methods provide a means of changing environment options from **Visual Basic** code. With these methods, you can set or read any option available in the **Options** dialog box, except for options on the **Modules** tab.

The available option settings depend on the type of option being set. There are three general types of options:

- Yes/No options that can be set by selecting or clearing a **check box**.
- Options that can be set by entering a **string** or numeric value.
- Predefined options that can be chosen from a **list box**, **combo box**, or **option group**.

For options that the user sets by selecting or clearing a check box, the **GetOption** method returns **True** (–1) if the option setting is Yes (the check box is selected) or **False** (0) if the option setting is No (the check box is cleared). To set an option of this kind by using the **SetOption** method, specify **True** or **False** for the **setting** argument, as in the following example:

```vba
Application.SetOption "Show Status Bar", True
```

For options that the user sets by typing a string or numeric value, the **GetOption** method returns the setting as it's displayed in the dialog box. The following example returns a string containing the left margin setting:

```vba
Dim varSetting As Variant
varSetting = Application.GetOption("Left Margin")
```

To set this type of option by using the **SetOption** method, specify the string or numeric value that would be typed in the dialog box. The following example sets the default form template to **OrderTemplate**:

```vba
Application.SetOption "Form Template", "OrderTemplate"
```
For options with settings that are choices in list boxes or combo boxes, the **GetOption** method returns a number corresponding to the position of the setting in the list. Indexing begins with zero, so the **GetOption** method returns zero for the first item, 1 for the second item, and so on. For example, if the **Default Field Type** option on the **Tables/Queries** tab is set to AutoNumber, the sixth item in the list, the **GetOption** method returns 5.

To set this type of option, specify the option's numeric position within the list as the *setting* argument for the **SetOption** method. The following example sets the **Default Field Type** option to AutoNumber:

```
Application.SetOption "Default Field Type", 5
```

Other options are set by clicking on an option button in an option group in the **Options** dialog box. In Visual Basic, these options are also set by specifying a particular option's position within the option group. The first option in the group is numbered zero, the second, 1, and so on. For example, if the **Selection Behavior** option on the **Forms/Reports** tab is set to Partially Enclosed, the **GetOption** method returns zero, as in the following example:

```
Debug.Print Application.GetOption("Selection Behavior")
```

To set an option that's a member of an option group, specify the index number of the option within the group. The following example sets **Selection Behavior** to Fully Enclosed:

```
Application.SetOption "Selection Behavior", 1
```

**Notes**

- When you use the **GetOption** method or the **SetOption** method to set an option in the **Options** dialog box, you don't need to specify the individual tab on which the option is found.

- You can't use the **GetOption** method or the **SetOption** method to read or set any of the options found on the **Module** tab of the **Options** dialog box.

- If the return value of the **GetOption** method is assigned to a variable, the
variable must be declared as a **Variant**.

- If your database may run on a version of Microsoft Access for a language other than the one in which you created it, then you must supply the arguments for the **GetOption** and **SetOption** methods in English.

When you quit Microsoft Access, you can reset all options to their original settings by using the **SetOption** method on all changed options. You may want to create public variables to store the values of the original settings. You might include code to reset options in the Close event procedure for a form, or in a custom exit procedure that the user must run to quit the application.
GoToControl Method

The GoToControl method carries out the GoToControl action in Visual Basic.

expression.GoToControl(ControlName)

expression Required. An expression that returns one of the objects in the Applies To list.

ControlName Required Variant. A string expression that's the name of a control on the active form or datasheet.
Remarks

For more information on how the action and its argument work, see the action topic.

Use only the name of the control for the controlname argument, not the full syntax.

You can also use a variable declared as a Control data type for this argument.

Dim ctl As Control
Set ctl = Forms!Form1!Field3
DoCmd.GoToControl ctl.Name

You can also use the SetFocus method to move the focus to a control on a form or any of its subforms, or to a field in an open table, query, or form datasheet. This is the preferred method for moving the focus in Visual Basic, especially to controls on subforms and nested subforms, because you can use the full syntax to specify the control you want to move to.
Example

The following example uses the `GoToControl` method to move the focus to the EmployeeID field:

```
DoCmd.GoToControl "EmployeeID"
```
GoToPage Method

GoToPage method as it applies to the Form object.

The GoToPage method moves the focus to the first control on a specified page in the active form.

expression.GoToPage(PageNumber, Right, Down)

expression  Required. An expression that returns one of the above objects.

PageNumber  Required Long. A numeric expression that's a valid page number for the active form.

Right  Optional Long. A numeric expression that's a valid horizontal offset (in twips) from the left side of the window to the part of the page to be viewed.

Down  Optional Long. A numeric expression that's a valid vertical offset (in twips) from the top of the window to the part of the page to be viewed.

GoToPage method as it applies to the DoCmd object.

The GoToPage method of the DoCmd object carries out the GoToPage action in Visual Basic. For more information on how the action and its arguments work, see the action topic.

expression.GoToPage(PageNumber, Right, Down)

expression  Required. An expression that returns one of the above objects.

PageNumber  Required Variant. A numeric expression that's a valid page number for the active form. If you leave this argument blank, the focus stays on the current page. You can use the right and down arguments to display the part of the page you want to see.

Right  Optional Variant. A numeric expression that's a valid horizontal offset for the page.
**Down**  Optional **Variant.** A numeric expression that's a valid vertical offset for the page.
Remarks

Remarks as it applies to the **Form** object.

When you use this method to move to a specified page of a form, the focus is set to the first control on the page, as defined by the form's **tab order**. To move to a particular control on the form, use the **SetFocus** method.

You can use the **GoToPage** method if you've created page breaks on a form to group related information. For example, you might have an **Employees** form with personal information on the first page, office information on the second page, and sales information on the third page. You can use the **GoToPage** method to move to the desired page.

You can use the **right** and **down** arguments for forms with pages larger than the Microsoft Access window. Use the **pagenumber** argument to move to the desired page, and then use the **right** and **down** arguments to display the part of the page you want to see. Microsoft Access displays the part of the page that's offset from the upper-left corner of the window by the distance specified in the **right** and **down** arguments.

Remarks as it applies to the **DoCmd** object.

The units for the **right** and **down** arguments are expressed in **twips**.

If you specify the **right** and **down** arguments and leave the **pagenumber** argument blank, you must include the **pagenumber** argument's comma. If you don't specify the **right** and **down** arguments, don't use a comma following the **pagenumber** argument.

The **GoToPage** method of the **DoCmd** object was added to provide backwards compatibility for running the GoToPage action in Visual Basic code in Microsoft Access 95. It's recommended that you use the existing **GoToPage** method of the **Form** object instead.
Example

**Exmaple as it applies to the Form object.**

The following example uses the `GoToPage` method to move the focus to the second page of the Customer form at the position specified by the `right` and `down` arguments:

```
Forms!Customer.GoToPage 2, 1440, 600
```

**Example as it applies to the DoCmd object.**

The following example uses the `GoToPage` method to move the focus to the position specified by the horizontal and vertical offsets on the second page of the active form:

```
DoCmd.GoToPage 2, 1440, 567
```
GoToRecord Method

The **GoToRecord** method carries out the **GoToRecord** action in Visual Basic.

*expression*.GoToRecord(*ObjectType*, *ObjectName*, *Record*, *Offset*)

*expression*  Required. An expression that returns one of the objects in the Applies To list.

**ObjectType**  Optional **AcDataObjectType**.

AcDataObjectType can be one of these AcDataObjectType constants.
acActiveDataObject default
acDataForm
acDataFunction
acDataQuery
acDataServerView
acDataStoredProcedure
acDataTable

**ObjectName**  Optional **Variant**. A string expression that's the valid name of an object of the type selected by the **objecttype** argument.

**Record**  Optional **AcRecord**.

AcRecord can be one of these AcRecord constants.
acFirst
acGoTo
acLast
acNewRec
acNext default
acPrevious

If you leave this argument blank, the default constant (acNext) is assumed.
**Offset** Optional **Variant**. A **numeric expression** that represents the number of records to move forward or backward if you specify **acNext** or **acPrevious** for the **record** argument, or the record to move to if you specify **acGoTo** for the **record** argument. The expression must result in a valid record number.
**Remarks**

For more information on how the action and its arguments work, see the action topic.

If you leave the `objecttype` and `objectname` arguments blank (the default constant, `acActiveDataObject`, is assumed for `objecttype`), the active object is assumed.

You can leave an optional argument blank in the middle of the syntax, but you must include the argument's comma. If you leave one or more trailing arguments blank, don't use a comma following the last argument you specify.
Example

The following example uses the **GoToRecord** method to make the seventh record in the form Employees current:

```vba
DoCmd.GoToRecord acDataForm, "Employees", acGoTo, 7
```
GUIDFromString Function

The **GUIDFromString** function converts a **string** to a **GUID**, which is an **array** of type **Byte**.

**GUIDFromString**(stringexpression)

The **GUIDFromString** function has the following argument:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stringexpression</td>
<td>A <strong>string expression</strong> which evaluates to a GUID in string form.</td>
</tr>
</tbody>
</table>
Remarks

The Microsoft Jet database engine stores GUIDs as arrays of type Byte. However, Microsoft Access can't return Byte data from a control on a form or report. In order to return the value of a GUID from a control, you must convert it to a string. To convert a GUID to a string, use the StringFromGUID function. To convert a string to a GUID, use the GUIDFromString function.
Example

The following example uses the **GUIDFromString** function to convert a string to a GUID. The string is a GUID stored in string form in a replicated Employees table. The field, s_GUID, is a hidden field added to every replicated table in a replicated database.

Sub CheckGUIDType()
    Dim dbsConn As ADODB.Connection
    Dim rstEmployees As ADODB.Recordset

    ' Make a connection to the current database.
    Set dbsConn = Application.CurrentProject.Connection
    Set rstEmployees = New ADODB.Recordset
    rstEmployees.Open "Employees", dbsConn, , , adCmdTable

    ' Print the GUID to the immediate window.
    Debug.Print rst!s_GUID
    Debug.Print TypeName(rst!s_GUID)
    Debug.Print TypeName(GuidFromString(rst!s_GUID))

    Set rstEmployees = Nothing
    Set dbsConn = Nothing
End Sub
Hourglass Method

The Hourglass method carries out the Hourglass action in Visual Basic.

expression.Hourglass(HourglassOn)

expression Required. An expression that returns one of the objects in the Applies To list.

HourglassOn Required Variant. Use True (-1) to display the hourglass icon (or another icon you've chosen). Use False (0) to display the normal mouse pointer.
Remarks

For more information on how the action and its argument work, see the action topic.
Example

The following example uses the **Hourglass** method to display an hourglass icon (or another icon you’ve chosen) while your Visual Basic code is executing:

```vba
DoCmd.Hourglass True
```
hWndAccessApp Method

You can use the hWndAccessApp method to determine the handle assigned by Microsoft Windows to the main Microsoft Access window.

expression.hWndAccessApp

expression  Required. An expression that returns one of the objects in the Applies To list.
Remarks

The hWndAccessApp method returns a Long Integer value set by Microsoft Access and is read-only.

You can use this method by using Visual Basic when making calls to Windows application programming interface (API) functions or other external procedures that require a window handle as an argument.

To get the handle to a window containing a Microsoft Access object such as a Form or Report, use the hWnd property.
Show All
HyperlinkPart Method

The **HyperlinkPart** method returns information about data stored as a Hyperlink data type. String.

```expression.HyperlinkPart(Hyperlink, Part)```

**expression**  Required. An expression that returns one of the objects in the Applies To list.

**Hyperlink**  Required **Variant**. A Variant representing the data stored in a Hyperlink field.

**Part**  Optional **AcHyperlinkPart**. The value for the `part` argument is an intrinsic constant representing the information you want returned by the HyperlinkPart method.

AcHyperlinkPart can be one of these AcHyperlinkPart constants.

**acAddress**. The *address* part of a Hyperlink field.

**acDisplayedValue** *default*. The underlined text displayed in a hyperlink.

**acDisplayText**. The *displaytext* part of a Hyperlink field.

**acFullAddress**. The *address* and *subaddress* parts of a Hyperlink field delimited by a "#" character.

**acScreenTip**. The tooltip part of a Hyperlink field.

**acSubAddress**. The *subaddress* part of a Hyperlink field.
Remarks

You use the HyperlinkPart method to return one of three values from a Hyperlink field or the displayed value. The value returned depends on the setting of the part argument. The part argument is optional. If it's not used, the function returns the value Microsoft Access displays for the hyperlink (which corresponds to the acDisplayedValue setting for the part argument). The returned values can be one of the four parts of the Hyperlink field (displaytext, address, subaddress, or screentip), the full address, address#subaddress, or the value Microsoft Access displays for the hyperlink.

Note If you use the HyperlinkPart method in a query, the part argument is required and you can't use the constants listed above but must use the actual value instead.

When a value is provided in the displaytext part of a Hyperlink field, the value displayed by Microsoft Access will be the same as the displaytext setting. When there's no value in the displaytext part of a Hyperlink field, the value displayed will be the address or subaddress part of the Hyperlink field, depending on which value is first present in the field.

The following table shows the values returned by the HyperlinkPart method for data stored in a Hyperlink field.

<table>
<thead>
<tr>
<th>Hyperlink field data</th>
<th>HyperlinkPart method returned values</th>
</tr>
</thead>
<tbody>
<tr>
<td>acDisplayedValue:</td>
<td><a href="http://www.microsoft.com">http://www.microsoft.com</a></td>
</tr>
<tr>
<td>displaytext:</td>
<td></td>
</tr>
<tr>
<td>address:</td>
<td></td>
</tr>
<tr>
<td>subaddress:</td>
<td></td>
</tr>
<tr>
<td>screentip:</td>
<td></td>
</tr>
<tr>
<td>acDisplayText:</td>
<td></td>
</tr>
<tr>
<td>address:</td>
<td></td>
</tr>
<tr>
<td>subaddress:</td>
<td></td>
</tr>
<tr>
<td>acAddress:</td>
<td><a href="http://www.microsoft.com">http://www.microsoft.com</a></td>
</tr>
<tr>
<td>acSubAddress:</td>
<td></td>
</tr>
<tr>
<td>acScreenTip:</td>
<td></td>
</tr>
</tbody>
</table>
Microsoft: Microsoft
http://www.microsoft.com

Customers: Customers
http://www.microsoft.com

**Form Customers#Enter Information**

acFullAddress:  
http://www.microsoft.com

acDisplayedValue: Microsoft

acDisplayText: Microsoft

acAddress:  
http://www.microsoft.com

acSubAddress:

acScreenTip:

acFullAddress:  
http://www.microsoft.com

acDisplayedValue: Customers

acDisplayText: Customers

acAddress:  
http://www.microsoft.com

acSubAddress: Form Customers

acScreenTip:

acFullAddress:  
http://www.microsoft.com#Form Customer

acDisplayedValue: Form Customers

acDisplayText:

acAddress:

acSubAddress: Form Customers
When you add an *address* part to a Hyperlink field by using the **Insert Hyperlink** dialog box (available by clicking **Hyperlink** on the **Insert** menu) or by typing an address part directly into a Hyperlink field, Microsoft Access adds the two # symbols that delimit parts of the hyperlink data.

You can add or edit the *displaytext* part of a hyperlink field by right-clicking a hyperlink in a table, form, or report, pointing to **Hyperlink** on the shortcut menu, and then typing the display text in the **Text to display** box.

When you add data to a Hyperlink field directly, you must include the two # symbols to delimit the parts of the hyperlink data.
Example

The following example uses all four of the part argument constants to display information returned by the HyperlinkPart method for each record in a table containing a Hyperlink field. To try this example, paste the DisplayHyperlinkParts procedure into the Declarations section of a module. You can call the DisplayHyperlinkParts procedure from the Debug window, passing to it the name of a table containing hyperlinks and the name of the field containing Hyperlink data. For example:

:DisplayHyperlinkParts "MyHyperlinkTableName", "MyHyperlinkFieldName"

Public Sub DisplayHyperlinkParts(ByVal strTable As String, _
    ByVal strField As String)
    Dim rst As New ADODB.Recordset
    Dim strMsg As String

    rst.Open strTable, CurrentProject.Connection, _
    adOpenForwardOnly, adLockReadOnly

    ' For each record in table.
    Do Until rst.EOF
        strMsg = "DisplayValue = " _
        & HyperlinkPart(rst(strField), acDisplayedValue) _
        & vbCrLf & "DisplayText = " _
        & HyperlinkPart(rst(strField), acDisplayText) _
        & vbCrLf & "Address = " _
        & HyperlinkPart(rst(strField), acAddress) _
        & vbCrLf & "SubAddress = " _
        & HyperlinkPart(rst(strField), acSubAddress) _
        & vbCrLf & "ScreenTip = " _
        & HyperlinkPart(rst(strField), acScreenTip) _
        & vbCrLf & "Full Address = " _
        & HyperlinkPart(rst(strField), acFullAddress)

        ' Show parts returned by HyperlinkPart function.
        MsgBox strMsg
        rst.MoveNext
    Loop

End Sub
When you use the **HyperlinkPart** method in a query, the *part* argument is required. For example, the following SQL statement uses the **HyperlinkPart** method to return information about data stored as a Hyperlink data type in the URL field of the Links table:

```sql
SELECT Links.URL, HyperlinkPart([URL],0) AS Display, HyperlinkPart([URL],1) AS Name, HyperlinkPart([URL],2) AS Addr, HyperlinkPart([URL],3) AS SubAddr, HyperlinkPart([URL],4) AS ScreenTip
FROM Links
```
Show All
ImportXML Method

The **ImportXML** method allows developers to import XML data and/or schema information into Microsoft SQL Server 2000 Desktop Engine (MSDE 2000), Microsoft SQL Server 7.0 or later, or the Microsoft Jet database engine.

```plaintext
eexpression.ImportXML(DataSource, ImportOptions)
```

**expression**  Required. An expression that returns an **Application** object.

**DataSource**  Required **String**. The name and path of the XML file to import.

**ImportOptions**  Optional **AcImportXMLOption**. The option to use when importing XML files.

**AcImportXMLOption** can be one of these **AcImportXMLOption** constants:

- acAppendData
- acStructureAndData **default**
- acStructureOnly
Example

The following example imports an XML file into a new table named Employees in the current database.

```
Application.ImportXML _
   DataSource:="employees.xml", _
   ImportOptions:=acStructureAndData
```
InsertLines Method

The InsertLines method inserts a line or group of lines of code in a standard module or a class module.

expression.InsertLines(Line, String)

expression  Required. An expression that returns one of the objects in the Applies To list.

Line  Required Long. The number of the line at which to begin inserting.

String  Required String. The text to be inserted into the module.
Remarks

When you use the InsertLines method, any existing code at the line specified by the line argument moves down.

To add multiple lines, include the intrinsic constant vbCrLf at the desired line breaks within the string that makes up the string argument. This constant forces a carriage return and line feed.

Lines in a module are numbered beginning with one. To determine the number of lines in a module, use the CountOfLines property.
Example

The following example creates a new form, adds a command button, and creates a Click event procedure for the command button:

Function ClickEventProc() As Boolean
    Dim frm As Form, ctl As Control, mdl As Module
    Dim lngReturn As Long

    On Error GoTo Error_ClickEventProc
    ' Create new form.
    Set frm = CreateForm
    ' Create command button on form.
    Set ctl = CreateControl(frm.Name, acCommandButton, , , , , 1000, 1000)
    ctl.Caption = "Click here"
    ' Return reference to form module.
    Set mdl = frm.Module
    ' Add event procedure.
    lngReturn = mdl.CreateEventProc("Click", ctl.Name)
    ' Insert text into body of procedure.
    mdl.InsertLines lngReturn + 1, vbTab & "MsgBox ""Way cool!"
    ClickEventProc = True

Exit_ClickEventProc:
    Exit Function

Error_ClickEventProc:
    MsgBox Err & ":" & Err.Description
    ClickEventProc = False
    Resume Exit_ClickEventProc
End Function
InsertText Method

InsertText method as it applies to the Module object.

The InsertText method inserts a specified string of text into a standard module or a class module.

\[\text{expression}.\text{InsertText} (\text{Text})\]

expression Required. An expression that returns one of the above objects.

Text Required String. The text to be inserted into the module.

InsertText method as it applies to the Application object.

The InsertText method inserts a specified string of text into an application.

\[\text{expression}.\text{InsertText} (\text{Text}, \text{ModuleName})\]

expression Required. An expression that returns one of the above objects.

Text Required String. The text to be inserted into the module.

ModuleName Required String. The name of the module for the application.
Remarks

When you insert a string by using the `InsertText` method, Microsoft Access places the new text at the end of the module, after all other procedures.

To add multiple lines, include the intrinsic constant `vbCrLf` at the desired line breaks within the string that makes up the `text` argument. This constant forces a carriage return and line feed.

To specify at which line the text is inserted, use the `InsertLines` method. To insert code into the Declarations section of the module, use the `InsertLines` method rather than the `InsertText` method.

**Note** In previous versions of Microsoft Access, the `InsertText` method was a method of the Application object. You can still use the `InsertText` method of the Application object, but it's recommended that you use the `InsertText` method of the Module object instead.
Example

As it applies to the Module object.

The following example inserts a string of text into a standard module:

Function InsertProc(strModuleName) As Boolean
    Dim mdl As Module, strText As String
    On Error GoTo Error_InsertProc
    ' Open module.
    DoCmd.OpenModule strModuleName
    ' Return reference to Module object.
    Set mdl = Modules(strModuleName)
    ' Initialize string variable.
    strText = "Sub DisplayMessage()" & vbCrLf _
              & vbTab & "MsgBox ""Wild!""" & vbCrLf _
              & "End Sub"
    ' Insert text into module.
    mdl.InsertText strText
    InsertProc = True

Exit_InsertProc:
    Exit Function

Error_InsertProc:
    MsgBox Err & ": " & Err.Description
    InsertProc = False
    Resume Exit_InsertProc
End Function
**IsDependentUpon Method**

Returns a **Boolean** value that indicates whether the specified object is dependent upon the **database object** specified in the **ObjectName** argument.

**expression**: `IsDependentUpon(ObjectType, ObjectName)`

**expression** Required. An expression that returns an **AccessObject** object.

**ObjectType** Required **AcObjectType**. The type of database object to check for dependency.

**AcObjectType** can be one of the following **AcObjectType** constants:
- **acDataAccessPage** Not supported for this method.
- **acDefault**
- **acDiagram** Not supported for this method.
- **acForm**
- **acFunction** Not supported for this method.
- **acMacro** Not supported for this method.
- **acModule** Not supported for this method.
- **acQuery**
- **acReport**
- **acServerView** Not supported for this method.
- **acStoredProcedure** Not supported for this method.
- **acTable**

**ObjectName** Required **String**. The name of the database object to check for dependency.
Remarks

This method will return a run-time error if any of the following conditions are true:

- The **AccessObject** object specified in the **ObjectType** argument is an unsupported type.
- The **Track name AutoCorrect info** setting (Tools menu, Options dialog box, General tab) is disabled. You can use the following code to enable the **Track name AutoCorrect info** setting and update the dependency information for all of the objects in the database:
  
  ```vba
  Application.SetOption "Track Name AutoCorrect Info", 1
  ```

- You have insufficient permissions to check the dependency information for the specified **AccessObject** object.
- The database object specified in the **ObjectName** argument cannot be found.
- This method is being called from an **Access project (.adp)**.
Item Method

The Item method returns a specific member of a collection either by position or by key. Reference object.

equation of expression.Item(var)

expression  Required. An expression that returns one of the objects in the Applies To list.

var  Required Variant. An expression that specifies the position of a member of the collection referred to by the expression argument. If a numeric expression, the var argument must be a number from 1 to the value of the collection's Count property. If a string expression, the var argument must be the name of a member of the collection.
Remarks

If the value provided for the var argument doesn't match any existing member of the collection, an error occurs.

The Item method is the default member of the References collection, so you don't have to specify it explicitly. For example, the following two lines of code are equivalent:

Debug.Print References(1).Name

Debug.Print References.Item(1).Name
**Line Method**

The **Line** method draws lines and rectangles on a **Report** object when the **Print** event occurs.

*expression*.**Line**(flags, x1, y1, x2, y2, color)

*expression*  Required. An expression that returns one of the objects in the Applies To list.

**flags**  Required **Integer**.

*x1*  Required **Single**. The value indicating the coordinate of the starting point for the line or rectangle. The Scale properties (**ScaleMode**, **ScaleLeft**, **ScaleTop**, **ScaleHeight**, and **ScaleWidth**) of the **Report** object specified by the **object** argument determine the unit of measure used. If this argument is omitted, the line begins at the position indicated by the **CurrentX** property.

*y1*  Required **Single**. The value indicating the coordinate of the starting point for the line or rectangle. The Scale properties (**ScaleMode**, **ScaleLeft**, **ScaleTop**, **ScaleHeight**, and **ScaleWidth**) of the **Report** object specified by the **object** argument determine the unit of measure used. If this argument is omitted, the line begins at the position indicated by the **CurrentY** property.

*x2*  Required **Single**. The value indicating the coordinate of the end point for the line to draw. This argument is required.

*y2*  Required **Single**. The value indicating the coordinate of the end point for the line to draw. This argument is required.

**color**  Required **Long**. The value indicating the RGB (red-green-blue) color used to draw the line. If this argument is omitted, the value of the **ForeColor** property is used. You can also use the **RGB** function or **QBColor** function to specify the color.
Remarks

You can use this method only in an event procedure or a macro specified by the OnPrint or OnFormat event property for a report section, or the OnPage event property for a report.

To connect two drawing lines, make sure that one line begins at the end point of the previous line.

The width of the line drawn depends on the DrawWidth property setting. The way a line or rectangle is drawn on the background depends on the settings of the DrawMode and DrawStyle properties.

When you apply the Line method, the CurrentX and CurrentY properties are set to the end point specified by the x2 and y2 arguments.
**Example**

The following example uses the **Line** method to draw a red rectangle five pixels inside the edge of a report named EmployeeReport. The **RGB** function is used to make the line red.

To try this example in Microsoft Access, create a new report named EmployeeReport. Paste the following code in the declarations section of the report's module, then switch to Print Preview.

```vba
Private Sub Detail_Print(Cancel As Integer, PrintCount As Integer)
    ' Call the DrawLine procedure
    DrawLine
End Sub

Sub DrawLine()
    Dim rpt As Report, lngColor As Long
    Dim sngTop As Single, sngLeft As Single
    Dim sngWidth As Single, sngHeight As Single

    Set rpt = Reports!EmployeeReport
    ' Set scale to pixels.
    rpt.ScaleMode = 3
    ' Top inside edge.
    sngTop = rpt.ScaleTop + 5
    ' Left inside edge.
    sngLeft = rpt.ScaleLeft + 5
    ' Width inside edge.
    sngWidth = rpt.ScaleWidth - 10
    ' Height inside edge.
    sngHeight = rpt.ScaleHeight - 10
    ' Make color red.
    lngColor = RGB(255, 0, 0)
    ' Draw line as a box.
    rpt.Line(sngTop, sngLeft) - (sngWidth, sngHeight), lngColor, B
End Sub
```
LoadPicture Method

The **LoadPicture** method loads a graphic into an [ActiveX control](https://msdn.microsoft.com/en-us/library/aa518206(v=vs.60).aspx).

`expression.LoadPicture(FileNamename)`

*expression*  Required. An expression that returns one of the objects in the Applies To list.

*FileName*  Required [String](https://msdn.microsoft.com/en-us/library/aa518206(v=vs.60).aspx). The file name of the graphic to be loaded. The graphic can be a bitmap file (.bmp), icon file (.ico), run-length encoded file (.rle), or metafile (.wmf).
Remarks

Assign the return value of the `LoadPicture` method to the `Picture` property of an ActiveX control to dynamically load a graphic into the control. The following example loads a bitmap into a control called OLECustomControl on an Orders form:

```vba
Set Forms!Orders!OLECustomControl.Picture = _
   LoadPicture("Stars.bmp")
```

The `LoadPicture` method returns an object of type `Picture`. You can assign this value to a variable of type `Object` by using the `Set` statement.

The `Picture` object is not a Microsoft Access object, but it is available to procedures in Microsoft Access.

**Note** You can't use the `LoadPicture` method to set the `Picture` property of an image control. This method works with ActiveX controls only. To set the `Picture` property of an image control, simply assign to it a `string` specifying the file name and path of the desired graphic.
Maximize Method

The **Maximize** method carries out the **Maximize** action in Visual Basic.

`expression.Maximize`

*expression* Required. An expression that returns a **DoCmd** object.
Remarks

This method has no arguments and can be called directly using the syntax `DoCmd.Maximize`.

**Note** This method cannot be applied to module windows in the Visual Basic Editor (VBE). For information about how to affect module windows see the `WindowState` property topic.
Minimize Method

The **Minimize** method carries out the [Minimize](#) action in Visual Basic.

*expression*.**Minimize**

*expression*  Required. An expression that returns a **Docmd** object.
Remarks

This method has no arguments and be called directly using the syntax DoCmd.Minimize.

Note This method cannot be applied to module windows in the Visual Basic Editor (VBE). For information about how to affect module windows see the WindowState property topic.
Modify Method

You can use the Modify method to change the format conditions of a FormatCondition object in the FormatConditions collection of a combo box or text box control.

expression.Modify(Type, Operator, Expression1, Expression2)

expression Required. An expression that returns one of the objects in the Applies To list.

Type Required AcFormatConditionType.

AcFormatConditionType can be one of these AcFormatConditionType constants.

acExpression
acFieldHasFocus
acFieldValue

Operator Optional AcFormatConditionOperator.

AcFormatConditionOperator can be one of these AcFormatConditionOperator constants.

acBetween default
acEqual
acGreaterThan
acGreaterThanOrEqual
acLessThan
acLessThanOrEqual
acNotBetween
acNotEqual

If the type argument is acExpression, the operator argument is ignored. If you leave this argument blank, the default constant (acBetween) is assumed.
Expression1  Optional Variant. A Variant value or expression associated with the first part of the conditional format. Can be a constant value or a string value.

Expression2  Optional Variant. A Variant value or expression associated with the second part of the conditional format when the operator argument is acBetween or acNotBetween (otherwise, this argument is ignored). Can be a constant value or a string value.
Move Method

Moves the specified object to the coordinates specified by the argument values.

\textit{expression}.\texttt{Move(Left, Top, Width, Height)}

\textit{expression} Required. An expression that returns one of the objects in the Applies To list.

\textit{Left} Required \texttt{Variant}. The screen position in twips for the left edge of the object relative to the left edge of the Microsoft Access window.

\textit{Top} Optional \texttt{Variant}. The screen position in twips for the top edge of the object relative to the top edge of the Microsoft Access window.

\textit{Width} Optional \texttt{Variant}. The desired width in twips of the object.

\textit{Height} Optional \texttt{Variant}. The desired height in twips of the object.
Remarks

Only the *Left* argument is required. However, to specify any other arguments, you must specify all the arguments that precede it. For example, you cannot specify *Width* without specifying *Left* and *Top*. Any trailing arguments that are unspecified remain unchanged.

This method overrides the *Moveable* property.

If a form or report is modal, it is still positioned relative to the Access window, but the values for *Left* and *Top* can be negative.

In Datasheet View or Print Preview, changes made using the *Move* method are saved if the user explicitly saves the database, but Access does not prompt the user to save such changes.
**Example**

The following example determines whether or not the first form in the current project can be moved; if it can, the example moves the form.

If Forms(0).Moveable Then
    Forms(0).Move _
        Left:=0, Top:=0, Width:=400, Height:=300
Else
    MsgBox "The form cannot be moved."
End If
MoveSize Method

The MoveSize method carries out the MoveSize action in Visual Basic.

expression.MoveSize(Right, Down, Width, Height)

expression  Required. An expression that returns one of the objects in the Applies To list.

Right  Optional Variant. A numeric expression.

Down  Optional Variant. A numeric expression.

Width  Optional Variant. A numeric expression.

Height  Optional Variant. A numeric expression.
Remarks

For more information on how the action and its arguments work, see the action topic.

You must include at least one argument for the **MoveSize** method. If you leave an argument blank, the current setting for the window is used.

You can leave an optional argument blank in the middle of the syntax, but you must include the argument's comma. If you leave one or more trailing arguments blank, don't use a comma following the last argument you specify.

The units for the arguments are **twips**.
Example

The following example moves the active window and changes its height, but leaves its width unchanged:

DoCmd.MoveSize 1440, 2400, , 2000
NewAccessProject Method

You can use the NewAccessProject method to create and open a new Microsoft Access project (.adp) as the current Access project in the Microsoft Access window.

expression.NewAccessProject(filepath, Connect)

expression Required. An expression that returns one of the objects in the Applies To list.

filepath Required String. A string expression that is the name of the new Access project, including the path name and the file name extension. If your network supports it, you can also specify a network path in the following form: \\Server\Share\Folder\Filename .adp

Connect Optional Variant. A string expression that's the valid connection string for the Access project. See the ADO ConnectionString property for details about this string.
Remarks

The **NewAccessProject** method enables you to create a new Access project from within Microsoft Access or another application through Automation, formally called OLE Automation. For example, you can use the **NewAccessProject** method from Microsoft Excel to create a new Access project in the Access window. Once you have created an instance of Microsoft Access from another application, you must also create a new Access project. This Access project opens in the Microsoft Access window.

If the Access project identified by *projname* already exists, an error occurs.

The new Access project is opened under the **Admin user account**.

**Note**  To open an **Access database** (.mdb), use the **NewCurrentDatabase** method of the **Application** object.
NewCurrentDatabase Method

You can use the NewCurrentDatabase method to create a new Microsoft Access database (.mdb) in the Microsoft Access window.

expression.NewCurrentDatabase(filepath)

expression   Required. An expression that returns one of the objects in the Applies To list.

filepath   Required String. A string expression that is the name of a new database file, including the path name and the file name extension. If your network supports it, you can also specify a network path in the following form: \Server\Share\Folder\Filename

Note   If you don't supply the filename extension, .mdb is appended to the filename
Remarks

You can use this method to create a new database from another application that is controlling Microsoft Access through Automation, formerly called OLE Automation. For example, you can use the `NewCurrentDatabase` method from Microsoft Excel to create a new database in the Microsoft Access window.

**Note** You can use the `NewAccessProject` method to create a new Microsoft Access project (.adp) in the Access window.

The `NewCurrentDatabase` method enables you to create a new Microsoft Access database from another application through Automation. Once you have created an instance of Microsoft Access from another application, you must also create a new database. This database opens in the Microsoft Access window.

If the database identified by `dbname` already exists, an error occurs.

The new database is opened under the `Admin user account`. 
Example

The following example creates a new Microsoft Access database from another application through Automation, and then creates a new table in that database.

You can enter this code in a Visual Basic module in any application that can act as a COM component. For example, you might run the following code from Microsoft Excel, Microsoft Visual Basic, or Microsoft Access.

When the variable pointing to the Application object goes out of scope, the instance of Microsoft Access that it represents closes as well. Therefore, you should declare this variable at the module level.

' Include following in Declarations section of module.
Dim appAccess As Access.Application

Sub NewAccessDatabase()
    Dim dbs As Object, tdf As Object, fld As Variant
    Dim strDB As String
    Const DB_Text As Long = 10
    Const FldLen As Integer = 40

    ' Initialize string to database path.
    strDB = "C:\My Documents\Newdb.mdb"
    ' Create new instance of Microsoft Access.
    Set appAccess = _
    CreateObject("Access.Application.9")
    ' Open database in Microsoft Access window.
    appAccess.NewCurrentDatabase strDB
    ' Get Database object variable.
    Set dbs = appAccess.CurrentDb
    ' Create new table.
    Set tdf = dbs.CreateTableDef("Contacts")
    ' Create field in new table.
    Set fld = tdf._
    CreateField("CompanyName", DB_Text, FldLen)
    ' Append Field and TableDef objects.
    tdf.Fields.Append fld
dbs.TableDefs.Append tdf
    Set appAccess = Nothing
End Sub
Nz Function

You can use the Nz function to return zero, a zero-length string (" "), or another specified value when a Variant is Null. For example, you can use this function to convert a Null value to another value and prevent it from propagating through an expression.

Nz(variant, [valueifnull])

The Nz function has the following arguments.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>variant</td>
<td>A variable of data type Variant. Optional (unless used in a query). A Variant that supplies a value to be returned if the variant argument is Null. This argument enables you to return a value other than zero or a zero-length string.</td>
</tr>
<tr>
<td>valueifnull</td>
<td>Note If you use the Nz function in an expression in a query without using the valueifnull argument, the results will be a zero-length string in the fields that contain null values.</td>
</tr>
</tbody>
</table>

If the value of the variant argument is Null, the Nz function returns the number zero or a zero-length string (always returns a zero-length string when used in a query expression), depending on whether the context indicates the value should be a number or a string. If the optional valueifnull argument is included, then the Nz function will return the value specified by that argument if the variant argument is Null. When used in a query expression, the NZ function should always include the valueifnull argument,

If the value of variant isn't Null, then the Nz function returns the value of variant.
Remarks

The **Nz** function is useful for expressions that may include **Null** values. To force an expression to evaluate to a non-**Null** value even when it contains a **Null** value, use the **Nz** function to return zero, a zero-length string, or a custom return value.

For example, the expression $2 + \text{varX}$ will always return a **Null** value when the **Variant** varX is **Null**. However, $2 + \text{Nz(varX)}$ returns 2.

You can often use the **Nz** function as an alternative to the **IIf** function. For example, in the following code, two expressions including the **IIf** function are necessary to return the desired result. The first expression including the **IIf** function is used to check the value of a variable and convert it to zero if it is **Null**.

```plaintext
varTemp = IIf(IsNull(varFreight), 0, varFreight)
varResult = IIf(varTemp > 50, "High", "Low")
```

In the next example, the **Nz** function provides the same functionality as the first expression, and the desired result is achieved in one step rather than two.

```plaintext
varResult = IIf(Nz(varFreight) > 50, "High", "Low")
```

If you supply a value for the optional argument **valueifnull**, that value will be returned when **variant** is **Null**. By including this optional argument, you may be able to avoid the use of an expression containing the **IIf** function. For example, the following expression uses the **IIf** function to return a string if the value of varFreight is **Null**.

```plaintext
varResult = IIf(IsNull(varFreight), "No Freight Charge", varFreight)
```

In the next example, the optional argument supplied to the **Nz** function provides the string to be returned if varFreight is **Null**.

```plaintext
varResult = Nz(varFreight, "No Freight Charge")
```
Example

The following example evaluates a control on a form and returns one of two strings based on the control's value. If the value of the control is **Null**, the procedure uses the **Nz** function to convert a **Null** value to a zero-length string.

```vba
Public Sub CheckValue()
    Dim frm As Form
    Dim ctl As Control
    Dim varResult As Variant

    ' Return Form object variable pointing to Orders form.
    Set frm = Forms!Orders

    ' Return Control object variable pointing to ShipRegion.
    Set ctl = frm!ShipRegion

    ' Choose result based on value of control.
    varResult = IIf(Nz(ctl.Value) = vbNullString, _
                   "No value.", "Value is " & ctl.Value & ")."

    ' Display result.
    MsgBox varResult, vbExclamation

End Sub
```
OpenAccessProject Method

You can use the OpenAccessProject method to open an existing Microsoft Access project (.adp) as the current Access project in the Microsoft Access window.

expression. OpenAccessProject(filepath, Exclusive)

expression  Required. An expression that returns one of the objects in the Applies To list.

filepath  Required String. A string expression that is the name of the existing Access project, including the path name and the file name extension. If your network supports it, you can also specify a network path in the following form: //Server/Share/Folder/Filename.adp

Note  If you don't supply the filename extension, .adp is appended to the filename. You can use this method or the OpenCurrentDatabase method to open .adp files.

Exclusive  Optional Boolean.
Remarks

The **OpenAccessProject** method enables you to open an existing project from within Microsoft Access or another application through Automation, formally called OLE Automation. For example, you can use the **OpenAccessProject** method from Microsoft Excel to open the Northwind.adp sample database in the Microsoft Access window. Once you have created an instance of Microsoft Access from another application, you must also create a new Access project or specify a particular Access project to open. This Access project opens in the Microsoft Access window.

If you have already opened a project and wish to open another project in the Microsoft Access window, you can use the **CloseCurrentDatabase** method to close the first Access project before opening another.

**Note** To open an **Access database** (.mdb), use the **OpenCurrentDatabase** method of the **Application** object.
OpenConnection Method

You can use the OpenConnection method to open an ADO connection to an existing Microsoft Access project (.adp) or Access database (.mdb) as the current Access project or database in the Microsoft Access window.

expression.OpenConnection(BaseConnectionString, UserID, Password)

expression Required. An expression that returns one of the objects in the Applies To list.

BaseConnectionString Optional Variant. A string expression that is the base connection string of the database.

UserID Optional Variant. A string expression that is the name of the existing Access project, including the path name and the file name extension. If your network supports it, you can also specify a network path in the following form: \Server\Share\Folder\Filename .adp

Password Optional Variant. Note If you don't supply the filename extension, .adp is appended to the filename. You can use this method or the OpenCurrentDatabase method to open .adp files.
Remarks

The **OpenConnection** method is similar to the **Open** method of an ADO **Connection** object. This method establishes the physical connection to the data source. After this method successfully completes, the connection is live, the **Connection** and **BaseConnectionString** properties are set, and the Database window or data access page should be repopulated with data from the new connection. All parameters of this method are optional. If no base connection string is supplied, then the connection is re-established using the previous base connection string (but the user must call **CloseConnection** before calling **OpenConnection** again). In the case of an Access project, the **BaseConnectionString** property can only specify the SQL Server OLE DB Provider.
OpenCurrentDatabase Method

You can use the **OpenCurrentDatabase** method to open an existing Microsoft Access database (.mdb) as the current database.

```
expression.OpenCurrentDatabase(filepath, Exclusive, bstrPassword)
```

**expression** Required. An expression that returns one of the objects in the Applies To list.

**filepath** Required **String**. A string expression that is the name of an existing database file, including the path name and the file name extension. If your network supports it, you can also specify a network path in the following form: `\Server \Share\Folder\Filename`

**Note** If you don't supply the filename extension, .mdb is appended to the filename.

**Exclusive** Optional **Boolean**. Specifies whether you want to open the database in **exclusive** mode. The default value is **False**, which specifies that the database should be opened in shared mode.

**bstrPassword** Optional **String**. The password to open the specified database.
Remarks

You can use this method to open a database from another application that is controlling Microsoft Access through Automation, formerly called OLE Automation. For example, you can use the OpenCurrentDatabase method from Microsoft Excel to open the Northwind.mdb sample database in the Microsoft Access window. Once you have created an instance of Microsoft Access from another application, you must also create a new database or specify a particular database to open. This database opens in the Microsoft Access window.

Note  Use the OpenAccessProject method to open an existing Microsoft Access project (.adp) as the current database.

If you have already opened a database and wish to open another database in the Microsoft Access window, you can use the CloseCurrentDatabase method to close the first database before opening another.

Set the Exclusive argument to True to open the database in exclusive mode. If you omit this argument, the database will open in shared mode.

Note  Don't confuse the OpenCurrentDatabase method with the ActiveX Data Objects (ADO) Open method or the Data Access Object (DAO) OpenDatabase method. The OpenCurrentDatabase method opens a database in the Microsoft Access window. The ADO Open method returns a Connection object variable, and the DAO OpenDatabase method returns a Database object variable, both of which represent a particular database but don't actually open that database in the Microsoft Access window.
Example

The following example opens a Microsoft Access database from another application through Automation and then opens a form in that database.

You can enter this code in a Visual Basic module in any application that can act as a COM component. For example, you might run the following code from Microsoft Excel, Microsoft Visual Basic, or Microsoft Access.

When the variable pointing to the Application object goes out of scope, the instance of Microsoft Access that it represents closes as well. Therefore, you should declare this variable at the module level.

' Include the following in Declarations section of module.
Dim appAccess As Access.Application

Sub DisplayForm()

    Dim strDB as String

    ' Initialize string to database path.
    Const strConPathToSamples = "C:\Program " & "Files\Microsoft Office\Office11\Samples\"
    strDB = strConPathToSamples & "Northwind.mdb"
    ' Create new instance of Microsoft Access.
    Set appAccess = _
        CreateObject("Access.Application")
    ' Open database in Microsoft Access window.
    appAccess.OpenCurrentDatabase strDB
    ' Open Orders form.
    appAccess.DoCmd.OpenForm "Orders"
End Sub
OpenDataAccessPage Method

The OpenDataAccessPage method carries out the OpenDataAccessPage action in Visual Basic.

\[ \text{expression.} \text{OpenDataAccessPage(DataAccessPageName, View)} \]

\textit{expression} Required. An expression that returns one of the objects in the Applies To list.

\textit{DataAccessPageName} Required Variant. A string expression that's the valid name of a data access page in the current database. If you execute Visual Basic code containing the OpenDataAccessPage method in a library database, Microsoft Access looks for the form with this name, first in the library database, then in the current database.

\textit{View} Optional \textit{AcDataAccessPageView}. The view in which to open the data access page.

\textit{AcDataAccessPageView} can be one of these AcDataAccessPageView constants.

\textit{acDataAccessPageBrowse} default Opens the form in Page view.
\textit{acDataAccessPageDesign} Opens the form in Design view.
Remarks

For more information on how the action and its arguments work, see the action topic.

If the connection password was not contained in the connection information for the data access page when it was created, the user is prompted to provide a password when the OpenDataAccessPage method is used.
Example

The following example opens the Employees data access page in Design view.

DoCmd.OpenDataAccessPage "Employees", acDataAccessPageDesign
OpenDiagram Method

The OpenDiagram method carries out the OpenDiagram action in Visual Basic.

expression. OpenDiagram(DiagramName)

expression  Required. An expression that returns a DoCmd object.

DiagramName  Required Variant. A string expression that's the valid name of a database diagram in the current database. If you execute Visual Basic code containing the OpenDiagram method in a library database, Microsoft Access looks for the database diagram with this name first in the library database, then in the current database.
Example

The following example opens the database diagram named "Data Model".

DoCmd.**OpenDiagram** " Data Model"
OpenForm Method

The OpenForm method carries out the **OpenForm** action in Visual Basic.

expression.**OpenForm**(/FormName, View, FilterName, WhereCondition, DataMode, WindowMode, OpenArgs)

expression  Required. An expression that returns one of the objects in the Applies To list.

**FormName**  Required **Variant**. A **string expression** that's the valid name of a form in the current database. If you execute Visual Basic code containing the OpenForm method in a **library database**, Microsoft Access looks for the form with this name first in the library database, then in the current database.

**View**  Optional **AcFormView**.

AcFormView can be one of these AcFormView constants.

- **acDesign**
- **acFormDS**
- **acFormPivotChart**
- **acFormPivotTable**
- **acNormal default**. Opens the form in **Form view**.
- **acPreview**

If you leave this argument blank, the default constant (**acNormal**) is assumed.

**FilterName**  Optional **Variant**. A string expression that's the valid name of a **query** in the current database.

**WhereCondition**  Optional **Variant**. A string expression that's a valid SQL **WHERE clause** without the word WHERE.

**DataMode**  Optional **AcFormOpenDataMode**. The data entry mode for the form. This applies only to forms opened in Form view or Datasheet view.
AcFormOpenDataMode can be one of these AcFormOpenDataMode constants.

**acFormAdd** The user can add new records but can't edit existing records.

**acFormEdit** The user can edit existing records and add new records.

**acFormPropertySettings** default

**acFormReadOnly** The user can only view records.

If you leave this argument blank (the default constant, **acFormPropertySettings**, is assumed), Microsoft Access opens the form in the data mode set by the form's **AllowEdits**, **AllowDeletions**, **AllowAdditions**, and **DataEntry** properties.

**WindowMode** Optional AcWindowMode. The window mode in which the form opens.

AcWindowMode can be one of these AcWindowMode constants.

**acDialog** The form's Modal and PopUp properties are set to Yes.

**acHidden** The form is hidden.

**acIcon** The form opens minimized in the Windows taskbar.

**acWindowNormal** default The form is in the mode set by its properties.

If you leave this argument blank, the default constant (**acWindowNormal**) is assumed.

**OpenArgs** Optional Variant. A string expression. This expression is used to set the form's **OpenArgs** property. This setting can then be used by code in a form module, such as the Open event procedure. The **OpenArgs** property can also be referred to in macros and expressions.

For example, suppose that the form you open is a continuous-form list of clients. If you want the focus to move to a specific client record when the form opens, you can specify the client name with the **openargs** argument, and then use the **FindRecord** method to move the focus to the record for the client with the specified name.

This argument is available only in Visual Basic.
Remarks

For more information on how the action and its arguments work, see the action topic.

The maximum length of the `wherecondition` argument is 32,768 characters (unlike the Where Condition action argument in the Macro window, whose maximum length is 256 characters).

You can leave an optional argument blank in the middle of the syntax, but you must include the argument's comma. If you leave a trailing argument blank, don't use a comma following the last argument you specify.
Example

The following example opens the Employees form in Form view and displays only records with King in the LastName field. The displayed records can be edited, and new records can be added.

DoCmd.OpenForm "Employees", ,"LastName = 'King'"
OpenFunction Method

Opens a user-defined function in a Microsoft SQL Server database for viewing in Microsoft Access.

expression.OpenFunction(FunctionName, View, DataMode)

expression   Required. An expression that returns a DoCmd object.

FunctionName Required Variant. The name of the function to open.

View   Optional AcView. The view in which to open the function.

AcView can be one of these AcView constants.

acViewDesign Opens the function in Design View.
acViewNormal default Opens the function in Datasheet View.
acViewPivotChart Opens the function in PivotChart View.
acViewPivotTable Opens the function in PivotTable View.
acViewPreview Opens the function in Print Preview.

DataMode   Optional AcOpenDataMode. The mode in which to open the function.

AcOpenDataMode can be one of these AcOpenDataMode constants.

acAdd Opens the function for data entry.
acEdit default Opens the function for updating existing data.
acReadOnly Opens the function in read-only mode.
Remarks

Use the **AllFunctions** collection to retrieve information about the available user-defined functions in a SQL Server database.
Example

The following example opens the first user-defined function in the current database in Design View and read-only mode.

Dim objFunction As AccessObject
Dim strFunction As String

Set objFunction = Application.AllFunctions(0)

DoCmd.**OpenFunction** FunctionName:=objFunction.Name, _
       View:=acViewDesign, Mode:=acReadOnly
OpenModule Method

The OpenModule method carries out the OpenModule action in Visual Basic.

\[ \text{expression}.\text{OpenModule}(\text{ModuleName}, \text{ProcedureName}) \]

\text{expression} Required. An expression that returns one of the objects in the Applies To list.

\text{ModuleName} Optional Variant. A string expression that's the valid name of the Visual Basic module you want to open. If you leave this argument blank, Microsoft Access searches all the standard modules in the database for the procedure you selected with the \text{procedurename} argument and opens the module containing the procedure to that procedure. If you execute Visual Basic code containing the OpenModule method in a library database, Microsoft Access looks for the module with this name first in the library database, then in the current database.

\text{ProcedureName} Optional Variant. A string expression that's the valid name for the procedure you want to open the module to. If you leave this argument blank, the module opens to the Declarations section.
Remarks

You must include at least one of the two OpenModule action arguments. If you enter a value for both arguments, Microsoft Access opens the specified module at the specified procedure.

If you leave the *procedurename* argument blank, don't use a comma following the *modulename* argument.
**Example**

The following example opens the Utility Functions module to the IsLoaded() function procedure:

```vba
DoCmd.OpenModule "Utility Functions", "IsLoaded"
```
OpenQuery Method

The **OpenQuery** method carries out the **OpenQuery** action in Visual Basic.

```
expression.OpenQuery(QueryName, View, DataMode)
```

*expression* Required. An expression that returns one of the objects in the Applies To list.

**QueryName** Required **Variant**. A **string expression** that's the valid name of a **query** in the current database. If you execute Visual Basic code containing the **OpenQuery** method in a **library database**, Microsoft Access looks for the query with this name first in the library database, then in the current database.

**View** Optional **AcView**.

AcView can be one of these AcView constants.

- **acViewDesign**
- **acViewNormal** default
- **acViewPivotChart**
- **acViewPivotTable**
- **acViewPreview**

If the *queryname* argument is the name of a **select**, **crosstab**, **union**, or **pass-through** query whose **ReturnsRecords** property is set to -1, **acViewNormal** displays the query’s **result set**. If the *queryname* argument refers to an **action**, **data-definition**, or pass-through query whose **ReturnsRecords** property is set to 0, **acViewNormal** runs the query.

If you leave this argument blank, the default constant (**acViewNormal**) is assumed.

**DataMode** Optional **AcOpenDataMode**.

AcOpenDataMode can be one of these AcOpenDataMode constants.
acAdd
acEdit default
acReadOnly

If you leave this argument blank, the default constant (acEdit) is assumed.
Remarks

For more information on how the action and its arguments work, see the action topic.

**Note** This method is only available in the [Microsoft Access database](https://support.office.com/en-us/article/OpenView-or-OpenStoredProcedure-7a8f5f0d-c9de-4a3f-9304-6f0e17f50c7c) environment (.mdb). See the [OpenView](https://support.office.com/en-us/article/OpenView-or-OpenStoredProcedure-7a8f5f0d-c9de-4a3f-9304-6f0e17f50c7c) or [OpenStoredProcedure](https://support.office.com/en-us/article/OpenView-or-OpenStoredProcedure-7a8f5f0d-c9de-4a3f-9304-6f0e17f50c7c) methods if using the Microsoft Access Project environment (.adp).

If you specify the *datamode* argument and leave the *view* argument blank, you must include the *view* argument's comma. If you leave a trailing argument blank, don't use a comma following the last argument you specify.
Example

The following example opens Sales Totals Query in Datasheet view and enables the user to view but not to edit or add records:

```
DoCmd.OpenQuery "Sales Totals Query", , acReadOnly
```
OpenReport Method

The **OpenReport** method carries out the **OpenReport** action in Visual Basic.

```
expression.OpenReport(ReportName, View, FilterName, WhereCondition, WindowMode, OpenArgs)
```

*expression* Required. An expression that returns a **DoCmd** object.

**ReportName** Required **Variant**. A string expression that's the valid name of a report in the current database. If you execute Visual Basic code containing the **OpenReport** method in a library database, Microsoft Access looks for the report with this name, first in the library database, then in the current database.

**View** Optional **AcView**. The view to apply to the specified report.

AcView can be one of these AcView constants.

- **acViewDesign**
- **acViewNormal** default Prints the report immediately.
- **acViewPivotChart** Not supported.
- **acViewPivotTable** Not supported.
- **acViewPreview**

**FilterName** Optional **Variant**. A string expression that's the valid name of a query in the current database.

**WhereCondition** Optional **Variant**. A string expression that's a valid SQL **WHERE** clause without the word WHERE.

**WindowMode** Optional **AcWindowMode**. The mode in which the form opens.

AcWindowMode can be one of these AcWindowMode constants.

- **acDialog** The form's Modal and PopUp properties are set to Yes.
- **acHidden** The form is hidden.
- **acIcon** The form opens minimized in the Windows taskbar.
**acWindowNormal** default The form is in the mode set by its properties.

**OpenArgs** Optional **Variant**. Sets the **OpenArgs** property.
Remarks

For more information on how the action and its arguments work, see the action topic.

The maximum length of the *WhereCondition* argument is 32,768 characters (unlike the Where Condition action argument in the Macro window, whose maximum length is 256 characters).

You can leave an optional argument blank in the middle of the syntax, but you must include the argument's comma. If you leave one or more trailing arguments blank, don't use a comma following the last argument you specify.
Example

The following example prints Sales Report while using the existing query Report Filter.

OpenStoredProcedure Method

The **OpenView** method carries out the **OpenStoredProcedure** action in Visual Basic.

`expression.OpenStoredProcedure(ProcedureName, View, DataMode)`

**expression**  Required. An expression that returns one of the objects in the Applies To list.

**ProcedureName**  Required **Variant**. A **string expression** that's the valid name of a [stored procedure](https://example.com) in the current database. If you execute Visual Basic code containing the **OpenStoredProcedure** method in a [library database](https://example.com), Microsoft Access looks for the stored procedure with this name first in the library database, then in the current database.

**View**  Optional **AcView**.

AcView can be one of these **AcView** constants.

- `acViewDesign`
- `acViewNormal` **default**
- `acViewPivotChart`
- `acViewPivotTable`
- `acViewPreview`

If you leave this argument blank, the default constant (**acViewNormal**) is assumed.

**DataMode**  Optional **AcOpenDataMode**.

AcOpenDataMode can be one of these **AcOpenDataMode** constants.

- `acAdd`
- `acEdit` **default**
- `acReadOnly`
If you leave this argument blank, the default constant \((\text{acEdit})\) is assumed.
Remarks

For more information on how the action and its arguments work, see the action topic.
Example

The following example opens the Employees stored procedure in Design view.

`DoCmd.OpenStoredProcedure "Employees", 1`
OpenTable Method

The **OpenTable** method carries out the **OpenTable** action in Visual Basic.

\[expression.\text{OpenTable}(TableName, \text{View}, DataMode)\]

*expression* Required. An expression that returns one of the objects in the Applies To list.

**TableName** Required **Variant**. A **string expression** that's the valid name of a table in the current database. If you execute Visual Basic code containing the **OpenTable** method in a **library database**, Microsoft Access looks for the table with this name first in the library database, then in the current database.

**View** Optional **AcView**.

AcView can be one of these AcView constants.

**acViewDesign**
**acViewNormal** *default*. Opens the table in **Datasheet view**.
**acViewPivotChart**
**acViewPivotTable**
**acViewPreview**

If you leave this argument blank, the default constant (**acViewNormal**) is assumed.

**DataMode** Optional **AcOpenDataMode**.

AcOpenDataMode can be one of these AcOpenDataMode constants.

**acAdd**
**acEdit** *default*
**acReadOnly**

If you leave this argument blank, the default constant (**acEdit**) is assumed.
Remarks

For more information on how the action and its arguments work, see the action topic.

If you specify the datamode argument and leave the view argument blank, you must include the view argument's comma. If you leave a trailing argument blank, don't use a comma following the last argument you specify.
Example

The following example opens the Employees table in Print Preview:

DoCmd.**OpenTable** "Employees", acViewPreview
OpenView Method

The **OpenView** method carries out the **OpenView** action in Visual Basic.

`expression.OpenView(ViewName, View, DataMode)`

- **expression** Required. An expression that returns one of the objects in the Applies To list.

  - **ViewName** Required **Variant**. A **string expression** that's the valid name of a view in the current database. If you execute Visual Basic code containing the **OpenView** method in a library database, Microsoft Access looks for the view with this name first in the library database, then in the current database.

  - **View** Optional **AcView**.

    AcView can be one of these AcView constants.
    - **acViewDesign**
    - **acViewNormal** default
    - **acViewPivotChart**
    - **acViewPivotTable**
    - **acViewPreview**

    If you leave this argument blank, the default constant (**acViewNormal**) is assumed.

  - **DataMode** Optional **AcOpenDataMode**.

    AcOpenDataMode can be one of these AcOpenDataMode constants.
    - **acAdd**
    - **acEdit** default
    - **acReadOnly**
    - **acViewPreview**

    If you leave this argument blank, the default constant (**acEdit**) is assumed.
Remarks

For more information on how the action and its arguments work, see the action topic.
Example

The following example opens the Employees view.

DoCmd.OpenView "Employees"
**OutputTo Method**

The **OutputTo** method carries out the **OutputTo** action in Visual Basic.

```vbscript
expression.OutputTo(ObjectType, ObjectName, OutputFormat, OutputFile, AutoStart, TemplateFile, Encoding)
```

*expression* Required. An expression that returns a **DoCmd** object.

**ObjectType** Required **AcOutputObjectType**. The type of object to output.

AcOutputObjectType can be one of these AcOutputObjectType constants.

- **acOutputDataAccessPage** Not supported.
- **acOutputForm**
- **acOutputFunction**
- **acOutputModule**
- **acOutputQuery**
- **acOutputReport**
- **acOutputServerView**
- **acOutput.StoredProcedure**
- **acOutputTable**

**ObjectName** Optional **Variant**. A string expression that's the valid name of an object of the type selected by the **ObjectType** argument. If you want to output the active object, specify the object's type for the **ObjectType** argument and leave this argument blank. If you run Visual Basic code containing the **OutputTo** method in a library database, Microsoft Access looks for the object with this name, first in the library database, then in the current database.

**OutputFormat** Optional **Variant**. The output format, expressed as an **AcFormat** constant. If you omit this argument, Microsoft Access prompts you for the output format.

AcFormat can be one of these AcFormat constants.

- **acFormatASP**
acFormatDAP  
acFormatHTML  
acFormatIIS  
acFormatRTF  
acFormatSNP  
acFormatTXT  
acFormatXLS

**OutputFile**   Optional **Variant**. A string expression that's the full name, including the path, of the file you want to output the object to. If you leave this argument blank, Microsoft Access prompts you for an output file name.

**AutoStart**   Optional **Variant**. Use **True** (–1) to start the appropriate Microsoft Windows–based application immediately, with the file specified by the **OutputFile** argument loaded. Use **False** (0) if you don't want to start the application. This argument is ignored for Microsoft Internet Information Server (.htx, .idc) files and Microsoft ActiveX Server (* .asp) files. If you leave this argument blank, the default (%False%) is assumed.

**TemplateFile**   Optional **Variant**. A string expression that's the full name, including the path, of the file you want to use as a template for an **HTML**, **HTX**, or **ASP** file.

**Encoding**   Optional **Variant**.
Remarks

For more information on how the action and its arguments work, see the action topic.

Modules can be output only in MS-DOS Text format, so if you specify acOutputModule for the ObjectType argument, you must specify acFormatTXT for the OutputFormat argument. Microsoft Internet Information Server and Microsoft ActiveX Server formats are available only for tables, queries, and forms, so if you specify acFormatIIS or acFormatASP for the OutputFormat argument, you must specify acOutputTable, acOutputQuery, or acOutputForm for the ObjectType argument.

You can leave an optional argument blank in the middle of the syntax, but you must include the argument's comma. If you leave a trailing argument blank, don't use a comma following the last argument you specify.
Example

The following example outputs the Employees table in rich-text format (.rtf) to the Employee.rtf file and immediately opens the file in Microsoft Word for Windows.

DoCmd.OutputTo acOutputTable, "Employees", _
acFormatRTF, "Employee.rtf", True
Print Method

The **Print** method prints text on a **Report** object using the current color and font.

```plaintext
expression.Print(Expr)
```

*expression* Required. An expression that returns one of the objects in the Applies To list.

*Expr* Required **String**. The string expressions to print. If this argument is omitted, the **Print** method prints a blank line. Multiple expressions can be separated with a space, a semicolon (;), or a comma. A space has the same effect as a semicolon.
Remarks

You can use this method only in an event procedure or macro specified by a section's **OnPrint** event property setting.

The expressions specified by the *Expr* argument are printed on the object starting at the position indicated by the **CurrentX** and **CurrentY** property settings.

When the *Expr* argument is printed, a carriage return is usually appended so that the next **Print** method begins printing on the next line. When a carriage return occurs, the **CurrentY** property setting is increased by the height of the *Expr* argument (the same as the value returned by the **TextHeight** method) and the **CurrentX** property is set to 0.

When a semicolon follows the *Expr* argument, no carriage return is appended, and the next **Print** method prints on the same line that the current **Print** method printed on. The **CurrentX** and **CurrentY** properties are set to the point immediately after the last character printed. If the *Expr* argument itself contains carriage returns, each such embedded carriage return sets the **CurrentX** and **CurrentY** properties as described for the **Print** method without a semicolon.

When a comma follows the *Expr* argument, the **CurrentX** and **CurrentY** properties are set to the next print zone on the same line.

When the *Expr* argument is printed on a **Report** object, lines that can't fit in the specified position don't scroll. The text is clipped to fit the object.

Because the **Print** method usually prints with proportionally spaced characters, it's important to remember that there's no correlation between the number of characters printed and the number of fixed-width columns those characters occupy. For example, a wide letter (such as W) occupies more than one fixed-width column, whereas a narrow letter (such as I) occupies less. You should make sure that your tabular columns are positioned far enough apart to accommodate the text you wish to print. Alternately, you can print with a fixed-pitch font (such as Courier) to ensure that each character uses only one column.
**Example**

The following example uses the **Print** method to display text on a report named Report1. It uses the **TextWidth** and **TextHeight** methods to center the text vertically and horizontally.

```vbscript
Private Sub Detail_Format(Cancel As Integer, _
    FormatCount As Integer)
    Dim rpt As Report
    Dim strMessage As String
    Dim intHorSize As Integer, intVerSize As Integer

    Set rpt = Me
    strMessage = "DisplayMessage"
    With rpt
        'Set scale to pixels, and set FontName and
        'FontSize properties.
        .ScaleMode = 3
        .FontName = "Courier"
        .FontSize = 24
    End With
    ' Horizontal width.
    intHorSize = Rpt.TextWidth(strMessage)
    ' Vertical height.
    intVerSize = Rpt.TextHeight(strMessage)
    ' Calculate location of text to be displayed.
    Rpt.CurrentX = (Rpt.ScaleWidth/2) - (intHorSize/2)
    Rpt.CurrentY = (Rpt.ScaleHeight/2) - (intVerSize/2)
    ' Print text on Report object.
    Rpt.Print strMessage
End Sub
```
PrintOut Method

The PrintOut method carries out the PrintOut action in Visual Basic.

expression.PrintOut(PrintRange, PageFrom, PageTo, PrintQuality, Copies, CollateCopies)

description Required. An expression that returns one of the objects in the Applies To list.

PrintRange Optional AcPrintRange.

AcPrintRange can be one of these AcPrintRange constants.
acPages
acPrintAll default
acSelection

If you leave this argument blank, the default constant (acPrintAll) is assumed.

PageFrom Optional Variant. A numeric expression that's a valid page number in the active form or datasheet. This argument is required if you specify acPages for the printrange argument.

PageTo Optional Variant. A numeric expression that's a valid page number in the active form or datasheet. This argument is required if you specify acPages for the printrange argument.

PrintQuality Optional AcPrintQuality.

AcPrintQuality can be one of these AcPrintQuality constants.
acDraft
acHigh default
acLow
acMedium
If you leave this argument blank, the default constant (acHigh) is assumed.

**Copies**  Optional **Variant**. A numeric expression. If you leave this argument blank, the default (1) is assumed.

**CollateCopies**  Optional **Variant**. Use True (−1) to collate copies and False (0) to print without collating. If you leave this argument blank, the default (True) is assumed.
Remarks

For more information on how the action and its arguments work, see the action topic.

You can leave an optional argument blank in the middle of the syntax, but you must include the argument's comma. If you leave one or more trailing arguments blank, don't use a comma following the last argument you specify.
Example

The following example prints two collated copies of the first four pages of the active form or datasheet:

```
DoCmd.PrintOut acPages, 1, 4, , 2
```
PSet Method

The PSet method sets a point on a Report object to a specified color when the Print event occurs.

(expression).PSet(flags, X, Y, color)

expression Required. An expression that returns one of the objects in the Applies To list.

flags Required Integer. A keyword that indicates the coordinates are relative to the current graphics position given by the settings for the CurrentX and CurrentY properties of the object argument.

X Required Single. Single value indicating the horizontal coordinate of the point to set.

Y Required Single. Single value indicating the vertical coordinate of the point to set.

color Required Long. A Long value indicating the RGB (red-green-blue) color to set the point to. If this argument is omitted, the value of the ForeColor property is used. You can also use the RGB function or QBColor function to specify the color.
Remarks

The size of the point depends on the **DrawWidth** property setting. When the **DrawWidth** property is set to 1, the **PSet** method sets a single pixel to the specified color. When the **DrawWidth** property is greater than 1, the point is centered on the specified coordinates.

The way the point is drawn depends on the settings of the **DrawMode** and **DrawStyle** properties.

When you apply the **PSet** method, the **CurrentX** and **CurrentY** properties are set to the point specified by the \textit{x} and \textit{y} arguments.

To clear a single pixel with the **PSet** method, specify the coordinates of the pixel and use `&HFFFFFF` (white) as the \textit{color} argument.

**Tip**

It's faster to draw a line by using the **Line** method rather than the **PSet** method.
**Example**

The following example uses the `PSet` method to draw a line through the horizontal axis of a report.

To try this example in Microsoft Access, create a new report. Set the `OnPrint` property of the Detail section to [Event Procedure]. Enter the following code in the report's module, then switch to Print Preview.

```vba
Sub Detail_Print(Cancel As Integer, PrintCount As Integer)
    Dim sngMidPt As Single, intI As Integer
    ' Set scale to pixels.
    Me.ScaleMode = 3
    ' Calculate midpoint.
    sngMidPt = Me.ScaleHeight / 2
    ' Loop to draw line down horizontal axis pixel by pixel.
    For intI = 1 To Me.ScaleWidth
        Me.PSet(intI, sngMidPt)
    Next intI
End Sub
```
Quit Method

The **Quit** method quits Microsoft Access. You can select one of several options for saving a database object before quitting.

```
expression.Quit(Option)
```

**expression** Required. An expression that returns an **Application** object.

**Option** Optional **AcQuitOption**. The quit option.

*AcQuitOption* can be one of these *AcQuitOption* constants.

- **acQuitPrompt** Displays a dialog box that asks whether you want to save any database objects that have been changed but not saved. (Formerly **acPrompt**).
- **acQuitSaveAll** default Saves all objects without displaying a dialog box. (Formerly **acSaveYes**).
- **acQuitSaveNone** Quits Microsoft Access without saving any objects. (Formerly **acExit**).
Remarks

The Quit method has the same effect as clicking Exit on the File menu. You can create a custom menu command or a command button on a form with a procedure that includes the Quit method. For example, you can place a Quit button on a form and include a procedure in the button's Click event that uses the Quit method with the Option argument set to acQuitSaveAll.

Quit method as it applies to the DoCmd object.

The Quit method of the DoCmd object carries out the Quit action in Visual Basic.

expression.Quit(Options)

expression  Required. An expression that returns a DoCmd object.

Options  Optional AcQuitOption. The quit option.

AcQuitOption can be one of these AcQuitOption constants.

acQuitPrompt Displays a dialog box that asks whether you want to save any database objects that have been changed but not saved.

acQuitSaveAll default Saves all objects without displaying a dialog box.

acQuitSaveNone Quits Microsoft Access without saving any objects.
Remarks

The Quit method of the DoCmd object was added to provide backward compatibility for running the Quit action in Visual Basic code in Microsoft Access 95. It's recommended that you use the existing Quit method of the Application object instead.
Example

As it applies to the **Application** object.

The following example shows the **Click** event procedure for a command button named AppExit. After the AppExit button is clicked, a dialog box prompts the user to save changes and the procedure quits Microsoft Access.

```vba
Private Sub AppExit_Click()
    Application.Quit acQuitPrompt
End Sub
```

As it applies to the **DoCmd** object.

The following example displays a dialog box or dialog boxes that ask if users want to save any changed objects before they quit Microsoft Access.

```vba
DoCmd.Quit acQuitPrompt
```
Recalc Method

The **Recalc** method immediately updates all calculated controls on a form.

*expression*.**Recalc**

*expression* Required. An expression that returns one of the objects in the Applies To list.
Remarks

Using this method is equivalent to pressing the F9 key when a form has the focus. You can use this method to recalculate the values of controls that depend on other fields for which the contents may have changed.
Example

The following example uses the `Recalc` method to update controls on an Orders form. This form includes the Freight text box, which displays the freight cost, and a calculated control that displays the total cost of an order including freight. If the statement containing the `Recalc` method is placed in the `AfterUpdate` event procedure for the Freight text box, the total cost of an order is recalculated every time a new freight amount is entered.

```vba
Sub Freight_AfterUpdate()
    Me.Recalc
End Sub
```
Refresh Method

The **Refresh** method immediately updates the records in the underlying record source for a specified form or datasheet to reflect changes made to the data by you and other users in a multiuser environment.

`expression.Refresh`

`expression`  Required. An expression that returns one of the objects in the Applies To list.
Remarks

Using the **Refresh** method is equivalent to clicking **Refresh** on the **Records** menu.

Microsoft Access refreshes records automatically, based on the **Refresh Interval** setting on the **Advanced** tab of the **Options** dialog box, available by clicking **Options** on the **Tools** menu. ODBC data sources are refreshed based on the **ODBC Refresh Interval** setting on the **Advanced** tab of the **Options** dialog box. You can use the **Refresh** method to view changes that have been made to the current set of records in a form or datasheet since the record source underlying the form or datasheet was last refreshed.

The **Refresh** method shows only changes made to records in the current set. Since the **Refresh** method doesn't actually requery the database, the current set won't include records that have been added or exclude records that have been deleted since the database was last requeried. Nor will it exclude records that no longer satisfy the criteria of the query or filter. To requery the database, use the **Requery** method. When the record source for a form is requeried, the current set of records will accurately reflect all data in the record source.

Notes

- It's often faster to refresh a form or datasheet than to requery it. This is especially true if the initial query was slow to run.
- Don't confuse the **Refresh** method with the **Repaint** method, which repaints the screen with any pending visual changes.
Example

The following example uses the **Refresh** method to update the records in the underlying record source for the form Customers whenever the form receives the focus:

```vba
Private Sub Form_Activate()
    Me.Refresh
End Sub
```
RefreshDatabaseWindow Method

The RefreshDatabaseWindow method updates the Database window after a database object has been created, deleted, or renamed.

expression.RefreshDatabaseWindow

expression  Required. An expression that returns one of the objects in the Applies To list.
Remarks

You can use the RefreshDatabaseWindow method to immediately reflect changes to objects in Microsoft Access in the Database window. For example, if you add a new form from Visual Basic and save it, you can use the RefreshDatabaseWindow method to display the name of the new form on the Forms tab of the Database window immediately after it has been saved.
Example

The following example creates a new form, saves it, and refreshes the Database window:

Sub CreateFormAndRefresh()
    Dim frm As Form

    Set frm = CreateForm
    DoCmd.Save , "NewForm"
    RefreshDatabaseWindow
End Sub
RefreshTitleBar Method

The **RefreshTitleBar** method refreshes the Microsoft Access **title bar** after the **AppTitle** or **AppIcon** property has been set in Visual Basic.

`expression.RefreshTitleBar`

*expression* Required. An expression that returns one of the objects in the Applies To list.
Remarks

For example, you can change the caption in the Microsoft Access title bar to "Contacts Database" by setting the AppTitle property.

The AppTitle and AppIcon properties enable you to customize your application by changing the title and icon that appear in the Microsoft Access title bar. The title bar is not automatically updated after you have set these properties. In order for the change to the title bar to appear, you must use the RefreshTitleBar method.

Note  In a Microsoft Access database (.mdb), you can reset the AppTitle and AppIcon properties to their default value by deleting them from the Properties collection representing the current database. After you delete these properties, you must use the RefreshTitleBar method to restore the Microsoft Access defaults to the title bar.

It the path to the icon specified by the AppIcon property is invalid, then no changes will be reflected in the title bar when you call this method.
Example

The following example sets the **AppTitle** property of the current database and applies the **RefreshTitleBar** method to update the title bar.

```vbscript
Sub ChangeTitle()
    Dim obj As Object
    Const conPropNotFoundError = 3270
    
    On Error GoTo ErrorHandler
    ' Return Database object variable pointing to the current database.
    Set dbs = CurrentDb
    ' Change title bar.
    dbs.Properties!AppTitle = "Contacts Database"
    ' Update title bar on screen.
    Application.RefreshTitleBar
    Exit Sub

ErrorHandler:
    If Err.Number = conPropNotFoundError Then
        Set obj = dbs.CreateProperty("AppTitle", dbText, "Contacts Database"
    dbs.Properties.Append obj
    Else
        MsgBox "Error: " & Err.Number & vbCrLf & Err.Description
    End If
    Resume Next
End Sub
```
Show All
Remove Method

Some of the content in this topic may not be applicable to some languages.

Remove method as it applies to the AccessObjectProperties collection object.

You can use the **Remove** method to remove an **AccessObjectProperty** object from the **AccessObjectProperties** collection of an **AccessObject** object.

```
expression.Remove(Item)
```

**expression**  Required. An expression that returns an **AccessObjectProperties** collection object.

**Item**  Required **Variant**. An expression that specifies the position of a member of the collection referred to by the object argument. If a numeric expression, the index argument must be a number from 0 to the value of the collection's **Count** property minus 1. If a string expression, the index argument must be the name of a member of the collection.

Remove method as it applies to the **Pages** collection object.

The **Remove** method removes a **Page** object from the **Pages** collection of a **tab control**.

```
expression.Remove(Item)
```

**expression**  Required. An expression that returns a **Pages** collection object.

**Item**  Optional **Variant**. An integer that specifies the index of the **Page** object to be removed. The index of the **Page** object corresponds to the value of the **PageIndex** property for that **Page** object. If you omit this argument, the last **Page** object in the collection is removed.
Remarks

The **Pages** collection is indexed beginning with zero. The leftmost page in the tab control has an index of 0, the page immediately to the right of the leftmost page has an index of 1, and so on.

You can remove a **Page** object from the **Pages** collection of a tab control only when the **form** is in **Design view**.

Remove method as it applies to the **References** collection object.

The **Remove** method removes a **Reference** object from the **References** collection.

```
expression.Remove(Reference)
```

**expression**  Required. An expression that returns a **References** collection object.

**Reference**  Required **Reference** object. The **Reference** object that represents the reference you wish to remove.
Remarks

To determine the name of the Reference object you wish to remove, check the Project/Library box in the Object Browser. The names of all references that are currently set appear there. These names correspond to the value of the Name property of a Reference object.
Example

As it applies to the Pages object.

The following example removes pages from a tab control:

Function RemovePage() As Boolean
    Dim frm As Form
    Dim tbc As TabControl, pge As Page

    On Error GoTo Error_RemovePage
    Set frm = Forms!Form1
    Set tbc = frm!TabCtl0
    tbc.Pages.Remove
    RemovePage = True

Exit_RemovePage:
    Exit Function

Error_RemovePage:
    MsgBox Err & ": " & Err.Description
    RemovePage = False
    Resume Exit_RemovePage
End Function

As it applies to the References object.

The first of the following two functions adds a reference to the calendar control for the References collection. The second function removes the reference to the calendar control.

Function AddReference() As Boolean
    Dim ref As Reference, strFile As String

    On Error GoTo Error_AddReference
    strFile = "C:\Windows\System\Mscal.ocx"
    ' Create reference to calendar control.
    Set ref = References.AddFromFile(strFile)
    AddReference = True

Exit_AddReference:
    Exit Function
Function AddReference() As Boolean
    Dim ref As Reference
    On Error GoTo Error_AddReference
    Set ref = References!MSCAL
    ' Remove calendar control reference.
    References.Remove ref
    RemoveReference = True

Exit_AddReference:
    Exit Function

Error_AddReference:
    MsgBox Err & ": " & Err.Description
    AddReference = False
    Resume Exit_AddReference
End Function

Function RemoveReference() As Boolean
    Dim ref As Reference
    On Error GoTo Error_RemoveReference
    Set ref = References!MSCAL
    ' Remove calendar control reference.
    References.Remove ref
    RemoveReference = True

Exit_RemoveReference:
    Exit Function

Error_RemoveReference:
    MsgBox Err & ": " & Err.Description
    RemoveReference = False
    Resume Exit_RemoveReference
End Function
**RemoveItem Method**

Removes an item from the list of values displayed by the specified list box control or combo box control.

\[expression.RemoveItem(Index)\]

*expression*  Required. An expression that returns one of the objects in the Applies To list.

*Index*  Required **Variant**. The item to be removed from the list, expressed as either an item number or the list item text.
Remarks

This method is only valid for list box or combo box controls on forms. Also, the RowSourceType property of the control must be set to "Value List".

List item numbers start from zero. If the value of the Index argument doesn't correspond to an existing item number or the text of an existing item, an error occurs.

Use the AddItem method to add items to the list of values.
Example

This example removes the specified item from the list in a list box control. For the function to work, you must pass it a **ListBox** object representing a list box control on a form and a **Variant** value representing the item to be removed.

```vba
Function RemoveListItem(ctrlListBox As ListBox, _
   ByVal varItem As Variant) As Boolean

    ' Trap for errors.
    On Error GoTo ERROR_HANDLER

    ' Remove the list box item and set the return value
    ' to True, indicating success.
    ctrlListBox.RemoveItem Index:=varItem
    RemoveListItem = True

    ' Reset the error trap and exit the function.
    On Error GoTo 0
    Exit Function

    ' Return False if an error occurs.
    ERROR_HANDLER:
        RemoveListItem = False

End Function
```
Rename Method

The **Rename** method carries out the **Rename** action in Visual Basic.

```vbnet
expression.Rename(NewName, ObjectType, OldName)
```

- **expression**  Required. An expression that returns a **DoCmd** object.

- **NewName**  Required **Variant**. A **string expression** that's the new name for the object you want to rename. The name must follow the object-naming rules for Microsoft Access objects.

- **ObjectType**  Optional **AcObjectType**. The type of object to rename.

  AcObjectType can be one of these AcObjectType constants.
  - **acDataAccessPage**
  - **acDefault default**
  - **acDiagram**
  - **acForm**
  - **acFunction**
  - **acMacro**
  - **acModule**
  - **acQuery**
  - **acReport**
  - **acServerView**
  - **acStoredProcedure**
  - **acTable**

- **OldName**  Optional **Variant**. A string expression that's the valid name of an object of the type specified by the **ObjectType** argument. If you execute Visual Basic code containing the **Rename** method in a **library database**, Microsoft Access looks for the object with this name, first in the library database, then in the current database.
Remarks

For more information on how the action and its arguments work, see the action topic.

If you leave the **ObjectType** and **OldName** arguments blank (the default constant, **acDefault**, is assumed for **ObjectType**), Microsoft Access renames the object selected in the Database window. To select an object in the Database window, you can use the SelectObject action or SelectObject method with the In Database Window argument set to Yes (**True**).

If you leave the **ObjectType** and **OldName** arguments blank, don't use a comma following the **NewName** argument.
Example

The following example renames the Employees table.

DoCmd.Rename "Old Employees Table", acTable, "Employees"
Repaint Method

The **Repaint** method completes any pending screen updates for a specified form. When performed on a form, the **Repaint** method also completes any pending recalculations of the form's controls.

*expression*.**Repaint**

*expression*  Required. An expression that returns one of the objects in the Applies To list.
Remarks

Microsoft Access sometimes waits to complete pending screen updates until it finishes other tasks. With the **Repaint** method, you can force immediate repainting of the controls on the specified form. You can use the **Repaint** method:

- When you change values in a number of fields. Unless you force a repaint, Microsoft Access might not display the changes immediately, especially if other fields, such as those in an expression in a **calculated control**, depend on values in the changed fields.

- When you want to make sure that a form displays data in all of its fields. For example, fields containing **OLE objects** often don't display their data immediately after you open a form.

This method doesn't cause a **requery** of the database, nor does it show new or changed records in the form's underlying record source. You can use the **Requery** method to requery the source of data for the form or one of its controls.

Notes

- Don't confuse the **Repaint** method with the **Refresh** method, or with the **Refresh** command on the **Records** menu. The **Refresh** method and **Refresh** command show changes you or other users have made to the underlying record source for any of the currently displayed records in forms and datasheets. The **Repaint** method simply updates the screen when repainting has been delayed while Microsoft Access completes other tasks.

- The **Repaint** method differs from the **Echo** method in that the **Repaint** method forces a single immediate repaint, while the **Echo** method turns repainting on or off.
Example

The following example uses the Repaint method to repaint a form when the form receives the focus:

Private Sub Form_Activate()
    Me.Repaint
End Sub
RepaintObject Method

The **RepaintObject** method carries out the **RepaintObject** action in Visual Basic.

**expression**.RepaintObject**(ObjectType, ObjectName)**

**expression**  Required. An expression that returns one of the objects in the Applies To list.

**ObjectType**  Optional **AcObjectType**.

AcObjectType can be one of these AcObjectType constants.

- **acDataAccessPage**
- **acDefault**  *default*
- **acDiagram**
- **acForm**
- **acFunction**
- **acMacro**
- **acModule**
- **acQuery**
- **acReport**
- **acServerView**
- **acStoredProcedure**
- **acTable**

**ObjectName**  Optional **Variant**. A **string expression** that's the valid name of an object of the type selected by the **objecttype** argument.
Remarks

For more information on how the action and its arguments work, see the action topic.

Using the **RepaintObject** method with no arguments (the default constant, **acDefault**, is assumed for the **objecttype** argument) repa**ints** the active window.

The **RepaintObject** method of the **DoCmd** object was added to provide backwards compatibility for running the **RepaintObject** method in Visual Basic code in Microsoft Access 95. If you want to repaint a **form**, it's recommended that you use the existing **Repaint** method of the **Form** object instead.
Example

The following example repaints the table Customers:

DoCmd.RepaintObject acTable, "Customers"
ReplaceLine Method

The **ReplaceLine** method replaces a specified line in a [standard module](#) or a [class module](#).

**expression.ReplaceLine(Line, String)**

*expression* Required. An expression that returns one of the objects in the Applies To list.

**Line** Required **Long**. A **Long** value that specifies the number of the line to be replaced.

**String** Required **String**. The text that is to replace the existing line.
Remarks

Lines in a module are numbered beginning with one. To determine the number of lines in a module, use the CountOfLines property.
Requery Method

Requery method as it applies to the DoCmd object.

The Requery method of the DoCmd object carries out the Requery action in Visual Basic.

\[ \text{expression}.\text{Requery}(\text{ControlName}) \]

\textit{expression}  
Required. An expression that returns one of the above objects.

\textit{ControlName}  
Optional Variant. A \textit{string expression} that's the name of a \textit{control} on the active object.

Requery method as it applies to all other objects in the Applies To list.

The Requery method updates the data underlying a specified form or a control that's on the active form by requerying the source of data for the form or control.

\[ \text{expression}.\text{Requery} \]

\textit{expression}  
Required. An expression that returns one of the objects in the Applies To list.
Remarks

You can use this method to ensure that a form or control displays the most recent data.

The **Requery** method does one of the following:

- Reruns the query on which the form or control is based.
- Displays any new or changed records or removes deleted records from the table on which the form or control is based.
- Updates records displayed based on any changes to the **Filter** property of the form.

Controls based on a query or table include:

- **List boxes** and **combo boxes**.
- **Subform** controls.
- **OLE objects**, such as charts.
- Controls for which the **ControlSource** property setting includes **domain aggregate functions** or SQL aggregate function.

If you specify any other type of control for the object specified by *expression*, the record source for the form is requieried.

If the object specified by *expression* isn't bound to a field in a table or query, the **Requery** method forces a recalculation of the control.

If you omit the object specified by *expression*, the **Requery** method requeries the underlying data source for the form or control that has the **focus**. If the control that has the focus has a record source or row source, it will be requieried; otherwise, the control's data will simply be refreshed.

If a subform control has the focus, this method only requeries the record source for the subform, not the parent form.

Notes

- The **Requery** method updates the data underlying a form or control to
reflect records that are new to or deleted from the record source since it was last queried. The Refresh method shows only changes that have been made to the current set of records; it doesn't reflect new or deleted records in the record source. The Repaint method simply repaints the specified form and its controls.

- The Requery method doesn't pass control to the operating system to allow Windows to continue processing messages. Use the DoEvents function if you need to relinquish temporary control to the operating system.
- The Requery method is faster than the Requery action. When you use the Requery action, Microsoft Access closes the query and reloads it from the database. When you use the Requery method, Microsoft Access reruns the query without closing and reloading it.
Example

As it applies to the DoCmd object.

The following example uses the Requery method to update the EmployeeList control:

DoCmd.Requery "EmployeeList"

As it applies to all other objects in the Applies To list.

The following example uses the Requery method to requery the data from the EmployeeList list box on an Employees form:

Public Sub RequeryList()
    Dim ctlCombo As Control
    ' Return Control object pointing to a combo box.
    Set ctlCombo = Forms!Employees!ReportsTo
    ' Requery source of data for list box.
    ctlCombo.Requery
End Sub
Restore Method

The Restore method carries out the Restore action in Visual Basic.

expression.Restore

expression  Required. An expression that returns a DoCmd object.
Remarks

This method has no arguments and be called directly using the syntax `DoCmd.Restore`.

**Note** This method cannot be applied to module windows in the Visual Basic Editor (VBE). For information about how to affect module windows see the `WindowState` property topic.
Run Method

You can use the **Run** method to carry out a specified Microsoft Access or user-defined **Function** or **Sub** procedure. **Variant**.


**expression** Required. An expression that returns one of the objects in the Applies To list.

**Procedure** Required **String**. The name of the **Function** or **Sub** procedure to be run. If you are calling a procedure in another database use the project name and the procedure name separated by a dot in the form: "projectname.procedurename". If you execute Visual Basic code containing the **Run** method in a library database, Microsoft Access looks for the procedure first in the library database, then in the current database.

**Arg1** Optional **Variant**.

**Arg2** Optional **Variant**.

**Arg3** Optional **Variant**.

**Arg4** Optional **Variant**.

**Arg5** Optional **Variant**.

**Arg6** Optional **Variant**.

**Arg7** Optional **Variant**.

**Arg8** Optional **Variant**.

**Arg9** Optional **Variant**.
Arg10  Optional Variant.
Arg11  Optional Variant.
Arg12  Optional Variant.
Arg13  Optional Variant.
Arg14  Optional Variant.
Arg15  Optional Variant.
Arg16  Optional Variant.
Arg17  Optional Variant.
Arg18  Optional Variant.
Arg19  Optional Variant.
Arg20  Optional Variant.
Arg21  Optional Variant.
Arg22  Optional Variant.
Arg23  Optional Variant.
Arg24  Optional Variant.
Arg25  Optional Variant.
Arg26  Optional Variant.
Arg27  Optional Variant.
Arg28  Optional Variant.
Arg29  Optional Variant.
Arg30  Optional Variant.
**Remarks**

This method is useful when you are controlling Microsoft Access from another application through Automation, formerly called OLE Automation. For example, you can use the **Run** method from an ActiveX component to carry out a **Sub** procedure that is defined within a Microsoft Access database.

You can set a reference to the Microsoft Access **type library** from any other ActiveX component and use the objects, methods, and properties defined in that library in your code. However, you can't set a reference to an individual Microsoft Access database from any application other than Microsoft Access.

For example, suppose you have defined a procedure named NewForm in a database with its **ProjectName** property set to "WizCode." The NewForm procedure takes a string argument. You can call NewForm in the following manner from Visual Basic:

```vbscript
Dim appAccess As New Access.Application
appAccess.OpenCurrentDatabase ("C:\My Documents\WizCode.mdb")
```

If another procedure with the same name may reside in a different database, qualify the **procedure** argument, as shown in the preceding example, with the name of the database in which the desired procedure resides.

You can also use the **Run** method to call a procedure in a referenced Microsoft Access database from another database.
Example

The following example runs a user-defined Sub procedure in a module in a Microsoft Access database from another application that acts as an Active X component.

To try this example, create a new database called WizCode.mdb and set its ProjectName property to WizCode. Open a new module in that database and enter the following code. Save the module, and close the database.

Note You set the ProjectName be selecting Tools, WizCode Properties... from the VBE main menu.

Public Sub Greeting(ByVal strName As String)
    MsgBox ("Hello, " & strName & "!"), vbInformation, "Greetings"
End Sub

Once you have completed this step, run the following code from Microsoft Excel or Microsoft Visual Basic. Make sure that you have added a reference to the Microsoft Access type library by clicking References on the Tools menu and selecting Microsoft Access 10.0 Object Library in the References dialog box.

Private Sub RunAccessSub()

    Dim appAccess As Access.Application

    ' Create instance of Access Application object.
    Set appAccess = CreateObject("Access.Application")

    ' Open WizCode database in Microsoft Access window.

    ' Run Sub procedure.
    appAccess.Run "Greeting", "Joe"
    Set appAccess = Nothing

End Sub
RunCommand Method

The **RunCommand** method runs a built-in menu or toolbar command.

```
expression.RunCommand(Command)
```

**expression**  Required. An expression that returns one of the objects in the Applies To list.

**Command**  Required **AcCommand**. An intrinsic constant that specifies which built-in menu or toolbar command is to be run.

AcCommand can be one of these AcCommand constants.

- `acCmdAboutMicrosoftAccess`
- `acCmdAddInManager`
- `acCmdAddToNewGroup`
- `acCmdAddWatch`
- `acCmdAdvancedFilterSort`
- `acCmdAlignBottom`
- `acCmdAlignCenter`
- `acCmdAlignLeft`
- `acCmdAlignmentAndSizing`
- `acCmdAlignMiddle`
- `acCmdAlignRight`
- `acCmdAlignToGrid`
- `acCmdAlignTop`
- `acCmdAlignToShortest`
- `acCmdAlignToTallest`
- `acCmdAnalyzePerformance`
- `acCmdAnalyzeTable`
- `acCmdAnswerWizard`
- `acCmdApplyDefault`
- `acCmdApplyFilterSort`
acCmdAppMaximize
acCmdAppMinimize
acCmdAppMove
acCmdAppRestore
acCmdAppSize
acCmdArrangeIconsAuto
acCmdArrangeIconsByCreated
acCmdArrangeIconsByModified
acCmdArrangeIconsByName
acCmdArrangeIconsByType
acCmdAutoCorrect
acCmdAutoDial
acCmdAutoFormat
acCmdBackgroundPicture
acCmdBackgroundSound
acCmdBackup
acCmdBookmarksClearAll
acCmdBookmarksNext
acCmdBookmarksPrevious
acCmdBookmarksToggle
acCmdBringToFront
acCmdCallStack
acCmdChangeToCheckBox
acCmdChangeToComboBox
acCmdChangeToCommandButton
acCmdChangeToImage
acCmdChangeToLabel
acCmdChangeToListBox
acCmdChangeToOptionButton
acCmdChangeToTextBox
acCmdChangeToToggleButton
acCmdChartSortAscByTotal
acCmdChartSortDescByTotal
acCmdClearAll
acCmdClearAllBreakpoints
acCmdClearGrid
acCmdClearHyperlink
acCmdClearItemDefaults
acCmdClose
acCmdCloseWindow
acCmdColumnWidth
acCmdCompactDatabase
acCmdCompileAllModules
acCmdCompileAndSaveAllModules
acCmdCompileLoadedModules
acCmdCompleteWord
acCmdConditionalFormatting
acCmdConnection
acCmdControlWizardsToggle
acCmdConvertDatabase
acCmdConvertMacrosToVisualBasic
acCmdCopy
acCmdCopyDatabaseFile
acCmdCopyHyperlink
acCmdCreateMenuFromMacro
acCmdCreateRelationship
acCmdCreateReplica
acCmdCreateShortcut
acCmdCreateShortcutMenuFromMacro
acCmdCreateToolbarFromMacro
acCmdCut
acCmdDataAccessPageAddToPage
acCmdDataAccessPageBrowse
acCmdDataAccessPageDesignView
acCmdDataAccessPageFieldListRefresh
acCmdDatabaseProperties
acCmdDocSize
acCmdDocumenter
acCmdDropSQLDatabase
acCmdDuplicate
acCmdEditHyperlink
acCmdEditingAllowed
acCmdEditRelationship
acCmdEditTriggers
acCmdEditWatch
acCmdEncryptDecryptDatabase
acCmdEnd
acCmdExit
acCmdExport
acCmdFavoritesAddTo
acCmdFavoritesOpen
acCmdFieldList
acCmdFilterByForm
acCmdFilterBySelection
acCmdFilterExcludingSelection
acCmdFind
acCmdFindNext
acCmdFindNextWordUnderCursor
acCmdFindPrevious
acCmdFindPrevWordUnderCursor
acCmdFitToWindow
acCmdFont
acCmdFormatCells
acCmdFormHdrFtr
acCmdFormView
acCmdFreezeColumn
acCmdGoBack
acCmdGoContinue
acCmdGoForward
acCmdGroupByTable
acCmdGroupControls
acCmdHideColumns
acCmdHidePane
acCmdHideTable
acCmdHorizontalSpacingDecrease
acCmdHorizontalSpacingIncrease
acCmdHorizontalSpacingMakeEqual
acCmdHyperlinkDisplayText
acCmdImport
acCmdIndent
acCmdIndexes
acCmdInsertActiveXControl
acCmdInsertChart
acCmdInsertFile
acCmdInsertFileIntoModule
acCmdInsertHyperlink
acCmdInsertLookupColumn
acCmdInsertLookupField
acCmdInsertMovieFromFile
acCmdInsertObject
acCmdInsertPage
acCmdInsertPicture
acCmdInsertPivotTable
acCmdInsertProcedure
acCmdInsertQueryColumn
acCmdInsertRows
acCmdInsertSpreadsheet
acCmdInsertSubdatasheet
acCmdInsertTableColumn
acCmdInsertUnboundSection
acCmdInvokeBuilder
acCmdJoinProperties
acCmdObjBrwHelp
acCmdObjBrwShowHiddenMembers
acCmdObjBrwViewDefinition
acCmdObjectBrowser
acCmdOfficeClipboard
acCmdOLEDDELinks
acCmdOLEObjectConvert
acCmdOLEObjectDefaultVerb
acCmdOpenDatabase
acCmdOpenHyperlink
acCmdOpenNewHyperlink
acCmdOpenSearchPage
acCmdOpenStartPage
acCmdOpenTable
acCmdOpenURL
acCmdOptions
acCmdOutdent
acCmdOutputToExcel
acCmdOutputToRTF
acCmdOutputToText
acCmdPageHdrFtr
acCmdPageNumber
acCmdPageProperties
acCmdPageSetup
acCmdParameterInfo
acCmdPartialReplicaWizard
acCmdPaste
acCmdPasteAppend
acCmdPasteAsHyperlink
acCmdPasteSpecial
acCmdPivotAutoAverage
acCmdPivotAutoCount
acCmdPivotAutoFilter
acCmdPivotAutoMax
acCmdPivotAutoMin
acCmdPivotAutoStdDev
acCmdPivotAutoStdDevP
acCmdPivotAutoSum
acCmdPivotAutoVar
acCmdPivotAutoVarP
acCmdPivotChartByRowByColumn
acCmdPivotChartDrillInto
acCmdPivotChartDrillOut
acCmdPivotChartMultiplePlots
acCmdPivotChartMultiplePlotsUnifiedScale
acCmdPivotChartShowLegend
acCmdPivotChartType
acCmdPivotChartUndo
acCmdPivotChartView
acCmdPivotCollapse
acCmdPivotDelete
acCmdPivotDropAreas
acCmdPivotExpand
acCmdPivotRefresh
acCmdPivotShowAll
acCmdPivotShowBottom1
acCmdPivotShowBottom10
acCmdPivotShowBottom10Percent
acCmdPivotShowBottom1Percent
acCmdPivotShowBottom2
acCmdPivotShowBottom25
acCmdPivotShowBottom25Percent
acCmdPivotShowBottom2Percent
acCmdPivotShowBottom5
acCmdPivotShowBottom5Percent
acCmdPivotShowBottomOther
acCmdPivotShowTop1
acCmdPivotShowTop10
acCmdPivotShowTop10Percent
acCmdPivotShowTop1Percent
acCmdPivotShowTop2
acCmdPivotShowTop25
acCmdPivotShowTop25Percent
acCmdPivotShowTop2Percent
acCmdPivotShowTop5
acCmdPivotShowTop5Percent
acCmdPivotShowTopOther
acCmdPivotTableClearCustomOrdering
acCmdPivotTableCreateCalcField
acCmdPivotTableCreateCalcTotal
acCmdPivotTableDemote
acCmdPivotTableExpandIndicators
acCmdPivotTableExportToExcel
acCmdPivotTableFilterBySelection
acCmdPivotTableGroupItems
acCmdPivotTableHideDetails
acCmdPivotTableMoveToColumnArea
acCmdPivotTableMoveToDetailArea
acCmdPivotTableMoveToFilterArea
acCmdPivotTableMoveToRowArea
acCmdPivotTablePercentColumnTotal
acCmdPivotTablePercentGrandTotal
acCmdPivotTablePercentParentColumnItem
acCmdPivotTablePercentParentRowItem
acCmdPivotTablePercentRowTotal
acCmdPivotTablePromote
acCmdPivotTableRemove
acCmdPivotTableShowAsNormal
acCmdPivotTableShowDetails
acCmdPivotTableSubtotal
acCmdPivotTableUngroupItems
acCmdPivotTableView
acCmdPreviewEightPages
acCmdPreviewFourPages
acCmdPreviewOnePage
acCmdPreviewTwelvePages
acCmdPreviewTwoPages
acCmdPrimaryKey
acCmdPrint
acCmdPrintPreview
acCmdPrintRelationships
acCmdProcedureDefinition
acCmdPromote
acCmdProperties
acCmdPublish
acCmdPublishDefaults
acCmdQueryAddToOutput
acCmdQueryGroupBy
acCmdQueryParameters
acCmdQueryTotals
acCmdQueryTypeAppend
acCmdQueryTypeCrosstab
acCmdQueryTypeDelete
acCmdQueryTypeMakeTable
acCmdQueryTypeSelect
acCmdQueryTypeSQLDataDefinition
acCmdQueryTypeSQLPassThrough
acCmdQueryTypeSQLUnion
acCmdQueryTypeUpdate
acCmdQuickInfo
acCmdQuickPrint
acCmdQuickWatch
acCmdRecordsGoToFirst
acCmdRecordsGoToLast
acCmdRecordsGoToNew
acCmdRecordsGoToNext
acCmdRecordsGoToPrevious
acCmdRecoverDesignMaster
acCmdRedo
acCmdReferences
acCmdRefresh
acCmdRefreshPage
acCmdRegisterActiveXControls
acCmdRelationships
acCmdRemove
acCmdRemoveFilterSort
acCmdRemoveTable
acCmdRename
acCmdRenameColumn
acCmdRenameGroup
acCmdRepairDatabase
acCmdReplace
acCmdReportHdrFtr
acCmdReset
acCmdResolveConflicts
acCmdRestore
acCmdRowHeight
acCmdRun
acCmdRunMacro
acCmdRunOpenMacro
acCmdSave
acCmdSaveAllModules
acCmdSaveAllRecords
acCmdSaveAs
acCmdSaveAsASP
acCmdSaveAsDataAccessPage
acCmdSaveAsHTML
acCmdSaveAsIDC
acCmdSaveAsQuery
acCmdSaveAsReport
acCmdSaveLayout
acCmdSaveModuleAsText
acCmdSaveRecord
acCmdSelectAll
acCmdSelectAllRecords
acCmdSelectDataAccessPage
acCmdSelectForm
acCmdSelectRecord
acCmdSelectReport
acCmdSend
acCmdSendToBack
acCmdServerFilterByForm
acCmdServerProperties
acCmdSetControlDefaults
acCmdSetDatabasePassword
acCmdSetNextStatement
acCmdShowAllRelationships
acCmdShowDirectRelationships
acCmdShowEnvelope
acCmdShowMembers
acCmdShowNextStatement
acCmdShowOnlyWebToolbar
acCmdShowTable
acCmdSingleStep
acCmdSizeToFit
acCmdSizeToFitForm
acCmdSizeToGrid
acCmdSizeToNarrowest
acCmdSizeToWidest
acCmdSnapToGrid
acCmdSortAscending
acCmdSortDescending
acCmdSortingAndGrouping
acCmdSpeech
acCmdSpelling
acCmdSQLView
acCmdStartupProperties
acCmdStepInto
acCmdStepOut
acCmdStepOver
acCmdStepToCursor
acCmdStopLoadingPage
acCmdSubdatasheetCollapseAll
acCmdSubdatasheetExpandAll
acCmdSubdatasheetRemove
acCmdSubformDatasheet
acCmdSubformDatasheetView
acCmdSubformFormView
acCmdSubformInNewWindow
acCmdSubformPivotChartView
acCmdSubformPivotTableView
acCmdSwitchboardManager
acCmdSynchronizeNow
acCmdTabControlPageOrder
acCmdTableAddTable
acCmdTableCustomView
acCmdTableNames
acCmdTabOrder
acCmdTestValidationRules
acCmdTileHorizontally
acCmdTileVertically
acCmdToggleBreakpoint
acCmdToggleFilter
acCmdToolbarControlProperties
acCmdToolbarsCustomize
acCmdTransferSQLDatabase
acCmdTransparentBackground
acCmdTransparentBorder
acCmdUndo
acCmdUndoAllRecords
acCmdUnfreezeAllColumns
acCmdUngroupControls
acCmdUnhideColumns
acCmdUpsizingWizard
acCmdUserAndGroupAccounts
acCmdUserAndGroupPermissions
acCmdUserLevelSecurityWizard
acCmdVerticalSpacingDecrease
acCmdVerticalSpacingIncrease
acCmdVerticalSpacingMakeEqual
acCmdViewCode
acCmdViewDataAccessPages
acCmdViewDetails
acCmdViewDiagrams
acCmdViewFieldList
acCmdViewForms
acCmdViewFunctions
acCmdViewGrid
acCmdViewLargeIcons
acCmdViewList
acCmdViewMacros
acCmdViewModules
acCmdViewQueries
acCmdViewReports
acCmdZoom500
acCmdZoom75
acCmdZoomBox
acCmdZoomSelection
Remarks

Each menu and toolbar command in Microsoft Access has an associated constant that you can use with the **RunCommand** method to run that command from Visual Basic.

You can't use the **RunCommand** method to run a command on a custom menu or toolbar. You can only use it with built-in menus and toolbars.

The **RunCommand** method replaces the **DoMenuItem** method of the **DoCmd** object.
Example

The following example uses the **RunCommand** method to open the **Options** dialog box (**Tools** menu).

```vba
Public Function OpenOptionsDialog() As Boolean

On Error GoTo Error_OpenOptionsDialog

    DoCmd.RunCommand acCmdOptions
    OpenOptionsDialog = True

Exit_OpenOptionsDialog:
    Exit Function

Error_OpenOptionsDialog:
    MsgBox Err & ": " & Err.Description
    OpenOptionsDialog = False
    Resume Exit_OpenOptionsDialog

End Function
```
Show All
RunMacro Method

The RunMacro method carries out the RunMacro action in Visual Basic.

expression.RunMacro(MacroName, RepeatCount, RepeatExpression)

expression  Required. An expression that returns one of the objects in the Applies To list.

MacroName  Required Variant. A string expression that's the valid name of a macro in the current database. If you run Visual Basic code containing the RunMacro method in a library database, Microsoft Access looks for the macro with this name in the library database and doesn't look for it in the current database.

RepeatCount  Optional Variant. A numeric expression that evaluates to an integer, which is the number of times the macro will run.

RepeatExpression  Optional Variant. A numeric expression that's evaluated each time the macro runs. When it evaluates to False (0), the macro stops running.
Remarks

For more information on how the action and its arguments work, see the action topic.

You can use `macrogroupname.macroname` syntax for the `macroname` argument to run a particular macro in a `macro group`.

If you specify the `repeatexpression` argument and leave the `repeatcount` argument blank, you must include the `repeatcount` argument's comma. If you leave a trailing argument blank, don't use a comma following the last argument you specify.
Example

The following example runs the macro Print Sales that will print the sales report twice:

```
DoCmd.RunMacro "Print Sales", 2
```
RunSQL Method

The **RunSQL** method carries out the **RunSQL** action in Visual Basic.

expression.**RunSQL**(SQLStatement, UseTransaction)

*expression* Required. An expression that returns one of the objects in the Applies To list.

*SQLStatement* Required **Variant**. A **string expression** that's a valid SQL statement for an **action query** or a **data-definition query**. It uses an INSERT INTO, DELETE, SELECT...INTO, UPDATE, CREATE TABLE, ALTER TABLE, DROP TABLE, CREATE INDEX, or DROP INDEX statement. Include an **IN** clause if you want to access another database.

*UseTransaction* Optional **Variant**. Use **True** (−1) to include this query in a **transaction**. Use **False** (0) if you don't want to use a transaction. If you leave this argument blank, the default (**True**) is assumed.
Remarks

For more information on how the action and its arguments work, see the action topic.

This method only applies to Microsoft Access databases (.mdb).

The maximum length of the sqlstatement argument is 32,768 characters (unlike the SQL Statement action argument in the Macro window, whose maximum length is 256 characters).

If you leave the usetransaction argument blank, don't use a comma following the sqlstatement argument.
Example

The following example updates the Employees table, changing each sales manager's title to Regional Sales Manager:

Public Sub DoSQL()
    Dim SQL As String

    SQL = "UPDATE Employees " & _
         "SET Employees.Title = 'Regional Sales Manager' " & _
         "WHERE Employees.Title = 'Sales Manager'"

    DoCmd.RunSQL SQL

End Sub
Save Method

The **Save** method carries out the **Save** action in Visual Basic.

\[ \text{expression}.\text{Save(ObjectType, ObjectName)} \]

*expression*  Required. An expression that returns one of the objects in the Applies To list.

**ObjectType**  Optional **AcObjectType**.

AcObjectType can be one of these AcObjectType constants.

- **acDataAccessPage**
- **acDefault**  *default*
- **acDiagram**
- **acForm**
- **acFunction**
- **acMacro**
- **acModule**
- **acQuery**
- **acReport**
- **acServerView**
- **acStoredProcedure**
- **acTable**

**Note**  If closing a module in the Visual Basic Editor (VBE), you must use **acModule** in the *objecttype* argument.

**ObjectName**  Optional **Variant**. A **string expression** that's the valid name of an object of the type selected by the *objecttype* argument.
Remarks

For more information on how the action and its arguments work, see the action topic.

If you leave the `objecttype` and `objectname` arguments blank (the default constant, `acDefault`, is assumed for the `objecttype` argument), Microsoft Access saves the active object. If you leave the `objecttype` argument blank, but enter a name in the `objectname` argument, Microsoft Access saves the active object with the specified name. If you enter an object type in the `objecttype` argument, you must enter an existing object's name in the `objectname` argument.

If you leave the `objecttype` argument blank, but enter a name in the `objectname` argument, you must include the `objecttype` argument's comma.

**Note** You can't use the `Save` method to save any of the following with a new name:

- A form in Form view or Datasheet view.
- A report in Print Preview.
- A module.
- A data access page in Page view.
- A server view in Datasheet view or Print Preview.
- A table in Datasheet view or Print Preview.
- A query in Datasheet view or Print Preview.
- A stored procedure in Datasheet view or Print Preview.

The `Save` method, whether it's run in the current database or in a library database, always saves the specified object or the active object in the database in which the object was created.
**Example**

The following example uses the **Save** method to save the form named "New Employees Form". This form must be open when the code containing this method runs.

**DoCmd.Save acForm, "New Employees Form"**
Scale Method

The **Scale** method defines the coordinate system for a **Report** object.

```plaintext
expression.Scale(flags, x1, y1, x2, y2)
```

**expression**  Required. An expression that returns one of the objects in the Applies To list.

**flags**  Required **Integer**.

**x1**  Required **Single**. A value for the horizontal coordinate that defines the position of the upper-left corner of the object.

**y1**  Required **Single**. A value for the vertical coordinate that defines the position of the upper-left corner of the object.

**x2**  Required **Single**. A value for the horizontal coordinate that defines the position of the lower-right corner of the object.

**y2**  Required **Single**. A value for the vertical coordinate that defines the position of the lower-right corner of the object.
Remarks

You can use this method only in an event procedure or a macro specified by the OnPrint or OnFormat event property for a report section, or the OnPage event property for a report.

You can use the Scale method to reset the coordinate system to any scale you choose. Using the Scale method with no arguments resets the coordinate system to twips. The Scale method affects the coordinate system for the Print method and the report graphics methods, which include the Circle, Line, and PSet methods.
Example

The following example draws a circle with one scale, then uses the Scale method to change the scale and draw another circle with the new scale.

Private Sub Detail_Print(Cancel As Integer, PrintCount As Integer)
    DrawCircle
End Sub

Sub DrawCircle()
    Dim sngHCtr As Single, sngVCtr As Single
    Dim sngNewH As Single, sngNewV As Single
    Dim sngRadius As Single

    Me.ScaleMode = 3 ' Set scale to pixels.
    sngHCtr = Me.ScaleWidth / 2 ' Horizontal center.
    sngVCtr = Me.ScaleHeight / 2 ' Vertical center.
    sngRadius = Me.ScaleHeight / 3 ' Circle radius.
    ' Draw circle.
    Me.Circle (sngHCtr, sngVCtr), sngRadius
    ' New horizontal scale.
    sngNewH = Me.ScaleWidth * 0.9
    ' New vertical scale.
    sngNewV = Me.ScaleHeight * 0.9
    ' Change to new scale.
    Me.Scale(0, 0)-(sngNewH, sngNewV)
    ' Draw circle.
    Me.Circle (sngHCtr + 100, sngVCtr), sngRadius, RGB(0, 256, 0)
End Sub
SelectObject Method

The **SelectObject** method carries out the **SelectObject** action in Visual Basic.

**expression.SelectObject(ObjectType, ObjectName, InDatabaseWindow)**

**expression**  Required. An expression that returns one of the objects in the Applies To list.

**ObjectType**  Required **AcObjectType**.

AcObjectType can be one of these AcObjectType constants.

- acDataAccessPage
- acDefault
- acDiagram
- acForm
- acFunction
- acMacro
- acModule
- acQuery
- acReport
- acServerView
- acStoredProcedure
- acTable

**Note**  The constant **acDefault**, which appears in the Auto List Members list for this argument, is invalid for this argument. You must choose one of the constants listed above.

**ObjectName**  Optional **Variant**. A **string expression** that's the valid name of an object of the type selected by the **objecttype** argument. This is a required argument, unless you specify **True** (-1) for the **indatabasewindow** argument. If you specify **True** for the **indatabasewindow** argument and leave the **objectname** argument blank, Microsoft Access selects the tab in the **Database**
window that corresponds to the database object you specify in the objecttype argument.

**InDatabaseWindow**  Optional Variant. Use True to select the object in the Database window. Use False (0) to select an object that's already open. If you leave this argument blank, the default (False) is assumed.
Remarks

For more information on how the action and its arguments work, see the action topic.

If you set the `indatabasewindow` argument to `True` and leave the `objectname` argument blank, you must include the `objectname` argument's comma. If you leave a trailing argument blank, don't use a comma following the last argument you specify.
Example

The following example selects the form Customers in the Database window:

```
DoCmd.SelectObject acForm, "Customers", True
```
SendObject Method

The **SendObject** method carries out the **SendObject** action in Visual Basic.

```vba
expression.SendObject(ObjectType, ObjectName, OutputFormat, To, Cc, Bcc, Subject, MessageText, EditMessage, TemplateFile)
```

**expression** Required. An expression that returns one of the objects in the Applies To list.

**ObjectType** Optional **AcSendObjectType**.

*AcSendObjectType* can be one of these *AcSendObjectType* constants.
- acSendDataAccessPage
- acSendForm
- acSendModule
- acSendNoObject *default*
- acSendQuery
- acSendReport
- acSendTable

**ObjectName** Optional **Variant**. A string expression that's the valid name of an object of the type selected by the *objecttype* argument. If you want to include the active object in the mail message, specify the object's type with the *objecttype* argument and leave this argument blank. If you leave both the *objecttype* and *objectname* arguments blank (the default constant, acSendNoObject, is assumed for the *objecttype* argument), Microsoft Access sends a message to the electronic mail application without an included database object. If you run Visual Basic code containing the **SendObject** method in a library database, Microsoft Access looks for the object with this name first in the library database, then in the current database.

**OutputFormat** Optional **Variant**.

**To** Optional **Variant**. A string expression that lists the recipients whose names you want to put on the To line in the mail message. Separate the recipient names
you specify in this argument and in the cc and bcc arguments with a semicolon (;) or with the list separator set on the Number tab of the Regional Settings Properties dialog box in Windows Control Panel. If the recipient names aren't recognized by the mail application, the message isn't sent and an error occurs. If you leave this argument blank, Microsoft Access prompts you for the recipients.

Cc  Optional Variant. A string expression that lists the recipients whose names you want to put on the Cc line in the mail message. If you leave this argument blank, the Cc line in the mail message is blank.

Bcc  Optional Variant. A string expression that lists the recipients whose names you want to put on the Bcc line in the mail message. If you leave this argument blank, the Bcc line in the mail message is blank.

Subject  Optional Variant. A string expression containing the text you want to put on the Subject line in the mail message. If you leave this argument blank, the Subject line in the mail message is blank.

MessageText  Optional Variant. A string expression containing the text you want to include in the body of the mail message, after the object. If you leave this argument blank, the object is all that's included in the body of the mail message.

EditMessage  Optional Variant. Use True (–1) to open the electronic mail application immediately with the message loaded, so the message can be edited. Use False (0) to send the message without editing it. If you leave this argument blank, the default (True) is assumed.

TemplateFile  Optional Variant. A string expression that's the full name, including the path, of the file you want to use as a template for an HTML file.
Remarks

For more information on how the action and its arguments work, see the action topic.

Modules can be sent only in MS-DOS Text format, so if you specify `acSendModule` for the `objecttype` argument, you must specify `acFormatTXT` for the `outputformat` argument.

To send a snapshot, set the `ObjectType` argument to `acSendReport` and the `OutputFormat` argument to `acFormatSNP`.

You can leave an optional argument blank in the middle of the syntax, but you must include the argument's comma. If you leave a trailing argument blank, don't use a comma following the last argument you specify.
Example

The following example includes the Employees table in a mail message in Microsoft Excel format and specifies To, Cc, and Subject lines in the mail message. The mail message is sent immediately, without editing.

DoCmd.SendObject acSendTable, "Employees", acFormatXLS, _
  "Nancy Davolio; Andrew Fuller", "Joan Weber", , _
  "Current Spreadsheet of Employees", , False
SetDefaultWorkgroupFile Method

Sets the default workgroup file to the specified file.

\[ \text{expression}.\text{SetDefaultWorkgroupFile} (\text{Path}) \]

*expression*  Required. An expression that returns one of the objects in the Applies To list.

*Path*  Required *String*. The full path and file name of the workgroup file to use as the default.
Remarks

If the file specified by *Path* does not exist, an error occurs.
Example

The following example sets the default workgroup file to the file system.mdw in the directory C:\Documents and Settings\Wendy Vasse\Application Data\Microsoft\Access.

Application.SetDefaultWorkgroupFile
    Path:="C:\Documents and Settings\Wendy Vasse" 
    & "Application Data\Microsoft\Access\system.mdw"
**SetFocus Method**

The **SetFocus** method moves the **focus** to the specified **form**, the specified **control** on the active form, or the specified field on the active **datasheet**.

\[ expression.SetFocus \]

*expression*  Required. An expression that returns one of the objects in the Applies To list.
Remarks

You can use the `SetFocus` method when you want a particular field or control to have the focus so that all user input is directed to this object.

In order to read some of the properties of a control, you need to ensure that the control has the focus. For example, a `text box` must have the focus before you can read its `Text` property.

Other properties can be set only when a control doesn't have the focus. For example, you can't set a control's `Visible` or `Enabled` properties to `False` (0) when that control has the focus.

You can also use the `SetFocus` method to navigate in a form according to certain conditions. For example, if the user selects `Not applicable` for the first of a set of questions on a form that's a questionnaire, your Visual Basic code might then automatically skip the questions in that set and move the focus to the first control in the next set of questions.

You can move the focus only to a visible control or form. A form and controls on a form aren't visible until the form's `Load` event has finished. Therefore, if you use the `SetFocus` method in a form's `Load` event to move the focus to that form, you must use the `Repaint` method before the `SetFocus` method.

You can't move the focus to a control if its `Enabled` property is set to `False`. You must set a control's `Enabled` property to `True` (–1) before you can move the focus to that control. You can, however, move the focus to a control if its `Locked` property is set to `True`.

If a form contains controls for which the `Enabled` property is set to `True`, you can't move the focus to the form itself. You can only move the focus to controls on the form. In this case, if you try to use `SetFocus` to move the focus to a form, the focus is set to the control on the form that last received the focus.

Tip

You can use the `SetFocus` method to move the focus to a `subform`, which is a type of control. You can also move the focus to a control on a subform by using
the `SetFocus` method twice, moving the focus first to the subform and then to the control on the subform.
Example

The following example uses the **SetFocus** method to move the focus to an EmployeeID text box on an Employees form:

```
Forms!Employees!EmployeeID.SetFocus
```
SetHiddenAttribute Method

The `SetHiddenAttribute` method sets the hidden attribute of an Access object.

`expression.SetHiddenAttribute(ObjectType, ObjectName, fHidden)`

`expression` Required. An expression that returns an `Application` object.

`ObjectType` Required `AcObjectType`. You must enter a constant for the `ObjectType` argument; `acDefault` is not a valid entry.

`ObjectName` Required `String`. A `string expression` identifying the name of the Access object.

`fHidden` Required `Boolean`. `True` sets the hidden attribute and `False` clears the attribute.
Remarks

Together with the **GetHiddenAttribute** method, the **SetHiddenAttribute** method provides a means of changing an object's visibility from Visual Basic code. With these methods, you can set or read the Hidden property available in the object's **Properties** dialog box.

To set this option by using the **SetHiddenAttribute** method, specify **True** or **False** for the setting, as in the following example.

```vbnet
Application.SetHiddenAttribute acTable,"Customers", True
```
SetMenuItem Method

The `SetMenuItem` method carries out the `SetMenuItem` action in Visual Basic.

```vba
expression.SetMenuItem(MenuIndex, CommandIndex, SubcommandIndex, Flag)
```

**expression** Required. An expression that returns one of the objects in the Applies To list.

**MenuIndex** Required **Variant**. An integer, counting from 0, that's the valid index of a menu on the custom menu bar or global menu bar for the active window, as defined in the menu bar macro for the custom menu bar or global menu bar. If you select a menu with this argument and leave the `commandindex` and `subcommandindex` arguments blank (or set them to –1), you can enable or disable the menu name itself. You can't, however, check or uncheck a menu name (Microsoft Access ignores the `acMenuCheck` and `acMenuUncheck` settings for the `flag` argument for menu names).

**CommandIndex** Optional **Variant**. An integer, counting from 0, that's the valid index of a command on the menu selected by the `menuindex` argument, as defined in the macro group that defines the selected menu for the custom menu bar or global menu bar for the active window.

**SubcommandIndex** Optional **Variant**. An integer, counting from 0, that's the valid index of a subcommand in the submenu selected by the `commandindex` argument, as defined in the macro group that defines the selected submenu for the custom menu bar or global menu bar for the active window.

**Flag** Optional **AcMenuType**.

`AcMenuType` can be one of these `AcMenuType` constants.

- `acMenuCheck`
- `acMenuGray`
- `acMenuUncheck`
**acMenuUngray default**

If you leave this argument blank, the default constant *(acMenuUngray)* is assumed.
Remarks

For more information on how the action and its arguments work, see the action topic.

Note  The **SetMenuItem** method works only with custom [menu bars](https://developer.mozilla.org/en-US/docs/Web/API/Menu) and [global menu bars](https://developer.mozilla.org/en-US/docs/Web/API/Menu) created by using [menu bar macros](https://developer.mozilla.org/en-US/docs/Web/API/Menu). The **SetMenuItem** method is included in this version of Microsoft Access only for compatibility with versions prior to Microsoft Access 97. It doesn't work with the new [command bars](https://developer.mozilla.org/en-US/docs/Web/API/Menu) functionality. In the current version of Microsoft Access, you must use the properties and methods of the [CommandBars](https://developer.mozilla.org/en-US/docs/Web/API/Menu) collection object to enable or disable top level menu items.

You can leave an optional argument blank in the middle of the syntax, but you must include the argument's comma. If you leave a trailing argument blank, don't use a comma following the last argument you specify.
Example

The following example uses the `SetMenuItem` method to disable the second command in the first menu on the custom menu bar for the active window:

```vba
DoCmd.SetMenuItem 0, 1, , acMenuGray
```
SetOption Method

The **SetOption** method sets the current value of an option in the **Options** dialog box.

\[expression.SetOption(\text{OptionName}, \text{Setting})\]

where:

**expression** Required. An expression that returns one of the objects in the Applies To list.

**OptionName** Required **String**. The name of the option. For a list of optionname argument strings, see Set Options from Visual Basic.

**Setting** Required **Variant**. A **Variant** value corresponding to the option setting. The value of the setting argument depends on the possible settings for a particular option.
Remarks

The **SetOption** method provides a means of changing environment options from **Visual Basic** code. With this method, you can set or read any option available in the **Options** dialog box, except for options on the **Modules** tab.

The available option settings depend on the type of option being set. There are three general types of options:

- **Yes/No** options that can be set by selecting or clearing a [check box](#).
- Options that can be set by entering a [string](#) or numeric value.
- Predefined options that can be chosen from a [list box](#), [combo box](#), or [option group](#).

For options that the user sets by selecting or clearing a check box, using the **SetOption** method, specify **True** or **False** for the **setting** argument, as in the following example:

```vbnet
Application.SetOption "Show Status Bar", True
```

To set a type of option using the **SetOption** method, specify the string or numeric value that would be typed in the dialog box. The following example sets the default form template to OrderTemplate:

```vbnet
Application.SetOption "Form Template", "OrderTemplate"
```

For options with settings that are choices in list boxes or combo boxes, specify the option's numeric position within the list as the **setting** argument for the **SetOption** method. The following example sets the **Default Field Type** option to AutoNumber:

```vbnet
Application.SetOption "Default Field Type", 5
```

To set an option that's a member of an option group, specify the index number of the option within the group. The following example sets **Selection Behavior** to
Fully Enclosed:

`Application.SetOption "Selection Behavior", 1`

**Notes**

- When you use the `SetOption` method to set an option in the `Options` dialog box, you don't need to specify the individual tab on which the option is found.

- You can't use the `SetOption` method to read or set any of the options found on the `Module` tab of the `Options` dialog box.

- If your database may run on a version of Microsoft Access for a language other than the one in which you created it, then you must supply the arguments for the `SetOption` method in English.

When you quit Microsoft Access, you can reset all options to their original settings by using the `SetOption` method on all changed options. You may want to create public variables to store the values of the original settings. You might include code to reset options in the Close event procedure for a form, or in a custom exit procedure that the user must run to quit the application.
SetWarnings Method

The **SetWarnings** method carries out the **SetWarnings** action in Visual Basic.

`expression.SetWarnings(WarningsOn)`

**expression** Required. An expression that returns one of the objects in the Applies To list.

**WarningsOn** Required **Variant**. Use **True** (–1) to turn on the display of system messages and **False** (0) to turn it off.
Remarks

For more information on how the action and its argument work, see the action topic.

If you turn the display of system messages off in Visual Basic, you must turn it back on, or it will remain off, even if the user presses CTRL+BREAK or Visual Basic encounters a breakpoint. You may want to create a macro that turns the display of system messages on and then assign that macro to a key combination or a custom menu command. You could then use the key combination or menu command to turn the display of system messages on if it has been turned off in Visual Basic.
Example

The following example turns the display of system messages off:

DoCmd.SetWarnings False
ShowAllRecords Method

The **ShowAllRecords** method carries out the **ShowAllRecords** action in Visual Basic.

\[expression.ShowAllRecords\]

*expression* Required. An expression that returns a **DoCmd** object.
Remarks

This method removes any existing filters that may exist on the current table, query, or form. It can be called directly using the syntax DoCmd.ShowAllRecords.

Note  This method only applies to tables, queries, and forms within a Microsoft database (.mdb).
ShowToolbar Method

The ShowToolbar method carries out the ShowToolbar action in Visual Basic.

expression.ShowToolbar(ToolbarName, Show)

expression Required. An expression that returns one of the objects in the
Applies To list.

ToolbarName Required Variant. A string expression that's the valid name of a
Microsoft Access built-in toolbar or a custom toolbar you've created. If you run
Visual Basic code containing the ShowToolbar method in a library database,
Microsoft Access looks for the toolbar with this name first in the library
database, then in the current database.

Show Optional AcShowToolbar.

AcShowToolbar can be one of these AcShowToolbar constants.

acToolbarNo
acToolbarWhereApprop
acToolbarYes default

If you leave this argument blank, the default constant (acToolbarYes) is
assumed.
Remarks

For more information on how the action and its arguments work, see the action topic.

If you leave the show argument blank, don't use a comma following the toolbarname argument.
Example

The following example displays the custom toolbar named CustomToolbar in all Microsoft Access windows that become active:

DoCmd.ShowToolbar "CustomToolbar", acToolbarYes
SizeToFit Method

You can use the `SizeToFit` method to size a control so it fits the text or image that it contains.

`expression.SizeToFit`

`expression` Required. An expression that returns one of the objects in the Applies To list.
Remarks

For example, you can apply the `SizeToFit` method to a command button that is too small to display all the text in its Caption property.

The use of the `SizeToFit` method is equivalent to selecting a control on a form or report, pointing to Size on the Format menu, and clicking To Fit. You can apply the `SizeToFit` method to controls only in form Design view or report Design view.

The `SizeToFit` method will make a control larger or smaller, depending on the size of the text or image it contains.

You can use the `SizeToFit` method in conjunction with the `CreateControl` method to size new controls that you have created in code.

**Note**  Not all controls that contain text or an image can be sized by the `SizeToFit` method. Several controls are bound to data that can vary in size from one record to the next. These controls include the text box, list box, combo box, and bound object frame controls. The `SizeToFit` method does not apply to controls on data access pages.
Example

The following example creates a new form and creates a new command button on the form. The procedure then sets the control's Caption property and sizes the control to fit the caption.

Sub SizeNewControl()
    Dim frm As Form, ctl As Control

    ' Create new form.
    Set frm = CreateForm
    ' Create new command button.
    Set ctl = CreateControl(frm.Name, _
        acCommandButton, , , , 500, 500)
    ' Restore form.
    DoCmd.Restore
    ' Set control's Caption property.
    ctl.Caption = "Extremely Long Control Caption"
    ' Size control to fit caption.
    ctl.SizeToFit
End Sub
StringFromGUID Function

The StringFromGUID function converts a GUID, which is an array of type Byte, to a string.

StringFromGUID(guid)

The StringFromGUID function has the following argument.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>guid</td>
<td>An array of Byte data used to uniquely identify an application, component, or item of data to the operating system.</td>
</tr>
</tbody>
</table>
Remarks

The Microsoft Jet database engine stores GUIDs as arrays of type Byte. However, Microsoft Access can't return Byte data from a control on a form or report. In order to return the value of a GUID from a control, you must convert it to a string. To convert a GUID to a string, use the StringFromGUID function. To convert a string back to a GUID, use the GUIDFromString function.

For example, you may need to refer to a field that contains a GUID when using database replication. To return the value of a control on a form bound to a field that contains a GUID, use the StringFromGUID function to convert the GUID to a string.

Note that in order to bind a control to the s_GUID field of a replicated table, you must select the System objects check box on the View tab of the Options dialog box (Tools menu).
Example

The following example returns the value of the s_GUID control on an Employees form in string form and assigns it to a string variable. The s_GUID control is bound to the s_GUID field, one of the system fields added to each replicated table in a replicated database.

Public Sub StringValueOfGUID()
    Dim ctl As Control
    Dim strGUID As String

    ' Get the GUID.
    Set ctl = Forms!Employees!s_GUID
    Debug.Print TypeName(ctl.Value)

    ' Convert the GUID to a string.
    strGUID = StringFromGUID(ctl.Value)
    Debug.Print TypeName(strGUID)

End Sub
SysCmd Method

You can use the **SysCmd** method to, display a progress meter or optional specified text in the status bar, return information about Microsoft Access and its associated files, or return the state of a specified [database object](#) (to indicate whether the object is open, is a new object, or has been changed but not saved).

**Variant.**

`expression.SysCmd(Action, Argument2, Argument3)`

*expression* Required. An expression that returns one of the objects in the Applies To list.

*Action* Required **AcSysCmdAction**. One of the following intrinsic constants identifying the type of action to take. The following set of constants applies to a progress meter. The **SysCmd** method returns a **Null** if these actions are successful. Otherwise, Microsoft Access generates a run-time error.

AcSysCmdAction can be one of these AcSysCmdAction constants.

**acSysCmdAccessDir**. Returns the name of the directory where Msaccess.exe is located.

**acSysCmdAccessVer**. Returns the version number of Microsoft Access.

**acSysCmdClearHelpTopic**

**acSysCmdClearStatus**. The following constant provides information on the state of a database object.

**acSysCmdGetObjectState**. Returns the state of the specified database object. You must specify *argument1* and *argument2* when you use this *action* value.

**acSysCmdGetWorkgroupFile**. Returns the path to the workgroup file (System.mdw).

**acSysCmdIniFile**. Returns the name of the .ini file associated with Microsoft Access.

**acSysCmdInitMeter**. Initializes the progress meter. You must specify the *argument1* and *argument2* arguments when you use this action.

**acSysCmdProfile**. Returns the /profile setting specified by the user when starting Microsoft Access from the command line.
**acSysCmdRemoveMeter.** Removes the progress meter.

**acSysCmdRuntime.** Returns True (–1) if a run-time version of Microsoft Access is running.

**acSysCmdSetStatus.** Sets the status bar text to the text argument.

**acSysCmdUpdateMeter.** Updates the progress meter with the specified value. You must specify the text argument when you use this action.

*Argument2* Optional Variant. A string expression identifying the text to be displayed left-aligned in the status bar. This argument is required when the action argument is acSysCmdInitMeter, acSysCmdUpdateMeter, or acSysCmdSetStatus; this argument isn't valid for other action argument values.

*Note* When you specify the acSysCmdGetObjectState value for the Action parameter, then you must specify the appropriate acObjectType.

- acTable
- acQuery
- acForm
- acReport
- acMacro
- acModule
- acDataAccessPage
- acDefault
- acDiagram
- acServerView
- acStoreProcedure

This argument isn't valid for other action argument values.

*Argument3* Optional Variant. A numeric expression that controls the display
of the progress meter. This argument is required when the action argument is acSysCmdInitMeter; this argument isn't valid for other action argument values.

Note  When you specify the acSysCmdGetObjectState value for the Action parameter, then you must specify the name of the database object.
Remarks

For example, if you are building a custom wizard that creates a new form, you can use the `SysCmd` method to display a progress meter indicating the progress of your wizard as it constructs the form.

By calling the `SysCmd` method with the various progress meter actions, you can display a progress meter in the status bar for an operation that has a known duration or number of steps, and update it to indicate the progress of the operation.

To display a progress meter in the status bar, you must first call the `SysCmd` method with the `acSysCmdInitMeter` action argument, and the `text` and `value` arguments. When the `action` argument is `acSysCmdInitMeter`, the `value` argument is the maximum value of the meter, or 100 percent.

To update the meter to show the progress of the operation, call the `SysCmd` method with the `acSysCmdUpdateMeter` action argument and the `value` argument. When the `action` argument is `acSysCmdUpdateMeter`, the `SysCmd` method uses the `value` argument to calculate the percentage displayed by the meter. For example, if you set the maximum value to 200 and then update the meter with a value of 100, the progress meter will be half-filled.

You can also change the text that's displayed in the status bar by calling the `SysCmd` method with the `acSysCmdSetStatus` action argument and the `text` argument. For example, during a sort you might change the text to "Sorting...". When the sort is complete, you would reset the status bar by removing the text. The `text` argument can contain approximately 80 characters. Because the status bar text is displayed by using a proportional font, the actual number of characters you can display is determined by the total width of all the characters specified by the `text` argument.

As you increase the width of the status bar text, you decrease the length of the meter. If the text is longer than the status bar and the `action` argument is `acSysCmdInitMeter`, the `SysCmd` method ignores the text and doesn't display anything in the status bar. If the text is longer than the status bar and the `action` argument is `acSysCmdSetStatus`, the `SysCmd` method truncates the text to fit the status bar.
You can't set the status bar text to a zero-length string (" "). If you want to remove the existing text from the status bar, set the text argument to a single space. The following examples illustrate ways to remove the text from the status bar:

```javascript
varReturn = SysCmd(acSysCmdInitMeter, " ", 100)
varReturn = SysCmd(acSysCmdSetStatus, " ")
```

If the progress meter is already displayed when you set the text by calling the `SysCmd` method with the `acSysCmdSetStatus` action argument, the `SysCmd` method automatically removes the meter.

Call the `SysCmd` method with other actions to determine system information about Microsoft Access, including which version number of Microsoft Access is running, whether it is a run-time version, the location of the Microsoft Access executable file, the setting for the `/profile` argument specified in the command line, and the name of an .ini file associated with Microsoft Access.

**Note** Both general and customized settings for Microsoft Access are now stored in the Windows Registry, so you probably won't need an .ini file with your Microsoft Access application. The `acSysCmdIniFile` action argument exists for compatibility with earlier versions of Microsoft Access.

Call the `SysCmd` method with the `acSysCmdGetObjectState` action argument and the `objecttype` and `objectname` arguments to return the state of a specified database object. An object can be in one of four possible states: not open or nonexistent, open, new, or changed but not saved.

For example, if you are designing a wizard that inserts a new field in a table, you need to determine whether the structure of the table has been changed but not yet saved, so that you can save it before modifying its structure. You can check the value returned by the `SysCmd` method to determine the state of the table.

The `SysCmd` method with the `acSysCmdGetObjectState` action argument can return any combination of the following constants.

<table>
<thead>
<tr>
<th>Constant</th>
<th>State of database object</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>acObjStateOpen</td>
<td>Open</td>
<td>1</td>
</tr>
<tr>
<td>acObjStateDirty</td>
<td>Changed but not saved</td>
<td>2</td>
</tr>
</tbody>
</table>
Note If the object referred to by the objectname argument is either not open or doesn’t exist, the SysCmd method returns a value of zero.

The following code can be used to enable the use of your ActiveX control in expressions when the ActiveX control has been added to a form:

SysCmd 14, "<ActiveX Control GUID>"

Note Replace <ActiveX Control GUID> with the globally unique identifier (GUID) that identifies the ActiveX control that you want to enable in expressions.

Note You cannot remove an ActiveX control once it has been added to the list of allowed controls.
TextHeight Method

The **TextHeight** method returns the height of a text string as it would be printed in the current font of a **Report** object.

*expression*.TextHeight(*Expr*)

*expression*  Required. An expression that returns one of the objects in the Applies To list.

*Expr*  Required **String**. The text string for which the text height will be determined.
Remarks

You can use the `TextHeight` method to determine the amount of vertical space a text string will require in the current font when the report is formatted and printed. For example, a text string formatted in 9-point Arial will require a different amount of space than one formatted in 12-point Courier. To determine the current font and font size for text in a report, check the settings for the report's `FontName` and `FontSize` properties.

The value returned by the `TextHeight` method is expressed in terms of the coordinate system in effect for the report, as defined by the `Scale` method. You can use the `ScaleMode` property to determine the coordinate system currently in effect for the report.

If the `stexpr` argument contains embedded carriage returns, the `TextHeight` method returns the cumulative height of the lines, including the leading space above and below each line. You can use the value returned by the `TextHeight` method to calculate the necessary space and positioning for multiple lines of text within a report.
Example

The following example uses the **TextHeight** and **TextWidth** methods to determine the amount of vertical and horizontal space required to print a text string in the report's current font.

To try this example in Microsoft Access, create a new report. Set the **OnPrint** property of the Detail section to [Event Procedure]. Enter the following code in the report's module, then switch to Print Preview.

```vba
Private Sub Detail_Print(Cancel As Integer, _
    PrintCount As Integer)
    ' Set unit of measure to twips (default scale).
    Me.Scalemode = 1
    ' Print name and font size of report font.
    Debug.Print "Report Font: "; Me.FontName
    Debug.Print "Report Font Size: "; Me.FontSize
    ' Print height and width required for text string.
    Debug.Print "Text Height (Twips): "; _
        Me.TextHeight("Product Report")
    Debug.Print "Text Width (Twips): "; _
        Me.TextWidth("Product Report")
End Sub
```
TextWidth Method

The **TextWidth** method returns the width of a text string as it would be printed in the current font of a **Report** object.

`expression.TextWidth(Expr)`

- **expression**   Required. An expression that returns one of the objects in the Applies To list.

- **Expr**   Required **String**. The text string for which the text width will be determined.
Remarks

You can use the **TextWidth** method to determine the amount of horizontal space a text string will require in the current font when the report is formatted and printed. For example, a text string formatted in 9-point Arial will require a different amount of space than one formatted in 12-point Courier. To determine the current font and font size for text in a report, check the settings for the report's **FontName** and **FontSize** properties.

The value returned by the **TextWidth** method is expressed in terms of the coordinate system in effect for the report, as defined by the **Scale** method. You can use the **ScaleMode** property to determine the coordinate system currently in effect for the report.

If the *strexp* argument contains embedded carriage returns, the **TextWidth** method returns the width of the longest line, from the beginning of the line to the carriage return. You can use the value returned by the **TextWidth** method to calculate the necessary space and positioning for multiple lines of text within a report.
TransferDatabase Method

The `TransferDatabase` method carries out the `TransferDatabase` action in Visual Basic.

`expression.TransferDatabase(TransferType, DatabaseType, DatabaseName, ObjectType, Source, Destination, StructureOnly, StoreLogin)`

`expression` Required. An expression that returns one of the objects in the Applies To list.

`TransferType` Optional `AcDataTransferType`.

`AcDataTransferType` can be one of these `AcDataTransferType` constants.

- `acExport`
- `acImport` `default`
- `acLink`

If you leave this argument blank, the default constant (`acImport`) is assumed.

**Note** The `acLink` transfer type is not supported for Microsoft Access projects (.adp).

`DatabaseType` Optional `Variant`. A `string expression` that's the name of one of the types of databases you can use to import, export, or link data.

Types of databases:
- Microsoft Access (default)
- Jet 2.x
- Jet 3.x
- dBase III
- dBase IV
- dBase 5.0
- Paradox 3.x
- Paradox 4.x
In the **Macro window**, you can view the database types in the list for the Database Type action argument of the TransferDatabase action.

**DatabaseName**  Optional **Variant.** A string expression that’s the full name, including the path, of the database you want to use to import, export, or link data.

**ObjectType**  Optional **AcObjectType.**

AcObjectType can be one of these AcObjectType constants.

- `acDataAccessPage`
- `acDefault`
- `acDiagram`
- `acForm`
- `acFunction`
- `acMacro`
- `acModule`
- `acQuery`
- `acReport`
- `acServerView`
- `acStoredProcedure`
- `acTable` default

This is the type of object whose data you want to import, export, or link. You can specify an object other than **acTable** only if you are importing or exporting data between two Microsoft Access databases. If you are exporting the results of a Microsoft Access **select query** to another type of database, specify **acTable** for this argument.

If you leave this argument blank, the default constant (**acTable**) is assumed.
**Note**  The constant `acDefault`, which appears in the Auto List Members list for this argument, is invalid for this argument. You must choose one of the constants listed above.

**Source**  Optional Variant. A string expression that's the name of the object whose data you want to import, export, or link.

**Destination**  Optional Variant. A string expression that's the name of the imported, exported, or linked object in the destination database.

**StructureOnly**  Optional Variant. Use **True** (–1) to import or export only the structure of a database table. Use **False** (0) to import or export the structure of the table and its data. If you leave this argument blank, the default (**False**) is assumed.

**StoreLogin**  Optional Variant. Use **True** to store the login identification (ID) and password for an ODBC database in the connection string for a linked table from the database. If you do this, you don't have to log in each time you open the table. Use **False** if you don't want to store the login ID and password. If you leave this argument blank, the default (**False**) is assumed. This argument is available only in Visual Basic.
Remarks

For more information on how the action and its arguments work, see the action topic.

You can leave an optional argument blank in the middle of the syntax, but you must include the argument's comma. If you leave a trailing argument blank, don't use a comma following the last argument you specify.

The administrator of an ODBC database can disable the feature provided by the `saveloginid` argument, requiring all users to enter the login ID and password each time they connect to the ODBC database.

**Note** You can also use **ActiveX Data Objects (ADO)** to create a link by using the **ActiveConnection** property for the **Recordset** object.

To link to a Windows SharePoint Services list, you must specify the following syntax for the **DatabaseName** argument:

```
WSS;HDR=NO;IMEX=2;DATABASE=<HTTP Server path>;LIST=<List GUID>;VIEW=
```

where `<HTTP Server path>` is the path to the Windows SharePoint Services site, `<List GUID>` is the globally unique identifier of the list, and `<List name>` is the friendly name of the list.

The easiest way to obtain this information is to perform the following steps:

1. In the database window, create a new table that links to the Windows SharePoint Services list.
2. Open the linked table in Design view.
3. Click **Properties** on the **View** menu.

The required string is listed in the **Description** property.
Example

The following example imports the NW Sales for April report from the Microsoft Access database NWSales.mdb into the Corporate Sales for April report in the current database:

DoCmd.TransferDatabase acImport, "Microsoft Access", _
  "C:\My Documents\NWSales.mdb", acReport, "NW Sales for April", _
  "Corporate Sales for April"

The next example links the ODBC database table Authors to the current database:

DoCmd.TransferDatabase acLink, "ODBC Database", _
  "ODBC;DSN=DataSource1;UID=User2;PWD=www;LANGUAGE=us_english;" _
  & "DATABASE=pubs", acTable, "Authors", "dboAuthors"

The following example exports the contents of the Customers table to a new list named Customer List on the "http://example/WSSSite" Windows SharePoint Services site.

  objecttype:=acTable, Source:="Customers", _
  Destination:="Customer List", structureonly:=

TransferSpreadsheet Method

The TransferSpreadsheet method carries out the TransferSpreadsheet action in Visual Basic.

expression. TransferSpreadsheet(TransferType, SpreadsheetType, TableName, FileName, HasFieldNames, Range, UseOA)

expression  Required. An expression that returns one of the objects in the Applies To list.

TransferType  Optional  AcDataTransferType.

AcDataTransferType can be one of these AcDataTransferType constants.
acExport
acImport default
acLink

If you leave this argument blank, the default constant (acImport) is assumed.

SpreadsheetType  Optional  AcSpreadSheetType.

AcSpreadSheetType can be one of these AcSpreadSheetType constants.
acSpreadsheetTypeExcel3
acSpreadsheetTypeExcel4
acSpreadsheetTypeExcel5
acSpreadsheetTypeExcel7
acSpreadsheetTypeExcel8 default
acSpreadsheetTypeExcel9 default
acSpreadsheetTypeLotusWJ2 - Japanese version only
acSpreadsheetTypeLotusWK1
acSpreadsheetTypeLotusWK3
acSpreadsheetTypeLotusWK4
**Note** You can import from and link to Lotus .WK4 files, but you can't export Microsoft Access data to this spreadsheet format. Microsoft Access also no longer supports importing, exporting, or linking data from Lotus .WKS or Microsoft Excel version 2.0 spreadsheets by using this method.

If you leave this argument blank, the default constant *(acSpreadsheetTypeExcel8)* is assumed.

**TableName** Optional **Variant**. A **string expression** that's the name of the Microsoft Access table you want to import spreadsheet data into, export spreadsheet data from, or link spreadsheet data to, or the Microsoft Access **select query** whose results you want to export to a spreadsheet.

**FileName** Optional **Variant**. A string expression that's the file name and path of the spreadsheet you want to import from, export to, or link to.

**HasFieldNames** Optional **Variant**. Use **True** (−1) to use the first row of the spreadsheet as field names when importing or linking. Use **False** (0) to treat the first row of the spreadsheet as normal data. If you leave this argument blank, the default (**False**) is assumed. When you export Microsoft Access table or select query data to a spreadsheet, the field names are inserted into the first row of the spreadsheet no matter what you enter for this argument.

**Range** Optional **Variant**. A string expression that's a valid range of cells or the name of a range in the spreadsheet. This argument applies only to importing. Leave this argument blank to import the entire spreadsheet. When you export to a spreadsheet, you must leave this argument blank. If you enter a range, the export will fail.

**UseOA** Optional **Variant**.
Remarks

For more information on how the action and its arguments work, see the action topic.

You can leave an optional argument blank in the middle of the syntax, but you must include the argument's comma. If you leave a trailing argument blank, don't use a comma following the last argument you specify.

Note  You can also use ActiveX Data Objects (ADO) to create a link by using the ActiveConnection property for the Recordset object.
Example

The following example imports the data from the specified range of the Lotus spreadsheet Newemps.wk3 into the Microsoft Access Employees table. It uses the first row of the spreadsheet as field names.

DoCmd.TransferSpreadsheet acImport, 3, _
"Employees","C:\Lotus\Newemps.wk3", True, "A1:G12"
TransferSQLDatabase Method

Transfers the entire specified Microsoft SQL Server database to another SQL Server database.

```
expression.TransferSQLDatabase(Server, Database, UseTrustedConnection, Login, Password, TransferCopyData)
```

**expression** Required. An expression that returns a **DoCmd** object.

**Server** Required **Variant**. The name of the SQL Server to which the database will be transferred.

**Database** Required **Variant**. The name of the new database on the specified server.

**UseTrustedConnection** Optional **Variant**. **True** if the current connection is using a login with system administrator privileges. If this argument is not **True**, you must specify a login and password in the **Login** and **Password** arguments.

**Login** Optional **Variant**. The name of a login on the destination server with system administrator privileges. If **UseTrustedConnection** is **True**, this argument is ignored.

**Password** Optional **Variant**. The password for the login specified in **Login**. If **UseTrustedConnection** is **True**, this argument is ignored.

**TransferCopyData** Optional **Variant**. **True** if all data in the database is transferred to the destination database. If this argument is not **True**, only the database schema will be transferred.
Remarks

The following conditions must be met or else an error occurs:

- The current and destination servers are SQL Server version 7.0 or later.
- The user has system administrator login rights on the destination server.
- The destination database doesn't already exist on the destination server.
Example

This example transfers the current SQL Server database to a new SQL Server database called Inventory on the server MainOffice. (It is assumed that the user has system administrator privileges on MainOffice.) The data is copied along with the database schema.

```vba
DoCmd.TransferCompleteSQLDatabase _
    Server:="MainOffice", _
    Database:="Inventory", _
    UseTrustedConnection:=True, _
    TransferCopyData:=False
```
TransferText Method

The `TransferText` method carries out the `TransferText` action in Visual Basic.

```vba
expression.TransferText(TransferType, SpecificationName, TableName, FileName, HasFieldNames, HTMLTableName, CodePage)
```

`expression` Required. An expression that returns one of the objects in the Applies To list.

`TransferType` Optional `AcTextTransferType`.

`AcTextTransferType` can be one of these `AcTextTransferType` constants.
- `acExportDelim`
- `acExportFixed`
- `acExportHTML`
- `acExportMerge`
- `acImportDelim` `default`
- `acImportFixed`
- `acImportHTML`
- `acLinkDelim`
- `acLinkFixed`
- `acLinkHTML`

If you leave this argument blank, the default constant (`acImportDelim`) is assumed.

**Notes**

Only `acImportDelim`, `acImportFixed`, `acExportDelim`, `acExportFixed`, or `acExportMerge` transfer types are supported in a Microsoft Access project (.adp).

`SpecificationName` Optional Variant. A `string expression` that's the name of an
import or export specification you've created and saved in the current database. For a fixed-width text file, you must either specify an argument or use a schema.ini file, which must be stored in the same folder as the imported, linked, or exported text file. To create a schema file, you can use the text import/export wizard to create the file. For delimited text files and Microsoft Word mail merge data files, you can leave this argument blank to select the default import/export specifications.

**TableName** Optional **Variant**. A string expression that's the name of the Microsoft Access table you want to import text data to, export text data from, or link text data to, or the Microsoft Access **query** whose results you want to export to a text file.

**FileName** Optional **Variant**. A string expression that's the full name, including the path, of the text file you want to import from, export to, or link to.

**HasFieldNames** Optional **Variant**. Use **True** (–1) to use the first row of the text file as field names when importing, exporting, or linking. Use **False** (0) to treat the first row of the text file as normal data. If you leave this argument blank, the default (**False**) is assumed. This argument is ignored for Microsoft Word mail merge data files, which must always contain the field names in the first row.

**HTMLTableName** Optional **Variant**. A string expression that's the name of the table or list in the HTML file that you want to import or link. This argument is ignored unless the **transfertype** argument is set to **acImportHTML** or **acLinkHTML**. If you leave this argument blank, the first table or list in the HTML file is imported or linked. The name of the table or list in the HTML file is determined by the text specified by the **<CAPTION>** tag, if there's a **<CAPTION>** tag. If there's no **<CAPTION>** tag, the name is determined by the text specified by the **<TITLE>** tag. If more than one table or list has the same name, Microsoft Access distinguishes them by adding a number to the end of each table or list name; for example, Employees1 and Employees2.

**CodePage** Optional **Variant**. A **Long** value indicating the character set of the code page.
Remarks

For more information on how the action and its arguments work, see the action topic.

You can leave an optional argument blank in the middle of the syntax, but you must include the argument's comma. If you leave a trailing argument blank, don't use a comma following the last argument you specify.

Note  You can also use ActiveX Data Objects (ADO) to create a link by using ActiveConnection property for the Recordset object.
Example

The following example exports the data from the Microsoft Access table External Report to the delimited text file April.doc by using the specification Standard Output:

DoCmd.TransferText acExportDelim, "Standard Output", _
   "External Report", "C:\Txtfiles\April.doc"
TransformXML Method

Applies an Extensible Stylesheet Language (XSL) stylesheet to an XML data file and writes the resulting XML to an XML data file.

expression. TransformXML(DataSource, TransformSource, OutputTarget, WellFormedXMLOutput, ScriptOption)

expression  Required. An expression that returns an Application object.

DataSource  Required String. The name and path of the XML file to import.

TransformSource  Required String. The name and path to the XSL file to apply to the XML data file.

OutputTarget  Required String. The file name and path for the resulting XML data file after applying the XSL file.

WellFormedXMLOutput  Optional Boolean. Setting this argument to True will create a well-formed XML file. Setting this argument to False will encode the resulting XML file in UTF-16 format. The default value is False.

ScriptOption  Optional AcTransformXMLScriptOption. Specifies the action taken if the XSL file contains scripting code.

AcTransformXMLScriptOption can be one of the following constants

acDisableScript The script is disabled.
acEnableScript The script is enabled.
acPromptScript default The user is prompted to disable or enable the script.
Remarks

Setting the **WellFormedXMLOutput** argument to **True** will result in a run-time error if the XSL file that you apply does not result in well-formed XML.
Undo Method

You can use the **Undo** method to reset a **control** or **form** when its value has been changed.

`expressionUndo`

`expression` Required. An expression that returns one of the objects in the Applies To list.
Remarks

For example, you can use the **Undo** method to clear a change to a record that contains an invalid entry.

If the **Undo** method is applied to a form, all changes to the current record are lost. If the **Undo** method is applied to a control, only the control itself is affected.

This method must be applied before the form or control is updated. You may want to include this method in a form's **BeforeUpdate** event or in a control's **Change** event.

The **Undo** method offers an alternative to using the **SendKeys** statement to send the value of the ESC key in an event procedure.
Example

The following example shows how you can use the **Undo** method within a control's **Change** event procedure to force a field named LastName to reset to its original value, if it changed.

```vbscript
Private Sub LastName_Change()
    Me!LastName.Undo
End Sub
```

The next example uses the **Undo** method to reset all changes to a form before the form is updated.

```vbscript
Private Sub Form_BeforeUpdate(Cancel As Integer)
    Me.Undo
End Sub
```
Show All
UpdateDependencyInfo Method

Updates the dependency information for the database.

expression.UpdateDependencyInfo()

expression Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **UpdateDependencyInfo** method opens, saves, and then closes every table, query, form, and report in the database; no messages are presented to the user.

This method will return a run-time error if any of the following conditions are true:

- This method is being called from an [Access project (.adp)](#).
- Any [database objects](#) are open.
UseDefaultFolderSuffix Method

You can use the **UseDefaultFolderSuffix** method to set the folder suffix for the specified data access page to the default suffix for the language support you have selected or installed.

\[ expression.UseDefaultFolderSuffix \]

*expression* Required. An expression that returns one of the objects in the Applies To list.
Remarks

Microsoft Access uses the folder suffix when you save a document as a Web page, use long file names, and choose to save supporting files in a separate folder (that is, if the `UseLongFileNames` and `OrganizeInFolder` properties are set to `True`).

The suffix appears in the folder name after the data access page name. For example, if the data access page is called "Book1" and the language is English, the folder name is Book1_files. The available folder suffixes are listed in the `FolderSuffix` property topic.
Example

This example sets the folder suffix for the data access page ("Inventory") to the default suffix.

DataAccessPages("Inventory").WebOptions.UseDefaultFolderSuffix
About Property

Returns or sets a **String** representing version and copyright information for an **ActiveX control**. Read/write.

**expression**: *About*

**expression**  Required. An expression that returns one of the objects in the Applies To list.
Remarks

To view an About box showing version and copyright information for an ActiveX control, click the About property box in the Microsoft Access property sheet. Then click the Build button to the right of the property box.

Note The About box is not available for ActiveX controls on a data access page.
AccessConnection Property

You can use the **AccessConnection** property to return a reference to the current Microsoft ActiveX Data Objects (ADO) **Connection** object and its related properties. Read-only **Connection**.

*expression*.AccessConnection

*expression* Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **AccessConnection** property returns an ADO connection that uses two [OLE DB providers](https://docs.microsoft.com/en-us/dotnet/framework/data/adodb/ole-db-service-providers). It uses the Microsoft OLE DB Provider for Microsoft Jet (Microsoft.Jet.OLEDB.4.0) as the data provider and the Microsoft OLE DB Provider for Microsoft Access 10 (Microsoft.Access.OLEDB.10.0) as the service provider.

You should use the **AccessConnection** property if you intend to create ADO recordsets that will be bound to Access forms. The form will not be updateable unless it is created by using the OLE DB Provider for Microsoft Access 10, even if the recordset is updateable in ADO.
Action Property

You can use the Action property in Visual Basic to specify the operation to perform on an OLE object. Read/write Integer.

expression.Action

expression Required. An expression that returns one of the objects in the Applies To list.
Setting

The **Action** property uses the following settings.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>acOLECreateEmbed</td>
<td>Creates an embedded object. To use this setting, you must first set the control's <strong>OLETypeAllowed</strong> property to acOLEEmbedded or acOLEEither. Set the <strong>Class</strong> property to the type of OLE object you want to create. You can use the <strong>SourceDoc</strong> property to use an existing file as a template.</td>
</tr>
<tr>
<td>acOLECreateLink</td>
<td>Creates a linked OLE object from the contents of a file. To use this setting, you must first set the control's <strong>OLETypeAllowed</strong> and <strong>SourceDoc</strong> properties. Set the <strong>OLETypeAllowed</strong> property to acOLELinked or acOLEEither. The <strong>SourceDoc</strong> property specifies the file used to create the OLE object. You can also set the control's <strong>SourceItem</strong> property (for example, to specify a row-and-column range if the object you're creating is a Microsoft Excel worksheet). When you create an OLE object by using this setting, the control displays a metafile graphic image of the file specified by the control's <strong>SourceDoc</strong> property. If you save the OLE object, only the link information, such as the name of the application that supplied the object and the name of the linked file, is saved because the control contains an image of the data but no source data.</td>
</tr>
<tr>
<td>acOLECopy</td>
<td>Copies the object to the Clipboard. When you copy an OLE object to the Clipboard, all the data and link information associated with the object is placed on the Clipboard as well. You can copy both linked and embedded objects onto the Clipboard. Using this setting is equivalent to clicking <strong>Copy</strong> on the <strong>Edit</strong> menu.</td>
</tr>
</tbody>
</table>

Pastes data from the Clipboard to the control. If the paste operation is successful, the control's **OLEType**
The property is set to **acOLELinked** or **acOLEEmbedded**. If the paste operation isn't successful, the OLEType property is set to **acOLENone**. Using the **acOLEPaste** setting is equivalent to clicking **Paste** on the **Edit** menu.

**acOLEUpdate (6)**

Retrieves the current data from the application that supplied the object and displays that data as a metafile graphic in the control.

**acOLEActivate (7)**

Opens an OLE object for an operation, such as editing. To use this setting, you must first set the control's **Verb** property. The **Verb** property specifies the operation to perform when the OLE object is activated.

**acOLEClose (9)**

Closes an OLE object and ends the connection with the application that supplied the object. This setting applies to embedded objects only. Using this setting is equivalent to clicking **Close** on the object's **Control** menu.

**acOLEDelete (10)**

Deletes the specified OLE object and frees the associated memory. This setting enables you to explicitly delete an OLE object. Objects are automatically deleted when a form is closed or when the object is updated to a new object. You can't use the **Action** property to delete a bound OLE object from its underlying table or query.

**acOLEInsertObjDlg (14)**

Displays the **Insert Object** dialog box. In **Form view** or **Datasheet view**, you display this dialog box to enable the user to create a new object or to link or embed an existing object. You can use the control's **OLETypeAllowed** property to determine the type of object the user can create (with the constant **acOLELinked**, **acOLEEmbedded**, or **acOLEEither**) by using this dialog box.

**acOLEPasteSpecialDlg (15)**

Displays the **Paste Special** dialog box. In Form view or Datasheet view, you display this dialog box to enable the user to paste an object from the Clipboard. The dialog box provides several options, including pasting either a linked or embedded object. You can use the control's **OLETypeAllowed** property to
determine the type of object that can be pasted (with the constant acOLELinked, acOLEEmbedded, or acOLEEither) by using this dialog box.

Updates the list of verbs an OLE object supports. To display the list of verbs, use the ObjectVerbs and ObjectVerbsCount properties.

You can set the Action property only by using Visual Basic. The Action property setting is an Integer data type value.

The Action property isn't available in Design view but can be read or set in other views.
Remarks

When a control's **Enabled** property is set to No or its **Locked** property is set to Yes, you can't use some **Action** property settings. The following table indicates which settings are allowed or not allowed under these conditions.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Enabled = No</th>
<th>Locked = Yes</th>
</tr>
</thead>
<tbody>
<tr>
<td>acOLECreateEmbed (0)</td>
<td>Not allowed</td>
<td>Not allowed</td>
</tr>
<tr>
<td>acOLECreateLink (1)</td>
<td>Not allowed</td>
<td>Not allowed</td>
</tr>
<tr>
<td>acOLECopy (4)</td>
<td>Allowed</td>
<td>Allowed</td>
</tr>
<tr>
<td>acOLEPaste (5)</td>
<td>Not allowed</td>
<td>Not allowed</td>
</tr>
<tr>
<td>acOLEUpdate (6)</td>
<td>Not allowed</td>
<td>Not allowed</td>
</tr>
<tr>
<td>acOLEActivate (7)</td>
<td>Allowed</td>
<td>Allowed</td>
</tr>
<tr>
<td>acOLEClose (9)</td>
<td>Not allowed</td>
<td>Allowed</td>
</tr>
<tr>
<td>acOLEDelete (10)</td>
<td>Not allowed</td>
<td>Not allowed</td>
</tr>
<tr>
<td>acOLEInsertObjDlg (14)</td>
<td>Not allowed</td>
<td>Not allowed</td>
</tr>
<tr>
<td>acOLEPasteSpecialDlg (15)</td>
<td>Not allowed</td>
<td>Not allowed</td>
</tr>
<tr>
<td>acOLEFetchVerbs (17)</td>
<td>Not allowed</td>
<td>Allowed</td>
</tr>
</tbody>
</table>
ActiveControl Property

You can use the ActiveControl property together with the Screen object to identify or refer to the control that has the focus. Read-only Control object.

expression.ActiveControl

expression  Required. An expression that returns one of the objects in the Applies To list.
Setting

This property setting contains a reference to the Control object that has the focus at run time.

This property is available by using a macro or Visual Basic and is read-only in all views.
Remarks

You can use the `ActiveControl` property to refer to the control that has the focus at run time together with one of its properties or methods. The following example assigns the name of the control with the focus to the `strControlName` variable.

```vba
Dim ctlCurrentControl As Control
Dim strControlName As String
Set ctlCurrentControl = Screen.ActiveControl
strControlName = ctlCurrentControl.Name
```

If no control has the focus when you use the `ActiveControl` property, or if all of the active form's controls are hidden or disabled, an error occurs.
Example

The following example assigns the active control to the `ctlCurrentControl` variable and then takes different actions depending on the value of the control's `Name` property.

```vba
dim ctlcurrentcontrol as control
set ctlcurrentcontrol = screen.activexcontrol
if ctlcurrentcontrol.name = "txtCustomerID" then
    ' Do something here.
else if ctlcurrentcontrol.name = "btnCustomerDetails" then
    ' Do something here.
end if
```
ActiveDataAccessPage Property

You can use the ActiveDataAccessPage property to identify or refer to the data access page that has the focus. Read-only DataAccessPage object.

expression.ActiveDataAccessPage

expression Required. An expression that returns one of the objects in the Applies To list.
Setting

The **ActiveDataAccessPage** property setting contains a reference to the **DataAccessPage** object that has the focus at **run time**.

This property is available by using a **macro** or **Visual Basic** and is read-only.
Remarks

Use the **ActiveDataAccessPage** property to refer to an active data access page together with one of its properties or methods. The following example displays the **Name** property setting of the active data access page:

```vbnet
Dim pgCurrentPage As DataAccessPage
Set pgCurrentPage = Screen.ActiveDataAccessPage
MsgBox "Current Data Page is " & pgCurrentPage.Name
```

If no data access page has the focus when you use the **ActiveDataAccessPage** property, an error occurs.
ActiveDatasheet Property

You can use the *ActiveDatasheet* property together with the *Screen* object to identify or refer to the *datasheet* that has the *focus*. Read-only *Form* object.

`expression.ActiveDatasheet`

`expression` Required. An expression that returns one of the objects in the Applies To list.
Setting

The **ActiveDatasheet** property setting contains the datasheet object that has the focus at **run time**.

This property is available by using a [macro](https://example.com) or [Visual Basic](https://example.com) and is read-only in all views.
Remarks

You can use this property to refer to an active datasheet together with one of its properties or methods. For example, the following code uses the `ActiveDatasheet` property to reference the top row of the selection in the active datasheet.

```plaintext
TopRow = Screen.ActiveDatasheet.SelTop
```
Example

The following example uses the **ActiveDatasheet** property to identify the datasheet cell with the focus, or if more than one cell is selected, the location of the first row and column in the selection.

```vba
Public Sub GetSelection()
    ' This procedure demonstrates how to get a pointer to the current active datasheet.
    Dim objDatasheet As Object
    Dim lngFirstRow As Long
    Dim lngFirstColumn As Long
    Const conNoActiveDatasheet = 2484

    On Error GoTo GetSelection_Err

    Set objDatasheet = Screen.ActiveDatasheet

    lngFirstRow = objDatasheet.SelTop
    lngFirstColumn = objDatasheet.SelLeft
    MsgBox "The first item in this selection is located at " & _
            "Row " & lngFirstRow & ", Column " & _
            lngFirstColumn, vbInformation

GetSelection_Bye:
    Exit Sub
GetSelection_Err:
    If Err = conNoActiveDatasheet Then
        MsgBox "No data sheet is active.", vbExclamation
        Resume GetSelection_Bye
    End If
End Sub
```
ActiveForm Property

You can use the **ActiveForm** property together with the **Screen** object to identify or refer to the form that has the **focus**. Read-only **Form** object.

```vba
expression.ActiveForm
```

*expression* Required. An expression that returns one of the objects in the Applies To list.
Setting

This property setting contains a reference to the **Form** object that has the focus at run time.

This property is available by using a [macro](https://example.com) or [Visual Basic](https://example.com) and is read-only in all views.
Remarks

You can use the `ActiveForm` property to refer to an active form together with one of its properties or methods. The following example displays the `Name` property setting of the active form.

```vba
Dim frmCurrentForm As Form
Set frmCurrentForm = Screen.ActiveForm
MsgBox "Current form is " & frmCurrentForm.Name
```

If a `subform` has the focus, `ActiveForm` refers to the `main form`. If no form or subform has the focus when you use the `ActiveForm` property, an error occurs.
Show All
ActiveReport Property

You can use the ActiveReport property together with the Screen object to identify or refer to the report that has the focus. Read-only Report object.

expression.ActiveReport

expression Required. An expression that returns one of the objects in the Applies To list.
Setting

This property setting contains a reference to the **Report** object that has the focus at **run time**.

This property is available only by using a **macro** or **Visual Basic** and is read-only in all views.
Remarks

You can use the **ActiveReport** property to refer to an active report together with one of its properties or methods. The following example displays the **Name** property setting of the active report.

```vbnet
Dim rptCurrentReport As Report
Set rptCurrentReport = Screen.ActiveReport
MsgBox "Current report is " & rptCurrentReport.Name

If no report has the focus when you use the **ActiveReport** property, an error occurs.
```
AddColon Property

The `AddColon` property specifies whether a colon follows the text in `labels` for new `controls`. Read/write `Boolean`.

`expression.AddColon`

`expression` Required. An expression that returns one of the objects in the Applies To list.
The **AddColon** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>A colon follows the text in labels for new controls.</td>
</tr>
<tr>
<td>No</td>
<td>A colon doesn't follow the text in labels for new controls.</td>
</tr>
</tbody>
</table>

You can set these properties only by using a control's [default control style](#) or the [DefaultControl](#) method in Visual Basic.
Remarks

Changes to the default control style setting affect only controls created on the current form or report. To change the default control style for all new forms or reports that you create without using a Microsoft Access wizard, see Specify a new template for forms and reports.
Address Property

You can use the Address property to specify or determine the path to an object, document, Web page or other destination for a Hyperlink object associated with a command button, image control, or label control. Read/write String.

expression.Address

expression Required. An expression that returns one of the objects in the Applies To list.
Setting

The **Address** property is a *string expression* representing the **HyperlinkAddress** property's path to a file (UNC path) or Web page (URL).

The **Address** property is only available by using **Visual Basic**. When you set the **Address** property, you automatically specify the **HyperlinkAddress** property for the control associated with the hyperlink.
Remarks

For more information about hyperlink addresses and their format, see the HyperlinkAddress and HyperlinkSubAddress property topics.
**AfterDelConfirm Property**

Returns or sets a **String** indicating which macro, event procedure, or user-defined function runs when the **AfterDelConfirm** event occurs. Read/write.

`expression.AfterDelConfirm`

- **expression** Required. An expression that returns one of the objects in the Applies To list.
Remarks

Valid values for this property are "macroname" where macroname is the name of macro, "[Event Procedure]" which indicates the event procedure associated with the AfterDelConfirm event for the specified object, or "=functionname()" where functionname is the name of a user-defined function. For a more detailed discussion of event properties, see "Event Properties."
Example

The following example specifies that when the AfterDelConfirm event occurs on the first form of the current project, the associated event procedure should run.

`Forms(0).After DelConfirm = "[Event Procedure]"`
AfterFinalRender Property

Returns or sets a String indicating which macro, event procedure, or user-defined function runs when the AfterFinalRender event occurs. Read/write.

expression.AfterFinalRender

expression Required. An expression that returns one of the objects in the Applies To list.
Remarks

Valid values for this property are "macroname" where macroname is the name of macro, "[Event Procedure]" which indicates the event procedure associated with the AfterFinalRender event for the specified object, or "=functionname()" where functionname is the name of a user-defined function. For a more detailed discussion of event properties, see "Event Properties."
Example

The following example specifies that when the AfterFinalRender event occurs on the first form of the current project, the associated event procedure should run.

Forms(0).AfterFinalRender = "[Event Procedure]"
AfterInsert Property

Returns or sets a String indicating which macro, event procedure, or user-defined function runs when the AfterInsert event occurs. Read/write.

expression.AfterInsert

expression  Required. An expression that returns one of the objects in the Applies To list.
Remarks

Valid values for this property are "macroname" where macroname is the name of macro, "[Event Procedure]" which indicates the event procedure associated with the AfterInsert event for the specified object, or "=functionname()" where functionname is the name of a user-defined function. For a more detailed discussion of event properties, see "Event Properties."
Example

The following example specifies that when the AfterInsert event occurs on the first form of the current project, the associated event procedure should run.

\texttt{Forms(0).After Insert = "[Event Procedure]"}
AfterLayout Property

Returns or sets a String indicating which macro, event procedure, or user-defined function runs when the AfterLayout event occurs. Read/write.

expression.AfterLayout

expression Required. An expression that returns one of the objects in the Applies To list.
Remarks

Valid values for this property are "macroname" where macroname is the name of macro, "[Event Procedure]" which indicates the event procedure associated with the AfterLayout event for the specified object, or "=functionname()" where functionname is the name of a user-defined function. For a more detailed discussion of event properties, see "Event Properties."
Example

The following example specifies that when the AfterLayout event occurs on the first form of the current project, the associated event procedure should run.

`Forms(0).AfterLayout = "[Event Procedure]"`
AfterRender Property

Returns or sets a String indicating which macro, event procedure, or user-defined function runs when the AfterRender event occurs. Read/write.

expression.AfterRender

expression Required. An expression that returns one of the objects in the Applies To list.
Remarks

Valid values for this property are "macroname" where macroname is the name of macro, "[Event Procedure]" which indicates the event procedure associated with the AfterRender event for the specified object, or "=functionname()" where functionname is the name of a user-defined function. For a more detailed discussion of event properties, see "Event Properties."
Example

The following example specifies that when the AfterRender event occurs on the first form of the current project, the associated event procedure should run.

`Forms(0).AfterRender = "[Event Procedure]"`
AfterUpdate Property

Returns or sets a **String** indicating which macro, event procedure, or user-defined function runs when the **AfterUpdate** event occurs. Read/write.

*expression*.AfterUpdate

*expression*  Required. An expression that returns one of the objects in the Applies To list.
Remarks

Valid values for this property are "macroname" where macroname is the name of macro, "[Event Procedure]" which indicates the event procedure associated with the AfterUpdate event for the specified object, or "=functionname()" where functionname is the name of a user-defined function. For a more detailed discussion of event properties, see "Event Properties."
Example

The following example specifies that when the AfterUpdate event occurs on the first form of the current project, the associated event procedure should run.

```
Forms(0).AfterUpdate = "[Event Procedure]"
```
**AllDataAccessPages Property**

You can use the `AllDataAccessPages` property to reference the `AllDataAccessPages` collection and its related properties. Read-only `AllDataAccessPages` object.

`expression.AllDataAccessPages`

`expression` Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **AllDataAccessPage** property is available only by using [Visual Basic](https://docs.microsoft.com/en-us/dotnet/vba/) and is read-only.
AllDatabaseDiagrams Property

You can use the `AllDatabaseDiagrams` property to reference the `AllDatabaseDiagrams` collection and its related properties. Read-only `AllDatabaseDiagrams` object.

`expression.AllDatabaseDiagrams`

`expression` Required. An expression that returns one of the objects in the Applies To list.
Remarks

The \textit{AllDatabaseDiagrams} property is available only by using \textit{Visual Basic} and is read-only.
AllForms Property

You can use the AllForms property to reference the AllForms collection and its related properties. Read-only AllForms object.

expression. AllForms

expression  Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **AllForms** property is available only by using **Visual Basic** and is read-only.
AllFunctions Property

Returns an AllFunctions collection representing all the user-defined functions in a Microsoft SQL Server database.

`expression.AllFunctions`

`expression`  Required. An expression that returns one of the objects in the Applies To list.
Remarks

You can use the AllFunctions property to reference the AllFunctions collection and its related properties.
Example

The following example prints the name of each open AccessObject object in the AllFunctions collection.

Dim objFunction As AccessObject

Debug.Print "Currently loaded functions:

For Each objFunction In Application.CurrentData.AllFunctions
    If objFunction.IsLoaded = msoTrue Then
        Debug.Print objFunction.Name
    End If
Next objFunction
AllMacros Property

You can use the **AllMacros** property to reference the **AllMacros** collection and its related properties. Read-only **AllMacros** object.

*expression*.AllMacros

*expression*  Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **AllMacros** property is available only by using **Visual Basic** and is read-only.
AllModules Property

You can use the **AllModules** property to reference the **AllModules** collection and its related properties. Read-only **AllModules** object.

*expression*.**AllModules*

*expression* Required. An expression that returns one of the objects in the Applies To list.
Remarks

The `AllModules` property is available only by using `Visual Basic` and is read-only.
AllowAdditions Property

You can use the AllowAdditions property to specify whether a user can add a record when using a form. Read/write Boolean.

expression.AllowAdditions

expression  Required. An expression that returns one of the objects in the Applies To list.
Setting

The **AllowAdditions** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>True</td>
<td>(Default) The user can add new records.</td>
</tr>
<tr>
<td>No</td>
<td>False</td>
<td>The user can't add new records.</td>
</tr>
</tbody>
</table>

You can set the **AllowAdditions** property by using the form's [property sheet](#), a [macro](#), or [Visual Basic](#).
Remarks

Set the AllowAdditions property to No to allow users to view or edit existing records but not add new records.

If you want to prevent changes to existing records (make a form read-only), set the AllowAdditions, AllowDeletions, and AllowEdits properties to No. You can also make records read-only by setting the RecordsetType property to Snapshot.

If you want to open a form for data entry only, set the form's DataEntry property to Yes.

When the AllowAdditions property is set to No, the Data Entry command on the Records menu isn't available.

Note  When the Data Mode argument of the OpenForm action is used, Microsoft Access will override a number of form property settings. If the Data Mode argument of the OpenForm action is set to Edit, Microsoft Access will open the form with the following property settings:

- AllowEdits— Yes
- AllowDeletions— Yes
- AllowAdditions— Yes
- DataEntry— No

To prevent the OpenForm action from overriding any of these existing property settings, omit the Data Mode argument setting so that Microsoft Access will use the property settings defined by the form.
Example

The following example examines the **ControlType** property for all controls on a form. For each label and text box control, the procedure toggles the **SpecialEffect** property for those controls. When the label controls' **SpecialEffect** property is set to Shadowed and the text box controls' **SpecialEffect** property is set to Normal and the **AllowAdditions**, **AllowDeletions**, and **AllowEdits** properties are all set to **True**, the **intCanEdit** variable is toggled to allow editing of the underlying data.

```vba
Sub ToggleControl(frm As Form)
    Dim ctl As Control
    Dim intI As Integer, intCanEdit As Integer
    Const conTransparent = 0
    Const conWhite = 16777215
    For Each ctl in frm.Controls
        With ctl
            Select Case .ControlType
                Case acLabel
                    If .SpecialEffect = acEffectShadow Then
                        .SpecialEffect = acEffectNormal
                        .BorderStyle = conTransparent
                        intCanEdit = True
                    Else
                        .SpecialEffect = acEffectShadow
                        intCanEdit = False
                    End If
                Case acTextBox
                    If .SpecialEffect = acEffectNormal Then
                        .SpecialEffect = acEffectSunken
                        .BackColor = conWhite
                    Else
                        .SpecialEffect = acEffectNormal
                    End If
            End Select
        End With
    Next ctl
    If intCanEdit = False Then
        With frm
            .AllowAdditions = False
            .AllowDeletions = False
            .AllowEdits = False
        End With
    Else
```
With frm
    .AllowAdditions = True
    .AllowDeletions = True
    .AllowEdits = True
End With
End If
End Sub
AllowAutoCorrect Property

You can use the `AllowAutoCorrect` property to specify whether a `textbox` or a `combobox` control will automatically correct entries made by the user. Read/write `Boolean`.

`expression.AllowAutoCorrect`

`expression` Required. An expression that returns one of the objects in the Applies To list.
Setting

The `AllowAutoCorrect` property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>True</td>
<td>(Default) Entries are automatically corrected according to the settings in the <code>AutoCorrect</code> dialog box.</td>
</tr>
<tr>
<td>No</td>
<td>False</td>
<td>Entries aren't corrected.</td>
</tr>
</tbody>
</table>

You can set the `AllowAutoCorrect` property by using a control's property sheet, a macro, or Visual Basic. You can set the default for this property by using a control's default control style or the `DefaultControl` method in Visual Basic.
AllowDatasheetView Property

Returns or sets a **Boolean** indicating whether the specified form may be viewed in Datasheet View. **True** if Datasheet View is allowed. Read/write.

`expression.AllowDatasheetView`

`expression` Required. An expression that returns one of the objects in the Applies To list.
Remarks

Use the AllowDatasheetView, AllowFormView, AllowPivotChartView, or AllowPivotTableView properties to control which views are allowed for a form.
**Example**

The following example makes Datasheet View valid for the specified form and then opens the form in Datasheet View.

```vba
Forms(0).AllowDatasheetView = True
DoCmd.OpenForm FormName:=Forms(0).Name, View:=acFormDS
```
AllowDeletions Property

You can use the AllowDeletions property to specify whether a user can delete a record when using a form. Read/write Boolean.

expression.AllowDeletions

expression Required. An expression that returns one of the objects in the Applies To list.
Setting

The AllowDeletions property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>True</td>
<td>(Default) The user can delete records.</td>
</tr>
<tr>
<td>No</td>
<td>False</td>
<td>The user can't delete records.</td>
</tr>
</tbody>
</table>

You can set the AllowDeletions property by using the form's property sheet, a macro, or Visual Basic.
Remarks

You can set this property to No to allow users to view and edit existing records but not to delete them. When AllowDeletions is set to Yes, records may be deleted so long as existing referential integrity rules aren't broken.

If you want to prevent changes to existing records (make a form read-only), set the AllowAdditions, AllowDeletions, and AllowEdits properties to No. You can also make records read-only by setting the RecordsetType property to Snapshot.

When the AllowDeletions property is set to No, the Delete Record command on the Edit menu isn't available.

**Note** When the Data Mode argument of the OpenForm action is set, Microsoft Access will override a number of form property settings. If the Data Mode argument of the OpenForm action is set to Edit, Microsoft Access will open the form with the following property settings:

- **AllowEdits**— Yes
- **AllowDeletions**— Yes
- **AllowAdditions**— Yes
- **DataEntry**— No

To prevent the OpenForm action from overriding any of these existing property settings, omit the Data Mode argument setting so that Microsoft Access will use the property settings defined by the form.
Example

The following example examines the **ControlType** property for all controls on a form. For each label and text box control, the procedure toggles the **SpecialEffect** property for those controls. When the label controls' **SpecialEffect** property is set to Shadowed and the text box controls' **SpecialEffect** property is set to Normal and the **AllowAdditions**, **AllowDeletions**, and **AllowEdits** properties are all set to **True**, the intCanEdit variable is toggled to allow editing of the underlying data.

Sub ToggleControl(frm As Form)
    Dim ctl As Control
    Dim intI As Integer, intCanEdit As Integer
    Const conTransparent = 0
    Const conWhite = 16777215
    For Each ctl in frm.Controls
        With ctl
            Select Case .ControlType
                Case acLabel
                    If .SpecialEffect = acEffectShadow Then
                        .SpecialEffect = acEffectNormal
                        .BorderStyle = conTransparent
                        intCanEdit = True
                    Else
                        .SpecialEffect = acEffectShadow
                        intCanEdit = False
                    End If
                Case acTextBox
                    If .SpecialEffect = acEffectNormal Then
                        .SpecialEffect = acEffectSunken
                        .BackColor = conWhite
                    Else
                        .SpecialEffect = acEffectNormal
                    End If
            End Select
        End With
    Next ctl
    If intCanEdit = False Then
        With frm
            .AllowAdditions = False
            .AllowDeletions = False
            .AllowEdits = False
        End With
    Else
        ' Do nothing
    End If
End Sub
With frm
    .AllowAdditions = True
    .AllowDeletions = True
    .AllowEdits = True
End With
End If
End Sub
AllowDesignChanges Property

You can use the AllowDesignChanges property to specify or determine if design changes can be made to a form in all views or Design view only. Read/write Boolean.

expression.AllowDesignChanges

type of expression. Required. An expression that returns one of the objects in the Applies To list.
Setting

The **AllowDesignChanges** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>All Views</td>
<td><strong>True</strong></td>
<td>(Default) Design changes can be made in all form views.</td>
</tr>
<tr>
<td>Design View Only</td>
<td><strong>False</strong></td>
<td>Design changes can be made in Design view only.</td>
</tr>
</tbody>
</table>

The **AllowDesignChanges** property can be set by using the form's property sheet or Visual Basic.
AllowEdits Property

You can use the **AllowEdits** property to specify whether a user can edit saved records when using a form. Read/write **Boolean**.

*expression*.AllowEdits

*expression* Required. An expression that returns one of the objects in the Applies To list.
Setting

The **AllowEdits** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td><strong>True</strong></td>
<td>(Default) The user can edit saved records.</td>
</tr>
<tr>
<td>No</td>
<td><strong>False</strong></td>
<td>The user can't edit saved records.</td>
</tr>
</tbody>
</table>

You can set the **AllowEdits** property by using the form's property sheet, a macro, or Visual Basic.
Remarks

You can use the **AllowEdits** property to prevent changes to existing data displayed by a form. If you want to prevent changes to data in a specific control, use the **Enabled** or **Locked** property.

If you want to prevent changes to existing records (make a form read-only), set the **AllowAdditions**, **AllowDeletions**, and **AllowEdits** properties to No. You can also make records read-only by setting the **RecordsetType** property to Snapshot.

When the **AllowEdits** property is set to No, the **Delete Record** and **Data Entry** menu commands aren't available for existing records. (They may still be available for new records if the **AllowAdditions** property is set to Yes.)

Changing a field value programmatically causes the current record to be editable, regardless of the **AllowEdits** property setting. If you want to prevent the user from making changes to a record (**AllowEdits** is No) that you need to edit programmatically, save the record after any programmatic changes; the **AllowEdits** property setting will be honored once again after any unsaved changes to the current record are saved.

**Note**  When the Data Mode argument of the **OpenForm** action is set, Microsoft Access will override a number of form property settings. If the Data Mode argument of the OpenForm action is set to Edit, Microsoft Access will open the form with the following property settings:

- **AllowEdits**— Yes
- **AllowDeletions**— Yes
- **AllowAdditions**— Yes
- **DataEntry**— No

To prevent the OpenForm action from overriding any of these existing property settings, omit the Data Mode argument setting so that Microsoft Access will use the property settings defined by the form.
Example

The following example examines the **ControlType** property for all controls on a form. For each label and text box control, the procedure toggles the **SpecialEffect** property for those controls. When the label controls' **SpecialEffect** property is set to Shadowed and the text box controls' **SpecialEffect** property is set to Normal and the **AllowAdditions**, **AllowDeletions**, and **AllowEdits** properties are all set to **True**, the intCanEdit variable is toggled to allow editing of the underlying data.

```vba
Sub ToggleControl(frm As Form)
    Dim ctl As Control
    Dim intI As Integer, intCanEdit As Integer
    Const conTransparent = 0
    Const conWhite = 16777215
    For Each ctl in frm.Controls
        With ctl
            Select Case .ControlType
            Case acLabel
                If .SpecialEffect = acEffectShadow Then
                    .SpecialEffect = acEffectNormal
                    .BorderStyle = conTransparent
                    intCanEdit = True
                Else
                    .SpecialEffect = acEffectShadow
                    intCanEdit = False
                End If
            Case acTextBox
                If .SpecialEffect = acEffectNormal Then
                    .SpecialEffect = acEffectSunken
                    .BackColor = conWhite
                Else
                    .SpecialEffect = acEffectNormal
                End If
            End Select
        End With
    Next ctl
    If intCanEdit = False Then
        With frm
            .AllowAdditions = False
            .AllowDeletions = False
            .AllowEdits = False
        End With
    Else
```

---

**Example**

The following example examines the **ControlType** property for all controls on a form. For each label and text box control, the procedure toggles the **SpecialEffect** property for those controls. When the label controls' **SpecialEffect** property is set to Shadowed and the text box controls' **SpecialEffect** property is set to Normal and the **AllowAdditions**, **AllowDeletions**, and **AllowEdits** properties are all set to **True**, the intCanEdit variable is toggled to allow editing of the underlying data.

```vba
Sub ToggleControl(frm As Form)
    Dim ctl As Control
    Dim intI As Integer, intCanEdit As Integer
    Const conTransparent = 0
    Const conWhite = 16777215
    For Each ctl in frm.Controls
        With ctl
            Select Case .ControlType
            Case acLabel
                If .SpecialEffect = acEffectShadow Then
                    .SpecialEffect = acEffectNormal
                    .BorderStyle = conTransparent
                    intCanEdit = True
                Else
                    .SpecialEffect = acEffectShadow
                    intCanEdit = False
                End If
            Case acTextBox
                If .SpecialEffect = acEffectNormal Then
                    .SpecialEffect = acEffectSunken
                    .BackColor = conWhite
                Else
                    .SpecialEffect = acEffectNormal
                End If
            End Select
        End With
    Next ctl
    If intCanEdit = False Then
        With frm
            .AllowAdditions = False
            .AllowDeletions = False
            .AllowEdits = False
        End With
    Else
```
With frm
    .AllowAdditions = True
    .AllowDeletions = True
    .AllowEdits = True
End With
End If
End Sub
AllowFilters Property

You can use the AllowFilters property to specify whether records in a form can be filtered. Read/write Boolean.

expression.AllowFilters

expression Required. An expression that returns one of the objects in the Applies To list.
The **AllowFilters** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td><strong>True</strong></td>
<td>(Default) Records can be filtered.</td>
</tr>
<tr>
<td>No</td>
<td><strong>False</strong></td>
<td>Records can't be filtered.</td>
</tr>
</tbody>
</table>

You can set this property by using the form's [property sheet](#), a [macro](#), or Visual Basic.
Remarks

Filters are commonly used to view a temporary subset of the records in a database. When you use a filter, you apply criteria to display only records that meet specific conditions. In an Employees form, for example, you can use a filter to display only records of employees with over 5 years of service. You can also use a filter to restrict access to records containing sensitive information, such as financial or medical data.

Note Setting the AllowFilters property to No does not affect the Filter, FilterOn, ServerFilter, or ServerFilterByForm properties. You can still use these properties to set and remove filters. You can also still use the following actions or methods to apply and remove filters.

<table>
<thead>
<tr>
<th>Actions</th>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>ApplyFilter</td>
<td>ApplyFilter</td>
</tr>
<tr>
<td>OpenForm</td>
<td>OpenForm</td>
</tr>
<tr>
<td>ShowAllRecords</td>
<td>ShowAllRecords</td>
</tr>
</tbody>
</table>
AllowFormView Property

Returns or sets a **Boolean** indicating whether the specified form may be viewed in Form View. **True** if Form View is allowed. Read/write.

*expression*.AllowFormView

*expression*  Required. An expression that returns one of the objects in the Applies To list.
Remarks

Use the AllowDatasheetView, AllowFormView, AllowPivotChartView, or AllowPivotTableView properties to control which views are allowed for a form.
Example

The following example makes Form View valid for the specified form and then opens the form in Form View.

Forms(0).AllowFormView = True
DoCmd.OpenForm FormName:=Forms(0).Name, View:=acNormal
AllowPivotChartView Property

Returns or sets a **Boolean** indicating whether the specified form may be viewed in PivotChart View. **True** if PivotChart View is allowed. Read/write.

*expression*.AllowPivotChartView

*expression*  Required. An expression that returns one of the objects in the Applies To list.
Remarks

Use the AllowDatasheetView, AllowFormView, AllowPivotChartView, or AllowPivotTableView properties to control which views are allowed for a form.
Example

The following example makes PivotChart View valid for the specified form and then opens the form in PivotChart View.

`Forms(0).AllowPivotChartView = True
DoCmd.OpenForm FormName:=Forms(0).Name, View:=acFormPivotChart`
AllowPivotTableView Property

Returns or sets a **Boolean** indicating whether the specified form may be viewed in PivotTable View. **True** if PivotTable View is allowed. Read/write.

*expression*.AllowPivotTableView

*expression* Required. An expression that returns one of the objects in the Applies To list.
Remarks

Use the AllowDatasheetView, AllowFormView, AllowPivotChartView, or AllowPivotTableView properties to control which views are allowed for a form.
Example

The following example makes PivotTable View valid for the specified form and then opens the form in PivotTable View.

Forms(0).AllowPivotTableView = True
DoCmd.OpenForm FormName:=Forms(0).Name, View:=acFormPivotTable
AllQueries Property

You can use the `AllQueries` property to reference the `AllQueries` collection and its related properties. Read-only `AllQueries` object.

*expression*.`AllQueries`

*expression*   Required. An expression that returns one of the objects in the Applies To list.
Remarks

The AllQueries property is available only by using Visual Basic and is read-only.
AllReports Property

You can use the AllReports property to reference the AllReports collection and its related properties. Read-only AllReports object.

expression.AllReports

expression Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **AllReports** property is available only by using [Visual Basic](https://msdn.microsoft.com/en-us/library/az9ytw8w.aspx) and is read-only.
AllStoredProcedures Property

You can use the AllStoredProcedures property to reference the AllStoredProcedures collection and its related properties. Read-only AllStoredProcedures object.

expression.AllStoredProcedures

expression Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **AllStoredProcedures** property is available only by using **Visual Basic** and is read-only.
AllTables Property

You can use the AllTables property to reference the AllTables collection and its related properties. Read-only AllTables object.

expression.AllTables

expression Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **AllTables** property is available only by using [Visual Basic](https://docs.microsoft.com/en-us/visualstudio/vs/2019/visual-basic/overview) and is read-only.
AllViews Property

You can use the AllViews property to reference the AllViews collection and its related properties. Read-only AllViews object.

expression.AllViews

description Required. An expression that returns one of the objects in the Applies To list.
Remarks

The AllViews property is available only by using Visual Basic and is read-only.
AlwaysSaveInDefaultEncoding Property

You can use the **AlwaysSaveInDefaultEncoding** property to specify or determine whether the Web browser opens a data access page with its default or original encoding (character set). Read/write **Boolean**.

*expression*.**AlwaysSaveInDefaultEncoding**

*expression* Required. An expression that returns one of the objects in the Applies To list.
**Remarks**

The `AlwaysSaveInDefaultEncoding` property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>True</td>
<td>Use the default encoding specified by the <code>Encoding</code> when saving a data access page.</td>
</tr>
<tr>
<td>No</td>
<td>False</td>
<td>(Default) Use the original encoding of the data access page.</td>
</tr>
</tbody>
</table>

**Note** The `AlwaysSaveInDefaultEncoding` property is available only by using `Visual Basic`.

The `Encoding` property can be used to set the default encoding.
Example

This example sets the encoding to the default encoding. The encoding is used when you save the data access page as a Web page.

Application.DefaultWebOptions.AlwaysSaveInDefaultEncoding = True
AnswerWizard Property

You can use the AnswerWizard property to return a reference to the current AnswerWizard object and its related properties. Read-only AnswerWizard object.

expression.AnswerWizard

expression Required. An expression that returns one of the objects in the Applies To list.

Remarks

The AnswerWizard property is available only by using Visual Basic.

Once you establish a reference to the AnswerWizard object, you can access all the properties and methods of the object. You can set a reference to the AnswerWizard object by clicking References on the Tools menu while in module Design view. Then set a reference to the Microsoft Office 11.0 Object Library in the References dialog box by selecting the appropriate check box. Microsoft Access can set this reference for you if you use a Microsoft Office 11.0 Object Library constant to set an AnswerWizard object's property or as an argument to an AnswerWizard object's method.
Application Property

You can use the Application property in Visual Basic to access the active Microsoft Access Application object and its related properties. Read-only Application object.

expression.Application

expression Required. An expression that returns one of the objects in the Applies To list.
Setting

The **Application** property is set by Microsoft Access and is read-only in all views.
Remarks

Each Microsoft Access object has an Application property that returns the current Application object. You can use this property to access any of the object's properties. For example, you could refer to the menu bar for the Application object from the current form by using the following syntax:

Me.Application.MenuBar
Example

The following example demonstrates how to change the cursor to an hourglass and back again to signify that some background activity is occurring.

Application.Screen.MousePointer = 11 ' Hourglass
' Do some background activity.
Application.Screen.MousePointer = 0 ' Back to normal
AsianLineBreak Property

Returns or sets a Boolean indicating whether line breaks in text boxes follow rules governing East Asian languages. True to control line breaks based on East Asian language rules. Read/write.

expression.AsianLineBreak

expression  Required. An expression that returns one of the objects in the Applies To list.
Remarks

Setting the **AsianLineBreak** property to **True** moves any punctuation marks and closing parentheses at the beginning of a line to the end of the previous line, and moves opening parentheses at the end of a line to the beginning of the next line.
Example

This example sets all the text boxes on the specified form to break lines according to East Asian language rules.

Dim ctlLoop As Control

For Each ctlLoop In Forms(0).Controls
    If ctlLoop.ControlType = acTextBox Then
        ctlLoop.AsiAnLineBreak = True
    End If
Next ctlLoop
Assistant Property

You can use the Assistant property to return a reference to the Assistant object. Read-only Assistant object.

_expression_.Assistant

expression  Required. An expression that returns one of the objects in the Applies To list.
Example

The following example uses the Assistant property of the Application object to manipulate various properties and methods of the Assistant object.

Public Sub AnimateAssistant()
    Dim blnState As Boolean

    With Assistant
        ' Save Assistant's visible state.
        blnState = .Visible

        ' Make Assistant visible.
        If blnState = False Then .Visible = True

        ' Animate Assistant.
        .Animation = msoAnimationAppear

        ' Display Assistant object's Item and FileName properties.
        MsgBox "Hello, my name is " & .Item & ". I live in " & _
            .FileName, , "Assistant Information:"

        ' Return Assistant's visible state to original setting.
        .Visible = blnState
    End With
End Sub
AutoActivate Property

You can use the **AutoActivate** property to specify how the user can activate an **OLE object**. Read/write **Integer**.

*expression*.AutoActivate

*expression*  Required. An expression that returns one of the objects in the Applies To list.
Setting

The **AutoActivate** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manual</td>
<td>acOLEActivateManual (0)</td>
<td>The OLE object isn't activated when it receives the focus or when the user double-clicks the control. You can activate an OLE object only by using Visual Basic to set the control's Action property to acOLEActivate. For unbound object frame and chart controls only) If the control contains an OLE object, the application that supplied the object is activated when the control receives the focus. (Default) If the control contains an OLE object, the application that supplied the object is activated when the user double-clicks the control or presses CTRL+ENTER when the control has the focus.</td>
</tr>
<tr>
<td>GetFocus</td>
<td>acOLEActivateGetFocus (1)</td>
<td>You can set this property by using the control's property sheet, a macro, or Visual Basic.</td>
</tr>
<tr>
<td>Double-Click</td>
<td>acOLEActivateDoubleClick (2)</td>
<td>The AutoActivate property can be set only in Design view.</td>
</tr>
</tbody>
</table>
Remarks

Some OLE objects can be activated from within the control. When such an object is activated, the object can be edited (or some other operation can be performed) from inside the boundaries of the control. This feature is called **in-place activation**. If an object supports in-place activation, see the documentation for the application that was used to create the object for information about using this feature.

With Visual Basic, you can determine if a control contains an object by checking the setting of its **OLEType** property.

**Note**  If you set a control's **AutoActivate** property to Double-Click and specify a **DblClick** event for the control, the DblClick event occurs before the object is activated.
AutoCenter Property

Returns or sets a Boolean indicating whether a form will be centered automatically in the application window when the form is opened. Read/write.

expression.AutoCenter

expression   Required. An expression that returns one of the objects in the Applies To list.
# Setting

The **AutoCenter** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>True</td>
<td>The form will be centered automatically on opening.</td>
</tr>
<tr>
<td>No</td>
<td>False</td>
<td>(Default) The form's upper-left corner will be in the same location as when the form was last saved.</td>
</tr>
</tbody>
</table>

You can set this property by using the form's [property sheet](#), a [macro](#), or [Visual Basic](#).

You can set this property only in [Design view](#).
Remarks

Depending on the size and placement of the application window, forms can appear off to one side of the application window, hiding part of the form. Centering the form automatically when it's opened makes it easier to view and use.

If you make any changes in Design view to a form whose **AutoResize** property is set to No and whose **AutoCenter** property is set to Yes, switch to **Form view** before saving the form. If you don't, Microsoft Access clips the form on the right and bottom edges the next time you open the form.
AutoCorrect Property

Returns an AutoCorrect object that represents the AutoCorrect settings for the specified control. Read-only.

expression.AutoCorrect

expression  Required. An expression that returns one of the objects in the Applies To list.
**AutoExpand Property**

You can use the **AutoExpand** property to specify whether Microsoft Access automatically fills the text box portion of a combo box with a value from the combo box list that matches the characters you enter as you type in the combo box. This lets you quickly enter an existing value in a combo box without displaying the list box portion of the combo box. Read/write **Boolean**.

`expression.AutoExpand`

`expression` Required. An expression that returns one of the objects in the Applies To list.
Setting

The **AutoExpand** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>True</td>
<td>(Default) Microsoft Access fills in the combo box from the list with the first value matching the first character that you type. As you type additional characters, Microsoft Access changes the value displayed to match all the characters typed.</td>
</tr>
<tr>
<td>No</td>
<td>False</td>
<td>You must select a value from the list or type the entire value.</td>
</tr>
</tbody>
</table>

You can set the **AutoExpand** property by using the combo box's property sheet, a macro, or Visual Basic.
Remarks

When you enter characters in the text box portion of a combo box, Microsoft Access searches the values in the list to find those that match the characters you have typed. If the **AutoExpand** property is set to Yes, Microsoft Access automatically displays the first underlying value that matches the characters entered so far.

When the **LimitToList** property is set to Yes and the combo box list is dropped down, Microsoft Access selects matching values in the list as the user enters characters in the text box portion of the combo box, even if the **AutoExpand** property is set to No. If the user presses ENTER or moves to another control or record, the selected value appears in the combo box.
AutoLabel Property

The **AutoLabel** property specifies whether labels are automatically created and attached to new controls. Read/write **Boolean**.

`expression.AutoLabel`

*expression* Required. An expression that returns one of the objects in the Applies To list.

**Setting**

The **AutoLabel** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>A label is attached to new controls.</td>
</tr>
<tr>
<td>No</td>
<td>A label isn't attached to new controls.</td>
</tr>
</tbody>
</table>

You can set these properties only by using a control's default control style or the **DefaultControl** method in Visual Basic.

**Remarks**

Changes to the default control style setting affect only controls created on the current **form** or **report**. To change the default control style for all new forms or reports that you create without using a Microsoft Access wizard, see Specify a new template for forms and reports.
AutomationSecurity Property

Returns or sets an MsoAutomationSecurity constant that represents the security mode that Microsoft Access uses when it is programmatically opening files. Read/write.

expression.AutomationSecurity

expression Required. An expression that returns an Application object.

MsoAutomationSecurity can be one of these MsoAutomationSecurity constants:

msoAutomationSecurityByUI Uses the security setting specified in the Security dialog box (Tools menu, Macro submenu). msoAutomationSecurityByUI is the default value when the application is started.

msoAutomationSecurityForceDisable Access will not open any database that contains macros, Visual Basic for Applications code, a form that contains an ActiveX control, a report that contains an ActiveX control, update queries, data-definition queries, pass-through queries, or action queries. No security messages are shown to the user.

msoAutomationSecurityLow Enables all macros.
AutoRepeat Property

You can use the **AutoRepeat** property to specify whether an event procedure or macro runs repeatedly while a command button on a form remains pressed in. Read/write **Boolean**.

\[ expression.Autorepeat \]

*expression*  Required. An expression that returns one of the objects in the Applies To list.
Setting

The **AutoRepeat** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td><strong>True</strong></td>
<td>The macro or event procedure in the <strong>Click</strong> event runs repeatedly while the command button remains pressed in.</td>
</tr>
<tr>
<td>No</td>
<td><strong>False</strong></td>
<td>(Default) The macro or event procedure runs once.</td>
</tr>
</tbody>
</table>

You can set this property by using the command button's [property sheet](#), a [macro](#), or [Visual Basic](#).
Remarks

The initial repeat of the event procedure or macro occurs 0.5 seconds after it first runs. Subsequent repeats occur either 0.25 seconds apart or the duration of the event procedure or macro, whichever is longer.

If the code attached to the command button causes the current record to change, the **AutoRepeat** property has no effect.

If the code attached to the command button causes changes to another **control** on a form, use the **DoEvents** function to ensure proper screen updating.
AutoResize Property

Returns or sets a **Boolean** indicating whether a **Form window** opens automatically sized to display complete records. Read/write.

*expression*.AutoResize

*expression*   Required. An expression that returns one of the objects in the Applies To list.


Setting

The **AutoResize** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>True</td>
<td>(Default) The Form window is automatically sized to display a complete record. When opened, the Form window has the last saved size. To save a Form window's size, open the form, size the window, save the form by clicking <strong>Save</strong> on the <strong>File</strong> menu, and close the form. When you next open the form, it will be the saved window size.</td>
</tr>
<tr>
<td>No</td>
<td>False</td>
<td></td>
</tr>
</tbody>
</table>

You can set this property by using the form's [property sheet](#), a [macro](#), or [Visual Basic](#).

This property can be set only in [Design view](#).
Remarks

The Form window resizes only if opened in **Form view**. If you open the form first in **Design view** or **Datasheet view** and then change to Form view, the Form window won't resize.

If you make any changes in Design view to a form whose **AutoResize** property is set to No and whose **AutoCenter** property is set to Yes, switch to Form view before saving the form. If you don't, Microsoft Access clips the form on the right and bottom edges the next time you open the form.

If the **AutoCenter** property is set to No, a Form window opens with its upper-left corner in the same location as when it was closed.
AutoTab Property

You can use the **AutoTab** property to specify whether an automatic tab occurs when the last character permitted by a text box control's [input mask](#) is entered. An automatic tab moves the [focus](#) to the next [control](#) in the form's [tab order](#). Read/write **Boolean**.

**expression.AutoTab**

**expression**  Required. An expression that returns one of the objects in the Applies To list.
Setting

The **AutoTab** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>True</td>
<td>Generates a tab when the last allowable character in a text box is entered.</td>
</tr>
<tr>
<td>No</td>
<td>False</td>
<td>(Default) Doesn't generate a tab when the last allowable character in a text box is entered.</td>
</tr>
</tbody>
</table>

You can set this property by using a form's [property sheet](#), a [macro](#), or [Visual Basic](#).

You can set the default for this property by using the control's [default control style](#) or the [DefaultControl](#) method in Visual Basic.

The **AutoTab** property affects tab behavior in both [Form view](#) and [Datasheet view](#).
Remarks

You create an input mask for a control by using the `InputMask` property.

You can also create an input mask for a text box control `bound` to a field by setting the `InputMask` property for the field in the form's underlying table or query. If the field is dragged to a form from the field list, the field's input mask is inherited by the text box control.

You could use the `AutoTab` property if you have a text box on a form for which you usually enter the maximum number of characters for each record. After you have entered the maximum number of characters, focus automatically moves to the next control in the tab order. For example, you could use this property for a `CategoryType` field that must always be five characters long.
BackColor Property

You can use the **BackColor** property to specify the color for the interior of a control or section. Read/write **Long**.

expression.**BackColor**

*expression*  Required. An expression that returns one of the objects in the Applies To list.
Setting

The **BackColor** property contains a **numeric expression** that corresponds to the color used to fill a control's or section's interior.

You can use the Color Builder to set this property by clicking the **Build** button to the right of the property box in the **property sheet**. Using the Color Builder enables you to define custom back colors for controls or sections.

You can also set this property by using **Fill/Back Color** on the **Formatting (Form/Report)** toolbar, a control's or section's **property sheet**, a **macro**, or **Visual Basic**.

In Visual Basic, use a numeric expression to set this property. This property setting has a data type of **Long**.

You can set the default for this property by using a control's **default control style** or the **DefaultControl** method in Visual Basic.

For **Table** objects you can set this property using **Fill/Back Color** on the **Formatting (Datasheet)** toolbar, or in Visual Basic by using the **DatasheetBackColor** property.
Remarks

To use the **BackColor** property, the **BackStyle** property, if available, must be set to Normal.
Example

The following example uses the RGB function to set the **BorderColor**, **BackColor**, and **ForeColor** properties depending on the value of the **txtPastDue** text box. You can also use the QBColor function to set these properties. Putting the following code in the Form_Current() event sets the control display characteristics as soon as the user opens a form or moves to a new record.

```vba
Sub Form_Current()
    Dim curAmntDue As Currency, lngBlack As Long
    Dim lngRed As Long, lngYellow As Long, lngWhite As Long

    If Not IsNull(Me!txtPastDue.Value) Then
        curAmntDue = Me!txtPastDue.Value
    Else
        Exit Sub
    End If
    lngRed = RGB(255, 0, 0)
    lngBlack = RGB(0, 0, 0)
    lngYellow = RGB(255, 255, 0)
    lngWhite = RGB(255, 255, 255)
    If curAmntDue > 100 Then
        Me!txtPastDue.BorderColor = lngRed
        Me!txtPastDue.ForeColor = lngRed
        Me!txtPastDue.BackColor = lngYellow
    Else
        Me!txtPastDue.BorderColor = lngBlack
        Me!txtPastDue.ForeColor = lngBlack
        Me!txtPastDue.BackColor = lngWhite
    End If
End Sub
```
BackStyle Property

You can use the BackStyle property to specify whether a control will be transparent. Read/write Byte.

expression.BackStyle

expression    Required. An expression that returns one of the objects in the Applies To list.
## Setting

The **BackStyle** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal</td>
<td>1</td>
<td>(Default for all controls except option group) The control has its interior color set by the BackColor property.</td>
</tr>
<tr>
<td>Transparent</td>
<td>0</td>
<td>(Default for option group) The control is transparent. The color of the form or report behind the control is visible.</td>
</tr>
</tbody>
</table>

You can set this property by using **Fill/Back Color** on the **Formatting (Form/Report)** toolbar, a control's **property sheet**, a **macro**, or **Visual Basic**.

You can set the default for this property by using a control's **default control style** or the **DefaultControl** method in Visual Basic.
Remarks

If the **Transparent** button on the **Back Color** button palette is selected, the **BackStyle** property is set to Transparent; otherwise the **BackStyle** property is set to Normal.

**Tip**

To make a **command button** invisible, set its **Transparent** property to **Yes**.
BaseConnectionString Property

You can use the **BaseConnectionString** property to return the base connection string for the **CurrentProject** or **CodeProject** object. Read-only String.

`expression.BaseConnectionString`

`expression`  Required. An expression that returns one of the objects in the Applies To list.
Remarks

The `BaseConnectionString` property is available only by using Visual Basic and is read-only.

The `BaseConnectionString` property returns the connection string that was set through the `OpenConnection` method or by clicking `Connection` on the File menu. When making a connection, Microsoft Access project modifies the `BaseConnectionString` property for use with the ADO environment.
Example

The following example displays the `BaseConnectionString` property setting of the current project:

```vba
Public Sub ShowConnectString()
    Dim objCurrent As Object

    Set objCurrent = Application.CurrentProject
    MsgBox "The current base connection is " & objCurrent.BaseConnectionString
End Sub
```
**BeforeDelConfirm Property**

Returns or sets a **String** indicating which macro, event procedure, or user-defined function runs when the **BeforeDelConfirm** event occurs. Read/write.

`expression.BeforeDelConfirm`

`expression` Required. An expression that returns one of the objects in the Applies To list.
Remarks

Valid values for this property are "macroname" where macroname is the name of macro, "[Event Procedure]" which indicates the event procedure associated with the BeforeDelConfirm event for the specified object, or "=functionname()" where functionname is the name of a user-defined function. For a more detailed discussion of event properties, see "Event Properties."
Example

The following example specifies that when the BeforeDelConfirm event occurs on the first form of the current project, the associated event procedure should run.

\texttt{Forms(0).BeforeDelConfirm = "[Event Procedure]"}
BeforeInsert Property

Returns or sets a String indicating which macro, event procedure, or user-defined function runs when the BeforeInsert event occurs. Read/write.

expression.BeforeInsert

expression Required. An expression that returns one of the objects in the Applies To list.
Remarks

Valid values for this property are "macroname" where macroname is the name of macro, "[Event Procedure]" which indicates the event procedure associated with the BeforeInsert event for the specified object, or "=functionname()" where functionname is the name of a user-defined function. For a more detailed discussion of event properties, see "Event Properties."
Example

The following example specifies that when the BeforeInsert event occurs on the first form of the current project, the associated event procedure should run.

Forms(0).**BeforeInsert**  = "[Event Procedure]"

**BeforeQuery Property**

Returns or sets a **String** indicating which macro, event procedure, or user-defined function runs when the **BeforeQuery** event occurs. Read/write.

*expression*.**BeforeQuery**

*expression* Required. An expression that returns one of the objects in the Applies To list.
Remarks

Valid values for this property are "macroname" where macroname is the name of macro, "[Event Procedure]" which indicates the event procedure associated with the BeforeQuery event for the specified object, or "=functionname()" where functionname is the name of a user-defined function. For a more detailed discussion of event properties, see "Event Properties."
Example

The following example specifies that when the BeforeQuery event occurs on the first form of the current project, the associated event procedure should run.

`Forms(0).BeforeQuery = "[Event Procedure]"`
**BeforeRender Property**

Returns or sets a **String** indicating which macro, event procedure, or user-defined function runs when the **BeforeRender** event occurs. Read/write.

`expression.BehoreRender`

`expression` Required. An expression that returns one of the objects in the Applies To list.
Remarks

Valid values for this property are "macroname" where macroname is the name of macro, "[Event Procedure]" which indicates the event procedure associated with the BeforeRender event for the specified object, or "=functionname()" where functionname is the name of a user-defined function. For a more detailed discussion of event properties, see "Event Properties."
Example

The following example specifies that when the BeforeRender event occurs on the first form of the current project, the associated event procedure should run.

Forms(0).BeforeRender = "[Event Procedure]"
**BeforeScreenTip Property**

Returns or sets a **String** indicating which macro, event procedure, or user-defined function runs when the **BeforeScreenTip** event occurs. Read/write.

`expression.BehoreScreenTip`

`expression` Required. An expression that returns one of the objects in the Applies To list.
Remarks

Valid values for this property are "macroname" where macroname is the name of macro, "[Event Procedure]" which indicates the event procedure associated with the BeforeScreenTip event for the specified object, or "=functionname()" where functionname is the name of a user-defined function. For a more detailed discussion of event properties, see "Event Properties."
Example

The following example specifies that when the BeforeScreenTip event occurs on the first form of the current project, the associated event procedure should run.

`Forms(0).BeforeScreenTip = "[Event Procedure]"`
BeforeUpdate Property

Returns or sets a String indicating which macro, event procedure, or user-defined function runs when the BeforeUpdate event occurs. Read/write.

expression.BeforeUpdate

expression  Required. An expression that returns one of the objects in the Applies To list.
Remarks

Valid values for this property are "macroName" where macroName is the name of macro, "[Event Procedure]" which indicates the event procedure associated with the BeforeUpdate event for the specified object, or "=functionName()" where functionName is the name of a user-defined function. For a more detailed discussion of event properties, see "Event Properties."
Example

The following example specifies that when the BeforeUpdate event occurs on the first form of the current project, the associated event procedure should run.

`Forms(0).BeforeUpdate = "[Event Procedure]"`
Bookmark Property

You can use the **Bookmark** property with forms to set a [bookmark] that uniquely identifies a particular [record] in the form's underlying table, query, or [SQL statement]. Read/write **Variant**.

*expression*.Bookmark

*expression* Required. An expression that returns one of the objects in the Applies To list.
Setting

The **Bookmark** property contains a string expression created by Microsoft Access.

You can set this property by using a [macro](#) or [Visual Basic](#).

**Note** You get or set the form's **Bookmark** property separately from the ADO **Bookmark** or DAO **Bookmark** property of the underlying table or query.
Remarks

When a bound form is opened in Form view, each record is assigned a unique bookmark. In Visual Basic, you can save the bookmark for the current record by assigning the value of the form's Bookmark property to a string variable. To return to a saved record after moving to a different record, set the form's Bookmark property to the value of the saved string variable. You can use the StrComp function to compare a Variant or string variable to a bookmark, or when comparing a bookmark against a bookmark. The third argument for the StrComp function must be set to a value of zero.

Note  Bookmarks are not saved with the records they represent and are only valid while the form is open. They are re-created by Microsoft Access each time a bound form is opened.

There is no limit to the number of bookmarks you can save if each is saved with a unique string variable.

The Bookmark property is only available for the form's current record. To save a bookmark for a record other than the current record, move to the desired record and assign the value of the Bookmark property to a string variable that identifies this record.

You can use bookmarks in any form that is based entirely on Microsoft Access tables. However, other database products may not support bookmarks. For example, you can't use bookmarks in a form based on a linked table that has no primary index.

Requerying a form invalidates any bookmarks set on records in the form. However, clicking Refresh on the Records menu doesn't affect bookmarks.

Since Microsoft Access creates a unique bookmark for each record in a form's recordset when a form is opened, a form's bookmark will not work on another recordset, even when the two recordsets are based on the same table, query, or SQL statement. For example, suppose you open a form bound to the Customers table. If you then open the Customers table by using Visual Basic and use the ADO Seek or DAO Seek method to locate a specific record in the table, you can't set the form's Bookmark property to the current table record. To perform
this kind of operation you can use the ADO Find method or DAO Find methods with the form's **RecordsetClone** property.

An error occurs if you set the **Bookmark** property to a string variable and then try to return to that record after the record has been deleted.

The value of the **Bookmark** property isn't the same as a record number.
Example

To test the following example with the Northwind sample database, you need to add a command button named cmdFindContactName to the Suppliers form, and then add the following code to the button's Click event. When the button is clicked, the user is asked to enter a portion of the contact name to find. If the name is found, the form's **Bookmark** property is set to the **Recordset** object's DAO **Bookmark** property, which moves the form's current record to the found name.

Private Sub cmdFindContactName_Click()
    Dim rst As DAO.Recordset
    Dim strCriteria As String

    strCriteria = "[ContactName] Like '*" & InputBox("Enter the " _
                      & "first few letters of the name to find") & "**"

    Set rst = Me.RecordsetClone
    rst.FindFirst strCriteria
    If rst.NoMatch Then
        MsgBox "No entry found.", vbInformation
    Else
        Me.Bookmark = rst.Bookmark
    End If

    Set rst = Nothing
End Sub
**BorderColor Property**

You can use the **BorderColor** property to specify the color of a control's border. Read/write **Long**.

`expression.BorderColor`

`expression` Required. An expression that returns one of the objects in the Applies To list.
Setting

The **BorderColor** property setting is a *numeric expression* that corresponds to the color you want to use for a control's border.

You can use the Color Builder to set this property by clicking the **Build** button to the right of the property box in the property sheet. Using the Color Builder enables you to define custom border colors for controls.

You can also set this property by using **Line/Border Color** on the Formatting (Form/Report) toolbar, a **macro**, or **Visual Basic**.

You can set the default for this property by using a control's **default control style** or the **DefaultControl** method in Visual Basic.
Remarks

A control's border color is visible only when its `SpecialEffect` property is set to Flat or Shadowed. If the `SpecialEffect` property is set to something other than Flat or Shadowed, setting the `BorderColor` property changes the `SpecialEffect` property setting to Flat.
Example

The following example uses the **RGB** function to set the **BorderColor**, **BackColor**, and **ForeColor** properties depending on the value of the `txtPastDue` text box. You can also use the **QBColor** function to set these properties. Putting the following code in the `Form_Current()` event sets the control display characteristics as soon as the user opens a form or moves to a new record.

```vbnet
Sub Form_Current()
    Dim curAmntDue As Currency, lngBlack As Long
    Dim lngRed As Long, lngYellow As Long, lngWhite As Long

    If Not IsNull(Me!txtPastDue.Value) Then
        curAmntDue = Me!txtPastDue.Value
    Else
        Exit Sub
    End If

    lngRed = RGB(255, 0, 0)
    lngBlack = RGB(0, 0, 0)
    lngYellow = RGB(255, 255, 0)
    lngWhite = RGB(255, 255, 255)

    If curAmntDue > 100 Then
        Me!txtPastDue.BorderColor = lngRed
        Me!txtPastDue.ForeColor = lngRed
        Me!txtPastDue.BackColor = lngYellow
    Else
        Me!txtPastDue.BorderColor = lngBlack
        Me!txtPastDue.ForeColor = lngBlack
        Me!txtPastDue.BackColor = lngWhite
    End If
End Sub
```
BorderStyle Property

- **Forms.** Specifies the type of border and border elements (**title bar**, **Control menu**, **Minimize** and **Maximize** buttons, or **Close** button) to use for the form. You typically use different border styles for normal forms, **pop-up forms**, and **custom dialog boxes**.
- **Controls.** Specifies how a control's border appears.

Read/write **Byte.**

*expression*.BorderStyle

*expression*  Required. An expression that returns one of the objects in the Applies To list.
Setting

For forms, the **BorderStyle** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>0</td>
<td>The form has no border or related border elements. The form isn't resizable. The form has a thin border and can include any of the border elements. The form isn't resizable (the <strong>Size</strong> command on the <strong>Control</strong> menu isn't available). You often use this setting for pop-up forms. (If you want a form to remain on top of all Microsoft Access windows, you must also set its <strong>PopUp</strong> property to Yes.)</td>
</tr>
<tr>
<td>Thin</td>
<td>1 Control</td>
<td>(Default) The form has the default border for Microsoft Access forms, can include any of the border elements, and can be resized. You often use this setting for normal Microsoft Access forms. The form has a thick (double) border and can include only a title bar, <strong>Close</strong> button, and <strong>Control</strong> menu. The form can't be maximized, minimized, or resized (the <strong>Maximize</strong>, <strong>Minimize</strong>, and <strong>Size</strong> commands aren't available on the <strong>Control</strong> menu). You often use this setting for custom dialog boxes. (If you want a form to be <strong>modal</strong>, however, you must also set its <strong>Modal</strong> property to Yes. If you want it to be a modal pop-up form, like most dialog boxes, you must set both its <strong>PopUp</strong> and <strong>Modal</strong> properties to Yes.)</td>
</tr>
<tr>
<td>Sizable</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Dialog</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

You can set the **BorderStyle** property for a form only in **form Design view** by using the form's **property sheet**, a **macro**, or **Visual Basic**.

For controls, the **BorderStyle** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transparent 0</td>
<td>(Default only for <strong>label</strong>, <strong>chart</strong>, and <strong>subreport</strong>)</td>
<td>Transparent</td>
</tr>
</tbody>
</table>
You can set the **BorderStyle** property for a control by using the control's property sheet, a macro, or Visual Basic.

You can set the default for this property by using a control's default control style or the **DefaultControl** method in Visual Basic.
Remarks

A control's border style is visible only when its SpecialEffect property is set to Flat or Shadowed. If the SpecialEffect property is set to something other than Flat or Shadowed, setting theBorderStyle property changes the SpecialEffect property setting to Flat.

For a form, theBorderStyle property establishes the characteristics that visually identify the form as a normal form, a pop-up form, or a custom dialog box. You may also set the Modal and PopUp properties to further define the form's characteristics.

You may also want to set the form's ControlBox, CloseButton, MinMaxButtons, ScrollBars, NavigationButtons, and RecordSelectors properties. These properties interact in the following ways:

- If the BorderStyle property is set to None or Dialog, the form doesn't have Maximize or Minimize buttons, regardless of its MinMaxButtons property setting.
- If the BorderStyle property is set to None, the form doesn't have a Control menu, regardless of its ControlBox property setting.
- The BorderStyle property setting doesn't affect the display of the scroll bars, navigation buttons, record number box, or record selectors.

The BorderStyle property takes effect only in Form view. The property setting is ignored in form Design view.

If you set the BorderStyle property of a pop-up form to None, you won't be able to close the form unless you add a Close button to it that runs a macro containing the Close action or an event procedure in Visual Basic that uses the Close method.

Pop-up forms are typically fixed in size, but you can make a pop-up form sizable by setting its PopUp property to Yes and its BorderStyle property to Sizable.

You can also use the Dialog setting of the Window Mode action argument of the OpenForm action to open a form with its Modal and PopUp properties set to Yes.
**BorderWidth Property**

You can use the **BorderWidth** property to specify the width of a control's border. Read/write **Byte**.

`expression.BorderWidth`

*expression* Required. An expression that returns one of the objects in the Applies To list.
## Setting

The **BorderWidth** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hairline</td>
<td>0</td>
<td>(Default) The narrowest border possible on your system.</td>
</tr>
<tr>
<td>1 pt to 6 pt</td>
<td>1 to 6</td>
<td>The width as indicated in points.</td>
</tr>
</tbody>
</table>

You can set this property by using Line/Border Width on the Formatting (Form/Report) toolbar, the control's property sheet, a macro, or Visual Basic.

You can set the default for this property by using the control's default control style or the DefaultControl method in Visual Basic.
Remarks

To use the `BorderWidth` property, the `SpecialEffect` property must be set to Flat or Shadowed and the `BorderStyle` property must not be set to Transparent. If the `SpecialEffect` property is set to any other value and/or the `BorderStyle` property is set to Transparent, and you set the `BorderWidth` property, the `SpecialEffect` property is automatically reset to Flat and the `BorderStyle` property is automatically reset to Solid.

The exact border width depends on your computer and printer. On some systems, the hairline and 1-point widths appear the same.

---
BottomMargin Property

BottomMargin property as it applies to the Label and TextBox objects.

Along with the LeftMargin, RightMargin, and TopMargin properties, specifies the location of information displayed within a label or text box control. Read/write Integer.

expression.BottomMargin

expression Required. An expression that returns one of the above objects.
Remarks

A control's displayed information location is the distance measured from the control's left, top, right, or bottom border to the left, top, right, or bottom edge of the displayed information. To use a unit of measurement different from the setting in the regional settings of Windows, specify the unit (for example, cm or in).

In Visual Basic, use a numeric expression to set the value of this property. Values are expressed in twips.

You can set these properties by using the property sheet, a macro, or Visual Basic.

**BottomMargin property as it applies to the Printer object.**

Along with the TopMargin, RightMargin, and LeftMargin properties, specifies the margins for a printed page. Read/write Long.

*expression*.BottomMargin

*expression* Required. An expression that returns a Printer object.
Example

As it applies to the **Label** and **TextBox** objects.

The following example offsets the caption in the label "EmployeeID_Label" of the "Purchase Orders" form by 100 twips from the bottom of the label's border.

```vba
With Forms.Item("Purchase Orders").Controls.Item("EmployeeID_Label")
    .BottomMargin = 100
End With
```
BoundColumn Property

When you make a selection from a list box or combo box, the BoundColumn property tells Microsoft Access which column's values to use as the value of the control. If the control is bound to a field, the value in the column specified by the BoundColumn property is stored in the field named in the ControlSource property. Read/write Long.

expression.BoundColumn

expression Required. An expression that returns one of the objects in the Applies To list.
Setting

The **BoundColumn** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The <strong>ListIndex</strong> property value, rather than the column value, is stored in the current record. The <strong>ListIndex</strong> property value of the first row is 0, the second row is 1, and so on. Microsoft Access sets the <strong>ListIndex</strong> property when an item is selected from a list box or the list box portion of a combo box. Setting the <strong>BoundColumn</strong> property to 0 and using the <strong>ListIndex</strong> property value of the control might be useful if, for example, you are only interested in storing a sequence of numbers. (Default is 1) The value in the specified column becomes the control's value. If the control is bound to a field, then this setting is stored in that field in the current record. The <strong>BoundColumn</strong> property can't be set to a value larger than the setting of the <strong>ColumnCount</strong> property.</td>
</tr>
<tr>
<td>1 or greater</td>
<td>You can set the <strong>BoundColumn</strong> property by using the control's property sheet, a macro, or Visual Basic.</td>
</tr>
</tbody>
</table>

For **table fields**, you can set this property on the **Lookup** tab in the Field Properties section of **table Design view** for fields with the **DisplayControl** property set to Combo Box or List Box.

**Tip**

Microsoft Access sets the **BoundColumn** property automatically when you select Lookup Wizard as the data type for a field in table Design view.

In Visual Basic, set the **BoundColumn** property by using a number or a numeric expression equal to a value from 0 to the setting of the **ColumnCount** property.
Remarks

The leftmost visible column in a combo box (the leftmost column whose setting in the combo box's `ColumnWidths` property is not 0) contains the data that appears in the text box part of the combo box in Form view or in a report. The `BoundColumn` property determines which column's value in the text box or combo box list will be stored when you make a selection. This allows you to display different data than you store as the value of the control.

**Note** If the bound column is not the same as the leftmost visible column in the control (or if you set the `BoundColumn` property to 0), the `LimitToList` property is set to Yes.

Microsoft Access uses zero-based numbers to refer to columns in the `Column` property. That is, the first column is referenced by using the expression `Column(0)`; the second column is referenced by using the expression `Column(1)`; and so on. However, the `BoundColumn` property uses 1-based numbers to refer to the columns. This means that if the `BoundColumn` property is set to 1, you could access the value stored in that column by using the expression `Column(0)`.

If the `AutoExpand` property is set to Yes, Microsoft Access automatically fills in a value in the text box portion of the combo box that matches a value in the combo box list as you type.
BrokenReference Property

Returns a **Boolean** indicating whether the current database has any broken references to databases or type libraries. **True** if there are any broken references. Read-only.

*expression*.BrokenReference

*expression*    Required. An expression that returns one of the objects in the Applies To list.
Remarks

To test the validity of a specific reference, use the `IsBroken` property of the `Reference` object.
Example

This example checks to see if there are any broken references in the current database and reports the results to the user.

' Looping variable.
Dim refLoop As Reference ' Output variable.
Dim strReport As String

' Test whether there are broken references.
If Application.BrokenReference = True Then
    strReport = "The following references are broken:" & vbCrLf

        ' Test validity of each reference.
    For Each refLoop In Application.References
        If refLoop.IsBroken = True Then
            strReport = strReport & "   " & refLoop.Name & vbCrLf
        End If
    Next refLoop
Else
    strReport = "All references in the current database are valid."
End If

' Display results.
MsgBox strReport
Build Property

Returns as a **Long** representing the build number of the currently installed copy of Microsoft Access. Read-only.

*expression*.Build

*expression*  Required. An expression that returns one of the objects in the Applies To list.
Example

The following example displays the version and build number of the currently-installed copy of Microsoft Access.

MsgBox "You are currently running Microsoft Access, " _
& " version " & Application.Version & ", build " _
& Application.Build & "."
BuiltIn Property

The `BuiltIn` property returns a `Boolean` value indicating whether a `Reference` object points to a default reference that's necessary for Microsoft Access to function properly. Read-only `Boolean`.

`expression.BuiltIn`

`expression`  Required. An expression that returns one of the objects in the Applies To list.
Setting

The **BuiltIn** property is available only by using Visual Basic and is read-only.

The **BuiltIn** property returns the following values.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>The <strong>Reference</strong> object refers to a default reference that can't be removed.</td>
</tr>
<tr>
<td>–1</td>
<td>The <strong>Reference</strong> object refers to a default reference that can't be removed.</td>
</tr>
<tr>
<td>False</td>
<td>The <strong>Reference</strong> object refers to a nondefault reference that isn't necessary for Microsoft Access to function properly.</td>
</tr>
<tr>
<td>0</td>
<td>The <strong>Reference</strong> object refers to a nondefault reference that isn't necessary for Microsoft Access to function properly.</td>
</tr>
</tbody>
</table>
Remarks

The default references in Microsoft Access include the Microsoft Access 10.0 object library, the Visual Basic For Applications library, OLE Automation library, and Microsoft ActiveX Data Objects 2.1 library.
Example

The following example prints the default references in the References collection:

Sub ReferenceBuiltInOnly()
    Dim ref As Reference

    ' Enumerate through References collection.
    For Each ref In References
        ' Check BuiltIn property.
        If ref.BuiltIn = True Then
            Debug.Print ref.Name
        End If
    Next ref
End Sub
Cancel Property

You can use the **Cancel** property to specify whether a command button is also the Cancel button on a form. Read/write **Boolean**.

`expression.Cancel`

`expression` Required. An expression that returns one of the objects in the Applies To list.
Setting

The **Cancel** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td><strong>True</strong></td>
<td>The command button is the Cancel button.</td>
</tr>
<tr>
<td>No</td>
<td><strong>False</strong></td>
<td>(Default) The command button isn't the Cancel button.</td>
</tr>
</tbody>
</table>

You can set this property by using the command button's [property sheet](#), a [macro](#), or [Visual Basic](#).
Remarks

Setting the `Cancel` property to Yes makes the command button the Cancel button in the form. However, you must still write the macro or Visual Basic `event procedure` that performs whatever action or actions you want the Cancel button to carry out (for example, closing the form without saving any changes to it). Set the command button's `OnClick event property` to the name of the macro or event procedure.

When a command button's `Cancel` property setting is Yes and the Form window is active, the user can choose the command button by pressing ESC, by pressing ENTER when the command button has the `focus`, or by clicking the command button.

**Note** If a text box has the focus when the user presses ESC, any changes made to the data in the text box will be lost, and the original data will be restored.

When the `Cancel` property is set to Yes for one command button on a form, it is automatically set to No for all other command buttons on the form.

**Tip**

For a form that supports irreversible operations, such as deletions, it's a good idea to make the Cancel button the default command button. To do this, set both the `Cancel` property and the `Default` property to Yes.
CanGrow Property

You can use the CanGrow property to control the appearance of sections or controls on forms and reports that are printed or previewed. For example, if you set the property to Yes, a section or control automatically adjusts vertically to print or preview all the data the section or control contains.

Notes

- The CanGrow property does not apply to a form or report page header and page footer sections, although it does apply to controls in such sections.
- This property affects the display of form sections and controls only when the form is printed or previewed, not when the form is displayed in Form view, Datasheet view, or Design view.

Read/write Boolean.

equation.CanGrow

equation Required. An expression that returns one of the objects in the Applies To list.
Setting

The **CanGrow** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td><strong>True</strong> (–1)</td>
<td>The section or control grows vertically so that all data it contains can be printed or previewed.</td>
</tr>
<tr>
<td>No</td>
<td><strong>False</strong> (0)</td>
<td>(Default) The section or control doesn't grow. Data that doesn't fit within the fixed size of the section or control won't be printed or previewed.</td>
</tr>
</tbody>
</table>

**Note**  You can set this property only by using the section or control's [property sheet](mailto:). For controls, you can set the default for these properties by using the [default control style](mailto:) or the **DefaultControl** method in Visual Basic.

This property setting is read-only in a [macro](mailto:) or Visual Basic in any view but Design view.
Remarks

You can use this property to control the appearance of printed forms and reports. When you set the property to Yes, the object automatically adjusts so any amount of data can be printed. When a control grows, the controls below it move down the page.

If you set a control's **CanGrow** property to Yes, Microsoft Access automatically sets the **CanGrow** property of the section containing the control to Yes.

Sections grow vertically across their entire width. To grow the data independently, you can place two **subform** or **subreport** controls side by side, and set their **CanGrow** property to Yes.

When you use the **CanGrow** property, remember that:

- The property settings don't affect the horizontal spacing between controls; they affect only the vertical space the controls occupy.
- Overlapping controls can't grow.
CanShrink Property

You can use the CanShrink property to control the appearance of sections or controls on forms and reports that are printed or previewed. For example, if you set the property to Yes, a section or control automatically adjusts vertically to print or preview all the data the section or control contains.

Notes

- The CanShrink property does not apply to form or report page header and page footer sections, although it does apply to controls in such sections.
- This property affects the display of form sections and controls only when the form is printed or previewed, not when the form is displayed in Form view, Datasheet view, or Design view.

Read/write Boolean.

expression. CanShrink

expression Required. An expression that returns one of the objects in the Applies To list.
### Setting

The **CanShrink** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td><strong>True</strong></td>
<td>The section or control shrinks vertically so that the data it contains can be printed or previewed without leaving blank lines.</td>
</tr>
<tr>
<td>No</td>
<td><strong>False</strong></td>
<td>(Default) The section or control doesn't shrink.</td>
</tr>
</tbody>
</table>

**Note** You can set this property only by using the section or control's [property sheet](#).

For controls, you can set the default for this property by using the [default control style](#) or the [DefaultControl](#) method in Visual Basic.

This property setting is read-only in a [macro](#) or Visual Basic in any view but Design view.
Remarks

You can use this property to control the appearance of printed forms and reports. When you set the property to Yes, the object automatically adjusts so any amount of data can be printed. When a control shrinks, the controls below it move up the page.

If you set a control's CanShrink property to Yes, Microsoft Access does not set the section's CanShrink property to Yes.

Sections shrink vertically across their entire width. For example, suppose a form has two text boxes side by side in a section, and each control has its CanShrink property set to Yes. If one text box contains one line of data and the other text box contains two lines of data, both text boxes will be two lines long because the section is sized across its entire width. To shrink the data independently, you can place two subform or subreport controls side by side, and set their CanShrink property to Yes.

When you use the CanShrink property, remember that:

- The property settings don't affect the horizontal spacing between controls; they affect only the vertical space the controls occupy.
- Overlapping controls can't shrink.
- The height of a large control can prevent controls beside it from shrinking. For example, if several short controls are on the left side of a report's detail section and one tall control, such as an unbound object frame, is on the right side, the controls on the left won't shrink, even if they contain no data.
Caption Property

You can use the **Caption** property to provide helpful information to the user through captions on objects in various views:

- Field captions specify the text for labels attached to **controls** created by dragging a field from the **field list** and serves as the column heading for the field in table or query **Datasheet view**.
- Form captions specify the text that appears in the title bar in **Form view**.
- Report captions specify the title of the report in **Print Preview**.
- Button and label captions specify the text that appears in the control.

Read/write **String**.

`expression.Caption`

`expression` Required. An expression that returns one of the objects in the Applies To list.
Setting

The **Caption** property is a *string expression* that can contain up to 2,048 characters. Captions for forms and reports that are too long to display in the title bar are truncated.

For controls, you can set this property by using the **property sheet**. For fields, you can set this property by using the **property sheet** in table **Design view** or in the **Query window** (in the Field Properties **property sheet**). You can also set this property by using a **macro** or **Visual Basic**.
Remarks

If you don't specify a caption for a table field, the field's FieldName property setting will be used as the caption of a label attached to a control or as the column heading in Datasheet view. If you don't specify the caption for a query field, the caption for the underlying table field will be used. If you don't set a caption for a form, button, or label, Microsoft Access will assign the object a unique name based on the object, such as "Form1".

If you create a control by dragging a field from the field list and haven't specified a Caption property setting for the field, the field's FieldName property setting will be copied to the control's Name property box and will also appear in the label of the control created.

Note The text of the Caption property for a label or command button control is the hyperlink display text when the HyperlinkAddress or HyperlinkSubAddress property is set for the control.

You can use the Caption property to assign an access key to a label or command button. In the caption, include an ampersand (&) immediately preceding the character you want to use as an access key. The character will be underlined. You can press ALT plus the underlined character to move the focus to that control on a form.

Tip

Include two ampersands (&&) in the setting for a caption if you want to display an ampersand itself in the caption text. For example, to display "Save & Exit", you should type Save && Exit in the Caption property box.
ChartSpace Property

Returns a `ChartSpace` object.

`expression.ChartSpace`

`expression`  Required. An expression that returns one of the objects in the Applies To list.
Remarks

You must set a reference to the Microsoft Office Web Components type library in order to use this property.
Example

This example reports the version of Microsoft Office Web Components in use for the specified form.

Dim objChartSpace As ChartSpace
Set objChartSpace = Forms(0).ChartSpace
MsgBox "Current version of Office Web Components: " & objChartSpace.Version
**CheckIfOfficeIsHTMLEditor Property**

You can use the `CheckIfOfficeIsHTMLEditor` property to specify or determine if the default system HTML editor is used when editing a Web page. Read/write Boolean.

```
expression.CheckIfOfficeIsHTMLEditor
```

`expression` Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **CheckIfOfficeIsHTMLEditor** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>True</td>
<td>(Default) Check to see whether an Office application is the default HTML editor when starting Microsoft Access.</td>
</tr>
<tr>
<td>No</td>
<td>False</td>
<td>Check is not performed.</td>
</tr>
</tbody>
</table>

**Note** The **CheckIfOfficeIsHTMLEditor** property is available only by using **Visual Basic**.

The **CheckIfOfficeIsHTMLEditor** property is used only if the Web browser you are using supports HTML editing and HTML editors.

To use a different HTML editor, you must set this property to **False** and then register the editor as the default system HTML editor.
Example

This example causes the default system HTML editor to be used (instead of Office applications) when editing a Web page.

Application.DefaultWebOptions.CheckIfOfficeIsHTMLEditor = False
Class Property

You can use the Class property to specify or determine the class name of an embedded OLE object. Read/write String.

expression.Class

expression    Required. An expression that returns one of the objects in the Applies To list.
Setting

The **Class** property setting is a [string expression](en) supplied by you or Microsoft Access when you create or paste an OLE object.

You can set the **Class** property by using the [control's property sheet](en), a [macro](en), or [Visual Basic](en).
Remarks

A class name defines the type of OLE object. For example, Microsoft Excel supports several types of OLE objects, including worksheets and charts. Their class names are "Excel.Sheet" and "Excel.Chart" respectively. When you create an OLE object in Design view by clicking Paste Special on the Edit menu or Object on the Insert menu, Microsoft Access enters the class name of the new object in the property sheet.

Note To determine the class name of an OLE object, see the documentation for the application supplying the object.

The Class property setting is updated when you copy an object from the Clipboard. For example, if you paste a Microsoft Excel chart from the Clipboard into an OLE object that previously contained a Microsoft Excel worksheet, the Class property setting changes from "Excel.Sheet" to "Excel.Chart". You can paste an object from the Clipboard by using Visual Basic to set the control's Action property to acOLEPaste or acOLEPasteSpecialDlg.

Note The OLEClass property and the Class property are similar but not identical. The OLEClass property setting is a general description of the OLE object whereas the Class property setting is the name used to refer to the OLE object in Visual Basic. Examples of OLEClass property settings are Microsoft Excel Chart, Microsoft Word Document, and Paintbrush Picture.
Example

The following example creates a linked OLE object using an unbound object frame named OLE1 and sizes the control to display the object’s entire contents when the user clicks a command button.

Sub Command1_Click
    OLE1.Class = "Excel.Sheet"  ' Set class name.
    ' Specify type of object.
    OLE1.OLETypeAllowed = acOLELinked
    ' Specify source file.
    OLE1.SourceDoc = "C:\Excel\Oletext.xls"
    ' Specify data to create link to.
    OLE1.SourceItem = "R1C1:R5C5"
    ' Create linked object.
    OLE1.Action = acOLECreateLink
    ' Adjust control size.
    OLE1.SizeMode = acOLESizeZoom
End Sub
CloseButton Property

Specifies whether the Close button on a form is enabled. Read/write Boolean.

expression.CloseButton

expression Required. An expression that returns one of the objects in the Applies To list.
Setting

The **CloseButton** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>True</td>
<td>(Default) The <strong>Close</strong> button is enabled.</td>
</tr>
<tr>
<td>No</td>
<td>False</td>
<td>The <strong>Close</strong> button is disabled and the <strong>Close</strong> command isn't available on the <strong>Control</strong> menu.</td>
</tr>
</tbody>
</table>

You can set the **CloseButton** property by using the form's [property sheet](#), a [macro](#), or [Visual Basic](#).

You can set the **CloseButton** property only in [form Design view](#).
Remarks

If you set the **CloseButton** property to No, the **Close** button remains visible but appears dimmed (grayed), and you must provide some other way to close the form— for example, a [command button](#) or custom menu command that runs a macro or [event procedure](#) that closes the form.

**Tip**

You can also close the form by pressing ALT+F4.
**CodeContextObject Property**

You can use the `CodeContextObject` property to determine the object in which a macro or Visual Basic code is executing. Read-only Object.

`expression.CodeContextObject`

`expression`  Required. An expression that returns one of the objects in the Applies To list.
Setting

The CodeContextObject property is set by Microsoft Access and is read-only in all views.
Remarks

The ActiveControl, ActiveDataAccessPage, ActiveDatasheet, ActiveForm, and ActiveReport properties of the Screen object always return the object that currently has the focus. The object with the focus may or may not be the object where a macro or Visual Basic code is currently running, for example, when Visual Basic code runs in the Timer event on a hidden form.
Example

In the following example the **CodeContextObject** property is used in a function to identify the name of the object in which an error occurred. The object name is then used in the message box title as well as in the body of the error message. The **Error** statement is used in the command button's click event to generate the error for this example.

Private Sub Command1_Click()
    On Error GoTo Command1_Err
    Error 11
        ' Generate divide-by-zero error.
    Exit Sub

    Command1_Err:
        If ErrorMessage("Command1_Click() Event", vbYesNo + _
            vbCrLf, Err) = vbYes Then
            Exit Sub
        Else
            Resume
        End If
    End Sub

Function ErrorMessage(strText As String, intType As Integer, _
    intErrVal As Integer) As Integer
    Dim objCurrent As Object
    Dim strMsgboxTitle As String
    Set objCurrent = **CodeContextObject**
    strMsgboxTitle = "Error in " & objCurrent.Name
    strText = strText & "Error #" & intErrVal _
        & " occurred in " & objCurrent.Name
    ErrorMessage = MsgBox(strText, intType, strMsgboxTitle)
    Err = 0
End Function
CodeData Property

You can use the CodeData property to access the CodeData object and its related collections. Read-only CodeData object.

expression.CodeData

expression  Required. An expression that returns one of the objects in the Applies To list.
Setting

The **CodeData** property is available only by using **Visual Basic** and is read-only.
Remarks

Use the **CodeData** property to refer to one of the following code database collections together with one of its properties or methods.

<table>
<thead>
<tr>
<th>AllTables</th>
<th>AllQueries</th>
</tr>
</thead>
<tbody>
<tr>
<td>AllViews</td>
<td>AllStoredProcedures</td>
</tr>
<tr>
<td>AllDatabaseDiagrams</td>
<td></td>
</tr>
</tbody>
</table>

**CodeProject Property**

You can use the **CodeProject** property to access the **CodeProject** object and its related collections, properties, and methods. Read-only **CodeProject** object.

`expression.CodeProject`

`expression`  Required. An expression that returns one of the objects in the Applies To list.
Setting

The **CodeProject** property is available only by using [Visual Basic](https://visualbasic.net) and is read-only.
Remarks

Use the `CodeProject` property to refer to one of the following code database collections together with one of its properties or methods.

- **AllForms**
- **AllReports**
- **AllDataAccessPages**
- **AllMacros**
- **AllModules**
- **AccessObjectProperties**
Collection Property

The **Collection** property returns a reference to the collection that contains an object. Read-only **References** object.

`expression.Collection`

`expression` Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **Collection** property is available only by using Visual Basic and is read-only.

You can use the **Collection** property to access the collection to which an object belongs. For example, the **Collection** property of a **Reference** object returns an object reference to the **References** collection.

The **Collection** property is similar to the **Parent** property.
ColorMode Property

Returns or sets an AcPrintColor constant representing whether the specified printer should print output in color or monochrome. Read/write.

AcPrintColor can be one of these AcPrintColor constants.

acPRCMColor
acPRCMMonochrome

expression.ColorMode

expression  Required. An expression that returns one of the objects in the Applies To list.
Example

The following example sets a variety of printer settings for the form specified in the `strFormname` argument of the procedure.

Sub SetPrinter(strFormname As String)

    DoCmd.OpenForm FormName:=strFormname, view:=acDesign, _
    datamode:=acFormEdit, windowmode:=acHidden

    With Forms(form1).Printer

        .TopMargin = 1440
        .BottomMargin = 1440
        .LeftMargin = 1440
        .RightMargin = 1440

        .ColumnSpacing = 360
        .RowSpacing = 360

        .ColorMode = acPRCMColor
        .DataOnly = False
        .DefaultSize = False
        .ItemSizeHeight = 2880
        .ItemSizeWidth = 2880
        .ItemLayout = acPRVerticalColumnLayout
        .ItemsAcross = 6

        .Copies = 1
        .Orientation = acPRORLandscape
        .Duplex = acPRDPVertical
        .PaperBin = acPRBNAuto
        .PaperSize = acPRPSLetter
        .PrintQuality = acPRPQMedium

    End With

    DoCmd.Close objecttype:=acForm, objectname:=strFormname, _
    Save:=acSaveYes

End Sub
Column Property

You can use the Column property to refer to a specific column, or column and row combination, in a multiple-column combo box or list box. Read-only Variant.

expression.Column(Index, Row)

expression Required. An expression that returns one of the objects in the Applies To list.

Index Required Long. A long integer that can range from 0 to the setting of the ColumnCount property minus one.

Row Optional Variant. An integer that can range from 0 to the setting of the ListCount property minus 1.

This property setting is only available by using a macro or Visual Basic. This property setting isn't available in Design view and is read-only in other views.
Remarks

Use 0 to refer to the first column, 1 to refer to the second column, and so on. Use 0 to refer to the first row, 1 to refer to the second row, and so on. For example, in a list box containing a column of customer IDs and a column of customer names, you could refer to the customer name in the second column and fifth row as:

Forms!Contacts!Customers.Column(1, 4)

You can use the **Column** property to assign the contents of a combo box or list box to another control, such as a **text box**. For example, to set the **ControlSource** property of a text box to the value in the second column of a list box, you could use the following expression:

=Forms!Customers!CompanyName.Column(1)

If the user has made no selection when you refer to a column in a combo box or list box, the **Column** property setting will be **Null**. You can use the **IsNull** function to determine if a selection has been made, as in the following example:

If IsNull(Forms!Customers!Country)
    Then MsgBox "No selection."
End If

**Note** To determine how many columns a combo box or list box has, you can inspect the **ColumnCount** property setting.
Example

The following example uses the **Column** property and the **ColumnCount** property to print the values of a list box selection.

```vba
Public Sub Read_ListBox()
    Dim intNumColumns As Integer
    Dim intI As Integer
    Dim frmCust As Form

    Set frmCust = Forms!frmCustomers
    If frmCust!lstCustomerNames.ItemsSelected.Count > 0 Then
        ' Any selection?
        intNumColumns = frmCust!lstCustomerNames.ColumnCount
        Debug.Print "The list box contains "; intNumColumns; 
        IIf(intNumColumns = 1, " column", " columns"); 
        " of data."

        Debug.Print "The current selection contains:
        For intI = 0 To intNumColumns - 1
            ' Print column data.
            Debug.Print frmCust!lstCustomerNames.Column(intI)
        Next intI
    Else
        Debug.Print "You haven't selected an entry in the ", & "list box."
    End If

    Set frmCust = Nothing

End Sub
```
ColumnCount Property

You can use the ColumnCount property to specify the number of columns displayed in a list box or in the list box portion of a combo box, or sent to OLE objects in a chart control or unbound object frame. Read/write Integer.

expression.ColumnCount

description Required. An expression that returns one of the objects in the Applies To list.
Setting

The **ColumnCount** property holds an integer between 1 and the maximum number of **fields** in the **table**, **query**, or **SQL statement**, or the maximum number of values in the value list, specified in the **RowSource** property of the **control**.

You can set the **ColumnCount** property by using the control's **property sheet**, a **macro**, or **Visual Basic**.

For **table fields**, you can set this property on the **Lookup** tab in the Field Properties section of **table Design view** for fields with the **DisplayControl** property set to Combo Box or List Box.

**Tip**

Microsoft Access sets the **ColumnCount** property automatically when you select Lookup Wizard as the data type for a field in table Design view.
Remarks

For example, if you set the **ColumnCount** property for a list box on an Employees form to 3, one column can list last names, another can list first names, and the third can list employee ID numbers.

A combo box or list box can have multiple columns. If the control's **RowSource** property contains the name of a table, query, or SQL statement, a combo box or list box will display the fields from that source, from left to right, up to the number specified by the **ColumnCount** property.

To display a different combination of fields, create either a new query or a new SQL statement for the **RowSource** property, specifying the fields and the order you want.

If the **RowSource** property contains a list of values (the **RowSourceType** property is set to Value List), the values are put into the rows and columns of the combo box or list box in the order they are listed in the **RowSource** property. For example, if the **RowSource** property contains the list "Red; Green; Blue; Yellow" and the **ColumnCount** property is set to 2, the first row of the combo box or list box list will contain "Red" in the first column and "Green" in the second column. The second row will contain "Blue" in the first column and "Yellow" in the second column.

You can use the **ColumnWidths** property to set the width of the columns displayed in the control, or to hide columns.
Example

The following example uses the **Column** property and the **ColumnCount** property to print the values of a list box selection.

```vba
Public Sub Read_ListBox()
    Dim intNumColumns As Integer
    Dim intI As Integer
    Dim frmCust As Form

    Set frmCust = Forms!frmCustomers
    If frmCust!lstCustomerNames.ItemsSelected.Count > 0 Then
        ' Any selection?
        intNumColumns = frmCust!lstCustomerNames.ColumnCount
        Debug.Print "The list box contains "; intNumColumns; _
        IIf(intNumColumns = 1, " column", " columns"); _
        " of data."

        Debug.Print "The current selection contains:"
        For intI = 0 To intNumColumns - 1
            ' Print column data.
            Debug.Print frmCust!lstCustomerNames.Column(intI)
        Next intI
    Else
        Debug.Print "You haven't selected an entry in the " _
        & "list box."
    End If

    Set frmCust = Nothing

End Sub
```
ColumnHeads Property

You can use the ColumnHeads property to display a single row of column headings for list boxes, combo boxes, and OLE objects that accept column headings. You can also use this property to create a label for each entry in a chart control. What is actually displayed as the first-row column heading depends on the object's RowSourceType property setting. Read/write Boolean.

expression.ColumnHeads

expression  Required. An expression that returns one of the objects in the Applies To list.
Setting

The **ColumnHeads** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>True</td>
<td>Column headings are enabled and either field captions, field names, or the first row of data items are used as column headings or chart labels.</td>
</tr>
<tr>
<td>No</td>
<td>False (Default)</td>
<td>Column headings are not enabled.</td>
</tr>
</tbody>
</table>

You can set the **ColumnHeads** property by using the control's property sheet, a macro, or Visual Basic.

For **table fields**, you can set this property on the **Lookup** tab of the Field Properties section of **table Design view** for fields with the **DisplayControl** property set to Combo Box or List Box.

**Tip**

Microsoft Access sets the **ColumnHeads** property automatically when you select Lookup Wizard as the data type for a field in table Design view.
Remarks

The **RowSourceType** property specifies whether field names or the first row of data items are used to create column headings. If the **RowSourceType** property is set to Table/Query, the field names are used as column headings. If the field has a caption, then the caption is displayed. For example, if a list box has three columns (the **ColumnCount** property is set to 3) and the **RowSourceType** property is set to Table/Query, the first three field names (or captions) are used as headings.

If the **RowSourceType** property is set to Value List, the first row of data items entered in the value list (as the setting of the **RowSource** property) will be column headings. For example, if a list box has three columns and the **RowSourceType** property is set to Value List, the first three items in the **RowSource** property setting are used as column headings.

**Tip**

If you can't select the first row of a list box or combo box in Form view, check to see if the **ColumnHeads** property is set to Yes.

Headings in combo boxes appear only when displaying the list in the control.
**ColumnHidden Property**

You can use the `ColumnHidden` property to show or hide a specified column in Datasheet view. Read/write `Boolean`.

`expression COLUMNHIDDEN`

`expression` Required. An expression that returns one of the objects in the Applies To list.
Setting

You can set the **ColumnHidden** property by clicking **Hide Columns** or **Unhide Columns** on the **Format** menu in Datasheet view.

You can also set this property in a [Microsoft Access database](https://www.microsoft.com/access) (.mdb) by using a **Long Integer** value in [Visual Basic](https://visual-basic.net) to specify the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>The column is hidden.</td>
</tr>
<tr>
<td>False</td>
<td>(Default) The column is visible.</td>
</tr>
</tbody>
</table>

**Note** To set or change this property for a table or query by using Visual Basic, you must use a column's **Properties** collection. For details on using the **Properties** collection, see [Properties](https://msdn.microsoft.com/en-us/library/aa376928.aspx).

**Note** The **ColumnHidden** property is not available in [Design view](https://msdn.microsoft.com/en-us/library/aa376928.aspx).
Remarks

For example, you might want to hide a CustomerAddress field that's too wide so you can view the CustomerName and PhoneNumber fields.

**Note** The ColumnHidden property applies to all fields in Datasheet view and to form controls when the form is in Datasheet view.

Hiding a column with the ColumnHidden property in Datasheet view doesn't hide fields from the same column in Form view. Similarly, setting a control's Visible property to False in Form view doesn't hide the corresponding column in Datasheet view.

You can display a field in a query even though the column for the field is hidden in table Datasheet view.

You can use values from a hidden column as the criteria for a filter even though the column remains hidden after the filter is applied.

You can't use the Copy, Paste, Find, and Replace commands on the Edit menu to affect hidden columns.

Setting a field's ColumnWidth property to 0, or resizing the field to a zero width in Datasheet view, causes Microsoft Access to set the corresponding ColumnHidden property to True. Unhiding a column restores the ColumnWidth property to the value it had before the field was hidden.
Example

The following example hides the ProductID field in Datasheet view of the Products form.

Forms!Products!ProductID\.[ColumnHidden] = -1

The next example also hides the ProductID field in Datasheet view of the Products table.

Public Sub SetColumnHidden()
    Dim dbs As DAO.Database
    Dim fld As DAO.Field
    Dim prp As DAO.Property
    Const conErrPropertyNotFound = 3270

    ' Turn off error trapping.
    On Error Resume Next

    Set dbs = CurrentDb

    ' Set field property.
    Set fld = dbs.TableDefs!Products.Fields!ProductID
    fld.Properties("[ColumnHidden]") = True

    ' Error may have occurred when value was set.
    If Err.Number <> 0 Then
        If Err.Number <> conErrPropertyNotFound Then
            On Error GoTo 0
            MsgBox "Couldn't set property 'ColumnHidden' " & _
            "on field " & fld.Name & ", vbCritical"
        Else
            On Error GoTo 0
            Set prp = fld.CreateProperty("[ColumnHidden]", dbLong, True)
            fld.Properties.Append prp
        End If
    End If
    Set prp = Nothing
    Set fld = Nothing
    Set dbs = Nothing
End Sub
ColumnOrder Property

You can use the **ColumnOrder** property to specify the order of the columns in **Datasheet view**. Read/write **Integer**.

*expression*.ColumnOrder

*expression* Required. An expression that returns one of the objects in the Applies To list.
Setting

You can set this property by selecting a column in Datasheet view and dragging it to a new position.

You can also set this property in a Microsoft Access database (.mdb) by using a Long Integer value in Visual Basic.

To set or change this property for a table or query by using Visual Basic, you must use a column's Properties collection. For details on using the Properties collection, see Properties.

Note  The ColumnOrder property isn't available in Design view.
Remarks

**Note**  The **ColumnOrder** property applies to all fields in Datasheet view and to form controls when the form is in Datasheet view.

In Datasheet view, a field's **ColumnOrder** property setting is determined by the field's position. For example, the field in the leftmost column in Datasheet view has a **ColumnOrder** property setting of 1, the next field has a setting of 2, and so on. Changing a field's **ColumnOrder** property resets the property for that field and every field to the left of its original position in Datasheet view.

In other views, the property setting is 0 unless you explicitly change the order of one or more fields in Datasheet view (either by dragging the fields to new positions or by changing their **ColumnOrder** property settings). Fields to the right of the moved field's new position will have a property setting of 0 in views other than Datasheet view.

The order of the fields in Datasheet view doesn't affect the order of the fields in **table Design view** or **Form view**.
Example

The following example displays the ProductName and QuantityPerUnit fields in the first two columns in Datasheet view of the Products form.

Forms!Products!ProductName.**ColumnOrder** = 1
Forms!Products!QuantityPerUnit.**ColumnOrder** = 2

The next example displays the ProductName and QuantityPerUnit fields in the first two columns of the Products table in Datasheet view. To set the **ColumnOrder** property, the example uses the SetFieldProperty procedure. If this procedure is run while the table is open, changes will not be displayed until it is closed and reopened.

Public Sub SetColumnOrder()
    Dim dbs As DAO.Database
    Dim tdf As DAO.TableDef

    Set dbs = CurrentDb
    Set tdf = dbs!Products

    ' Call the procedure to set the ColumnOrder property.
    SetFieldProperty tdf!ProductName, "ColumnOrder", dbLong, 2
    SetFieldProperty tdf!QuantityPerUnit, "ColumnOrder", dbLong, 3

    Set tdf = Nothing
    Set dbs = Nothing
End Sub

Private Sub SetFieldProperty(ByRef fld As DAO.Field, _
                               ByVal strPropertyName As String, _
                               ByVal intPropertyType As Integer, _
                               ByVal varPropertyValue As Variant)
    ' Set field property without producing nonrecoverable run-time e
    Const conErrPropertyNotFound = 3270
    Dim prp As Property

    ' Turn off error handling.
    On Error Resume Next

    fld.Properties(strPropertyName) = varPropertyValue
End Sub
' Check for errors in setting the property.
    If Err <> 0 Then
        If Err <> conErrPropertyNotFound Then
            On Error GoTo 0
            MsgBox "Couldn't set property "" & strPropertyName & "" on field "" & fld.Name & "", vbCritical
        Else
            On Error GoTo 0
            Set prp = fld.CreateProperty(strPropertyName, intPropertyType,
                                         varPropertyValue)
            fld.Properties.Append prp
        End If
    End If

    Set prp = Nothing

End Sub
ColumnSpacing Property

Returns or sets a Long representing the vertical space between detail sections in twips. Read/write.

expression.ColumnSpacing

expression  Required. An expression that returns one of the objects in the Applies To list.
Example

The following example sets a variety of printer settings for the form specified in the `strFormname` argument of the procedure.

Sub SetPrinter(strFormname As String)

    DoCmd.OpenForm FormName:=strFormname, view:=acDesign, _
    datamode:=acFormEdit, windowmode:=acHidden

    With Forms(form1).Printer

        TopMargin = 1440
        BottomMargin = 1440
        LeftMargin = 1440
        RightMargin = 1440

        ColumnSpacing = 360
        RowSpacing = 360

        ColorMode = acPRCMColor
        DataOnly = False
        DefaultSize = False
        ItemSizeHeight = 2880
        ItemSizeWidth = 2880
        ItemLayout = acPRVerticalColumnLayout
        ItemsAcross = 6

        Copies = 1
        Orientation = acPRORLandscape
        Duplex = acPRDPVertical
        PaperBin = acPRBNAuto
        PaperSize = acPRPSLetter
        PrintQuality = acPRPQMedium

    End With

    DoCmd.Close objecttype:=acForm, objectname:=strFormname, _
    Save:=acSaveYes

End Sub
ColumnWidth Property

You can use the ColumnWidth property to specify the width of a column in Datasheet view. Read/write Integer.

expression.ColumnWidth

expression Required. An expression that returns one of the objects in the Applies To list.


**Setting**

You can set this property by dragging the right border of the column selector or by clicking **Column Width** on the **Format** menu in Datasheet view and entering the desired value. When you set the **ColumnWidth** property by using the **ColumnWidth** command, the value is expressed in **points**.

In **Visual Basic**, the **ColumnWidth** property setting is an **Integer** value that represents the column width in **twips**. You can specify a width or use one of the following predefined settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Hides the column.</td>
</tr>
<tr>
<td>–1</td>
<td>(Default) Sizes the column to the default width.</td>
</tr>
</tbody>
</table>
Remarks

Note  The **ColumnWidth** property applies to all fields in Datasheet view and to form controls when the form is in Datasheet view.

Setting this property to 0, or resizing the field to a zero width in Datasheet view, sets the field's **ColumnHidden** property to **True** (–1) and hides the field in Datasheet view.

Setting a field's **ColumnHidden** property to **False** (0) restores the field's **ColumnWidth** property to the value it had before the field was hidden. For example, if the **ColumnWidth** property was –1 prior to the field being hidden by setting the property to 0, changing the field's **ColumnHidden** property to **False** resets the **ColumnWidth** to –1.

The **ColumnWidth** property for a field isn't available when the field's **ColumnHidden** property is set to **True**.
Example

This example takes effect in Datasheet view of the open Customers form. It sets the row height to 450 twips and sizes the column to fit the size of the visible text.

Forms![Customers].RowHeight = 450
Forms![Customers]![Address].ColumnWidth = -2
ColumnWidths Property

You can use the `ColumnWidths` property to specify the width of each column in a multiple-column combo box or list box. Read/write String.

expression. ColumnWidths

expression Required. An expression that returns one of the objects in the Applies To list.
Setting

The **ColumnWidths** property holds a value specifying the width of each column in inches or centimeters, depending on the measurement system (U.S. or Metric) selected in the **Measurement system** box on the **Number** tab of the **Regional Options** dialog box of Windows Control Panel. The default setting is 1 inch or 2.54 centimeters. The **ColumnWidths** property setting must be a value from 0 to 22 inches (55.87 cm) for each column in the list box or combo box.

To separate your column entries, use semicolons (;) as list separators (or the list separator selected in the **List separator** box on the **Number** tab of the **Regional Options** dialog box).

A width of 0 hides a column. Any or all of the **ColumnWidths** property settings can be blank. You create a blank setting by typing a list separator without a preceding value. Blank values result in Microsoft Access automatically setting a default column width that varies depending on the number of columns and the width of the combo box or list box.

**Note** In a combo box, the first visible column is displayed in the text box portion of the control.

You can set the **ColumnWidths** property by using the control's **property sheet**, a **macro**, or **Visual Basic**.

For **table fields**, you can set this property on the **Lookup** tab of the Field Properties section of **table Design view** for fields with the **DisplayControl** property set to Combo Box or List Box.

**Tip**

Microsoft Access sets the **ColumnWidths** property automatically when you select Lookup Wizard as the data type for a field in table Design view.

In Visual Basic, use a **string expression** to set the column width values in **twips**. Column widths are separated by semicolons. To specify a different unit of measurement, include the unit of measure (cm or in). For example, the following string expression specifies three column widths in centimeters.
6 cm; 6 cm
Remarks

You can also use this property to hide one or more columns.

If you leave the ColumnWidths property setting blank, Microsoft Access sets the width of each column as the overall width of the list box or combo box divided by the number of columns.

If the column widths you set are too wide to be fully displayed within the combo box or list box, the rightmost columns are hidden and a horizontal scroll bar appears.

If you specify the width for some columns but leave the setting for others blank, Microsoft Access divides the remaining width by the number of columns for which you haven't specified a width. The minimum calculated column width is 1,440 twips (1 inch).

For example, the following settings are applied to a 4-inch list box with three columns.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.5 in;0;2.5 in</td>
<td>The first column is 1.5 inches, the second column is hidden, and the third column is 2.5 inches.</td>
</tr>
<tr>
<td>2 in;:2 in</td>
<td>The first column is 2 inches, the second column is 1 inch (default), and the third column is 2 inches. Because only half of the third column is visible, a horizontal scroll bar appears.</td>
</tr>
<tr>
<td>(Blank)</td>
<td>The three columns are the same width (1.33 inches).</td>
</tr>
</tbody>
</table>

Note  This property is different than the ColumnWidth property, which specifies the width of a specified column in a datasheet.
CommandBars Property

You can use the CommandBars property to return a reference to the CommandBars collection object. Read-only CommandBars object.

expression.CommandBars

expression Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **CommandBars** property is available only by using **Visual Basic**.

The **CommandBars** collection object is the collection of all built-in and custom command bars in an application. You can refer to individual members of the collection by using the member object's index or a string expression that is the name of the member object. The first member object in the collection has an index value of 1 and the total number of member objects in the collection is the value of the **CommandBars** collection's **Count** property.

Once you establish a reference to the **CommandBars** collection object, you can access all the properties and methods of the object. You can set a reference to the **CommandBars** collection object by clicking **References** on the **Tools** menu while in module **Design view**. Set a reference to the Microsoft Office 11.0 Object Library in the **References** dialog box by selecting the appropriate check box.
CommandBeforeExecute Property

Returns or sets a String indicating which macro, event procedure, or user-defined function runs when the CommandBeforeExecute event occurs. Read/write.

expression.CommandBeforeExecute

expression  Required. An expression that returns one of the objects in the Applies To list.
Remarks

Valid values for this property are "macroname" where macroname is the name of macro, "[Event Procedure]" which indicates the event procedure associated with the CommandBeforeExecute event for the specified object, or "=functionname()" where functionname is the name of a user-defined function. For a more detailed discussion of event properties, see "Event Properties."
Example

The following example specifies that when the CommandBeforeExecute event occurs on the first form of the current project, the associated event procedure should run.

```
Forms(0).CommandBeforeExecute = "[Event Procedure]"
```
CommandChecked Property

Returns or sets a String indicating which macro, event procedure, or user-defined function runs when the CommandChecked event occurs. Read/write.

expression.CommandChecked

expression  Required. An expression that returns one of the objects in the Applies To list.
Remarks

Valid values for this property are "macroname" where macroname is the name of macro, "[Event Procedure]" which indicates the event procedure associated with the CommandChecked event for the specified object, or "=functionname()" where functionname is the name of a user-defined function. For a more detailed discussion of event properties, see "Event Properties."
Example

The following example specifies that when the CommandChecked event occurs on the first form of the current project, the associated event procedure should run.

```
Forms(0).CommandChecked = "[Event Procedure]"
```
CommandEnabled Property

Returns or sets a **String** indicating which macro, event procedure, or user-defined function runs when the CommandEnabled event occurs. Read/write.

expression.CommandEnabled

*expression* Required. An expression that returns one of the objects in the Applies To list.
Remarks

Valid values for this property are "macroname" where macroname is the name of macro, "[Event Procedure]" which indicates the event procedure associated with the CommandEnabled event for the specified object, or "=functionname()" where functionname is the name of a user-defined function. For a more detailed discussion of event properties, see "Event Properties."
**Example**

The following example specifies that when the CommandEnabled event occurs on the first form of the current project, the associated event procedure should run.

```
Forms(0).CommandEnabled = "[Event Procedure]"
```
CommandExecute Property

Returns or sets a String indicating which macro, event procedure, or user-defined function runs when the CommandExecute event occurs. Read/write.

expression.CommandExecute

expression Required. An expression that returns one of the objects in the Applies To list.
Remarks

Valid values for this property are "macroname" where macroname is the name of macro, "[Event Procedure]" which indicates the event procedure associated with the CommandExecute event for the specified object, or "=functionname()" where functionname is the name of a user-defined function. For a more detailed discussion of event properties, see "Event Properties."
Example

The following example specifies that when the CommandExecute event occurs on the first form of the current project, the associated event procedure should run.

Forms(0).CommandExecute = "[Event Procedure]"
Connection Property

You can use the Connection property to return a reference to the current ActiveX Data Objects (ADO) Connection object and its related properties. Read-only Connection.

expression.Connection

expression  Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **Connection** property is available only by using **Visual Basic** and is read-only.

Use the **Connection** property of the **CurrentProject** object to refer to the **Connection** object of the current **Microsoft Access project** (.adp) or **Access database** (.mdb) object. Use the **Connection** property of the **CodeProject** object to refer to the **Connection** object of the Access project or Access database code database object. You can use the **Connection** property to call methods on the **Connection** object such as **BeginTrans** and **CommitTrans**.

Notes

The **Connection** property actually returns a reference to a copy of the ActiveX Data Object (ADO) connection for the active database. Thus, applying the **Close** method or in anyway attempting to alter the connection through the **Connection** object’s methods or properties will have no affect on the actual connection object used by Microsoft Access to hold a live connection to the current database. Since the **Connection** property is the main Shape provider connection, the following information is necessary when using this property.

1. **MSDataShape** uses **Recordset.CursorLocation** = **adUseClient**. Do not set **CursorLocation** prior to assigning a recordset to **CurrentProject.Connect**.

2. **MSDataShape** uses **Recordset.CursorType** = **adOpenStatic**. Do not set **CursorType** prior to assigning a recordset to **CurrentProject.Connection**.

3. **MSDataShape** accepts **Recordset.LockType** = **adLockOptimistic**, **adLockBatchOptimistic**, or **adLockReadOnly** (default). If set to **adLockPessimistic**, it is changed to **adLockOptimistic**.

4. The shape connection does not support the all ADOX operation, specifically the **Columns.Properties** collection is not supported.

5. In order to ensure that a shape connection will work correctly, the **Command.CommandType** must be set to **adCmdTable**.
Show All
ConnectionString Property

You can use the **ConnectionString** property returns the base connection string for the **DataAccessPage** object. Read/write **String**.

`expression.Connectionstring`

`expression` Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **ConnectionString** property is available only by using **Visual Basic** and is read-only.

The **ConnectionString** property returns the connection string that was set through the **OpenConnection** method or by clicking **Connection** on the **File** menu. When making a connection, Microsoft Access project modifies the **ConnectionString** property for use with the ADO environment. The Microsoft Office **Data Source Control** (MSODSC) likewise modifies the **ConnectionString**.
Example

The following example displays the **ConnectionString** property setting of the currently active data access page:

```vba
Dim objCurrent As Object
Set objCurrent = Application.DataAccessPages(0)
MsgBox "The current base connection is " & objCurrent.ConnectionString
```
ControlBox Property

Specifies whether a form has a Control menu in Form view and Datasheet view. Read/write Boolean.

equation \text{ControlBox}

equation Required. An expression that returns one of the objects in the Applies To list.
Setting

The **ControlBox** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td><strong>True</strong> (–1)</td>
<td>(Default) The form has a <strong>Control</strong> menu in Form view and Datasheet view.</td>
</tr>
<tr>
<td>No</td>
<td><strong>False</strong> (0)</td>
<td>The form doesn't have a <strong>Control</strong> menu in Form view and Datasheet view.</td>
</tr>
</tbody>
</table>

**Note** Setting the **ControlBox** property to No also removes the **Minimize**, **Maximize**, and **Close** buttons on a form.

You can set this property by using the form's **property sheet**, a **macro**, or **Visual Basic**.

It can only be set in **form Design view**.
Remarks

To display a **Control** menu on a form, the **ControlBox** property must be set to Yes and the form's **BorderStyle** property must be set to Thin, Sizable, or Dialog.

Even when a form's **ControlBox** property is set to No, the form always has a **Control** menu when opened in Design view.

Setting the **ControlBox** property to No suppresses the **Control** menu when you:

- Open the form in Form view from the Database window.
- Open the form from a macro.
- Open the form from Visual Basic.
- Open the form in Datasheet view.
- Switch to Form or Datasheet view from Design view.
Example

The following example sets the `ControlBox` property on the `WarningDialog` form to **False** (0):

```vba
Forms!WarningDialog.ControlBox = False
```
Controls Property

Returns the Controls collection of a form, subform, report or section.

expression.Controls

expression  Required. An expression that returns one of the objects in the Applies To list.
Remarks

Use the **Controls** property to refer to one of the controls on a form, subform, report, or section within or attached to another control. For example, the first code syntax below returns the number of controls located on Form1. The second references the name of a property within a control.

```
Forms("Form1").Controls.Count
Forms("Form1").Controls("Textbox1").Properties(5).Name
```

The **Controls** property is available only by using [Visual Basic](https://visualbasic.net).

---
**ControlSource Property**

You can use the *ControlSource* property to specify what data appears in a *control*. You can display and edit data *bound* to a field in a *table*, *query*, or *SQL statement*. You can also display the result of an *expression*. Read/write *String*.

*expression*.ControlSource

*expression*  Required. An expression that returns one of the objects in the Applies To list.
Setting

The **ControlSource** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>A field name</strong></td>
<td>The control is bound to a field in a table, query, or SQL statement. Data from the field is displayed in the control. Changes to the data inside the control change the corresponding data in the field. (To make the control read-only, set the <strong>Locked</strong> property to Yes.) If you click a control bound to a field that has a <a href="#">Hyperlink data type</a>, you jump to the destination specified in the hyperlink address.</td>
</tr>
<tr>
<td><strong>An expression</strong></td>
<td>The control displays data generated by an expression. This data can be changed by the user but isn't saved in the database.</td>
</tr>
</tbody>
</table>

You can set the **ControlSource** property for a control by using the control's [property sheet](#), a [macro](#), or [Visual Basic](#).

You can also set the **ControlSource** property for a [text box](#) by typing a field name or expression directly in the text box in form Design view or report Design view.

For a report, you can set this property by selecting a field or typing an expression in the **Field/Expression** column of the [Sorting And Grouping](#) box. For details, see the **GroupLevel** property.

In Visual Basic, use a [string expression](#) to set the value of this property.
Remarks

For a report group level, the ControlSource property determines the field or expression to group on.

Note  The ControlSource property doesn't apply to check box, option button, or toggle button controls in an option group. It applies only to the option group itself.

For reports, the ControlSource property applies only to report group levels.

Forms and reports act as "windows" into your database. You specify the primary source of data for a form or report by setting its RecordSource property to a table, query, or SQL statement. You can then set the ControlSource property to a field in the source of data or to an expression. If the ControlSource property setting is an expression, the value displayed is read-only and not saved in the database. For example, you can use the following settings.

<table>
<thead>
<tr>
<th>Sample setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LastName</td>
<td>For a control, data from the LastName field is displayed in the control. For a report group level, Microsoft Access groups the data on last name.</td>
</tr>
<tr>
<td>=Date( ) + 7</td>
<td>For a control, this expression displays a date seven days from today in the control.</td>
</tr>
<tr>
<td>=DatePart(&quot;q&quot;,ShippedDate)</td>
<td>For a control, this expression displays the quarter of the shipped date. For a report group level, Microsoft Access groups the data on the quarter of the shipped date.</td>
</tr>
</tbody>
</table>
Example

The following example sets the `ControlSource` property for a text box named AddressPart to a field named City:

```
Forms!Customers!AddressPart.ControlSource = "City"
```

The next example sets the `ControlSource` property for a text box named Expected to the expression `=Date() + 7`.

```
Me!Expected.ControlSource = "=Date() + 7"
```
ControlTipText Property

You can use the ControlTipText property to specify the text that appears in a ScreenTip when you hold the mouse pointer over a control. Read/write String.

expression.ControlTipText

expression Required. An expression that returns one of the objects in the Applies To list.
Setting

You set the `ControlTipText` property by using a `string expression` up to 255 characters long.

You can set the `ControlTipText` property by using the control’s `property sheet`, a `macro`, or `Visual Basic`.

For controls on forms, you can set the default for this property by using the `default control style` or the `DefaultControl` method in Visual Basic.

You can set the `ControlTipText` property in any view.
Remarks

The **ControlTipText** property provides an easy way to provide helpful information about controls on a form.

There are other ways to provide information about a form or a control on a form. You can use the **StatusBarText** property to display information in the status bar about a control. To provide more extensive help for a form or control, use the **HelpFile** and **HelpContextID** properties.
ControlType Property

You can use the ControlType property in Visual Basic to determine the type of a control on a form or report. Read/write Byte.

expression.ControlType

expression Required. An expression that returns one of the objects in the Applies To list.
Setting

The **ControlType** property setting is an *intrinsic constant* that specifies the control type.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>acBoundObjectFrame</td>
<td>Bound object frame</td>
</tr>
<tr>
<td>acCheckBox</td>
<td>Check box</td>
</tr>
<tr>
<td>acComboBox</td>
<td>Combo box</td>
</tr>
<tr>
<td>acCommandButton</td>
<td>Command button</td>
</tr>
<tr>
<td>acCustomControl</td>
<td>ActiveX (custom) control</td>
</tr>
<tr>
<td>acImage</td>
<td>Image</td>
</tr>
<tr>
<td>acLabel</td>
<td>Label</td>
</tr>
<tr>
<td>acLine</td>
<td>Line</td>
</tr>
<tr>
<td>acListBox</td>
<td>List box</td>
</tr>
<tr>
<td>acObjectFrame</td>
<td>Unbound object frame or chart</td>
</tr>
<tr>
<td>acOptionButton</td>
<td>Option button</td>
</tr>
<tr>
<td>acOptionGroup</td>
<td>Option group</td>
</tr>
<tr>
<td>acPage</td>
<td>Page</td>
</tr>
<tr>
<td>acPageBreak</td>
<td>Page break</td>
</tr>
<tr>
<td>acRectangle</td>
<td>Rectangle</td>
</tr>
<tr>
<td>acSubform</td>
<td>Subform/subreport</td>
</tr>
<tr>
<td>acTabCtl</td>
<td>Tab</td>
</tr>
<tr>
<td>acTextBox</td>
<td>Text box</td>
</tr>
<tr>
<td>acToggleButton</td>
<td>Toggle button</td>
</tr>
</tbody>
</table>

**Note**  The **ControlType** property can only be set by using Visual Basic in form Design view or report Design view, but it can be read in all views.
Remarks

The **ControlType** property is useful not only for checking for a specific control type in code, but also for changing the type of control to another type. For example, you can change a text box to a combo box by setting the **ControlType** property for the text box to **acComboBox** while in form Design view.

You can use the **ControlType** property to change characteristics of similar controls on a form according to certain conditions. For example, if you don't want users to edit existing data in text boxes, you can set the **SpecialEffect** property for all text boxes to Flat and set the form's **AllowEdits** property to No. (The **SpecialEffect** property doesn't affect whether data can be edited; it's used here to provide a visual cue that the control behavior has changed.)

The **ControlType** property is also used to specify the type of control to create when you are using the **CreateControl** method.
Example

The following example examines the **ControlType** property for all controls on a form. For each label and text box control, the procedure toggles the **SpecialEffect** property for those controls. When the label controls' **SpecialEffect** property is set to Shadowed and the text box controls' **SpecialEffect** property is set to Normal and the **AllowAdditions**, **AllowDeletions**, and **AllowEdits** properties are all set to **True**, the `intCanEdit` variable is toggled to allow editing of the underlying data.

```vbscript
Sub ToggleControl(frm As Form)
    Dim ctl As Control
    Dim intI As Integer, intCanEdit As Integer
    Const conTransparent = 0
    Const conWhite = 16777215
    For Each ctl In frm.Controls
        With ctl
            Select Case .ControlType
                Case acLabel
                    If .SpecialEffect = acEffectShadow Then
                        .SpecialEffect = acEffectNormal
                        .BorderStyle = conTransparent
                        intCanEdit = True
                    Else
                        .SpecialEffect = acEffectShadow
                        intCanEdit = False
                    End If
                Case acTextBox
                    If .SpecialEffect = acEffectNormal Then
                        .SpecialEffect = acEffectSunken
                        .BackColor = conWhite
                    Else
                        .SpecialEffect = acEffectNormal
                    End If
            End Select
        End With
    Next ctl
    If intCanEdit = False Then
        With frm
            .AllowAdditions = False
            .AllowDeletions = False
            .AllowEdits = False
        End With
    Else
```
With frm
  .AllowAdditions = True
  .AllowDeletions = True
  .AllowEdits = True
End With
End If
End Sub
Copies Property

Returns or sets a `Long` indicating the number of copies to be printed. Read/write.

`expression.Copies`

`expression` Required. An expression that returns one of the objects in the Applies To list.
Example

The following example sets a variety of printer settings for the form specified in the **strFormname** argument of the procedure.

Sub SetPrinter(strFormname As String)
    DoCmd.OpenForm FormName:=strFormname, view:=acDesign, _
                    datamode:=acFormEdit, windowmode:=acHidden
    With Forms(form1).Printer
        .TopMargin = 1440
        .BottomMargin = 1440
        .LeftMargin = 1440
        .RightMargin = 1440

        .ColumnSpacing = 360
        .RowSpacing = 360

        .ColorMode = acPRCMColor
        .DataOnly = False
        .DefaultSize = False
        .ItemSizeHeight = 2880
        .ItemSizeWidth = 2880
        .ItemLayout = acPRVerticalColumnLayout
        .ItemsAcross = 6

        .Copies = 1
        .Orientation = acPRORLandscape
        .Duplex = acPRDPVertical
        .PaperBin = acPRBNAuto
        .PaperSize = acPRPSLetter
        .PrintQuality = acPRPQMedium
    End With
    DoCmd.Close objecttype:=acForm, objectname:=strFormname, _
                   Save:=acSaveYes
End Sub
Count Property

Count property as it applies to the Form and Report objects.

You can use the **Count** property to determine the number of items in a specified collection. Read/write **Integer**.

*expression*.Count

**expression**  Required. An expression that returns one of the above objects.

Count property as it applies to the all other object in the Applies To list.

You can use the **Count** property to determine the number of items in a specified collection. Read-only **Long**.

*expression*.Count

**expression**  Required. An expression that returns one of the above objects.
Setting

The **Count** property setting is an **Integer** value and is read-only in all views.

You can determine the **Count** property for an object by using a **macro** or **Visual Basic**.
**Remarks**

For example, if you want to determine the number of forms currently open or existing on the database, you would use the following code strings:

' Determine the number of open forms.
forms.count

' Determine the number of forms (open or closed)
' in the current database.
currentproject.allforms.count
Example

The following example uses the Count property to control a loop that prints information about all open forms and their controls.

Sub Print_Form_Controls()
    Dim frm As Form, intI As Integer
    Dim intJ As Integer
    Dim intControls As Integer, intForms As Integer
    intForms = Forms.Count ' Number of open forms.
    If intForms > 0 Then
        For intI = 0 To intForms - 1
            Set frm = Forms(intI)
            Debug.Print frm.Name
            intControls = frm.Count
            If intControls > 0 Then
                For intJ = 0 To intControls - 1
                    Debug.Print vbTab; frm(intJ).Name
                Next intJ
            Else
                Debug.Print vbTab; "(no controls)"
            End If
        Next intI
    Else
        MsgBox "No open forms.", vbExclamation, "Form Controls"
    End If
End Sub

The next example determines the number of controls on a form and a report and assigns the number to a variable.

Dim intFormControls As Integer
Dim intReportControls As Integer
intFormControls = Forms!Employees.Count
intReportControls = Reports!FreightCharges.Count
CountOfDeclarationLines Property

The `CountOfDeclarationLines` property returns a `Long` value indicating the number of lines of code in the `Declarations section` in a `standard module` or `class module`. Read-only `Long`.

`expression.CountOfDeclarationLines`

`expression` Required. An expression that returns one of the objects in the `Applies To` list.
Setting

The `CountOfDeclarationLines` property is available only by using Visual Basic and is read-only.
Remarks

Lines in a module are numbered beginning with 1.

The value of the `CountOfDeclarationLines` property is equal to the line number of the last line of the Declarations section. You can use this property to determine where the Declarations section ends and the body of the module begins.
Example

The following example counts the number of lines and declaration lines in each standard module in the *Modules* collection. Note that the *Modules* collection contains only modules that are open in the module editor.

Public Sub ModuleLineTotal(ByVal strModuleName As String)
    Dim mdl As Module

    ' Open module to include in Modules collection.
    DoCmd.OpenModule strModuleName

    ' Return reference to Module object.
    Set mdl = Modules(strModuleName)

    ' Print number of lines in module.
    Debug.Print "Number of lines: ", mdl.CountOfLines

    ' Print number of declaration lines.
    Debug.Print "Number of declaration lines: ", _
        mdl.CountOfDeclarationLines

End Sub
CountOfLines Property

The CountOfLines property returns a Long value indicating the number of lines of code in a standard module or class module. Read-only Long.

expression.CountOfLines

expression  Required. An expression that returns one of the objects in the Applies To list.
Setting

The **CountOfLines** property is available only by using Visual Basic and is read-only.
Remarks

Lines in a module are numbered beginning with 1.  

The line number of the last line in a module is the value of the CountOfLines property.
Example

The following example counts the number of lines and declaration lines in each standard module in the Modules collection. Note that the Modules collection contains only modules that are open in the module editor.

Public Sub ModuleLineTotal(ByVal strModuleName As String)

    Dim mdl As Module

    ' Open module to include in Modules collection.
    DoCmd.OpenModule strModuleName

    ' Return reference to Module object.
    Set mdl = Modules(strModuleName)

    ' Print number of lines in module.
    Debug.Print "Number of lines: ", mdl.CountOfLines

    ' Print number of declaration lines.
    Debug.Print "Number of declaration lines: ", _
    mdl.CountOfDeclarationLines

End Sub
CurrentData Property

You can use the CurrentData property to access the CurrentData object and its related collections. Read-only CurrentData object.

expression.CurrentData

expression  Required. An expression that returns one of the objects in the Applies To list.
Remarks

This property is available only by using Visual Basic and is read-only.

Use the **CurrentData** property to refer to one of the following current database collections together with one of its properties or methods.

- **AllTables**
- **AllQueries**
- **AllViews**
- **AllStoredProcedures**
- **AllDatabaseDiagrams**
**CurrentObjectName Property**

You can use the **CurrentObjectName** property with the **Application** object to determine the name of the active database object. The active database object is the object that has the **focus** or in which code is running. Read-only **String**.

```
expression.CurrentObjectName
```

**expression**  Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **CurrentObjectName** property is set by Microsoft Access to a string expression containing the name of the active object.

This property is available only by using **Visual Basic**.

The following conditions determine which object is considered the active object:

- If the active object is a property sheet, command bar, menu, palette, or field list of an object, the **CurrentObjectName** property returns the name of the underlying object.
- If the active object is a pop-up form, the **CurrentObjectName** property refers to the pop-up form itself, not the form from which it was opened.
- If the active object is the Database window, the **CurrentObjectName** property returns the item selected in the Database window.
- If no object is selected, the **CurrentObjectName** property returns a zero-length string (" ").
- If the current state is ambiguous (the active object isn't a table, query, form, report, macro, or module) for example, if a dialog box has the focus the **CurrentObjectName** property returns the dialog box name.

You can use this property with the **SysCmd** method to determine the active object and its state (for example, if the object is open, new, or has been changed but not saved).
Example

The following example uses the CurrentObjectType and CurrentObjectName properties with the SysCmd function to determine if the active object is the Products form and if this form is open and has been changed but not saved. If these conditions are true, the form is saved and then closed.

Public Sub CheckProducts()

    Dim intState As Integer
    Dim intCurrentType As Integer
    Dim strCurrentName As String

    intCurrentType = Application.CurrentObjectType
    strCurrentName = Application.CurrentObjectName

    If intCurrentType = acForm And strCurrentName = "Products" Then
        intState = SysCmd(acSysCmdGetObjectState, intCurrentType, _
            strCurrentName)

        ' Products form changed but not saved.
        If intState = acObjStateDirty + acObjStateOpen Then
            ' Close Products form and save changes.
            DoCmd.Close intCurrentType, strCurrentName, acSaveYes
        End If
    End If
End Sub
CurrentObjectType Property

You can use the CurrentObjectType property together with the Application object to determine the type of the active database object (table, query, form, report, macro, module, data access page, server view, database diagram, or stored procedure). The active database object is the object that has the focus or in which code is running.

The CurrentObjectType property is set by Microsoft Access to one of the following Microsoft Access intrinsic constants.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>acTable (0)</td>
<td>The active object is a table.</td>
</tr>
<tr>
<td>acQuery (1)</td>
<td>The active object is a query.</td>
</tr>
<tr>
<td>acForm (2)</td>
<td>The active object is a form.</td>
</tr>
<tr>
<td>acReport (3)</td>
<td>The active object is a report.</td>
</tr>
<tr>
<td>acMacro (4)</td>
<td>The active object is a macro.</td>
</tr>
<tr>
<td>acModule (5)</td>
<td>The active object is a module.</td>
</tr>
<tr>
<td>acDataAccessPage (6)</td>
<td>The active object is a data access page.</td>
</tr>
<tr>
<td>acServerView (7)</td>
<td>The active object is a server view.</td>
</tr>
<tr>
<td>acDiagram (8)</td>
<td>The active object is a database diagram.</td>
</tr>
<tr>
<td>acStoredProcedure (9)</td>
<td>The active object is a stored procedure.</td>
</tr>
</tbody>
</table>

The following conditions determine which object is considered the active object:

- If the active object is a property sheet, command bar, menu, palette, or field list of an object, the CurrentObjectType property returns the type of the underlying object.

- If the active object is a pop-up form, the CurrentObjectType property refers to the pop-up form itself, not the form from which it was opened.

- If the active object is the Database window, the CurrentObjectType property returns the item selected in the Database window.
- If no object is selected, the **CurrentObjectType** property returns **True**.

- If the current state is ambiguous (the active object isn't a table, query, form, report, macro, or module) for example, if a dialog box has the focus the **CurrentObjectType** property returns **True**.

You can use this property with the **SysCmd** method to determine the active object and its state (for example, if the object is open, new, or has been changed but not saved).
Example

The following example uses the `CurrentObjectType` and `CurrentObjectName` properties with the `SysCmd` function to determine if the active object is the Products form and if this form is open and has been changed but not saved. If these conditions are true, the form is saved and then closed.

```vba
Public Sub CheckProducts()
    Dim intState As Integer
    Dim intCurrentType As Integer
    Dim strCurrentName As String

    intCurrentType = Application.CurrentObjectType
    strCurrentName = Application.CurrentObjectName

    If intCurrentType = acForm And strCurrentName = "Products" Then
        intState = SysCmd(acSysCmdGetObjectState, intCurrentType, _
                          strCurrentName)

        ' Products form changed but not saved.
        If intState = acObjStateDirty + acObjStateOpen Then
            ' Close Products form and save changes.
            DoCmd.Close intCurrentType, strCurrentName, acSaveYes
        End If
    End If
End Sub
```
**CurrentProject Property**

You can use the `CurrentProject` property to access the `CurrentProject` object and its related collections, properties, and methods. Read-only `CurrentProject` object.

*expression*.`CurrentProject`

*expression* Required. An expression that returns one of the objects in the Applies To list.
Setting

The **CurrentProject** property setting contains a reference to the **CurrentProject** for the current code database.

This property is available only by using [Visual Basic](https://msdn.microsoft.com/en-us/library/178k444w.aspx) and is read-only.
Remarks

Use the CurrentProject property to refer to one of the following current database collections together with one of its properties or methods.

| AllForms   | AllReports |
| AllDataAccessPages | AllMacros |
| AllModules    | AccessObjectProperties |
CurrentRecord Property

You can use the CurrentRecord property to identify the current record in the recordset being viewed on a form. Read/write Long.

expression.CurrentRecord

expression Required. An expression that returns one of the objects in the Applies To list.
Setting

Microsoft Access sets this property to a Long Integer value that represents the current record number displayed on a form.

The CurrentRecord property is read-only in Form view and Datasheet view. It's not available in Design view. This property is available only by using a macro or Visual Basic.
Remarks

The value specified by this property corresponds to the value shown in the record number box found in the lower-left corner of the form.
Example

The following example shows how to use the `CurrentRecord` property to determine the record number of the current record being displayed. The general-purpose `CurrentFormRecord` procedure assigns the value of the current record to the variable `lngrecordnum`.

```vbnet
Sub CurrentFormRecord(frm As Form)
    Dim lngrecordnum As Long

    lngrecordnum = frm.CurrentRecord
End Sub
```
CurrentSectionLeft Property

You can use this property to determine the distance in twips from the left side of
the current section to the left side of the form. Read/write Integer.

expression.CurrentSectionLeft

description Required. An expression that returns one of the objects in the
Applies To list.
Remarks

This property setting is available only by using a macro or Visual Basic.

The CurrentSectionLeft property setting changes whenever a user scrolls through a form.

For forms whose DefaultView property is set to Single Form, if the user scrolls to the right of the left edge of the form, the property setting is a negative value.

The CurrentSectionLeft property is useful for finding the positions of detail sections displayed in Form view as continuous forms or in Datasheet view.
The following example displays the **CurrentSectionLeft** and **CurrentSectionTop** property settings for a control on a continuous form. Whenever the user moves to a new record, the property settings for the current section are displayed in the `lblStatus` label in the form's header.

```vbnet
Private Sub Form_Current()

    Dim intCurTop As Integer
    Dim intCurLeft As Integer

    intCurTop = Me.CurrentSectionTop
    intCurLeft = Me.CurrentSectionLeft
    Me!lblStatus.Caption = intCurLeft & ", " & intCurTop

End Sub
```
CurrentSectionTop Property

You can use this property to determine the distance in twips from the top edge of the current section to the top edge of the form.

expression.CurrentSectionTop

expression Required. An expression that returns one of the objects in the Applies To list.
**Remarks**

This property setting is available only by using a macro or Visual Basic.

The **CurrentSectionTop** property setting changes whenever a user scrolls through a form.

For forms whose **DefaultView** property is set to Single Form, if the user scrolls above the upper-left corner of the section, the property settings are negative values.

For forms whose **DefaultView** property is set to Continuous Forms, if a section isn't visible, the **CurrentSectionTop** property is equal to the **InsideHeight** property of the form.

The **CurrentSectionTop** property is useful for finding the positions of detail sections displayed in Form view as continuous forms or in Datasheet view. Each detail section has a different **CurrentSectionTop** property setting, depending on the section's position on the form.
Example

The following example displays the `CurrentSectionLeft` and `CurrentSectionTop` property settings for a control on a continuous form. Whenever the user moves to a new record, the property settings for the current section are displayed in the `lblStatus` label in the form's header.

Private Sub Form_Current()

    Dim intCurTop As Integer
    Dim intCurLeft As Integer

    intCurTop = Me.CurrentSectionTop
    intCurLeft = Me.CurrentSectionLeft
    Me!lblStatus.Caption = intCurLeft & " , " & intCurTop

End Sub
CurrentView Property

CurrentView property as it applies to the **AccessObject** object.

Returns the current view for the specified Microsoft Access object. Read-only **AcCurrentView**.

AcCurrentView can be one of these AcCurrentView constants.

acCurViewDatasheet  
acCurViewDesign  
acCurViewFormBrowse  
acCurViewPivotChart  
acCurViewPivotTable  
acCurViewPreview

*expression*.CurrentView

*expression* Required. An expression that returns an **AccessObject** object.

CurrentView property as it applies to the **DataAccessPage** object.

You can use the **CurrentView** property to determine how a data access page is currently displayed. Read-only **Integer**.

*expression*.CurrentView

*expression* Required. An expression that returns a **DataAccessPage** object.

CurrentView property as it applies to the **Form** object.

You can use the **CurrentView** property to determine how a **form** is currently displayed. Read/write **Integer**.

*expression*.CurrentView
expression  Required. An expression that returns a **Form** object.
Settings

The **CurrentView** property uses the following settings.

**Setting Form Displayed In: Data Access Page Displayed In:**

<table>
<thead>
<tr>
<th>Setting</th>
<th>Form Displayed In</th>
<th>Data Access Page Displayed In</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td><strong>Design view</strong></td>
<td><strong>Design view</strong></td>
</tr>
<tr>
<td>1</td>
<td>Form view</td>
<td><strong>Page view</strong></td>
</tr>
<tr>
<td>2</td>
<td>Datasheet view</td>
<td>Not applicable</td>
</tr>
</tbody>
</table>

**Note**  This property is available only by using a [macro](https://developer.microsoft.com/en-us/visualstudio/visual-basic) or [Visual Basic](https://developer.microsoft.com/en-us/visual-basic) and is read-only in all views.
Remarks

Use this property to perform different tasks depending on the current view. For example, an event procedure could determine which view the form is displayed in and perform one task if the form is displayed in Form view or another task if it's displayed in Datasheet view.
Example

The following example uses the GetCurrentView subroutine to determine whether a form is in Form or Datasheet view. If it's in Form view, a message to the user is displayed in a text box on the form; if it's in Datasheet view, the same message is displayed in a message box.

GetCurrentView Me, "Please contact system administrator."

Sub GetCurrentView(frm As Form, strDisplayMsg As String)
  Const conFormView = 1
  Const conDataSheet = 2
  Dim intView As Integer
  intView = frm.CurrentView
  Select Case intView
    Case conFormView
      frm!MessageTextBox.SetFocus
      ' Display message in text box.
      frm!MessageTextBox = strDisplayMsg
    Case conDataSheet
      ' Display message in message box.
      MsgBox strDisplayMsg
  End Select
End Sub
CurrentX Property

You can use the `CurrentX` property (along with the `CurrentY` property) to specify the horizontal and vertical coordinates for the starting position of the next printing and drawing method on a report.

`expression.CurrentX`

`expression` Required. An expression that returns one of the objects in the Applies To list.
Remarks

For example, you can use these properties to determine where the center point of a circle is drawn on a report section.

The CurrentX property specifies a Single value used in a numeric expression to set the horizontal coordinate of the next printing drawing method.

The coordinates are measured from the upper-left corner of the report section that contains the reference to the CurrentX or CurrentY property. The CurrentX property setting is 0 at the section's left edge, and the CurrentY property setting is 0 at its top edge.

You can set the CurrentX and CurrentY properties by using a macro or a Visual Basic event procedure specified by the OnPrint property setting of a report section.

Use the ScaleMode property to define the unit of measure, such as points, pixels, characters, inches, millimeters, or centimeters, that the coordinates will be based on.

When you use the following graphics methods, the CurrentX and CurrentY property settings are changed as indicated.

**Method** Sets CurrentX, CurrentY properties to

- **Circle** The center of the object.
- **Line** The end point of the line (the x2, y2 coordinates).
- **Print** The next print position.
Example

The following example uses the **Print** method to display text on a report named Report1. It uses the **TextWidth** and **TextHeight** methods to center the text vertically and horizontally.

```vba
Private Sub Detail_Format(Cancel As Integer, _
   FormatCount As Integer)
   Dim rpt as Report
   Dim strMessage As String
   Dim intHorSize As Integer, intVerSize As Integer

   Set rpt = Me
   strMessage = "DisplayMessage"
   With rpt
       ' Set scale to pixels, and set FontName and
       ' FontSize properties.
       .ScaleMode = 3
       .FontName = "Courier"
       .FontSize = 24
   End With
   ' Horizontal width.
   intHorSize = Rpt.TextWidth(strMessage)
   ' Vertical height.
   intVerSize = Rpt.TextHeight(strMessage)
   ' Calculate location of text to be displayed.
   Rpt.CurrentX = (Rpt.ScaleWidth/2) - (intHorSize/2)
   Rpt.CurrentY = (Rpt.ScaleHeight/2) - (intVerSize/2)
   ' Print text on Report object.
   Rpt.Print strMessage
End Sub
```
CurrentY Property

You can use the `CurrentY` property (along with the `CurrentX` property) to specify the horizontal and vertical coordinates for the starting position of the next printing and drawing method on a report.

`expression.CurrentY`

`expression` Required. An expression that returns one of the objects in the Applies To list.
Remarks

For example, you can use these properties to determine where the center point of a circle is drawn on a report section.

The CurrentY property specifies a Single value used in a numeric expression to set the vertical coordinate of the next printing drawing method.

The coordinates are measured from the upper-left corner of the report section that contains the reference to the CurrentX or CurrentY property. The CurrentX property setting is 0 at the section's left edge, and the CurrentY property setting is 0 at its top edge.

You can set the CurrentX and CurrentY properties by using a macro or a Visual Basic event procedure specified by the OnPrint property setting of a report section.

Use the ScaleMode property to define the unit of measure, such as points, pixels, characters, inches, millimeters, or centimeters, that the coordinates will be based on.

When you use the following graphics methods, the CurrentX and CurrentY property settings are changed as indicated.

**Method** | Sets CurrentX, CurrentY properties to
--- | ---
Circle | The center of the object.
Line | The end point of the line (the x2, y2 coordinates).
Print | The next print position.
Example

The following example uses the **Print** method to display text on a report named Report1. It uses the **TextWidth** and **TextHeight** methods to center the text vertically and horizontally.

```vbscript
Private Sub Detail_Format(Cancel As Integer, _
    FormatCount As Integer)
    Dim rpt As Report
    Dim strMessage As String
    Dim intHorSize As Integer, intVerSize As Integer

    Set rpt = Me
    strMessage = "DisplayMessage"
    With rpt
        'Set scale to pixels, and set FontName and
        'FontSize properties.
        .ScaleMode = 3
        .FontName = "Courier"
        .FontSize = 24
    End With
    ' Horizontal width.
    intHorSize = Rpt.TextWidth(strMessage)
    ' Vertical height.
    intVerSize = Rpt.TextHeight(strMessage)
    ' Calculate location of text to be displayed.
    Rpt.CurrentX = (Rpt.ScaleWidth/2) - (intHorSize/2)
    Rpt.CurrentY = (Rpt.ScaleHeight/2) - (intVerSize/2)
    ' Print text on Report object.
    Rpt.Print strMessage
End Sub
```
Custom Property

Returns or sets a **String** representing the custom properties dialog box for an **ActiveX control**. Read/write.

\[ expression..Custom \]

**expression** Required. An expression that returns one of the objects in the Applies To list.
Remarks

Not all ActiveX controls provide a custom properties dialog box. To see whether a control provides this custom properties dialog box, look for the Custom property in the Microsoft Access property sheet for this control. If the list of properties contains the name Custom, then the control provides the custom properties dialog box.

After you click the Custom property box in the Microsoft Access property sheet, click the Build button to the right of the property box to display the control’s custom properties dialog box, often presented as a tabbed dialog box. Choose the tab that contains the interface for setting the properties that you want to set.

After you make changes on one tab, you can often apply those changes immediately by clicking the Apply button (if provided). You can click other tabs to set other properties as needed. To approve all changes made in the custom properties dialog box, click the OK button. To return to the Microsoft Access property sheet without changing any property settings, click the Cancel button.

You can also view the custom properties dialog box by clicking the Properties subcommand of the ActiveX control Object command (for example, Calendar Control Object) on the Edit menu, or by clicking this same subcommand on the shortcut menu for the ActiveX control. In addition, some properties in the Microsoft Access property sheet for the ActiveX control, like the GridFontColor property of the Calendar control, have a Build button to the right of the property box. When you click the Build button, the custom properties dialog box is displayed, with the appropriate tab selected (for example, Colors).
Cycle Property

You can use the **Cycle** property to specify what happens when you press the TAB key and the **focus** is in the last **control** on a bound form. Read/write **Byte**.

`expression.Cycle`

*expression* Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **Cycle** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>All Records</td>
<td>0 (Default)</td>
<td>Pressing the TAB key from the last control on a form moves the focus to the first control in the <a href="https://example.com">tab order</a> in the next record.</td>
</tr>
<tr>
<td>Current Record</td>
<td>1</td>
<td>Pressing the TAB key from the last control on a record moves the focus to the first control in the tab order in the same record.</td>
</tr>
<tr>
<td>Current Page</td>
<td>2</td>
<td>Pressing the TAB key from the last control on a page moves the focus back to the first control in the tab order on the page.</td>
</tr>
</tbody>
</table>

**Note** You can set the **Cycle** property by using the form's [property sheet](https://example.com), a [macro](https://example.com), or [Visual Basic](https://example.com).

You can set the **Cycle** property in any view.

When you press the TAB key on a form, the focus moves through the controls on the form according to each control's place in the tab order.

You can set the **Cycle** property to All Records for forms designed for data entry. This allows the user to move to a new record by pressing the TAB key.

**Note** The **Cycle** property only controls the TAB key behavior on the form where the property is set. If a subform control is in the tab order, once the subform control receives the focus, the **Cycle** property setting for the subform determines what happens when you press the TAB key.

To move the focus outside a subform control, press CTRL+TAB.
DataAccessPages Property

You can use the **DataAccessPages** property to reference the read-only **DataAccessPages** collection and its related properties.

`expression.DataAccessPages`

`expression` Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **DataAccessPage** property is available only by using **Visual Basic**.
**DataChange Property**

Returns or sets a **String** indicating which macro, event procedure, or user-defined function runs when the **DataChange** event occurs. Read/write.

*expression*. **DataChange**

*expression* Required. An expression that returns one of the objects in the Applies To list.
Remarks

Valid values for this property are "macroname" where macroname is the name of macro, "[Event Procedure]" which indicates the event procedure associated with the DataChange event for the specified object, or "=functionname()" where functionname is the name of a user-defined function. For a more detailed discussion of event properties, see "Event Properties."
Example

The following example specifies that when the DataChange event occurs on the first form of the current project, the associated event procedure should run.

```
Forms(0).DataChange = "[Event Procedure]"
```
DataEntry Property

You can use the **DataEntry** property to specify whether a bound *form* opens to allow data entry only. The **Data Entry** property doesn't determine whether records can be added; it only determines whether existing records are displayed. Read/write **Boolean**.

```
expression.DataEntry
```

*expression* Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **DataEntry** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td><strong>True</strong></td>
<td>The form opens showing only a blank record.</td>
</tr>
<tr>
<td>No</td>
<td><strong>False</strong></td>
<td>(Default) The form opens showing existing records.</td>
</tr>
</tbody>
</table>

**Note** You can set the **DataEntry** property by using a form's property sheet, a macro, or Visual Basic.

This property can be set in any view.

The **DataEntry** property has an effect only when the **AllowAdditions** property is set to Yes.

Setting the **DataEntry** property to Yes by using Visual Basic has the same effect as clicking **Data Entry** on the Records menu. Setting it to No by using Visual Basic is equivalent to clicking **Remove Filter/Sort** on the Records menu.

**Note** When the Data Mode argument of the **OpenForm** action is set, Microsoft Access will override a number of form property settings. If the Data Mode argument of the OpenForm action is set to Edit, Microsoft Access will open the form with the following property settings:

- **AllowEdits**— Yes
- **AllowDeletions**— Yes
- **AllowAdditions**— Yes
- **DataEntry**— No

To prevent the OpenForm action from overriding any of these existing property settings, omit the Data Mode argument setting so that Microsoft Access will use the property settings defined by the form.
DataOnly Property

True if Microsoft Access prints only the data from a table or query in Datasheet View and not the labels, control borders, gridlines, and display graphics. Read/write Boolean.

expression.DataOnly

expression  Required. An expression that returns one of the objects in the Applies To list.
**Example**

The following example sets a variety of printer settings for the form specified in the `strFormname` argument of the procedure.

```vba
Sub SetPrinter(strFormname As String)
    DoCmd.OpenForm FormName:=strFormname, view:=acDesign, _
        datamode:=acFormEdit, windowmode:=acHidden
    With Forms(form1).Printer
        .TopMargin = 1440
        .BottomMargin = 1440
        .LeftMargin = 1440
        .RightMargin = 1440
        .ColumnSpacing = 360
        .RowSpacing = 360
        .ColorMode = acPRCMColor
        .DataOnly = False
        .DefaultSize = False
        .ItemSizeHeight = 2880
        .ItemSizeWidth = 2880
        .ItemLayout = acPRVerticalColumnLayout
        .ItemsAcross = 6
        .Copies = 1
        .Orientation = acPRORLandscape
        .Duplex = acPRDPVertical
        .PaperBin = acPRBNAuto
        .PaperSize = acPRPSLetter
        .PrintQuality = acPRPQMedium
    End With
    DoCmd.Close objecttype:=acForm, objectname:=strFormname, _
        Save:=acSaveYes
End Sub
```
DataSetChange Property

Returns or sets a String indicating which macro, event procedure, or user-defined function runs when the DataSetChange event occurs. Read/write.

expression.DataSetChange

expression  Required. An expression that returns one of the objects in the Applies To list.
Remarks

Valid values for this property are "macroname" where macroname is the name of macro, "[Event Procedure]" which indicates the event procedure associated with the DataSetChange event for the specified object, or "=functionname()" where functionname is the name of a user-defined function. For a more detailed discussion of event properties, see "Event Properties."
Example

The following example specifies that when the DataSetChange event occurs on the first form of the current project, the associated event procedure should run.

Forms(0).DataSetChange = "[Event Procedure]"
DatasheetBackColor Property

You can use the **DatasheetBackColor** property in Visual Basic to specify or determine the background color of an entire table, query, or form in Datasheet view within a Microsoft Access database (.mdb). Read/write Long.

\[expression.DatasheetBackColor\]

**expression** Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **DatasheetBackColor** property is a **Long Integer** value representing the background color and font color setting for a datasheet.

The following setting information applies to both Microsoft Access database and **Access projects** (.adp):

- You can also set this property by clicking **Fill/BackColor** on the **Formatting (Datasheet)** toolbar and clicking the desired color displayed on the color palette.

- You can also set the default **DatasheetBackColor** property by using the **Datasheet** tab of the **Options** dialog box, available by clicking **Options** on the **Tools** menu.

Setting the **DatasheetBackColor** property for a table or query won't affect this property setting for a form that uses the table or query as its source of data.

The following table contains the properties that don't exist in the DAO **Properties** collection of until you set them by using the **Formatting (Datasheet)** toolbar or you can add them in an Access database (.mdb) by using the **CreateProperty** method and append it to the DAO **Properties** collection.

<table>
<thead>
<tr>
<th>Property</th>
<th>Property</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DatasheetBackColor</strong></td>
<td><strong>DatasheetFontUnderline</strong></td>
</tr>
<tr>
<td><strong>DatasheetCellsEffect</strong></td>
<td><strong>DatasheetFontWeight</strong></td>
</tr>
<tr>
<td><strong>DatasheetFontHeight</strong></td>
<td><strong>DatasheetForeColor</strong></td>
</tr>
<tr>
<td><strong>DatasheetFontItalic</strong></td>
<td><strong>DatasheetGridlinesBehavior</strong></td>
</tr>
<tr>
<td><strong>DatasheetFontName</strong></td>
<td><strong>DatasheetGridlinesColor</strong></td>
</tr>
</tbody>
</table>

**Note**  When you add or set any property listed with an asterisk, Microsoft Access automatically adds it to the **Properties** collection.
**Example**

The following example uses the `SetTableProperty` procedure to set a table's font color to dark blue and its background color to light gray. If a "Property not found" error occurs when the property is set, the `CreateProperty` method is used to add the property to the object's `Properties` collection.

```vba
Dim dbs As Object, objProducts As Object
Const lngForeColor As Long = 8388608 ' Dark blue.
Const lngBackColor As Long = 12632256 ' Light gray.
Const DB_Long As Long = 4
Set dbs = CurrentDb
Set objProducts = dbs!Products
SetTableProperty objProducts, "DatasheetBackColor", DB_Long, lngBackColor
SetTableProperty objProducts, "DatasheetForeColor", DB_Long, lngForeColor

Sub SetTableProperty(objTableObj As Object, strPropertyName As String, intPropertyType As Integer, varPropertyValue As Variant)
    Const conErrPropertyNotFound = 3270
    Dim prpProperty As Variant
    On Error Resume Next ' Don't trap errors.
    objTableObj.Properties(strPropertyName) = varPropertyValue
    If Err <> 0 Then ' Error occurred when value
        If Err <> conErrPropertyNotFound Then
            ' Error is unknown.
            MsgBox "Couldn't set property " & strPropertyName & " on table " & objTableObj.Name & ", " & Err.Description, vbExclamation
            Err.Clear
        Else
            ' Error is "Property not found", so add it to collection
            Set prpProperty = objTableObj.CreateProperty(strPropertyName, intPropertyType, varPropertyValue)
            objTableObj.Properties.Append prpProperty
            Err.Clear
        End If
    End If
End Sub
```
DatasheetBorderStyle Property

Returns or sets a **Byte** indicating the line style to use for the border of the specified datasheet. Read/write.

*expression*.DatasheetBorderStyle

*expression* Required. An expression that returns one of the objects in the Applies To list.
Remarks

Valid values are between zero and eight. Values greater than eight are ignored; negative values or values above 255 cause an error.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Transparent border</td>
</tr>
<tr>
<td>1</td>
<td>Solid</td>
</tr>
<tr>
<td>2</td>
<td>Dashes</td>
</tr>
<tr>
<td>3</td>
<td>Short dashes</td>
</tr>
<tr>
<td>4</td>
<td>Dots</td>
</tr>
<tr>
<td>5</td>
<td>Sparse dots</td>
</tr>
<tr>
<td>6</td>
<td>Dash-dot</td>
</tr>
<tr>
<td>7</td>
<td>Dash-dot-dot</td>
</tr>
<tr>
<td>8</td>
<td>Double solid</td>
</tr>
</tbody>
</table>

This property is not supported when saving a form as a data access page.
Example

This example sets the datasheet border line style on the first open form to short dashes. The form must be set to Datasheet View in order for you to see the change.

Forms(0).DatasheetBorderStyle = 3
Show All
**DatasheetCellsEffect Property**

You can use the `DatasheetCellsEffect` property to specify whether special effects are applied to cells in a datasheet. Read/write Byte.

`expression.DatasheetCellsEffect`

`expression`  Required. An expression that returns one of the objects in the Applies To list.
Remarks

The DatasheetCellsEffect property applies only to objects in Datasheet view.

This property is only available in Visual Basic within a Microsoft Access database (.mdb).

The DatasheetCellsEffect property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flat</td>
<td>acEffectNormal</td>
<td>(Default) No special effects are applied to the cells in the datasheet.</td>
</tr>
<tr>
<td>Raised</td>
<td>acEffectRaised</td>
<td>Cells in the datasheet appear raised.</td>
</tr>
<tr>
<td>Sunken</td>
<td>acEffectSunken</td>
<td>Cells in the datasheet appear sunken.</td>
</tr>
</tbody>
</table>

Note You can set this property by using Special Effect button on the Formatting (Datasheet) toolbar, and in an Access database (.mdb), by using a macro, or by using Visual Basic.

The following setting information applies to both Access databases (.mdb) and Access projects (.adp):

You can also set this property by selecting the type of cell effect under Cell Effect in the Cells Effects dialog box, available by clicking Cells on the Format menu.

You can set the default DatasheetCellsEffect property by using the settings under Default Cell Effect on the Datasheet tab of the Options dialog box, available by clicking Options on the Tools menu.

This property applies the selected effect to the entire datasheet.

When this property is set to Raised or Sunken, gridlines will be visible on the datasheet regardless of the DatasheetGridlinesBehavior property setting.

The following table contains the properties that don't exist in the DAO Properties collection of until you set them by using the Formatting (Datasheet) toolbar or you can add them in an Access database (.mdb) by using
the **CreateProperty** method and append it to the DAO **Properties** collection.

<table>
<thead>
<tr>
<th>Property</th>
<th>Property</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DatasheetFontItalic</strong> *</td>
<td><strong>DatasheetForeColor</strong> *</td>
</tr>
<tr>
<td><strong>DatasheetFontHeight</strong> *</td>
<td><strong>DatasheetBackColor</strong></td>
</tr>
<tr>
<td><strong>DatasheetFontName</strong> *</td>
<td><strong>DatasheetGridlinesColor</strong></td>
</tr>
<tr>
<td><strong>DatasheetFontUnderline</strong> *</td>
<td><strong>DatasheetGridlinesBehavior</strong></td>
</tr>
<tr>
<td><strong>DatasheetFontWeight</strong> *</td>
<td><strong>DatasheetCellsEffect</strong></td>
</tr>
</tbody>
</table>

**Note**  When you add or set any property listed with an asterisk, Microsoft Access automatically adds all the properties listed with an asterisk to the **Properties** collection in the database.
DatasheetColumnHeaderUnderlineStyle Property

Returns or sets a Byte indicating the line style to use for the bottom edge of the column headers on the specified datasheet. Read/write.

expression.DatasheetColumnHeaderUnderlineStyle

expression    Required. An expression that returns one of the objects in the Applies To list.
Remarks

Valid values are between zero and eight. Values greater than eight are ignored; negative values or values above 255 cause an error.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Transparent border</td>
</tr>
<tr>
<td>1</td>
<td>Solid</td>
</tr>
<tr>
<td>2</td>
<td>Dashes</td>
</tr>
<tr>
<td>3</td>
<td>Short dashes</td>
</tr>
<tr>
<td>4</td>
<td>Dots</td>
</tr>
<tr>
<td>5</td>
<td>Sparse dots</td>
</tr>
<tr>
<td>6</td>
<td>Dash-dot</td>
</tr>
<tr>
<td>7</td>
<td>Dash-dot-dot</td>
</tr>
<tr>
<td>8</td>
<td>Double solid</td>
</tr>
</tbody>
</table>

This property is not supported when saving a form as a data access page.
**Example**

This example sets the column header underline style for the first open form to sparse dots. The form must be set to Datasheet View in order for you to see the change.

```
Forms(0).DatasheetColumnHeaderUnderlineStyle = 5
```
DatasheetFontHeight Property

You can use the **DatasheetFontHeight** property to specify the font point size used to display and print field names and data in **Datasheet view**. Read/write **Integer**.

*expression*.**DatasheetFontHeight**

*expression*  Required. An expression that returns one of the objects in the Applies To list.
Remarks

This property is only available in [Visual Basic](https://docs.microsoft.com/en-us/visualstudio/vbasic/visual-basic-overview?view=vs-2019) within a [Microsoft Access database](https://docs.microsoft.com/en-us/dynamics365/customer-engagement/access/additional-help) (.mdb)

You can set this property by selecting the font size from the **Font Size** box on the **Formatting (Datasheet)** toolbar.

You can also set this property with the **Size** boxes in the **Font** dialog box, available by clicking **Font** on the **Format** menu in Datasheet view.

For the **DatasheetFontHeight** property, the font size you specify must be valid for the font specified by the **DatasheetFontName** property. For example, MS Sans Serif is available only in sizes 8, 10, 12, 14, 18, and 24 points.

You can set the default **DatasheetFontHeight** property by using the settings under **Default Font** on the **Datasheet** tab of the **Options** dialog box, available by clicking **Options** on the **Tools** menu.

The following table contains the properties that don't exist in the DAO **Properties** collection of until you set them by using the **Formatting (Datasheet)** toolbar or you can add them in an Access database (.mdb) by using the **CreateProperty** method and append it to the **DAO Properties** collection.

<table>
<thead>
<tr>
<th>Property</th>
<th>Property</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DatasheetFontItalic</strong></td>
<td><strong>DatasheetForeColor</strong></td>
</tr>
<tr>
<td><strong>DatasheetFontHeight</strong></td>
<td><strong>DatasheetBackColor</strong></td>
</tr>
<tr>
<td><strong>DatasheetFontName</strong></td>
<td><strong>DatasheetGridlinesColor</strong></td>
</tr>
<tr>
<td><strong>DatasheetFontUnderline</strong></td>
<td><strong>DatasheetGridlinesBehavior</strong></td>
</tr>
<tr>
<td><strong>DatasheetFontWeight</strong></td>
<td><strong>DatasheetCellsEffect</strong></td>
</tr>
</tbody>
</table>

**Note** When you add or set any property listed with an asterisk, Microsoft Access automatically adds all the properties listed with an asterisk to the **Properties** collection of the database.
Example

The following example sets the font to MS Serif, the font size to 10 points, and the font weight to medium (500) in Datasheet view of the Products table.

To set these properties, the example uses the SetTableProperty procedure, which is shown in the **DatasheetFontItalic**, **DatasheetFontUnderline** properties example.

```vba
Dim dbs As Object, objProducts As Object
Set dbs = CurrentDb
Const DB_Text As Long = 10
Const DB_Integer As Long = 3
Set objProducts = dbs!Products
SetTableProperty objProducts, "DatasheetFontName", DB_Text, "MS Serif"
SetTableProperty objProducts, "DatasheetFontHeight", DB_Integer, 10
SetTableProperty objProducts, "DatasheetFontWeight", DB_Integer, 500

The next example makes the same changes as the preceding example in Datasheet view of the open Products form.

Forms!Products.DatasheetFontName = "MS Serif"
Forms!Products.DatasheetFontHeight = 10
Forms!Products.DatasheetFontWeight = 500
**DatasheetFontItalic Property**

You can use the **DatasheetFontItalic** property to specify an italicized appearance for field names and data in Datasheet view. Read/write **Boolean**.

`expression.DatasheetFontItalic`

`expression`  Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **DatasheetFontItalic** property applies to all fields in Datasheet view and to form controls when the form is in Datasheet view.

This property is only available in **Visual Basic** within a **Microsoft Access database (.mdb)**.

In Visual Basic, the **DatasheetFontItalic** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>The text is italicized.</td>
</tr>
<tr>
<td>False</td>
<td>(Default) The text isn't italicized.</td>
</tr>
</tbody>
</table>

**Note** You can set this property by clicking **Italic** on the **Formatting (Datasheet)** toolbar.

You can also set this property in the **Font** dialog box, available by clicking **Font** on the **Format** menu in Datasheet view.

The following table contains the properties that don't exist in the DAO **Properties** collection of until you set them by using the **Formatting (Datasheet)** toolbar or you can add them in an Access database (.mdb) by using the **CreateProperty** method and append it to the DAO **Properties** collection.

- **DatasheetFontItalic** *
- **DatasheetFontHeight** *
- **DatasheetFontName** *
- **DatasheetFontUnderline** *
- **DatasheetFontWeight** *
- **DatasheetForeColor** *
- **DatasheetForeColor** *
- **DatasheetBackColor** *
- **DatasheetGridlinesColor** *
- **DatasheetGridlinesBehavior** *
- **DatasheetCellsEffect** *

**Note** When you add or set any property listed with an asterisk, Microsoft Access automatically adds all the properties listed with an asterisk to the **Properties** collection of the database.
Example

The following example displays the data and field names in Datasheet view of the Products form as italic and underlined.

Forms![Products].DatasheetFontItalic = True
Forms![Products].DatasheetFontUnderline = True

The next example displays the data and field names in Datasheet view of the Products table as italic and underlined.

To set the **DatasheetFontItalic** and **DatasheetFontUnderline** properties, the example uses the SetTableProperty procedure, which is in the database's standard module.

Dim dbs As Object, objProducts As Object
Const DB_Boolean As Long = 1
Set dbs = CurrentDb
Set objProducts = dbs![Products]
SetTableProperty objProducts, "DatasheetFontItalic", DB_Boolean, True
SetTableProperty objProducts, "DatasheetFontUnderline", DB_Boolean,

Sub SetTableProperty(objTableObj As Object, strPropertyName As String, intPropertyType As Integer, varPropertyValue As Variant)
    ' Set Microsoft Access-defined table property without causing nonrecoverable run-time error.
    Const conErrPropertyNotFound = 3270
    Dim prpProperty As Variant
    On Error Resume Next ' Don't trap errors.
    objTableObj.Properties(strPropertyName) = varPropertyValue
    If Err <> 0 Then ' Error occurred when value
        If Err <> conErrPropertyNotFound Then
            On Error GoTo @0
            MsgBox "Couldn't set property " & strPropertyName & " on table " & objTableObj.Name & ",", 48, "SetTableProperty"
        Else
            On Error GoTo @0
            Set prpProperty = objTableObj.CreateProperty(strPropertyName, intPropertyType, varPropertyValue)
            objTableObj.Properties.Append prpProperty
            End If
    End If
End Sub
DatasheetFontName Property

You can use the **DatasheetFontName** property to specify the **font** used to display and print field names and data in [Datasheet view](#).
Remarks

The **DatasheetFontName** property applies to all fields in Datasheet view and to form **controls** when the form is in Datasheet view.

This property is only available in **Visual Basic** within a **Microsoft Access database** (.mdb).

You can set this property by selecting the font name from the **Font** box on the **Formatting (Datasheet)** toolbar.

You can also set this property with the **Font** box in the **Font** dialog box, available by clicking **Font** on the **Format** menu in Datasheet view.

For the **DatasheetFontName** property, the font names you can specify depend on the fonts installed on your system and for your printer. If you specify a font that your system can't display or that isn't installed, Microsoft Windows will substitute a similar font.

You can set the default **DatasheetFontName** property by using the settings under **Default Font** on the **Datasheet** tab of the **Options** dialog box, available by clicking **Options** on the **Tools** menu.

The following table contains the properties that don't exist in the DAO **Properties** collection of until you set them by using the **Formatting (Datasheet)** toolbar or you can add them in an Access database (.mdb) by using the **CreateProperty** method and append it to the **DAO Properties** collection.

<table>
<thead>
<tr>
<th><strong>DatasheetFontItalic</strong> *</th>
<th><strong>DatasheetForeColor</strong> *</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DatasheetFontHeight</strong> *</td>
<td><strong>DatasheetBackColor</strong></td>
</tr>
<tr>
<td><strong>DatasheetFontName</strong> *</td>
<td><strong>DatasheetGridlinesColor</strong></td>
</tr>
<tr>
<td><strong>DatasheetFontUnderline</strong> *</td>
<td><strong>DatasheetGridlinesBehavior</strong></td>
</tr>
<tr>
<td><strong>DatasheetFontWeight</strong> *</td>
<td><strong>DatasheetCellsEffect</strong></td>
</tr>
</tbody>
</table>

**Note** When you add or set any property listed with an asterisk, Microsoft Access automatically adds all the properties listed with an asterisk to the **Properties** collection of the database.
DatasheetFontUnderline Property

You can use the **DatasheetFontUnderline** property to specify an underlined appearance for field names and data in **Datasheet view**. Read/write **Boolean**.

\[ expression.\text{DatasheetFontUnderline} \]

*expression*  Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **DatasheetFontUnderline** property applies to all fields in Datasheet view and to form controls when the form is in Datasheet view.

This property is only available in **Visual Basic** within a **Microsoft Access database** (.mdb).

In Visual Basic, the **DatasheetFontUnderline** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>The text is underlined.</td>
</tr>
<tr>
<td>False</td>
<td>(Default) The text isn't underlined.</td>
</tr>
</tbody>
</table>

**Note** You can set this property by clicking **Underline** on the **Formatting (Datasheet)** toolbar.

You can also set this property in the **Font** dialog box, available by clicking **Font** on the **Format** menu in Datasheet view.

The following table contains the properties that don't exist in the DAO **Properties** collection of until you set them by using the **Formatting (Datasheet)** toolbar or you can add them in an Access database (.mdb) by using the **CreateProperty** method and append it to the DAO **Properties** collection.

- **DatasheetFontItalic** *
- **DatasheetFontHeight** *
- **DatasheetFontName** *
- **DatasheetFontUnderline** *
- **DatasheetFontWeight** *
- **DatasheetForeColor** *
- **DatasheetBackColor**
- **DatasheetFontName**
- **DatasheetGridlinesColor**
- **DatasheetGridlinesBehavior**
- **DatasheetCellsEffect**

**Note** When you add or set any property listed with an asterisk, Microsoft Access automatically adds all the properties listed with an asterisk to the **Properties** collection of the database.
**Example**

The following example displays the data and field names in Datasheet view of the Products form as italic and underlined.

Forms![Products].DatasheetFontItalic = True  
Forms![Products].DatasheetFontUnderline = True

The next example displays the data and field names in Datasheet view of the Products table as italic and underlined.

To set the `DatasheetFontItalic` and `DatasheetFontUnderline` properties, the example uses the `SetTableProperty` procedure, which is in the database's standard module.

```vba
Dim dbs As Object, objProducts As Object
Const DB_Boolean As Long = 1
Set dbs = CurrentDb
Set objProducts = dbs![Products]
SetTableProperty objProducts, "DatasheetFontItalic", DB_Boolean, True
SetTableProperty objProducts, "DatasheetFontUnderline", DB_Boolean, True

Sub SetTableProperty(objTableObj As Object, strPropertyName As String, intPropertyType As Integer, varPropertyValue As Variant)
    ' Set Microsoft Access-defined table property without causing nonrecoverable run-time error.
    Const conErrPropertyNotFound = 3270
    Dim prpProperty As Variant
    On Error Resume Next  ' Don't trap errors.
    objTableObj.Properties(strPropertyName) = varPropertyValue
    If Err <> 0 Then  ' Error occurred when value
        If Err <> conErrPropertyNotFound Then
            On Error GoTo 0
            MsgBox "Couldn't set property " & strPropertyName & ", " & objTableObj.Name & "," & "," & varPropertyValue, 48, "SetT
        Else
            On Error GoTo 0
            Set prpProperty = objTableObj.CreateProperty(strPropertyName, intPropertyType, varPropertyValue)
            objTableObj.Properties.Append prpProperty
        End If
    End If
End Sub
```
**DatasheetFontWeight Property**

You can use the **DatasheetFontWeight** property to specify the line width of the font used to display and print characters for field names and data in **Datasheet view**. Read/write **Integer**.

`expression.DatasheetFontWeight`

*expression*  Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **DatasheetFontWeight** property applies to all fields in Datasheet view and to form **controls** when the form is in Datasheet view.

These properties are only available in **Visual Basic** within a **Microsoft Access database** (.mdb)

In Visual Basic, the **DatasheetFontWeight** property setting uses the following **Integer** values.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>Thin</td>
</tr>
<tr>
<td>200</td>
<td>Extra Light</td>
</tr>
<tr>
<td>300</td>
<td>Light</td>
</tr>
<tr>
<td>400</td>
<td>(Default) Normal</td>
</tr>
<tr>
<td>500</td>
<td>Medium</td>
</tr>
<tr>
<td>600</td>
<td>Semi-bold</td>
</tr>
<tr>
<td>700</td>
<td>Bold</td>
</tr>
<tr>
<td>800</td>
<td>Extra Bold</td>
</tr>
<tr>
<td>900</td>
<td>Heavy</td>
</tr>
</tbody>
</table>

The following setting information applies to both Access databases (.mdb) and **Access projects** (.adp):

- You can also set Normal and Bold for this property in the **Font** dialog box, available by clicking **Font** on the **Format** menu in Datasheet view. In the **Font** dialog box’s **Font Style** box, the only available font weight settings are Regular (identical to Normal) and Bold.

- Another method of setting only the Regular and Bold settings is to click **Bold** on the **Formatting (Datasheet)** toolbar.

- You can set the default **DatasheetFontWeight** property by using the settings under **Default Font** on the **Datasheet** tab of the **Options** dialog box, available by clicking **Options** on the **Tools** menu.
Remarks

The following table contains the properties that don't exist in the DAO Properties collection of until you set them by using the Formatting (Datasheet) toolbar or you can add them in an Access database (.mdb) by using the CreateProperty method and append it to the DAO Properties collection.

<table>
<thead>
<tr>
<th>DatasheetFontItalic *</th>
<th>DatasheetForeColor *</th>
</tr>
</thead>
<tbody>
<tr>
<td>DatasheetFontHeight *</td>
<td>DatasheetBackColor</td>
</tr>
<tr>
<td>DatasheetFontName *</td>
<td>DatasheetGridlinesColor</td>
</tr>
<tr>
<td>DatasheetFontUnderline *</td>
<td>DatasheetGridlinesBehavior</td>
</tr>
<tr>
<td>DatasheetFontWeight *</td>
<td>DatasheetCellsEffect</td>
</tr>
</tbody>
</table>

**Note**  When you add or set any property listed with an asterisk, Microsoft Access automatically adds all the properties listed with an asterisk to the Properties collection of the database.
Example

The following example sets the font to MS Serif, the font size to 10 points, and the font weight to medium (500) in Datasheet view of the Products table.

To set these properties, the example uses the SetTableProperty procedure, which is shown in the **DatasheetFontItalic**, **DatasheetFontUnderline** properties example.

Dim dbs As Object, objProducts As Object
Set dbs = CurrentDb
Const DB_Text As Long = 10
Const DB_Integer As Long = 3
Set objProducts = dbs!Products
SetTableProperty objProducts, "DatasheetFontName", DB_Text, "MS Serif"
SetTableProperty objProducts, "DatasheetFontHeight", DB_Integer, 10
SetTableProperty objProducts, "DatasheetFontWeight", DB_Integer, 500

The next example makes the same changes as the preceding example in Datasheet view of the open Products form.

Forms!Products.DatasheetFontName = "MS Serif"
Forms!Products.DatasheetFontHeight = 10
Forms!Products.DatasheetFontWeight = 500
Show All
**DatasheetForeColor Property**

You can use the **DatasheetForeColor** property in Visual Basic to specify or determine the color of all text in a table, query, or form in Datasheet view within an Access database (.mdb). Read/write **Long**.

`expression.DatasheetForeColor`

`expression` Required. An expression that returns one of the objects in the Applies To list.
Remarks

The following setting information applies to both Microsoft Access database and Access projects (.adp):

- You can also set this property by clicking **Font/ForeColor** on the **Formatting (Datasheet)** toolbar and clicking the desired color displayed on the color palette.

- You can also set the default **DatasheetForeColor** property by using the **Datasheet** tab of the **Options** dialog box, available by clicking **Options** on the **Tools** menu.
Remarks

Setting the `DatasheetForeColor` property for a table or query won't affect this property setting for a form that uses the table or query as its source of data.

The following table contains the properties that don't exist in the DAO `Properties` collection of until you set them by using the `Formatting (Datasheet)` toolbar or you can add them in an Access database (.mdb) by using the `CreateProperty` method and append it to the DAO `Properties` collection.

<table>
<thead>
<tr>
<th>DatasheetBackColor</th>
<th>DatasheetFontUnderline *</th>
</tr>
</thead>
<tbody>
<tr>
<td>DatasheetCellsEffect</td>
<td>DatasheetFontWeight *</td>
</tr>
<tr>
<td>DatasheetFontHeight *</td>
<td>DatasheetForeColor*</td>
</tr>
<tr>
<td>DatasheetFontItalic *</td>
<td>DatasheetGridlinesBehavior</td>
</tr>
<tr>
<td>DatasheetFontName *</td>
<td>DatasheetGridlinesColor</td>
</tr>
</tbody>
</table>

**Note**  When you add or set any property listed with an asterisk, Microsoft Access automatically adds it to the `Properties` collection.
Example

The following example uses the SetTableProperty procedure to set a table's font color to dark blue and its background color to light gray. If a "Property not found" error occurs when the property is set, the CreateProperty method is used to add the property to the object's Properties collection.

```
Dim dbs As Object, objProducts As Object
Const lngForeColor As Long = 8388608 ' Dark blue.
Const lngBackColor As Long = 12632256 ' Light gray.
Const DB_Long As Long = 4
Set dbs = CurrentDb
Set objProducts = dbs!Products
SetTableProperty objProducts, "DatasheetBackColor", DB_Long, lngBackColor
SetTableProperty objProducts, "DatasheetForeColor", DB_Long, lngForeColor

Sub SetTableProperty(objTableObj As Object, strPropertyName As String, intPropertyType As Integer, varPropertyValue As Variant)
Const conErrPropertyNotFound = 3270
Dim prpProperty As Variant
On Error Resume Next ' Don't trap errors.
objTableObj.Properties(strPropertyName) = varPropertyValue
If Err <> 0 Then ' Error occurred when value set.
    If Err <> conErrPropertyNotFound Then
        MsgBox "Couldn't set property " & strPropertyName _
        & " on table " & objTableObj.Name & "", vbExclamation
        Err.Clear
    Else ' Error is "Property not found", so add it to collection
        Set prpProperty = objTableObj.CreateProperty(strPropertyName, intPropertyType, varPropertyValue)
        objTableObj.Properties.Append prpProperty
        Err.Clear
    End If
End If
End Sub
```
**DateCreated Property**

Returns a Date indicating the date and time when the design of the specified object was last modified. Read-only.

*expression*.DateCreated

*expression*  Required. An expression that returns one of the objects in the Applies To list.
Example

The following example lists all the reports in the current database and when their designs were created and modified.

Dim acobjLoop As AccessObject

For Each acobjLoop In CurrentProject.AllReports
    With acobjLoop
        Debug.Print .Name & " - Created " & .DateCreated _
        & " - Modified " & .DateModified
    End With
Next acobjLoop
**DateGrouping Property**

You can use the `DateGrouping` property to specify how you want to group dates in a report. Read/write `Byte`.

`expression.DateGrouping`

`expression` Required. An expression that returns one of the objects in the Applies To list.
Remarks

For example, using the US Defaults setting will cause the week to begin on Sunday. If you set a Date/Time field's GroupOn property to Week, the report will group dates from Sunday to Saturday.

Note The DateGrouping property setting applies to the entire report, not to a particular group in the report.

The DateGrouping property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>US Defaults</td>
<td>0</td>
<td>Microsoft Access uses the U.S. settings for the first day of the week (Sunday) and the first week of the year (starts on January 1).</td>
</tr>
<tr>
<td>Use System Settings</td>
<td>1</td>
<td>(Default) Microsoft Access uses settings based on the locale selected in the Regional Options dialog box in Windows Control Panel.</td>
</tr>
</tbody>
</table>

Note You can set this property by using the report's property sheet, a macro, or Visual Basic.

You can set the DateGrouping property only in report Design view or in the Open event procedure of a report.

The sort order used in a report isn't affected by the DateGrouping property setting.
DateModified Property

Returns a **Date** indicating the date and time when the design of the specified object was last modified. Read-only.

*expression*.**DateModified**

*expression*  Required. An expression that returns one of the objects in the Applies To list.
Example

The following example lists all the reports in the current database and when their designs were created and modified.

Dim acobjLoop As AccessObject

For Each acobjLoop In CurrentProject.AllReports
    With acobjLoop
        Debug.Print .Name & " - Created " & .DateCreated _
        & " - Modified " & .DateModified
    End With
Next acobjLoop
DBEngine Property

You can use the **DBEngine** property in **Visual Basic** to access the current **DBEngine** object and its related properties.

*expression*.**DBEngine**

*expression*  Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **DBEngine** property is set by Microsoft Access and is read-only in all views.

The **DBEngine** property of the **Application** object represents the Microsoft Jet database engine. The **DBEngine** object is the top-level object in the Data Access Objects (DAO) model and it contains and controls all other objects in the hierarchy of Data Access Objects.
Example

The following example displays the **DBEngine** properties in a message box.

```vbnet
Private Sub Command1_Click()
    DisplayApplicationInfo Me
End Sub

Function DisplayApplicationInfo(obj As Object) As Integer
    Dim objApp As Object, intI As Integer, strProps As String
    On Error Resume Next
        ' Form Application property.
        Set objApp = obj.Application
        MsgBox "Application Visible property = " & objApp.Visible
        If objApp.UserControl = True Then
            For intI = 0 To objApp.DBEngine.Properties.Count - 1
                strProps = strProps & objApp.DBEngine.Properties(intI).Name & ", "
            Next intI
        End If
    MsgBox Left(strProps, Len(strProps) - 2) & ",", vbOK, "DBEng
End Function
```
DecimalPlaces Property

You can use the **DecimalPlaces** property to specify the number of decimal places Microsoft Access uses to display numbers. Read/write **Byte**.

```
expression.DecimalPlaces
```

*expression* Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **DecimalPlaces** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Auto</td>
<td>255</td>
<td>(Default) Numbers appear as specified by the <strong>Format</strong> property setting. Digits to the right of the decimal separator appear with the specified number of decimal places; digits to the left of the decimal separator appear as specified by the <strong>Format</strong> property setting.</td>
</tr>
<tr>
<td>0 to 15</td>
<td>0 to 15</td>
<td></td>
</tr>
</tbody>
</table>

**Note** You can set this property for **text boxes** and **combo boxes** by using the control's property sheet and for table fields by using the table's property sheet. You can also set this property in the Field Properties property sheet in query Design view.

**Tip**

You should set the **DecimalPlaces** property in the table's property sheet. A bound control you create on a form or report inherits the **DecimalPlaces** property set in the field in the underlying table or query, so you won't have to specify the property individually for every bound control you create.

For controls, you can also set this property by using a **macro** or **Visual Basic**

**Note** The **DecimalPlaces** property setting has no effect if the **Format** property is blank or is set to General Number.

The **DecimalPlaces** property affects only the number of decimal places that display, not how many decimal places are stored. To change the way a number is stored you must change the **FieldSize** property in table Design view.

You can use the **DecimalPlaces** property to display numbers differently from the **Format** property setting or from the way they are stored. For example, the Currency setting of the **Format** property displays only two decimal places
($5.35). To display Currency numbers with four decimal places (for example, $5.3523), set the **DecimalPlaces** property to 4.
Default Property

You can use the Default property to specify whether a command button is the default button on a form. Read/write Boolean.

expression.Default

expression Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **Default** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>True</td>
<td>The command button is the default button.</td>
</tr>
<tr>
<td>No</td>
<td>False</td>
<td>(Default) The command button isn't the default button.</td>
</tr>
</tbody>
</table>

You can set this property by using the command button's [property sheet](#), a [macro](#), or [Visual Basic](#).

When the command button's **Default** property setting is Yes and the Form window is active, the user can choose the command button by pressing ENTER (if no other command button has the [focus](#)) as well as by clicking the command button.

Only one command button on a form can be the default button. When the **Default** property is set to Yes for one command button, it is automatically set to No for all other command buttons on the form.

**Tip**

For a form that supports irreversible operations, such as deletions, it's a good idea to make the [Cancel](#) button the default command button. To do this, set both the **Default** property and the [Cancel](#) property to Yes.
DefaultControl Property

The DefaultControl property returns a Control object with which you can set the default properties for a particular type of control on a particular form or report. For example, before you create a text box on a form, you might want to set the default properties for the text box. Then you can create a number of text boxes that have the same base property settings, rather than having to set properties for each text box individually once it has been created.

expression.DefaultControl(controltype)

The DefaultControl property has the following arguments.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>expression</td>
<td>An expression that evaluates to a Form or Report object on which controls are to be created. The default property settings defined for a type of control apply only to controls of the same type created on this form or report.</td>
</tr>
<tr>
<td>controltype</td>
<td>An intrinsic constant indicating the type of control for which default property settings are to be set.</td>
</tr>
</tbody>
</table>
Remarks

The **DefaultControl** property enables you to set a control's default properties from code. Once you have set the default properties for a particular type of control, each subsequently created control of that type will have the same default values.

For example, if you set the **FontSize** property of the default **command button** to 12, each new command button will have a font size of 12 **points**.

Not all of a control's properties are available as default properties. The default properties available for a control depend on the type of control.

The **DefaultControl** property returns a **Control** object of the type specified by the **controltype** argument. This **Control** object doesn't represent an actual control on a form, but rather a default control that is a template for all subsequently created controls of that type. You set the default control properties for the **Control** object returned by the **DefaultControl** property in the same manner that you would set properties for an individual control on a form.

For a list of intrinsic constants that can be passed as the **controltype** argument, see the **CreateControl** function.

The **DefaultControl** property can be used only in **form Design view** or **report Design view**. If you try to apply this property to a form or report that is not in Design view, a **run-time error** will result.

If you try to set a property that can't be set as a default property with the **DefaultControl** property, a run-time error will result. To determine which properties can be default properties, list the **Properties** collection of the **Control** object returned by the **DefaultControl** property.
Example

The following example creates a new form and uses the **DefaultControl** property to return a **Control** object representing the default command button. The procedure sets some of the default properties for the command button, then creates a new command button on the form.

Sub SetDefaultProperties()
    Dim frm As Form, ctlDefault As Control, ctlNew As Control

    ' Create new form.
    Set frm = CreateForm
    ' Return Control object representing default command button.
    Set ctlDefault = frm.DefaultControl(acCommandButton)
    ' Set some default properties.
    With ctlDefault
        .FontWeight = 700
        .FontSize = 12
        .Width = 3000
        .Height = 1000
    End With
    ' Create new command button.
    Set ctlNew = CreateControl(frm.Name, acCommandButton, , , , 500,
    ' Set control's caption.
    ctlNew.caption = "New Command Button"
    ' Restore form.
    DoCmd.Restore
End Sub
DefaultSize Property

**True** if the size of the detail section in Design View is used for printing; otherwise, the values of the **ItemSizeHeight** and **ItemSizeWidth** properties are used. Read/write **Boolean**.

*expression*.DefaultSize

*expression*  Required. An expression that returns one of the objects in the Applies To list.
Remarks

When this property is **True**, the `ItemSizeHeight` and `ItemSizeWidth` properties are ignored.
Example

The following example sets a variety of printer settings for the form specified in the **strFormname** argument of the procedure.

```vba
Sub SetPrinter(strFormname As String)
    DoCmd.OpenForm FormName:=strFormname, view:=acDesign, _
                    datamode:=acFormEdit, windowmode:=acHidden

    With Forms(form1).Printer
        .TopMargin = 1440
        .BottomMargin = 1440
        .LeftMargin = 1440
        .RightMargin = 1440

        .ColumnSpacing = 360
        .RowSpacing = 360

        .ColorMode = acPRCMColor
        .DataOnly = False
        .DefaultSize = False
        .ItemSizeHeight = 2880
        .ItemSizeWidth = 2880
        .ItemLayout = acPRVerticalColumnLayout
        .ItemsAcross = 6

        .Copies = 1
        .Orientation = acPRORLandscape
        .Duplex = acPRDPVertical
        .PaperBin = acPRBNAuto
        .PaperSize = acPRPSLetter
        .PrintQuality = acPRPQMedium
    End With

    DoCmd.Close objecttype:=acForm, objectname:=strFormname, _
                    Save:=acSaveYes
End Sub
```
DefaultValue Property

Specifies a String value that is automatically entered in a field when a new record is created. For example, in an Addresses table you can set the default value for the City field to New York. When users add a record to the table, they can either accept this value or enter the name of a different city. Read/write.

expression.DefaultValue

expression Required. An expression that returns one of the objects in the Applies To list.
Remarks

- The **DefaultValue** property doesn't apply to check box, option button, or toggle button controls when they are in an option group. It does however apply to the option group itself.
- The **DefaultValue** property applies to all table fields except those fields with the data type of **AutoNumber** or OLE Object.

The **DefaultValue** property specifies text or an expression that's automatically entered in a control or field when a new record is created. For example, if you set the **DefaultValue** property for a text box control to `=Now()` , the control displays the current date and time. The maximum length for a **DefaultValue** property setting is 255 characters.

For a control, you can set this property in the control's **property sheet**. For a field, you can set this property in table Design view (in the Field Properties section), in a **macro**, or by using **Visual Basic**.

In Visual Basic, use a **string expression** to set the value of this property. For example, the following code sets the **DefaultValue** property for a text box control named PaymentMethod to "Cash":

```vbnet
Forms!frmInvoice!PaymentMethod.DefaultValue = """"Cash""
```

**Note**  To set this property for a field by using Visual Basic, use the ADO **DefaultValue** property or the DAO **DefaultValue** property.

The **DefaultValue** property is applied only when you add a new record. If you change the **DefaultValue** property, the change isn't automatically applied to existing records.

If you set the **DefaultValue** property for a form control that's bound to a field that also has a **DefaultValue** property setting defined in the table, the control setting overrides the table setting.

If you create a control by dragging a field from the **field list**, the field's **DefaultValue** property setting, as defined in the table, is applied to the control on the form although the control's **DefaultValue** property setting will remain

```text
Remarks

- The **DefaultValue** property doesn't apply to check box, option button, or toggle button controls when they are in an option group. It does however apply to the option group itself.
- The **DefaultValue** property applies to all table fields except those fields with the data type of **AutoNumber** or OLE Object.

The **DefaultValue** property specifies text or an expression that's automatically entered in a control or field when a new record is created. For example, if you set the **DefaultValue** property for a text box control to `=Now()` , the control displays the current date and time. The maximum length for a **DefaultValue** property setting is 255 characters.

For a control, you can set this property in the control's **property sheet**. For a field, you can set this property in table Design view (in the Field Properties section), in a **macro**, or by using **Visual Basic**.

In Visual Basic, use a **string expression** to set the value of this property. For example, the following code sets the **DefaultValue** property for a text box control named PaymentMethod to "Cash":

```vbnet
Forms!frmInvoice!PaymentMethod.DefaultValue = """"Cash""
```

**Note**  To set this property for a field by using Visual Basic, use the ADO **DefaultValue** property or the DAO **DefaultValue** property.

The **DefaultValue** property is applied only when you add a new record. If you change the **DefaultValue** property, the change isn't automatically applied to existing records.

If you set the **DefaultValue** property for a form control that's bound to a field that also has a **DefaultValue** property setting defined in the table, the control setting overrides the table setting.

If you create a control by dragging a field from the **field list**, the field's **DefaultValue** property setting, as defined in the table, is applied to the control on the form although the control's **DefaultValue** property setting will remain
```
One control can provide the default value for another control. For example, if you set the `DefaultValue` property for a control to the following expression, the control's default value is set to the `DefaultValue` property setting for the `txtShipTo` control.

```vba
=Forms!frmInvoice!txtShipTo
```

If the controls are on the same form, the control that's the source of the default value must appear earlier in the `tab order` than the control containing the expression.
DefaultView Property

You can use the DefaultView property to specify the opening view of a form.
Remarks

The **DefaultView** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single Form</td>
<td>0</td>
<td>(Default) Displays one record at a time.</td>
</tr>
<tr>
<td>Continuous Forms</td>
<td>1</td>
<td>Displays multiple records (as many as will fit in the current window), each in its own copy of the form's detail section.</td>
</tr>
<tr>
<td>Datasheet</td>
<td>2</td>
<td>Displays the form fields arranged in rows and columns like a spreadsheet.</td>
</tr>
<tr>
<td>PivotTable</td>
<td>3</td>
<td>Displays the form as a PivotTable.</td>
</tr>
<tr>
<td>PivotChart</td>
<td>4</td>
<td>Displays the form as a PivotChart.</td>
</tr>
</tbody>
</table>

The views displayed in the **View** button list and on the **View** menu depend on the setting of the **ViewsAllowed** property. For example, if the **ViewsAllowed** property is set to Datasheet, **Form View** is disabled in the **View** button list and on the **View** menu.

The combination of these properties creates the following conditions.

<table>
<thead>
<tr>
<th>DefaultView</th>
<th>ViewsAllowed</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single, Continuous Forms, or Datasheet</td>
<td>Both</td>
<td>Users can switch between Form view and Datasheet view.</td>
</tr>
<tr>
<td>Single or Continuous Forms</td>
<td>Form</td>
<td>Users can't switch from Form view to Datasheet view.</td>
</tr>
<tr>
<td>Single or Continuous Forms</td>
<td>Datasheet</td>
<td>Users can switch from Form view to Datasheet view but not back again.</td>
</tr>
<tr>
<td>Datasheet</td>
<td>Form</td>
<td>Users can switch from Datasheet view to Form view but not back again.</td>
</tr>
<tr>
<td>Datasheet</td>
<td>Datasheet</td>
<td>Users can't switch from Datasheet view to Form view.</td>
</tr>
</tbody>
</table>
DefaultWebOptions Property

You can use the DefaultWebOptions property to reference the read-only DefaultWebOptions object and its related properties.

\textit{expression}.DefaultWebOptions

\textit{expression}  Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **DefaultWebOptions** property is available by using [Visual Basic](https://msdn.microsoft.com/en-us/library/aa166460(v=vs.60).aspx).

**DefaultWebOptions** property to identify or set the **Application** object's **DefaultWebOptions** object properties. These properties can be used to set or change the default web page settings available in the **Web Options** dialog box. To display this dialog box, click **Options** on the **Tools** menu. Click the **General** tab and click the **Web Pages** button.
Example

The following example checks to see whether Microsoft Office Web components are downloaded when a saved data access page is displayed and sets the download flag accordingly.

Set objAppWebOptions = Application.DefaultWebOptions
With objAppWebOptions
    If .DownloadComponents = True Then
        strCompDownload = "Loaded"
    Else
        strCompDownload = "Not Loaded"
    End If
End With
Dependants Property

Returns a `DependencyObjects` collection that represents the objects that depend upon the specified `AccessObject` object. Read-only.

`expression.Dependants`

`expression` Required. An expression that returns a `DependencyInfo` object.
Remarks

To return the collection of objects that an AccessObject object depends upon, use the Dependencies property of the DependencyInfo object.
Dependencies Property

Returns a `DependencyObjects` collection that represents the objects that the specified `AccessObject` object depends upon. Read-only.

`expression.Dependencies`

`expression`  Required. An expression that returns a `DependencyInfo` object.
Remarks

To return the collection of objects that depend upon the specified AccessObject object, use the Dependants property of the DependencyInfo object.
DeviceName Property

Returns a String indicating name of the specified printer device. Read-only.

expression.DeviceName

expression  Required. An expression that returns one of the objects in the Applies To list.
Example

The following example displays information about all the printers available to the system.

Dim prtLoop As Printer

For Each prtLoop In Application.Printers
    With prtLoop
        MsgBox "Device name: " & .DeviceName & vbCrLf _
            & "Driver name: " & .DriverName & vbCrLf _
            & "Port: " & .Port
    End With
Next prtLoop
Dirty Property

You can use the Dirty property to determine whether the current record has been modified since it was last saved. For example, you may want to ask the user whether changes to a record were intended and, if not, allow the user to move to the next record without saving the changes. Read/write Boolean.

expression. Dirty

expression  Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **Dirty** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>True True</td>
<td>The current record has been changed.</td>
</tr>
<tr>
<td>False False</td>
<td>The current record has not been changed.</td>
</tr>
</tbody>
</table>

This property is available in [Form view](#) and [Datasheet view](#).

This property set or read using a [macro](#) or [Visual Basic](#).

When a record is saved, Microsoft Access sets the **Dirty** property to **False**.
When a user makes changes to a record, the property is set to **True**.
Example

The following example enables the btnUndo button when data is changed. The UndoEdits() subroutine is called from the AfterUpdate event of text box controls. Clicking the enabled btnUndo button restores the original value of the control by using the **OldValue** property.

```vbnet
Sub UndoEdits()
    If Me.Dirty Then
        Me!btnUndo.Enabled = True ' Enable button.
    Else
        Me!btnUndo.Enabled = False ' Disable button.
    End If
End Sub

Sub btnUndo_Click()
    Dim ctlC As Control
    ' For each control.
    For Each ctlC in Me.Controls
        If ctlC.ControlType = acTextBox Then
            ' Restore Old Value.
            ctlC.Value = ctlC.OldValue
        End If
    Next ctlC
End Sub
```
DisplayAutoCorrectOptions Property

Returns or sets a **Boolean** value that specifies whether the **AutoCorrect Options** button is displayed for the specified control. The default value is **True**.

Read/write.

```
expression.DisplayAutoCorrectOptions
```

**expression**  Required. An expression that returns one of the objects in the **Applies To** list.
DisplayType Property

You can use the **DisplayType** property to specify whether Microsoft Access displays an **OLE object's** content or an icon. Read/write **Boolean**.

```
expression.DisplayType
```

*expression* Required. An expression that returns one of the objects in the Applies To list.
Remarks

For example, if the OLE object is a Microsoft Word document and you set this property to Content, the control displays the Word document; if you set this property to Icon, the control displays the Microsoft Word icon.

The **DisplayType** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Content</td>
<td>acOLEDisplayContent (Default)</td>
<td>When the control contains an OLE object, the control displays the object's data, such as a document or spreadsheet.</td>
</tr>
<tr>
<td>Icon</td>
<td>acOLEDisplayIcon</td>
<td>When the control contains an OLE object, the control displays the object's icon.</td>
</tr>
</tbody>
</table>

**Note** You can set the **DisplayType** property in a **property sheet**, in a **macro**, or by using **Visual Basic**. You can set the default for this property by using a control's **default control style** or the **DefaultControl** method in Visual Basic.

For a **bound object frame**, the **DisplayType** property can be set either in **Design view** or in **Form view** or **Datasheet view** for new records while they are being added; it can be read in all views. For an **unbound object frame** or **chart**, the property can be set in the **Insert Object** dialog box when the object is created (the default setting is Content or, if you select the **Display As Icon** check box, the setting is Icon).

The **DisplayType** property determines the default setting of the **Display As Icon** check box in the **Paste Special** dialog box, available by clicking **Paste Special** on the **Edit** menu, and the **Insert Object** dialog box, displayed when inserting an unbound object frame. When you display these dialog boxes in **Form view**, **Datasheet view**, or **Design view**, the **Display As Icon** check box is automatically selected if the **DisplayType** property is set to Icon. For example, you will see these boxes selected when using Visual Basic to set the control's **Action** property to **acOLEInsertObjDlg** or **acOLEPasteSpecialDlg**.

The **DisplayType** property setting has no effect on the state of the **Display As Icon** check box in the **Object** dialog box when you insert an object into an
unbound object frame. When you paste an object from the Clipboard, the **Display As Icon** check box reflects the state of the object on the Clipboard.

Changing the **DisplayType** property of a bound object frame doesn't affect the display of existing objects in the control. However, it will affect new objects that you add to the control by using the **Object** command on the **Insert** menu.
DisplayWhen Property

You can use the **DisplayWhen** property to specify which of a form's *sections* or *controls* you want displayed on screen and in print. Read/write **Byte**.

*expression*.DisplayWhen

*expression* Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **DisplayWhen** property applies only to the following form sections: detail, form header, and form footer. It also applies to all controls (except page breaks) on a form.

The **DisplayWhen** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Always</td>
<td>0 (Default)</td>
<td>The object appears in Form view and when printed.</td>
</tr>
<tr>
<td>Print Only</td>
<td>1</td>
<td>The object is hidden in Form view but appears when printed.</td>
</tr>
<tr>
<td>Screen Only</td>
<td>2</td>
<td>The object appears in Form view but not when printed.</td>
</tr>
</tbody>
</table>

**Note**  You can set this property by using the object's property sheet, a macro, or Visual Basic.

For controls, you can set the default for this property by using the default control style or the `DefaultControl` method in Visual Basic.

In many cases, certain controls are useful only in Form view. To prevent Microsoft Access from printing these controls, you can set their **DisplayWhen** property to Screen Only. For example, you might have a command button or instructions on a form that you don't want printed. Or you might have form header and form footer sections that you don't want displayed on screen but that you do want printed. In this case, you should set the **DisplayWhen** property to Print Only.

**Tip**

For reports, use the `Format` and `Retreat` events to specify an event procedure or macro that sets the Visible property of controls you don't want printed. You can also cancel the Format or Print event for a report section to prevent the section from being printed.
DividingLines Property

You can use the **DividingLines** property to specify whether dividing lines will separate sections on a **form** or records displayed on a **continuous form**. Read/write **Boolean**.

*expression*.DividingLines

*expression* Required. An expression that returns one of the objects in the Applies To list.
**Remarks**

The **DividingLines** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>True</td>
<td>(Default) Dividing lines will separate sections and records on continuous forms.</td>
</tr>
<tr>
<td>No</td>
<td>False</td>
<td>There are no dividing lines.</td>
</tr>
</tbody>
</table>

**Note**  You can set the **DividingLines** property by using the form's property sheet, a macro, or Visual Basic.

This property can be set in any view.
DoCmd Property

You can use the **DoCmd** property to access the read-only **DoCmd** object and its related methods.

`expression.DoCmd`

*expression* Required. An expression that returns one of the objects in the Applies To list.
Remarks

This property is available only by using Visual Basic.
Example

The following example opens a form in Form view and moves to a new record.

Sub ShowNewRecord()
    DoCmd.OpenForm "Employees", acNormal
    DoCmd.GoToRecord , , acNewRec
End Sub
Document Property

You can use the Document property to access the Microsoft Internet Explorer Dynamic HTML Document object for HTML pages.

expression.Document

expression Required. An expression that returns one of the objects in the Applies To list.
Remarks

This property is available only by using Visual Basic.

For more information about the DHTML and the Document object model, see the following topics:

DHTML Tutorials

Document Object Model

Document Object References
Example

This procedure illustrates how to use VBA code to add text to a data access page. The following information is supplied in the arguments to this procedure:

strPageName The name of an existing data access page.
strID The ID property (attribute) for the tag that contains the text you want to work with.
strText The text to insert.
blnReplace Whether to replace existing text in the tag.

Function DAPInsertText(strPageName As String, _
    strID As Variant, strText As String, _
    Optional blnReplace As Boolean = True) As Boolean

    Dim blnWasLoaded As Boolean

    On Error GoTo DAPInsertText_Err

    ' Determine if the page exists and whether it is ' currently open. If not open then open it in ' design view.
    If DAPExists(strPageName) = True Then
        If CurrentProject.AllDataAccessPages(strPageName)_.IsLoaded = False Then
            blnWasLoaded = False
            With DoCmd
                .Echo False
                .OpenDataAccessPage strPageName, _
                acDataAccessPageDesign
            End With
            Else
            blnWasLoaded = True
        End If
    Else
        DAPInsertText = False
    Exit Function
    End If

    ' Add the new text to the specified tag.
    With DataAccessPages(strPageName).Document
        If blnReplace = True Then
            .All(strID).innerText = strText
        Else
            .All(strID).innerText = strText
        End If
    End With

End Function

DAPInsertText_Err:
.All(strID).innerText = .All(strID).innerText & strText
End If
' Make sure the text is visible.
With .All(strID).Style
    If .display = "none" Then .display = ""
End With
End With

' Clean up after yourself.
With DoCmd
    If blnWasLoaded = True Then
        .Save
    Else
        .Close acDataAccessPage, strPageName, acSaveYes
    End If
End With
DAPInsertText = True
DAPInsertText_End:
    DoCmd.Echo True
Exit Function
DAPInsertText_Err:
    MsgBox "Error #" & Err.Number & ": " & Err.Description
    DAPInsertText = False
    Resume DAPInsertText_End
End Function
**DownloadComponents Property**

You can use the `DownloadComponents` property to specify or determine if the Microsoft Office tools are automatically downloaded with the Web page. Read/write `Boolean`.

`expression.DownloadComponents`

`expression` Required. An expression that returns one of the objects in the Applies To list.
Remarks

The DownloadComponents property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>True</td>
<td>The necessary Office controls are downloaded when you view the saved document in a Web browser.</td>
</tr>
<tr>
<td>No</td>
<td>False</td>
<td>(Default) Do not download the controls.</td>
</tr>
</tbody>
</table>

Note  The DownloadComponents property is available only by using Visual Basic.

You can set the LocationOfComponents property to a central URL, where the controls can be downloaded by authorized users viewing your saved data access page. The path must be valid and must point to a location that contains the necessary components, and the user must have a valid Microsoft Office license.

Office Web components add interactivity to data access pages that you save as Web pages. If you view a Web page in a browser on a computer that does not have the components installed, the interactive portions of the page will be static.
Example

This example allows the Office Web components to be downloaded with the specified Web page, if they are not already installed.

```csharp
Application.DefaultWebOptions.DownloadComponents = True
Application.DefaultWebOptions.LocationOfComponents = Application.CurrentProject & "\foo"
```
**DrawMode Property**

You can use the **DrawMode** property to specify how the pen (the color used in drawing) interacts with existing background colors on a report when the **Line**, **Circle**, or **Pset** method is used to draw on a report when printing. Read/write **Integer**.

*expression*.DrawMode

*expression*    Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **DrawMode** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Black pen color.</td>
</tr>
<tr>
<td>2</td>
<td>The inverse of setting 15 (NotMergePen).</td>
</tr>
<tr>
<td>3</td>
<td>The combination of the colors common to the background color and the inverse of the pen (MaskNotPen).</td>
</tr>
<tr>
<td>4</td>
<td>The inverse of setting 13 (NotCopyPen).</td>
</tr>
<tr>
<td>5</td>
<td>The combination of the colors common to both the pen and the inverse of the display (MaskPenNot).</td>
</tr>
<tr>
<td>6</td>
<td>The inverse of the display color (Invert).</td>
</tr>
<tr>
<td>7</td>
<td>The combination of the colors in the pen and in the display color, but not in both (XorPen).</td>
</tr>
<tr>
<td>8</td>
<td>The inverse of setting 9 (NotMaskPen).</td>
</tr>
<tr>
<td>9</td>
<td>The combination of the colors common to both the pen and the display (MaskPen).</td>
</tr>
<tr>
<td>10</td>
<td>The inverse of setting 7 (NotXorPen).</td>
</tr>
<tr>
<td>11</td>
<td>No operation— the output remains unchanged. In effect, this setting turns drawing off (Nop).</td>
</tr>
<tr>
<td>12</td>
<td>The combination of the display color and the inverse of the pen color (MergeNotPen).</td>
</tr>
<tr>
<td>13</td>
<td>(Default) The color specified by the <strong>ForeColor</strong> property (CopyPen).</td>
</tr>
<tr>
<td>14</td>
<td>The combination of the pen color and the inverse of the display color (MergePenNot).</td>
</tr>
<tr>
<td>15</td>
<td>The combination of the pen color and the display color (MergePen).</td>
</tr>
<tr>
<td>16</td>
<td>White pen color.</td>
</tr>
</tbody>
</table>

**Note**  The **DrawMode** property setting is an **Integer** value.

You can set the **DrawMode** property by using a **macro** or a **Visual Basic** event procedure specified by a section's **OnPrint** property setting.

Use this property to produce visual effects when drawing on a report. Microsoft
Access compares each pixel in the draw pattern to the corresponding pixel in the existing background to determine how the drawing appears on the report. For example, setting 7 uses the Xor operator to combine a draw-pattern pixel with a background pixel.
Show All
**DrawStyle Property**

You can use the **DrawStyle** property to specify the line style when using the **Line** and **Circle** methods to print lines on reports. Read/write **Integer**.

*expression*.DrawStyle

*expression*  Required. An expression that returns one of the objects in the Applies To list.
## Remarks

The **DrawStyle** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(Default) Solid line, transparent interior</td>
</tr>
<tr>
<td>1</td>
<td>Dash, transparent interior</td>
</tr>
<tr>
<td>2</td>
<td>Dot, transparent interior</td>
</tr>
<tr>
<td>3</td>
<td>Dash-dot, transparent interior</td>
</tr>
<tr>
<td>4</td>
<td>Dash-dot-dot, transparent interior</td>
</tr>
<tr>
<td>5</td>
<td>Invisible line, transparent interior</td>
</tr>
<tr>
<td>6</td>
<td>Invisible line, solid interior</td>
</tr>
</tbody>
</table>

**Note**  You can set this property by using a [macro](https://example.com) or a [Visual Basic](https://example.com) event procedure specified by a section's [OnPrint](https://example.com) property setting.

The **DrawStyle** property produces the results described in the preceding table if the **DrawWidth** property is set to 1. If the **DrawWidth** property setting is greater than 3, the **DrawStyle** property settings 1 through 4 produce a solid line (the **DrawStyle** property value isn't changed).
**DrawWidth Property**

You can use the `DrawWidth` property to specify the line width for the `Line`, `Circle`, and `Pset` methods to print lines on reports. Read/write `Integer`.

`expression.DrawWidth`

`expression` Required. An expression that returns one of the objects in the Applies To list.
Remarks

You can set the **DrawWidth** property to an **Integer** value of 1 through 32,767. This value represents the width of the line in **pixels**. The default is 1, or 1 pixel wide.

You can set this property by using a **macro** or a **Visual Basic** event procedure specified by a **section's OnPrint** property setting.

Increase the value of this property to increase the width of the line. If the **DrawWidth** property setting is greater than 3, **DrawStyle** property settings 1 through 4 produce a solid line (the **DrawStyle** property setting isn't changed). Setting the **DrawWidth** property to 1 enables the **DrawStyle** property to produce the results shown in the setting table of the **DrawStyle** property.
DriverName Property

Returns a String indicating the name of the driver used by the specified printer. Read-only.

expression.DriverName

expression Required. An expression that returns one of the objects in the Applies To list.
Example

The following example displays information about all the printers available to the system.

Dim prtLoop As Printer

For Each prtLoop In Application.Printers
    With prtLoop
        MsgBox "Device name: " & .DeviceName & vbCrLf & "Driver name: " & .DriverName & vbCrLf & "Port: " & .Port
    End With
Next prtLoop
**Duplex Property**

Returns or sets an *AcPrintDuplex* constant indicating how the specified printer handles duplex printing. Read/write.

*AcPrintDuplex* can be one of these *AcPrintDuplex* constants.

- `acPRDPHorizontal`
- `acPRDPSimplex`
- `acPRDPVertical`

`expression.Duplex`

`expression` Required. An expression that returns one of the objects in the Applies To list.
Example

The following example sets a variety of printer settings for the form specified in the `strFormname` argument of the procedure.

```vba
Sub SetPrinter(strFormname As String)
    DoCmd.OpenForm FormName:=strFormname, view:=acDesign,_
    datamode:=acFormEdit, windowmode:=acHidden
    With Forms(form1).Printer
        .TopMargin = 1440
        .BottomMargin = 1440
        .LeftMargin = 1440
        .RightMargin = 1440
        .ColumnSpacing = 360
        .RowSpacing = 360
        .ColorMode = acPRCMColor
        .DataOnly = False
        .DefaultSize = False
        .ItemSizeHeight = 2880
        .ItemSizeWidth = 2880
        .ItemLayout = acPRVerticalColumnLayout
        .ItemsAcross = 6
        .Copies = 1
        .Orientation = acPRORLandscape
        .Duplex = acPRDPVertical
        .PaperBin = acPRBNAuto
        .PaperSize = acPRPSLetter
        .PrintQuality = acPRPQMedium
    End With
    DoCmd.Close objecttype:=acForm, objectname:=strFormname, _
    Save:=acSaveYes
End Sub
```
EmailSubject Property

You can use the **EmailSubject** property to specify or determine return the email subject line of a hyperlink to an object, document, Web page or other destination for a [command button](#), [image control](#), or [label](#) control. Read/write **String**.

**expression**.EmailSubject

**expression**  Required. An expression that returns one of the objects in the Applies To list.
Remarks

The EmailSubject property is a string expression representing the subject line within a file (UNC path) or Web page (URL).

You can set the EmailSubject property by using a macro or Visual Basic.

You can also use the Insert Hyperlink dialog box to set this property by clicking the Build button to the right of the property box in the property sheet.

Note When you create a hyperlink by using the Insert Hyperlink dialog box, Microsoft Access automatically sets the EmailSubject property to the location specified in the Subject box of the E-Mail Address tab.

When you move the cursor over a command button, image control, or label control whose HyperlinkAddress property is set, the cursor changes to an upward-pointing hand. Clicking the control displays the object or Web page specified by the link.

To open objects in the current database, leave the HyperlinkAddress property blank and specify the object type and object name you want to open in the HyperlinkSubAddress property by using the syntax "objecttype objectname". If you want to open an object contained in another Microsoft Access database, enter the database path and file name in the HyperlinkAddress property and specify the database object to open by using the HyperlinkSubAddress property.

The HyperlinkAddress property can contain an absolute or a relative path to a target document. An absolute path is a fully qualified URL or UNC path to a document. A relative path is a path related to the base path specified in the Hyperlink Base setting in the DatabaseName Properties dialog box (available by clicking Database Properties on the File menu) or to the current database path. If Microsoft Access can't resolve the HyperlinkAddress property setting to a valid URL or UNC path, it will assume you've specified a path relative to the base path contained in the Hyperlink Base setting or the current database path.

Note When you follow a hyperlink to another Microsoft Access database object,
the database Startup properties are applied. For example, if the destination database has a Display form set, that form is displayed when the database opens.
Enabled Property

You can use the **Enabled** property to set or return the status of the conditional format in the **FormatCondition** object. Read/write **Boolean**.

`expression.Enabled`  

**expression** Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **Enabled** property setting is a value that indicates whether the conditional format is enabled or disabled. **True** enables the conditional format. **False** disables the conditional format. The default is **True**.

The **Enabled** property is available only by using **Visual Basic**.

When the **Enabled** property is **True**, the conditional format can be displayed in the **Conditional Formatting** dialog box. The **Conditional Formatting** dialog box is available by clicking **Conditional Formatting** on the **Format** menu.
Encoding Property

You can use the **Encoding** property to specify or determine the data access page encoding (code page or character set) to be used by the Web browser when you view the saved data access page. Read/write **MsoEncoding**.

MsoEncoding can be one of these MsoEncoding constants.

- `msoEncodingArabic`
- `msoEncodingArabicASMO`
- `msoEncodingArabicAutoDetect`
- `msoEncodingArabicTransparentASMO`
- `msoEncodingAutoDetect`
- `msoEncodingBaltic`
- `msoEncodingCentralEuropean`
- `msoEncodingCyrillic`
- `msoEncodingCyrillicAutoDetect`
- `msoEncodingEBCDICArabic`
- `msoEncodingEBCDICDenmarkNorway`
- `msoEncodingEBCDICFinlandSweden`
- `msoEncodingEBCDICFrance`
- `msoEncodingEBCDICGermany`
- `msoEncodingEBCDICGreek`
- `msoEncodingEBCDICGreekModern`
- `msoEncodingEBCDICHebrew`
- `msoEncodingEBCDICIcelandic`
- `msoEncodingEBCDICInternational`
- `msoEncodingEBCDICItaly`
- `msoEncodingEBCDICJapaneseKatakanaExtended`
- `msoEncodingEBCDICJapaneseKatakanaExtendedAndJapanese`
- `msoEncodingEBCDICJapaneseLatinExtendedAndJapanese`
- `msoEncodingEBCDICKoreanExtended`
- `msoEncodingEBCDICKoreanExtendedAndKorean`
msoEncodingEBCDILatinAmericaSpain
msoEncodingEBCDICMultilingualROECELatin2
msoEncodingEBCDICRussian
msoEncodingEBCDICSerbianBulgarian
msoEncodingEBCDICSimplifiedChineseExtendedAndSimplifiedChinese
msoEncodingEBCDICThai
msoEncodingEBCDICTurkish
msoEncodingEBCDICTurkishLatin5
msoEncodingEBCDICUnitedKingdom
msoEncodingEBCDICUSCanada
msoEncodingEBCDICUSCanadaAndJapanese
msoEncodingEBCDICUSCanadaAndTraditionalChinese
msoEncodingEUCChineseSimplifiedChinese
msoEncodingEUCJapanese
msoEncodingEUCKorean
msoEncodingEUCTaiwaneseTraditionalChinese
msoEncodingEuropa3
msoEncodingExtAlphaLowercase
msoEncodingGreek
msoEncodingGreekAutoDetect
msoEncodingHebrew
msoEncodingHZGBSimplifiedChinese
msoEncodingIA5German
msoEncodingIA5IRV
msoEncodingIA5Norwegian
msoEncodingIA5Swedish
msoEncodingISO2022CNSimplifiedChinese
msoEncodingISO2022CNTraditionalChinese
msoEncodingISO2022JPJISX02011989
msoEncodingISO2022JPJISX02021984
msoEncodingISO2022JPNoHalfwidthKatakana
msoEncodingISO2022KR
msoEncodingISO6937NonSpacingAccent
msoEncodingWestern

expression.Encoding

expression Required. An expression that returns one of the objects in the Applies To list.
Example

This example checks to see whether the default document encoding is Western.

If Application.DefaultWebOptions.Encoding = msoEncodingWestern Then
    strDocEncoding = "Western"
Else
    strDocEncoding = "Other"
End If
EnterKeyBehavior Property

You can use the **EnterKeyBehavior** property to specify what happens when you press ENTER in a text box control in Form view or Datasheet view. Read/write **Boolean**.

```
expression.EnterKeyBehavior
```

*expression* Required. An expression that returns one of the objects in the Applies To list.
Remarks

For example, you can use this property if you have a control bound to a Memo field in a table to make entering multiple-line text easier. If you don't set this property to New Line In Field, you must press CTRL+ENTER to enter a new line in the text box.

The **EnterKeyBehavior** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>False</td>
<td>(Default) Microsoft Access uses the result specified under <strong>Move after enter</strong> area on the Keyboard tab of the Options dialog box, available by clicking Options on the Tools menu. For details, see the Remarks section.</td>
</tr>
<tr>
<td>New Line In</td>
<td>True</td>
<td>Pressing ENTER in the control creates a new line in the control so you can enter additional text.</td>
</tr>
</tbody>
</table>

**Note** You can set this property by using a form's property sheet, a macro, or Visual Basic.

You can set the default for this property by using the control's default control style or the DefaultControl method in Visual Basic.

The following options are available under **Move after enter** area on the Keyboard tab of the Options dialog box.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Don't move</td>
<td>Pressing ENTER has no effect.</td>
</tr>
<tr>
<td>Next field</td>
<td>Pressing ENTER moves the insertion point to the next control or field in the form or datasheet in the tab order.</td>
</tr>
<tr>
<td>Next record</td>
<td>Pressing ENTER moves the insertion point to the first control or field in the next record on the form or datasheet.</td>
</tr>
</tbody>
</table>
EventProcPrefix Property

You can use the `EventProcPrefix` property to get the prefix portion of an event procedure name. Read/write String.

`expression.EventProcPrefix`

`expression` Required. An expression that returns one of the objects in the Applies To list.
Remarks

For example, if you have a command button with an event procedure named Details_Click, the EventProcPrefix property returns the string "Details".

The EventProcPrefix property setting is a string expression.

This property is available only by using a macro or Visual Basic and is read-only in all views.

Microsoft Access adds the prefix portion of an event procedure name to the event name with an underscore character (_).
Expression1 Property

You can use the Expression1 property to return the values of a conditional format within a FormatCondition object. Read-only String.

expression.Expression1

expression Required. An expression that returns one of the objects in the Applies To list.
Remarks

Note  The Expression1 property returns a Variant value or expression associated with the first part of the conditional format.

The Expression1 property is available only by using Visual Basic.

Conditional formatting can also be set on a combo box or text box from the Conditional Formatting dialog box. The Conditional Formatting dialog box is available by clicking Conditional Formatting on the Format menu when a form is in Design view.
Expression2 Property

You can use the Expression2 property to return the values of a conditional format within a FormatCondition object. Read-only String.

expression.Expression2

expression Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **Expression2** property returns a **Variant** value or expression associated with the second part of the conditional format when the **Operator** property of the **FormatCondition** object is **acBetween** or **acNotBetween**, otherwise, the **Expression2** property is **NULL**.

The **Expression2** property is available only by using **Visual Basic**.

Conditional formatting can also be set on a combo box or text box from the **Conditional Formatting** dialog box. The **Conditional Formatting** dialog box is available by clicking **Conditional Formatting** on the **Format** menu when a form is in Design view.
FastLaserPrinting Property

You can use the FastLaserPrinting property to specify whether lines and rectangles are replaced by text character lines—similar to the underscore (_) and vertical bar (|) characters—when you print a form or report using most laser printers. Replacing lines and rectangles with text character lines can make printing much faster. Read/write Boolean.

expression.FastLaserPrinting

expression Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **FastLaserPrinting** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>True</td>
<td>(Default) Lines and rectangles are replaced by text character lines.</td>
</tr>
<tr>
<td>No</td>
<td>False</td>
<td>Lines and rectangles aren't replaced by text character lines.</td>
</tr>
</tbody>
</table>

**Note**  You can set this property by using a form's or report's property sheet, a macro, or Visual Basic.

The **FastLaserPrinting** property affects any line or rectangle on a form or report, including controls that have these shapes (for example, a border around a text box).

**Note**  This property has no effect on PostScript printers, dot-matrix printers, or earlier versions of laser printers that don't support text character lines.

When this property is set to Yes and the form or report being printed has overlapping rectangles or lines, the rectangles or lines on top don't erase the rectangles or lines they overlap. If you require overlapping graphics on your report, set the **FastLaserPrinting** property to No.
Example

The following example shows how to set the **FastLaserPrinting** property for the Invoice report while in report Design view:

```vba
DoCmd.OpenReport "Invoice", acDesign
Reports!Invoice.FastLaserPrinting = True
DoCmd.Close acReport, "Invoice", acSaveYes
```
FeatureInstall Property

You can use the **FeatureInstall** property to specify or determine how Microsoft Access handles calls to methods and properties that require features not yet installed. Read/write **MsoFeatureInstall**.

MsoFeatureInstall can be one of these MsoFeatureInstall constants.

- **msoFeatureInstallNone** *(Default)* An Automation error occurs at run time when uninstalled features are called.
- **msoFeatureInstallOnDemand** The user is prompted to install new features.
- **msoFeatureInstallOnDemandWithUI** The feature is installed automatically and a progress meter is displayed during installation. The user isn't prompted to install new features.

**expression.FeatureInstall**

expression Required. An expression that returns one of the objects in the Applies To list.
Remarks

When VBA code references an object that is not installed the Microsoft Installer technology will attempt to install the required feature. You use the **FeatureInstall** property to control what happens when an uninstalled object is referenced. When this property is set to the default, any attempt to use an uninstalled object causes the Installer technology to try to install the requested feature. In some circumstances this may take some time, and the user may believe that the machine has stopped responding to additional commands.

You can set the **FeatureInstall** property to **msoFeatureInstallOnDemandWithUI** so users can see that something is happening as the feature is being installed. You can set the **FeatureInstall** property to **msoFeatureInstallNone** if you want to trap the error that is returned and display your own dialog box to the user or take some other custom action.

If you have the **UserControl** property set to **False**, users will not be prompted to install new features even if the **FeatureInstall** property is set to **msoFeatureInstallOnDemand**. If the **UserControl** property is set to **True**, an installation progress meter will appear if the **FeatureInstall** property is set to **msoFeatureInstallOnDemand**.
Example

This example checks the value of the **FeatureInstall** property. If the property is set to **msoFeatureInstallNone**, the code displays a message box that asks the user whether they want to change the property setting. If the user responds "Yes", the property is set to **msoFeatureInstallOnDemand**. The example uses an object variable named MyOfficeApp that is dimensioned as an application object.

Dim myofficeapp As Access.Application
Set myofficeapp = New Access.Application

With MyOfficeApp
    If .FeatureInstall = msoFeatureInstallNone Then
        Reply = MsgBox("Uninstalled features for " & vbCrLf & "this application may " & vbCrLf & "cause a run-time error when called." & vbCrLf & "Would you like to change this setting" & vbCrLf & "to automatically install missing features?", vbYesNo, "Feature Install Setting")
        If Reply = vbYes Then
            .FeatureInstall = msoFeatureInstallOnDemand
        End If
    End If
End With
FetchDefaults Property

Returns or sets a Boolean indicating whether Microsoft Access shows default values for new rows on the specified form before the row is saved. True if Access shows the default values for new rows on the specified form. Read/write.

expression.FetchDefaults

expression Required. An expression that returns a Form object.
**Example**

The following example sets a form to show default values for new rows.

```plaintext
Forms(0).FetchDefaults = True
```
FileDialog Property

Returns a **FileDialog** object which represents a single instance of a file dialog box.

*expression*.**FileDialog**(dialogType)

*expression* Required. An expression that returns one of the objects in the Applies To list.

dialogType Required **MsoFileDialogType**. The type of file dialog box.

MsoFileDialogType can be one of these MsoFileDialogType constants.

  * **msoFileDialogFilePicker**
  * **msoFileDialogFolderPicker**
  * **msoFileDialogOpen**
  * **msoFileDialogSaveAs**
Example

This example displays the **Save As** dialog box.

```vba
Dim dlgSaveAs As FileDialog
Set dlgSaveAs = Application.FileDialog(FileDialogType:=msoFileDialogSaveAs)
dlgSaveAs.Show
```

This example displays the **Open** dialog box and allows a user to select multiple files to open.

```vba
Dim dlgOpen As FileDialog
Set dlgOpen = Application.FileDialog(FileDialogType:=msoFileDialogOpen)
With dlgOpen
    .AllowMultiSelect = True
    .Show
End With
```
FileFormat Property

Returns an AcFileFormat constant indicating the Microsoft Access version format of the specified project. Read-only.

AcFileFormat can be one of these AcFileFormat constants.
- acFileFormatAccess2
- acFileFormatAccess2000
- acFileFormatAccess2002
- acFileFormatAccess95
- acFileFormatAccess97

*expression*.FileFormat

*expression* Required. An expression that returns one of the objects in the Applies To list.
Remarks

Use the `ConvertAccessProject` method to convert an Access project from one version to another.
Example

This example evaluates the current project and displays a message indicating to which version of Microsoft Access its file format corresponds.

Dim strFormat As String

Select Case CurrentProject.FileFormat
  Case acFileFormatAccess2
    strFormat = "Microsoft Access 2"
  Case acFileFormatAccess95
    strFormat = "Microsoft Access 95"
  Case acFileFormatAccess97
    strFormat = "Microsoft Access 97"
  Case acFileFormatAccess2000
    strFormat = "Microsoft Access 2000"
  Case acFileFormatAccess2002
    strFormat = "Access 2002 - 2003"
End Select

MsgBox "This is a " & strFormat & " project."
FileSearch Property

You can use the FileSearch property to return a read-only reference to the current FileSearch object and its related properties and methods.

expression.FileSearch

expression Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **FileSearch** property is available only by using [Visual Basic](https://docs.microsoft.com/en-us/previous-versions/dotnet Framework/aa251771(v=netframework-4.0)).

Once you establish a reference to the **FileSearch** object, you can access all the properties and methods of the object. You can set a reference to the **FileSearch** object by clicking **References** on the **Tools** menu while in module **Design view**. Then set a reference to the Microsoft Office Object Library in the **References** dialog box by selecting the appropriate check box. Microsoft Access can set this reference for you if you use a Microsoft Office Object Library constant to set a **FileSearch** object's property or as an argument to a **FileSearch** object's method.
FillColor Property

You use the FillColor property to specify the color that fills in boxes and circles drawn on reports with the Line and Circle methods. You can also use this property with Visual Basic to create special visual effects on custom reports when you print using a color printer or preview the reports on a color monitor. Read/write Long.

expression.FillColor

expression  Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **FillColor** property contains a *numeric expression* that specifies the fill color for all boxes and circles.

You can set this property only by using a *macro* or a Visual Basic event procedure specified by a section's **OnPrint** event property.

You can use the **RGB** or **QBColor** functions to set this property. The **FillColor** property setting has a data type of **Long**.
Example

The following example uses the **Circle** method to draw a circle and create a pie slice within the circle. Then it uses the **FillColor** and **FillStyle** properties to color the pie slice red. It also draws a line from the upper left to the center of the circle.

To try this example in Microsoft Access, create a new report. Set the **OnPrint** property of the Detail section to [Event Procedure]. Enter the following code in the report's module, then switch to Print Preview.

```vba
Private Sub Detail_Print(Cancel As Integer, PrintCount As Integer)

Const conPI = 3.14159265359
Dim sngHCtr As Single
Dim sngVCtr As Single
Dim sngRadius As Single
Dim sngStart As Single
Dim sngEnd As Single

sngHCtr = Me.ScaleWidth / 2 ' Horizontal center.
sngVCtr = Me.ScaleHeight / 2 ' Vertical center.
sngRadius = Me.ScaleHeight / 3 ' Circle radius.
Me.Circle (sngHCtr, sngVCtr), sngRadius ' Draw circle.
sngStart = -0.000000001 ' Start of pie slice.
sngEnd = -2 * conPI / 3 ' End of pie slice.
Me.FillColor = RGB(255, 0, 0) ' Color pie slice red.
Me.FillStyle = 0 ' Fill pie slice.

' Draw pie slice within circle
Me.Circle (sngHCtr, sngVCtr), sngRadius, , sngStart, sngEnd

' Draw line to center of circle.
Dim intColor As Integer
Dim sngTop As Single, sngLeft As Single
Dim sngWidth As Single, sngHeight As Single

Me.ScaleMode = 3 ' Set scale to pixels.
sngTop = Me.ScaleTop ' Top inside edge.
sngLeft = Me.ScaleLeft ' Left inside edge.
sngWidth = Me.ScaleWidth / 2 ' Width inside edge.
sngHeight = Me.ScaleHeight / 2 ' Height inside edge.
intColor = RGB(255, 0, 0) ' Make color red.
```
' Draw line.
Me.Line (sngTop, sngLeft)-(sngWidth, sngHeight), intColor

End Sub
FillStyle Property

You can use the FillStyle property to specify whether a circle or line drawn by the Circle or Line method on a report is transparent, opaque, or filled with a pattern. Read/write Integer.

expression.FillStyle

expression Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **FillStyle** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Opaque</td>
</tr>
<tr>
<td>1</td>
<td>(Default) Transparent</td>
</tr>
<tr>
<td>2</td>
<td>Horizontal Line</td>
</tr>
<tr>
<td>3</td>
<td>Vertical Line</td>
</tr>
<tr>
<td>4</td>
<td>Upward Diagonal</td>
</tr>
<tr>
<td>5</td>
<td>Downward Diagonal</td>
</tr>
<tr>
<td>6</td>
<td>Cross</td>
</tr>
<tr>
<td>7</td>
<td>Diagonal Cross</td>
</tr>
</tbody>
</table>

**Note**  You can set the **FillStyle** property by using a macro or a Visual Basic event procedure specified by a section's **OnPrint** property setting.

When the **FillStyle** property is set to 0, a circle or line has the color set by the **FillColor** property. When the **FillStyle** property is set to 1, the interior of the circle or line is transparent and has the color of the report behind it.

To use the **FillStyle** property, the **SpecialEffect** property must be set to Normal.
Example

The following example uses the **Circle** method to draw a circle and create a pie slice within the circle. Then it uses the **FillColor** and **FillStyle** properties to color the pie slice red. It also draws a line from the upper left to the center of the circle.

To try this example in Microsoft Access, create a new report. Set the **OnPrint** property of the Detail section to [Event Procedure]. Enter the following code in the report's module, then switch to Print Preview.

```vba
Private Sub Detail_Print(Cancel As Integer, PrintCount As Integer)
    Const conPI = 3.14159265359
    Dim sngHCtr As Single
    Dim sngVCtr As Single
    Dim sngRadius As Single
    Dim sngStart As Single
    Dim sngEnd As Single

    sngHCtr = Me.ScaleWidth / 2  ' Horizontal center.
    sngVCtr = Me.ScaleHeight / 2  ' Vertical center.
    sngRadius = Me.ScaleHeight / 3 ' Circle radius.
    Me.Circle (sngHCtr, sngVCtr), sngRadius ' Draw circle.
    sngStart = -0.000000001 ' Start of pie slice.
    sngEnd = -2 * conPI / 3 ' End of pie slice.
    Me.FillColor = RGB(255, 0, 0) ' Color pie slice red.
    Me.FillStyle = 0 ' Fill pie slice.

    ' Draw Pie slice within circle
    Me.Circle (sngHCtr, sngVCtr), sngRadius, , sngStart, sngEnd

    ' Draw line to center of circle.
    Dim intColor As Integer
    Dim sngTop As Single, sngLeft As Single
    Dim sngWidth As Single, sngHeight As Single

    Me.ScaleMode = 3 ' Set scale to pixels.
    sngTop = Me.ScaleTop ' Top inside edge.
    sngLeft = Me.ScaleLeft ' Left inside edge.
    sngWidth = Me.ScaleWidth / 2 ' Width inside edge.
    sngHeight = Me.ScaleHeight / 2 ' Height inside edge.
    intColor = RGB(255, 0, 0) ' Make color red.
```
' Draw line. 
Me.Line (sngTop, sngLeft)-(sngWidth, sngHeight), intColor

End Sub
Filter Property

You can use the **Filter** property to specify a subset of records to be displayed when a filter is applied to a **form**, **report** **query**, or **table**. Read/write **String**.

*expression*.**Filter**

*expression*  Required. An expression that returns one of the objects in the Applies To list.
Remarks

If you want to specify a server filter within a Microsoft Access project (.adp) for data located on a server, use the ServerFilter property.

The Filter property is a string expression consisting of a WHERE clause without the WHERE keyword. For example, the following Visual Basic code defines and applies a filter to show only customers from the USA:

Me.Filter = "Country = 'USA'"
Me.FilterOn = True

You can set this property by using a table's or form's property sheet, a macro, or Visual Basic.

You can also set this property in Form view or Datasheet view by pointing to Filter on the Records menu and clicking one of the commands on the submenu.

Note Setting the Filter property has no effect on the ADO Filter property.

You can use the Filter property to save a filter and apply it at a later time. Filters are saved with the objects in which they are created. They are automatically loaded when the object is opened, but they aren't automatically applied.

When a new object is created, it inherits the RecordSource, Filter, OrderBy, and OrderByOn properties of the table or query it was created from.

To apply a saved filter to a form, query, or table, you can click Apply Filter on the toolbar, click Apply Filter/Sort on the Records menu, or use a macro or Visual Basic to set the FilterOn property to True. For reports, you can apply a filter by setting the FilterOn property to Yes in the report's property sheet.

The Apply Filter button indicates the state of the Filter and FilterOn properties. The button remains disabled until there is a filter to apply. If an existing filter is currently applied, the Apply Filter button appears pressed in.

To apply a filter automatically when a form is opened, specify in the OnOpen event property setting of the form either a macro that uses the ApplyFilter action
or an event procedure that uses the **ApplyFilter** method of the **DoCmd** object.

You can remove a filter by clicking the pressed-in **Apply Filter** button, clicking **Remove Filter/Sort** on the **Records** menu, or using Visual Basic to set the **FilterOn** property to **False**.

**Note** You can save a filter as a query by clicking **Save As Query** on the **File** menu while in the **Filter By Form** window or the **Advanced Filter/Sort** window.

When the **Filter** property is set in **form Design view**, Microsoft Access does not attempt to validate the SQL expression. If the SQL expression is invalid, an error occurs when the filter is applied.
FilterLookup Property

You can use the FilterLookup property to specify whether values appear in a bound text box control when using the Filter By Form or Server Filter By Form window. Read/write Byte.

expression.FilterLookup

expression Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **FilterLookup** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Never</td>
<td>0</td>
<td>The field values aren't displayed. You can specify whether the filtered records can contain <strong>Null</strong> values. (Default) The field values are displayed according to the settings under <strong>Filter by form defaults</strong> on the <strong>Edit/Find</strong> tab of the <strong>Option</strong> dialog box, available by clicking <strong>Options</strong> on the <strong>Tools</strong> menu.</td>
</tr>
<tr>
<td>Database Default</td>
<td>1</td>
<td>The field values are always displayed.</td>
</tr>
<tr>
<td>Always</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

You can set the **FilterLookup** property by using the text box's **property sheet**, a **macro**, or **Visual Basic**.

You can set the default for this property by using the text box control's **default control style** or the **DefaultControl** method in Visual Basic.

If you want to limit the types of fields to display, clear the appropriate check box under **Filter by form defaults** on the **Edit/Find** tab of the **Option** dialog box, available by clicking **Options** on the **Tools** menu. If field lists aren't displayed, you should increase the number in the **Don't display lists where more than this number of records read** box so the setting is greater than or equal to the maximum number of records in any field in the underlying source of records.
FilterOn Property

You can use the FilterOn property to specify or determine whether the Filter property for a form or report is applied. Read/write Boolean.

expression.FilterOn

expression Required. An expression that returns one of the objects in the Applies To list.
Remarks

If you have specified a server filter within a Microsoft Access project (.adp), use the ServerFilterByForm property.

The FilterOn property uses the following settings.

<table>
<thead>
<tr>
<th>Setting Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>True</td>
</tr>
<tr>
<td>No</td>
<td>False (Default)</td>
</tr>
</tbody>
</table>

The object's Filter property is applied.

Note For reports, you can set the FilterOn property by using the report's property sheet or Visual Basic.

For forms, you can set the FilterOn property in a macro or by using Visual Basic. You can also set this property by clicking Apply Filter on the Form View toolbar or the Filter/Sort toolbar.

To apply a saved filter, press the Apply Filter button for forms, or apply the filter by using a macro or Visual Basic by setting the FilterOn property to True for forms or reports. For reports, you can set the FilterOn property to Yes in the report's property sheet.

The Apply Filter button indicates the state of the Filter and FilterOn properties. The button remains disabled until there is a filter to apply. If an existing filter is currently applied, the Apply Filter button appears pressed in. To apply a filter automatically when a form or report is opened, specify in the OnOpen event property setting of the form either a macro that uses the ApplyFilter action or an event procedure that uses the ApplyFilter method of the DoCmd object.

You can remove a filter by clicking the pressed-in Apply Filter button, clicking Remove Filter/Sort on the Records menu, or by using Visual Basic to set the FilterOn property to False. For reports, you can remove a filter by setting the FilterOn property to No in the report's property sheet.

Note When a new object is created, it inherits the RecordSource, Filter, ServerFilter, OrderBy, and OrderByOn properties of the table or query it was
created from. For forms and reports, inherited filters aren't automatically applied when an object is opened.
FolderSuffix Property

You can use the **FolderSuffix** property to determine the folder suffix that Microsoft Access uses when you save a data access page as a Web page, use long file names, and choose to save supporting files in a separate folder (that is, if the **UseLongFileNames** and **OrganizeInFolder** properties are set to **True**.) Read-only **String**.

`expression.FolderSuffix`

*expression* Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **FolderSuffix** property represents the text displayed as a hyperlink.

This property is available only by using **Visual Basic**.

Newly created data access pages use the suffix returned by the **FolderSuffix** property of the **DefaultWebOptions** object. The value of the **FolderSuffix** property of the **WebOptions** object may differ from that of the **DefaultWebOptions** object if the data access page was previously edited in a different language version of Microsoft Access. You can use the **UseDefaultFolderSuffix** method to change the suffix to the language you are currently using in Microsoft Office.

By default, the name of the supporting folder is the name of the Web page plus an underscore (_), a period (.), or a hyphen (-) and the word "files" (appearing in the language of the version of Microsoft Access in which the file was saved as a Web page). For example, suppose that you use the Dutch language version of Access to save a file called "Page1" as a Web page. The default name of the supporting folder would be Page1_bestanden.

The following table lists each language version of Office, and gives its corresponding **LanguageID** property value and folder suffix. For the languages that are not listed in the table, the suffix ".files" is used.

<table>
<thead>
<tr>
<th>Language</th>
<th>LanguageID</th>
<th>Folder suffix</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arabic</td>
<td>1025</td>
<td>.files</td>
</tr>
<tr>
<td>Basque</td>
<td>1069</td>
<td>_fitxategiak</td>
</tr>
<tr>
<td>Brazilian</td>
<td>1046</td>
<td>_arquivos</td>
</tr>
<tr>
<td>Bulgarian</td>
<td>1026</td>
<td>.files</td>
</tr>
<tr>
<td>Catalan</td>
<td>1027</td>
<td>_fitxers</td>
</tr>
<tr>
<td>Chinese - Simplified</td>
<td>2052</td>
<td>.files</td>
</tr>
<tr>
<td>Chinese - Traditional</td>
<td>1028</td>
<td>.files</td>
</tr>
<tr>
<td>Croatian</td>
<td>1050</td>
<td>_datoteke</td>
</tr>
<tr>
<td>Language</td>
<td>Code</td>
<td>Translation</td>
</tr>
<tr>
<td>------------------</td>
<td>--------</td>
<td>--------------</td>
</tr>
<tr>
<td>Czech</td>
<td>1029</td>
<td>_soubory</td>
</tr>
<tr>
<td>Danish</td>
<td>1030</td>
<td>-filer</td>
</tr>
<tr>
<td>Dutch</td>
<td>1043</td>
<td>_bestanden</td>
</tr>
<tr>
<td>English</td>
<td>1033</td>
<td>_files</td>
</tr>
<tr>
<td>Estonian</td>
<td>1061</td>
<td>_failid</td>
</tr>
<tr>
<td>Finnish</td>
<td>1035</td>
<td>_tiedostot</td>
</tr>
<tr>
<td>French</td>
<td>1036</td>
<td>_fichiers</td>
</tr>
<tr>
<td>German</td>
<td>1031</td>
<td>-Dateien</td>
</tr>
<tr>
<td>Greek</td>
<td>1032</td>
<td>.files</td>
</tr>
<tr>
<td>Hebrew</td>
<td>1037</td>
<td>.files</td>
</tr>
<tr>
<td>Hungarian</td>
<td>1038</td>
<td>_elemei</td>
</tr>
<tr>
<td>Italian</td>
<td>1040</td>
<td>-file</td>
</tr>
<tr>
<td>Japanese</td>
<td>1041</td>
<td>.files</td>
</tr>
<tr>
<td>Korean</td>
<td>1042</td>
<td>.files</td>
</tr>
<tr>
<td>Latvian</td>
<td>1062</td>
<td>_fails</td>
</tr>
<tr>
<td>Lithuanian</td>
<td>1063</td>
<td>_bylos</td>
</tr>
<tr>
<td>Norwegian</td>
<td>1044</td>
<td>-filer</td>
</tr>
<tr>
<td>Polish</td>
<td>1045</td>
<td>_pliki</td>
</tr>
<tr>
<td>Portuguese</td>
<td>2070</td>
<td>_ficheiros</td>
</tr>
<tr>
<td>Romanian</td>
<td>1048</td>
<td>.files</td>
</tr>
<tr>
<td>Russian</td>
<td>1049</td>
<td>.files</td>
</tr>
<tr>
<td>Serbian (Cyrillic)</td>
<td>3098</td>
<td>.files</td>
</tr>
<tr>
<td>Serbian (Latin)</td>
<td>2074</td>
<td>_fajlovi</td>
</tr>
<tr>
<td>Slovakian</td>
<td>1051</td>
<td>.files</td>
</tr>
<tr>
<td>Slovenian</td>
<td>1060</td>
<td>_datoteke</td>
</tr>
<tr>
<td>Spanish</td>
<td>3082</td>
<td>_archivos</td>
</tr>
<tr>
<td>Swedish</td>
<td>1053</td>
<td>-filer</td>
</tr>
<tr>
<td>Thai</td>
<td>1054</td>
<td>.files</td>
</tr>
<tr>
<td>Turkish</td>
<td>1055</td>
<td>_dosyalar</td>
</tr>
<tr>
<td>Ukranian</td>
<td>1058</td>
<td>.files</td>
</tr>
<tr>
<td>Vietnamese</td>
<td>1066</td>
<td>.files</td>
</tr>
</tbody>
</table>
Example

This example returns the folder suffix used by the data access page ("Inventory"). The suffix is returned in the string variable strFolderSuffix.

strFolderSuffix = DataAccessPages("Inventory").WebOptions.FolderSuffix
FollowedHyperlinkColor Property

You can use the FollowedHyperlinkColor property to specify or determine the default color of all followed hyperlinks within the Application object.
Remarks

The **FollowedHyperlinkColor** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black</td>
<td><code>acColorIndexBlack</code> (0)</td>
</tr>
<tr>
<td>Maroon</td>
<td><code>acColorIndexMaroon</code> (1)</td>
</tr>
<tr>
<td>Green</td>
<td><code>acColorIndexGreen</code> (2)</td>
</tr>
<tr>
<td>Olive</td>
<td><code>acColorIndexOlive</code> (3)</td>
</tr>
<tr>
<td>Dark Blue</td>
<td><code>acColorIndexDarkBlue</code> (4)</td>
</tr>
<tr>
<td>Violet (default)</td>
<td><code>acColorIndexViolet</code> (5)</td>
</tr>
<tr>
<td>Teal</td>
<td><code>acColorIndexTeal</code> (6)</td>
</tr>
<tr>
<td>Gray</td>
<td><code>acColorIndexGray</code> (7)</td>
</tr>
<tr>
<td>Silver</td>
<td><code>acColorIndexSilver</code> (8)</td>
</tr>
<tr>
<td>Red</td>
<td><code>acColorIndexRed</code> (9)</td>
</tr>
<tr>
<td>Bright Green</td>
<td><code>acColorIndexBrightGreen</code> (10)</td>
</tr>
<tr>
<td>Yellow</td>
<td><code>acColorIndexYellow</code> (11)</td>
</tr>
<tr>
<td>Blue</td>
<td><code>acColorIndexBlue</code> (12)</td>
</tr>
<tr>
<td>Fushia</td>
<td><code>acColorIndexFushia</code> (13)</td>
</tr>
<tr>
<td>Aqua</td>
<td><code>acColorIndexAqua</code> (14)</td>
</tr>
<tr>
<td>White</td>
<td><code>acColorIndexWhite</code> (15)</td>
</tr>
</tbody>
</table>

**Note** You can set the **FollowedHyperlinkColor** property through the **DefaultWebOptions** property or the **SetOption** method by using **Visual Basic**.

You can to set or change the default followed hyperlink color available in the **Web Options** dialog box. To display this dialog box, click **Options** on the **Tools** menu. Click the **General** tab and click the **Web Pages** button.

The default color of a hyperlink is changed to the followed hyperlink color when a hyperlink control has been pressed.

Use the **DefaultWebOptions** property to identify or set the **Application** object's **DefaultWebOptions** object properties.
FontBold Property

You can use the **FontBold** property to specify whether a font appears in a bold style in the following situations:

- When displaying or printing controls on forms and reports.
- When using the **Print** method on a report.

Remarks

The **FontBold** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>The text is bold.</td>
</tr>
<tr>
<td>False</td>
<td>(Default) The text isn't bold.</td>
</tr>
</tbody>
</table>

**Note** You can set the **FontBold** property only by using a macro or Visual Basic.

To use the **FontBold** property on a report, first create a Print event procedure that prints the desired text.

A font's appearance on screen and in print may differ, depending on your computer and printer.

The **FontWeight** property, which is available in the property sheet for controls, can also be used to set the line width for a control's text. The **FontBold** property gives you a quick way to make text bold; the **FontWeight** property gives you finer control over the line width setting for text. The following table shows the relationship between these properties' settings.

<table>
<thead>
<tr>
<th>If</th>
<th>Then</th>
</tr>
</thead>
<tbody>
<tr>
<td>FontBold = False</td>
<td>FontWeight = Normal (400)</td>
</tr>
<tr>
<td>FontBold = True</td>
<td>FontWeight = Bold (700)</td>
</tr>
<tr>
<td>FontWeight &lt; 700</td>
<td>FontBold = False</td>
</tr>
</tbody>
</table>
FontWeight \geq 700 \textbf{FontBold} = \textbf{True}
Example

The following Print event procedure prints a report title and the current date in a bold style on a report at the coordinates specified by the **CurrentX** and **CurrentY** property settings.

```
Private Sub ReportHeader0_Print(Cancel As Integer, _
    PrintCount As Integer)
    Dim MyDate

    MyDate = Date
    Me.FontBold = True
    ' Print report title in bold.
    Me.Print("Sales Management Report")
    Me.Print(MyDate)
End Sub
```
**FontItalic Property**

You can use the **FontItalic** property to specify whether text is italic in the following situations:

- When displaying or printing **controls** on **forms** and **reports**.
- When using the **Print** method on a report.

Read/write **Boolean** for the following objects: **ComboBox**, **CommandButton**, **FormatCondition**, **Label**, **ListBox**, **TabControl**, **TextBox**, and **ToggleButton**. Read/write **Integer** for the **Report** object.

`expression.FontItalic`  

`expression` Required. An expression that returns one of the above objects.

Remarks

The **FontItalic** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td><strong>True</strong> (-1)</td>
<td>The text is italic.</td>
</tr>
<tr>
<td>No</td>
<td><strong>False</strong> (0)</td>
<td>(Default) The text isn't italic.</td>
</tr>
</tbody>
</table>

For controls on forms and reports, you can set this property by using the control's **property sheet**, a **macro**, or **Visual Basic**.

For reports, you can use this property only in an **event procedure** or in a macro specified by the **OnPrint** **event property** setting.

You can also set this property by clicking **Italic** on the **Formatting (Form/Report)** toolbar.

You can set the default for this property by using the **default control style** or the **DefaultControl** method in **Visual Basic**.
FontName Property

You can use the **FontName** property to specify the font for text in the following situations:

- When displaying or printing **controls** on **forms** and **reports**.
- When using the **Print** method on a report.

Read/write **String**.

\[\text{expression}.\text{FontName}\]

**expression**  Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **FontName** property setting is the name of the font that the text is displayed in.

For controls on forms and reports, you can set this property by using the property sheet, a **macro**, or **Visual Basic**.

You can also set this property by clicking the **Font** box on the **Formatting (Form/Report)** toolbar.

You can set the default for this property by using a control's **default control style** or the **DefaultControl** method in Visual Basic.

For reports, you can set this property only in an **event procedure** or in a macro specified by the **OnPrint** event property setting.

In Visual Basic, you set the **FontName** property by using a string expression that is the name of the desired font.

Font availability depends on your system and printer. If you select a font that your system can't display or that isn't installed, Windows substitutes a similar font.
Example

The following example uses the **Print** method to display text on a report named Report1. It uses the **TextWidth** and **TextHeight** methods to center the text vertically and horizontally.

```vbscript
Private Sub Detail_Format(Cancel As Integer, _
    FormatCount As Integer)
    Dim rpt as Report
    Dim strMessage As String
    Dim intHorSize As Integer, intVerSize As Integer

    Set rpt = Me
    strMessage = "DisplayMessage"
    With rpt
        'Set scale to pixels, and set FontName and
        'FontSize properties.
        .ScaleMode = 3
        .FontName = "Courier"
        .FontSize = 24
    End With

    'Horizontal width.
    intHorSize = Rpt.TextWidth(strMessage)

    'Vertical height.
    intVerSize = Rpt.TextHeight(strMessage)

    'Calculate location of text to be displayed.
    Rpt.CurrentX = (Rpt.ScaleWidth/2) - (intHorSize/2)
    Rpt.CurrentY = (Rpt.ScaleHeight/2) - (intVerSize/2)

    'Print text on Report object.
    Rpt.Print strMessage
End Sub
```
FontSize Property

You can use the `FontSize` property to specify the point size for text in the following situations:

- When displaying or printing controls on forms and reports.
- When using the `Print` method on a report.

Read/write `Integer`.

`expression.FontSize`

`expression` Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **FontSize** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>(Default for all reports and controls except <a href="#">command buttons</a>) The text is 8-point type.</td>
</tr>
<tr>
<td>10</td>
<td>(Default for command buttons) The text is 10-point type.</td>
</tr>
<tr>
<td>Other sizes</td>
<td>The text is the indicated size.</td>
</tr>
</tbody>
</table>

**Note** For controls on forms and reports, you can set this property by using the [property sheet](#), a [macro](#), or [Visual Basic](#).

You can also set this property by clicking the **Font Size** box on the **Formatting (Form/Report)** toolbar.

You can set the default for this property by using a control's [default control style](#) or the [DefaultControl](#) method in Visual Basic.

For reports, you can set this property only in an [event procedure](#) or in a macro specified by the [OnPrint](#) event property setting.

In Visual Basic, you set the **FontSize** property by using a [numeric expression](#) equal to the desired size of the font. The setting for the **FontSize** property can be between 1 and 127, inclusive.
Example

The following example uses the **Print** method to display text on a report named Report1. It uses the **TextWidth** and **TextHeight** methods to center the text vertically and horizontally.

```vbnet
Private Sub Detail_Format(Cancel As Integer, _
    FormatCount As Integer)
    Dim rpt As Report
    Dim strMessage As String
    Dim intHorSize As Integer, intVerSize As Integer

    Set rpt = Me
    strMessage = "DisplayMessage"
    With rpt
        'Set scale to pixels, and set FontName and
        'FontSize properties.
        .ScaleMode = 3
        .FontName = "Courier"
        .FontSize = 24
    End With

    'Horizontal width.
    intHorSize = Rpt.TextWidth(strMessage)

    'Vertical height.
    intVerSize = Rpt.TextHeight(strMessage)

    'Calculate location of text to be displayed.
    Rpt.CurrentX = (Rpt.ScaleWidth/2) - (intHorSize/2)
    Rpt.CurrentY = (Rpt.ScaleHeight/2) - (intVerSize/2)

    'Print text on Report object.
    Rpt.Print strMessage
End Sub
```
FontUnderline Property

You can use the **FontUnderline** properties to specify whether text is underlined in the following situations:

- When displaying or printing **controls** on **forms** and **reports**.
- When using the **Print** method on a report.

Read/write **Boolean** for the following objects: **ComboBox**, **CommandButton**, **FormatCondition**, **Label**, **ListBox**, **TabControl**, **TextBox**, and **ToggleButton**. Read/write **Integer** for the **Report** object.

`expression.FontUnderline`

`expression` Required. An expression that returns one of the above objects.
## Remarks

The **FontUnderline** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td><strong>True</strong> (-1)</td>
<td>The text is underlined.</td>
</tr>
<tr>
<td>No</td>
<td><strong>False</strong> (0)</td>
<td>(Default) The text isn't underlined.</td>
</tr>
</tbody>
</table>

**Note**  For controls on forms and reports, you can set this property by using the control's [property sheet](https://example.com), a [macro](https://example.com), or [Visual Basic](https://example.com).

For reports, you can use this property only in an [event procedure](https://example.com) or in a macro specified by the **OnPrint** event property setting.

You can also set this property by clicking **Underline** on the **Formatting (Form/Report)** toolbar.

You can set the default for this property by using the [default control style](https://example.com) or the [DefaultControl](https://example.com) method in Visual Basic.

For a [text box](https://example.com), [combo box](https://example.com), [label](https://example.com), or [command button](https://example.com) that [contains a hyperlink](https://example.com), Microsoft Access automatically sets the **FontUnderline** property to **Yes** if the **Underline Hyperlinks** box is checked on the [Hyperlinks/HTML](https://example.com) tab of the **Options** dialog box, available by clicking **Options** on the **Tools** menu. If you remove the hyperlink from the control (for example, by changing the **ControlSource** property of a bound text box to a source that isn't a Hyperlink field), Microsoft Access sets the **FontUnderline** property back to the default control style. For command buttons, the **FontUnderline** property setting takes effect only if the command button contains a caption rather than a picture.
FontWeight Property

Use the FontWeight property to specify the line width that Windows uses to display and print characters in a control. Read/write Integer.

expression.FontWeight

expression Required. An expression that returns one of the objects in the Applies To list.
Remarks

The FontWeight property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thin</td>
<td>100</td>
</tr>
<tr>
<td>Extra Light</td>
<td>200</td>
</tr>
<tr>
<td>Light</td>
<td>300</td>
</tr>
<tr>
<td>Normal</td>
<td>400</td>
</tr>
<tr>
<td>Medium</td>
<td>500</td>
</tr>
<tr>
<td>Semi-bold</td>
<td>600</td>
</tr>
<tr>
<td>Bold</td>
<td>700</td>
</tr>
<tr>
<td>Extra Bold</td>
<td>800</td>
</tr>
<tr>
<td>Heavy</td>
<td>900</td>
</tr>
</tbody>
</table>

**Note** You can set this property by using the control's property sheet, a macro, or Visual Basic. You can also click Bold on the Formatting (Form/Report) toolbar. This sets the FontWeight property to Bold (700).

You can set the default for this property by using a control's default control style or the DefaultControl method in Visual Basic.

A font's appearance on screen and in print may differ, depending on your computer and printer. For example, a FontWeight property setting of Thin may look identical to Normal on screen but appear lighter when printed.

The FontBold property, which is available only in Visual Basic and macros, can also be used to set the line width for a control's or report's text to bold. The FontBold property gives you a quick way to make text bold; the FontWeight property gives you finer control over the line width setting for text. The following table shows the relationship between these properties' settings.

<table>
<thead>
<tr>
<th>If</th>
<th>Then</th>
</tr>
</thead>
<tbody>
<tr>
<td>FontBold = False</td>
<td>FontWeight = Normal (400)</td>
</tr>
<tr>
<td>FontBold = True</td>
<td>FontWeight = Bold (700)</td>
</tr>
</tbody>
</table>
**FontWeight** < 700  **FontBold** = False
**FontWeight** >= 700 **FontBold** = True
ForceNewPage Property

You can use the **ForceNewPage** property to specify whether form sections (detail, footer) or report sections (header, detail, footer) print on a separate page, rather than on the current page. Read/write **Byte**.

*expression*.ForceNewPage

*expression*  Required. An expression that returns one of the objects in the Applies To list.
Remarks

For example, you may have designed the last page of a report as an order form. If the report footer's **ForceNewPage** property is set to Before Section, the order form is always printed on a new page.

**Note** The **ForceNewPage** property does not apply to page headers or page footers.

The **ForceNewPage** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>0</td>
<td>(Default) The current section (the section for which you're setting the property) is printed on the current page.</td>
</tr>
<tr>
<td>Before Section</td>
<td>1</td>
<td>The current section is printed at the top of a new page.</td>
</tr>
<tr>
<td>After Section</td>
<td>2</td>
<td>The section immediately following the current section is printed at the top of a new page.</td>
</tr>
<tr>
<td>Before &amp; After</td>
<td>3</td>
<td>The current section is printed at the top of a new page, and the next section is printed at the top of a new page.</td>
</tr>
</tbody>
</table>

**Note** You can set this property by using the section's property sheet, a macro, or Visual Basic.

Here are some examples of the **ForceNewPage** property setting.

<table>
<thead>
<tr>
<th>Section</th>
<th>Sample setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A group header displaying the year</td>
<td>Before Section</td>
<td>The group header is printed at the top of the page, followed by the detail section, group footer, and page footer.</td>
</tr>
<tr>
<td>A report detail section</td>
<td>After Section</td>
<td>The group footer is printed at the top of a new page.</td>
</tr>
<tr>
<td>A report header containing the report title</td>
<td>After</td>
<td>The report title and logo are printed on a</td>
</tr>
</tbody>
</table>
and company logo. Section separate page at the beginning of the report.
Example

The following example returns the `ForceNewPage` property setting for the detail section of the Sales By Date report and assigns it to the `intGetVal` variable.

```vba
Dim intGetVal As Integer
intGetVal = Reports![Sales By Year].Section(acDetail) ForceNewPage
```
ForeColor Property

You can use the **ForeColor** property to specify the color for text in a control. Read/write **Long**.

`expression.ForeColor`

`expression`  Required. An expression that returns one of the objects in the Applies To list.
Remarks

You can use this property for controls on forms or reports to make them easy to read or to convey a special meaning. For example, you can change the color of the text in the UnitsInStock control when its value falls below the reorder level.

You can also use this property on reports to create special visual effects when you print with a color printer. When used on a report, this property specifies the printing and drawing color for the Print, Line, and Circle methods.

The ForeColor property contains a numeric expression that represents the value of the text color in the control.

You can use the Color Builder to set this property by clicking the Build button to the right of the property box in the property sheet. Using the Color Builder enables you to define custom colors for text in controls.

For controls, you can set this property by using Font/Fore Color on the Formatting (Form/Report) toolbar, the control's property sheet, a macro, or Visual Basic.

You can set the default for this property by using a control's default control style or the DefaultControl method in Visual Basic.

For reports, you can set the ForeColor property only by using a macro or a Visual Basic event procedure specified in a section's OnPrint event property setting.

For Table objects, you can set this property using Font/Fore Color on the Formatting (Datasheet) toolbar, or in Visual Basic by using the DatasheetForeColor property.

For a text box, combo box, label, or command button that contains a hyperlink, Microsoft Access automatically sets the ForeColor property to the color specified in the Followed Hyperlink Color or Hyperlink Color box on the Hyperlinks/HTML tab of the Options dialog box, available by clicking Options on the Tools menu. If you remove the hyperlink from the control (for example, by changing the ControlSource property of a bound text box to a
source that isn't a Hyperlink field), Microsoft Access sets the **ForeColor** property back to the default control style. For command buttons, the **ForeColor** property setting takes effect only if the command button contains a caption rather than a picture.
Example

The following example uses the **rgb** function to set the **BorderColor**, **BackColor**, and **ForeColor** properties depending on the value of the **txtPastDue** text box. You can also use the **QBColor** function to set these properties. Putting the following code in the **Form_Current()** event sets the control display characteristics as soon as the user opens a form or moves to a new record.

```vba
Sub Form_Current()
    Dim curAmntDue As Currency, lngBlack As Long
    Dim lngRed As Long, lngYellow As Long, lngWhite As Long

    If Not IsNull(Me!txtPastDue.Value) Then
        curAmntDue = Me!txtPastDue.Value
    Else
        Exit Sub
    End If
    lngRed = RGB(255, 0, 0)
    lngBlack = RGB(0, 0, 0)
    lngYellow = RGB(255, 255, 0)
    lngWhite = RGB(255, 255, 255)
    If curAmntDue > 100 Then
        Me!txtPastDue.BorderColor = lngRed
        Me!txtPastDue.ForeColor = lngRed
        Me!txtPastDue.BackColor = lngYellow
    Else
        Me!txtPastDue.BorderColor = lngBlack
        Me!txtPastDue.ForeColor = lngBlack
        Me!txtPastDue.BackColor = lngWhite
    End If
End Sub
```
Show All
Form Property

You can use the **Form** property to refer to a **form** or to refer to the form associated with a **subform control**.
**Remarks**

This property refers to a form object. It is read-only in all views.

You can use this property by using a macro or Visual Basic.

This property is typically used to refer to the form or report contained in a subform control. For example, the following code uses the Form property to access the OrderID control on a subform contained in the OrderDetails subform control.

```vbnet
Dim intOrderID As Integer
intOrderID = Forms!Orders!OrderDetails.Form!OrderID
```

The next example calls a function from a property sheet by using the Form property to refer to the active form that contains the control named CustomerID.

```vbnet
=MyFunction(Form!CustomerID)
```

When you use the Form property in this manner, you are referring to the active form, and the name of the form isn't necessary.

The next example is the Visual Basic equivalent of the preceding example.

```vbnet
X = MyFunction(Forms!Customers!CustomerID)
```

**Note**  When you use the Forms collection, you must specify the name of the form.
Example

The following example uses the **Form** property to refer to a control on a subform.

```vba
Dim curTotalAmount As Currency
curTotalAmount = Forms!Orders!OrderDetails.Form!TotalAmount
```
Format Property

You can use the **Format** property to customize the way numbers, dates, times, and text are displayed and printed. Read/write **String**.

*expression*.Format

*expression*   Required. An expression that returns one of the objects in the Applies To list.
Remarks

You can use one of the predefined formats or you can create a custom format by using formatting symbols.

The **Format** property uses different settings for different data types. For information about settings for a specific data type, see one of the following topics:

- [Date/Time Data Type](#)
- [Number and Currency Data Types](#)
- [Text and Memo Data Types](#)
- [Yes/No Data Type](#)

For a control, you can set this property in the control's property sheet. For a field, you can set this property in table Design view (in the Field Properties section) or in Design view of the Query window (in the Field Properties property sheet). You can also use a macro or Visual Basic.

**Note** In Visual Basic, enter a **string expression** that corresponds to one of the predefined formats or enter a custom format.

The **Format** property affects only how data is displayed. It doesn't affect how data is stored.

Microsoft Access provides predefined formats for Date/Time, Number and Currency, Text and Memo, and Yes/No data types. The predefined formats depend on the country/region specified by double-clicking Regional Options in Windows Control Panel. Microsoft Access displays formats appropriate for the country/region selected. For example, with **English (United States)** selected on the **General** tab, 1234.56 in the Currency format appears as $1,234.56, but when **English (British)** is selected on the **General** tab, the number appears as £1,234.56.

If you set a field's **Format** property in table Design view, Microsoft Access uses that format to display data in datasheets. It also applies the field's **Format**
property to new controls on forms and reports.

You can use the following symbols in custom formats for any data type.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>(space)</td>
<td>Display spaces as literal characters.</td>
</tr>
<tr>
<td>&quot;ABC&quot;</td>
<td>Display anything inside quotation marks as literal characters.</td>
</tr>
<tr>
<td>!</td>
<td>Force left alignment instead of right alignment.</td>
</tr>
<tr>
<td>*</td>
<td>Fill available space with the next character.</td>
</tr>
<tr>
<td>\</td>
<td>Display the next character as a literal character.</td>
</tr>
<tr>
<td>[color]</td>
<td>Display the formatted data in the color specified between the brackets.</td>
</tr>
</tbody>
</table>

Available colors: Black, Blue, Green, Cyan, Red, Magenta, Yellow, White.

You can't mix custom formatting symbols for the Number and Currency data types with Date/Time, Yes/No, or Text and Memo formatting symbols.

When you have defined an input mask and set the Format property for the same data, the Format property takes precedence when the data is displayed and the input mask is ignored. For example, if you create a Password input mask in table Design view and also set the Format property for the same field, either in the table or in a control on a form, the Password input mask is ignored and the data is displayed according to the Format property.
**Example**

The following three examples set the **Format** property by using a predefined format:

Me!Date.Format = "Medium Date"

Me!Time.Format = "Long Time"

Me!Registered.Format = "Yes/No"

The next example sets the **Format** property by using a custom format. This format displays a date as: Jan 1995.

Forms!Employees!HireDate.Format = "mmm yyy"

The following example demonstrates a Visual Basic function that formats numeric data by using the Currency format and formats text data entirely in capital letters. The function is called from the OnLostFocus event of an unbound control named TaxRefund.

Function FormatValue() As Integer
    Dim varEnteredValue As Variant
    
    varEnteredValue = Forms!Survey!TaxRefund.Value
    If IsNumeric(varEnteredValue) = True Then
        Forms!Survey!TaxRefund.Format = "Currency"
    Else
        Forms!Survey!TaxRefund.Format = " >"
    End If
End Function
FormatConditions Property

You can use the `FormatConditions` property to return a read-only reference to the `FormatConditions` collection and its related properties.

`expression.FormatConditions`

`expression` Required. An expression that returns one of the objects in the Applies To list.
Remarks

This property is available only by using Visual Basic.

Conditional formatting can also be set on a combo box or text box from the Conditional Formatting dialog box. The Conditional Formatting dialog box is available by clicking Conditional Formatting on the Format menu when a form is in Design view.
Example

The following example sets format properties for an existing conditional format for the "Textbox1" control.

With forms("forms1").Controls("Textbox1").FormatConditions(1)
  .BackColor = RGB(255,255,255)
  .FontBold = True
  .ForeColor = RGB(255,0,0)
End With
**FormatCount Property**

You can use the `FormatCount` property to determine the number of times the `OnFormat` property has been evaluated for the current section on a report. Read/write `Integer`.

`expression.FormatCount`

*expression*  Required. An expression that returns one of the objects in the Applies To list.
Remarks

You can use this property only in a [macro] or an [Visual Basic event procedure] specified by a section's [OnFormat] property setting.

This property isn't available in [report Design view].

Microsoft Access increments the [FormatCount] property each time the [OnFormat] property setting is evaluated for the current section. As the next section is formatted, Microsoft Access resets the [FormatCount] property to 1.

Under some circumstances, Microsoft Access formats a section more than once. For example, you might design a report in which the [KeepTogether] property for the detail section is set to Yes. When Microsoft Access reaches the bottom of a page, it formats the current detail section once to see if it will fit. If it doesn't fit, Microsoft Access moves to the next page and formats the detail section again. In this case, the setting for the [FormatCount] property for the detail section is 2 because it was formatted twice before it was printed.

You can use the [FormatCount] property to ensure that an operation that affects formatting gets executed only once for a section.
Example

In the following example, the **DLookup** function is evaluated only when the **FormatCount** property is set to 1:

```vba
Private Sub Detail_Format(Cancel As Integer, _
    FormatCount As Integer)
Const conBold = 700
Const conNormal = 400
If FormatCount = 1 Then
    If DLookup("CompanyName", _
        "Customers", "CustomerID = Reports!" _
        & "[Customer Labels]!CustomerID") _
        Like "B*" Then
        CompanyNameLine.FontWeight = conBold
    Else
        CompanyNameLine.FontWeight = conNormal
    End If
End If
End Sub
```
Forms Property

You can use the **Forms** property to return a read-only reference to the **Forms** collection and its related properties.

`expression.Forms`

`expression`  Required. An expression that returns one of the objects in the Applies To list.
Remarks

This property is available only by using Visual Basic.

The properties of the Forms collection in Visual Basic refer to forms that are currently open.
Show All
FrozenColumns Property

You can use the `FrozenColumns` property to determine how many columns in a datasheet are frozen. Read/write `Integer`.

`expression.FrozenColumns`

`expression` Required. An expression that returns one of the objects in the Applies To list.
Remarks

Frozen columns are displayed on the left side of the datasheet and don't move when you scroll horizontally through the datasheet.

**Note** The **FrozenColumns** property applies only to tables, forms, and queries in **Datasheet view**.

The **FrozenColumns** property is set by Microsoft Access when you click **Freeze Columns** on the **Format** menu.

In **Visual Basic**, this property setting is an **Integer** value indicating the number of columns in the datasheet that have been frozen by using the **Freeze Columns** command. The record selector column is always frozen, so the default value is 1. Consequently, if you freeze one column, the **FrozenColumns** property is set to 2; if you freeze two columns, it's set to 3; and so on.

This property setting is read-only in all views.

When you freeze columns by using the **Freeze Columns** command, Microsoft Access automatically moves the columns to the leftmost edge of the datasheet in the order in which you freeze them. For example, if you freeze three columns, these become the first, second, and third columns in the datasheet. Because the record selector column is always frozen, the **FrozenColumns** property in this case will be set to 4. The three columns you freeze will have their **ColumnOrder** properties set to 1, 2, and 3 (in the order they are frozen).

If you click **Unfreeze All Columns** on the **Format** menu, all frozen columns will be unfrozen, and the **FrozenColumns** property will be set to 1.

**Note** The **Unfreeze All Columns** command will not restore the original order of columns if the **Freeze Columns** command caused the column order to change.
Example

The following example uses the **FrozenColumns** property to determine how many columns are frozen in a table in Datasheet view. If more than three columns are frozen, the table size is maximized so you can see as many unfrozen columns as possible.

Sub CheckFrozen(strTableName As String)
    Dim dbs As Object
    Dim tdf As Object
    Dim prp As Variant
    Const DB_Integer As Integer = 3
    Const conPropertyNotFound = 3270 ' Property not found error.
    Set dbs = CurrentDb ' Get current database.
    Set tdf = dbs.TableDefs(strTableName) ' Get object for table.
    DoCmd.OpenTable strTableName, acNormal ' Open table.
    tdf.Properties.Refresh
    On Error GoTo Frozen_Err
    If tdf.Properties("FrozenColumns") > 3 Then ' Check property.
        DoCmd.Maximize
    End If
Frozen_Bye:
    Exit Sub
Frozen_Err:
    If Err = conPropertyNotFound Then ' Property not in collection
        Set prp = tdf.CreateProperty("FrozenColumns", DB_Integer, 1)
        tdf.Properties.Append prp
        Resume Frozen_Bye
    End If
End Sub
FullName Property

Sets or returns the full path (including file name) of a specific object. Read/write String for the AccessObject object; read-only String for the CodeProject and CurrentProject objects.

expression.FullName

expression Required. An expression that returns one of the objects in the Applies To list.
Example

The following example iterates through the **AllDataAccessPages** collection and returns the name of the link and the path of each data access page in the current project.

```vba
Sub PrintDAPLocationInfo()
    Dim dapObject As AccessObject
    For Each dapObject In CurrentProject.AllDataAccessPages
        Debug.Print "The '" & dapObject.Name & _
        " is located at: "; dapObject.FullName
    Next dapObject
End Sub
```
**FullPath Property**

The **FullPath** property returns a **string** containing the path and file name of the referenced **type library**.

`expression.FullPath`

*expression*  Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **FullPath** property is available only by using Visual Basic and is read-only.

Type libraries reside in files. The following table shows the file extensions for files that commonly contain type libraries.

<table>
<thead>
<tr>
<th>File extension</th>
<th>Type of file</th>
</tr>
</thead>
<tbody>
<tr>
<td>.olb, .tlb</td>
<td>Type library file</td>
</tr>
<tr>
<td>.adp, .ade, .mdb, .mda, .mde</td>
<td>Database</td>
</tr>
<tr>
<td>.exe, .dll</td>
<td>Executable file</td>
</tr>
<tr>
<td>.ocx</td>
<td>ActiveX control</td>
</tr>
</tbody>
</table>

If the **IsBroken** property setting of a **Reference** object is **True**, reading the **FullPath** property generates an error.
**Example**

The following example prints the value of the **FullPath**, **GUID**, **IsBroken**, **Major**, and **Minor** properties for each **Reference** object in the **References** collection:

```vbscript
Sub ReferenceProperties()
    Dim ref As Reference

    ' Enumerate through References collection.
    For Each ref In References
        ' Check IsBroken property.
        If ref.IsBroken = False Then
            Debug.Print "Name: ", ref.Name
            Debug.Print "FullPath: ", ref.FullPath
        Else
            Debug.Print "GUIDs of broken references:"
            Debug.Print ref.GUID
        End If
    Next ref
End Sub
```
FuriganaControl Property

Language-specific information

You can use the **FuriganaControl** property to indicate a target control and automatically create furigana for text entered in a text box. Read/write **String**.

`expression.FuriganaControl`

`expression`  Required. An expression that returns one of the objects in the Applies To list.
Remarks

The setting value is the name of the control for entering furigana.

You can set this property by using the property sheet, a macro, or Visual Basic.

If the FuriganaControl property is set in the control, furigana will automatically be created, and can be displayed in a separately designated control. Only if a control name in the same form is set with the FuriganaControl property will the form run properly when executed. If text is entered in a control other than the designated control name in the same form, an error will occur. The type of furigana characters is determined by the IMEMode/KanjiConversionMode property settings in the control.

FuriganaControl property in ADP

When you use FuriganaControl property in ADP file, be sure to change the data type from CHAR/NCHAR to VARCHAR/NVARCHAR. Otherwise, you cannot insert any furigana string into the target field. The FuriganaControl property inserts furigana strings to an existing target field, but if the data type definition of the field stays as CHAR/NCHAR, any string insertion fails because the field length is fixed, which result in an error message.

Note If you enter text in the target control, the furigana of the newly entered text is automatically added to the end of the designated target control content. Even if the text of the target control is revised or deleted, the characters before the change in the target control will not be revised or deleted. Changing the content of the target control revises the text in the furigana control as necessary. The FuriganaControl property will not run if text is pasted into the target control.
GridX Property

You can use the GridX property (along with the GridY property) to specify the horizontal and vertical divisions of the alignment grid in form Design view and report Design view. Read/write Integer.

expression.GridX

expression Required. An expression that returns one of the objects in the Applies To list.
Remarks

Enter an integer between 1 and 64 representing the number of subdivisions per unit of measurement. If the **Measurement system** box is set to U.S. on the **Numbers** tab of the **Regional Options** dialog box of Windows Control Panel, the default setting is 24 for the **GridX** property (horizontal) and 24 for the **GridY** property (vertical).

You can set this property by using the **property sheet**, a **macro**, or **Visual Basic**.

In Visual Basic, you set this property by using a **numeric expression**.

The **GridX** and **GridY** properties provide control over the placement and alignment of objects on a form or report. You can adjust the grid for greater or lesser precision. To see the grid, click **Grid** on the **View** menu. If the setting for either the **GridX** or **GridY** properties is greater than 24, the grid points disappear from view (although the grid lines are still displayed).
GridY Property

You can use the **GridY** property (along with the **GridX** property) to specify the horizontal and vertical divisions of the alignment grid in form Design view and report Design view. Read/write **Integer**.

```
expression.GridY
```

*expression* Required. An expression that returns one of the objects in the Applies To list.
Remarks

Enter an integer between 1 and 64 representing the number of subdivisions per unit of measurement. If the Measurement system box is set to U.S. on the Numbers tab of the Regional Options dialog box of Windows Control Panel, the default setting is 24 for the GridX property (horizontal) and 24 for the GridY property (vertical).

You can set this property by using the property sheet, a macro, or Visual Basic.

In Visual Basic, you set this property by using a numeric expression.

The GridX and GridY properties provide control over the placement and alignment of objects on a form or report. You can adjust the grid for greater or lesser precision. To see the grid, click Grid on the View menu. If the setting for either the GridX or GridY properties is greater than 24, the grid points disappear from view (although the grid lines are still displayed).
GroupFooter Property

You can use the GroupFooter property (along with the GroupHeader property) to create a group header or a group footer for a selected field or expression in a report. Read/write Boolean.

expression.GroupFooter

expression  Required. An expression that returns one of the objects in the Applies To list.
**Remarks**

You can use group headers and footers to label or summarize data in a group of records. For example, if you set the **GroupHeader** property to Yes for the Categories field, each group of products will begin with its category name.

The **GroupHeader** and **GroupFooter** properties use the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>Creates a group header or footer.</td>
</tr>
<tr>
<td>No</td>
<td>(Default) Doesn't create a group header or footer.</td>
</tr>
</tbody>
</table>

You set these properties in the **Sorting And Grouping** box.

You can set these properties only in **report Design view**.

**Note** You can't set or refer to these properties directly in Visual Basic. To create a group header or footer for a field or **expression** in Visual Basic, use the **CreateGroupLevel** method.

To set the grouping properties— **GroupOn**, **GroupInterval**, and **KeepTogether** — to other than their default values, you must first set the **GroupHeader** or **GroupFooter** property or both to Yes for the selected field or expression.
GroupHeader Property

You can use the GroupHeader property (along with the GroupFooter property) to create a group header or a group footer for a selected field or expression in a report. Read/write Boolean.

expression.GroupHeader

expression  Required. An expression that returns one of the objects in the Applies To list.
Remarks

You can use group headers and footers to label or summarize data in a group of records. For example, if you set the GroupHeader property to Yes for the Categories field, each group of products will begin with its category name.

The GroupHeader and GroupFooter properties use the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>Creates a group header or footer.</td>
</tr>
<tr>
<td>No</td>
<td>(Default) Doesn't create a group header or footer.</td>
</tr>
</tbody>
</table>

You set these properties in the Sorting And Grouping box.

You can set these properties only in report Design view.

Note  You can't set or refer to these properties directly in Visual Basic. To create a group header or footer for a field or expression in Visual Basic, use the CreateGroupLevel method.

To set the grouping properties— GroupOn, GroupInterval, and KeepTogether — to other than their default values, you must first set the GroupHeader or GroupFooter property or both to Yes for the selected field or expression.
Show All
GroupInterval Property

You can use the GroupInterval property with the GroupOn property to specify how records are grouped in a report. Read/write Long.

expression.GroupInterval

expression Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **GroupInterval** property specifies an interval value that records are grouped by. This interval differs depending on the data type and **GroupOn** property setting of the field or expression you're grouping on. For example, you can set the **GroupInterval** property to 1 if you want to group records by the first character of a Text field, such as ProductName.

The **GroupInterval** property settings are Long values that depend on the field's data type and its **GroupOn** property setting. The default **GroupInterval** setting is 1.

You can set this property by using the **Sorting And Grouping** box, a macro, or **Visual Basic**.

You can set the **GroupInterval** property only in **report Design view** or in the **Open** event procedure of a report.

Here are examples of **GroupInterval** property settings for different field data types.

<table>
<thead>
<tr>
<th>Field data type</th>
<th>GroupOn setting</th>
<th><strong>GroupInterval</strong> setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>Each value</td>
<td>(Default) Set to 1.</td>
</tr>
<tr>
<td>Text</td>
<td>Prefix characters</td>
<td>Set to 3 for grouping by the first three characters in the field (for example, Chai, Chartreuse, and Chang would be grouped together).</td>
</tr>
<tr>
<td>Date/Time Week</td>
<td></td>
<td>Set to 2 to return data in biweekly groups.</td>
</tr>
<tr>
<td>Date/Time Hour</td>
<td></td>
<td>Set to 12 to return data in half-day groups.</td>
</tr>
</tbody>
</table>

Note To set the **GroupInterval** property to a value other than its default setting (1), you must first set the **GroupHeader** or **GroupFooter** property or both to Yes for the selected field or expression.
Example

The following example sets the **SortOrder** and grouping properties for the first group level in the Products By Category report to create an alphabetical list of products.

Private Sub Report_Open(Cancel As Integer)
    ' Set SortOrder property to ascending order.
    Me.GroupLevel(0).SortOrder = False
    ' Set GroupOn property.
    Me.GroupLevel(0).GroupOn = 1
    ' Set GroupInterval property to 1.
    Me.GroupLevel(0).GroupInterval = 1
    ' Set KeepTogether property to With First Detail.
    Me.GroupLevel(0).KeepTogether = 2
End Sub
GroupLevel Property

You can use the `GroupLevel` property in Visual Basic to refer to the group level you are grouping or sorting on in a report. Read-only `GroupLevel` object.

`expression.GroupLevel(Index)`

`expression` Required. An expression that returns one of the objects in the Applies To list.

`Index` Required `Long`. The group level, starting with 0. The first field or `expression` you group on is group level 0, the second is group level 1, and so on.
Remarks

The **GroupLevel** property setting is an array in which each entry identifies a group level. You can have up to 10 group levels (0 to 9).

The following sample settings show how you use the **GroupLevel** property to refer to a group level.

<table>
<thead>
<tr>
<th>Group level (0)</th>
<th>Refers to</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>GroupLevel</strong></td>
<td>The first field or expression you sort or group on.</td>
</tr>
<tr>
<td><strong>GroupLevel</strong></td>
<td>The second field or expression you sort or group on.</td>
</tr>
<tr>
<td><strong>GroupLevel</strong></td>
<td>The third field or expression you sort or group on.</td>
</tr>
</tbody>
</table>

**Note** You can use this property only by using Visual Basic to set the **SortOrder**, **GroupOn**, **GroupInterval**, **KeepTogether**, and **ControlSource** properties. You set these properties in the **Open** event procedure of a report.

In reports, you can group or sort on more than one field or expression. Each field or expression you group or sort on is a group level.

You specify the fields and expressions to sort and group on by using the **CreateGroupLevel** method.

If a group is already defined for a report (the **GroupLevel** property is set to 0), then you can use the **ControlSource** property to change the group level in the report's Open event procedure.
Example

The following code changes the **ControlSource** property to a value contained in the **txtPromptYou** text box on the open form named **SortForm**:

```vba
Private Sub Report_Open(Cancel As Integer)
    Me.GroupLevel(0).ControlSource = Forms!SortForm!txtPromptYou
End Sub
```
GroupOn Property

You can use the `GroupOn` property in a report to specify how to group data in a field or `expression` by `data type`. For example, this property lets you group a Date field by month. Read/write `Integer`.

`expression.GroupOn`

`expression`  Required. An expression that returns one of the objects in the Applies To list.
**Remarks**

The **GroupOn** property settings available for a field depend on its data type, as the following table shows. For an expression, all of the settings are available. The default setting for all data types is Each Value.

<table>
<thead>
<tr>
<th>Field data type</th>
<th>Setting</th>
<th>Groups records with</th>
<th>Visual Basic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text</td>
<td>(Default)</td>
<td>The same value in the field or expression.</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Each Value</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Prefix Characters</td>
<td>The same first (n) number of characters in the field or expression.</td>
<td>1</td>
</tr>
<tr>
<td>Date/Time</td>
<td>(Default)</td>
<td>The same value in the field or expression.</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Each Value</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Year</td>
<td>Dates in the same calendar year.</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Qtr</td>
<td>Dates in the same calendar quarter.</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>Month</td>
<td>Dates in the same month.</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>Week</td>
<td>Dates in the same week.</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>Day</td>
<td>Dates on the same date.</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>Hour</td>
<td>Times in the same hour.</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>Minute</td>
<td>Times in the same minute.</td>
<td>8</td>
</tr>
<tr>
<td>AutoNumber, Currency, Number</td>
<td>(Default)</td>
<td>The same value in the field or expression.</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Each Value</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Interval</td>
<td>Values within an interval you specify.</td>
<td>9</td>
</tr>
</tbody>
</table>

**Note** You can set the **GroupOn** property by using the **Sorting And Grouping** box, a **macro**, or **Visual Basic**.

In Visual Basic, you set this property in the **Open** event procedure of a report.

To set the **GroupOn** property to a value other than Each Value, you must first set the **GroupHeader** or **GroupFooter** property or both to Yes for the selected field or expression.
Example

The following example sets the **SortOrder** and grouping properties for the first
group level in the Products By Category report to create an alphabetical list of
products.

```vba
Private Sub Report_Open(Cancel As Integer)
    ' Set SortOrder property to ascending order.
    Me.GroupLevel(0).SortOrder = False
    ' Set GroupOn property.
    Me.GroupLevel(0).GROUPON = 1
    ' Set GroupInterval property to 1.
    Me.GroupLevel(0).GroupInterval = 1
    ' Set KeepTogether property to With First Detail.
    Me.GroupLevel(0).KeepTogether = 2
End Sub
```
You can use the **GrpKeepTogether** property to specify whether groups in a multiple column report that have their **KeepTogether** property for a group set to Whole Group or With First Detail will be kept together by page or by column. Read/write **Byte**.

**expression**.**GrpKeepTogether**

**expression**  Required. An expression that returns one of the objects in the Applies To list.
Remarks

The `GrpKeepTogether` property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Per Page</td>
<td>0</td>
<td>Groups are kept together by page.</td>
</tr>
<tr>
<td>Per Column</td>
<td>1 (Default)</td>
<td>(Default) Groups are kept together by column.</td>
</tr>
</tbody>
</table>

**Note** You can set the `GrpKeepTogether` property by using the report's property sheet, a macro, or Visual Basic.

This property can be set only in report Design view.

You can use this property to specify whether all the data for a group will appear in the same column. For example, if you have a list of employees by department in a multiple-column format, you can use this property to keep all members of the same department in the same column.

The `GrpKeepTogether` property setting has no effect if the `KeepTogether` property for a group is set to No.
HasContinued Property

You can use the HasContinued property to determine if part of the current section begins on the previous page. Read/write Boolean.

expression.HasContinued

expression  Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **HasContinued** property is set by Microsoft Access and is read-only in all views.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>Part of the current section has been printed on the previous page.</td>
</tr>
<tr>
<td>False</td>
<td>Part of the current section hasn't been printed on the previous page.</td>
</tr>
</tbody>
</table>

**Note**  You can get the value of the **HasContinued** property by using a macro or Visual Basic.

You can use this property to determine whether to show or hide certain controls depending on the value of the property. For example, you may have a hidden label in a page header containing the text "Continued from previous page". If the value of the **HasContinued** property is **True**, you can make the hidden label visible.
HasData Property

You can use the **HasData** property to determine if a report is **bound** to an empty *recordset*. Read/write **Long**.

expression.**HasData**

*expression* Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **HasData** property is set by Microsoft Access. The value of this property can be read only while printing or while in **Print Preview**.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>–1</td>
<td>The object has data.</td>
</tr>
<tr>
<td>0</td>
<td>The object doesn't have data.</td>
</tr>
<tr>
<td>1</td>
<td>The object is <strong>unbound</strong>.</td>
</tr>
</tbody>
</table>

**Note**  You can get the value of the **HasData** property by using a **macro** or **Visual Basic**.

You can use this property to determine whether to hide a subreport that has no data. For example, the following expression hides the subreport control when its report has no data.

```plaintext
```
HasModule Property

You can use the HasModule property to specify or determine whether a form or report has a class module. Setting this property to No can improve the performance and decrease the size of your database. Read/write Boolean.

expression.HasModule

expression  Required. An expression that returns one of the objects in the Applies To list.
Remarks

The HasModule property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>True</td>
<td>The form or report has a class module.</td>
</tr>
<tr>
<td>No</td>
<td>False</td>
<td>(Default) The form or report doesn't have a class module.</td>
</tr>
</tbody>
</table>

Note  You can set the HasModule property by using the form or report's property sheet, a macro, or Visual Basic.

The HasModule property can be set only in form or report Design view but can be read in any view.

Forms or reports that have the HasModule property set to No are considered lightweight objects. Lightweight objects are smaller and typically load and display faster than objects with associated class modules. In many cases, a form or report doesn't need to use event procedures and doesn't require a class module.

If your application uses a switchboard form to navigate to other forms, instead of using command buttons with event procedures, you can use a command button with a macro or hyperlink. For example, to open the Employees form from a command button on a switchboard, you can set the control's HyperlinkSubAddress property to "Form Employees".

Lightweight objects don't appear in the Object Browser and you can't use the New keyword to create a new instance of the object. A lightweight form or report can be used as a subform or subreport and will appear in the Forms or Reports collection. Lightweight objects support the use of macros, and public procedures that exist in standard modules when called from the object's property sheet.

Microsoft Access sets the HasModule property to True as soon as you attempt to view an object's module, even if no code is actually added to the module. For example, selecting Code from the View menu for a form in Design view causes Microsoft Access to add a class module to the Form object and set its
**HasModule** property to **True**. You can add a class module to an object in the same way by setting the **HasModule** property to Yes in an object's property sheet.

**Warning** If you set the **HasModule** property to No by using an object's property sheet or set it to **False** by using Visual Basic, Microsoft Access deletes the object's class module and any code it may contain.

When you use a method of the **Module** object or refer to the **Module** property for a form or report in Design view, Microsoft Access creates the associated module and sets the object's **HasModule** property to **True**. If you refer to the **Module** property of a form or report at run-time and the object has its **HasModule** property set to **False**, an error will occur.

Objects created by using the **CreateForm** or **CreateReport** methods are lightweight by default.
Show All
**Height Property**

You can use the **Height** property (along with the **Width** property) to size an object to specific dimensions. Read/write **Integer** for all objects in the Applies To list except for the **Report** object, which is a read/write **Long**.

expression.Height

**expression**   Required. An expression that returns one of the above objects.

Remarks

The **Height** property applies only to **form sections** and **report sections**, not to **forms** and **reports**.

Enter a number for the desired height in the current unit of measurement. To use a unit of measurement different from the setting in the **Regional Options** dialog box in Windows Control Panel, specify the unit, such as cm or in (for example, 5 cm or 3 in).

You can set this property by using the object's **property sheet**, a **macro**, or **Visual Basic**.

For controls, you can set the default for this property by using the **default control style** or the **DefaultControl** method in Visual Basic.

In Visual Basic, use a **numeric expression** to set the value of this property. Values are expressed in **twips**.

For report sections, you can't use a macro or Visual Basic to set the **Height** property when you print or preview a report. For report controls, you can set the **Height** property when you print or preview a report only by using a macro or an **event procedure** specified in a section's **OnFormat** event property setting.

You can't set this property for an object once the print process has started. For example, attempting to set the **Height** property in a report's **Print** event generates an error.
Microsoft Access automatically sets the **Height** property when you create or size a control or when you size a window in **form Design View** or **report Design view**.

The height of sections is measured from the inside of their borders. The height of controls is measured from the center of their borders so controls with different border widths align correctly. The margins for forms and reports are set in the **Page Setup** dialog box, available by clicking **Page Setup** on the **File** menu.

**Note** To set the left and top location of an object, use the **Left** and **Top** properties.
Example

The following code resizes a command button to a 1-inch by 1-inch square button (the default unit of measurement in Visual Basic is twips; 1440 twips equals one inch):

```vbs
Me!cmdSizeButton.Height = 1440  ' 1440 twips = 1 inch.
Me!cmdSizeButton.Width = 1440
```
HelpContextId Property

The **HelpContextID** property specifies the context ID of a topic in the custom Help file specified by the **HelpFile** property setting. Read/write **Long**.

\[ \text{expression}.\text{HelpContextId} \]

**expression** Required. An expression that returns one of the objects in the Applies To list.
Remarks

Note If you enter the context ID of the Help file topic as a positive number, the help topic will display in a "full" help topic window. If you add a minus sign ("-"), in front of the context ID, the help topic will be displayed in a "pop-up" window. It is important to note the the context id does not have to have a negative number when authored in Microsoft Help Workshop. You must add the minus sign when setting the property to make the topic display in the pop-up window.

You can create a custom Help file to document forms, reports, or applications you create with Microsoft Access.

When you press the F1 key in Form view, Microsoft Access calls the Microsoft Help Workshop or Microsoft HTML Help Workshop application, loads the custom Help file specified by the HelpFile property setting for the form or report, and displays the Help topic specified by the HelpContextID property setting.

If a control's HelpContextID property setting is 0 (the default), Microsoft Access uses the form's HelpContextID and HelpFile properties to identify the Help topic to display. If you press F1 in a view other than Form view or if the HelpContextID property setting for both the form and the control is 0, a Microsoft Access Help topic is displayed.
Example

This example uses the **HelpContext** property of the **Err** object to show the Visual Basic Help topic for the **overflow** error.

```vba
Dim Msg
Err.Clear
On Error Resume Next
Err.Raise 6 ' Generate "Overflow" error.
If Err.Number <> 0 Then
    Msg = "Press F1 or HELP to see " & Err.HelpFile & " topic for" & 
    " the following HelpContext: " & Err.HelpContext
    MsgBox Msg, , "Error: " & Err.Description, Err.HelpFile, _
    Err.HelpContext
End If
```
HelpFile Property

The name of a help file associated with a form or report. Read/write String.

expression.HelpFile

expression  Required. An expression that returns one of the objects in the Applies To list.
Remarks

In Microsoft Access, you can use the *Toolbar Control Properties* dialog box for command bar controls to set the *HelpFile* property for a control on a command bar. Display the *Customize* dialog box by pointing to *Toolbars* on the *View* menu and clicking *Customize*. For menu bar and toolbar controls, display the menu bar or toolbar, and then right-click the control whose *HelpFile* property you want to set. For shortcut menu controls, select the *Shortcut Menus* check box on the *Toolbars* tab of the *Customize* dialog box, then find the shortcut menu control you want on the menu bar that's displayed and right-click the control. Click the *Properties* command. Enter the name of the Help file you want in the *Help File* box.
Example

This example adds a custom command bar with a combo box that tracks stock data. The example also specifies the Help topic to be displayed for the combo box when the user presses SHIFT+F1.

    Set myBar = CommandBars._
        .Add(Name:="Custom", Position:=msoBarTop, _
             Temporary:=True)
    With myBar
        .Controls.Add Type:=msoControlComboBox, ID:=1
        .Visible = True
    End With
    With CommandBars("Custom").Controls(1)
        .AddItem "Get Stock Quote", 1
        .AddItem "View Chart", 2
        .AddItem "View Fundamentals", 3
        .AddItem "View News", 4
        .Caption = "Stock Data"
        .DescriptionText = "View Data For Stock"
        .HelpFile = "C:\corphelp\custom.hlp"
        .HelpContextID = 47
    End With
HideDuplicates Property

You can use the **HideDuplicates** property to hide a control on a report when its value is the same as in the preceding record. Read/write **Boolean**.

**expression**.HideDuplicates

**expression**  Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **HideDuplicates** property applies only to controls (check box, combo box, list box, option button, option group, text box, toggle button) on a report.

The property doesn't apply to check boxes, option buttons, or toggle buttons when they appear in an option group. It does apply to the option group itself.

The **HideDuplicates** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>True</td>
<td>If the value of a control or the data it contains is the same as in the preceding record, the control is hidden.</td>
</tr>
<tr>
<td>No</td>
<td>False</td>
<td>(Default) The control is visible regardless of the value in the preceding record.</td>
</tr>
</tbody>
</table>

**Note** You can set this property by using the control's **property sheet**, a **macro**, or **Visual Basic**.

You can set the **HideDuplicates** property only in **report Design view**.

You can use the **HideDuplicates** property to create a grouped report by using only the detail section rather than a **group header** and the detail section.
Example

The following example returns the **HideDuplicates** property setting for the CategoryName text box and assigns the value to the `intCurVal` variable.

```vbnet
Dim intCurVal As Integer
intCurVal = Me!CategoryName.HideDuplicates
```
**HorizontalDatasheetGridlineStyle Property**

Returns or sets a **Byte** indicating the line style to use for horizontal gridlines on the specified datasheet. Read/write.

*expression.HorizontalDatasheetGridlineStyle*

*expression*  Required. An expression that returns one of the objects in the Applies To list.
**Remarks**

Valid values are between zero and seven. Values greater than seven are ignored; negative values or values above 255 cause an error.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Transparent border</td>
</tr>
<tr>
<td>1</td>
<td>Solid</td>
</tr>
<tr>
<td>2</td>
<td>Dashes</td>
</tr>
<tr>
<td>3</td>
<td>Short dashes</td>
</tr>
<tr>
<td>4</td>
<td>Dots</td>
</tr>
<tr>
<td>5</td>
<td>Sparse dots</td>
</tr>
<tr>
<td>6</td>
<td>Dash-dot</td>
</tr>
<tr>
<td>7</td>
<td>Dash-dot-dot</td>
</tr>
</tbody>
</table>

This property is not supported when saving a form as a data access page.
Example

This example sets the horizontal gridline style on the first open form to dash-dot. The form must be set to Datasheet View in order for you to see the change.

Forms(0).HorizontalDatasheetGridlineStyle = 6
Hyperlink Property

You can use the **Hyperlink** property to return a reference to a **Hyperlink** object. You can use the **Hyperlink** property to access the properties and methods of a **Hyperlink** object associated with a **command button**, **image**, or **label** control. Read-only.

*expression*.Hyperlink

*expression*  Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **Hyperlink** property is available only by using **Visual Basic**.
Example

The CreateHyperlink procedure in the following example sets the hyperlink properties for a command button, label, or image control to the address and subaddress values passed to the procedure. The address setting is an optional argument, because a hyperlink to an object in the current database uses only the subaddress setting. To try this example, create a form with two text box controls (txtAddress and txtSubAddress) and a command button (cmdFollowLink) and paste the following into the Declarations section of the form's module:

```vbnet
Private Sub cmdFollowLink_Click()
    CreateHyperlink Me!cmdFollowLink, Me!txtSubAddress, _
    Me!txtAddress
End Sub

Sub CreateHyperlink(ctlSelected As Control, _
    strSubAddress As String, Optional strAddress As String)
    Dim hlk As Hyperlink
    Select Case ctlSelected.ControlType
        Case acLabel, acImage, acCommandButton
            Set hlk = ctlSelected.Hyperlink
            With hlk
                If Not IsMissing(strAddress) Then
                    .Address = strAddress
                Else
                    .Address = ""
                End If
                .SubAddress = strSubAddress
                .Follow
                .Address = ""
                .SubAddress = ""
            End With
        Case Else
            MsgBox "The control "" & ctlSelected.Name _
                & "" does not support hyperlinks."
    End Select
End Sub
```
HyperlinkAddress Property

You can use the HyperlinkAddress property to specify or determine the path to an object, document, Web page or other destination for a hyperlink associated with a command button, image control, or label control. Read/write String.

expression.HyperlinkAddress

expression Required. An expression that returns one of the objects in the Applies To list.
Remarks

The HyperlinkAddress property is a string expression representing the path to a file (UNC path) or Web page (URL).

You can set the HyperlinkAddress property by using the control's property sheet, a macro, or Visual Basic.

Note When you set the HyperlinkAddress property, you automatically specify the Address property for the Hyperlink object associated with the control.

You can also use the Insert Hyperlink dialog box to set this property by clicking the Build button to the right of the property box in the property sheet.

Note When you create a hyperlink by using the Insert Hyperlink dialog box, Microsoft Access automatically sets the HyperlinkAddress property and HyperlinkSubAddress to the location specified in the Type the file or Web page name box. The HyperlinkSubAddress property can also be set to the location specified in the Select an object in this database box.

If you copy a hyperlink from another application and paste it into a form or report, Microsoft Access creates a label control with its Caption property, HyperlinkAddress property, and HyperlinkSubAddress property automatically set.

When you move the cursor over a command button, image control, or label control whose HyperlinkAddress property is set, the cursor changes to an upward-pointing hand. Clicking the control displays the object or Web page specified by the link.

To open objects in the current database, leave the HyperlinkAddress property blank and specify the object type and object name you want to open in the HyperlinkSubAddress property by using the syntax "objecttype objectname". If you want to open an object contained in another Microsoft Access database, enter the database path and file name in the HyperlinkAddress property and specify the database object to open by using the HyperlinkSubAddress property.
The **HyperlinkAddress** property can contain an absolute or a relative path to a target document. An absolute path is a fully qualified URL or UNC path to a document. A relative path is a path related to the base path specified in the **Hyperlink Base** setting in the *DatabaseName Properties* dialog box (available by clicking **Database Properties** on the **File** menu) or to the current database path. If Microsoft Access can't resolve the **HyperlinkAddress** property setting to a valid URL or UNC path, it will assume you've specified a path relative to the base path contained in the **Hyperlink Base** setting or the current database path.

**Note** When you follow a hyperlink to another Microsoft Access database object, the database Startup properties are applied. For example, if the destination database has a Display form set, that form is displayed when the database opens.

The following table contains examples of **HyperlinkAddress** and **HyperlinkSubAddress** property settings.

<table>
<thead>
<tr>
<th>HyperlinkAddress</th>
<th>HyperlinkSubAddress</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>C:\Program Files\Microsoft Office\Office\Samples\Cajun.htm</td>
<td></td>
<td>The Cajun Delights page in the Access sample applications subdirectory.</td>
</tr>
<tr>
<td>C:\Program Files\Microsoft Office\Office\Samples\Cajun.htm</td>
<td>NewProducts</td>
<td>The &quot;NewProducts&quot; Name tag in the Cajun Delights page.</td>
</tr>
<tr>
<td>C:\Personal\MyResume.doc</td>
<td>References</td>
<td>The bookmark named &quot;References&quot; in the Microsoft Word document &quot;MyResume.doc&quot;.</td>
</tr>
<tr>
<td>C:\Finance\First Quarter.xls</td>
<td>Sheet1!TotalSales</td>
<td>The range named &quot;TotalSales&quot; in the Microsoft Excel spreadsheet &quot;First Quarter.xls&quot;.</td>
</tr>
<tr>
<td>C:\Presentation\NewPlans.ppt</td>
<td>10</td>
<td>The 10th slide in the Microsoft PowerPoint document &quot;NewPlans.ppt&quot;.</td>
</tr>
</tbody>
</table>
HyperlinkColor Property

You can use the HyperlinkColor property to specify or determine the default color of all hyperlinks within the Application object. Read/write AcColorIndex.

AcColorIndex can be one of these AcColorIndex constants.

- acColorIndexAqua
- acColorIndexBlack
- acColorIndexBlue Default.
- acColorIndexBrightGreen
- acColorIndexDarkBlue
- acColorIndexFuschia
- acColorIndexGray
- acColorIndexGreen
- acColorIndexMaroon
- acColorIndexOlive
- acColorIndexRed
- acColorIndexSilver
- acColorIndexTeal
- acColorIndexViolet
- acColorIndexWhite
- acColorIndexYellow

expression.HyperlinkColor

expression Required. An expression that returns one of the objects in the Applies To list.
Remarks

You can set the HyperlinkColor property through the DefaultWebOptions property or the SetOption method by using Visual Basic.

You can set or change the default hyperlink color available in the Web Options dialog box. To display this dialog box, click Options on the Tools menu. Click the General tab and click the Web Pages button.

The default color of a hyperlink is changed to the followed hyperlink color when a hyperlink control has been pressed.

Use the DefaultWebOptions property to identify or set the Application object's DefaultWebOptions object properties.
HyperlinkSubAddress Property

You can use the **HyperlinkSubAddress** property to specify or determine a location within the target document specified by the **HyperlinkAddress** property. Read/write **String**.

expression.HyperlinkSubAddress

expression Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **HyperlinkSubAddress** property can be an object within a Microsoft Access database, a bookmark within a Microsoft Word document, a named range within a Microsoft Excel spreadsheet, a slide within a Microsoft PowerPoint presentation, or a location within an HTML document.

The **HyperlinkSubAddress** property is a *string expression* representing a named location within the target document specified by the **HyperlinkAddress** property.

You can set the **HyperlinkSubAddress** property by using a control's property sheet, a macro, or Visual Basic.

You can also use the Insert Hyperlink dialog box to set this property by clicking the **Build** button to the right of the property box in the property sheet.

**Note** When you create a hyperlink by using the Insert Hyperlink dialog box, Microsoft Access automatically sets the **HyperlinkAddress** property and HyperlinkSubAddress to the location specified in the **Type the file or Web page name** box. The **HyperlinkSubAddress** property is set to the location specified in the **Select an object in this database** box.

When you move the cursor over a command button, image control, or label control whose **HyperlinkSubAddress** property is set, the cursor changes to an upward-pointing hand. Clicking the control displays the object or Web page specified by the link.

To open objects in the current database, leave the **HyperlinkAddress** property blank and specify the object type and object name you want to open in the **HyperlinkSubAddress** property in the format "objecttype objectname". For example, to create a hyperlink for a command button that opens the Employees form you could set the control's **HyperlinkSubAddress** property to "Form Employees". If you want to open an object contained in another Microsoft Access database, enter the database path and file name in the **HyperlinkAddress** property and specify the database object to open by using the **HyperlinkSubAddress** property.
**Note** When you follow a hyperlink to another Microsoft Access database object, the database Startup properties are applied. For example, if the destination database has a Display form set, that form is displayed when the database opens.

The following table contains shows examples of **HyperlinkAddress** and **HyperlinkSubAddress** property settings.

<table>
<thead>
<tr>
<th>HyperlinkAddress</th>
<th>HyperlinkSubAddress</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>C:\Program Files\Microsoft Office\Office\Samples\Cajun.htm</td>
<td></td>
<td>The Cajun Delights page in the Access sample applications subdirectory.</td>
</tr>
<tr>
<td>C:\Program Files\Microsoft Office\Office\Samples\Cajun.htm</td>
<td>NewProducts</td>
<td>The &quot;NewProducts&quot; Name tag in the Cajun Delights page.</td>
</tr>
<tr>
<td>C:\Personal\MyResume.doc</td>
<td>References</td>
<td>The bookmark named &quot;References&quot; in the Microsoft Word document &quot;MyResume.doc&quot;.</td>
</tr>
<tr>
<td>C:\Finance\First Quarter.xls</td>
<td>Sheet1!TotalSales</td>
<td>The range named &quot;TotalSales&quot; in the Microsoft Excel spreadsheet &quot;First Quarter.xls&quot;.</td>
</tr>
<tr>
<td>C:\Presentation\NewPlans.ppt</td>
<td>10</td>
<td>The 10th slide in the Microsoft PowerPoint document &quot;NewPlans.ppt&quot;.</td>
</tr>
</tbody>
</table>
Show All
ImageHeight Property

You can use the ImageHeight property in Visual Basic to determine the height in twips of the picture in an image control. Read/write Long.

expression.ImageHeight

expression Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **ImageHeight** property is an **Integer** value equal to the height of a picture in twips.

This property is read-only in all views.

You can use the **ImageHeight** property together with the **ImageWidth** property to determine the size of a picture in an image control. You could then use this information to change the image control's **Height** and **Width** properties to match the size of the picture displayed.
Example

The following example prompts the user to enter the name of a bitmap and then assigns that bitmap to the **Picture** property of the Image1 image control. The **ImageHeight** and **ImageWidth** properties are used to resize the image control to fit the size of the bitmap.

Sub GetNewPicture(frm As Form)
    Dim ctlImage As Control
    Set ctlImage = frm!Image1
    ctlImage.Picture = InputBox("Enter path and " & 
                         & "file name for new bitmap")
    ctlImage.Height = ctlImage.**ImageHeight**
    ctlImage.Width = ctlImage.**ImageWidth**
End Sub
ImageWidth Property

You can use the **ImageWidth** property in **Visual Basic** to determine the width in **twips** of a picture in an **image control**. Read/write **Long**.

*expression*.**ImageWidth**

*expression*  Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **ImageWidth** property is an **Integer** value equal to the width of a picture in twips.

This property is read-only in all views.

You can use the **ImageWidth** property together with the **ImageHeight** property to determine the size of a picture in an image control. You could then use this information to change the image control's **Height** and **Width** properties to match the size of the picture displayed.
Example

The following example prompts the user to enter the name of a bitmap and then assigns that bitmap to the Picture property of the Image1 image control. The ImageHeight and ImageWidth properties are used to resize the image control to fit the size of the bitmap.

Sub GetNewPicture(frm As Form)
    Dim ctlImage As Control
    Set ctlImage = frm!Image1
    ctlImage.Picture = InputBox("Enter path and " _
                  & "file name for new bitmap")
    ctlImage.Height = ctlImage.ImageHeight
    ctlImage.Width = ctlImage.ImageWidth
End Sub
IMEHold/HoldKanjiConversionMode Property

Language-specific information

You can use the IMEHold/Hold KanjiConversionMode property to show whether the Kanji Conversion Mode is maintained when the control loses the focus.
Remarks

The **IMEHold/Hold KanjiConversionMode** property uses the following settings.

<table>
<thead>
<tr>
<th>Settings</th>
<th>Description</th>
<th>Visual Basic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>Maintains the Kanji Conversion Mode set in the last control.</td>
<td><strong>True</strong></td>
</tr>
<tr>
<td>No</td>
<td>Does not maintain the Kanji Conversion Mode set in the last control (default).</td>
<td><strong>False</strong></td>
</tr>
</tbody>
</table>

You can set this property by using the [property sheet](#), a [macro](#), or [Visual Basic](#).

By setting the **IMEHold/Hold KanjiConversionMode** property, you can designate whether or not the Kanji Conversion Mode is maintained when the control loses the focus. If this property is set to Yes, the Kanji Conversion Mode setting is maintained when the control loses the focus and once that control regains the focus, the Kanji Conversion Mode setting for that control is restored. If this property is set to No (default setting), the Kanji Conversion Mode will be set by the **IMEMode/KanjiConversionMode** property for that control.

**Note** To set the Kanji Conversion Mode when the focus shifts to the control, set the **IMEMode/KanjiConversionMode** property.
Show All
IMEMode/KanjiConversion Property

You can use the **IMEMode** property to set a control's Kanji Conversion Mode when the **focus** shifts to the control.
Remarks

The **IMEMode** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
<th>Visual Basic</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Control</td>
<td>Kanji Conversion Mode not set (default).</td>
<td>0</td>
</tr>
<tr>
<td>On</td>
<td>Turns on Kanji Conversion Mode.</td>
<td>1</td>
</tr>
<tr>
<td>Off</td>
<td>Turns off Kanji Conversion Mode.</td>
<td>2</td>
</tr>
<tr>
<td>Disable</td>
<td>Disables Kanji Conversion Mode.</td>
<td>3</td>
</tr>
<tr>
<td>Hiragana</td>
<td>Sets full pitch hiragana.</td>
<td>4</td>
</tr>
<tr>
<td>Full pitch Katakana</td>
<td>Sets full pitch katakana.</td>
<td>5</td>
</tr>
<tr>
<td>Half pitch Katakana</td>
<td>Sets half pitch katakana.</td>
<td>6</td>
</tr>
<tr>
<td>Full pitch Alpha/Num</td>
<td>Sets full pitch letters/numbers.</td>
<td>7</td>
</tr>
<tr>
<td>Half pitch Alpha/Num</td>
<td>Sets half pitch letters/numbers.</td>
<td>8</td>
</tr>
<tr>
<td>HangulFull</td>
<td>Sets full pitch hangul.</td>
<td>9</td>
</tr>
<tr>
<td>Hangul</td>
<td>Sets half pitch hangul.</td>
<td>10</td>
</tr>
</tbody>
</table>

**Note** You can set this property by using the property sheet, a macro, or Visual Basic.

You can specify the Kanji Conversion Mode when the focus shifts to control by setting the **IMEMode** property. If set to No Control (default) the setting before the focus shifted to that control is used. For any other setting, the Kanji Conversion Mode setting for that control is used. For example, if the **IMEMode** property is set to Off, the Kanji Conversion Mode is turned off, and if the **IMEMode** property is set to On, the Kanji Conversion Mode is turned on. The Kanji Conversion Mode automatically changes each time the focus shifts between controls.

**Note** If set to Disable, the Kanji Conversion Mode settings cannot be changed. If any other setting is used, the Kanji Conversion Mode can be changed, but when the focus changes, the settings are lost. If you want to save the settings before the control loses the focus, set the **IMEHold/HoldKanjiConversionMode** property.
IMESentenceMode Property

Language-specific information

You can use the IMESentenceMode property to specify or determine the IME Sentence Mode of fields of a table or controls of a form that switch when the focus moves.
**Remarks**

The **IMESentenceMode** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
<th>Visual Basic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal</td>
<td>(Default) Set IME Sentence Mode to “Normal” mode.</td>
<td>0</td>
</tr>
<tr>
<td>Plural</td>
<td>Set IME Sentence Mode to “Plural” mode.</td>
<td>1</td>
</tr>
<tr>
<td>Speaking</td>
<td>Set IME Sentence Mode to “Speaking” mode.</td>
<td>2</td>
</tr>
<tr>
<td>No Conversion</td>
<td>Doesn’t set IME Sentence Mode.</td>
<td>3</td>
</tr>
</tbody>
</table>

**Note**  You can set this property by using the property sheet, a macro, or Visual Basic.

**Normal mode**

Use this mode when creating a literary Japanese document.

**Plural mode**

Use this mode when entering name or address data. In this mode, two additional dictionaries are available. The “Biographical/Geographical Dictionary” contains names not covered in the normal dictionary and the “Postal Code Dictionary”, useful in creating addresses. (Factory setting.)

**Speaking mode**

Use this mode when entering data that contains conversational language.

**No Conversion**

In this mode, inputted characters are settled without conversion.
InputMask Property

You can use the **InputMask** property to make data entry easier and to control the values users can enter in a *text box control*. Read/write **String**.

*expression*.InputMask

*expression* Required. An expression that returns one of the objects in the Applies To list.
Remarks

Input masks are helpful for data-entry operations such as an input mask for a Phone Number field that shows you exactly how to enter a new number: (___) ___-____. It is often easier to use the Input Mask Wizard to set the property for you.

The **InputMask** property can contain up to three sections separated by semicolons (;).

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>First</td>
<td>Specifies the input mask itself; for example, !(999) 999-9999. For a list of characters you can use to define the input mask, see the following table. Specifies whether Microsoft Access stores the literal display characters in the table when you enter data. If you use 0 for this section, all literal display characters (for example, the parentheses in a phone number input mask) are stored with the value; if you enter 1 or leave this section blank, only characters typed into the control are stored.</td>
</tr>
<tr>
<td>Second</td>
<td>Specifies the character that Microsoft Access displays for the space where you should type a character in the input mask. For this section, you can use any character; to display an empty string, use a space enclosed in quotation marks (&quot; &quot;).</td>
</tr>
<tr>
<td>Third</td>
<td></td>
</tr>
</tbody>
</table>

In **Visual Basic** you use a string expression to set this property. For example, the following specifies an input mask for a text box control used for entering a phone number:

```vbnet
Forms!Customers!Telephone.InputMask = "(###) ###-####"
```

When you create an input mask, you can use special characters to require that certain data be entered (for example, the area code for a phone number) and that other data be optional (such as a telephone extension). These characters specify the type of data, such as a number or character, that you must enter for each character in the input mask.

You can define an input mask by using the following characters.
<table>
<thead>
<tr>
<th>Character</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Digit (0 to 9, entry required, plus [+] and minus [-] signs not allowed). Digit or space (entry not required, plus and minus signs not allowed).</td>
</tr>
<tr>
<td>9</td>
<td>Digit or space (entry not required; spaces are displayed as blanks while in Edit mode, but blanks are removed when data is saved; plus and minus signs allowed).</td>
</tr>
<tr>
<td>#</td>
<td>Digit or space (entry not required; spaces are displayed as blanks while in Edit mode, but blanks are removed when data is saved; plus and minus signs allowed).</td>
</tr>
<tr>
<td>L</td>
<td>Letter (A to Z, entry required).</td>
</tr>
<tr>
<td>A</td>
<td>Letter or digit (entry required).</td>
</tr>
<tr>
<td>a</td>
<td>Letter or digit (entry optional).</td>
</tr>
<tr>
<td>&amp;</td>
<td>Any character or a space (entry required). Any character or a space (entry optional).</td>
</tr>
<tr>
<td>C</td>
<td>Decimal placeholder and thousand, date, and time separators. (The actual character used depends on the settings in the Regional Settings Properties dialog box in Windows Control Panel).</td>
</tr>
<tr>
<td>. , : ; - /</td>
<td>Causes all characters to be converted to lowercase. Causes all characters to be converted to uppercase. Causes the input mask to display from right to left, rather than from left to right. Characters typed into the mask always fill it from left to right. You can include the exclamation point anywhere in the input mask. Causes the character that follows to be displayed as the literal character (for example, \A is displayed as just A).</td>
</tr>
</tbody>
</table>

**Note** Setting the **InputMask** property to the word "Password" creates a password-entry control. Any character typed in the control is stored as the character but is displayed as an asterisk (*). You use the Password input mask to prevent displaying the typed characters on the screen.

For a control, you can set this property in the control's **property sheet**. For a field in a table, you can set the property in **table Design view** (in the Field Properties section) or in Design view of the **Query window** (in the Field Properties **property sheet**).
You can also set the **InputMask** property by using a [macro](https://www.microsoft.com) or Visual Basic.

When you type data in a field for which you've defined an input mask, the data is always entered in Overtype mode. If you use the BACKSPACE key to delete a character, the character is replaced by a blank space.

If you move text from a field for which you've defined an input mask onto the Clipboard, the literal display characters are copied, even if you have specified that they not be saved with data.

**Note** Only characters that you type directly in a control or combo box are affected by the input mask. Microsoft Access ignores any input masks when you import data, run an action query, or enter characters in a control by setting the control's **Text** property in Visual Basic or by using the **SetValue** action in a macro.

When you've defined an input mask and set the **Format** property for the same field, the **Format** property takes precedence when the data is displayed. This means that even if you've saved an input mask, the input mask is ignored when data is formatted and displayed. The data in the underlying table itself isn't changed; the **Format** property affects only how the data is displayed.
## Example

The following table shows some useful input masks and the type of values you can enter in them.

<table>
<thead>
<tr>
<th>Input mask</th>
<th>Sample values</th>
</tr>
</thead>
<tbody>
<tr>
<td>(000) 000-0000</td>
<td>(206) 555-0248</td>
</tr>
<tr>
<td>(999) 999-9999</td>
<td>(206) 555-0248</td>
</tr>
<tr>
<td>( ) 555-0248</td>
<td></td>
</tr>
<tr>
<td>(000) AAA-AAAA</td>
<td>(206) 555-TELE</td>
</tr>
<tr>
<td>#999</td>
<td>–20</td>
</tr>
<tr>
<td></td>
<td>2000</td>
</tr>
<tr>
<td>&gt;L????L?000L0</td>
<td>GREENGR339M3</td>
</tr>
<tr>
<td></td>
<td>MAY R 452B7</td>
</tr>
<tr>
<td>&gt;L0L 0L0</td>
<td>T2F 8M4</td>
</tr>
<tr>
<td>00000-9999</td>
<td>98115-</td>
</tr>
<tr>
<td></td>
<td>98115-3007</td>
</tr>
<tr>
<td>&gt;L&lt;??????????????</td>
<td>Maria</td>
</tr>
<tr>
<td></td>
<td>Brendan</td>
</tr>
<tr>
<td>SSN 000-00-0000</td>
<td>SSN 555-55-5555</td>
</tr>
<tr>
<td>&gt;LL00000-0000</td>
<td>DB51392-0493</td>
</tr>
</tbody>
</table>
InputParameters Property

You can use the InputParameters property to specify or determine the input parameters that are passed to a SQL statement in the RecordSource property of a form or report or a stored procedure when used as the record source within a Microsoft Access project (.adp). Read/write String.

expression.InputParameters

expression Required. An expression that returns one of the objects in the Applies To list.
**Remarks**

You can set this property by using the [property sheet](#) or [Visual Basic](#).

When used with a **RecordSource** property:

An example **InputParameter** property string used with a SQL statement in the **RecordSource** property would use the following syntax.

```plaintext
state char=[Forms]![formname]![StateList], salesyear smallint=[Forms]![formname]![Enter year of interest]
```

This would result in the state parameter being set to the current value of the StateList control, and the user getting prompted for the salesyear parameter. If there were any other parameters that were not in this list, they would get default values assigned.

The query should be executed with one ? marker for each non-default parameter in the **InputParameter** list.

A refresh or requery command (via menu, keyboard, or Navigation Bar) in Access should trigger a reexecute of the query. Users can do this in code by calling the standard **Recordset.Requery** method. If the value of a parameter is bound to a control on the form, the current value of the control is used at requery time. The query is not automatically reexecuted when the value of the control changes.

When used with a stored procedure:

An example **InputParameter** property string used with stored procedure would be:

```plaintext
@state char=[Forms]![formname]![StateList], @salesyear smallint=[Forms]![formname]![Enter year of interest]
```

This would result in the @state parameter being set to the current value of the StateList control, and the user getting prompted for the @salesyear parameter. If there were any other parameters to the stored proc that were not in this list, they
would get default values assigned.

The stored procedure should be executed using a command string containing the {call } syntax with one ? marker for each non-default parameter in the InputParameter list.

A refresh or requery command (via menu, keyboard, or Navigation Bar) in Access should trigger a reexecuted of the stored procedure. Users can do this in code by calling the standard Recordset.Requery method. If the value of a parameter is bound to a control on the form, the current value of the control is used at requery time. The stored procedure is not automatically reexecuted when the value of the control changes.

This builder dialog is invoked when a stored procedure is first selected as the record source of a form if the stored procedure has any parameters. After initial creation of the InputParameters string, this same dialog is used as a builder for changing the string. In this case however the list of parameters comes from what already exists in the string.

Parameter values are also settable in code using the ActiveX Data Object's (ADO) Command and Parameter objects. If the result returns a result set, a form can be bound to it by setting the form's Recordset property. ADO coding is the only way to handle stored procedures that do not return result sets such as action queries, those that return output parameters, or those that return multiple result sets.
InSelection Property

You can use the InSelection property to determine or specify whether a control on a form in Design view is selected. Read/write Boolean.

expression.InSelection

expression Required. An expression that returns one of the objects in the Applies To list.
Remarks

The InSelection property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>The control is selected.</td>
</tr>
<tr>
<td>False</td>
<td>The control isn't selected.</td>
</tr>
</tbody>
</table>

Note  This property is available only by using a macro or Visual Basic.

When a control is selected, its sizing handles are visible and it can be resized by the user. More than one control can be selected at a time.
Example

The following function uses the **InSelection** property to determine whether the `strControlName` control on a form is selected.

To test this code, paste the `IsControlSelected` function code in the Declarations section of a code module in the Northwind sample database, open the Customers form in Design view, and select the `CompanyName` control. Then enter the following line in the Debug window:

```vba
? IsControlSelected (Forms!Customers, "CompanyName")
```

```vba
Function IsControlSelected(frm As Form, _
    strControlName As String) As Integer
    Dim intI As Integer, ctl As Control
    If frm.CurrentView <> 0 Then
        ' Form is not in Design view.
        Exit Function
    Else
        For intI = 0 To frm.Count - 1
            Set ctl = frm(intI)
            If ctl.InSelection = True Then
                ' Is desired control selected?
                If UCase(ctl.Name) = UCase(strControlName) Then
                    IsControlSelected = True
                    Exit Function
                End If
            Else
                IsControlSelected = False
            End If
        Next intI
    End If
End Function
```
InsideHeight Property

You can use the **InsideHeight** property (along with the **InsideWidth** property) to determine the height and width (in **twips**) of the window containing a **form**. Read/write **Long**.

*expression*.InsideHeight

*expression*  Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **InsideHeight** and **InsideWidth** properties are available only by using a macro or Visual Basic and can be set at any time.

If you want to determine the interior dimensions of the form itself, you use the **Width** property to determine the form width and the sum of the heights of the form's visible sections to determine its height (the **Height** property applies only to form sections, not to forms). The interior of a form is the region inside the form, excluding the scroll bars and the record selectors.

You can also use the **WindowHeight** and **WindowWidth** properties to determine the height and width of the window containing a form.

If a window is maximized, setting these properties doesn't have any effect until the window is restored to its normal size.
Example

The following example shows how to use the **InsideHeight** and **InsideWidth** properties to compare the inside height and width of a form with the height and width of the form's window. If the window's dimensions don't equal the size of the form, then the window is resized to match the form's height and width.

```vba
Sub ResetWindowSize(frm As Form)
    Dim intWindowHeight As Integer
    Dim intWindowWidth As Integer
    Dim intTotalFormHeight As Integer
    Dim intTotalFormWidth As Integer
    Dim intHeightHeader As Integer
    Dim intHeightDetail As Integer
    Dim intHeightFooter As Integer

    ' Determine form's height.
    intHeightHeader = frm.Section(acHeader).Height
    intHeightDetail = frm.Section(acDetail).Height
    intHeightFooter = frm.Section(acFooter).Height
    intTotalFormHeight = intHeightHeader _
                  + intHeightDetail + intHeightFooter
    ' Determine form's width.
    intTotalFormWidth = frm.Width
    ' Determine window's height and width.
    intWindowHeight = frm.InsideHeight
    intWindowWidth = frm.InsideWidth

    If intWindowWidth <> intTotalFormWidth Then
        frm.InsideWidth = intTotalFormWidth
    End If
    If intWindowHeight <> intTotalFormHeight Then
        frm.InsideHeight = intTotalFormHeight
    End If
End Sub
```
InsideWidth Property

You can use the **InsideWidth** property (along with the **InsideHeight** property) to determine the height and width (in **twips**) of the window containing a **form**. Read/write **Long**.

*expression*.InsideWidth

*expression* Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **InsideHeight** and **InsideWidth** properties are available only by using a [macro](#) or [Visual Basic](#) and can be set at any time.

If you want to determine the interior dimensions of the form itself, you use the **Width** property to determine the form width and the sum of the heights of the form's visible sections to determine its height (the **Height** property applies only to form sections, not to forms). The interior of a form is the region inside the form, excluding the scroll bars and the [record selectors](#).

You can also use the **WindowHeight** and **WindowWidth** properties to determine the height and width of the window containing a form.

If a window is maximized, setting these properties doesn't have any effect until the window is restored to its normal size.
Example

The following example shows how to use the `InsideHeight` and `InsideWidth` properties to compare the inside height and width of a form with the height and width of the form's window. If the window's dimensions don't equal the size of the form, then the window is resized to match the form's height and width.

```vba
Sub ResetWindowSize(frm As Form)
    Dim intWindowHeight As Integer
    Dim intWindowWidth As Integer
    Dim intTotalFormHeight As Integer
    Dim intTotalFormWidth As Integer
    Dim intHeightHeader As Integer
    Dim intHeightDetail As Integer
    Dim intHeightFooter As Integer

    ' Determine form's height.
    intHeightHeader = frm.Section(acHeader).Height
    intHeightDetail = frm.Section(acDetail).Height
    intHeightFooter = frm.Section(acFooter).Height
    intTotalFormHeight = intHeightHeader + intHeightDetail + intHeightFooter

    ' Determine form's width.
    intTotalFormWidth = frm.Width

    ' Determine window's height and width.
    intWindowHeight = frm.InsideHeight
    intWindowWidth = frm.InsideWidth

    If intWindowWidth <> intTotalFormWidth Then
        frm.InsideWidth = intTotalFormWidth
    End If
    If intWindowHeight <> intTotalFormHeight Then
        frm.InsideHeight = intTotalFormHeight
    End If
End Sub
```
InsufficientPermissions Property

Returns a **DependencyObjects** collection that contains a collection of objects to which you don't have sufficient permissions to check for dependencies. Read-only.

expression.InsufficientPermissions

*expression* Required. An expression that returns a **DependencyInfo** object.
Remarks

The **DependencyObjects** collection returned by this property contains a collection of **AccessObject** objects.
IsBroken Property

The **IsBroken** property returns a **Boolean** value indicating whether a **Reference** object points to a valid reference in the Windows Registry. Read-only **Boolean**.

`expression.IsBroken`

`expression` Required. An expression that returns one of the objects in the Applies To list.
Remarks

The `IsBroken` property is available only by using Visual Basic and is read-only.

The default value of the `IsBroken` property is `False`. The `IsBroken` property returns `True` only if the `Reference` object no longer points to a valid reference in the Registry.

By evaluating the `IsBroken` property, you can determine whether or not the file associated with a particular `Reference` object has been moved to a different directory or deleted.

If the `IsBroken` property is `True`, Microsoft Access generates an error when you try to read the `Name` or `FullPath` properties.
Example

The following example prints the value of the FullPath, GUID, IsBroken, Major, and Minor properties for each Reference object in the References collection:

Sub ReferenceProperties()
    Dim ref As Reference

    ' Enumerate through References collection.
    For Each ref In References
        ' Check IsBroken property.
        If ref.IsBroken = False Then
            Debug.Print "Name: ", ref.Name
            Debug.Print "FullPath: ", ref.FullPath
        Else
            Debug.Print "GUIDs of broken references:",
            Debug.Print ref.GUID
        EndIf
    Next ref
End Sub
IsCompiled Property

The **IsCompiled** property returns a **Boolean** value indicating whether the Visual Basic **project** is in a compiled state. Read-only **Boolean**.

*expression*.IsCompiled

*expression*  Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **IsCompiled** property is available only by using Visual Basic.

The **IsCompiled** property returns **True** if the project is in a compiled state.

The **IsCompiled** property of the **Application** object is **False** when the project has never been fully compiled, if a module has been added, edited, or deleted after compilation, or if a module hasn't been saved in a compiled state.
IsConnected Property

You can use the `IsConnected` property to determine if the `CurrentProject` or `CodeProject` object is currently connected. Read-only `Boolean`.

(expression).IsConnected

`expression` Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **IsConnected** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>True</td>
<td>The <strong>CurrentProject</strong> or <strong>CodeProject</strong> object is currently connected.</td>
</tr>
<tr>
<td>No</td>
<td>False</td>
<td>The <strong>CurrentProject</strong> or <strong>CodeProject</strong> object is not connected</td>
</tr>
</tbody>
</table>

**Note** The **IsConnected** property is available only by using **Visual Basic**.
IsHyperlink Property

You can use the IsHyperlink property to specify or determine if the data contained in a text box or combo box is a hyperlink. Read/write Boolean.

expression.IsHyperlink

expression Required. An expression that returns one of the objects in the Applies To list.
Remarks

The IsHyperlink property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>True</td>
<td>The data displayed is a hyperlink.</td>
</tr>
<tr>
<td>No</td>
<td>False</td>
<td>The data displayed is not a hyperlink.</td>
</tr>
</tbody>
</table>

**Note**  You can set the IsHyperlink property by using the property sheet, a macro, or Visual Basic.
IsLoaded Property

You can use the **IsLoaded** property to determine if an **AccessObject** is currently loaded. Read-only **Boolean**.

`expression.IsLoaded`

`expression`  Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **IsLoaded** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>True</td>
<td>The specified <strong>AccessObject</strong> is loaded.</td>
</tr>
<tr>
<td>No</td>
<td>False</td>
<td>The specified <strong>AccessObject</strong> is not loaded.</td>
</tr>
</tbody>
</table>

**Note** The **IsLoaded** property is available only by using [Visual Basic](https://msdn.microsoft.com/en-us/library/aa212670(VS.60).aspx) and is read-only.
Example

This procedure illustrates how to use VBA code to add text to a data access page. The following information is supplied in the arguments to this procedure:

strPageName The name of an existing data access page.
strID The ID property (attribute) for the tag that contains the text you want to work with.
strText The text to insert.
blnReplace Whether to replace existing text in the tag.

Function DAPInsertText(strPageName As String, _
                      strID As Variant, strText As String, _
                      Optional blnReplace As Boolean = True) As Boolean
    Dim blnWasLoaded As Boolean
    On Error GoTo DAPInsertText_Err
    ' Determine if the page exists and whether it is currently open. If not open then open it in design view.
    If DAPExists(strPageName) = True Then
        If CurrentProject.AllDataAccessPages(strPageName) . IsLoaded = False Then
            blnWasLoaded = False
            With DoCmd
                .Echo False
                .OpenDataAccessPage strPageName, _
                acDataAccessPageDesign
            End With
        Else
            blnWasLoaded = True
        End If
    Else
        DAPInsertText = False
        Exit Function
    End If
    ' Add the new text to the specified tag.
    With DataAccessPages(strPageName).Document
        If blnReplace = True Then
            .All(strID).innerText = strText
        Else
.All(strID).innerText = .All(strID).innerText & strText
End If
' Make sure the text is visible.
With .All(strID).Style
    If .display = "none" Then .display = ""
End With
End With

' Clean up after yourself.
With DoCmd
    If blnWasLoaded = True Then
        .Save
    Else
        .Close acDataAccessPage, strPageName, acSaveYes
    End If
End With
DAPInsertText = True
DAPInsertText_End:
    DoCmd.Echo True
    Exit Function
DAPInsertText_Err:
    MsgBox "Error #" & Err.Number & ": " & Err.Description
    DAPInsertText = False
    Resume DAPInsertText_End
End Function
IsMissing Property

Returns **True** if the specified `smart tag` is not installed or is installed incorrectly. Read-only **Boolean**.

`expression.IsMissing`

`expression` Required. An expression that returns a `SmartTag` object.
IsVisible Property

You can use the IsVisible property in Visual Basic to determine whether a control on a report is visible. Read/write Boolean.

expression.IsVisible

description Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **IsVisible** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>(Default) The control is visible.</td>
</tr>
<tr>
<td>False</td>
<td>The control isn't visible.</td>
</tr>
</tbody>
</table>

**Note** You can set the **IsVisible** property only in the **Print** event of a report section that contains the control.

You can use the **IsVisible** property together with the **HideDuplicates** property to determine when a control on a report is visible and show or hide other controls as a result. For example, you could hide a **line control** when a **text box** control is hidden because it contains duplicate values.
Example

The following example uses the **IsVisible** property of a text box to control the display of a line control on a report. The report is based on a Products table and uses three controls with the following properties.

<table>
<thead>
<tr>
<th>Properties</th>
<th>Line control</th>
<th>Text box #1</th>
<th>Text box #2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Line0</td>
<td>CategoryID</td>
<td>ProductName</td>
</tr>
<tr>
<td>ControlSource</td>
<td>CategoryID</td>
<td>ProductName</td>
<td></td>
</tr>
<tr>
<td>HideDuplicates</td>
<td>Yes</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Left</td>
<td>0</td>
<td>0</td>
<td>2.0</td>
</tr>
<tr>
<td>Top</td>
<td>0</td>
<td>.1</td>
<td>.1</td>
</tr>
<tr>
<td>Width</td>
<td>4.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
</tbody>
</table>

Paste the following code into the Declarations section of the report module, and then view the report to see the line formatting controlled by the **IsVisible** property:

```vba
Private Sub Detail_Print(Cancel As Integer, PrintCount As Integer)
    If Me!CategoryID.IsVisible Then
        Me!Line0.Visible = True
    Else
        Me!Line0.Visible = False
    End If
End Sub
```
Item Property

The **Item** property returns a specific member of a **collection** either by position or by index. This property is read-only for all objects in the Applies To list except the **ObjectFrame** object, which is read/write.

```
expression.Item(Index)
```

*expression*   Required. An expression that returns one of the above objects.

*Index*      Required **Variant**. An expression that specifies the position of a member of the collection referred to by the *expression* argument. If a **numeric expression**, the *index* argument must be a number from 0 to the value of the collection's **Count** property minus 1. If a **string expression**, the *index* argument must be the name of a member of the collection.
Remarks

The Item property is available only by using Visual Basic.

If the value provided for the index argument doesn't match any existing member of the collection, an error occurs.

The Item property is the default member of a collection, so you don't have to specify it explicitly. For example, the following two lines of code are equivalent:

Debug.Print Modules(0)

Debug.Print Modules.Item(0)
**ItemData Property**

The **ItemData** property returns the data in the **bound column** for the specified row in a **combo box** or **list box**. Read-only **Variant**.

\[ expression.ItemData(Index) \]

*expression* Required. An expression that returns one of the objects in the Applies To list.

*Index* Required **Long**. The row in the combo box or list box containing the data you want to return. Rows in combo and list boxes are indexed starting with zero. For example, to return the item in the sixth row of a combo box, you'd specify 5 for the *rowindex* argument.
Remarks

The **ItemData** property enables you to iterate through the list of entries in a combo box or list box. For example, suppose you wanted to iterate through all of the items in a list box to search for a particular entry. You can use the **ListCount** property to determine the number of rows in the list box, and then use the **ItemData** property to return the data for the bound column in each row.

You can also use the **ItemData** property to return data only from selected rows in a list box. You can iterate through the **ItemsSelected** collection to determine which row or rows in the list box have been selected, and use the **ItemData** property to return the data in those rows. You must set the **MultiSelect** property of the list box to Simple or Extended to enable the user to select more than one row at a time.

**Tip**

You can use the **Column** property to return data from a specified row and column, even if the specified column isn't the bound column.
**Example**

The following example prints the value of the bound column for each selected row in a list box EmployeeList on an Employees form. The list box's **MultiSelect** property must be set to Simple or Extended.

```vba
Sub RowsSelected()
    Dim ctlList As Control, varItem As Variant

    ' Return Control object variable pointing to list box.
    Set ctlList = Forms!Employees!EmployeeList
    ' Enumerate through selected items.
    For Each varItem in ctlList.ItemsSelected
        ' Print value of bound column.
        Debug.Print ctlList.ItemData(varItem)
    Next varItem
End Sub
```
ItemLayout Property

Returns or sets an AcPrintItemLayout constant indicating whether the printer lays columns across, then down, or down, then across. Read/write.

AcPrintItemLayout can be one of these AcPrintItemLayout constants.
acPRHorizontalColumnLayout Columns are laid across, then down.
acPRVerticalColumnLayout Columns are laid down, then across.

expression.ItemLayout

expression  Required. An expression that returns one of the objects in the Applies To list.
Example

The following example sets a variety of printer settings for the form specified in the `strFormname` argument of the procedure.

Sub SetPrinter(strFormname As String)
    DoCmd.OpenForm FormName:=strFormname, view:=acDesign, _
        datamode:=acFormEdit, windowmode:=acHidden
    With Forms(form1).Printer
        .TopMargin = 1440
        .BottomMargin = 1440
        .LeftMargin = 1440
        .RightMargin = 1440
        .ColumnSpacing = 360
        .RowSpacing = 360
        .ColorMode = acPRCMColor
        .DataOnly = False
        .DefaultSize = False
        .ItemSizeHeight = 2880
        .ItemSizeWidth = 2880
        .ItemLayout = acPRVerticalColumnLayout
        .ItemsAcross = 6
        .Copies = 1
        .Orientation = acPRORLandscape
        .Duplex = acPRDPVertical
        .PaperBin = acPRBNAuto
        .PaperSize = acPRPSLetter
        .PrintQuality = acPRPQMedium
    End With
    DoCmd.Close objecttype:=acForm, objectname:=strFormname, _
        Save:=acSaveYes
End Sub
**ItemsAcross Property**

Returns or sets a **Long** indicating the number of columns to print across a page for multiple-column reports or labels. Read/write.

```
expression.ItemsAcross
```

`expression`  Required. An expression that returns one of the objects in the Applies To list.
Example

The following example sets a variety of printer settings for the form specified in the `strFormname` argument of the procedure.

```vba
Sub SetPrinter(strFormname As String)
    DoCmd.OpenForm FormName:=strFormname, view:=acDesign,
                   datamode:=acFormEdit, windowmode:=acHidden
    With Forms(form1).Printer
        .TopMargin = 1440
        .BottomMargin = 1440
        .LeftMargin = 1440
        .RightMargin = 1440
        .ColumnSpacing = 360
        .RowSpacing = 360
        .ColorMode = acPRCMColor
        .DataOnly = False
        .DefaultSize = False
        .ItemSizeHeight = 2880
        .ItemSizeWidth = 2880
        .ItemLayout = acPRVerticalColumnLayout
        .ItemsAcross = 6
        .Copies = 1
        .Orientation = acPRORLandscape
        .Duplex = acPRDPVertical
        .PaperBin = acPRBNAuto
        .PaperSize = acPRPSLetter
        .PrintQuality = acPRPQMedium
    End With
    DoCmd.Close objecttype:=acForm, objectname:=strFormname,
                   Save:=acSaveYes
End Sub
```
**ItemSizeHeight Property**

Returns or sets a *Long* indicating the height of the detail section of a form or report in twips. Read/write.

*expression*.**ItemSizeHeight**

*expression*  Required. An expression that returns one of the objects in the Applies To list.
Remarks

If the DefaultSize property is True, this property is ignored.
Example

The following example sets a variety of printer settings for the form specified in the `strFormname` argument of the procedure.

```
Sub SetPrinter(strFormname As String)
    DoCmd.OpenForm FormName:=strFormname, view:=acDesign, _
        datamode:=acFormEdit, windowmode:=acHidden
    With Forms(form1).Printer
        .TopMargin = 1440
        .BottomMargin = 1440
        .LeftMargin = 1440
        .RightMargin = 1440
        .ColumnSpacing = 360
        .RowSpacing = 360
        .ColorMode = acPRCMColor
        .DataOnly = False
        .DefaultSize = False
        .ItemSizeHeight = 2880
        .ItemSizeWidth = 2880
        .ItemLayout = acPRVerticalColumnLayout
        .ItemsAcross = 6
        .Copies = 1
        .Orientation = acPRORLandscape
        .Duplex = acPRDPVertical
        .PaperBin = acPRBNAuto
        .PaperSize = acPRPSLetter
        .PrintQuality = acPRPQMedium
    End With
    DoCmd.Close objecttype:=acForm, objectname:=strFormname, _
        Save:=acSaveYes
End Sub
```
ItemSizeWidth Property

Returns or sets a **Long** indicating the height of the detail section of a form or report in twips. Read/write.

`expression.ItemSizeWidth`

`expression` Required. An expression that returns one of the objects in the Applies To list.
Remarks

If the `DefaultSize` property is `True`, this property is ignored.
Example

The following example sets a variety of printer settings for the form specified in the strFormname argument of the procedure.

Sub SetPrinter(strFormname As String)
    DoCmd.OpenForm FormName:=strFormname, view:=acDesign, _
    datamode:=acFormEdit, windowmode:=acHidden
    With Forms(form1).Printer
        .TopMargin = 1440
        .BottomMargin = 1440
        .LeftMargin = 1440
        .RightMargin = 1440
        .ColumnSpacing = 360
        .RowSpacing = 360
        .ColorMode = acPRCMColor
        .DataOnly = False
        .DefaultSize = False
        .ItemSizeHeight = 2880
        **.ItemSizeWidth** = 2880
        .ItemLayout = acPRVerticalColumnLayout
        .ItemsAcross = 6
        .Copies = 1
        .Orientation = acPRORLandscape
        .Duplex = acPRDPVertical
        .PaperBin = acPRBNAuto
        .PaperSize = acPRPSLetter
        .PrintQuality = acPRPQMedium
    End With
    DoCmd.Close objecttype:=acForm, objectname:=strFormname, _
    Save:=acSaveYes
End Sub
ItemsSelected Property

You can use the `ItemsSelected` property to return a read-only reference to the hidden `ItemsSelected` collection. This hidden collection can be used to access data in the selected rows of a multiselect list box control.

`expression.ItemsSelected`

`expression` Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **ItemsSelected** collection is unlike other collections in that it is a collection of **Variants** rather than of objects. Each **Variant** is an integer index referring to a selected row in a list box or combo box.

Use the **ItemsSelected** collection in conjunction with the **Column** property or the **ItemData** property to retrieve data from selected rows in a list box or combo box. You can list the **ItemsSelected** collection by using the **For Each...Next** statement.

For example, if you have an Employees list box on a form, you can list the **ItemsSelected** collection and use the control's **ItemData** property to return the value of the bound column for each selected row in the list box.

Tip

To enable multiple selection of rows in a list box, set the control's **MultiSelect** property to Simple or Extended.

The **ItemsSelected** collection has no methods and two properties, the **Count** and **Item** properties.
The following example prints the value of the bound column for each selected row in a Names list box on a Contacts form. To try this example, create the list box and set its **BoundColumn** property as desired and its **MultiSelect** property to Simple or Extended. Switch to Form view, select several rows in the list box, and run the following code:

```vba
Sub BoundData()
    Dim frm As Form, ctl As Control
    Dim varItm As Variant

    Set frm = Forms!Contacts
    Set ctl = frm!Names
    For Each varItm In ctl.ItemsSelected
        Debug.Print ctl.ItemData(varItm)
    Next varItm
End Sub
```

The next example uses the same list box control, but prints the values of each column for each selected row in the list box, instead of only the values in the bound column.

```vba
Sub AllSelectedData()
    Dim frm As Form, ctl As Control
    Dim varItm As Variant, intI As Integer

    Set frm = Forms!Contacts
    Set ctl = frm!Names
    For Each varItm In ctl.ItemsSelected
        For intI = 0 To ctl.ColumnCount - 1
            Debug.Print ctl.Column(intI, varItm)
        Next intI
        Debug.Print
    Next varItm
End Sub
```
KeepTogether Property

KeepTogether property as it applies to the Section object.

You can use the **KeepTogether** property for a section to print a form or report section all on one page. For example, you might have a group of related information that you don't want printed across two pages. The **KeepTogether** property applies only to form and report sections (except page headers and page footers). Read/write **Boolean**.

**expression.KeepTogether**

**expression** Required. An expression that returns one of the above objects.
Remarks

The **KeepTogether** property for a section uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>True</td>
<td>Microsoft Access starts printing the section at the top of the next page if it can't print the entire section on the current page.</td>
</tr>
<tr>
<td>No</td>
<td>False</td>
<td>(Default) Microsoft Access prints as much of the section as possible on the current page and prints the rest on the next page.</td>
</tr>
</tbody>
</table>

You can set this property by using the section's [property sheet](https://www.microsoft.com), a [macro](https://www.microsoft.com), or [Visual Basic](https://www.microsoft.com).

You can set the **KeepTogether** property for a section only in [form Design view](https://www.microsoft.com) or [report Design view](https://www.microsoft.com).

Usually, when a page break occurs while a section is being printed, Microsoft Access continues printing the section on the next page. By using the section's **KeepTogether** property, you can print the section all on one page. If a section is longer than one page, Microsoft Access starts printing it on the next page and continues on the following page.

If the **KeepTogether** property for a group is set to Whole Group or With First Detail and the **KeepTogether** property for a section is set to No, the **KeepTogether** property setting for the section is ignored.

**KeepTogether property as it applies to the GroupLevel object.**

You can use the **KeepTogether** property for a group in a report to keep parts of a group— including the group header, detail section, and group footer— together on the same page. For example, you might want a group header to always be printed on the same page with the first detail section. Read/write **Byte**.

\[\text{expression.KeepTogether} \]

**expression**  Required. An expression that returns one of the above objects.
**Remarks**

The *KeepTogether* property for a group uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>0</td>
<td>(Default) Prints the group without keeping the group header, detail section, and group footer on the same page.</td>
</tr>
<tr>
<td>Whole Group</td>
<td>1</td>
<td>Prints the group header, detail section, and group footer on the same page.</td>
</tr>
<tr>
<td>With First Detail</td>
<td>2</td>
<td>Prints the group header on a page only if it can also print the first detail record.</td>
</tr>
</tbody>
</table>

You can set the *KeepTogether* property for a group by using the Sorting And Grouping box, a macro, or Visual Basic.

In Visual Basic, you set the *KeepTogether* property for a group in report Design view or the Open event procedure of a report by using the GroupLevel property.

To set the *KeepTogether* property for a group to a value other than No, you must set the GroupHeader or GroupFooter property or both to Yes for the selected field or expression.

A group includes the group header, detail section, and group footer. If you set the *KeepTogether* property for a group to Whole Group and the group is too large to fit on one page, Microsoft Access will ignore the setting for that group.

Similarly, if you set this property to With First Detail and either the group header or detail record is too large to fit on one page, the setting will be ignored.

If the *KeepTogether* property for a section is set to No and the *KeepTogether* property for a group is set to Whole Group or With First Detail, the *KeepTogether* property setting for the section is ignored.
Example

As it applies to the **Section** object.

The following example returns the **KeepTogether** property setting for a report's detail section and assigns the value to the `intGetVal` variable.

```vba
Dim intGetVal As Integer
intGetVal = Me.Section(acDetail).KeepTogether
```
KeyboardLanguage Property

Language-specific information

You can use the KeyboardLanguage property to specify or determine the keyboard language on entry into a control. Read/write Byte.

expression.KeyboardLanguage

expression Required. An expression that returns one of the objects in the Applies To list.
Remarks

You can set this property by using the property sheet or Visual Basic.

Valid values for this property are 0 (zero), which corresponds to the default system language, or \( plid + 2 \) where \( plid \) is the primary language ID of a language installed on the current system. For example, the primary language ID of English is 9, so the corresponding **KeyboardLanguage** setting is 11. For a list of languages and their primary language IDs, search for "Primary Language IDs" in the MSDN Web site. (An exception to this list is Traditional Chinese which is represented by the value 200.)

Setting this property to a language that is not installed may either have no effect or cause an error.
**KeyPreview Property**

You can use the **KeyPreview** property to specify whether the form-level keyboard **event procedures** are invoked before a **control's** keyboard event procedures. Read/write **Boolean**.

`expression.KeyPreview`

`expression` Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **KeyPreview** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>True</td>
<td>The form receives keyboard events first, then the active control receives keyboard events.</td>
</tr>
<tr>
<td>No</td>
<td>False</td>
<td>(Default) Only the active control receives keyboard events.</td>
</tr>
</tbody>
</table>

**Note** You can set the **KeyPreview** property by using the form's property sheet, a macro, or Visual Basic.

You can set the **KeyPreview** property in any view.

You can use the **KeyPreview** property to create a keyboard-handling procedure for a form. For example, when an application uses function keys, setting the **KeyPreview** property to **True** allows you to process keystrokes at the form level rather than writing code for each control that might receive keystroke events.

To handle keyboard events only at the form level and prevent controls from receiving keyboard events, set the KeyAscii argument to 0 in the form's KeyPress event procedure, and set the KeyCode argument to 0 in the form's KeyDown and KeyUp event procedures.

If a form has no visible or enabled controls, it automatically receives all keyboard events.
Example

In the following example, the **KeyPreview** property is set to **True** in the form's Load event procedure. This causes the form to receive keyboard events before they are received by any control. The form KeyDown event then checks the KeyCode argument value to determine if the F2, F3, or F4 keys were pushed.

```vbnet
Private Sub Form_Load()
    Me.KeyPreview = True
End Sub

Private Sub Form_KeyDown(KeyCode As Integer, Shift As Integer)
    Select Case KeyCode
        Case vbKeyF2
            ' Process F2 key events.
        Case vbKeyF3
            ' Process F3 key events.
        Case vbKeyF4
            ' Process F4 key events.
        Case Else
            End Select
    End Sub
```
Kind Property

The Kind property indicates the type of reference that a Reference object represents. Read-only vbext_RefKind.

expression.Kind

expression  Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **Kind** property is read-only and can be read only from Visual Basic.

The **Kind** property returns the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vbext_rk_Project</td>
<td>The <strong>Reference</strong> object represents a reference to a Visual Basic project.</td>
</tr>
<tr>
<td>(Project)</td>
<td></td>
</tr>
<tr>
<td>vbext_rk_TypeLib</td>
<td>The <strong>Reference</strong> object represents a reference to a file that contains a</td>
</tr>
<tr>
<td>(TypeLib)</td>
<td>type library.</td>
</tr>
</tbody>
</table>
LabelAlign Property

The **LabelAlign** property specifies the text alignment within attached **labels** on new **controls**. Read/write **Byte**.

*expression*.LabelAlign

*expression* Required. An expression that returns one of the objects in the Applies To list.
## Remarks

The **LabelAlign** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>General</td>
<td>0</td>
<td>(Default) The label text aligns to the left.</td>
</tr>
<tr>
<td>Left</td>
<td>1</td>
<td>The label text aligns to the left.</td>
</tr>
<tr>
<td>Center</td>
<td>2</td>
<td>The label text is centered.</td>
</tr>
<tr>
<td>Right</td>
<td>3</td>
<td>The label text aligns to the right.</td>
</tr>
<tr>
<td>Distribute</td>
<td>4</td>
<td>The label text is evenly distributed.</td>
</tr>
</tbody>
</table>

**Note**  You can set the **LabelAlign** property by using a control's default control style or the **DefaultControl** method in Visual Basic.

When created, controls have an attached label (as long as their **AutoLabel** property is set to Yes). Changes to the **LabelAlign** default control style setting affect only controls created on the current form or report. To change the default control style for all new forms or reports that you create without using a Microsoft Access wizard, see [Specify a new template for forms and reports](#).
LabelX Property

Language-specific information

The LabelX property (along with the LabelY property) specifies the placement of the label for a new control. Read/write Integer.

expression.LabelX

expression Required. An expression that returns one of the objects in the Applies To list.
Remarks

If the orientation is left to right for a form or report, LabelX and LabelY behavior matches standard Microsoft Access left-to-right orientation. For more information about orientation, see the Orientation property.

If orientation is right to left, the origin of the coordinate system for LabelX and LabelY is the upper right corner of the attached control. A negative number for LabelX places the label to the right of the control. A negative number for LabelY places the label above the control.

For General and Right alignment when orientation is RTL, LabelX and LabelY specify the location of the upper-right corner of the label relative to the upper-right corner of the label’s attached control. For Left and Center alignment, LabelX and LabelY specify the location of the upper-left corner and top center, respectively, of the label relative to the upper-right corner of the label’s attached control.
LabelY Property

Language-specific information

The **LabelY** property (along with the **LabelX** property) specifies the placement of the label for a new control. Read/write **Integer**.

*expression*.LabelY

*expression* Required. An expression that returns one of the objects in the Applies To list.
Remarks

If the orientation is left to right for a form or report, **LabelX** and **LabelY** behavior matches standard Microsoft Access left-to-right orientation. For more information about orientation, see the **Orientation** property.

If orientation is right to left, the origin of the coordinate system for **LabelX** and **LabelY** is the upper right corner of the attached control. A negative number for **LabelX** places the label to the right of the control. A negative number for **LabelY** places the label above the control.

For General and Right alignment when orientation is RTL, **LabelX** and **LabelY** specify the location of the upper-right corner of the label relative to the upper-right corner of the label’s attached control. For Left and Center alignment, **LabelX** and **LabelY** specify the location of the upper-left corner and top center, respectively, of the label relative to the upper-right corner of the label’s attached control.
LanguageSettings Property

You can use the LanguageSettings property to return a read-only reference to the current LanguageSettings object and its related properties.

expression.LanguageSettings

expression  Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **LanguageSettings** property is available only by using [Visual Basic](https://en.wikipedia.org/wiki/Visual_Basic). Once you establish a reference to the **LanguageSettings** object, you can access all the properties and methods of the object. You can set a reference to the **LanguageSettings** object by clicking **References** on the **Tools** menu while in module **Design view**. Then set a reference to the Microsoft Office Object Library in the **References** dialog box by selecting the appropriate check box. Microsoft Access can set this reference for you if you use a Microsoft Office Object Library constant to set a **LanguageSettings** object's property or as an argument to a **LanguageSettings** object's method.
Example

The following example displays a message indicating the language Access uses for Help on the user's machine. A listing of all the available languages and their identification numbers is available in the Visual Basic Editor by selecting **Object Browser** from the **View** menu, typing the word "MsoLanguageID" in the **Search Text box**, and clicking the **Search** button.

Dim mli As MsoLanguageID

mli = Application.LanguageSettings.LanguageID(msoLanguageIDHelp)
MsgBox "The language ID used for Access Help is " & mli
LayoutForPrint Property

You can use the `LayoutForPrint` property to specify whether the [form](#) or [report](#) uses printer or screen [fonts](#). Read/write [Boolean](#).

`expression/LayoutForPrint`

`expression`  Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **LayoutForPrint** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>True</td>
<td>(Default for reports) Uses printer fonts.</td>
</tr>
<tr>
<td>No</td>
<td>False</td>
<td>(Default for forms) Uses screen fonts.</td>
</tr>
</tbody>
</table>

**Note** You can set this property by using the form's or report's property sheet, a *macro*, or *Visual Basic*.

The property can be set only in form **Design view** or report **Design view**.

When you choose a font in Microsoft Access, you are choosing either a screen font or a printer font, depending on the setting of the **LayoutForPrint** property. Remember that printer fonts and screen fonts can differ, and characters on screen may not look exactly like those displayed on the printed page.

**Tip**

If you select a scalable font, such as a TrueType font, the screen and printer characters will look nearly the same.

Screen fonts are the images of letters, numbers, and symbols that are installed on your system to be displayed on the screen. If you installed a printer, additional screen fonts may have been installed automatically.

Printer fonts are the letters, numbers, and symbols that are produced when you print a report or a form. The available fonts are those fonts that were installed as part of your printer's setup, and depend on your printer.

If you set the **LayoutForPrint** property to Yes, the *Formatting (Form/Report)* toolbar displays the fonts and **point** sizes available for your printer.

If you design a form or report on a system with a different printer than the one you will use to print, Microsoft Access displays a message when you print the form or report to let you know that it was designed for another kind of printer. If
you print the form or report anyway, your printer may substitute different fonts. Similarly, Microsoft Access may substitute fonts if you change the **LayoutForPrint** property setting. For example, you might design a form or report with **LayoutForPrint** set to No, then change the setting to Yes. You can reselect the font for each control to specify the appearance of the form or report.
Example

The following example instructs Microsoft Access to use screen fonts for a given form.

Forms("Purchase Orders").\texttt{LayoutForPrint} = \texttt{False}
Left Property

You can use the **Left** property to specify an object's location on a [form](#) or [report](#). Read/write **Integer** for all of the items in the Applies To list except for the the [Report](#) object, which is read/write **Long**.

`expression.Left`

*expression*  
Required. An expression that returns one of the objects in the Applies To list.
Remarks

A control's location is the distance measured from its left or top border to the left or top edge of the section containing the control. Setting the **Left** property to 0 places the control's edge at the very left of the section. To use a unit of measurement different from the setting in the **Regional Options** dialog box in Windows Control Panel, specify the unit, such as cm or in (for example, 3 cm or 2 in).

In Visual Basic, use a **numeric expression** to set the value of this property. Values are expressed in **twips**.

For controls, you can set these properties by using a control's **property sheet**, a **macro**, or **Visual Basic**.

For reports, you can set these properties only by using a macro or **event procedure** in Visual Basic while the report is in **Print Preview** or being printed.

When you move a control, its new **Left** property setting is automatically entered in the property sheet. When you view a form or report in Print Preview or when you print a form, a control's location is determined by its **Left** property setting along with the margin settings in the **Page Setup** dialog box, available by clicking **Page Setup** on the **File** menu.

For reports, the **Left** property setting is the amount the current section is offset from the left of the page. This property is expressed in twips. You can use this property to specify how far down the page you want a section to print in the section's **Format** event procedure.
Example

The following example checks the `Left` property setting for the current report. If the value is less than the minimum margin setting, the `NextRecord` and `PrintSection` properties are set to `False` (0). The section doesn't advance to the next record, and the next section isn't printed.

Sub Detail1_Format(Cancel As Integer, FormatCount As Integer)

    Const conLeftMargin = 1880

    ' Don't advance to next record or print next section
    ' if Left property setting is less than 1880 twips.
    If Me.Left < conLeftMargin Then
        Me.NextRecord = False
        Me.PrintSection = False
    End If

End Sub
LeftMargin Property

LeftMargin property as it applies to the Label and TextBox objects.

Along with the TopMargin, RightMargin, and BottomMargin properties.

specifies the location of information displayed within a label or text box control.

Read/write Integer.

expression.LeftMargin

expression Required. An expression that returns one of the above objects.
Remarks

A control's displayed information location is measured from the control's left, top, right, or bottom border to the left, top, right, or bottom edge of the displayed information. Setting the LeftMargin or TopMargin property to 0 places the displayed information's edge at the very left or top of the control. To use a unit of measurement different from the setting in the regional settings of Windows, specify the unit (for example, cm or in).

In Visual Basic, use a numeric expression to set the value of this property. Values are expressed in *twips*.

You can set these properties by using the property sheet, a macro, or Visual Basic.

LeftMargin property as it applies to the Printer object.

Along with the TopMargin, RightMargin, and BottomMargin properties, specifies the margins for a printed page. Read/write Long.

expression.LeftMargin

twips

expression Required. An expression that returns a Printer object.
Example

As it applies to the Label and TextBox objects.

The following example offsets the caption in the label "EmployeeID_Label" in the "Purchase Orders" form by 100 twips from the left of the label's border.

With Forms.Item("Purchase Orders").Controls.Item("EmployeeID_Label")
  .LeftMargin = 100
End With
LimitToList Property

You can use the LimitToList property to limit a combo box's values to the listed items. Read/write Boolean.

expression.LimitToList

expression Required. An expression that returns one of the objects in the Applies To list.
Remarks

The LimitToList property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>True</td>
<td>If the user selects an item from the list in the combo box or enters text that matches a listed item, Microsoft Access accepts it. If the entered text doesn't match a listed item, the text isn't accepted and the user must then retype the entry, select a listed item, press ESC, or click Undo on the Edit menu.</td>
</tr>
<tr>
<td>No</td>
<td>False</td>
<td>(Default) Microsoft Access accepts any text that conforms to the ValidationRule property.</td>
</tr>
</tbody>
</table>

You can set the LimitToList property by using the combo box's property sheet, a macro, or Visual Basic.

For table fields, you can set this property on the Lookup tab of the Field Properties section of table Design view for fields with the DisplayControl property set to Combo Box.

Tip

Microsoft Access sets the LimitToList property automatically when you select Lookup Wizard as the data type for a field in table Design view.

When the LimitToList property of a bound combo box is set to No, you can enter a value in the combo box that isn't included in the list. Microsoft Access stores the new value in the form's underlying table or query (in the field specified in the combo box's ControlSource property), not the table or query set for the combo box by the RowSource property. To have newly entered values appear in the combo box, you must add the new value to the table or query set in the RowSource property by using a macro or Visual Basic event procedure that runs when the NotInList event occurs.

Note  If you set the combo box's BoundColumn property to any column other than the first visible column (or if you set BoundColumn to 0), the LimitToList
property is automatically set to Yes.

Setting both the LimitToList property and the AutoExpand property to Yes lets Microsoft Access find matching values from the list as the user enters characters in the text box portion of the combo box, and restricts the entries to only those values.

When the LimitToList property is set to Yes and the user clicks the arrow next to the combo box, Microsoft Access selects matching values in the list as the user enters characters in the text box portion of the combo box, even if the AutoExpand property is set to No. If the user presses ENTER or moves to another control or record, the selected value appears in the combo box.

Combo boxes accept Null values when the LimitToList property is set to Yes or True, whether or not the list contains Null values. If you want to prevent users from entering a Null value in a combo box, set the Required property of the field in the table to which the combo box is bound to Yes.
Example

The following example limits a given combo box's values to its listed items.

Forms("Order Entry").Controls("States").**LimitToList** = True
**Lines Property**

The **Lines** property returns a **string** containing the contents of a specified line or lines in a **standard module** or a **class module**. Read-only **String**.

*expression*.Lines(*Line*, *NumLines*)

*expression*  Required. An expression that returns one of the objects in the Applies To list.

*Line*  Required **Long**. The line number of the first line to return.

*NumLines*  Required **Long**. The number of lines to return.
Remarks

The **Lines** property is available only by using Visual Basic.

Lines in a module are numbered beginning with 1. For example, if you read the **Lines** property with a value of 1 for the *line* argument and 1 for the *numlines* argument, the **Lines** property returns a string containing the text of the first line in the module.

To insert a line of text into a module, use the **InsertLines** method.
Example

The following example deletes a specified line from a module.

```vba
Function DeleteWholeLine(strModuleName, strText As String) As Boolean
    Dim mdl As Module, lngNumLines As Long
    Dim lngSLine As Long, lngSCol As Long
    Dim lngELine As Long, lngECol As Long
    Dim strTemp As String

    On Error GoTo Error_DeleteWholeLine
    DoCmd.OpenModule strModuleName
    Set mdl = Modules(strModuleName)

    If mdl.Find(strText, lngSLine, lngSCol, lngELine, lngECol) Then
        lngNumLines = Abs(lngELine - lngSLine) + 1
        strTemp = LTrim$(mdl.Lines(lngSLine, lngNumLines))
        strTemp = RTrim$(strTemp)
        If strTemp = strText Then
            mdl.DeleteLines lngSLine, lngNumLines
        Else
            MsgBox "Line contains text in addition to " & strText & ", not found." & vbCr & "DeleteWholeLine = True"
        End If
    Else
        MsgBox "Text " & strText & ", not found." & vbCrLf & "DeleteWholeLine = False"
    End If

    DeleteWholeLine = True
    Exit_DeleteWholeLine:
    Exit Function

Error_DeleteWholeLine:
    MsgBox Err & ":" & Err.Description
    DeleteWholeLine = False
    Resume Exit_DeleteWholeLine
End Function
```

You could call this function from a procedure such as the following, which searches the module Module1 for a constant declaration and deletes it.

```vba
Sub DeletePiConst()
    If DeleteWholeLine("Module1", "Const conPi = 3.14") Then
```

Debug.Print "Constant declaration deleted successfully."
Else
   Debug.Print "Constant declaration not deleted."
End If
End Sub
LineSlant Property

You use the LineSlant property to specify whether a line control slants from upper left to lower right or from upper right to lower left. Read/write Boolean.

expression.LineSlant

expression  Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **LineSlant** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\</td>
<td>False</td>
<td>(Default) Upper left to lower right</td>
</tr>
<tr>
<td>/</td>
<td>True</td>
<td>Upper right to lower left</td>
</tr>
</tbody>
</table>

**Note** You can set this property by using the control's property sheet, a macro, or Visual Basic.

Use the **LineSlant** property to change a line's direction. To position and size the line on your form or report, use the mouse.
Example

The following example slants a line on a form from upper right to lower left.

Forms("Purchase Orders").Controls("Section Separator").LineSlant = True
LineSpacing Property

You can use the LineSpacing property to specify or determine the location of information displayed within a label or text box control. Read/write Integer.

expression.LineSpacing

expression Required. An expression that returns one of the objects in the Applies To list.
Remarks

A control's displayed information location is the distance measured between each line of the displayed information. To use a unit of measurement different from the setting in the **Regional Options** dialog box in **Windows Control Panel**, specify the unit, such as cm or in (for example, 3 cm or 2 in).

In Visual Basic, use a **numeric expression** to set the value of this property. Values are expressed in **twips**.

You can set these properties by using the **property sheet**, a **macro**, or **Visual Basic**.
Example

The following example sets the line spacing to 0.25 inches for the text box "PurchaseOrderInformation" on the "Purchase Order Form"

' 0.25 inches = 360/1440 twips.
Forms("Purchase Orders").Controls("PurchaseOrderDescription").LineSpacing = 360
LinkChildFields Property

You can use the `LinkChildFields` property (along with the `LinkMasterFields` property) together to specify how Microsoft Access links records in a form or report to records in a subform, subreport, or embedded object, such as a chart. If these properties are set, Microsoft Access automatically updates the related record in the subform when you change to a new record in a main form. Read/write `String`.

`expression.LinkChildFields`

`expression` Required. An expression that returns one of the objects in the Applies To list.
Remarks

You can set the LinkChildFields and LinkMasterFields properties for the subform, subreport, or embedded object as follows:

- The LinkChildFields property. Enter the name of one or more linking fields in the subform, subreport, or embedded object.

- The LinkMasterFields property. Enter the name of one or more linking fields or controls in the main form or report.

You can use the Subform/Subreport Field Linker to set these properties by clicking the Build button to the right of the property box in the property sheet.

You can also set these properties by using a string expression in a macro or Visual Basic.

The properties can only be set in Design view or during the Open event of a form or report.

The fields or controls you use to set these properties don't need to have the same names, but they must contain the same kind of data and have the same or a compatible data type and field size. For example, an AutoNumber field is compatible with a Number field if the FieldSize property for the Number field is set to Long Integer.

You can use the name of a control (including the name of a calculated control) to set the LinkMasterFields property, but you can't use the name of a control to set the LinkChildFields property. If you want to use a calculated value as the link for a subform, subreport, or embedded object, define a calculated field in the child object's underlying query and set the LinkChildFields property to the field.

When you specify more than one field or control name for these property settings, you must enter the same number of fields or controls for each property setting and separate the names with a semicolon (;).

When you create a subform or subreport by dragging a form or report from the
Database window onto another form or report or by using the Form Wizard, Microsoft Access automatically sets the LinkChildFields and LinkMasterFields properties under the following conditions:

- Both the main form or report and the child object are based on tables, and a relationship between those tables has been defined with the Relationships command. Microsoft Access uses the fields that relate the two tables as the linking fields.

- The main form or report is based on a table with a primary key, and the subform or subreport is based on a table or query that contains a field with the same name and the same or a compatible data type as the primary key. Microsoft Access uses the primary key from the main object's underlying table and the identically named field from the child object's underlying table or query as the linking fields.

**Note** The linking fields don't have to be included in the main object or in the child object. As long as they are contained in the objects' underlying tables or queries, you can use the fields to link the objects. When you use a wizard, Microsoft Access automatically includes the linking fields.
LinkMasterFields Property

You can use the LinkMasterFields property (along with the LinkChildFields property) together to specify how Microsoft Access links records in a form or report to records in a subform, subreport, or embedded object, such as a chart. If these properties are set, Microsoft Access automatically updates the related record in the subform when you change to a new record in a main form. Read/write String.

expression.LinkMasterFields

evaluation Required. An expression that returns one of the objects in the Applies To list.
Remarks

You can set the LinkChildFields and LinkMasterFields properties for the subform, subreport, or embedded object as follows:

- The **LinkChildFields** property. Enter the name of one or more linking fields in the subform, subreport, or embedded object.

- The **LinkMasterFields** property. Enter the name of one or more linking fields or controls in the main form or report.

You can use the Subform/Subreport Field Linker to set these properties by clicking the **Build** button to the right of the property box in the property sheet.

You can also set these properties by using a string expression in a macro or Visual Basic.

The properties can only be set in Design view or during the Open event of a form or report.

The fields or controls you use to set these properties don't need to have the same names, but they must contain the same kind of data and have the same or a compatible data type and field size. For example, an AutoNumber field is compatible with a Number field if the FieldSize property for the Number field is set to Long Integer.

You can use the name of a control (including the name of a calculated control) to set the **LinkMasterFields** property, but you can't use the name of a control to set the **LinkChildFields** property. If you want to use a calculated value as the link for a subform, subreport, or embedded object, define a calculated field in the child object's underlying query and set the **LinkChildFields** property to the field.

When you specify more than one field or control name for these property settings, you must enter the same number of fields or controls for each property setting and separate the names with a semicolon (;).

When you create a subform or subreport by dragging a form or report from the
Database window onto another form or report or by using the Form Wizard, Microsoft Access automatically sets the LinkChildFields and LinkMasterFields properties under the following conditions:

- Both the main form or report and the child object are based on tables, and a relationship between those tables has been defined with the Relationships command. Microsoft Access uses the fields that relate the two tables as the linking fields.

- The main form or report is based on a table with a primary key, and the subform or subreport is based on a table or query that contains a field with the same name and the same or a compatible data type as the primary key. Microsoft Access uses the primary key from the main object's underlying table and the identically named field from the child object's underlying table or query as the linking fields.

Note The linking fields don't have to be included in the main object or in the child object. As long as they are contained in the objects' underlying tables or queries, you can use the fields to link the objects. When you use a wizard, Microsoft Access automatically includes the linking fields.
ListCount Property

You can use the ListCount property to determine the number of rows in a list box or the list box portion of a combo box. Read/write Long.

expression.ListCount

expression  Required. An expression that returns one of the objects in the Applies To list.
Remarks

Microsoft Access sets the **ListCount** property to the number of rows in the list box or the list box portion of the combo box. The value of the **ListCount** property is read-only and can't be set by the user.

This property is available only by using a [macro](/macro) or [Visual Basic](/visual-basic). You can read this property only in Form view and Datasheet view.

The **ListCount** property setting contains the total number of rows in the combo box list or list box, as determined by the control's **RowSource** and **RowSourceType** properties. If the control is based on a table or query (the **RowSourceType** property is set to Table/Query and the **RowSource** property is set to a particular table or query), the **ListCount** property setting contains the number of records in the table or query result set. If the **RowSourceType** property is set to Value List, the **ListCount** property setting contains the number of rows the value list specified in the **RowSource** property results in (this depends on the value list and the number of columns in the list box or combo box list, as set by the **ColumnCount** property).

If you set the **ColumnHeads** property to Yes, the row of column headings is included in the number of rows returned by the **ListCount** property. For combo boxes and list boxes based on a table or query, adding column headings adds an additional row. For combo boxes and list boxes based on a value list, adding column headings leaves the number of rows unchanged (the first row of values becomes the column headings).

You can use the **ListCount** property with the **ListRows** property to specify how many rows you want to display in the list box portion of a combo box.
Example

The following example uses the **ListCount** property to find the number of rows in the list box portion of the CustomerList combo box on a Customers form. It then sets the **ListRows** property to display a specified number of rows in the list.

```vba
Public Sub SizeCustomerList()
    Dim ListControl As Control
    Set ListControl = Forms!Customers!CustomerList
    With ListControl
        If .ListCount < 8 Then
            .ListRows = .ListCount
        Else
            .ListRows = 8
        End If
    End With
End Sub
```
Show All
ListIndex Property

You can use the **ListIndex** property to determine which item is selected in a list box or combo box. Read-only **Long**.

`expression.ListIndex`

*expression* Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **ListIndex** property is an integer from 0 to the total number of items in a list box or combo box minus 1. Microsoft Access sets the **ListIndex** property value when an item is selected in a list box or list box portion of a combo box. The **ListIndex** property value of the first item in a list is 0, the value of the second item is 1, and so on.

This property is available only by using a [macro](https://support.microsoft.com/en-us/help/16985) or [Visual Basic](https://msdn.microsoft.com/en-us/library/office/bb667557.aspx). You can read this property only in **Form view** and **Datasheet view**. This property is read-only and isn't available in other views.

The **ListIndex** property value is also available by setting the **BoundColumn** property to 0 for a combo box or list box. If the **BoundColumn** property is set to 0, the underlying [table field](https://msdn.microsoft.com/en-us/library/office/bb438222.aspx) to which the combo box or list box is bound will contain the same value as the **ListIndex** property setting.

List boxes also have a [MultiSelect](https://msdn.microsoft.com/en-us/library/office/bb438309.aspx) property that allows the user to select multiple items from the control. When multiple selections are made in a list box, you can determine which items are selected by using the **Selected** property of the control. The **Selected** property is an array of values from 0 to the **ListCount** property value minus 1. For each item in the list box the **Selected** property will be **True** if the item is selected and **False** if it is not selected.

The **ItemsSelected** collection also provides a way to access data in the selected rows of a list box or combo box.
Example

To return the value of the **ListIndex** property, you can use the following:

```vba
Dim l As Long
l = Forms(formname).Controls(controlname).ListIndex
```

To set the **ListIndex** property value, you can use the following:

```vba
Forms(formname).Controls(controlname).SetFocus
Forms(formname).Controls(controlname).ListIndex = index
```

Where `formname` and `controlname` are the names of the form and list box or combo box control, respectively, expressed as **String** values, and `index` is the index value of the item.
ListRows Property

You can use the **ListRows** property to set the maximum number of rows to display in the list box portion of a combo box. Read/write **Integer**.

*expression*.ListRows

*expression* Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **ListRows** property holds an integer that indicates the maximum number of rows to display. The default setting is 8. The setting for the **ListRows** property must be from 1 to 255.

You can set this property by using the combo box's *property sheet*, a *macro*, or *Visual Basic*.

For *table fields*, you can set this property on the **Lookup** tab of the Field Properties section of *table Design view* for fields with the **DisplayControl** property set to Combo Box.

**Tip**

Microsoft Access sets the **ListRows** property automatically when you select Lookup Wizard as the data type for a field in table Design view.

In Visual Basic, use a *numeric expression* to set the value of this property.

You can set the default for this property by using a combo box's *default control style* or the **DefaultControl** method in Visual Basic.

If the actual number of rows exceeds the number specified by the **ListRows** property setting, a vertical scroll bar appears in the list box portion of the combo box.
Example

The following example uses the **ListCount** property to find the number of rows in the list box portion of the CustomerList combo box on a Customers form. It then sets the **ListRows** property to display a specified number of rows in the list.

```vba
Public Sub SizeCustomerList()

    Dim ListControl As Control

    Set ListControl = Forms!Customers!CustomerList
    With ListControl
        If .ListCount < 8 Then
            .ListRows = .ListCount
        Else
            .ListRows = 8
        End If
    End With

End Sub
```
ListWidth Property

You can use the ListWidth property to set the width of the list box portion of a combo box. Read/write String.

expression.ListWidth

expression  Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **ListWidth** property holds a value specifying the width of the list box portion of a combo box in inches or centimeters, depending on the measurement system (U.S. or Metric) selected in the **Measurement system** box on the **Numbers** tab of the **Regional Options** dialog box of Windows Control Panel. To use a unit other than the default, include a measurement indicator, such as cm or in. The default setting (Auto) makes the list box portion of the combo box the same width as the combo box.

You can set the **ListWidth** property by using the combo box's [property sheet](#), a [macro](#), or [Visual Basic](#).

For [table fields](#), you can set this property on the **Lookup** tab of the Field Properties section of [table Design view](#) for fields with the **DisplayControl** property set to Combo Box.

**Tip**

Microsoft Access sets the **ListWidth** property automatically when you select Lookup Wizard as the data type for a field in table Design view.

In Visual Basic, use a **numeric expression** to set the value of this property. The default unit of measurement in Visual Basic is **twips**.

You can also set the default for this property by using a combo box's [default control style](#) or the **DefaultControl** method in Visual Basic.

The list portion of the combo box can be wider than the combo box but can't be narrower.

If you want to display a multiple-column list, enter a value that will make the list box wide enough to show all the columns.

**Tip**

When designing combo boxes, be sure to leave enough space to display your data and for Microsoft Access to insert a vertical scroll bar.
Example

The following example returns the value of the **ListWidth** property for the "States" combo box on the "Order Entry" form.

```vba
Dim str As String
str = Forms("Order Entry").Controls("States").**ListWidth**
```
LocationOfComponents Property

You can use the **LocationOfComponents** property to specify or determine the central [URL](#) or path where Microsoft Office controls can be downloaded by authorized users viewing your saved database. Read/write **String**.

*expression.*LocationOfComponents

*expression* Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **LocationOfComponents** property is available only by using [Visual Basic](#).

The Microsoft Office controls are automatically downloaded with the Web page if the **DownloadComponents** property is **True**, the components are not already installed, the path is valid and points to a location that contains the necessary components, and the user has a valid Microsoft Office license.
Example

This example sets the path where Office components are downloaded.

Application.DefaultWebOptions.LocationOfComponents = _
"\\Server1\CompLoc"
Locked Property

The **Locked** property specifies whether you can edit data in a control in Form view. Read/write **Boolean**.

`expression.Locked`

`expression` Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **Locked** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>True</td>
<td>(Default for unbound object frames) The control functions normally but doesn't allow editing, adding, or deleting data.</td>
</tr>
<tr>
<td>No</td>
<td>False</td>
<td>(Default for all controls except unbound object frames) The control functions normally and allows editing, adding, and deleting data.</td>
</tr>
</tbody>
</table>

You can set these properties by using a form's [property sheet](#), a [macro](#), or [Visual Basic](#).

Use the **Locked** property to protect data in a [field](#) by making it read-only. For example, you might want a control to only display information without allowing editing, or you might want to lock a control until a specific condition is met.
Example

The following example toggles the **Enabled** property of a command button and the **Enabled** and **Locked** properties of a control, depending on the type of employee displayed in the current record. If the employee is a manager, then the SalaryDetails button is enabled and the PersonalInfo control is unlocked and enabled.

```vba
Sub Form_Current()
    If Me!EmployeeType = "Manager" Then
        Me!SalaryDetails.Enabled = True
        Me!PersonalInfo.Enabled = True
        Me!PersonalInfo.Locked = False
    Else
        Me!SalaryDetails.Enabled = False
        Me!PersonalInfo.Enabled = False
        Me!PersonalInfo.Locked = True
    End If
End Sub
```
MailEnvelope Property

Returns an MsoEnvelope object that represents an e-mail header for a data access page.

expression.MailEnvelope

expression Required. An expression that returns one of the objects in the Applies To list.
Example

This example sets the comments for the e-mail header of the specified data access page.

Dim envDAP As MsoEnvelope

Set envDAP = DataAccessPages(0).MailEnvelope

envDAP.Introduction = _
    "Please review this report and let me know " & _
    "what you think. I need your input by Friday. " & _
    "Thanks."
Major Property

The **Major** property of a **Reference** object returns a read-only **Long** value indicating the major version number of an application to which you have set a reference.

**expression.Major**

**expression**  Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **Major** property is available only by using Visual Basic.

The **Major** property returns the value to the left of the decimal point in a version number. For example, if you've set a reference to an application whose version number is 2.5, the **Major** property returns 2.
Example

The following example displays a message with information about all the references in the current project.

Dim r As Reference
Dim strInfo As String

For Each r In Application.References
    strInfo = strInfo & r.Name & " " & r.Major & "." & r.Minor & vbCrLf
Next

MsgBox "Current References: " & vbCrLf & strInfo
MaxRecButton Property

You can use the MaxRecButton property to specify or determine if the maximum record limit button is available on the navigation bar of a form in Datasheet view or Form view. This property is only available for forms within a Microsoft Access project (.adp). Read/write Boolean.

expression.MaxRecButton

expression Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **MaxRecButton** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td><strong>True</strong></td>
<td>(Default) The maximum record limit button is available to the user.</td>
</tr>
<tr>
<td>No</td>
<td><strong>False</strong></td>
<td>The maximum record limit button is unavailable to the user.</td>
</tr>
</tbody>
</table>

**Note**  You can set this property by using the [property sheet](#), or [Visual Basic](#).
Example

This example makes the maximum record limit button on the "Order Entry" form unavailable to the user.

Forms("Order Entry").MaxRecButton = False
MaxRecords Property

Specifies the maximum number of records that will be returned by:

- A query that returns data from an ODBC database to an Microsoft Access database (.mdb).
- A view that returns data from a SQL database to an Access project (.adp).

expression.MaxRecords

expression Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **MaxRecords** property setting is a **Long Integer** value representing the number of records that will be returned.

In a Microsoft Access database, you can set this property by using the query's property sheet or Visual Basic.

When you set this property in Visual Basic you use the ADO **MaxRecords** property.

Records are returned in the order specified by the query's ORDER BY clause.

You can use the **MaxRecords** property in situations where limited system resources might prohibit a large number of returned records.
Example

To return the **MaxRecords** property for a form, you can use the following:

```vbnet
Dim l As Long
l = Forms(formname).MaxRecords
```

To set the **MaxRecords** property, you can use the following:

```vbnet
Forms(formname).MaxRecords = numrecords
```

where `formname` is the name of the form expressed as a **String**, and `numrecords` is a **Long Integer** value representing the number of records that will be returned.
MenuBar Property

You can use the `MenuBar` property to specify the menu bar to use for a Microsoft Access database (.mdb), Access project (.adp), form, or report. You can also use the `MenuBar` property to specify the menu bar macro that will be used to display a custom menu bar for a database, form, or report. Read/write String.

`expression.MenuBar`

`expression` Required. An expression that returns one of the objects in the Applies To list.
Remarks

Enter the name of the menu bar you want to display. If you leave the MenuBar property setting blank, Microsoft Access displays the built-in (default) menu bar or the application's global menu bar. If you set the MenuBar property to a value that is not the name of an existing menu bar or menu bar macro, the form or report will not have a menu bar (the default menu bar will not be shown).

You can set this property by using the object's property sheet, a macro, or Visual Basic.

In Visual Basic, set this property by using a string expression that is the name of the menu bar you want to display.

To display the built-in menu bar or global menu bar for a database, form, or report by using a macro or Visual Basic, set the property to a zero-length string (" ").

When you use the MenuBar property with forms and reports, Microsoft Access displays the specified menu bar when the form or report is opened. This menu bar is displayed whenever the form or report has the focus.

When used with the Application object, the MenuBar property enables you to display a custom menu bar throughout the database. However, if you've set the MenuBar property for a form or report in the database, the custom menu bar of the form or report will be displayed in place of the database's custom menu bar whenever the form or report has the focus. When the form or report loses the focus, the custom menu bar for the database is displayed.

Note  You can switch between a database's custom menu bar and the built-in menu bar by pressing CTRL+F11.
Example

The following example sets the `MenuBar` property to a menu bar named CustomerMenu:

Forms!Customers.

To display the built-in menu bar for the form or the application global menu bar, you set the `MenuBar` property to a zero-length string (" ").

Forms!Customers.

MinMaxButtons Property

You can use the **MinMaxButtons** property to specify whether the **Maximize** and **Minimize** buttons will be visible on a **form**. Read/write **Byte**.

`expression.MinMaxButtons`

`expression` Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **MinMaxButtons** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>0</td>
<td>The <strong>Maximize</strong> and <strong>Minimize</strong> buttons aren't visible.</td>
</tr>
<tr>
<td>Min Enabled</td>
<td>1</td>
<td>Only the <strong>Minimize</strong> button is visible.</td>
</tr>
<tr>
<td>Max Enabled</td>
<td>2</td>
<td>Only the <strong>Maximize</strong> button is visible.</td>
</tr>
<tr>
<td>Both Enabled</td>
<td>3</td>
<td>(Default) Both the <strong>Minimize</strong> and <strong>Maximize</strong> buttons are visible.</td>
</tr>
</tbody>
</table>

**Note** You can set these properties by using the form's property sheet, a macro, or Visual Basic.

You can set the **MinMaxButtons** property only in form Design view.

Clicking a form's **Maximize** button enlarges the form so it fills the Microsoft Access window. Clicking a form's **Minimize** button reduces the form to a short title bar at the bottom of the Microsoft Access window.

To display the **Maximize** and **Minimize** buttons on a form, you must set the form's **BorderStyle** property to Thin or Sizable and the **ControlBox** property to Yes. If you set the **BorderStyle** property to None or Dialog, or if you set the **ControlBox** property to No, the form won't have **Maximize** or **Minimize** buttons, regardless of the **MinMaxButtons** property setting.

Even when the **MinMaxButtons** property is set to None, a form always has **Maximize** and **Minimize** buttons in Design view.

If a form's **MinMaxButtons** property is set to None, the **Maximize** and **Minimize** commands aren't available on the form's **Control** menu.
Example

The following example returns the value of the MinMaxButtons property for the "Order Entry" form.

Dim b As Byte
b = Forms("Order Entry").MinMaxButtons
Minor Property

The **Minor** property of a **Reference** object returns a **Long** value indicating the minor version number of the application to which you have set a reference.

*expression*.Minor

*expression* Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **Minor** property is available only by using Visual Basic and is read-only.

The **Minor** property returns the value to the right of the decimal point in a version number. For example, if you've set a reference to an application whose version number is 2.5, the **Minor** property returns 5.
**Example**

The following example displays a message with information about all the references in the current project.

```vbnet
Dim r As Reference
Dim strInfo As String

For Each r In Application.References
    strInfo = strInfo & r.Name & " " & r.Major & "." & r.Minor & vbCrLf
Next

MsgBox "Current References: " & vbCrLf & strInfo
```
Modal Property

You can use the Modal property to specify whether a form or report opens as a modal window. When a form or report opens as a modal window, you must close the window before you can move the focus to another object. Read/write Boolean.

expression Modal

expression Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **Modal** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td><strong>True</strong></td>
<td>The form or report opens as a modal window.</td>
</tr>
<tr>
<td>No</td>
<td><strong>False</strong></td>
<td>(Default) The form opens as a non-modal window.</td>
</tr>
</tbody>
</table>

You can set this property by using the form or report's [property sheet](#), a [macro](#), or [Visual Basic](#).

When you open a modal window, other windows in Microsoft Access are disabled until you close the form (although you can switch to windows in other applications). To disable menus and toolbars in addition to other windows, set both the **Modal** and **PopUp** properties to Yes.

You can use the **BorderStyle** property to specify the kind of border a form will have. Typically, modal forms have the **BorderStyle** property set to Dialog.

**Tip**

You can use the **Modal**, **PopUp**, and **BorderStyle** properties to create a [custom dialog box](#). You can set **Modal** to Yes, **PopUp** to Yes, and **BorderStyle** to Dialog for custom dialog boxes.

Setting the **Modal** property to Yes makes the form modal only when you:

- Open it in Form view from the [Database window](#).
- Open it in Form view by using a macro or Visual Basic.
- Switch from [Design view](#) to Form view.

When the form is modal, you can't switch to [Datasheet view](#) from Form view, although you can switch to Design view and then to Datasheet view.

The form isn't modal in Design view or Datasheet view and also isn't modal if you switch from Datasheet view to Form view.

**Note** You can use the Dialog setting of the Window Mode action argument of
the OpenForm action to open a form with its Modal and PopUp properties set to Yes.
Example

To return the value of the **Modal** property for the "Order Entry" form, you can use the following:

```vba
Dim b As Boolean
b = Forms("Order Entry").Modal
```

To set the value of the **Modal** property, you can use the following:

```vba
Forms("Order Entry").Modal = True
```
Module Property

You can use the Module property to specify a form module or report module. Read-only Module object.

`expression.Module`

`expression`  Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **Module** property is available only by using [Visual Basic](https://msdn.microsoft.com/en-us/library/office/aa463949(v=office.15).aspx) and is read-only in all views.

The **Module** property also returns a reference to a specified **Module** object.

Use the **Module** property to access the properties and methods of a **Module** object associated with a **Form** or **Report** object.

The setting of the **HasModule** property of a form or report determines whether it has an associated module. If the **HasModule** property is **False**, the form or report does not have an associated module. When you refer to the **Module** property of that form or report while in design view, Microsoft Access creates the associated module and sets the **HasModule** property to **True**. If you refer to the **Module** property of a form or report at run-time and the object has its **HasModule** property set to **False**, an error will occur.

You could use this property with any of the properties and methods of the module object.
The following example uses the **Module** property to insert the **Beep** method in a form's Open event.

```vba
Dim strFormOpenCode As String
Dim mdl As Module

Set mdl = Forms!MyForm.Module
strFormOpenCode = "Sub Form_Open(Cancel As Integer)" _
  & vbCrLf & "Beep" & vbCrLf & "End Sub"
With mdl
    .InsertText strFormOpenCode
End With
```
Modules Property

You can use the Modules property to access the Modules collection and its related properties. Read-only Modules object.

expression.Modules

expression  Required. An expression that returns one of the objects in the Applies To list.
Remarks

This property is available only by using Visual Basic.

Use the properties of the Modules collection in Visual Basic to refer to all open standard modules and class modules.
Example

The following example uses the **Module** property to insert the **Beep** method in a form's Open event.

```vba
Dim strFormOpenCode As String
Dim mdl As Module

Set mdl = Forms!MyForm.Module
strFormOpenCode = "Sub Form_Open(Cancel As Integer)"
    & vbCrLf & "Beep" & vbCrLf & "End Sub"
With mdl
    .InsertText strFormOpenCode
End With
```
MousePointer Property

You can use the MousePointer property together with the Screen object to specify or determine the type of mouse pointer currently displayed. Read/write Integer.

expression.MousePointer

expression Required. An expression that returns one of the objects in the Applies To list.
Remarks

The setting for the **MousePointer** property is an **Integer** value representing one of the following pointers.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(Default) The shape is determined by Microsoft Access</td>
</tr>
<tr>
<td>1</td>
<td>Normal Select (Arrow)</td>
</tr>
<tr>
<td>3</td>
<td>Text Select (I-Beam)</td>
</tr>
<tr>
<td>7</td>
<td>Vertical Resize (Size N, S)</td>
</tr>
<tr>
<td>9</td>
<td>Horizontal Resize (Size E, W)</td>
</tr>
<tr>
<td>11</td>
<td>Busy (Hourglass)</td>
</tr>
</tbody>
</table>

**Note** Setting the **MousePointer** property to an integer other than one that appears in the preceding table will cause the property to be set to 0.

You can set the **MousePointer** property only by using **Visual Basic**.

The **MousePointer** property affects the appearance of the mouse pointer over the entire screen. Some custom controls have a **MousePointer** property that, if set, will specify how the mouse pointer is displayed when it's positioned over the control.

You could use the **MousePointer** property to indicate that your application is busy by setting the property to 11 to display an hourglass icon. You can also read the **MousePointer** property to determine what's being displayed. This could be useful if you wanted to prevent a user from clicking a **command button** while the mouse pointer is displaying an hourglass icon.

Setting the **MousePointer** property to 11 is the same as passing the **True** (–1) argument to the **Hourglass** method of the **DoCmd** object. Conversely, passing the **True** argument to the **Hourglass** method also sets the **MousePointer** property to 11.
Example

The following example changes the mouse pointer to an hourglass.

Screen.MousePointer = 11
MouseWheel Property

Returns or sets a String indicating which macro, event procedure, or user-defined function runs when the MouseWheel event occurs. Read/write.

expression.MouseWheel

expression Required. An expression that returns one of the objects in the Applies To list.
Remarks

Valid values for this property are "macroname" where macroname is the name of macro, "[Event Procedure]" which indicates the event procedure associated with the MouseWheel event for the specified object, or "=functionname()" where functionname is the name of a user-defined function. For a more detailed discussion of event properties, see "Event Properties."
Example

The following example specifies that when the MouseWheel event occurs on the first form of the current project, the associated event procedure should run.

`Forms(0).MouseWheel = "[Event Procedure]"`
Moveable Property

Returns or sets a **Boolean** indicating whether the specified form or report can be moved by the user; **True** if it can be moved. Read/write.

*expression*.**Moveable**

*expression*  Required. An expression that returns one of the objects in the Applies To list.
Remarks

You can use the **Move** method to programmatically move a form or report regardless of the value of the **Moveable** property.
Example

The following example determines whether or not the first form in the current project can be moved.

If Forms(0).Moveable Then
    MsgBox "You may move the form."
Else
    MsgBox "The form cannot be moved."
End If
MoveLayout Property

The MoveLayout property specifies whether Microsoft Access should move to the next printing location on the page. Read/write Boolean.

expression.MoveLayout

expression Required. An expression that returns one of the objects in the Applies To list.
**Remarks**

The **MoveLayout** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>(Default) The section's <strong>Left</strong> and <strong>Top</strong> properties are advanced to the next print location.</td>
</tr>
<tr>
<td>False</td>
<td>The section's <strong>Left</strong> and <strong>Top</strong> properties are unchanged.</td>
</tr>
</tbody>
</table>

To set this property, specify a [macro](#) or [event procedure](#) for a section's **OnFormat** property.

Microsoft Access sets this property to **True** before each section's **Format** event.
Example

The following example sets the `MoveLayout` property for the "Purchase Order" report to its default setting.

```csharp
Reports("Purchase Order").MoveLayout = True
```
MSODSC Property

Returns a `DataSourceControl` object.

`expression.MSODSC`

`expression`  Required. An expression that returns one of the objects in the Applies To list.
Remarks

You must set a reference to the Microsoft Office Web Components type library in order to use this property.
Example

This example reports the version of Microsoft Office Web Components in use for the specified form. This example assumes that the current project contains a data access page, and that the data access page has a **Microsoft Office Data Source** control.

```vba
Dim objMSODSC As DataSourceControl
Set objMSODSC = DataAccessPages(0).MSODSC
MsgBox "Current version of Office Web Components: " _
 & objMSODSC.Version
```
MultiRow Property

You can use the MultiRow property to specify or determine whether a tab control can display more than one row of tabs. Read/write Boolean.

expression.MultiRow

expression  Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **MultiRow** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>True</td>
<td>Multiple rows are allowed.</td>
</tr>
<tr>
<td>No</td>
<td>False</td>
<td>(Default) Multiple rows aren’t allowed.</td>
</tr>
</tbody>
</table>

**Note** You can set the **MultiRow** property by using the control’s *property sheet*, a *macro*, or *Visual Basic*.

You can also set the default for this property by using a control's *default control style* or the *DefaultControl* method in Visual Basic.

When the **MultiRow** property is set to **True**, the number of rows is determined by the width and number of tabs. The number of rows may change if the control is resized or if additional tabs are added to the control.

When the **MultiRow** property is set to **False** and the width of the tabs exceeds the width of the control, navigation buttons appear on the right side of the tab control. You can use the navigation buttons to scroll through all the tabs on the tab control.
Example

To return the value of the `MultiRow` property for a tab control named "Details" on the "Order Entry" form, you can use the following:

```vba
Dim b As Boolean
b = Forms("Order Entry").Controls("Details").MultiRow
```

To set the value of the `MultiRow` property, you can use the following:

```vba
Forms("Order Entry").Controls("Details").MultiRow = True
```
MultiSelect Property

You can use the MultiSelect property to specify whether a user can make multiple selections in a list box on a form and how the multiple selections can be made. Read/write Byte.

expression.MultiSelect

expression  Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **MultiSelect** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>0</td>
<td>(Default) Multiple selection isn't allowed.</td>
</tr>
<tr>
<td>Simple</td>
<td>1</td>
<td>Multiple items are selected or deselected by clicking them with the mouse or pressing the SPACEBAR.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Multiple items are selected by holding down SHIFT and clicking them with the mouse or by holding down SHIFT and pressing an arrow key to extend the selection from the previously selected item to the current item. You can also select items by dragging with the mouse. Holding down CTRL and clicking an item selects or deselects that item.</td>
</tr>
<tr>
<td>Extended</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

**Note**  You can set the **MultiSelect** property by using the list box's [property sheet](#), a [macro](#), or [Visual Basic](#).

This property can be set only in [form Design view](#).

You can use the **ListIndex** property to return the index number for the selected item. When the **MultiSelect** property is set to Extended or Simple, you can use the list box's **Selected** property or **ItemsSelected** collection to determine the items that are selected. In addition, when the **MultiSelect** property is set to Extended or Simple, the value of the list box control will always be **Null**.

If the **MultiSelect** property is set to Extended, requerying the list box clears any selections made by the user.
Example

To return the value of the MultiSelect property for a list box named "Country" on the "Order Entry" form, you can use the following:

```vba
Dim b As Byte
b = Forms("Order Entry").Controls("Country").MultiSelect
```

To set the MultiSelect property, you can use the following:

```vba
Forms("Order Entry").Controls("Country").MultiSelect = 2 ' Extended.
```
Name Property

You can use the **Name** property to specify or determine the **string expression** that identifies the name of an object. Read/write **String** for the following objects: `AdditionalData`, `BoundObjectFrame`, `CheckBox`, `ComboBox`, `CommandButton`, `CustomControl`, `Form`, `Image`, `Label`, `Line`, `ListBox`, `Module`, `ObjectFrame`, `OptionButton`, `OptionGroup`, `Page`, `PageBreak`, `Rectangle`, `Report`, `Section`, `SmartTagProperty`, `SubForm`, `TabControl`, `TextBox`, and `ToggleButton`. Read-only **String** for the following objects: `AccessObject`, `AccessObjectProperty`, `Application`, `CodeProject`, `CurrentProject`, `DataAccessPage`, `Reference`, `SmartTag`, and `SmartTagAction`.

`expression.Name`

`expression`  Required. An expression that returns one of the above objects.
Remarks

A valid name must conform to the standard naming conventions for Microsoft Access. For Microsoft Access objects, the name may be up to 64 characters long. For controls, the name may be as long as 255 characters.

For objects, set the Name property by clicking Save on the File menu in Design view and entering a valid name. To change the name of an existing object in the Database window, click the name, then either click Rename on the Edit menu or click the name again. You can also change the name by right-clicking it and clicking Rename on the shortcut menu. To change the name of an existing object when the object is open, click Save As or Export on the File menu.

For a section or control, you can set this property by using the property sheet, a macro, or Visual Basic. You can use the Name property in expressions for objects.

The default name for new objects is the object name plus a unique integer. For example, the first new form is Form1, the second new form is Form2, and so on. A form can't have the same name as another system object, such as the Screen object.

For an unbound control, the default name is the type of control plus a unique integer. For example, if the first control you add to a form is a text box, its Name property setting is Text1.

For a bound control, the default name is the name of the field in the underlying source of data. If you create a control by dragging a field from the field list, the field's FieldName property setting is copied to the control's Name property box.

You can't use "Form" or "Report" to name a control or section.

Controls on the same form, report, or data access page can't have the same name, but controls on different forms, reports or data access pages can have the same name. A control and a section on the same form can't share the same name.
Example

The following example returns the Name property for the first form in the Forms collection.

Dim strFormName As String

strFormName = Forms(0).Name
NavigationButtons Property

You can use the **NavigationButtons** property to specify whether navigation buttons and a record number box are displayed on a form. Read/write **Boolean**.

`expression.NavigationButtons`

*expression* Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **NavigationButtons** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>True</td>
<td>(Default) The form has navigation buttons and a record number box.</td>
</tr>
<tr>
<td>No</td>
<td>False</td>
<td>The form doesn't have navigation buttons or a record number box.</td>
</tr>
</tbody>
</table>

**Note** You can set this property by using the form's [property sheet](https://example.com), a [macro](https://example.com), or [Visual Basic](https://example.com).

Navigation buttons provide an efficient way to move to the first, previous, next, last, or blank (new) record. The record number box displays the number of the current record. The total number of records is displayed next to the navigation buttons. You can enter a number in the record number box to move to a particular record.

If you remove the navigation buttons from a form and want to create your own means of navigation for the form, you can create custom navigation buttons and add them to the form.
Example

The following example returns the value of the **Navigation Buttons** property for the "Order Entry" form.

```vba
Dim b As Boolean
b = Forms("Order Entry").NavigationButtons
```
NewFileTaskPane Property

Returns a NewFile object that represents a document listed on the New File task pane.

expression.NewFile

expression  Required. An expression that returns one of the objects in the Applies To list.
Example

This example creates a file list item on the New File task pane in the New from existing file section.

Dim nftpTemp As Office.NewFile

Set nftpTemp = Application.NewFileTaskPane

nftpTemp.Add FileName:="C:\Sales_Quarterly.mdb", _
Section:=msoNewfromExistingFile, DisplayName:="Quarterly Sales", _
Action:=msoCreateNewFile
**NewRecord Property**

You can use the `NewRecord` property to determine whether the current record is a new record. Read-only `Integer`.

`expression.NewRecord`

`expression` Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **NewRecord** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>The current record is new.</td>
</tr>
<tr>
<td>False</td>
<td>The current record isn't new.</td>
</tr>
</tbody>
</table>

**Note** The **NewRecord** property is read-only in [Form view](#) and [Datasheet view](#). It isn't available in [Design view](#). This property is available only by using a [macro](#) or [Visual Basic](#).

When a user has moved to a new record, the **NewRecord** property setting will be **True** whether the user has started to edit the record or not.
Example

The following example shows how to use the **NewRecord** property to determine if the current record is a new record. The NewRecordMark procedure sets the current record to the variable `intnewrec`. If the record is new, a message is displayed notifying the user of this. You could run this procedure when the Current event for a form occurs.

```vba
Sub NewRecordMark(frm As Form)
    Dim intnewrec As Integer

    intnewrec = frm.NewRecord
    If intnewrec = True Then
        MsgBox "You're in a new record." & 
        & "Do you want to add new data?" & 
        & "If not, move to an existing record."
    End If
End Sub
```
NewRowOrCol Property

You can use the `NewRowOrCol` property to specify whether a `section` and its associated data is printed in a new row or column within a `multiple-column report` or multiple-column form. Read/write `Byte`.

`expression.NewRowOrCol`

*expression* Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **NewRowOrCol** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>0</td>
<td>(Default) The row or column breaks are determined by the settings in the Page Setup dialog box (available by clicking Page Setup on the File menu) and the available space on the page.</td>
</tr>
<tr>
<td>Before Section</td>
<td>1</td>
<td>Microsoft Access starts printing the current section (the section for which you're setting the property, such as a group header section) in a new row or column. It then prints the next section, such as a detail section, in that same row or column.</td>
</tr>
<tr>
<td>After Section</td>
<td>2</td>
<td>Microsoft Access starts printing the current section, such as a group header section, in the current row or column. It starts printing the next section, such as a detail section, in the next row or column.</td>
</tr>
<tr>
<td>Before &amp; After</td>
<td>3</td>
<td>Microsoft Access starts printing the current section in a new row or column. It starts printing the following section in the next row or column.</td>
</tr>
</tbody>
</table>

You can set this property by using the section's **property sheet**, a **macro**, or **Visual Basic**.

You can set the **NewRowOrCol** property only in **form Design view** or **report Design view**.

The following items are some sample settings for a group header section in a multiple-column report. Make sure **Down, Then Across** is selected under **Column Layout** on the **Columns** tab of the **Page Setup** dialog box.

<table>
<thead>
<tr>
<th>Sample setting</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Before Section</td>
<td>The group header is printed at the top of a new column.</td>
</tr>
</tbody>
</table>
After Section

Before & After

Sections in a form or report are normally printed vertically down a page. The default Column Layout option is Across, then Down. You can print the sections in multiple columns across a page by clicking Down, then Across under Column Layout on the Columns tab of the Page Setup dialog box.

If you set the NewRowOrCol property to Before Section, the vertical or horizontal orientation of the page affects how the section appears when printed. If you click Across, then Down under Column Layout on the Columns tab of the Page Setup dialog box, Microsoft Access starts printing the section at the beginning of a new row; if you click Down, then Across, Microsoft Access starts printing the section at the beginning of a new column.
Example

The following example returns the `NewRowOrCol` property setting and assigns it to the `intGetVal` variable.

```vba
Dim intGetVal As Integer
intGetVal = Me.Section(1).NewRowOrCol
```

The next example presents two layouts for a report that divides data into four groups (Head1 to Head4). Each group includes three to six records, and each record has field a and field b. The layouts differ only in their settings under **Column Layout** on the **Columns** tab of the **Page Setup** dialog box and the values of their `NewRowOrCol` properties. Note that the **Width** box under **Column Size** on the **Columns** tab must be set to the actual width of the field. Also, the Before Section setting of the `NewRowOrCol` property requires a page header section greater than zero for the **Down, then Across** option to function correctly.

- **Column Layout**— **Across, then Down**
  - Head1
    - 1a 1b 2a 2b 3a 3b 4a 4b
  - 5a 5b
  - Head2
    - 1a 1b 2a 2b 3a 3b 4a 4b
  - Head3
    - 1a 1b 2a 2b 3a 3b
  - Head4

- **Column Layout**— **Down, then Across**
  - Head1 Head2 Head3 Head4
    - 1a 1b 1a 1b 1a 1b 1a 1b
  - 2a 2b 2a 2b 2a 2b 2a 2b
  - 3a 3b 3a 3b 3a 3b 3a 3b
  - 4a 4b 4a 4b 4a 4b
  - 5a 5b 5a 5b
  - 6a 6b
1a 1b 2a 2b 3a 3b 4a 4b
5a 5b 6a 6b

- **Grid Settings**— Number of Columns set to 4

- **NewRowOrCol** property setting for group header section— Before & After

- **Grid Settings**— Number of Columns set to 4

- **NewRowOrCol** property setting for group header section— Before Section
NextRecord Property

The **NextRecord** property specifies whether a section should advance to the next record. Read/write **Boolean**.

`expression.NextRecord`

*expression*  Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **NextRecord** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>(Default) The section advances to the next record.</td>
</tr>
<tr>
<td>False</td>
<td>The section doesn't advance to the next record.</td>
</tr>
</tbody>
</table>

To set this property, specify a [macro](#) or event procedure for a section's **OnFormat** property.

Microsoft Access sets this property to **True** before each section's **Format** event.
**Example**

The following example sets the *NextRecord* property to *False* for a given report.

```vbnet
Public Sub ChangeNextRecord(r As Report)
    r.NextRecord = False
End Sub
```
**NumeralShapes Property**

*Language-specific information*

You can use the **NumeralShapes** property to specify or determine numeral shapes to be displayed and printed in a combo box, label, list box, or text box. Read/write **Byte**.

*expression*. **NumeralShapes**

*expression*  Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **NumeralShapes** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>System</td>
<td>0</td>
<td>Numeral shapes determined by the <strong>Numeral Shapes</strong> system setting.</td>
</tr>
<tr>
<td>Arabic</td>
<td>1</td>
<td>Arabic digit shapes will be used to display and print numerals.</td>
</tr>
<tr>
<td>National</td>
<td>2</td>
<td>National digit shapes will be used to display and print numerals.</td>
</tr>
<tr>
<td>Context</td>
<td>3</td>
<td>Numeral shapes determined by <strong>Unicode</strong> context rules for adjacent text.</td>
</tr>
</tbody>
</table>

You can set this property by using the [property sheet](#) or [Visual Basic](#).
Example

The following example changes the **NumeralShapes** property for the selected control to 0 (numeral shapes will be determined by the **Numeral Shapes** system setting).

```vba
Public Sub ChangeNumeralShapes(ctl As Control)
    ctl.NumeralShapes = 0
End Sub
```
Show All
Object Property

You can use the Object property in Visual Basic to return a reference to the ActiveX object that is associated with a linked or embedded OLE object in a control. By using this reference, you can access the properties or invoke the methods of the OLE object.

\[\text{expression} \text{.Object}\]

expression Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **Object** property returns a reference to an ActiveX object. You can use the **Set** statement to assign this ActiveX object to an object variable. The type of object reference returned depends on which application created the OLE object.

The **Object** property setting is read-only in all views.

When you embed or link an OLE object in a Microsoft Access form, you can set properties that determine the type of object and the behavior of the container control. However, you can't directly set or read the OLE object's properties or apply its methods, as you can when performing Automation. The **Object** property returns a reference to an Automation object that represents the linked or embedded OLE object. By using this reference, you can change the OLE object by setting or reading its properties or applying its methods. For example, Microsoft Excel is an **COM component** that supports **Automation**. If you've embedded a Microsoft Excel worksheet in a Microsoft Access form, you can use the **Object** property to set a reference to the **Worksheet** object associated with that worksheet. You can then use any of the properties and methods of the **Worksheet** object.

For information on which properties and methods an ActiveX object supports, see the documentation for the application that was used to create the OLE object.
Example

The following example uses the **Object** property of an unbound object frame named OLE1. Customer name and address information is inserted in an embedded Microsoft Word document formatted as a form letter with placeholders for the name and address information and boilerplate text in the body of the letter. The procedure replaces the placeholder information for each record and prints the form letter. It doesn't save copies of the printed form letter.

```vba
Sub PrintFormLetter_Click()
    Dim objWord As Object
    Dim strCustomer As String, strAddress As String
    Dim strCity As String, strRegion As String

    ' Assign object property of control to variable.
    Set objWord = Me!OLE1.Object.Application.Wordbasic

    ' Assign customer address to variables.
    strCustomer = Me!CompanyName
    strAddress = Me!Address
    strCity = Me!City & ", "
    If Not IsNull(Me!Region) Then
        strRegion = Me!Region
    Else
        strRegion = Me!Country
    End If

    ' Activate ActiveX control.
    Me!OLE1.Action = acOLEActivate

    With objWord
        .StartOfDocument
        ' Go to first placeholder.
        .LineDown 2
        ' Highlight placeholder text.
        .EndOfLine 1
        ' Insert customer name.
        .Insert strCustomer
        ' Go to next placeholder.
        .LineDown
        .StartOfLine
        ' Highlight placeholder text.
        .EndOfLine 1
        ' Insert address.
        .Insert strAddress
        ' Go to last placeholder.
        .LineDown
        .StartOfLine
    End With

```

'Highlight placeholder text.
.EndOfLine 1
' Insert City and Region.
.Insert strCity & strRegion
.FilePrint
.FileClose
End With
Set objWord = Nothing
End Sub
ObjectPalette Property

The **ObjectPalette** property specifies the palette in the application used to create:

- An **OLE object** contained in a **bound object frame**, **chart**, or **unbound object frame**.
- A **bitmap** or other graphic that is loaded into a **command button**, image control, or **toggle button** by using the **Picture** property.

*expression*.ObjectPalette

*expression*  Required. An expression that returns one of the objects in the Applies To list.
Remarks

Microsoft Access sets the value of the ObjectPalette property to a String data type containing the palette information. You can use this setting to set the value of the PaintPalette property for a form or report.

For the following objects, views, and controls, the ObjectPalette property setting is read-only. This property setting is unavailable for these controls in other views.

<table>
<thead>
<tr>
<th>Object</th>
<th>View</th>
<th>Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>Forms</td>
<td>Form</td>
<td>Command button, chart, image control, toggle button, and unbound object frame.</td>
</tr>
<tr>
<td></td>
<td>Design view and Form view</td>
<td></td>
</tr>
<tr>
<td>Forms</td>
<td>Form view</td>
<td>Bound object frame.</td>
</tr>
<tr>
<td>Reports</td>
<td>Report Design view</td>
<td>Command button, chart, image control, toggle button, and unbound object frame. This property setting is unavailable for a bound object frame in all views on a report.</td>
</tr>
</tbody>
</table>

**Note** You can use the setting of this property only in a macro or Visual Basic.

If the application associated with the OLE object, bitmap, or other graphic doesn't have an associated palette, the ObjectPalette property is set to an zerolength string.

The setting of the ObjectPalette property makes the palette of the application associated with the OLE object, bitmap, or other graphic contained in a control available to the PaintPalette property of a form or report. For example, to make the palette used in Microsoft Graph available when you're designing a form in Microsoft Access, you set the form's PaintPalette property to the ObjectPalette value of an existing chart control.

**Note** Windows can have only one color palette active at a time. Microsoft Access allows you to have multiple graphics on a form, each using a different color palette. The PaintPalette and PaletteSource properties let you specify
which color palette a form should use when displaying graphics.
**Example**

The following example sets the **PaintPalette** property of the Seascape form to the **ObjectPalette** property of the Ocean control on the DisplayPictures form. (Ocean can be a bound object frame, command button, chart, toggle button, or unbound object frame.)

\[
\text{Forms!Seascape}.\text{PaintPalette} = _
\]
\[
\text{Forms!DisplayPictures!Ocean}.\text{ObjectPalette}
\]
ObjectVerbs Property

You can use the ObjectVerbs property in Visual Basic to determine the list of verbs an OLE object supports. Read-only String.

expression.ObjectVerbs(Index)

expression Required. An expression that returns one of the objects in the Applies To list.

Index Required Long. An element in the array of supported verbs. This is a zero-based index, meaning zero (0) represents the first verb in the array, one (1) represents the second verb in the array, and so on.
**Remarks**

This property setting isn't available in [Design view](#).

You can use the **ObjectVerbs** property with the **ObjectVerbsCount** property to display a list of the verbs supported by an OLE object. The **Verb** property uses this list of verbs to determine which operation to perform when an OLE object is activated (when the **Action** property is set to **acOLEActivate**).

The **Verb** property setting is the position of a particular verb in the list of verbs returned by the **ObjectVerbs** property. For example, 1 specifies the first verb in the list (the Visual Basic command `objectVerbs(0)`, or the first verb in the **ObjectVerbs** property array), 2 specifies the second verb in the list (the Visual Basic command `objectVerbs(1)`, or the second verb in the **ObjectVerbs** property array), and so on.

The first verb in the **ObjectVerbs** property array, called by the Visual Basic command `objectVerbs(0)`, is the default verb. If the **Verb** property hasn't been set, this verb specifies the operation performed when the OLE object is activated.

Applications that expose OLE objects typically include the **Object** command on the **Edit** menu. When the user points to the **Object** command, a submenu displays the object's verbs. You can use the **ObjectVerbs** and **ObjectVerbsCount** properties to display a list of verbs in a form or report instead of on a menu.

The list of verbs an object supports varies, depending on the state of the object. To update the list of verbs an object supports, set the control's **Action** property to **acOLEFetchVerbs**. Be sure to update the list of verbs before presenting it to the user.
Example

The following example returns the verbs supported by the OLE object in the OLE1 control and displays each verb in a message box.

```vba
Sub GetVerbList(frm As Form, OLE1 As Control)
    Dim intX As Integer, intNumVerbs As Integer
    Dim strVerbList As String

    ' Update verb list.
    With frm!OLE1
        .Action = acOLEFetchVerbs
        intNumVerbs = .ObjectVerbsCount
        For intX = 0 To intNumVerbs - 1
            strVerbList = strVerbList & .ObjectVerbs(intX) & "; "
        Next intX
    End With

    ' Display verbs in message box.
    MsgBox Left(strVerbList, Len(strVerbList) - 2)
End Sub
```
ObjectVerbsCount Property

You can use the **ObjectVerbsCount** property in Visual Basic to determine the number of verbs supported by an **OLE object**. Read/write **Long**.

*expression*.ObjectVerbsCount

*expression*  Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **ObjectVerbsCount** property setting is a value that specifies the number of elements in the **ObjectVerbs** property array.

This property setting isn't available in **Design view**.

The list of verbs an OLE object supports may vary, depending on the state of the object. To update the list of supported verbs, set the control's **Action** property to **acOLEFetchVerbs**.
Example

The following example returns the verbs supported by the OLE object in the OLE1 control and displays each verb in a message box.

Sub GetVerbList(frm As Form, OLE1 As Control)
    Dim intX As Integer, intNumVerbs As Integer
    Dim strVerbList As String
    ' Update verb list.
    With frm!OLE1
        .Action = acOLEFetchVerbs
        intNumVerbs = .ObjectVerbsCount
        For intX = 0 To intNumVerbs - 1
            strVerbList = strVerbList & .ObjectVerbs(intX) & "; "
        Next intX
    End With
    ' Display verbs in message box.
    MsgBox Left(strVerbList, Len(strVerbList) - 2)
End Sub
OldBorderStyle Property

You can use this property to set or returns the unedited value of the BorderStyle property for a form or control. This property is useful if you need to revert to an unedited or preferred border style. Read/write Byte.

expression.OldBorderStyle

expression Required. An expression that returns one of the objects in the Applies To list.
### Remarks

For forms, the **BorderStyle** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>0</td>
<td>The form has no border or related border elements. The form isn't resizable. The form has a thin border and can include any of the border elements. The form isn't resizable (the <strong>Size</strong> command on the <strong>Control</strong> menu isn't available). You often use this setting for pop-up forms. (If you want a form to remain on top of all Microsoft Access windows, you must also set its <strong>PopUp</strong> property to Yes.)</td>
</tr>
<tr>
<td>Thin</td>
<td>1</td>
<td><strong>Sizable</strong> (Default) The form has the default border for Microsoft Access forms, can include any of the border elements, and can be resized. You often use this setting for normal Microsoft Access forms. The form has a thick (double) border and can include only a title bar, a <strong>Close</strong> button, and a <strong>Control</strong> menu. The form can't be maximized, minimized, or resized (the <strong>Maximize</strong>, <strong>Minimize</strong>, and <strong>Size</strong> commands aren't available on the <strong>Control</strong> menu). You often use this setting for custom dialog boxes. (If you want a form to be <strong>modal</strong>, however, you must also set its <strong>Modal</strong> property to Yes. If you want it to be a modal pop-up form, which dialog boxes typically are, you must set both its <strong>PopUp</strong> and <strong>Modal</strong> properties to Yes.)</td>
</tr>
</tbody>
</table>

For controls, the **OldBorderStyle** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transparent</td>
<td>0</td>
<td>(Default only for <strong>label</strong>, <strong>chart</strong>, and <strong>subreport</strong>) Transparent</td>
</tr>
<tr>
<td>Solid</td>
<td>1</td>
<td>(Default) Solid line</td>
</tr>
<tr>
<td>Dashes</td>
<td>2</td>
<td>Dashed line</td>
</tr>
<tr>
<td>Short</td>
<td>3</td>
<td>Dashed line with short dashes</td>
</tr>
<tr>
<td></td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>--------------</td>
<td>-------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>Dots</td>
<td>4 Dotted line</td>
<td></td>
</tr>
<tr>
<td>Sparse dots</td>
<td>5 Dotted line with dots spaced far apart</td>
<td></td>
</tr>
<tr>
<td>Dash dot</td>
<td>6 Line with a dash-dot combination</td>
<td></td>
</tr>
<tr>
<td>Dash dot dot</td>
<td>7 Line with a dash-dot-dot combination</td>
<td></td>
</tr>
<tr>
<td>Double solid</td>
<td>8 Double solid lines</td>
<td></td>
</tr>
</tbody>
</table>

- If the **OldBorderStyle** property is set to None or Dialog, the form doesn't have **Maximize** or **Minimize** buttons, regardless of its **MinMaxButtons** property setting.

- If the **OldBorderStyle** property is set to None, the form doesn't have a **Control** menu, regardless of its **ControlBox** property setting.

- The **OldBorderStyle** property setting doesn't affect the display of the scroll bars, **navigation buttons**, the **record number box**, or **record selectors**.
**Example**

The following example demonstrates the effect of changing a control's `BorderStyle` property, while leaving the `OldBorderStyle` unaffected. The example concludes with setting the `BorderStyle` property to its original unedited value.

```vba
With Forms("Order Entry").Controls("Zip Code")
    .BorderStyle = 3 ' Short dashed border.
    MsgBox "BorderStyle = " & .BorderStyle & vbCrLf & _
        "OldBorderStyle = " & .OldBorderStyle ' Prints 3, 1.

    .BorderStyle = 2 ' Dashed border.
    MsgBox "BorderStyle = " & .BorderStyle & vbCrLf & _
        "OldBorderStyle = " & .OldBorderStyle ' Prints 2, 1

    .BorderStyle = .OldBorderStyle ' Solid (default) border.
    MsgBox "BorderStyle = " & .BorderStyle & vbCrLf & _
        "OldBorderStyle = " & .OldBorderStyle ' Prints 1, 1

End With
```
OldValue Property

You can use the OldValue property to determine the unedited value of a bound control. Read-only Variant.

_expression_.OldValue

_expression_ Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **OldValue** property contains the unedited data from a bound control and is read-only in all views.

You can access this property only by using a macro or *Visual Basic*.

The **OldValue** property can be assigned to a variable by using the following syntax:

```
OriginalValue = Forms!Customers!AmountPaid.OldValue
```

Microsoft Access uses the **OldValue** property to store the original value of a bound control. When you edit a bound control on a form, your changes aren't saved until you move to another record. The **OldValue** property contains the unedited version of the underlying data.

You can provide your own undo capability by assigning the **OldValue** property setting to a control. The following example shows how you can undo any changes to *text box* controls on a form:

```vba
Private Sub btnUndo_Click()
    Dim ctlTextbox As Control
    For Each ctlTextbox In Me.Controls
        If ctlTextbox.ControlType = acTextBox Then
            ctlTextbox.Value = ctl.OldValue
        End If
    Next ctlTextbox
End Sub
```

If the control hasn't been edited, this code has no effect. When you move to another record, the record source is updated, so the current value and the **OldValue** property will be the same.

The **OldValue** property setting has the same data type as the field to which the control is bound.
Example

The following example checks to determine if new data entered in a field is within 10 percent of the value of the original data. If the change is greater than 10 percent, the **OldValue** property is used to restore the original value. This procedure could be called from the BeforeUpdate event of the control that contains data you want to validate.

```vbscript
Public Sub Validate_Field()
    Dim curNewValue As Currency
    Dim curOriginalValue As Currency
    Dim curChange As Currency
    Dim strMsg As String

    curNewValue = Forms!Products!UnitPrice
    curOriginalValue = Forms!Products!UnitPrice.OldValue
    curChange = Abs(curNewValue - curOriginalValue)

    If curChange > (curOriginalValue * .1) Then
        strMsg = "Change is more than 10% of original unit price. " & "Restoring original unit price."
        MsgBox strMsg, vbExclamation, "Invalid change."
        Forms!Products!UnitPrice = curOriginalValue
    End If
End Sub
```
**OLEClass Property**

You can use the **OLEClass** property to obtain a description of the kind of **OLE object** contained in a **chart control** or an **unbound object frame**. Read/write **String**.

`expression.OLEClass`

`expression` Required. An expression that returns one of the objects in the Applies To list.
Remarks

This property is set automatically in the control's property sheet to a string expression when you click Object on the Insert menu to add an OLE object to a form. The OLEClass property setting is read-only in all views.

Note  If you are using Automation (formerly called OLE Automation) and need to specify a name to refer to the OLE object, use the Class property.

The OLEClass property and the Class property are similar but not identical. The OLEClass property setting is a general description of the OLE object whereas the Class property setting is the name used to refer to the OLE object in Visual Basic. Examples of OLEClass property settings are Microsoft Excel Chart, Microsoft Word Document, and Paintbrush Picture.
Example

The following example displays a message indicating the OLE class for the "Customer Picture" unbound object frame on the "Order Entry" form.

MsgBox "The OLE class = " & Forms("Order Entry").Controls("Customer
OnActivate Property

Sets or returns the value of the *On Activate* box in the *Properties* window of a form or report. Read/write *String*.

*expression*.OnActivate

*expression*   Required. An expression that returns one of the objects in the Applies To list.
Remarks

This property is helpful for programmatically changing the action Microsoft Access takes when an event is triggered. For example, between event calls you may want to change an expression's parameters, or switch from an event procedure to an expression or macro, depending on the circumstances under which the event was triggered.

The **Activate** event occurs when the form or report receives the focus and becomes the active window.

The **OnActivate** value will be one of the following, depending on the selection chosen in the **Choose Builder** window (accessed by clicking the **Build** button next to the **On Activate** box in the form or report's **Properties** window):

- If Expression Builder is chosen, the value will be "'=expression'", where *expression* is the expression from the Expression Builder window.
- If Macro Builder is chosen, the value is the name of the macro.
- If Code Builder is chosen, the value will be "'[Event Procedure]'".

If the **On Activate** box is blank, the property value is an empty string.
Example

The following example associates the **Activate** event with the macro "Activate_Macro" for the "Order Entry" form.

```vba
Forms("Order Entry").OnActivate = "Activate_Macro"
```
OnApplyFilter Property

Sets or returns the value of the **On Apply Filter** box in the **Properties** window of a form. Read/write **String**.

`expression.OnApplyFilter`

`expression` Required. An expression that returns one of the objects in the Applies To list.
Remarks

This property is helpful for programmatically changing the action Microsoft Access takes when an event is triggered. For example, between event calls you may want to change an expression's parameters, or switch from an event procedure to an expression or macro, depending on the circumstances under which the event was triggered.

The **Apply Filter** event occurs when a filter is applied or removed.

The **OnApplyFilter** value will be one of the following, depending on the selection chosen in the **Choose Builder** window (accessed by clicking the **Build** button next to the **On Apply Filter** box in the form's **Properties** window):

- If Expression Builder is chosen, the value will be "'=expression'", where *expression* is the expression from the Expression Builder window.
- If Macro Builder is chosen, the value is the name of the macro.
- If Code Builder is chosen, the value will be "[Event Procedure]".

If the **On Apply Filter** box is blank, the property value is an empty string.
Example

The following example associates the OnApplyFilter property for the "Order Entry" form to the event "Form_ApplyFilter".

Forms("Order Entry").OnFilter = "[Event Procedure]"
OnChange Property

Sets or returns the value of the On Change box in the Properties window of one of the objects in the Applies To list. Read/write String.

expression.OnChange

expression  Required. An expression that returns one of the objects in the Applies To list.
Remarks

This property is helpful for programmatically changing the action Microsoft Access takes when an event is triggered. For example, between event calls you may want to change an expression's parameters, or switch from an event procedure to an expression or macro, depending on the circumstances under which the event was triggered.

The **Change** event occurs when the contents of a text box or the text portion of a combo box changes. It also occurs when you move from one page to another page in a tab control.

The **OnChange** value will be one of the following, depending on the selection chosen in the **Choose Builder** window (accessed by clicking the **Build** button next to the **On Change** box in the object's **Properties** window):

- If Expression Builder is chosen, the value will be "=expression", where *expression* is the expression from the Expression Builder window.
- If Macro Builder is chosen, the value is the name of the macro.
- If Code Builder is chosen, the value will be "[Event Procedure]".

If the **On Change** box is blank, the property value is an empty string.
**Example**

The following example prints the value of the **OnChange** property in the Immediate window for the "Address" text box on the "Order Entry" form.

Debug.Print Forms("Order Entry").Controls("Address").**OnChange**
OnClick Property

Sets or returns the value of the **On Click** box in the Properties window of one of the objects in the Applies To list. Read/write **String**.

*expression*.OnClick

*expression* Required. An expression that returns one of the objects in the Applies To list.
Remarks

This property is helpful for programmatically changing the action Microsoft Access takes when an event is triggered. For example, between event calls you may want to change an expression's parameters, or switch from an event procedure to an expression or macro, depending on the circumstances under which the event was triggered.

The **Click** event occurs when a user presses and releases the left mouse button over an object.

The **OnClick** value will be one of the following, depending on the selection chosen in the **Choose Builder** window (accessed by clicking the **Build** button next to the **On Click** box in the object's **Properties** window):

- If Expression Builder is chosen, the value will be "'=expression'", where *expression* is the expression from the Expression Builder window.
- If Macro Builder is chosen, the value is the name of the macro.
- If Code Builder is chosen, the value will be "'[Event Procedure]'".

If the **On Click** box is blank, the property value is an empty string.
Example

The following example associates the **Click** event with the "OK_Click" event procedure for the button named "OK" on the "Order Entry" form, if there is currently no association.

```vba
With Forms("Order Entry").Controls("OK")
    If .OnClick = "" Then
        .OnClick = "[Event Procedure]"
    End If
End With
```
OnClose Property

Sets or returns the value of the On Close box in the Properties window of a form or report. Read/write String.

expression.OnClose

expression  Required. An expression that returns one of the objects in the Applies To list.
Remarks

This property is helpful for programmatically changing the action Microsoft Access takes when an event is triggered. For example, between event calls you may want to change an expression's parameters, or switch from an event procedure to an expression or macro, depending on the circumstances under which the event was triggered.

The Close event occurs when a form or report is closed and removed from the screen.

The OnClose value will be one of the following, depending on the selection chosen in the Choose Builder window (accessed by clicking the Build button next to the On Close box in the form or report's Properties window):

- If Expression Builder is chosen, the value will be "=expression", where expression is the expression from the Expression Builder window.
- If Macro Builder is chosen, the value is the name of the macro.
- If Code Builder is chosen, the value will be "[Event Procedure]".

If the On Close box is blank, the property value is an empty string.
Example

The following example prints the value of the **OnOpen** property in the Immediate window for the "Purchase Order" report.

```
Debug.Print Reports("Purchase Order").OnClose
```
OnConnect Property

Returns or sets a String indicating which macro, event procedure, or user-defined function runs when the OnConnect event occurs. Read/write.

expression.OnConnect

expression  Required. An expression that returns one of the objects in the Applies To list.
Remarks

Valid values for this property are "macroname" where macroname is the name of macro, "[Event Procedure]" which indicates the event procedure associated with the OnConnect event for the specified object, or "=functionname()" where functionname is the name of a user-defined function. For a more detailed discussion of event properties, see "Event Properties."
Example

The following example specifies that when the OnConnect event occurs on the first form of the current project, the associated event procedure should run.

`Forms(0).OnConnect = "[Event Procedure]"`
OnCurrent Property

Sets or returns the value of the On Current box in the Properties window of a form. Read/write String.

expression.OnCurrent

expression  Required. An expression that returns one of the objects in the Applies To list.
Remarks

This property is helpful for programmatically changing the action Microsoft Access takes when an event is triggered. For example, between event calls you may want to change an expression's parameters, or switch from an event procedure to an expression or macro, depending on the circumstances under which the event was triggered.

The Current event occurs when the focus moves to a record, making it the current record, or when the form is refreshed or requiered.

The OnCurrent value will be one of the following, depending on the selection chosen in the Choose Builder window (accessed by clicking the Build button next to the On Current box in the form or report's Properties window):

- If Expression Builder is chosen, the value will be "=expression", where expression is the expression from the Expression Builder window.
- If Macro Builder is chosen, the value is the name of the macro.
- If Code Builder is chosen, the value will be "[Event Procedure]".

If the On Current box is blank, the property value is an empty string.
Example

The following example associates the **Current** event with the macro "Current_Macro" for the "Order Entry" form.

Forms("Order Entry").**OnDeactivate** = "Current_Macro"
OnDblClick Property

Sets or returns the value of the On Dbl Click box in the Properties window of one of the objects in the Applies To list. Read/write String.

expression.OnDblClick

expression  Required. An expression that returns one of the objects in the Applies To list.
Remarks

This property is helpful for programmatically changing the action Microsoft Access takes when an event is triggered. For example, between event calls you may want to change an expression's parameters, or switch from an event procedure to an expression or macro, depending on the circumstances under which the event was triggered.

The **DblClick** event occurs when a user presses and releases the left mouse button twice over an object within the double-click time limit of the system.

The **OnDblClick** value will be one of the following, depending on the selection chosen in the Choose Builder window (accessed by clicking the Build button next to the On Dbl Click box in the object's Properties window):

- If Expression Builder is chosen, the value will be "=expression", where *expression* is the expression from the Expression Builder window.
- If Macro Builder is chosen, the value is the name of the macro.
- If Code Builder is chosen, the value will be "[Event Procedure]".

If the On Dbl Click box is blank, the property value is an empty string.
Example

The following example associates the **Db1_Click** event with the "OK_DblClick" event procedure for the button named "OK" on the "Order Entry" form, if there is currently no association.

With Forms("Order Entry").Controls("OK")

If **OnDblClick** = "" Then

**OnDblClick** = "[Event Procedure]"

End If
End With
OnDeactivate Property

Sets or returns the value of the **On Deactivate** box in the **Properties** window of a form or report. Read/write **String**.

*expression*.OnDeactivate

*expression* Required. An expression that returns one of the objects in the Applies To list.
Remarks

This property is helpful for programmatically changing the action Microsoft Access takes when an event is triggered. For example, between event calls you may want to change an expression's parameters, or switch from an event procedure to an expression or macro, depending on the circumstances under which the event was triggered.

The **Deactivate** event occurs when the form or report loses the focus to a Table, Query, Form, Report, Macro, or Module window, or to the Database window.

The **OnDeactivate** value will be one of the following, depending on the selection chosen in the **Choose Builder** window (accessed by clicking the **Build** button next to the **On Deactivate** box in the form or report's **Properties** window):

- If Expression Builder is chosen, the value will be "=expression", where `expression` is the expression from the Expression Builder window.
- If Macro Builder is chosen, the value is the name of the macro.
- If Code Builder is chosen, the value will be "[Event Procedure]".

If the **On Deactivate** box is blank, the property value is an empty string.
Example

The following example associates the **Deactivate** event with the macro "Deactivate_Macro" for the "Order Entry" form.

Forms("Order Entry").**OnDeactivate** = "Deactivate_Macro"
OnDelete Property

Sets or returns the value of the **On Delete** box in the **Properties** window of a form. Read/write **String**.

*expression*.OnDelete

*expression*  Required. An expression that returns one of the objects in the Applies To list.
Remarks

This property is helpful for programmatically changing the action Microsoft Access takes when an event is triggered. For example, between event calls you may want to change an expression's parameters, or switch from an event procedure to an expression or macro, depending on the circumstances under which the event was triggered.

The Delete event occurs when the user performs some action, such as pressing the DEL key to delete a record, but before the record is actually deleted.

The OnDelete value will be one of the following, depending on the selection chosen in the Choose Builder window (accessed by clicking the Build button next to the On Delete box in the form's Properties window):

- If Expression Builder is chosen, the value will be "=expression", where expression is the expression from the Expression Builder window.
- If Macro Builder is chosen, the value is the name of the macro.
- If Code Builder is chosen, the value will be ":Event Procedure]".

If the On Delete box is blank, the property value is an empty string.
Example

The following example associates the **Delete** event with the "Form_Delete" event for the "Order Entry" form.

```
Forms("Order Entry").OnDelete = "[Event Procedure]"
```
OnDirty Property

Sets or returns the value of the On Dirty box in the Properties window of one of the objects in the Applies To list. Read/write String.

expression.OnDirty

expression  Required. An expression that returns one of the objects in the Applies To list.
Remarks

This property is helpful for programmatically changing the action Microsoft Access takes when an event is triggered. For example, between event calls you may want to change an expression's parameters, or switch from an event procedure to an expression or macro, depending on the circumstances under which the event was triggered.

The Dirty event occurs when the contents of a form or the text portion of a combo box changes. It also occurs when you move from one page to another page in a tab control.

The OnDirty value will be one of the following, depending on the selection chosen in the Choose Builder window (accessed by clicking Build in the object's Properties window):

- If Expression Builder is chosen, the value will be "=expression", where expression is the expression from the Expression Builder window.
- If Macro Builder is chosen, the value is the name of the macro.
- If Code Builder is chosen, the value will be "[Event Procedure]".

If the On Dirty box is blank, the property value is an empty string.
Example

The following example prints the value of the **OnDirty** property in the Immediate window for the "Order Entry" form.

Debug.Print Forms("Order Entry").**OnDirty**
OnDisconnect Property

Returns or sets a **String** indicating which macro, event procedure, or user-defined function runs when the **OnDisconnect** event occurs. Read/write.

\[ \text{expression} \cdot \text{OnDisconnect} \]

\[ \text{expression} \quad \text{Required. An expression that returns one of the objects in the Applies To list.} \]
Remarks

Valid values for this property are "macroname" where macroname is the name of macro, "[Event Procedure]" which indicates the event procedure associated with the OnDisconnect event for the specified object, or "=functionname()" where functionname is the name of a user-defined function. For a more detailed discussion of event properties, see "Event Properties."
Example

The following example specifies that when the OnDisconnect event occurs on the first form of the current project, the associated event procedure should run.

```
Forms(0).OnDisconnect = "[Event Procedure]"
```
**OnEnter Property**

Sets or returns the value of the **On Enter** box in the **Properties** window of one of the objects in the Applies To list. Read/write **String**.

*expression*. **OnEnter**

*expression*   Required. An expression that returns one of the objects in the Applies To list.
Remarks

This property is helpful for programmatically changing the action Microsoft Access takes when an event is triggered. For example, between event calls you may want to change an expression's parameters, or switch from an event procedure to an expression or macro, depending on the circumstances under which the event was triggered.

The **Enter** event occurs before a control actually receives the focus from a control on the same form.

The **OnEnter** value will be one of the following, depending on the selection chosen in the **Choose Builder** window (accessed by clicking the **Build** button next to the **On Enter** box in the object's **Properties** window):

- If Expression Builder is chosen, the value will be "=expression", where *expression* is the expression from the Expression Builder window.
- If Macro Builder is chosen, the value is the name of the macro.
- If Code Builder is chosen, the value will be "[Event Procedure]".

If the **On Enter** box is blank, the property value is an empty string.
Example

The following example associates the Enter event with the macro "Enter_Macro" for the button named "OK" on the "Order Entry" form.

Forms("Order Entry").Controls("OK").OnEnter = "Enter_Macro"
OnError Property

Sets or returns the value of the OnError box in the Properties window for a form or report. Read/write String.

expression.OnError

expression   Required. An expression that returns one of the objects in the Applies To list.
**Remarks**

This property is helpful for programmatically changing the action Microsoft Access takes when an event is triggered. For example, between event calls you may want to change an expression's parameters, or switch from an event procedure to an expression or macro, depending on the circumstances under which the event was triggered.

The **Error** event occurs when a run-time error is produced in Microsoft Access when a form or report has the focus. This includes Microsoft Jet database engine errors, but not run-time errors in Visual Basic.

The **OnError** value will be one of the following, depending on the selection chosen in the **Choose Builder** window (accessed by clicking the **Build** button next to the **On Error** box in the form or report's **Properties** window):

- If Expression Builder is chosen, the value will be "=expression", where *expression* is the expression from the Expression Builder window.
- If Macro Builder is chosen, the value is the name of the macro.
- If Code Builder is chosen, the value will be "[Event Procedure]".

If the **On Error** box is blank, the property value is an empty string.
**Example**

The following example associates the **Error** event with the macro "Error_Handler_Macro" for the "Order Entry" form.

`Forms("Order Entry").OnError = "Error_Handler_Macro"`
OnExit Property

Sets or returns the value of the On Exit box in the Properties window of one of the objects in the Applies To list. Read/write String.

expression.OnExit

expression Required. An expression that returns one of the objects in the Applies To list.
Remarks

This property is helpful for programmatically changing the action Microsoft Access takes when an event is triggered. For example, between event calls you may want to change an expression's parameters, or switch from an event procedure to an expression or macro, depending on the circumstances under which the event was triggered.

The Exit event occurs just before a control loses the focus to another control on the same form.

The On Exit value will be one of the following, depending on the selection chosen in the Choose Builder window (accessed by clicking the Build button next to the On Exit box in the object's Properties window):

- If Expression Builder is chosen, the value will be "=expression", where expression is the expression from the Expression Builder window.
- If Macro Builder is chosen, the value is the name of the macro.
- If Code Builder is chosen, the value will be "[Event Procedure]".

If the On Exit box is blank, the property value is an empty string.
Example

The following example associates the **Exit** event with the macro "Exit_Macro" for the button named "OK" on the "Order Entry" form.

```
Forms("Order Entry").Controls("OK").**OnExit** = "Exit_Macro"
```
OnFilter Property

Sets or returns the value of the On Filter box in the Properties window of a form. Read/write String.

expression.OnFilter

expression Required. An expression that returns one of the objects in the Applies To list.
Remarks

This property is helpful for programmatically changing the action Microsoft Access takes when an event is triggered. For example, between event calls you may want to change an expression's parameters, or switch from an event procedure to an expression or macro, depending on the circumstances under which the event was triggered.

The **Filter** event occurs when a form is opened and its records are displayed.

The **OnFilter** value will be one of the following, depending on the selection chosen in the **Choose Builder** window (accessed by clicking the **Build** button next to the **On Filter** box in the form's **Properties** window):

- If Expression Builder is chosen, the value will be "=expression", where *expression* is the expression from the Expression Builder window.
- If Macro Builder is chosen, the value is the name of the macro.
- If Code Builder is chosen, the value will be "[Event Procedure]".

If the **On Filter** box is blank, the property value is an empty string.
Example

The following example associates the Filter property for the "Order Entry" form to the event "Form_Filter".

Forms("Order Entry").**OnFilter** = "[Event Procedure]"
OnFormat Property

Sets or returns the value of the On Format box in the Properties window of a report section. Read/write String.

expression.OnFormat

expression Required. An expression that returns one of the objects in the Applies To list.
Remarks

This property is helpful for programatically changing the action Microsoft Access takes when an event is triggered. For example, between event calls you may want to change an expression's parameters, or switch from an event procedure to an expression or macro, depending on the circumstances under which the event was triggered.

The **Format** event occurs when Microsoft Access determines which data belongs in a report section, but before Access formats the section for previewing or printing.

The **OnFormat** value will be one of the following, depending on the selection chosen in the **Choose Builder** window (accessed by clicking the **Build** button next to the **On Format** box in the report section's **Properties** window):

- If Expression Builder is chosen, the value will be "=expression", where *expression* is the expression from the Expression Builder window.
- If Macro Builder is chosen, the value is the name of the macro.
- If Code Builder is chosen, the value will be "[Event Procedure]".

If the **On Format** box is blank, the property value is an empty string.
Example

The following example prints the value of the OnFormat property in the Immediate window for the "GroupHeader0" section in the "Purchase Order" report.

Debug.Print Reports("Purchase Order").Section("GroupHeader0").OnFormat
OnGotFocus Property

Sets or returns the value of the On Got Focus box in the Properties window of one of the objects in the Applies To list. Read/write String.

expression.OnGotFocus

expression  Required. An expression that returns one of the objects in the Applies To list.
Remarks

This property is helpful for programmatically changing the action Microsoft Access takes when an event is triggered. For example, between event calls you may want to change an expression's parameters, or switch from an event procedure to an expression or macro, depending on the circumstances under which the event was triggered.

The GotFocus event occurs when the object receives the focus.

The OnGotFocus value will be one of the following, depending on the selection chosen in the Choose Builder window (accessed by clicking the Build button next to the On Got Focus box in the object's Properties window):

- If Expression Builder is chosen, the value will be "=expression", where expression is the expression from the Expression Builder window.
- If Macro Builder is chosen, the value is the name of the macro.
- If Code Builder is chosen, the value will be "[Event Procedure]".

If the On Got Focus box is blank, the property value is an empty string.
Example

The following example prints the value of the OnGotFocus property in the Immediate window for the button named "OK" on the "Order Entry" form.

Debug.Print Forms("Order Entry").Controls("OK").OnGotFocus
OnInsert Property

Sets or returns the value of the Before Insert box in the Properties window of a form. Read/write String.

expression.OnInsert

expression Required. An expression that returns one of the objects in the Applies To list.
Remarks

This property is helpful for programmatically changing the action Microsoft Access takes when an event is triggered. For example, between event calls you may want to change an expression's parameters, or switch from an event procedure to an expression or macro, depending on the circumstances under which the event was triggered.

Although the name of this property is `OnInsert`, setting this property actually sets the value of the `Before Insert` box.

The `BeforeInsert` event occurs when the user types the first character in a new record, but before the record is actually created.

The `OnInsert` value will be one of the following, depending on the selection chosen in the `Choose Builder` window (accessed by clicking the `Build` button next to the `Before Insert` box in the form's `Properties` window):

- If Expression Builder is chosen, the value will be `"=expression"`, where `expression` is the expression from the Expression Builder window.
- If Macro Builder is chosen, the value is the name of the macro.
- If Code Builder is chosen, the value will be `"[Event Procedure]"`.

If the `Before Insert` box is blank, the property value is an empty string.
Example

The following example prints the value of the OnInsert property in the Immediate window for the "Order Entry" form.

Debug.Print Forms("Order Entry").OnInsert
OnKeyDown Property

Sets or returns the value of the On Key Down box in the Properties window of one of the objects in the Applies To list. Read/write String.

expression.OnKeyDown

expression  Required. An expression that returns one of the objects in the Applies To list.
Remarks

This property is helpful for programmatically changing the action Microsoft Access takes when an event is triggered. For example, between event calls you may want to change an expression's parameters, or switch from an event procedure to an expression or macro, depending on the circumstances under which the event was triggered.

The **KeyDown** event occurs when a user presses a key while a form or control has the focus. This event also occurs if you send a keystroke to a form or control by using the SendKeys action in a macro or the **SendKeys** statement in Visual Basic.

The **OnKeyDown** value will be one of the following, depending on the selection chosen in the Choose Builder window (accessed by clicking the **Build** button next to the **On Key Down** box in the object's **Properties** window):

- If Expression Builder is chosen, the value will be "=expression", where *expression* is the expression from the Expression Builder window.
- If Macro Builder is chosen, the value is the name of the macro.
- If Code Builder is chosen, the value will be "[Event Procedure]".

If the **On Key Down** box is blank, the property value is an empty string.
Example

The following example prints the value of the **OnKeyDown** property in the Immediate window for the button named "OK" on the "Order Entry" form.

```vba
Debug.Print Forms("Order Entry").Controls("OK").OnKeyDown
```
OnKeyPress Property

Sets or returns the value of the On Key Press box in the Properties window of one of the objects in the Applies To list. Read/write String.

expression.OnKeyPress

expression  Required. An expression that returns one of the objects in the Applies To list.
**Remarks**

This property is helpful for programmatically changing the action Microsoft Access takes when an event is triggered. For example, between event calls you may want to change an expression's parameters, or switch from an event procedure to an expression or macro, depending on the circumstances under which the event was triggered.

The **KeyPress** event occurs when a user presses and releases a key or key combination that corresponds to an ANSI code while a form or control has the focus. This event also occurs if you send an ANSI keystroke to a form or control by using the SendKeys action in a macro or the **SendKeys** statement in Visual Basic.

The **OnKeyPress** value will be one of the following, depending on the selection chosen in the **Choose Builder** window (accessed by clicking the **Build** button next to the **On Key Press** box in the object's **Properties** window):

- If Expression Builder is chosen, the value will be "=expression", where \textit{expression} is the expression from the Expression Builder window.
- If Macro Builder is chosen, the value is the name of the macro.
- If Code Builder is chosen, the value will be "[Event Procedure]".

If the **On Key Press** box is blank, the property value is an empty string.
Example

The following example prints the value of the **OnKeyPress** property in the Immediate window for the button named "OK" on the "Order Entry" form.

```vbs
Debug.Print Forms("Order Entry").Controls("OK").OnKeyPress
```
OnKeyUp Property

Sets or returns the value of the On Key Up box in the Properties window of one of the objects in the Applies To list. Read/write String.

```plaintext
expression.OnKeyUp
```

`expression` Required. An expression that returns one of the objects in the Applies To list.
Remarks

This property is helpful for programmatically changing the action Microsoft Access takes when an event is triggered. For example, between event calls you may want to change an expression's parameters, or switch from an event procedure to an expression or macro, depending on the circumstances under which the event was triggered.

The **KeyUp** event occurs when a user releases a key while a form or control has the focus. This event also occurs if you send a keystroke to a form or control by using the SendKeys action in a macro or the **SendKeys** statement in Visual Basic.

The **OnKeyUp** value will be one of the following, depending on the selection chosen in the Choose Builder window (accessed by clicking the Build button next to the On Key Up box in the object's Properties window):

- If Expression Builder is chosen, the value will be "=expression", where expression is the expression from the Expression Builder window.
- If Macro Builder is chosen, the value is the name of the macro.
- If Code Builder is chosen, the value will be "[Event Procedure]".

If the On Key Up box is blank, the property value is an empty string.
Example

The following example prints the value of the **OnKeyUp** property in the Immediate window for the button named "OK" on the "Order Entry" form.

```
Debug.Print Forms("Order Entry").Controls("OK").OnKeyUp
```
OnLoad Property

Sets or returns the value of the On Load box in the Properties window of a form. Read/write String.

expression.OnLoad

expression  Required. An expression that returns one of the objects in the Applies To list.
**Remarks**

This property is helpful for programmatically changing the action Microsoft Access takes when an event is triggered. For example, between event calls you may want to change an expression's parameters, or switch from an event procedure to an expression or macro, depending on the circumstances under which the event was triggered.

The **Load** event occurs when a form is opened and its records are displayed.

The **OnLoad** value will be one of the following, depending on the selection chosen in the **Choose Builder** window (accessed by clicking the **Build** button next to the **On Load** box in the form's **Properties** window):

- If Expression Builder is chosen, the value will be "=expression", where *expression* is the expression from the Expression Builder window.
- If Macro Builder is chosen, the value is the name of the macro.
- If Code Builder is chosen, the value will be "[Event Procedure]".

If the **On Load** box is blank, the property value is an empty string.
Example

The following example prints the value of the **OnLoad** property in the Immediate window for the "Order Entry" form.

```vba
Debug.Print Forms("Order Entry").OnLoad
```
OnLostFocus Property

Sets or returns the value of the On Lost Focus box in the Properties window of one of the objects in the Applies To list. Read/write String.

expression. OnLostFocus

expression  Required. An expression that returns one of the objects in the Applies To list.
Remarks

This property is helpful for programatically changing the action Microsoft Access takes when an event is triggered. For example, between event calls you may want to change an expression's parameters, or switch from an event procedure to an expression or macro, depending on the circumstances under which the event was triggered.

The **LostFocus** event occurs when the object loses the focus.

The **OnLostFocus** value will be one of the following, depending on the selection chosen in the **Choose Builder** window (accessed by clicking the **Build** button next to the **On Lost Focus** box in the object's **Properties** window):

- If Expression Builder is chosen, the value will be "'=expression'", where *expression* is the expression from the Expression Builder window.
- If Macro Builder is chosen, the value is the name of the macro.
- If Code Builder is chosen, the value will be "[Event Procedure]".

If the **On Lost Focus** box is blank, the property value is an empty string.
Example

The following example prints the value of the `OnLostFocus` property in the Immediate window for the button named "OK" on the "Order Entry" form.

```
Debug.Print Forms("Order Entry").Controls("OK").OnLostFocus
```
OnMenu Property

Sets or returns the value of the On Menu box in the Properties window of a form or report. Read/write String.

expression.OnMenu

expression  Required. An expression that returns one of the objects in the Applies To list.
Remarks

This property is helpful for programmatically changing the action Microsoft Access takes when an event is triggered. For example, between event calls you may want to change an expression's parameters, or switch from an event procedure to an expression or macro, depending on the circumstances under which the event was triggered.

The OnMenu value will be one of the following, depending on the selection chosen in the Choose Builder window (accessed by clicking the Build button next to the On Menu box in the form or report's Properties window):

- If Expression Builder is chosen, the value will be "expression", where expression is the expression from the Expression Builder window.
- If Macro Builder is chosen, the value is the name of the macro.
- If Code Builder is chosen, the value will be "[Event Procedure]".

If the On Menu box is blank, the property value is an empty string.
Example

The following example prints the value of the OnMenu property in the Immediate window for the "Purchase Order" report.

Debug.Print Reports("Purchase Order").OnMenu
OnMouseDown Property

Sets or returns the value of the **On Mouse Down** box in the **Properties** window of one of the objects in the Applies To list. Read/write **String**.

_expression_.**OnMouseDown**

.expression Required. An expression that returns one of the objects in the Applies To list.
Remarks

This property is helpful for programmatically changing the action Microsoft Access takes when an event is triggered. For example, between event calls you may want to change an expression's parameters, or switch from an event procedure to an expression or macro, depending on the circumstances under which the event was triggered.

The **MouseDown** event occurs when the user clicks the mouse button while the mouse pointer rests over the object.

The **OnMouseDown** value will be one of the following, depending on the selection chosen in the **Choose Builder** window (accessed by clicking the **Build** button next to the **On Mouse Down** box in the object's **Properties** window):

- If Expression Builder is chosen, the value will be "'=expression"", where *expression* is the expression from the Expression Builder window.
- If Macro Builder is chosen, the value is the name of the macro.
- If Code Builder is chosen, the value will be "[Event Procedure]".

If the **On Mouse Down** box is blank, the property value is an empty string.
Example

The following example prints the value of the **OnMouseDown** property in the Immediate window for the button named "OK" on the "Order Entry" form.

```vbnet
Debug.Print Forms("Order Entry").Controls("OK").OnMouseDown
```
OnMouseMove Property

Sets or returns the value of the On Mouse Move box in the Properties window of one of the objects in the Applies To list. Read/write String.

expression.OnMouseMove

expression  Required. An expression that returns one of the objects in the Applies To list.
Remarks

This property is helpful for programmatically changing the action Microsoft Access takes when an event is triggered. For example, between event calls you may want to change an expression's parameters, or switch from an event procedure to an expression or macro, depending on the circumstances under which the event was triggered.

The **MouseMove** event occurs when the user moves the mouse over the object.

The **OnMouseMove** value will be one of the following, depending on the selection chosen in the **Choose Builder** window (accessed by clicking the **Build** button next to the **On Mouse Move** box in the object's **Properties** window):

- If Expression Builder is chosen, the value will be "=expression", where `expression` is the expression from the Expression Builder window.
- If Macro Builder is chosen, the value is the name of the macro.
- If Code Builder is chosen, the value will be "[Event Procedure]".

If the **On Mouse Move** box is blank, the property value is an empty string.
Example

The following example prints the value of the `OnMouseMove` property in the Immediate window for the button named "OK" on the "Order Entry" form.

Debug.Print Forms("Order Entry").Controls("OK").OnMouseMove
OnMouseUp Property

Sets or returns the value of the **On Mouse Up** box in the **Properties** window of one of the objects in the Applies To list. Read/write **String**.

`expression.OnMouseUp`

`expression` Required. An expression that returns one of the objects in the Applies To list.
Remarks

This property is helpful for programmatically changing the action Microsoft Access takes when an event is triggered. For example, between event calls you may want to change an expression's parameters, or switch from an event procedure to an expression or macro, depending on the circumstances under which the event was triggered.

The **MouseUp** event occurs when the user releases a mouse button.

The **OnMouseUp** value will be one of the following, depending on the selection chosen in the **Choose Builder** window (accessed by clicking the **Build** button next to the **On Mouse Up** box in the object's **Properties** window):

- If Expression Builder is chosen, the value will be "=expression", where *expression* is the expression from the Expression Builder window.
- If Macro Builder is chosen, the value is the name of the macro.
- If Code Builder is chosen, the value will be "[Event Procedure]".

If the **On Mouse Up** box is blank, the property value is an empty string.
Example

The following example prints the value of the **OnMouseUp** property in the Immediate window for the button named "OK" on the "Order Entry" form.

```
Debug.Print Forms("Order Entry").Controls("OK").OnMouseUp
```
**OnNoData Property**

Sets or returns the value of the **On No Data** box in the **Properties** window of a report. Read/write **String**.

-expression OnNoData

*expression* Required. An expression that returns one of the objects in the Applies To list.
Remarks

This property is helpful for programmatically changing the action Microsoft Access takes when an event is triggered. For example, between event calls you may want to change an expression's parameters, or switch from an event procedure to an expression or macro, depending on the circumstances under which the event was triggered.

The **NoData** event occurs after Microsoft Access formats a report for printing that has no data (the report is bound to an empty recordset), but before the report is printed.

The **OnNoData** value will be one of the following, depending on the selection chosen in the **Choose Builder** window (accessed by clicking the **Build** button next to the **On No Data** box in the report's **Properties** window):

- If Expression Builder is chosen, the value will be "=expression", where *expression* is the expression from the Expression Builder window.
- If Macro Builder is chosen, the value is the name of the macro.
- If Code Builder is chosen, the value will be "[Event Procedure]".

If the **On No Data** box is blank, the property value is an empty string.
Example

The following example prints the value of the **OnNoData** property in the Immediate window for the "Purchase Order" report.

```
Debug.Print Reports("Purchase Order").OnNoData
```
OnNotInList Property

Sets or returns the value of the **On Not in List** box in the **Properties** window of a combo box. Read/write **String**.

`expression.OnNotInList`

`expression` Required. An expression that returns one of the objects in the Applies To list.
Remarks

This property is helpful for programmatically changing the action Microsoft Access takes when an event is triggered. For example, between event calls you may want to change an expression's parameters, or switch from an event procedure to an expression or macro, depending on the circumstances under which the event was triggered.

The **NotInList** event occurs when the user enters a value in the text box portion of a combo box that isn't in the combo box list.

The **OnNotInList** value will be one of the following, depending on the selection chosen in the Choose Builder window (accessed by clicking the Build button next to the On Not in List box in the combo box's Properties window):

- If Expression Builder is chosen, the value will be "=expression", where expression is the expression from the Expression Builder window.
- If Macro Builder is chosen, the value is the name of the macro.
- If Code Builder is chosen, the value will be "[Event Procedure]".

If the On Not in List box is blank, the property value is an empty string.
Example

The following example prints the value of the OnNotInList property in the Immediate window for the "State" combo box in the "Order Entry" form.

Debug.Print Forms("Order Entry").Controls("State").OnNotInList
OnOpen Property

Sets or returns the value of the On Open box in the Properties window of a form or report. Read/write String.

expression.OnOpen

expression Required. An expression that returns one of the objects in the Applies To list.
**Remarks**

This property is helpful for programmatically changing the action Microsoft Access takes when an event is triggered. For example, between event calls you may want to change an expression's parameters, or switch from an event procedure to an expression or macro, depending on the circumstances under which the event was triggered.

The **Open** event occurs when a form is opened, but before the first record is displayed. For reports, the event occurs before a report is previewed or printed.

The **OnOpen** value will be one of the following, depending on the selection chosen in the **Choose Builder** window (accessed by clicking the **Build** button next to the **On Open** box in the form or report's **Properties** window):

- If Expression Builder is chosen, the value will be "=expression", where *expression* is the expression from the Expression Builder window.
- If Macro Builder is chosen, the value is the name of the macro.
- If Code Builder is chosen, the value will be "[Event Procedure]".

If the **On Open** box is blank, the property value is an empty string.
Example

The following example prints the value of the `OnOpen` property in the Immediate window for the "Purchase Order" report.

`Debug.Print Reports("Purchase Order").OnOpen`
OnPage Property

Sets or returns the value of the On Page box in the Properties window of a report. Read/write String.

expression.OnPage

expression Required. An expression that returns one of the objects in the Applies To list.
Remarks

This property is helpful for programmatically changing the action Microsoft Access takes when an event is triggered. For example, between event calls you may want to change an expression's parameters, or switch from an event procedure to an expression or macro, depending on the circumstances under which the event was triggered.

The Page event occurs after Microsoft Access formats a page of a report for printing, but before the page is printed.

The OnPage value will be one of the following, depending on the selection chosen in the Choose Builder window (accessed by clicking the Build button next to the On Page box in the report's Properties window):

- If Expression Builder is chosen, the value will be "=expression", where expression is the expression from the Expression Builder window.
- If Macro Builder is chosen, the value is the name of the macro.
- If Code Builder is chosen, the value will be "[Event Procedure]".

If the On Page box is blank, the property value is an empty string.
Example

The following example prints the value of the OnPage property in the Immediate window for the "Purchase Order" report.

Debug.Print Reports("Purchase Order").OnPage
OnPrint Property

Sets or returns the value of the On Print box in the Properties window of a report section. Read/write String.

expression.OnPrint

description Required. An expression that returns one of the objects in the Applies To list.
Remarks

This property is helpful for programmatically changing the action Microsoft Access takes when an event is triggered. For example, between event calls you may want to change an expression's parameters, or switch from an event procedure to an expression or macro, depending on the circumstances under which the event was triggered.

The Print event occurs after data in a report section is formatted for printing, but before the section is printed.

The OnPrint value will be one of the following, depending on the selection chosen in the Choose Builder window (accessed by clicking the Build button next to the On Print box in the report section's Properties window):

- If Expression Builder is chosen, the value will be "=expression", where expression is the expression from the Expression Builder window.
- If Macro Builder is chosen, the value is the name of the macro.
- If Code Builder is chosen, the value will be "[Event Procedure]".

If the On Print box is blank, the property value is an empty string.
Example

The following example prints the value of the **OnPrint** property in the Immediate window for the "GroupHeader0" section in the "Purchase Order" report.

```
Debug.Print Reports("Purchase Order").Section("GroupHeader0").OnPrint
```
OnPush Property

Sets or returns the value of the **On Click** box in the **Properties** window of a command button. Read/write **String**.

`expression.OnPush`

`expression`  Required. An expression that returns one of the objects in the Applies To list.
Remarks

This property is helpful for programmatically changing the action Microsoft Access takes when an event is triggered. For example, between event calls you may want to change an expression's parameters, or switch from an event procedure to an expression or macro, depending on the circumstances under which the event was triggered.

Although the name of this property is OnPush, setting this property actually sets the value of the On Click box.

The OnPush value will be one of the following, depending on the selection chosen in the Choose Builder window (accessed by clicking the Build button next to the On Click box in the command button's Properties window):

- If Expression Builder is chosen, the value will be "=expression", where expression is the expression from the Expression Builder window.
- If Macro Builder is chosen, the value is the name of the macro.
- If Code Builder is chosen, the value will be "[Event Procedure]".

If the On Click box is blank, the property value is an empty string.
Example

The following example prints the value of the **OnPush** property in the Immediate window for the "OK" button on the "Order Entry" form.

Debug.Print Forms("Order Entry").Controls("OK").**OnResize**
OnResize Property

Sets or returns the value of the On Resize box in the Properties window of a form. Read/write String.

expression. OnResize

expression  Required. An expression that returns one of the objects in the Applies To list.
Remarks

This property is helpful for programmatically changing the action Microsoft Access takes when an event is triggered. For example, between event calls you may want to change an expression's parameters, or switch from an event procedure to an expression or macro, depending on the circumstances under which the event was triggered.

The **Resize** event occurs when a form is opened and whenever the size of a form changes.

The **OnResize** value will be one of the following, depending on the selection chosen in the **Choose Builder** window (accessed by clicking the **Build** button next to the **On Resize** box in the form's **Properties** window):

- If Expression Builder is chosen, the value will be "=expression", where *expression* is the expression from the Expression Builder window.
- If Macro Builder is chosen, the value is the name of the macro.
- If Code Builder is chosen, the value will be "[Event Procedure]".

If the **On Resize** box is blank, the property value is an empty string.
Example

The following example prints the value of the **OnResize** property in the Immediate window for the "Order Entry" form.

```vba
Debug.Print Forms("Order Entry").OnResize
```
OnRetreat Property

Sets or returns the value of the On Retreat box in the Properties window of a report section. Read/write String.

expression.OnRetreat

expression  Required. An expression that returns one of the objects in the Applies To list.
Remarks

This property is helpful for programmatically changing the action Microsoft Access takes when an event is triggered. For example, between event calls you may want to change an expression's parameters, or switch from an event procedure to an expression or macro, depending on the circumstances under which the event was triggered.

The **Retreat** event occurs when Microsoft Access returns to a previous report section during report formatting.

The **OnRetreat** value will be one of the following, depending on the selection chosen in the **Choose Builder** window (accessed by clicking the **Build** button next to the **On Retreat** box in the report section's **Properties** window):

- If Expression Builder is chosen, the value will be "=expression", where *expression* is the expression from the Expression Builder window.
- If Macro Builder is chosen, the value is the name of the macro.
- If Code Builder is chosen, the value will be "[Event Procedure]".

If the **On Retreat** box is blank, the property value is an empty string.
Example

The following example prints the value of the **OnRetreat** property in the Immediate window for the "GroupHeader0" section in the "Purchase Order" report.

```vbscript
Debug.Print Reports("Purchase Order").Section("GroupHeader0").OnRetreat
```
OnTimer Property

Sets or returns the value of the On Timer box in the Properties window of a form. Read/write String.

expression.OnTimer

expression  Required. An expression that returns one of the objects in the Applies To list.
Remarks

This property is helpful for programmatically changing the action Microsoft Access takes when an event is triggered. For example, between event calls you may want to change an expression's parameters, or switch from an event procedure to an expression or macro, depending on the circumstances under which the event was triggered.

The **Timer** event occurs for a form at regular intervals as specified by the form's **TimerInterval** property.

The **OnTimer** value will be one of the following, depending on the selection chosen in the **Choose Builder** window (accessed by clicking the **Build** button next to the **On Timer** box in the form's **Properties** window):

- If Expression Builder is chosen, the value will be "=expression", where *expression* is the expression from the Expression Builder window.
- If Macro Builder is chosen, the value is the name of the macro.
- If Code Builder is chosen, the value will be "[Event Procedure]".

If the **On Timer** box is blank, the property value is an empty string.
Example

The following example prints the value of the **OnTimer** property in the Immediate window for the "Order Entry" form.

```vbnet
Debug.Print Forms("Order Entry").OnTimer
```
OnUndo Property

Returns or sets a String indicating which macro, event procedure, or user-defined function runs when the Undo event occurs. Read/write.

expression.OnUndo

expression Required. An expression that returns one of the objects in the Applies To list.
Remarks

Valid values for this property are "macroname" where macroname is the name of macro, "[Event Procedure]" which indicates the event procedure associated with the Undo event for the specified object, or "=functionname()" where functionname is the name of a user-defined function. For a more detailed discussion of event properties, see "Event Properties."
Example

The following example specifies that when the Undo event occurs on the first form of the current project, the associated event procedure should run.

Forms(0).OnUndo = "[Event Procedure]"

The following example specifies that when the Undo event occurs in any text box on the first form of the current project, the associated event procedure should run.

Dim ctlLoop As Control

For Each ctlLoop In Forms(0).Controls
    If ctlLoop.Type = acTextBox Then
        ctlLoop.OnUndo = "[Event Procedure]"
    End If
Next ctlLoop
OnUnload Property

Sets or returns the value of the **On Unload** box in the **Properties** window of a form. Read/write **String**.

`expression.OnUnload`

`expression` Required. An expression that returns one of the objects in the Applies To list.
Remarks

This property is helpful for programmatically changing the action Microsoft Access takes when an event is triggered. For example, between event calls you may want to change an expression's parameters, or switch from an event procedure to an expression or macro, depending on the circumstances under which the event was triggered.

The **Unload** event occurs after a form is closed but before it's removed from the screen.

The **OnUnload** value will be one of the following, depending on the selection chosen in the Choose Builder window (accessed by clicking the Build button next to the On Unload box in the form's Properties window):

- If Expression Builder is chosen, the value will be "=expression", where *expression* is the expression from the Expression Builder window.
- If Macro Builder is chosen, the value is the name of the macro.
- If Code Builder is chosen, the value will be "[Event Procedure]".

If the On Unload box is blank, the property value is an empty string.
Example

The following example prints the value of the **OnUnload** property in the Immediate window for the "Order Entry" form.

Debug.Print Forms("Order Entry").**OnUnload**
OnUpdated Property

Sets or returns the value of the On Updated box in the Properties window of an OLE container control (such as a bound or unbound object frame or custom control). Read/write String.

*expression*.OnUpdated

*expression*  Required. An expression that returns one of the objects in the Applies To list.
Remarks

This property is helpful for programmatically changing the action Microsoft Access takes when an event is triggered. For example, between event calls you may want to change an expression's parameters, or switch from an event procedure to an expression or macro, depending on the circumstances under which the event was triggered.

The **OnUpdated** value will be one of the following, depending on the selection chosen in the **Choose Builder** window (accessed by clicking the **Build** button next to the **On Updated** box in the control's **Properties** window):

- If Expression Builder is chosen, the value will be "=expression", where *expression* is the expression from the Expression Builder window.
- If Macro Builder is chosen, the value is the name of the macro.
- If Code Builder is chosen, the value will be "[Event Procedure]".

If the **On Updated** box is blank, the property value is an empty string.
Example

The following example prints the value of the **OnUpdated** property in the Immediate window for the "Customer Picture" control on the "Order Entry" form.

Debug.Print Forms("Order Entry").Controls("Customer Picture").OnUpdate
OpenArgs Property

Determines the string expression specified by the OpenArgs argument of the OpenForm method that opened a form. Read/write Variant.

expression.OpenArgs

expression  Required. An expression that returns one of the objects in the Applies To list.
Remarks

This property is available only by using a macro or by using Visual Basic with the OpenForm method of the DoCmd object. This property setting is read-only in all views.

To use the OpenArgs property, open a form by using the OpenForm method of the DoCmd object and set the OpenArgs argument to the desired string expression. The OpenArgs property setting can then be used in code for the form, such as in an Open event procedure. You can also refer to the property setting in a macro, such as an Open macro, or an expression, such as an expression that sets the ControlSource property for a control on the form.

For example, suppose that the form you open is a continuous-form list of clients. If you want the focus to move to a specific client record when the form opens, you can set the OpenArgs property to the client's name, and then use the FindRecord action in an Open macro to move the focus to the record for the client with the specified name.
Example

The following example uses the OpenArgs property to open the Employees form to a specific employee record and demonstrates how the OpenForm method sets the OpenArgs property. You can run this procedure as appropriate — for example, when the AfterUpdate event occurs for a custom dialog box used to enter new information about an employee.

```
Sub OpenToCallahan()
    DoCmd.OpenForm "Employees", acNormal, , , acReadOnly, , , "Callahan"
End Sub

Sub Form_Open(Cancel As Integer)
    Dim strEmployeeName As String
    '
    ' If OpenArgs property contains employee name, find
    ' corresponding employee record and display it on form. For
    ' example, if the OpenArgs property contains "Callahan",
    ' move to first "Callahan" record.
    strEmployeeName = Forms!Employees.OpenArgs
    If Len(strEmployeeName) > 0 Then
        DoCmd.GoToControl "LastName"
        DoCmd.FindRecord strEmployeeName, , True, , True, , True
    End If
End Sub
```

The next example uses the FindFirst method to locate the employee named in the OpenArgs property.

```
Private Sub Form_Open(Cancel As Integer)
    If Not IsNull(Me.OpenArgs) Then
        Dim strEmployeeName As String
        strEmployeeName = Me.OpenArgs
        Dim RS As DAO.Recordset
        Set RS = Me.RecordsetClone
        RS.FindFirst "LastName = '" & strEmployeeName & '""
        If Not RS.NoMatch Then
            Me.Bookmark = RS.Bookmark
        End If
    End If
End Sub
```
Operator Property

You can use the **Operator** property to return the operator value for the conditional format or data validation of a **FormatCondition** object. Read-only **AcFormatConditionOperator**.

AcFormatConditionOperator can be one of these AcFormatConditionOperator constants.

- `acBetween`
- `acEqual`
- `acGreaterThan`
- `acGreaterThanOrEqual`
- `acLessThan`
- `acLessThanOrEqual`
- `acNotBetween`
- `acNotEqual`

```plaintext
expression.Operator
```

*expression* Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **Operator** property is available only by using [Visual Basic](https://msdn.microsoft.com/en-us/library/aa270497%28VS.60%29). The **Operator** property's intrinsic constants are used in conjunction with the **Expression1** and **Expression2** properties and the **Add** method of the **FormatConditions** object for conditional formatting and data validation.
OptionValue Property

Each control in an option group has a numeric value that you can set with the OptionValue property. Read/write Long.

expression.OptionValue

expression Required. An expression that returns one of the objects in the Applies To list.
Remarks

When the control is selected, the number is assigned to the option group. If the option group is bound to a field, the value of the selected control's OptionValue property is stored in the field.

For example, this Region option group is bound to the Region field in a table. The Europe button has an OptionValue property setting of 1, the Asia button has a setting of 2, and the Africa button has a setting of 3. When one of these buttons is selected, the Region option group value will be the same as the OptionValue property setting for the selected control. In this case, because the Region option group is bound to the Region field, the value of this field in the table also equals 2.

Note The OptionValue property applies only to the check box, option button, and toggle button controls in an option group.

You can set the OptionValue property by using the control's property sheet, a macro, or Visual Basic.

Unless you change the OptionValue property yourself, the first control you place in an option group has a value of 1, the second control has a value of 2, and so on.

The OptionValue property is only available when the control is placed inside an option group control. When a check box, a toggle button, or an option button isn't in an option group, the control has no OptionValue property. Instead, each such control has a ControlSource property, and the value of each control will be either True if selected or False if not selected.
Example

The following example sets the OptionValue property for three option buttons in the "Ship Method Group" option group when a form opens. When an option button is selected in the option group, a message displays indicating the shipper's assigned ID number.

Private Sub Form_Open(Cancel As Integer)
    Me.Controls("ABC Couriers").OptionValue = 15876
    Me.Controls("Speedy Delivery").OptionValue = 742
    Me.Controls("Lightning Express").OptionValue = 1256
End Sub

Private Sub Ship_Method_Group_Click()
    MsgBox "The ID for the selected shipper is " & Me.Controls("Ship Method Group").Value
End Sub
OrderBy Property

You can use the **OrderBy** property to specify how you want to sort records in a form, query, report, or table. Read/write **String**.

*expression*.OrderBy

*expression* Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **OrderBy** property is a *string expression* that is the name of the field or fields on which you want to sort records. When you use more than one field name, separate the names with a comma (,). Use the **OrderBy** property to save an ordering value and apply it at a later time. **OrderBy** values are saved with the objects in which they are created. They are automatically loaded when the object is opened, but they aren't automatically applied.

When you set the **OrderBy** property by entering one or more field names, the records are sorted in ascending order. Similarly, Visual Basic sorts these fields in ascending order by default.

If you want to sort records in descending order, type **DESC** at the end of the string expression. For example, to sort customer records in descending order by contact name, set the **OrderBy** property to "ContactName DESC".

You can set the **OrderBy** property by using the object's *property sheet*, a *macro*, or *Visual Basic*.

For reports, the **OrderByOn** property must be set to Yes to apply the sort order specified by the object's **OrderBy** property. For forms, select the field by which you want to sort the records and either click the appropriate Sort button on the *toolbar*, or point to *Sort* on the *Records* menu and click the appropriate command on the submenu. You can also set the **OrderByOn** property for either forms or reports by using Visual Basic.

Setting the **OrderBy** property for an open report will run the report's Close and Open event procedures.

**Note**  When a new object is created, it inherits the **RecordSource**, **Filter**, **OrderBy**, and **OrderByOn** properties of the table or query it was created from. For forms and reports, inherited filters aren't automatically applied when an object is opened.
OrderByOn Property

You can use the **OrderByOn** property to specify whether an object's **OrderBy** property setting is applied. Read/write **Boolean**.

*expression*.OrderByOn

*expression*  Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **OrderByOn** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>True</td>
<td>The <strong>OrderBy</strong> property setting is applied when the object is opened.</td>
</tr>
<tr>
<td>No</td>
<td>False</td>
<td>(Default) The <strong>OrderBy</strong> property setting isn't applied when the object is opened.</td>
</tr>
</tbody>
</table>

**Note** For reports, you can set the **OrderByOn** property by using the report's [property sheet](https://example.com), a [macro](https://example.com), or [Visual Basic](https://example.com).

For all other objects, you can set the **OrderByOn** property by clicking a Sort button on the [toolbar](https://example.com) or by using Visual Basic.

When a new object is created, it inherits the **RecordSource**, **Filter**, **OrderBy**, **OrderByOn**, and **FilterOn** properties of the table or query it was created from.
Example

The following example displays a message indicating the state of the **OrderByOn** property for the "Mailing List" form.

MsgBox "OrderByOn property is " & Forms("Mailing List").**OrderByOn**
**OrganizeInFolder Property**

You can use the **OrganizeInFolder** property to specify or determine if all supporting files, such as image files, are stored in their own folder or with the data access page. Read/write **Boolean**.

\[expression . OrganizeInFolder\]

expression  Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **OrganizeInFolder** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td><strong>True</strong></td>
<td>(Default) The supporting files are organized in a folder when you save the document as a data access page.</td>
</tr>
<tr>
<td>No</td>
<td><strong>False</strong></td>
<td>The supporting files are saved in the same folder as the data access page.</td>
</tr>
</tbody>
</table>

**Note** The **OrganizeInFolder** property is available only by using **Visual Basic**.

The new folder is created in the folder where you have saved the Web page, and is named after the document. If long file names are used, a suffix is added to the folder name. The **FolderSuffix** property returns the folder suffix for the language support you have selected or installed, or the default folder suffix.

If you save a document that was previously saved with the **OrganizeInFolder** property set to a different value, Microsoft Access automatically moves the supporting files into or out of the folder, as appropriate.

If you don't use long file names (that is, if the **UseLongFileNames** property is set to **False**), Microsoft Access automatically saves any supporting files in a separate folder. The files cannot be saved in the same folder as the Web page.
Example

This example specifies that all image files are saved in the same folder when the document is saved as a Web page.

Application.DefaultWebOptions.OrganizeInFolder = False
Orientation Property

Orientation property as it applies to the Printer object.

You can use the Orientation property to specify or determine the print orientation. Read/write AcPrintOrientation.

AcPrintOrientation can be one of these AcPrintOrientation constants.
- acPRORLandscape
- acPRORPortrait

expression.Orientation

expression Required. An expression that returns one of the above objects.

Orientation property as it applies to the Form and Report objects.

You can use the Orientation property to specify or determine the view orientation. Read/write Byte.

expression.Orientation

expression Required. An expression that returns one of the above objects.
Remarks

The **Orientation** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left-to-Right</td>
<td>0</td>
<td>Sets the view orientation to left to right.</td>
</tr>
<tr>
<td>Right-to-Left</td>
<td>1</td>
<td>Sets the view orientation to right to left.</td>
</tr>
</tbody>
</table>

You can set this property by using the [property sheet](#) or [Visual Basic](#).
Example

As it applies to the Printer object.

The following example sets the print orientation to landscape.

Printer.Orientation = acPROLandscape

As it applies to the Form and Report objects.

The following example sets the view orientation to right-to-left for the "Purchase Order" report.

Reports("Purchase Order").Orientation = 1
OutOfDateObjects Property

Returns a `DependencyObjects` collection that represents the `AccessObject` objects for which the dependency information is outdated.

`expression.OutOfDateObjects`

`expression` Required. An expression that returns a `DependencyInfo` object.
Remarks

You can use the following code to update the dependency information for all of the objects in the database:

Application.CurrentProject.UpdateDependencyInfo
Page Property

The **Page** property specifies the current page number when a form or report is being printed. Read/write **Long**.

`expression.Page`

*expression*  
Required. An expression that returns one of the objects in the Applies To list.
Remarks

Although you can set the **Page** property to a value, you most often use this property to return information about page numbers.

You can use the **Page** property in an expression, a **macro**, or **Visual Basic**.

This property is only available in **Print Preview** or when printing.
Example

The following example updates the report's caption to display the current position in the report as the user pages back and forth in the report.

Private Sub Report_Page()
Me.Caption = "Now Viewing Page " & Me.Page & " Of " & Me.Pages & " Page(s)"
End Sub
PageFooter Property

You can use the PageFooter property to specify whether a report's page footer is printed on the same page as a report footer. Read/write Byte.

expression.PageFooter

expression Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **PageHeader** and **PageFooter** properties use the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>All Pages</td>
<td>0</td>
<td>(Default) The page footer is printed on all pages of a report.</td>
</tr>
<tr>
<td>Not With Rpt Hdr</td>
<td>1</td>
<td>The page footer isn't printed on the same page as the report header.</td>
</tr>
<tr>
<td>Not With Rpt Ftr</td>
<td>2</td>
<td>The page footer isn't printed on the same page as the report footer. Microsoft Access prints the report footer on a new page.</td>
</tr>
<tr>
<td>Not With Rpt Hdr/Ftr</td>
<td>3</td>
<td>The page footer isn't printed on a page that has either a report header or a report footer. Microsoft Access prints the report footer on a new page.</td>
</tr>
</tbody>
</table>

**Note** You can set these properties by using the report's [property sheet](https://www.example.com), a [macro](https://www.example.com) or [Visual Basic](https://www.example.com).

You can set the **PageFooter** property only in [report Design view](https://www.example.com).

Microsoft Access normally prints report page footers on every page in a report, including the first and last.

In report Design view, click **Page Header/Footer** on the **View** menu to display the page footer sections.

**Note** When forms are printed, page footers are always printed on all pages.
Example

The following example sets the **PageFooter** property for a report to Not With Rpt Hdr. To run this example, type the following line in the Debug window for a report named Report1.

```apl
Reports!Report1.PageFooter = 1
```
PageHeader Property

You can use the **PageHeader** property to specify whether a report's page header is printed on the same page as a report header. Read/write **Byte**.

*expression*. **PageHeader**

*expression* Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **PageHeader** property use the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>All Pages</td>
<td>0</td>
<td>(Default) The page header is printed on all pages of a report.</td>
</tr>
<tr>
<td>Not With Rpt Hdr</td>
<td>1</td>
<td>The page header isn't printed on the same page as the report header.</td>
</tr>
<tr>
<td>Not With Rpt Ftr</td>
<td>2</td>
<td>The page header isn't printed on the same page as the report footer.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Microsoft Access prints the report footer on a new page.</td>
</tr>
<tr>
<td>Not With Rpt Hdr/Ftr</td>
<td>3</td>
<td>The page header isn't printed on a page that has either a report header or a report footer. Microsoft Access prints the report footer on a new page.</td>
</tr>
</tbody>
</table>

**Note**  You can set these properties by using the report's property sheet, a macro, or Visual Basic.

You can set the **PageHeader** property only in report **Design view**.

Microsoft Access normally prints report page headers on every page in a report, including the first and last.

In report Design view, click **Page Header/Footer** on the **View** menu to display the page header and page footer sections.

**Note**  When forms are printed, page headers are always printed on all pages.
Example

The following example sets the `PageHeader` property for a report to Not With Rpt Hdr. To run this example, type the following line in the Debug window for a report named Report1.

```vbnet
Reports!Report1.PageHeader = 1
```
PageIndex Property

You can use the `PageIndex` property to specify or determine the position of a `Page` object within a `Pages` collection. The `PageIndex` property specifies the order in which the pages on a `tab control` appear. Read/write `Integer`.

`expression.PageIndex`

`expression` Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **PageIndex** property setting is an **Integer** value between 0 and the **Pages** collection **Count** property setting minus 1.

You can set the **PageIndex** property by using a **Page** object's **property sheet**, a **macro**, or **Visual Basic**.

The **PageIndex** property can be set in any view.

Changing the value of the **PageIndex** property changes the location of a **Page** object in the **Pages** collection and visually changes the order of pages on a tab control.

You can also change the order of **Page** objects in the **Pages** collection by using the **Page Order** dialog box available by right-clicking the tab control in **Design view** and then clicking **Page Order**.
Example

The following example moves the page named "Notes" on the tab control named "Information" on the "Order Entry" form to the first page.

Forms("Order Entry").Controls("Information").Pages("Notes").PageIndex
Pages Property

Pages property as it applies to the Form and Report objects.

You can use the Pages property to return information needed to print page numbers in a form or report. Read/write Integer.

expression.Pages

expression Required. An expression that returns one of the above objects.
Remarks

You can use the **Pages** properties in an expression, a macro, or **Visual Basic**.

This property is only available in **Print Preview** or when printing.

To refer to the **Pages** property in a macro or Visual Basic, the form or report must include a text box whose **ControlSource** property is set to an expression that uses **Pages**. For example, you can use the following expressions as the **ControlSource** property setting for a text box in a page footer.

<table>
<thead>
<tr>
<th>This expression</th>
<th>Prints</th>
</tr>
</thead>
<tbody>
<tr>
<td>=Page</td>
<td>A page number (for example, 1, 2, 3) in the page footer.</td>
</tr>
<tr>
<td>=&quot;Page &quot; &amp; Page &amp; &quot; of &quot; &amp; &quot;Page n of nn&quot;</td>
<td>The total number pages in the form or report (for example, 5).</td>
</tr>
<tr>
<td>&amp; Pages</td>
<td></td>
</tr>
<tr>
<td>=Pages</td>
<td></td>
</tr>
</tbody>
</table>

Returns the number of pages in a control that supports tabbed pages (for example, a **TabControl** object). Read-only.

**expression**.Pages

**expression** Required. An expression that returns one of the above objects.
Example

As it applies to the **Form** and **Report** objects.

The following example displays a message that tells how many pages the report contains. For this example to work, the report must include a text box for which the **ControlSource** property is set to the expression =Pages. To test this example, paste the following code into the Page Event for the Alphabetical List of Products form.

```vba
Dim intTotalPages As Integer
Dim strMsg As String

intTotalPages = Me.Pages
strMsg = "This report contains " & intTotalPages & " pages."
MsgBox strMsg
```

As it applies to the **Control** and **TabControl** objects.

The following example displays a message indicating the number of tabbed pages on tab control TabCtl1.

```vba
MsgBox "Number of pages in TabCtl1:" & TabCtl1.Pages.Count
```
Painting Property

You can use the **Painting** property to specify whether forms or reports are repainted. Read/write **Boolean**.

*expression.Painting*

*expression* Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **Painting** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>(Default) The form or report is repainted.</td>
</tr>
<tr>
<td>False</td>
<td>The form or report isn't repainted.</td>
</tr>
</tbody>
</table>

**Note** You can set this property by using a [macro](https://www.example.com) or [Visual Basic](https://www.example.com).

This property can be set and applies only in [Form view](https://www.example.com) and is unavailable in other views.

The **Painting** property is similar to the **Echo** action. However, the **Painting** property prevents repainting of a single form or report, whereas the Echo action prevents repainting of all open windows in an application.

Setting the **Painting** property for a form or report to **False** also prevents all [controls](https://www.example.com) (except [subform](https://www.example.com) or [subreport](https://www.example.com) controls) on a form or report from being repainted. To prevent a subform or subreport control from being repainted, you must set the **Painting** property for the subform or subreport to **False**. (Note that you set the **Painting** property for the subform or subreport, not the subform or subreport control.)

The **Painting** property is automatically set to **True** whenever the form or report gets or loses the [focus](https://www.example.com). You can set this property to **False** while you are working on a form or report if you don't want to see changes to the form or report or to its controls. For example, if a form has a set of controls that are automatically resized when the form is resized and you don't want the user to see each individual control move, you can turn **Painting** off, and then move all of the controls, then turn **Painting** back on.
Example

The following example uses the **Painting** property to enable or disable form painting depending on whether the SetPainting variable is set to **True** or **False**. If form painting is turned off, Microsoft Access displays the hourglass icon while painting is turned off.

```vba
Public Sub EnablePaint(ByRef frmName As Form, _
                      ByVal SetPainting As Integer)
    frmName.Painting = SetPainting

    ' Form painting is turned off.
    If SetPainting = False Then
        DoCmd.Hourglass True
    Else
        DoCmd.Hourglass False
    End If

End Sub
```
**PaintPalette Property**

You can use the **PaintPalette** property to specify a palette to be used by a **form** or **report**. Read/write **Variant**.

*expression*.**PaintPalette**

*expression*  Required. An expression that returns one of the objects in the Applies To list.
Remarks

You can set the **PaintPalette** property by using a [macro](https://www.example.com) or [Visual Basic](https://www.example.com). The property setting must be a [String](https://www.example.com) data type containing the palette information.

You can set the **PaintPalette** property by assigning the value of the **ObjectPalette** property to the **PaintPalette** property in a macro or Visual Basic, by setting the **PaletteSource** property (in which case Microsoft Access automatically sets the **PaintPalette** property to this **PaletteSource**), or by setting the **PaintPalette** property of one form or report to the **PaintPalette** property of another form or report.

For a form, you can set the **PaintPalette** property in form [Design view](https://www.example.com) and [Form view](https://www.example.com).

For a report, you can set the **PaintPalette** property in report [Design view](https://www.example.com) only.

When you set the **PaintPalette** property, Microsoft Access makes a copy of the palette that you specify and saves it with the form or report. The palette is then available if you modify that form or report.

Changes to the palette you specified when you set the **PaintPalette** property don't affect the copy of the palette stored with the form or report. If you want to update the copy of the palette stored with the form or report, you must rerun the code or macro that sets the **PaintPalette** property or reset the **PaletteSource** property when the form or report is open.

When you set the **PaintPalette** property for a form or report, Microsoft Access automatically updates its **PaletteSource** property. Conversely, when you set the **PaletteSource** property for a form or report, the **PaintPalette** property is also updated. For example, when you specify a custom palette with the **PaintPalette** property, the **PaletteSource** property setting is changed to (Custom). The **PaintPalette** property (which is available only in a macro or Visual Basic) is used to set the palette for the form or report. The **PaletteSource** property gives you a way to set the palette for the form or report in the property sheet by using an existing graphics file.

**Note** Windows can have only one color palette active at a time. Microsoft
Access allows you to have multiple graphics on a form, each using a different color palette. The **PaintPalette** and **PaletteSource** properties let you specify which color palette a form should use when displaying graphics.

You can use the **ObjectPalette** property to make the palette of an application associated with an [OLE object](/en-US/office/previousversions/access/ole-object), [bitmap](/en-US/office/previousversions/access/OLEobject-bitmap), or other graphic contained in a control on a form or report available to the **PaintPalette** property. For example, to make the palette used in Microsoft Graph available when you're designing a form in Microsoft Access, you set the form's **PaintPalette** property to the **ObjectPalette** value of an existing chart control.
Example

The **ObjectPalette** and **PaintPalette** properties are useful for programmatically altering the color palette in use by an open form at run time. A common use of these properties is to set the current form's **PaintPalette** property to the palette of a graphic displayed in a control that has the focus.

For example, you can have a form with an ocean picture, showing many shades of blue, and a sunset picture, showing many shades of red. Since Windows only allows one color palette active at a time, one picture will look much better than the other. The following example uses a control's Enter event for setting the form's **PaintPalette** property to the control's **ObjectPalette** property so the graphic that has the focus will have an optimal appearance.

```vba
Sub OceanPicture_Enter()
    Me.PaintPalette = Me!OceanPicture.ObjectPalette
End Sub

Sub SunsetPicture_Enter()
    Me.PaintPalette = Me!SunsetPicture.ObjectPalette
End Sub
```
PaletteSource Property

You can use the **PaletteSource** property to specify the palette for a [form](#) or [report](#). Read/write **String**.

\[expression.PaletteSource\]

*expression* Required. An expression that returns one of the objects in the Applies To list.
Remarks

Enter the path and file name of one of the following file types:

- .dib (device-independent bitmap file)
- .pal (Windows palette file)
- .ico (Windows icon file)
- .bmp (Windows bitmap file)
- .wmf or .emf file, or other graphics file for which you have a graphics filter

The default setting is (Default), which specifies the palette included with Microsoft Access. If you change this setting by entering a path and file name, the property setting displays (Custom).

You can set this property by using the form's or report's property sheet, a macro, or Visual Basic.

For a form, you can set the PaletteSource property in form Design view and Form view. The property setting is unavailable in other views.

For a report, you can set the PaletteSource property only in report Design view. The property setting is unavailable in other views.

Windows can have only one color palette active at a time. Microsoft Access allows you to have multiple graphics on a form, each using a different color palette. The PaletteSource and PaintPalette properties let you specify which color palette a form uses when displaying graphics.

When you set the PaletteSource property for a form or report, Microsoft Access automatically updates its PaintPalette property. Conversely, when you set a form's or report's PaintPalette property, the PaletteSource property is also updated. For example, when you specify a custom palette with the PaintPalette property, the PaletteSource property setting changes to (Custom). The PaintPalette property (which is available only in a macro or Visual Basic) is
used to set the palette for the form or report. The **PaletteSource** property gives you a way to set the palette for the form or report in the property sheet by using an existing graphics file.
Example

The following example sets the **PaintPalette** property of the Seascape form to the **ObjectPalette** property of the Ocean control on the DisplayPictures form. (Ocean can be a bound object frame, command button, chart, toggle button, or unbound object frame.)

```
Forms!Seascape.PaintPalette = _
    Forms!DisplayPictures!Ocean.ObjectPalette
```

The **ObjectPalette** and **PaintPalette** properties are useful for programmatically altering the color palette in use by an open form at run time. A common use of these properties is to set the current form's **PaintPalette** property to the palette of a graphic displayed in a control that has the focus.

For example, you can have a form with an ocean picture, showing many shades of blue, and a sunset picture, showing many shades of red. Since Windows only allows one color palette active at a time, one picture will look much better than the other. The following example uses a control's Enter event for setting the form's **PaintPalette** property to the control's **ObjectPalette** property so the graphic that has the focus will have an optimal appearance.

```
Sub OceanPicture_Enter()
    Me.PaintPalette = Me!OceanPicture.ObjectPalette
End Sub

Sub SunsetPicture_Enter()
    Me.PaintPalette = Me!SunsetPicture.ObjectPalette
End Sub
```
PaperBin Property

Returns or sets an AcPrintPaperBin constant indicating which paper bin the specified printer should use. Read/write.

AcPrintPaperBin can be one of these AcPrintPaperBin constants.
- acPRBNAuto
- acPRBNCassette
- acPRBNEnvelope
- acPRBNEnvManual
- acPRBNFormSource
- acPRBNLargeCapacity
- acPRBNLargeFmt
- acPRBNLower
- acPRBNManual
- acPRBNMiddle
- acPRBNSmallFmt
- acPRBNTractor
- acPRBNUpper

expression.PaperBin

expression Required. An expression that returns one of the objects in the Applies To list.
Example

The following example sets a variety of printer settings for the form specified in the `strFormname` argument of the procedure.

```vba
Sub SetPrinter(strFormname As String)
    DoCmd.OpenForm FormName:=strFormname, view:=acDesign, 
                    datamode:=acFormEdit, windowmode:=acHidden

    With Forms(form1).Printer
        .TopMargin = 1440
        .BottomMargin = 1440
        .LeftMargin = 1440
        .RightMargin = 1440

        .ColumnSpacing = 360
        .RowSpacing = 360

        .ColorMode = acPRCMColor
        .DataOnly = False
        .DefaultSize = False
        .ItemSizeHeight = 2880
        .ItemSizeWidth = 2880
        .ItemLayout = acPRVerticalColumnLayout
        .ItemsAcross = 6

        .Copies = 1
        .Orientation = acPRORLandscape
        .Duplex = acPRDPVertical
        .PaperBin = acPRBNAuto
        .PaperSize = acPRPSLetter
        .PrintQuality = acPRPQMedium

    End With

    DoCmd.Close objecttype:=acForm, objectname:=strFormname, _
                      Save:=acSaveYes
End Sub
```
PaperSize Property

Returns or sets an AcPrintPaperSize constant indicating the paper size to use when printing. Read/write.

AcPrintPaperSize can be one of these AcPrintPaperSize constants.
acPRPS10x14
acPRPS11x17
acPRPSA3
acPRPSA4
acPRPSA4Small
acPRPSA5
acPRPSB4
acPRPSB5
acPRPSCSheet
acPRPSDSheet
acPRPSEnv10
acPRPSEnv11
acPRPSEnv12
acPRPSEnv14
acPRPSEnv9
acPRPSEnvB4
acPRPSEnvB5
acPRPSEnvB6
acPRPSEnvC3
acPRPSEnvC4
acPRPSEnvC5
acPRPSEnvC6
acPRPSEnvC65
acPRPSEnvDL
acPRPSEnvItaly
acPRPSEnvMonarch
expression.PaperSize

expression  Required. An expression that returns one of the objects in the Applies To list.
**Example**

The following example sets a variety of printer settings for the form specified in the `strFormname` argument of the procedure.

```vba
Sub SetPrinter(strFormname As String)
    DoCmd.OpenForm FormName:=strFormname, view:=acDesign, __
        datamode:=acFormEdit, windowmode:=acHidden
    With Forms(form1).Printer
        .TopMargin = 1440
        .BottomMargin = 1440
        .LeftMargin = 1440
        .RightMargin = 1440
        .ColumnSpacing = 360
        .RowSpacing = 360
        .ColorMode = acPRCMColor
        .DataOnly = False
        .DefaultSize = False
        .ItemSizeHeight = 2880
        .ItemSizeWidth = 2880
        .ItemLayout = acPRVerticalColumnLayout
        .ItemsAcross = 6
        .Copies = 1
        .Orientation = acPRORLandscape
        .Duplex = acPRDPVertical
        .PaperBin = acPRBNAuto
        .**PaperSize** = acPRPSLetter
        .PrintQuality = acPRPQMedium
    End With
    DoCmd.Close objecttype:=acForm, objectname:=strFormname, __
        Save:=acSaveYes
End Sub
```
Parent Property

You can use the Parent property to refer to the parent of a control, section, or control that contains other controls. The Parent property returns a control object if the parent is a control; it returns an AccessObject object if the parent is an Microsoft Access object. Read-only.

expression.Parent

expression  Required. An expression that returns one of the objects in the Applies To list.
Remarks

You can use the **Parent** property to determine which form or report is currently the parent when you have a **subform** or **subreport** that has been inserted in multiple forms or reports.

For example, you might insert an OrderDetails subform into both a form and a report. The following example uses the **Parent** property to refer to the OrderID field, which is present on the **main form** and report. You can enter this expression in a **bound control** on the subform.

\[ \text{=Parent!OrderID} \]

The **Parent** property of a label control is the control the label is linked to. The **Parent** property for a check box, option button, or toggle button in an option group is the name of the option group control. The **Parent** property of an option group control is the name of the form.
Example

The following example uses the **Parent** property to examine the parent of the Speedy Label label control, the Speedy check box control, and the ShipVia option group. To run this example, open the Orders form in the Northwind sample database then run this code.

```vba
Public Sub ShowParent()
    Dim frm As Form
    Dim ctl As Control

    Set frm = Forms!Orders
    Set ctl = frm.[Speedy Label]

    ' Returns name of control attached to label.
    MsgBox "The parent control is " & ctl.Parent.Name
    Set ctl = frm.Speedy

    ' Returns name of control containing control.
    MsgBox "The parent control is " & ctl.Parent.Name
    Set ctl = frm.ShipVia

    ' Returns name of form containing option group control.
    MsgBox "The parent control is " & ctl.Parent.Name

End Sub
```

The next example also returns the name of the form containing the option group control.

```vba
MsgBox Forms!Orders![Speedy Label].Parent.Parent.Parent.Name
```
Path Property

You can use the **Path** property to determine the location where data is stored for a [Microsoft Access project](#) (.adp) or [Microsoft Access database](#) (.mdb). Read-only **String**.

*expression*.Path

*expression* Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **Path** property is a string expression that is the pathname to the disk location where data is stored for an Access database.

This property is available only by using **Visual Basic**.

You can use the **Path** property to determine the location of information stored through the `CurrentProject` or `CodeProject` objects of a project or database.
Example

The following example displays a message indicating the disk location of the current Access project or database.

MsgBox "The current database is located at " & Application.CurrentPr
Picture Property

You can use the **Picture** property to specify a **bitmap** or other type of graphic to be displayed on a **command button**, **image control**, **toggle button**, page on a **tab control** or as a background picture on a **form** or **report**. Read/write **String**.

*expression*.Picture

*expression*  Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **Picture** property contains (bitmap) or the path and file name of a bitmap or other type of graphic to be displayed.

You can set this property by using:

- The **property sheet**. Click the **Build** button to the right of the property box (for command buttons and toggle buttons). When you select one of the bitmap files from the **Available Pictures** list, the property setting is (bitmap).

- A **macro**.

- **Visual Basic**. You can use a **string expression** that includes the path and the name of the graphic, as in the following example:

  ```vba
  btnShowLogo.Picture = "C:\Windows\Winlogo.bmp"
  ```

- The **Picture** command on the **Insert** menu (for image controls or background pictures on forms and reports) to select a bitmap or other type of graphic.

The default setting is (none). After the graphic is loaded into the object, the property setting is (bitmap) or the path and file name of the graphic. If you delete (bitmap) or the path and file name of the graphic from the property setting, the picture is deleted from the object and the property setting is again (none).

If the **PictureType** property is set to Embedded, the graphic is stored with the object.

You can create custom bitmaps by using Microsoft Paintbrush or another application that creates bitmap files. A bitmap file must have a .bmp, .ico, or .dib extension. You can also use graphics files in the .wmf or .emf formats, or any other graphic file type for which you have a graphics filter. Forms, reports, and image controls support all graphics. Command buttons and toggle buttons only support bitmaps.
Buttons can display either a caption or a picture. If you assign both to a button, the picture will be visible, the caption won't. If the picture is deleted, the caption reappears. Microsoft Access displays the picture centered on the button and clipped if the picture is larger than the button.

Tip

To create a command button or toggle button with a caption and a picture, you could include the desired caption as part of the bitmap and assign the bitmap to the **Picture** property of the control.
Example

The following example sets the background picture "Logo.gif" for the "Purchase Order" report.

Reports("Purchase Order").**Picture** = "C:\Picture Files\Logo.gif"
Show All
PictureAlignment Property

You can use the PictureAlignment property to specify where a background picture will appear in an image control or on a form or report. Read/write Byte.

expression. PictureAlignment

expression Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **PictureAlignment** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Top Left</td>
<td>0</td>
<td>The picture is displayed in the top-left corner of the image control, Form window, or page of a report.</td>
</tr>
<tr>
<td>Top Right</td>
<td>1</td>
<td>The picture is displayed in the top-right corner of the image control, Form window, or page of a report.</td>
</tr>
<tr>
<td>Center</td>
<td>2</td>
<td>(Default) The picture is centered in the image control, Form window, or page of a report.</td>
</tr>
<tr>
<td>Bottom Left</td>
<td>3</td>
<td>The picture is displayed in the bottom-left corner of the image control, Form window, or page of a report.</td>
</tr>
<tr>
<td>Bottom Right</td>
<td>4</td>
<td>The picture is displayed in the bottom-right corner of the image control, Form window, or page of a report.</td>
</tr>
<tr>
<td>Form Center</td>
<td>5</td>
<td>(Forms only) The form's picture is centered horizontally in relation to the width of the form and vertically in relation to the height the entire form.</td>
</tr>
</tbody>
</table>

**Note** You can set the **PictureAlignment** property by using a form's or report's property sheet, a macro, or Visual Basic.

You can also set the default for this property by using a control's default control style or the **DefaultControl** method in Visual Basic.

This property can be set in any view.

The Form Center setting aligns a form's picture in the center of the form itself. All other **PictureAlignment** property settings align a form's picture in relation to the Form window. If you want to make sure that a form's picture is displayed only on the form or tiled across only the form, set the **PictureAlignment** property to Form Center.

For reports, the picture appears relative to a full page and not in relation to the size of the actual report. If your report is less than a full page and you want a picture to appear at a location not available through the **PictureAlignment**
property settings, use an image control instead.

When you set the **PictureTiling** property to Yes, tiling of the picture will begin from the **PictureAlignment** property setting.
Example

The following example displays the picture "Logo.gif" in the top left corner of the "Purchase Order" report.

With Reports("Purchase Order")
    .Picture = "C:\Picture Files\Logo.gif"
    .PictureAlignment = 0
End With
PictureData Property

You can use the **PictureData** property to copy the picture in a form, report, or control to another object that supports the **Picture** property. Read/write **Variant**.

*expression*.**PictureData**

*expression* Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **PictureData** property setting is the **PictureData** property of another image control, command button, toggle button, form, or report.

You set this property by using **Visual Basic**.

You can use this property to display different background pictures in a form, depending on actions taken by the user. For example, you might open a Customers form using a different background picture depending on whether the form is opened for data entry or for browsing.

You can also use the **PictureData** property together with the **Timer** event and the **TimerInterval** property to perform simple animation on a form.
Example

The following example uses three image controls to animate a butterfly image across a form. The Hidden1 image control contains a picture of a butterfly with its wings up and the Hidden2 image control contains a picture of the same butterfly with its wings down. Both image controls have their Visible property set to False. The TimerInterval property is set to 200. Each time the Timer event occurs, the picture in the image control Visible1 is changed by using the PictureData property of the hidden image controls, and the visible image control is moved 200 twips to the right. The visible image control is moved back to the left side of the form when its Left property value is greater than the width of the form stored in the public variable gfrmWidth. The value of gfrmWidth is set to Me.Width in the form's open event.

Private Sub Form_Timer()
    Static intPic As Integer
    Select Case intPic
        Case Is = 1
            Me!Visible1.PictureData = Me!Hidden1.PictureData
        Case Is = 2
            Me!Visible1.PictureData = Me!Hidden2.PictureData
        Case Else
    End Select
    If intPic = 2 Then intPic = 0
    intPic = intPic + 1
    If (Me!Visible1.Left > gfrmWidth) Then Me!Visible1.Left = 0
    Me!Visible1.Left = Me!Visible1.Left + 200
End Sub
PicturePages Property

You can use the PicturePages property to specify on which page or pages of a report a picture will be displayed. Read/write Byte.

expression.PicturePages

expression  Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **PicturePages** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>All Pages</td>
<td>0</td>
<td>(Default) The picture appears on all pages of the report.</td>
</tr>
<tr>
<td>First Page</td>
<td>1</td>
<td>The picture appears only on the first page of the report.</td>
</tr>
<tr>
<td>No Pages</td>
<td>2</td>
<td>The picture doesn't appear on the report.</td>
</tr>
</tbody>
</table>

You can set the **PicturePages** property by using a report's [property sheet](#), a [macro](#), or [Visual Basic](#).
Example

The following example prints a stretched version of the picture "Logo.gif" on only the first page of the "Purchase Order" report.

With Reports("Purchase Order")
    .Picture = "C:\Picture Files\Logo.gif"
    .PictureSizeMode = 1
    .PicturePages = 1
End With
PictureSizeMode Property

You can use the **PictureSizeMode** property to specify how a picture for a form or report is sized. Read/write.

*expression*.**PictureSizeMode**

*expression*  Required. An expression that returns one of the objects in the Applies To list.
## Remarks

The **PictureSizeMode** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clip</td>
<td>0 (Default)</td>
<td>The picture is displayed in its actual size. If the picture is larger than the form or report, then the picture is clipped. The picture is stretched horizontally and vertically to fill the entire form, even if its original ratio of height to width is distorted.</td>
</tr>
<tr>
<td>Stretch</td>
<td>1</td>
<td>The picture is enlarged to the maximum extent possible while keeping its original ratio of height to width.</td>
</tr>
</tbody>
</table>

**Note** You can set the **PictureSizeMode** property by using a form's or report's property sheet, a macro, or Visual Basic.

When a small picture is used for the **Picture** property of a form or report, setting the **PictureSizeMode** property to Stretch or Zoom can cause substantial distortion of its resolution. Smaller pictures can be tiled across the entire form or report by using the **PictureTiling** property.
**Example**

The following example sets the background picture of the "Order Entry" form to "Contacts.gif", and stretches the picture to fit the entire form's background.

```
With Forms("Order Entry")
    .Picture = "C:\Picture Files\Contacts.gif"
    .SizeMode = 1
End With
```
**PictureTiling Property**

You can use the `PictureTiling` property to specify whether a background picture is tiled across the entire image control, Form window, form, or page of a report. Read/write **Boolean**.

`expression.PictureTiling`

`expression`  Required. An expression that returns one of the objects in the Applies To list.
**Remarks**

The `PictureTiling` property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>True</td>
<td>The picture is tiled.</td>
</tr>
<tr>
<td>No</td>
<td>False</td>
<td>(Default) The picture isn't tiled.</td>
</tr>
</tbody>
</table>

**Note** You can set the `PictureTiling` property by using the object's [property sheet](#), a [macro](#) or [Visual Basic](#).

You can also set the default for this property by using a control's [default control style](#) or the [DefaultControl](#) method in Visual Basic.

You can create interesting effects by placing a picture on a form or report and setting the `PictureTiling` property to Yes. The alignment of the tiled images is affected by the `PictureAlignment` property setting. For example, if the `PictureAlignment` property is set to Top Left, tiling begins at the top left of the image control, Form window, or page of a report.

**Note** If the `PictureAlignment` property is set to Form Center and the `PictureTiling` property is set to Yes, the background picture of a form will be tiled across the form, not across the Form window.
Example

The following example tiles the picture in the "CustomerPhoto" image control on the "Order Entry" form.

Forms("Order Entry").Controls("CustomerPhoto").PictureTiling = True
PictureType Property

You can use the **PictureType** property to specify whether Microsoft Access stores an object's picture as a [linked](#) or an [embedded](#) object. Read/write.

**expression**.**PictureType**

**expression** Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **PictureType** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Embedded</td>
<td>0</td>
<td>(Default) The picture is embedded in the object and becomes part of the database file.</td>
</tr>
<tr>
<td>Linked</td>
<td>1</td>
<td>The picture is linked to the object. Microsoft Access stores a pointer to the location of the picture on the disk.</td>
</tr>
</tbody>
</table>

**Note** You can set the **PictureType** property by using the object's property sheet, a macro, or Visual Basic.

This property can be set only in form Design view or report Design view.

For controls, you can set the default for this property by using the default control style or the **DefaultControl** method in Visual Basic.

When this property is set to Embedded, the size of the database increases by the size of the picture file and, with some .wmf files, the size may increase as much as twice the size of the picture file. When this property is set to Linked, there is no increase in the size of the database because Microsoft Access only saves a pointer to the picture's location on the disk.

**Note** If a linked file is moved to another location on the disk, you must re-establish the link by using the object's Picture property.

For embedded pictures, the object's **PictureData** property stores the individual bits that make up a picture's image. For linked pictures, this property stores the path to the picture's file.

You can modify a linked picture by using a separate application and changes to the picture will appear the next time the object containing that picture is displayed in the database.
**Example**

The following example links the picture in the "Customer Photo" image control on the "Order Entry" form to the picture on the disk. The picture is not part of the database file.

`Forms("Order Entry").Controls("Customer Photo").PictureType = 1`
PivotTable Property

Returns a PivotTable object representing a PivotTable View on a form.

expression.PivotTable

expression  Required. An expression that returns one of the objects in the Applies To list.
Example

This example reports the version of Microsoft Office Web Components in use for the specified form, assuming that there is a PivotTable View on the form.

Dim objChartSpace As PivotTable
Set objChartSpace = Forms(0).PivotTable

MsgBox "Current version of Office Web Components: " & objChartSpace.Version
PivotTableChange Property

Returns or sets a String indicating which macro, event procedure, or user-defined function runs when the PivotTableChange event occurs. Read/write.

expression.PivotTableChange

expression Required. An expression that returns one of the objects in the Applies To list.
Remarks

Valid values for this property are "macroname" where macroname is the name of macro, "[Event Procedure]" which indicates the event procedure associated with the PivotTableChange event for the specified object, or "=functionname()" where functionname is the name of a user-defined function. For a more detailed discussion of event properties, see "Event Properties."
Example

The following example specifies that when the **PivotTableChange** event occurs on the first form of the current project, the associated event procedure should run.

`Forms(0).PivotTableChange = "[Event Procedure]"`
PopUp Property

Specifies whether a form or report opens as a pop-up window. Read/write Boolean.

expression.PopUp

expression Required. An expression that returns one of the objects in the Applies To list.
Remarks

The PopUp property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>True</td>
<td>The form opens as a pop-up window. It remains on top of all other Microsoft Access windows.</td>
</tr>
<tr>
<td>No</td>
<td>False</td>
<td>(Default) The form or report isn't a pop-up window.</td>
</tr>
</tbody>
</table>

You can set this property by using the property sheet, a macro, or Visual Basic.

The PopUp property can be set only in form Design view.

To specify the type of border you want on a pop-up window, use the BorderStyle property. You typically set the BorderStyle property to Thin for pop-up windows.

Tip

To create a custom dialog box, set the Modal property to Yes, the PopUp property to Yes, and the BorderStyle property to Dialog.

Setting the PopUp property to Yes makes the form or report a pop-up window only when you do one of the following:

- Open it in Form view from the Database window.
- Open it in Form view by using a macro or Visual Basic.
- Switch from Design view to Form view.

When the PopUp property is set to Yes, you can't switch to other views from Form view because the form's toolbar isn't available. (You can't switch a pop-up form from Form view to Datasheet view, even in a macro or Visual Basic.) You must close the form and reopen it in Design or Datasheet view.

The form isn't a pop-up form in Design or Datasheet view, and also isn't if you switch from Datasheet to Form view.
**Note** You can use the Dialog setting of the Window Mode argument of the OpenForm action to open a form with its PopUp and Modal properties set to Yes.

**Tip**

When you maximize a window in Microsoft Access, all other windows are also maximized when you open them or switch to them. However, pop-up forms aren't maximized. If you want a form to maintain its size when other windows are maximized, set its PopUp property to Yes.
Example

The following example sets the "Switchboard" form to be a modal pop-up form that has just a Close button.

With Forms("Switchboard")
    .PopUp = True
    .Modal = True
    .BorderStyle = 3 ' Dialog style.
End With
Port Property

Returns a String indicating the port name of the specified printer. Read-only.

expression.Port

expression Required. An expression that returns one of the objects in the Applies To list.
Example

The following example displays information about all the printers available to the system.

Dim prtLoop As Printer

For Each prtLoop In Application.Printers
    With prtLoop
        MsgBox "Device name: " & .DeviceName & vbCrLf & "Driver name: " & .DriverName & vbCrLf & "Port: " & .Port
    End With
Next prtLoop
PostalAddress Property

You can use the **PostalAddress Property** property to specify or determine the postal code and the Customer Barcode data corresponding to the address information displayed in a specified field/textbox. The PostalAddress Property wizard enables the setting of these properties. Read/write **String**.

`expression.PostalAddress Property`

*expression*  Required. An expression that returns one of the objects in the Applies To list.

**Remarks**

For processing the conversion, correct settings are necessary for all properties of field/textbox that will contain postal code, address, Customer Barcode data.

For settings, use section 1 to 3, delimiting with semicolon (;).

**Setting for field/textbox for Postal code**

Specifies the type of postal code for the field/textbox.

<table>
<thead>
<tr>
<th><strong>Section</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Specifies field/textbox for Prefecture names</td>
</tr>
<tr>
<td>2</td>
<td>Specifies field/textbox for City/Ward/County</td>
</tr>
<tr>
<td>3</td>
<td>Specifies field/textbox for Street/Town/Village</td>
</tr>
</tbody>
</table>

**Setting for field/textbox for address**

Specifies the field/textbox contains a postal code or Customer Barcode data.

<table>
<thead>
<tr>
<th><strong>Section</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Specifies field/textbox for postal code</td>
</tr>
<tr>
<td>2</td>
<td>Specifies field/textbox for Customer Barcode data</td>
</tr>
</tbody>
</table>
Note Two semicolons are required at the end of the value.

Setting for field/textbox for Customer Barcode data

Specifies the type of Customer Barcode data in the field/textbox. This setting is the same as the field/textbox for postal code.

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Specifies field/textbox for Prefecture names</td>
</tr>
<tr>
<td>2</td>
<td>Specifies field/textbox for City/Ward/County</td>
</tr>
<tr>
<td>3</td>
<td>Specifies field/textbox for Street/Town/Village</td>
</tr>
</tbody>
</table>

The postal code consists of 3 address items; Prefecture, City/Ward/County, Street/Town/Village. Sections in **PostalAddress Property** property of field/textbox for a postal code can be omitted. The following table shows how to omit sections from the property setting.

<table>
<thead>
<tr>
<th>Property Settings</th>
<th>Address input in field/textbox</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address1</td>
<td>Prefecture+City/Ward/County+Street/Town/Village</td>
</tr>
<tr>
<td>Address1;</td>
<td>Prefecture</td>
</tr>
<tr>
<td>;Address1</td>
<td>City/Ward/County+Street/Town/Village</td>
</tr>
<tr>
<td>;Address1;</td>
<td>City/Ward/County</td>
</tr>
<tr>
<td>;;Address1</td>
<td>Street/Town/Village</td>
</tr>
<tr>
<td>Address1;Address2</td>
<td>Prefecture</td>
</tr>
<tr>
<td>Address1;Address1</td>
<td>Prefecture+City/Ward/County+Street/Town/Village</td>
</tr>
<tr>
<td>Address1;Address2;</td>
<td>Prefecture</td>
</tr>
<tr>
<td>Address1;Address1;</td>
<td>Prefecture+City/Ward/County</td>
</tr>
<tr>
<td>;Address1;Address2;</td>
<td>City/Ward/County</td>
</tr>
<tr>
<td>;Address1;Address1;</td>
<td>City/Ward/County+Street/Town/Village</td>
</tr>
<tr>
<td>Address1;Address2;Address3</td>
<td>Prefecture</td>
</tr>
<tr>
<td>Address1;Address2;Address2</td>
<td>Prefecture</td>
</tr>
<tr>
<td>Address1;Address1;Address2</td>
<td>Prefecture+City/Ward/County</td>
</tr>
<tr>
<td>Address1;Address1;Address1</td>
<td>Prefecture+City/Ward/County+Street/Town/Village</td>
</tr>
</tbody>
</table>
The postal code converter program has been developed and licensed by Advanced Giken Corporation for Microsoft Corporation.
**PreviousControl Property**

You can use the `PreviousControl` property with the `Screen` object to return a reference to the control that last received the `focus`. Read-only.

`expression.PreviousControl`

`expression` Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **PreviousControl** property contains a reference to the control that last had the focus. Once you establish a reference to the control, you can access all the properties and methods of the control.

This property is available only by using a [macro](/macro) or [Visual Basic](/visualbasic).

You can't use the **PreviousControl** property until more than one control on any form has received the focus after a form is opened. Microsoft Access generates an error if you attempt to use this property when only one control on a form has received the focus.
Example

The following example displays a message if the control that last received the focus wasn't the txtFinalEntry text box.

Public Function ProcessData() As Integer
    ' No previous control error.
    Const conNoPreviousControl = 2483
    Dim ctlPrevious As Control

    On Error GoTo Process_Err

    Set ctlPrevious = Screen.PreviousControl
    If ctlPrevious.Name = "txtFinalEntry" Then
        ' Process Data Here.
        ' ProcessData = True
    Else
        ' Set focus to txtFinalEntry and display message.
        Me!txtFinalEntry.SetFocus
        MsgBox "Please enter a value here."
        ProcessData = False
    End If

    Process_Exit:
        Set ctlPrevious = Nothing
        Exit Function

    Process_Err:
        If Err = conNoPreviousControl Then
            Me!txtFinalEntry.SetFocus
            MsgBox "Please enter a value to process.", vbInformation
            ProcessData = False
        End If
        Resume Process_Exit

    End Function
PrintCount Property

You can use the `PrintCount` property to identify the number of times the `OnPrint` property has been evaluated for the current section of a report. Read/write `Integer`.

`expression.PrintCount`

`expression`  Required. An expression that returns one of the objects in the Applies To list.
Remarks

You can use this property only in a macro or an event procedure specified by a section's OnPrint property setting.

This property isn't available in report Design view.

Microsoft Access increments the PrintCount property each time the OnPrint property setting is evaluated for the current section. As the next section is printed, Microsoft Access resets the PrintCount property to 0.

The PrintCount property is incremented, for example, when the KeepTogether property is set to No for the current section and the section is printed on more than one page. If you print a report containing order information, you might keep a running total of the order amounts.
Example

The following example shows how you can use the **PrintCount** property to make sure the value in the OrderAmount control is added only once to the running total:

```vba
Private Sub Detail_Format(Cancel As Integer, FormatCount As Integer)
    If PrintCount = 1 Then
        RunningTotal = RunningTotal + OrderAmount
    End If
End Sub
```

In the previous example, RunningTotal can be a **public variable** or the name of an **unbound control** that is incremented each time a section is printed.
**Printer Property**

Returns or sets a `Printer` object representing the default printer on the current system. Read/write.

`expression.Printer`

`expression` Required. An expression that returns one of the objects in the Applies To list.
Example

The following example makes the first printer in the **Printers** collection the default printer for the system, and then reports its name, driver information, and port information.

Dim prtDefault As Printer

Set Application.Printer = Application.Printers(0)

Set prtDefault = Application.Printer

With prtDefault
    MsgBox "Device name: " & .DeviceName & vbCr 
    & "Driver name: " & .DriverName & vbCr 
    & "Port: " & .Port
End With
Printers Property

Returns the `Printers` collection representing all the available printers on the current system.

`expression.Printers`

`expression`  Required. An expression that returns one of the objects in the Applies To list.
Example

The following example displays information about all the printers available on the current system.

Dim prtLoop As Printer

For Each prtLoop In Application.Printers
    With prtLoop
        MsgBox "Device name: " & .DeviceName & vbCr & "Driver name: " & .DriverName & vbCr & "Port: " & .Port
    End With
Next prtLoop
PrintQuality Property

Returns or sets an AcPrintObjQuality constant indicating the resolution at which the specified printer should print jobs. Read/write.

AcPrintObjQuality can be one of these AcPrintObjQuality constants.

- acPRPQDraft
- acPRPQHigh
- acPRPQLow
- acPRPQMedium

expression.PrintQuality

expression Required. An expression that returns one of the objects in the Applies To list.
Example

The following example sets a variety of printer settings for the form specified in the **strFormname** argument of the procedure.

Sub SetPrinter(strFormname As String)

    DoCmd.OpenForm FormName:=strFormname, view:=acDesign,
    datamode:=acFormEdit, windowmode:=acHidden

    With Forms(form1).Printer
    .TopMargin = 1440
    .BottomMargin = 1440
    .LeftMargin = 1440
    .RightMargin = 1440
    .ColumnSpacing = 360
    .RowSpacing = 360
    .ColorMode = acPRCMColor
    .DataOnly = False
    .DefaultSize = False
    .ItemSizeHeight = 2880
    .ItemSizeWidth = 2880
    .ItemLayout = acPRVerticalColumnLayout
    .ItemsAcross = 6
    .Copies = 1
    .Orientation = acPRORLandscape
    .Duplex = acPRDPVertical
    .PaperBin = acPRBNAuto
    .PaperSize = acPRPSLetter
    .PrintQuality = acPRPQMedium
    End With

    DoCmd.Close objecttype:=acForm, objectname:=strFormname,
    Save:=acSaveYes

End Sub
**PrintSection Property**

The **PrintSection** property specifies whether a section should be printed. Read/write **Boolean**.

\[ expression.PrintSection \]

*expression*  Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **PrintSection** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>(Default) The section is printed.</td>
</tr>
<tr>
<td>False</td>
<td>The section isn't printed.</td>
</tr>
</tbody>
</table>

**Note** To set this property, specify a [macro](#) or event procedure for a section's **OnFormat** property.

Microsoft Access sets this property to **True** before each section's **Format** event.
Example

The following example does not print the section "PageHeaderSection" of the "Product Summary" report.

Private Sub PageHeaderSection_Format(Cancel As Integer, FormatCount As Integer)
    Reports("Product Summary").PrintSection = False
End Sub
**ProcBodyLine Property**

The **ProcBodyLine** property returns a **Long** value containing the number of the line at which the body of a specified **procedure** begins in a **standard module** or a **class module**. Read-only.

*expression*.**ProcBodyLine**(**ProcName**, **ProcKind**)  

*expression*  Required. An expression that returns one of the objects in the Applies To list.

**ProcName**  Required **String**. The name of a procedure in the module.

**ProcKind**  Required **vbext_ProcKind**. The type of procedure. The constant may be one of the following values.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vbext_pk_Get</td>
<td>A <strong>Property Get</strong> procedure.</td>
</tr>
<tr>
<td>vbext_pk_Let</td>
<td>A <strong>Property Let</strong> procedure.</td>
</tr>
<tr>
<td>vbext_pk_Proc</td>
<td>A <strong>Sub</strong> or <strong>Function</strong> procedure.</td>
</tr>
<tr>
<td>vbext_pk_Set</td>
<td>A <strong>Property Set</strong> procedure.</td>
</tr>
</tbody>
</table>
Remarks

The ProcBodyLine property is available only by using Visual Basic.

The body of a procedure begins with the procedure definition, denoted by one of the following:

- A Sub statement.
- A Function statement.
- A Property Get statement.
- A Property Let statement.
- A Property Set statement.

The ProcBodyLine property returns a number that identifies the line on which the procedure definition begins. In contrast, the ProcStartLine property returns a number that identifies the line at which a procedure is separated from the preceding procedure in a module. Any comments or compilation constants that precede the procedure definition (the body of a procedure) are considered part of the procedure, but the ProcBodyLine property ignores them.

Note  The ProcBodyLine property treats Sub and Function procedures similarly, but distinguishes between each type of Property procedure.
Example

The following example displays a message indicating on which line the procedure definition begins.

Dim strForm As String
Dim strProc As String

strForm = "Products"
strProc = "Products_Subform_Enter"

MsgBox "The definition of the " & strProc & " procedure begins on line Forms(strForm).Module.ProcStartLine(strProc, vbext_pk_Proc) & "."
ProcCountLines Property

The **ProcCountLines** property returns a **Long** value containing the number of lines in a specified **procedure** in a **standard module** or a **class module**. Read-only.

**expression.ProcCountLines(ProcName, ProcKind)***

**expression** Required. An expression that returns one of the objects in the Applies To list.

**ProcName** Required **String**. The name of a procedure in the module.

**ProcKind** Required **vbext_ProcKind**. The type of procedure. The constant may be one of the following values.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vbext_pk_Get</td>
<td>A <strong>Property Get</strong> procedure.</td>
</tr>
<tr>
<td>vbext_pk_Let</td>
<td>A <strong>Property Let</strong> procedure.</td>
</tr>
<tr>
<td>vbext_pk_Proc</td>
<td>A <strong>Sub</strong> or <strong>Function</strong> procedure.</td>
</tr>
<tr>
<td>vbext_pk_Set</td>
<td>A <strong>Property Set</strong> procedure.</td>
</tr>
</tbody>
</table>
Remarks

The **ProcCountLines** property is available only by using Visual Basic.

The procedure begins with any comments and compilation constants that immediately precede the procedure definition, denoted by one of the following:

- A **Sub** statement.
- A **Function** statement.
- A **Property Get** statement.
- A **Property Let** statement.
- A **Property Set** statement.

The **ProcCountLines** property returns the number of lines in a procedure, beginning with the line returned by the **ProcStartLine** property and ending with the line that ends the procedure. The procedure may be ended with **End Sub**, **End Function**, or **End Property**.

**Note** The **ProcCountLines** property treats **Sub** and **Function** procedures similarly, but distinguishes between each type of Property procedure.
Example

The following example displays a message indicating the number of lines in a given procedure.

Dim strForm As String
Dim strProc As String

strForm = "Products"
strProc = "Form_Activate"

MsgBox "There are " & Forms(strForm).Module.ProcCountLines(strProc, " lines in the " & strProc & " procedure."
ProcOfLine Property

The **ProcOfLine** property returns a read-only **string** containing the name of the **procedure** that contains a specified line in a **standard module** or a **class module**.

**expression.**ProcOfLine(*Line, pprockind*)

*expression*  Required. An expression that returns one of the objects in the Applies To list.

*Line*  Required **Long**. The number of a line in the module.

*pprockind*  Required **vbext_ProcKind**. The type of procedure containing the line specified by the *Line* argument. The constant may be one of the following values.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vbext_pk_Get</td>
<td>A <strong>Property Get</strong> procedure.</td>
</tr>
<tr>
<td>vbext_pk_Let</td>
<td>A <strong>Property Let</strong> procedure.</td>
</tr>
<tr>
<td>vbext_pk_Proc</td>
<td>A <strong>Sub</strong> or <strong>Function</strong> procedure.</td>
</tr>
<tr>
<td>vbext_pk_Set</td>
<td>A <strong>Property Set</strong> procedure.</td>
</tr>
</tbody>
</table>

Remarks

The `ProcOfLine` property is available only by using Visual Basic.

For any given line number, the `ProcOfLine` property returns the name of the procedure that contains that line. Since comments and compilation constants immediately preceding a procedure definition are considered part of that procedure, the `ProcOfLine` property may return the name of a procedure for a line that isn't within the body of the procedure. The `ProcStartLine` property indicates the line on which a procedure begins; the `ProcBodyLine` property indicates the line on which the procedure definition begins (the body of the procedure).

Note that the `pprockind` argument indicates whether the line belongs to a Sub or Function procedure, a Property Get procedure, a Property Let procedure, or a Property Set procedure. To determine what type of procedure a line is in, pass a variable of type `Long` to the `ProcOfLine` property, then check the value of that variable.

Note  The `ProcBodyLine` property treats Sub and Function procedures similarly, but distinguishes between each type of Property procedure.
Example

The following function procedure lists the names of all procedures in a specified module:

```vba
Public Function AllProcs(ByVal strModuleName As String)
    Dim mdl As Module
    Dim lngCount As Long
    Dim lngCountDecl As Long
    Dim lngI As Long
    Dim strProcName As String
    Dim astrProcNames() As String
    Dim intI As Integer
    Dim strMsg As String
    Dim lngR As Long

    ' Open specified Module object.
    DoCmd.OpenModule strModuleName

    ' Return reference to Module object.
    Set mdl = Modules(strModuleName)

    ' Count lines in module.
    lngCount = mdl.CountOfLines

    ' Count lines in Declaration section in module.
    lngCountDecl = mdl.CountOfDeclarationLines

    ' Determine name of first procedure.
    strProcName = mdl.ProcOfLine(lngCountDecl + 1, lngR)

    ' Initialize counter variable.
    intI = 0

    ' Redimension array.
    ReDim Preserve astrProcNames(intI)

    ' Store name of first procedure in array.
    astrProcNames(intI) = strProcName

    ' Determine procedure name for each line after declarations.
    For lngI = lngCountDecl + 1 To lngCount
        ' Compare procedure name with ProcOfLine property value.
        If strProcName <> mdl.ProcOfLine(lngI, lngR) Then
            ' Increment counter.
```
intI = intI + 1
strProcName = mdl.ProcOfLine(lngI, lngR)
ReDim Preserve astrProcNames(intI)
' Assign unique procedure names to array.
astrProcNames(intI) = strProcName
End If
Next lngI

strMsg = "Procedures in module " & strModuleName & ": " & vbCrLf
For intI = 0 To UBound(astrProcNames)
    strMsg = strMsg & astrProcNames(intI) & vbCrLf
Next intI

' Message box listing all procedures in module.
MsgBox strMsg
End Function

You could call this function with a procedure such as the following:

Public Sub GetAllProcs()
    AllProcs "Utility Functions"
End Sub
ProcStartLine Property

The **ProcStartLine** property returns a read-only **Long** value identifying the line at which a specified **procedure** begins in a **standard module** or a **class module**.

*expression*.**ProcStartLine**(**ProcName**, **ProcKind**)  

*expression*  Required. An expression that returns one of the objects in the Applies To list.

**ProcName**  Required **String**. The name of a procedure in the module.

**ProcKind**  Required **vbext_ProcKind**. An **intrinsic constant** that specifies the type of procedure. The constant may be one of the following values.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vbext_pk_Get</td>
<td>A <strong>Property Get</strong> procedure.</td>
</tr>
<tr>
<td>vbext_pk_Let</td>
<td>A <strong>Property Let</strong> procedure.</td>
</tr>
<tr>
<td>vbext_pk_Proc</td>
<td>A <strong>Sub</strong> or <strong>Function</strong> procedure.</td>
</tr>
<tr>
<td>vbext_pk_Set</td>
<td>A <strong>Property Set</strong> procedure.</td>
</tr>
</tbody>
</table>
Remarks

The **ProcStartLine** property is available only by using Visual Basic.

A procedure begins with any comments and compilation constants that immediately precede the procedure definition, denoted by one of the following:

- A **Sub** statement.
- A **Function** statement.
- A **Property Get** statement.
- A **Property Let** statement.
- A **Property Set** statement.

The **ProcStartLine** property returns the number of the line on which the specified procedure begins. The beginning of the procedure may include comments or compilation constants that precede the procedure definition.

To determine the line on which the procedure definition begins, use the **ProcBodyLine** property. This property returns the number of the line that begins with a **Sub**, **Function**, **Property Get**, **Property Let**, or **Property Set** statement.

The **ProcStartLine** and **ProcBodyLine** properties can have the same value, if the procedure definition is the first line of the procedure. If the procedure definition isn't the first line of the procedure, the **ProcBodyLine** property will have a greater value than the **ProcStartLine** property.

It may be easier to determine where a procedure begins if you have the **Procedure Separator** option selected. With this option selected, there is a line between the end of a procedure and the beginning of the next procedure. The first line of code (or blank line) below the procedure separator is the first line of the following procedure, which is the line returned by the **ProcStartLine** property. The **Procedure Separator** option is located on the **Editor** tab of the **Options** dialog box, available by clicking **Options** on the **Tools** menu.
The `ProcStartLine` property treats `Sub` and `Function` procedures similarly, but distinguishes between each type of Property procedure.
Example

The following example displays a message indicating where a particular procedure starts in a particular form module.

Dim strForm As String
Dim strProc As String

strForm = "Products"
strProc = "Form_Activate"

MsgBox "The procedure " & strProc & " starts on line " & 
ProductCode Property

You can use the **ProductCode** property to determine the Microsoft Access **globally unique identifier** (GUID). Read-only **String**.

*expression*.**ProductCode**

*expression*  Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **ProductCode** property is a *string expression* that is the GUID of the Microsoft Access product.

This property is available only by using **Visual Basic**.
Example

The following example displays a message indicating the GUID for Microsoft Access for the user's computer.

MsgBox "The GUID for Microsoft Access on this computer is " & Application
ProjectType Property

You can use the **ProjectType** property to determine the type of project that is currently open through the **CurrentProject** or **CodeProject** objects. Read-only **AcProjectType**.

AcProjectType can be one of these AcProjectType constants.  
- acADP  
- acMDB  
- acNull

*expression*.**ProjectType**

*expression* Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **ProjectType** property is available only by using **Visual Basic**.
Example

The following example displays a message with details about the type of project that is currently open.

Dim intProjType As Integer

intProjType = Application.CurrentProject.ProjectType

Select Case intProjType
    Case 0 ' acNull
        MsgBox "ProjectType is acNull"
    Case 1 ' acADP
        MsgBox "ProjectType is acADP"
    Case 2 ' acMDB
        MsgBox "ProjectType is acMDB"
    Case Else
        MsgBox "Can't determine ProjectType"
End Select
Show All
Properties Property

Properties property as it applies to the AccessObject, CodeProject, and CurrentProject objects.


expression.Properties

expression  Required. An expression that returns one of the above objects.
Remarks

The `AccessObjectProperties` collection object is the collection of all the properties related to an `AccessObject`, `CurrentProject`, or `CodeProject` object. You can refer to individual members of the collection by using the member object's index or a string expression that is the name of the member object. The first member object in the collection has an index value of 0 and the total number of member objects in the collection is the value of the `AccessObjectProperties` collection's `Count` property minus 1.

You cannot use the `Properties` property to return properties from an `AccessObject` object which is a member of a collection accessed from a `CurrentData` object.

Properties property as it applies to the `SmartTag` object.

Returns a `SmartTagProperties` collection that represents the collection of properties for a particular smart tag.

```
expression.Properties()
```

expression Required. An expression that returns a `SmartTag` object.

Properties property as it applies to all other objects in the Applies To list.

Return a reference to a control's `Properties` collection object.

```
expression.Properties
```

expression Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **Properties** collection object is the collection of all the properties related to a control. You can refer to individual members of the collection by using the member object's index or a string expression that is the name of the member object. The first member object in the collection has an index value of 0 and the total number of member objects in the collection is the value of the **Properties** collection's **Count** property minus 1.
Example

The following procedure uses the Properties property to print all the properties associated with the controls on a form to the Debug window. To run this code, place a command button named cmdListProperties on a form and paste the following code into the form's Declarations section. Click the command button to print the list of properties in the Debug window.

Private Sub cmdListProperties_Click()
    ListControlProps Me
End Sub

Public Sub ListControlProps(ByRef frm As Form)
    Dim ctl As Control
    Dim prp As Property

    On Error GoTo props_err

    For Each ctl In frm.Controls
        Debug.Print ctl.Properties("Name")
        For Each prp In ctl.Properties
            Debug.Print vbTab & prp.Name & " = " & prp.Value
        Next prp
    Next ctl

    props_exit:
        Set ctl = Nothing
        Set prp = Nothing
    Exit Sub

    props_err:
        If Err = 2187 Then
            Debug.Print vbTab & prp.Name & " = Only available at design time."
            Resume Next
        Else
            Debug.Print vbTab & prp.Name & " = Error Occurred: " & Err.D
            Resume Next
        End If
    End Sub
PrtDevMode Property

You can use the **PrtDevMode** property to set or return printing device mode information specified for a form or report in the **Print** dialog box. Read/write **Variant**.

*expression*.PrtDevMode

*expression*  Required. An expression that returns one of the objects in the Applies To list.
Remarks

It is strongly recommended that you consult the Win32 Software Development Kit for complete documentation on the **PrtDevMode**, **PrtDevNames**, and **PrtMip** properties.

The **PrtDevMode** property setting is a 94-byte structure that mirrors the DEVMODE structure defined in the Win32 Software Development Kit. For complete information on the **PrtDevMode** property members, consult the Win32 Software Development Kit.

The **PrtDevMode** property uses the following members.

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DeviceName</td>
<td>A <strong>string</strong> with a maximum of 32 bytes that specifies the name of the device the driver supports— for example, &quot;HP LaserJet IIISi&quot; if the Hewlett-Packard LaserJet IIISi is the specified printer. Each printer driver has a unique string.</td>
</tr>
<tr>
<td>SpecVersion</td>
<td>An <strong>Integer</strong> that specifies the version number of the DEVMODE structure in the Win32 Software Development Kit.</td>
</tr>
<tr>
<td>DriverVersion</td>
<td>An <strong>Integer</strong> that specifies the printer driver version number assigned by the printer driver developer.</td>
</tr>
<tr>
<td>Size</td>
<td>An <strong>Integer</strong> that specifies the size, in bytes, of the DEVMODE structure. (This value doesn't include the optional <strong>dmDriverData</strong> member for device-specific data, which can follow this structure.) If an application manipulates only the driver-independent portion of the data, you can use this member to find out the length of this structure without having to account for different versions.</td>
</tr>
<tr>
<td>DriverExtra</td>
<td>An <strong>Integer</strong> that specifies the size, in bytes, of the optional <strong>dmDriverData</strong> member for device-specific data, which can follow this structure. If an application doesn't use device-specific information, you set this member to 0.</td>
</tr>
<tr>
<td>Fields</td>
<td>A <strong>Long</strong> value that specifies which of the remaining members in the DEVMODE structure have been initialized. An <strong>Integer</strong> that specifies the orientation of the paper. It can be</td>
</tr>
</tbody>
</table>
Orientation

either 1 (portrait) or 2 (landscape).

PaperSize

An **Integer** that specifies the size of the paper to print on. If you set this member to 0 or 256, the length and width of the paper are specified by the PaperLength and PaperWidth members, respectively. Otherwise, you can set the PaperSize member to a predefined value. For available values, see the PaperSize member values.

PaperLength

An **Integer** that specifies the paper length in units of 1/10 of a millimeter. This member overrides the paper length specified by the PaperSize member for custom paper sizes or for devices such as dot-matrix printers that can print on a variety of paper sizes.

PaperWidth

An **Integer** that specifies the paper width in units of 1/10 of a millimeter. This member overrides the paper width specified by the PaperSize member.

Scale

An **Integer** that specifies the factor by which the printed output will be scaled. The apparent page size is scaled from the physical page size by a factor of scale/100. For example, a piece of paper measuring 8.5 by 11 inches (letter-size) with a Scale value of 50 would contain as much data as a page measuring 17 by 22 inches because the output text and graphics would be half their original height and width.

Copies

An **Integer** that specifies the number of copies printed if the printing device supports multiple-page copies.

DefaultSource

An **Integer** that specifies the default bin from which the paper is fed. For available values, see the DefaultSource member values.

PrintQuality

An **Integer** that specifies the printer resolution. The values are –4 (high), –3 (medium), –2 (low), and –1 (draft).

Color

An **Integer**. For a color printer, specifies whether the output is printed in color. The values are 1 (color) and 2 (monochrome).

Duplex

An **Integer**. For a printer capable of duplex printing, specifies whether the output is printed on both sides of the paper. The values are 1 (simplex), 2 (horizontal), and 3 (vertical).

YResolution

An **Integer** that specifies the y-resolution of the printer in dots per inch (dpi). If the printer initializes this member, the PrintQuality member specifies the x-resolution of the printer in dpi.
TTOption  An Integer that specifies how TrueType fonts will be printed. For available values, see the TTOption member values.

Collate  An Integer that specifies whether collation should be used when printing multiple copies. Using uncollated copies provides faster, more efficient output, since the data is sent to the printer just once.

FormName  A string with a maximum of 16 characters that specifies the size of paper to use; for example, "Letter" or "Legal".

Pad  A Long value that is used to pad out spaces, characters, or values for future versions.

Bits  A Long value that specifies in bits per pixel the color resolution of the display device.

PW  A Long value that specifies the width, in pixels, of the visible device surface (screen or printer).

PH  A Long value that specifies the height, in pixels, of the visible device surface (screen or printer).

DFI  A Long value that specifies the device's display mode.

DFR  A Long value that specifies the frequency, in hertz (cycles per second), of the display device in a particular mode.

Note  You can set the PrtDevMode property using Visual Basic.

This property setting is read/write in Design view and read-only in other views.

Caution  Printer drivers can add device-specific data immediately following the 94 bytes of the DEVMODE structure. For this reason, it is important that the DEVMODE data outlined above not exceed 94 bytes.

Only printer drivers that export the ExtDeviceMode function use the DEVMODE structure.

An application can retrieve the paper sizes and names supported by a printer by using the DC_PAPERS, DC_PAPERSIZE, and DC_PAPERNAMES values to call the DeviceCapabilities function.

Before setting the value of the TTOption member, applications should find out how a printer driver can use TrueType fonts by using the DC_TRUETYPE value to call the DeviceCapabilities function.
The following example uses the **PrtDevMode** property to check the user-defined page size for a report:

```vba
Private Type str_DEVMODE
    RGB As String * 94
End Type

Private Type type_DEVMODE
    strDeviceName As String * 32
    intSpecVersion As Integer
    intDriverVersion As Integer
    intSize As Integer
    intDriverExtra As Integer
    lngFields As Long
    intOrientation As Integer
    intPaperSize As Integer
    intPaperLength As Integer
    intPaperWidth As Integer
    intScale As Integer
    intCopies As Integer
    intDefaultSource As Integer
    intPrintQuality As Integer
    intColor As Integer
    intDuplex As Integer
    intResolution As Integer
    intTTOption As Integer
    intCollate As Integer
    strFormName As String * 32
    lngPad As Long
    lngBits As Long
    lngPW As Long
    lngPH As Long
    lngDFI As Long
    lngDFr As Long
End Type

Public Sub CheckCustomPage(ByVal rptName As String)
    Dim DevString As str_DEVMODE
    Dim DM As type_DEVMODE
    Dim strDevModeExtra As String
    Dim rpt As Report
    Dim intResponse As Integer
```
' Opens report in Design view.
DoCmd.OpenReport rptName, acDesign
Set rpt = Reports(rptName)

If Not IsNull(rpt.PrtDevMode) Then
    strDevModeExtra = rpt.PrtDevMode

    ' Gets current DEVMODE structure.
    DevString.RGB = strDevModeExtra
    LSet DM = DevString
    If DM.intPaperSize = 256 Then

        ' Display user-defined size.
        intResponse = MsgBox("The current custom page size is ",
                           DM.intPaperWidth / 254 & " inches wide by ",
                           DM.intPaperLength / 254 & " inches long. Do ",
                           "to change the settings?", vbYesNo + vbQue
    Else

        ' Currently not user-defined.
        intResponse = MsgBox("The report does not have a custom ",
                             "Do you want to define one?", vbYesNo + vb

    End If

    If intResponse = vbYes Then

        ' User wants to change settings. Initialize fields.
        DM.lngFields = DM.lngFields Or DM.intPaperSize Or _
                       DM.intPaperLength Or DM.intPaperWidth

        ' Set custom page.
        DM.intPaperSize = 256

        ' Prompt for length and width.
        DM.intPaperLength = InputBox("Please enter page length in ",
                                      DM.intPaperWidth = InputBox("Please enter page width in 

        ' Update property.
        LSet DevString = DM
        Mid(strDevModeExtra, 1, 94) = DevString.RGB
        rpt.PrtDevMode = strDevModeExtra

    End If

    End If

    Set rpt = Nothing

End Sub

The following example shows how to change the orientation of the report. This example will switch the orientation from portrait to landscape or landscape to
portrait depending on the report's current orientation.

Public Sub SwitchOrient(ByVal strName As String)

    Const DM_PORTRAIT = 1
    Const DM_LANDSCAPE = 2
    Dim DevString As str_DEVMODE
    Dim DM As type_DEVMODE
    Dim strDevModeExtra As String
    Dim rpt As Report

    ' Opens report in Design view.
    DoCmd.OpenReport strName, acDesign
    Set rpt = Reports(strName)

    If Not IsNull(rpt.PrtDevMode) Then
        strDevModeExtra = rpt.PrtDevMode
        DevString.RGB = strDevModeExtra
        LSet DM = DevString
        DM.lngFields = DM.lngFields Or DM.intOrientation

        ' Initialize fields.
        If DM.intOrientation = DM_PORTRAIT Then
            DM.intOrientation = DM_LANDSCAPE
        Else
            DM.intOrientation = DM_PORTRAIT
        End If

        ' Update property.
        LSet DevString = DM
        Mid(strDevModeExtra, 1, 94) = DevString.RGB
        rpt.PrtDevMode = strDevModeExtra
    End If

    Set rpt = Nothing

End Sub
PrtDevNames Property

You can use the **PrtDevNames** property to set or return information about the printer selected in the **Print** dialog box for a form or report. Read/write **Variant**.

```vba
expression.PrtDevNames
```

**expression**  Required. An expression that returns one of the objects in the Applies To list.
Remarks

It is strongly recommended that you consult the Win32 Software Development Kit for complete documentation on the **PrtDevMode**, **PrtDevNames**, and **PrtMip** properties.

The **PrtDevNames** property is a variable-length structure that mirrors the DEVNAMES structure defined in the Win32 Software Development Kit.

The **PrtDevNames** property uses the following members.

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DriverOffset</td>
<td>Specifies the offset from the beginning of the structure to a <strong>Null</strong>-terminated string that specifies the file name (without an extension) of the device driver. This string is used to specify which printer is initially displayed in the <strong>Print</strong> dialog box. Specifies the offset from the beginning of the structure to the <strong>Null</strong>-terminated string that specifies the name of the device. This string can't be longer than 32 bytes (including the null character) and must be identical to the DeviceName member of the DEVMODE structure.</td>
</tr>
<tr>
<td>DeviceOffset</td>
<td>Specifies the offset from the beginning of the structure to the <strong>Null</strong>-terminated string that specifies the MS-DOS device name for the physical output medium (output port); for example, &quot;LPT1:&quot;. Specifies whether the strings specified in the DEVNAMES structure identify the default printer. Before the <strong>Print</strong> dialog box is displayed, if Default is set to 1 and all of the values in the DEVNAMES structure match the current default printer, the selected printer is set to the default printer. Default is set to 1 if the current default printer has been selected.</td>
</tr>
<tr>
<td>OutputOffset</td>
<td><strong>Null</strong>-terminated string that specifies the MS-DOS device name for the physical output medium (output port); for example, &quot;LPT1:&quot;.</td>
</tr>
<tr>
<td>Default</td>
<td><strong>null</strong>-terminated string that specifies the MS-DOS device name for the physical output medium (output port); for example, &quot;LPT1:&quot;. Specifies whether the strings specified in the DEVNAMES structure identify the default printer. Before the <strong>Print</strong> dialog box is displayed, if Default is set to 1 and all of the values in the DEVNAMES structure match the current default printer, the selected printer is set to the default printer. Default is set to 1 if the current default printer has been selected.</td>
</tr>
</tbody>
</table>

Microsoft Access sets the **PrtDevNames** property when you make selections in the Printer section of the **Print** dialog box. You can also set the property by using **Visual Basic**.

Microsoft Access uses the DEVNAMES structure to initialize the **Print** dialog box. When the user chooses **OK** to close the dialog box, information about the
selected printer is returned by the **PrtDevNames** property.
PrtMip Property

You can use the **PrtMip** property in Visual Basic to set or return the device mode information specified for a form or report in the **Print** dialog box.

`expression.PrtMip`

*expression* Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **PrtMip** property setting is a 28-byte structure that maps to settings on the **Margins** tab for a form or report in the **Page Setup** dialog box.

The **PrtMip** property has the following members.

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LeftMargin, RightMargin, TopMargin, BottomMargin</td>
<td>A <strong>Long</strong> that specifies the distance between the edge of the page and the item to be printed in <strong>twips</strong>.</td>
</tr>
<tr>
<td>DataOnly</td>
<td>A <strong>Long</strong> that specifies the elements to be printed. When <strong>True</strong>, prints only the data in a table or query in <strong>Datasheet view</strong>, form, or report, and suppresses labels, <strong>control</strong> borders, grid lines, and display graphics such as lines and boxes. When <strong>False</strong>, prints data, labels, and graphics.</td>
</tr>
<tr>
<td>ItemsAcross</td>
<td>A <strong>Long</strong> that specifies the number of columns across for multiple-column reports or labels. This member is equivalent to the value of the <strong>Number of Columns</strong> box under <strong>Grid Settings</strong> on the <strong>Columns</strong> tab of the <strong>Page Setup</strong> dialog box.</td>
</tr>
<tr>
<td>RowSpacing</td>
<td>A <strong>Long</strong> that specifies the horizontal space between detail sections in units of 1/20 of a point.</td>
</tr>
<tr>
<td>ColumnSpacing</td>
<td>A <strong>Long</strong> that specifies the vertical space between detail sections in <strong>twips</strong>.</td>
</tr>
<tr>
<td>DefaultSize</td>
<td>A <strong>Long</strong>. When <strong>True</strong>, uses the size of the detail section in <strong>Design view</strong>. When <strong>False</strong>, uses the values specified by the <strong>ItemSizeWidth</strong> and <strong>ItemSizeHeight</strong> members.</td>
</tr>
<tr>
<td>ItemSizeWidth</td>
<td>A <strong>Long</strong> that specifies the width of the detail section in <strong>twips</strong>. This member is equivalent to the value of the <strong>Width</strong> box under <strong>Column Size</strong> on the <strong>Columns</strong> tab of the <strong>Page Setup</strong> dialog box.</td>
</tr>
<tr>
<td>ItemSizeHeight</td>
<td>A <strong>Long</strong> that specifies the height of the detail section <strong>twips</strong>. This member is equivalent to the value of the <strong>Height</strong> box under <strong>Column Size</strong> on the <strong>Columns</strong> tab of the <strong>Page Setup</strong> dialog box.</td>
</tr>
</tbody>
</table>
ItemLayout: A Long that specifies horizontal (1953) or vertical (1954) layout of columns. This member is equivalent to Across, then Down or Down, then Across respectively under Column Layout on the Columns tab of the Page Setup dialog box.

FastPrint: Reserved.

Datasheet: Reserved.

The PrtMip property setting is read/write in Design view and read-only in other views.
Example

The following **PrtMip** property example demonstrates how to set up the report with two horizontal columns.

Private Type str_PRTMIP
    strRGB As String * 28
End Type

Private Type type_PRTMIP
    xLeftMargin As Long
    yTopMargin As Long
    xRightMargin As Long
    yBotMargin As Long
    fDataOnly As Long
    xWidth As Long
    yHeight As Long
    fDefaultSize As Long
    cxColumns As Long
    yColumnSpacing As Long
    xRowSpacing As Long
    rItemLayout As Long
    fFastPrint As Long
    fDatasheet As Long
End Type

Public Sub PrtMipCols(ByVal strName As String)

    Dim PrtMipString As str_PRTMIP
    Dim PM As type_PRTMIP
    Dim rpt As Report
    Const PM_HORIZONTALCOLS = 1953
    Const PM_VERTICALCOLS = 1954

    ' Open the report.
    DoCmd.OpenReport strName, acDesign
    Set rpt = Reports(strName)
    PrtMipString.strRGB = rpt.PrtMip
    LSet PM = PrtMipString

    ' Create two columns.
    PM.cxColumns = 2

    ' Set 0.25 inch between rows.
    PM.xRowSpacing = 0.25 * 1440
' Set 0.5 inch between columns.
PM.yColumnSpacing = 0.5 * 1440
PM.rItemLayout = PM_HORIZONTALCOLS

' Update property.
LSet PrtMipString = PM
rpt.PrtMip = PrtMipString.strRGB

Set rpt = Nothing
End Sub

The next PrtMip property example shows how to set all margins to 1 inch.

Public Sub SetMarginsToDefault(ByVal strName As String)
    Dim PrtMipString As str_PRTMIP
    Dim PM As type_PRTMIP
    Dim rpt As Report

    ' Open the report.
    DoCmd.OpenReport strName, acDesign
    Set rpt = Reports(strName)
    PrtMipString.strRGB = rpt.PrtMip
    LSet PM = PrtMipString

    ' Set margins.
    PM.xLeftMargin = 1 * 1440
    PM.yTopMargin = 1 * 1440
    PM.xRightMargin = 1 * 1440
    PM.yBotMargin = 1 * 1440

    ' Update property.
    LSet PrtMipString = PM
    rpt.PrtMip = PrtMipString.strRGB

    Set rpt = Nothing
End Sub
Query Property

Returns or sets a **String** indicating which macro, event procedure, or user-defined function runs when the **Query** event occurs. Read/write.

`expression.Query`

*expression*  Required. An expression that returns one of the objects in the Applies To list.
Remarks

Valid values for this property are "macroname" where macroname is the name of macro, "[Event Procedure]" which indicates the event procedure associated with the Query event for the specified object, or "=functionname()" where functionname is the name of a user-defined function. For a more detailed discussion of event properties, see "Event Properties."

This property corresponds to the On Query property that is displayed in a form's property sheet.
Example

The following example specifies that when the Query event occurs on the first form of the current project, the associated event procedure should run.

Forms(0).Query = "[Event Procedure]"
ReadingOrder Property

You can use the **ReadingOrder** property to specify or determine the reading order of words in text. Read/write.

*expression*.ReadingOrder

*expression*  Required. An expression that returns one of the objects in the Applies To list.
## Remarks

The **ReadingOrder** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Context 0</td>
<td></td>
<td>Reading order is determined by the language of the first character entered. If a right-to-left language character is entered first, reading order is right to left. If a left-to-right language character is entered first, reading order is left to right.</td>
</tr>
<tr>
<td>Left-to-Right 1</td>
<td></td>
<td>Sets the reading order to left to right.</td>
</tr>
<tr>
<td>Right-to-Left 2</td>
<td></td>
<td>Sets the reading order to right to left.</td>
</tr>
</tbody>
</table>

**Note**  You can set this property by using the [property sheet](#) or [Visual Basic](#).

In a combo box or list box, the **ReadingOrder** property determines reading order behavior for both the text box and list box components of the control.
Example

The following example sets the reading order to right to left for the "Address" text box on the "International Shipping" form.

Forms("International Shipping").Controls("Address").ReadingOrder = 2
RecordLocks Property

You can use the `RecordLocks` property to determine how records are locked and what happens when two users try to edit the same record at the same time. Read/write.

`expression.RecordLocks`

`expression` Required. An expression that returns one of the objects in the Applies To list.
Remarks

When you edit a record, Microsoft Access can automatically lock that record to prevent other users from changing it before you are finished.

- **Forms.** Specifies how records in the underlying table or query are locked when data in a multiuser database is updated.

- **Reports.** Specifies whether records in the underlying table or query are locked while a report is previewed or printed.

- **Queries.** Specifies whether records in a query (typically an action query in a multiuser database) are locked while the query is run.

**Note** The **RecordLocks** property only applies to forms, reports, or queries in a Microsoft Access database (.mdb).

The **RecordLocks** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Locks</td>
<td>0</td>
<td>(Default) In forms, two or more users can edit the same record simultaneously. This is also called &quot;optimistic&quot; locking. If two users attempt to save changes to the same record, Microsoft Access displays a message to the user who tries to save the record second. This user can then discard the record, copy the record to the Clipboard, or replace the changes made by the other user. This setting is typically used on read-only forms or in single-user databases. It is also used in multiuser databases to permit more than one user to be able to make changes to the same record at the same time. In reports, records aren't locked while the report is previewed or printed. In queries, records aren't locked while the query is run. All records in the underlying table or query are locked while the</td>
</tr>
</tbody>
</table>
form is open in **Form view** or **Datasheet view**, while the report is previewed or printed, or while the query is run. Although users can read the records, no one can edit, add, or delete any records until the form is closed, the report has finished printing, or the query has finished running.

(Form and queries only) A page of records is locked as soon as any user starts editing any field in the record and stays locked until the user moves to another record. Consequently, a record can be edited by only one user at a time. This is also called "pessimistic" locking.

**Note** You can set this property by using a form's **property sheet**, a **macro**, or **Visual Basic**.

**Note** Changing the **RecordLocks** property of an open form or report causes an automatic recreation of the recordset.

You can use the No Locks setting for forms if only one person uses the underlying tables or queries or makes all the changes to the data.

In a multiuser database, you can use the No Locks setting if you want to use optimistic locking and warn users attempting to edit the same record on a form. You can use the Edited Record setting if you want to prevent two or more users editing data at the same time.

You can use the All Records setting when you need to ensure that no changes are made to data after you start to preview or print a report or run an **append**, **delete**, **make-table**, or **update** query.

In **Form view** or **Datasheet view**, each locked record has a locked indicator in its record selector.

**Tip**

To change the default **RecordLocks** property setting for forms, click **Options** on the **Tools** menu, click the **Advanced** tab on the **Options** dialog box, and then select the desired option under **Default record locking**.
Data in a form, report, or query from an Open Database Connectivity (ODBC) database is treated as if the No Locks setting were chosen, regardless of the RecordLocks property setting.
Example

The following example sets the `RecordLocks` property of the "Employees" form to Edited Record (a page of records is locked as soon as any user starts editing any field in the record and stays locked until the user moves to another record).

```plaintext
Forms("Employees").RecordLocks = 2
```
RecordSelectors Property

You can use the RecordSelectors property to specify whether a form displays record selectors in Form view. Read/write Boolean.

expression.RecordSelectors

expression Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **RecordSelectors** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>True</td>
<td>(Default) Each record has a record selector.</td>
</tr>
<tr>
<td>No</td>
<td>False</td>
<td>No record has a record selector.</td>
</tr>
</tbody>
</table>

**Note** You can set this property by using the form's [property sheet](https://example.com), a [macro](https://example.com), or [Visual Basic](https://example.com).

You can use this property to remove record selectors when you create or use a form as a [custom dialog box](https://example.com) or a palette. You can also use this property for forms whose [DefaultView](https://example.com) property is set to Single Form.

The record selector displays the unsaved record indicator when a record is being edited. When the **RecordSelectors** property is set to No and the **RecordLocks** property is set to Edited Record (record locking is "pessimistic"— only one person can edit a record at a time), there is no visual clue that the record is locked.
Example

The following example specifies that no record has a record selector in the "Employees" form.

Forms("Employees").RecordSelectors = False
Recordset Property

Returns or sets the ADO Recordset or DAO Recordset object representing the record source for the specified form, report, list box control, or combo box control. Read/write.

expression.Recordset

expression Required. An expression that returns one of the objects in the Applies To list.
Remarks

You cannot use this property with ODBCDirect recordset types in DAO.

The **Recordset** property returns the recordset object that provides the data being browsed in a form, report, list box control, or combo box control. If a form is based on a query, for example, referring to the **Recordset** property is the equivalent of cloning a **Recordset** object by using the same query. However, unlike using the **RecordsetClone** property, changing which record is current in the recordset returned by the form's **Recordset** property also sets the current record of the form.

This property is available only by using **Visual Basic**.

The read/write behavior of the **Recordset** property is determined by the type of recordset (ADO or DAO) and the type of data (Jet or SQL) contained in the recordset identified by the property.

### Recordset type Based on SQL data Based on Jet data

<table>
<thead>
<tr>
<th></th>
<th>ADO</th>
<th>DAO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read/Write</td>
<td>Read/Write</td>
<td>N/A</td>
</tr>
<tr>
<td>Read/Write</td>
<td></td>
<td>Read/Write</td>
</tr>
</tbody>
</table>

The following example opens a form, opens a recordset, and then binds the form to the recordset by setting the form's **Recordset** property to the newly created **Recordset** object.

```vba
Global rstSuppliers As ADODB.Recordset
Sub MakeRW()
    DoCmd.OpenForm "Suppliers"
    Set rstSuppliers = New ADODB.Recordset
    rstSuppliers.CursorLocation = adUseClient
    rstSuppliers.Open "Select * From Suppliers", _
    CurrentProject.Connection, adOpenKeyset, adLockOptimistic
    Set Forms("Suppliers").Recordset = rstSuppliers
End Sub
```

Use the **Recordset** property:

- To bind multiple forms to a common data set. This allows synchronization of multiple forms. For example,
Set Me.Recordset = Forms!Form1.Recordset

- To use methods with the **Recordset** object that aren't directly supported on forms. For example, you can use the **Recordset** property with the ADO **Find** or DAO **Find** methods in a custom dialog for finding a record.
- To wrap a transaction (which can be rolled back) around a set of edits that affect multiple forms.

Changing a form's **Recordset** property may also change the **RecordSource**, **RecordsetType**, and **RecordLocks** properties. Also, some data-related properties may be overridden, for example, the **Filter**, **FilterOn**, **OrderBy**, and **OrderByOn** properties.

Calling the **Requery** method of a form's recordset (for example, `Forms(0).Recordset.Requery`) can cause the form to become unbound. To refresh the data in a form bound to a recordset, set the **RecordSource** property of the form to itself (`Forms(0).RecordSource = Forms(0).RecordSource`).

When a form is bound to a recordset, an error occurs if you use the Filter by Form command.
Example

The following example uses the `Recordset` property to create a new copy of the `Recordset` object from the current form and then prints the names of the fields in the Debug window.

```vba
Sub Print_Field_Names()
    Dim rst As DAO.Recordset, intI As Integer
    Dim fld As Field

    Set rst = Me.Recordset
    For Each fld in rst.Fields
        ' Print field names.
        Debug.Print fld.Name
    Next
End Sub
```

The next example uses the `Recordset` property and the `Recordset` object to synchronize a recordset with the form's current record. When a company name is selected from a combo box, the `FindFirst` method is used to locate the record for that company, causing the form to display the found record.

```vba
Sub SupplierID_AfterUpdate()
    Dim rst As DAO.Recordset
    Dim strSearchName As String

    Set rst = Me.Recordset
    strSearchName = CStr(Me!SupplierID)
    rst.FindFirst "SupplierID = " & strSearchName
    If rst.NoMatch Then
        MsgBox "Record not found"
    End If
    rst.Close
End Sub
```

The following code helps to determine what type of recordset is returned by the `Recordset` property under different conditions.

```vba
Sub CheckRSType()
    Dim rs as Object

    Set rs=Forms(0).Recordset
```
If TypeOf rs Is DAO.Recordset Then
    MsgBox "DAO Recordset"
ElseIf TypeOf rs is ADODB.Recordset Then
    MsgBox "ADO Recordset"
End If
End Sub
RecordsetClone Property

You can use the **RecordsetClone** property to refer to a form's **Recordset** object specified by the form's **RecordSource** property. Read-only.

`expression.RecordsetClone`

`expression` Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **RecordsetClone** property setting is a copy of the underlying query or table specified by the form's **RecordSource** property. If a form is based on a query, for example, referring to the **RecordsetClone** property is the equivalent of cloning a **Recordset** object by using the same query. If you then apply a filter to the form, the **Recordset** object reflects the filtering.

This property is available only by using **Visual Basic** and is read-only in all views.

You use the **RecordsetClone** property to navigate or operate on a form's records independent of the form itself. For example, you can use the **RecordsetClone** property when you want to use a method, such as the DAO **Find** methods, that can't be used with forms.

When a new **Recordset** object is opened, its first record is the current record. If you one of the **Find** method or one of the **Move** methods to make any other record in the **Recordset** object current, you must synchronize the current record in the **Recordset** object with the form's current record by assigning the value of the DAO **Bookmark** property to the form's **Bookmark** property.

You can use the **RecordCount** property to count the number of records in a **Recordset** object. The following example shows how you can combine the **RecordCount** property and the **RecordsetClone** property to count the records in a form:

```vba
Forms!Orders.RecordsetClone.MoveLast
MsgBox "My form contains " & Forms!Orders.RecordsetClone.RecordCount & " records.", vbInformation, "Record Count"
```

**Note** If you close the form or if you change the form's **RecordSource** property, the **Recordset** object is no longer valid. If you subsequently refer to the **Recordset** object or to previously saved bookmarks in the form or the **Recordset** object, an error will occur.
Example

The following example uses the RecordsetClone property to create a new clone of the Recordset object from the Orders form and then prints the names of the fields in the Immediate window.

Sub Print_Field_Names()
    Dim rst As Recordset, intI As Integer
    Dim fld As Field

    Set rst = Me.RecordsetClone
    For Each fld in rst.Fields
        ' Print field names.
        Debug.Print fld.Name
    Next
End Sub

The next example uses the RecordsetClone property and the Recordset object to synchronize a recordset's record with the form's current record. When a company name is selected from a combo box, the FindFirst method is used to locate the record for that company and the Recordset object's DAO Bookmark property is assigned to the form's Bookmark property, causing the form to display the found record.

Sub SupplierID_AfterUpdate()
    Dim rst As Recordset
    Dim strSearchName As String

    Set rst = Me.RecordsetClone
    strSearchName = Str(Me!SupplierID)
    rst.FindFirst "SupplierID = " & strSearchName
    If rst.NoMatch Then
        MsgBox "Record not found"
    Else
        Me.Bookmark = rst.Bookmark
    End If
    rst.Close
End Sub
RecordsetType Property

You can use the RecordsetType property to specify what kind of recordset is made available to a form. Read/write.

expression.RecordsetType

expression Required. An expression that returns one of the objects in the Applies To list.
## Remarks

The **RecordsetType** property uses the following settings in a Microsoft Access database (.mdb).

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dynaset</td>
<td>0 (Default)</td>
<td>You can edit bound controls based on a single table or tables with a one-to-one relationship. For controls bound to fields based on tables with a one-to-many relationship, you can't edit data from the join field on the &quot;one&quot; side of the relationship unless cascade update is enabled between the tables. For more information, see the topic that explains <a href="https://support.microsoft.com/en-us/kb/300826">when you can update records from a query</a>.</td>
</tr>
<tr>
<td>Dynaset (Inconsistent Updates)</td>
<td></td>
<td>All tables and controls bound to their fields can be edited.</td>
</tr>
<tr>
<td>Snapshot</td>
<td>2</td>
<td>No tables or the controls bound to their fields can be edited.</td>
</tr>
</tbody>
</table>

**Note** If you don't want data in bound controls to be edited when a form is in *Form view* or *Datasheet view*, you can set the **RecordsetType** property to *Snapshot*.

The **RecordsetType** property uses the following settings in a Microsoft Access project (.adp).

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Snapshot</td>
<td>3</td>
<td>No tables or the controls bound to their fields can be edited.</td>
</tr>
<tr>
<td>Updatable Snapshot</td>
<td>4 (Default)</td>
<td>All tables and controls bound to their fields can be edited.</td>
</tr>
</tbody>
</table>
Note  You can set this property by using a form's property sheet, a macro, or Visual Basic.

Note  Changing the RecordsetType property of an open form or report causes an automatic recreation of the recordset.

You can create forms based on multiple underlying tables with fields bound to controls on the forms. Depending on the RecordsetType property setting, you can limit which of these bound controls can be edited.

In addition to the editing control provided by RecordsetType, each control on a form has a Locked property that you can set to specify whether the control and its underlying data can be edited. If the Locked property is set to Yes, you can't edit the data.
Example

In the following example, only if the user ID is ADMIN can records be updated. This code sample sets the **RecordsetType** property to Snapshot if the public variable `gstrUserID` value is not ADMIN.

```vba
Sub Form_Open(Cancel As Integer)
    Const conSnapshot = 2
    If gstrUserID <> "ADMIN" Then
        Forms!Employees.RecordsetType = conSnapshot
    End If
End Sub
```
RecordSource Property

You can use the RecordSource property to specify the source of the data for a form or report. Read/write String.

expression.RecordSource

expression Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **RecordSource** property setting can be a table name, a query name, or an SQL statement. For example, you can use the following settings.

<table>
<thead>
<tr>
<th>Sample setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Employees</td>
<td>A table name specifying the Employees table as the source of data.</td>
</tr>
<tr>
<td>SELECT Orders!OrderDate FROM Orders;</td>
<td>An SQL statement specifying the OrderDate field on the Orders table as the source of data. You can bind a control on the form or report to the OrderDate field in the Orders table by setting the control's <strong>ControlSource</strong> property to OrderDate.</td>
</tr>
</tbody>
</table>

**Note** You can set the **RecordSource** property by using the form's or report's **property sheet**, a **macro**, or **Visual Basic**.

In Visual Basic, use a **string expression** to set this property.

**Note** Changing the record source of an open form or report causes an automatic requery of the underlying data. If a form's **Recordset** property is set at runtime, the form's **RecordSource** property is updated.

After you have created a form or report, you can change its source of data by changing the **RecordSource** property. The **RecordSource** property is also useful if you want to create a reusable form or report. For example, you could create a form that incorporates a standard design, then copy the form and change the **RecordSource** property to display data from a different table, query, or SQL statement.

Limiting the number of records contained in a form's record source can enhance performance, especially when your application is running on a network. For example, you can set a form's **RecordSource** property to an SQL statement that returns a single record and change the form's record source depending on criteria selected by the user.
Example

The following example sets a form's `RecordSource` property to the Customers table:

```
Forms!frmCustomers.RecordSource = "Customers"
```

The next example changes a form's record source to a single record in the Customers table, depending on the company name selected in the `cmboCompanyName` combo box control. The combo box is filled by an SQL statement that returns the customer ID (in the bound column) and the company name. The CustomerID has a Text data type.

```
Sub cmboCompanyName_AfterUpdate()
    Dim strNewRecord As String
    strNewRecord = "SELECT * FROM Customers " _
    & "WHERE CustomerID = " _
    & Me!cmboCompanyName.Value & ""
    Me.RecordSource = strNewRecord
End Sub
```
RecordSourceQualifier Property

Returns or sets a String indicating the SQL Server owner name of the record source for the specified form or report. Read/write.

expression.RecordSourceQualifier

expression   Required. An expression that returns one of the objects in the Applies To list.
Example

The following example displays all the record source information for the specified form.

With Forms(0)
    MsgBox "Record Source: " & .RecordSource & vbCrLf _
    & "Record Source Qualifier: " _
    & .RecordSourceQualifier
End With
References Property

You can use the **References** property to access the **References** collection and its related properties, methods, and events.

`expression.References`

`expression`  Required. An expression that returns one of the objects in the Applies To list.
Remarks

This property is available only by using Visual Basic and is read-only.

The References collection corresponds to the list of references in the References dialog box, available by clicking References on the Tools menu. Each Reference object represents one selected reference in the list. References that appear in the References dialog box but haven't been selected aren't in the References collection.
Example

The following example displays a message indicating the number of boxes checked in the References dialog box.

MsgBox "There are " & Application.References.Count & " references."
RemovePersonalInformation Property

Returns or sets a Boolean indicating whether personal information about the user is stored in the specified project or data access page. True if personal information is removed. Read-write.

expression.RemovePersonalInformation

expression Required. An expression that returns one of the objects in the Applies To list.
Remarks

This property corresponds to the **RemoveFileProperties** property that is available on a data access page's property sheet.
Example

This example sets Microsoft Access to remove personal information from the current project the next time the user saves it.

`CurrentProject.RemovePersonalInformation = True`

This example sets Microsoft Access to remove personal information from the active data access page the next time the user saves it.

`Screen.ActiveDataAccessPage .RemovePersonalInformation = True`
RepeatSection Property

You can use the RepeatSection property to specify whether a group header is repeated on the next page or column when a group spans more than one page or column. Read/write Boolean.

expression.RepeatSection

expression Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **RepeatSection** property only applies to group headers on a report.

The **RepeatSection** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>True</td>
<td>The group header is repeated.</td>
</tr>
<tr>
<td>No</td>
<td>False</td>
<td>(Default) The group header isn't repeated.</td>
</tr>
</tbody>
</table>

**Note**  You can set the **RepeatSection** property by using the group header section's **property sheet**, a **macro**, or **Visual Basic**.

When printing a report that contains a subreport, the subreport's **RepeatSection** property will determine if the subreport group headers are repeated across pages or columns.
**Example**

The following example prints the group header "GroupHeader0" at the top of each page.

```vbnet
Reports("Purchase Order").Section("GroupHeader0").RepeatSection = True
```
Report Property

You can use the **Report** property to refer to a **report** or to refer to the report associated with a **subreport** control.

`expression.Report`

*expression* Required. An expression that returns one of the objects in the Applies To list.
Remarks

This property refers to a report object. It is read-only in all views.

You can use this property by using a macro or Visual Basic.

This property is typically used to refer to the report contained in a subreport control.

**Note** When you use the Reports collection, you must specify the name of the report.
Example

The following example uses the `Report` property to refer to a control on a subreport.

```vba
Dim curTotalSales As Currency
curTotalSales = Reports!Sales!Employees.Report!TotalSales
```
Reports Property

You can use the Reports property to access the read-only Reports collection and its related properties.

expression.Reports

expression Required. An expression that returns one of the objects in the Applies To list.
Remarks

This property is available only by using Visual Basic and is read-only.

The Reports collection contains all of the currently open reports in a Microsoft Access database (.mdb) or Microsoft Access project (.adp).
ResyncCommand Property

You can use the `ResyncCommand` property to specify or determine the SQL statement or stored procedure that will be used in an updateable snapshot of a table. Read/write `String`.

`expression.ResyncCommand`

`expression` Required. An expression that returns one of the objects in the Applies To list.
Remarks

The ResyncCommand property is a string expression representing a SQL statement or stored procedure that is parameterized by the key columns from the Unique Table in the output cursor, using ? as parameter markers.

You can set this property by using the property sheet or Visual Basic.

The parameters must match in number and ordering to the set of key columns for the table identified by the UniqueTable property. The purpose of the ResyncCommand property is to pull in the "fixed up" values of a row in a recordset after an update has been made, including an update to a join column.

For data access pages and for forms based on views or non-parameterized SQL queries containing a join, if the ResyncCommand property is null, Microsoft Access determines an appropriate query to use for the resync operation. For data access pages and forms based on stored procedures or parameterized SQL statements, Access cannot determine an appropriate resync query at run time, so the user must supply the ResyncCommand string in order to get the correct row fix up behavior. If the ResyncCommand property is empty and Access cannot determine an appropriate query to use, the default ADO resync operation (to display the current values) occurs after an update or insert.
RightMargin Property

RightMargin property as it applies to the Label and TextBox objects.

Along with the TopMargin, Left Margin, and BottomMargin properties, specifies the location of information displayed within a label or text box control. Read/write Integer. Read/write Integer.

expression.RightMargin

expression Required. An expression that returns one of the above objects.
Remarks

A control's displayed information location is measured from the control's left, top, right, or bottom border to the left, top, right, or bottom edge of the displayed information. To use a unit of measurement different from the setting in the regional settings of Windows, specify the unit (for example, cm or in).

In Visual Basic, use a numeric expression to set the value of this property. Values are expressed in **twips**.

You can set these properties by using the property sheet, a macro, or Visual Basic.

**RightMargin** property as it applies to the **Printer** object.

Along with the **TopMargin**, **LeftMargin**, and **BottomMargin** properties, specifies the margins for a printed page. Read/write **Long**.

*expression*.**RightMargin**

*expression*  Required. An expression that returns a **Printer** object.
Example

As it applies to the Label and TextBox objects.

The following example offsets the caption in the label "EmployeeID_Label" in the "Purchase Orders" form by 100 twips from the right of the label's border.

With Forms.Item("Purchase Orders").Controls.Item("EmployeeID_Label")
  .RightMargin = 100
End With
RowHeight Property

You can use the RowHeight property to specify the height of all rows in Datasheet view. Read/write Integer.

expression.RowHeight

expression  Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **RowHeight** property applies to all fields in Datasheet view and to form controls when the form is in Datasheet view.

You can set the **RowHeight** property in Datasheet view by dragging the lower border of the **record selector** or by clicking **Row Height** on the **Format** menu. When you set the **RowHeight** property by using the **Row Height** command, the value is expressed in **points**.

In **Visual Basic**, the **RowHeight** property setting is a **Long Integer** value that represents the datasheet row height in **twips**. To specify the default height for the current font, you can set the **RowHeight** property to **True**.
Example

This example takes effect in Datasheet view of the open Customers form. It sets the row height to 450 twips and sizes the column to fit the size of the visible text.

Forms![Customers].RowHeight = 450
Forms![Customers]![Address].ColumnWidth = -2
RowSource Property

You can use the `RowSource` property (along with the `RowSourceType` property) to tell Microsoft Access how to provide data to a list box, a combo box, or an unbound OLE object such as a chart. For example, to display rows of data in a list box from a query named CustomerList, set the list box's `RowSourceType` property to Table/Query and its `RowSource` property to the query named CustomerList. Read/write `String`.

`expression.RowSource`

`expression` Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **RowSource** property setting depends on the **RowSourceType** property setting.

<table>
<thead>
<tr>
<th>For this <strong>RowSourceType</strong> setting</th>
<th>Enter this <strong>RowSource</strong> setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table/Query</td>
<td>A table name, query name, or SQL statement.</td>
</tr>
<tr>
<td>Value List</td>
<td>A list of items with semicolons (;) as separators.</td>
</tr>
<tr>
<td>Field List</td>
<td>A table name, query name, or SQL statement.</td>
</tr>
</tbody>
</table>

**Note** If the **RowSourceType** property is set to a user-defined function, the **RowSource** property can be left blank.

You can set the **RowSource** property by using the control’s **property sheet**, a **macro**, or **Visual Basic**.

For table fields, you can set these properties on the **Lookup** tab in the Field Properties section of **table Design view** for fields with the **DisplayControl** property set to Combo Box or List Box.

**Note** Microsoft Access sets these properties automatically when you select Lookup Wizard as the data type for a field in table Design view.

In Visual Basic, set the **RowSourceType** property by using a **string expression** with one of these values: "Table/Query", "Value List", or "Field List". You also use a string expression to set the value of the **RowSource** property. To set the **RowSourceType** property to a user-defined function, enter the name of the function.

When you have a limited number of values that don't change, you can set the **RowSourceType** property to Value List and then enter the values that make up the list in the **RowSource** property.
Example

The following example sets the \texttt{RowSourceType} property for a combo box to Table/Query, and it sets the \texttt{RowSource} property to a query named EmployeeList.

\verbatim{Forms!Employees!cmboNames.RowSourceType = "Table/Query"
Forms!Employees!cmboNames.RowSource = "EmployeeList"}
RowSourceType Property

You can use the RowSourceType property (along with the RowSource property) to tell Microsoft Access how to provide data to a list box, a combo box, or an unbound OLE object such as a chart. For example, to display rows of data in a list box from a query named CustomerList, set the list box's RowSourceType property to Table/Query and its RowSource property to the query named CustomerList. Read/write String.

expression.RowSourceType

expression Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **RowSourceType** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table/Query</td>
<td>(Default) The data is from a table, query, or SQL statement specified by the <strong>RowSource</strong> setting.</td>
</tr>
<tr>
<td>Value List</td>
<td>The data is a list of items specified by the <strong>RowSource</strong> setting.</td>
</tr>
<tr>
<td>Field List</td>
<td>The data is a list of field names from a table, query, or SQL statement specified by the <strong>RowSource</strong> setting.</td>
</tr>
</tbody>
</table>

**Note** You can also set the **RowSourceType** property with a **user-defined function**. The function name is entered without a preceding equal sign (=) and without the trailing pair of parentheses. You must provide **specific function code arguments** to tell Microsoft Access how to fill the control.

You can set the **RowSourceType** property by using the control's **property sheet**, a **macro**, or **Visual Basic**. In Visual Basic, set the **RowSourceType** property by using a **string expression** with one of these values: "Table/Query", "Value List", or "Field List". To set the **RowSourceType** property to a user-defined function, enter the name of the function.

When you have a limited number of values that don't change, you can set the **RowSourceType** property to Value List and then enter the values that make up the list in the **RowSource** property.

When you create a user-defined function to insert items into a list box or combo box, Microsoft Access calls the function repeatedly to get the information it needs. User-defined **RowSourceType** functions must be written in a very specific **function format**.
Example

The following example sets the RowSourceType property for a combo box to Table/Query, and it sets the RowSource property to a query named EmployeeList.

Forms!Employees!cmboNames.RowSourceType = "Table/Query"
Forms!Employees!cmboNames.RowSource = "EmployeeList"
RowSpacing Property

Returns or sets a Long indicating the horizontal space between detail sections in twips. Read/write.

expression.RowSpacing

expression  Required. An expression that returns one of the objects in the Applies To list.
Example

The following example sets a variety of printer settings for the form specified in the **strFormname** argument of the procedure.

Sub SetPrinter(strFormname As String)

    DoCmd.OpenForm FormName:=strFormname, view:=acDesign, _
    datamode:=acFormEdit, windowmode:=acHidden

    With Forms(form1).Printer

        .TopMargin = 1440
        .BottomMargin = 1440
        .LeftMargin = 1440
        .RightMargin = 1440

        .ColumnSpacing = 360
        .RowSpacing = 360

        .ColorMode = acPRCMColor
        .DataOnly = False
        .DefaultSize = False
        .ItemSizeHeight = 2880
        .ItemSizeWidth = 2880
        .ItemLayout = acPRVerticalColumnLayout
        .ItemsAcross = 6

        .Copies = 1
        .Orientation = acPRORLandscape
        .Duplex = acPRDPVertical
        .PaperBin = acPRBNAuto
        .PaperSize = acPRPSLetter
        .PrintQuality = acPRPQMedium

    End With

    DoCmd.Close objecttype:=acForm, objectname:=strFormname, _
    Save:=acSaveYes

End Sub
RunningSum Property

You can use the RunningSum property to calculate record-by-record or group-by-group totals in a report. Read/write.

expression.RunningSum

expression Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **RunningSum** property specifies whether a text box on a report displays a running total and lets you set the range over which values are accumulated. For example, you can group data by month and show the sum of each month's sales in the group footer. You can show the running sum of accumulated sales over the entire report (sales for January in the January footer, sales for January plus February in the February footer, and so on) by adding a text box to the footer that shows the sum of sales and setting its **RunningSum** property to Over All.

**Note**  The **RunningSum** property applies only to a text box on a report.

The **RunningSum** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>0</td>
<td>(Default) The text box displays the data from the underlying field for the current record.</td>
</tr>
<tr>
<td>Over Group</td>
<td>1</td>
<td>The text box displays a running sum of values in the same group level. The value accumulates until another group level section is encountered.</td>
</tr>
<tr>
<td>Over All</td>
<td>2</td>
<td>The text box displays a running sum of values in the same group level. The value accumulates until the end of the report.</td>
</tr>
</tbody>
</table>

**Note** You can set the **RunningSum** property by using the text box's property sheet, a macro, or Visual Basic. You can set the **RunningSum** property only in **Design view**.

Place the text box in the Detail section to calculate a record-by-record total. For example, to number the records appearing in a detail section of a report, set the **ControlSource** property for the text box to ",=1", and set the **RunningSum** property to Over Group.

Place the text box in a group header or group footer to calculate a group-by-group total.

You can have up to 10 nested group levels in a report.
Example

The following example sets the **RunningSum** property for a text box named SalesTotal to 2 (Over All):

```
Reports!rptSales!SalesTotal. RunningSum = 2
```
Show All
ScaleHeight Property

You can use the ScaleHeight property to specify the number of units for the vertical measurement of the page when the Circle, Line, Pset, or Print method is used while a report is printed or previewed, or its output is saved to a file. Read/write Single.

expression.ScaleHeight

expression Required. An expression that returns one of the objects in the Applies To list.
Remarks

The default setting is the internal height of a report page in twips.

You can set the ScaleHeight property by using a macro or a Visual Basic event procedure specified by a section's OnPrint property setting.

You can use the ScaleHeight property to create a custom coordinate scale for drawing or printing. For example, the statement ScaleHeight = 100 defines the internal height of the section as 100 units, or one vertical unit as one one-hundredth of the height.

Use the ScaleMode property to define a scale based on a standard unit of measurement, such as points, pixels, characters, inches, millimeters, or centimeters.

Setting the ScaleHeight property to a positive value makes coordinates increase in value from top to bottom. Setting it to a negative value makes coordinates increase in value from bottom to top.

By using these properties and the related ScaleLeft and ScaleTop properties, you can set up a custom coordinate system with both positive and negative coordinates. All four of these Scale properties interact with the ScaleMode property in the following ways:

- Setting any other Scale property to any value automatically sets the ScaleMode property to 0.
- Setting the ScaleMode property to a number greater than 0 changes the ScaleHeight and ScaleWidth properties to the new unit of measurement and sets the ScaleLeft and ScaleTop properties to 0. Also, the CurrentX and CurrentY property settings change to reflect the new coordinates of the current point.

You can also use the Scale method to set the ScaleHeight, ScaleWidth, ScaleLeft, and ScaleTop properties in one statement.

*Note* The ScaleHeight property isn't the same as the Height property.
Example

The following example uses the Print method to display text on a report named Report1. It uses the TextWidth and TextHeight methods to center the text vertically and horizontally.

Private Sub Detail_Format(Cancel As Integer, _
    FormatCount As Integer)
    Dim rpt As Report
    Dim strMessage As String
    Dim intHorSize As Integer, intVerSize As Integer

    Set rpt = Me
    strMessage = "DisplayMessage"
    With rpt
        ' Set scale to pixels, and set FontName and
        ' FontSize properties.
        .ScaleMode = 3
        .FontName = "Courier"
        .FontSize = 24
    End With
    ' Horizontal width.
    intHorSize = Rpt.TextWidth(strMessage)
    ' Vertical height.
    intVerSize = Rpt.TextHeight(strMessage)
    ' Calculate location of text to be displayed.
    Rpt.CurrentX = (Rpt.ScaleWidth/2) - (intHorSize/2)
    Rpt.CurrentY = (Rpt.ScaleHeight/2) - (intVerSize/2)
    ' Print text on Report object.
    Rpt.Print strMessage
End Sub
ScaleLeft Property

You can use the `ScaleLeft` property to specify the units for the horizontal coordinates that describe the location of the left edge of a page when the `Circle`, `Line`, `Pset`, or `Print` method is used while a report is previewed, printed, or its output is saved to a file. Read / write `Single`.

`expression.ScaleLeft`

`expression`  Required. An expression that returns one of the objects in the Applies To list.
Remarks

You can set the `ScaleLeft` property by using a `macro` or a `Visual Basic` event procedure specified by a section's `OnPrint` property setting.

By using these properties and the related `ScaleHeight` and `ScaleWidth` properties, you can set up a custom coordinate system with both positive and negative coordinates. All four of these Scale properties interact with the `ScaleMode` property in the following ways:

- Setting any other Scale property to any value automatically sets the `ScaleMode` property to 0.

- Setting the `ScaleMode` property to a number greater than 0 changes the `ScaleHeight` and `ScaleWidth` property settings to the new unit of measurement and sets the `ScaleLeft` and `ScaleTop` properties to 0. Also, the `CurrentX` and `CurrentY` property settings change to reflect the new coordinates of the current point.

You can also use the `Scale` method to set the `ScaleHeight`, `ScaleWidth`, `ScaleLeft`, and `ScaleTop` properties in one statement.

**Note** The `ScaleLeft` property isn't the same as the `Left` property.
Example

The following example uses the Circle method to draw a circle and create a pie slice within the circle. Then it uses the FillColor and FillStyle properties to color the pie slice red. It also draws a line from the upper left to the center of the circle.

To try this example in Microsoft Access, create a new report. Set the OnPrint property of the Detail section to [Event Procedure]. Enter the following code in the report's module, then switch to Print Preview.

Private Sub Detail_Print(Cancel As Integer, PrintCount As Integer)

Const conPI = 3.14159265359

Dim sngHCtr As Single
Dim sngVCtr As Single
Dim sngRadius As Single
Dim sngStart As Single
Dim sngEnd As Single

sngHCtr = Me.ScaleWidth / 2 ' Horizontal center.
sngVCtr = Me.ScaleHeight / 2 ' Vertical center.
sngRadius = Me.ScaleHeight / 3 ' Circle radius.
Me.Circle (sngHCtr, sngVCtr), sngRadius ' Draw circle.
sngStart = -0.000000001 ' Start of pie slice.

sngEnd = -2 * conPI / 3 ' End of pie slice.
Me.FillColor = RGB(255, 0, 0) ' Color pie slice red.
Me.FillStyle = 0 ' Fill pie slice.

' Draw Pie slice within circle
Me.Circle (sngHCtr, sngVCtr), sngRadius, , sngStart, sngEnd

' Draw line to center of circle.
Dim intColor As Integer
Dim sngTop As Single, sngLeft As Single
Dim sngWidth As Single, sngHeight As Single

Me.ScaleMode = 3 ' Set scale to pixels.
sngTop = Me.ScaleTop ' Top inside edge.
sngLeft = Me.ScaleLeft ' Left inside edge.
sngWidth = Me.ScaleWidth / 2 ' Width inside edge.
sngHeight = Me.ScaleHeight / 2 ' Height inside edge.
intColor = RGB(255, 0, 0) ' Make color red.
' Draw line.
Me.Line (sngTop, sngLeft)-(sngWidth, sngHeight), intColor

End Sub
ScaleMode Property

You can use the **ScaleMode** property in Visual Basic to specify the unit of measurement for coordinates on a page when the **Circle**, **Line**, **Pset**, or **Print** method is used while a report is previewed or printed, or its output is saved to a file. Read/write **Integer**.

`expression.ScaleMode`  

*expression* Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **ScaleMode** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Custom values used by one or more of the <strong>ScaleHeight</strong>, <strong>ScaleWidth</strong>, <strong>ScaleLeft</strong>, and <strong>ScaleTop</strong> properties</td>
</tr>
<tr>
<td>1</td>
<td>(Default) <strong>Twips</strong></td>
</tr>
<tr>
<td>2</td>
<td><strong>Points</strong></td>
</tr>
<tr>
<td>3</td>
<td><strong>Pixels</strong></td>
</tr>
<tr>
<td>4</td>
<td>Characters (horizontal = 120 twips per unit; vertical = 240 twips per unit)</td>
</tr>
<tr>
<td>5</td>
<td><strong>Inches</strong></td>
</tr>
<tr>
<td>6</td>
<td><strong>Millimeters</strong></td>
</tr>
<tr>
<td>7</td>
<td><strong>Centimeters</strong></td>
</tr>
</tbody>
</table>

The property setting has an **Integer** value.

You can set the **ScaleMode** property by using a **macro** or a **Visual Basic** event procedure specified by a section's **OnPrint** property setting.

By using the related **ScaleHeight**, **ScaleWidth**, **ScaleLeft**, and **ScaleTop** properties, you can create a custom coordinate system with both positive and negative coordinates. All four properties interact with the **ScaleMode** property in the following ways:

- Setting any other Scale property to any value automatically sets the **ScaleMode** property to 0.

- Setting the **ScaleMode** property to a number greater than 0 changes the **ScaleHeight** and **ScaleWidth** property settings to the new unit of measurement and sets the **ScaleLeft** and **ScaleTop** properties to 0. Also, the **CurrentX** and **CurrentY** property settings change to reflect the new coordinates of the current point.
ScaleTop Property

You can use the ScaleTop property to specify the units for the vertical coordinates that describe the location of the top edge of a page when the Circle, Line, Pset, or Print method is used while a report is previewed, printed, or its output is saved to a file. Read / write Single.

expression.ScaleTop

expression Required. An expression that returns one of the objects in the Applies To list.
Remarks

You can set the ScaleTop property by using a macro or a Visual Basic event procedure specified by a section's OnPrint property setting.

By using these properties and the related ScaleHeight and ScaleWidth properties, you can set up a custom coordinate system with both positive and negative coordinates. All four of these Scale properties interact with the ScaleMode property in the following ways:

- Setting any other Scale property to any value automatically sets the ScaleMode property to 0.

- Setting the ScaleMode property to a number greater than 0 changes the ScaleHeight and ScaleWidth property settings to the new unit of measurement and sets the ScaleLeft and ScaleTop properties to 0. Also, the CurrentX and CurrentY property settings change to reflect the new coordinates of the current point.

You can also use the Scale method to set the ScaleHeight, ScaleWidth, ScaleLeft, and ScaleTop properties in one statement.

Note   The ScaleTop property isn't the same as the Top property.
**Example**

The following example uses the `Circle` method to draw a circle and create a pie slice within the circle. Then it uses the `FillColor` and `FillStyle` properties to color the pie slice red. It also draws a line from the upper left to the center of the circle.

To try this example in Microsoft Access, create a new report. Set the `OnPrint` property of the Detail section to `[Event Procedure]`. Enter the following code in the report's module, then switch to Print Preview.

```vba
Private Sub Detail_Print(Cancel As Integer, PrintCount As Integer)
  Const conPI = 3.14159265359
  Dim sngHCtr As Single
  Dim sngVCtr As Single
  Dim sngRadius As Single
  Dim sngStart As Single
  Dim sngEnd As Single

  sngHCtr = Me.ScaleWidth / 2 ' Horizontal center.
  sngVCtr = Me.ScaleHeight / 2 ' Vertical center.
  sngRadius = Me.ScaleHeight / 3 ' Circle radius.
  Me.Circle (sngHCtr, sngVCtr), sngRadius ' Draw circle.
  sngStart = -0.000000001 ' Start of pie slice.
  sngEnd = -2 * conPI / 3 ' End of pie slice.
  Me.FillColor = RGB(255, 0, 0) ' Color pie slice red.
  Me.FillStyle = 0 ' Fill pie slice.

  ' Draw Pie slice within circle
  Me.Circle (sngHCtr, sngVCtr), sngRadius, , sngStart, sngEnd

  ' Draw line to center of circle.
  Dim intColor As Integer
  Dim sngTop As Single, sngLeft As Single
  Dim sngWidth As Single, sngHeight As Single

  Me.ScaleMode = 3 ' Set scale to pixels.
  sngTop = Me.ScaleTop ' Top inside edge.
  sngLeft = Me.ScaleLeft ' Left inside edge.
  sngWidth = Me.ScaleWidth / 2 ' Width inside edge.
  sngHeight = Me.ScaleHeight / 2 ' Height inside edge.
  intColor = RGB(255, 0, 0) ' Make color red.

  Dim sngX As Single, sngY As Single
  sngX = sngHCtr + sngRadius
  sngY = sngVCtr
  Me.Line (sngX, sngY), sngHCtr ' Draw line.
```

' Draw line from upper left to center of circle.
Dim sngX As Single, sngY As Single
sngX = sngHCtr + sngRadius
sngY = sngVCtr
Me.Line (sngX, sngY), sngHCtr ' Draw line.
```
' Draw line.
Me.Line (sngTop, sngLeft)-(sngWidth, sngHeight), intColor

End Sub
ScaleWidth Property

You can use the ScaleWidth property to specify the number of units for the horizontal measurement of the page when the Circle, Line, Pset, or Print method is used while a report is printed or previewed, or its output is saved to a file. Read/write Single.

expression.ScaleWidth

expression  Required. An expression that returns one of the objects in the Applies To list.
Remarks

The default setting is the internal width of a report page in twips.

You can set the ScaleWidth property by using a macro or a Visual Basic event procedure specified by a section's OnPrint property setting.

You can use the ScaleWidth property to create a custom coordinate scale for drawing or printing. For example, the statement ScaleWidth = 100 defines the internal width of the section as 100 units, or one horizontal unit as one one-hundredth of the width.

Use the ScaleMode property to define a scale based on a standard unit of measurement, such as points, pixels, characters, inches, millimeters, or centimeters.

Setting the ScaleWidth property to a positive value makes coordinates increase in value from left to right. Setting it to a negative value makes coordinates increase in value from right to left.

By using these properties and the related ScaleLeft and ScaleTop properties, you can set up a custom coordinate system with both positive and negative coordinates. All four of these Scale properties interact with the ScaleMode property in the following ways:

- Setting any other Scale property to any value automatically sets the ScaleMode property to 0.
- Setting the ScaleMode property to a number greater than 0 changes the ScaleHeight and ScaleWidth properties to the new unit of measurement and sets the ScaleLeft and ScaleTop properties to 0. Also, the CurrentX and CurrentY property settings change to reflect the new coordinates of the current point.

You can also use the Scale method to set the ScaleHeight, ScaleWidth, ScaleLeft, and ScaleTop properties in one statement.

Note  The ScaleWidth properties isn't the same as the Width property.
Example

The following example uses the Print method to display text on a report named Report1. It uses the TextWidth and TextHeight methods to center the text vertically and horizontally.

```
Private Sub Detail_Format(Cancel As Integer, _
    FormatCount As Integer)
    Dim rpt as Report
    Dim strMessage As String
    Dim intHorSize As Integer, intVerSize As Integer

    Set rpt = Me
    strMessage = "DisplayMessage"
    With rpt
        ' Set scale to pixels, and set FontName and
        ' FontSize properties.
        .ScaleMode = 3
        .FontName = "Courier"
        .FontSize = 24
    End With
    ' Horizontal width.
    intHorSize = Rpt.TextWidth(strMessage)
    ' Vertical height.
    intVerSize = Rpt.TextHeight(strMessage)
    ' Calculate location of text to be displayed.
    Rpt.CurrentX = (Rpt.ScaleWidth/2) - (intHorSize/2)
    Rpt.CurrentY = (Rpt.ScaleHeight/2) - (intVerSize/2)
    ' Print text on Report object.
    Rpt.Print strMessage

End Sub
```
Scaling Property

Controls how the contents of an object frame control are displayed. Read/write Byte.

expression.Scaling

expression   Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **Scaling** property corresponds to the **Size Mode** box in the object frame's **Properties** window. This property accepts the following values.

- **0 (Clip)** If the object exceeds the control's boundaries, the object is clipped at the boundaries of the control.
- **1 (Stretch)** If the object does not exceed the control's boundaries, the object is stretched to the edges of the control's boundary.
- **2 (Zoom)** The object is zoomed in or out to fit the control's boundaries. This is different from the Stretch setting, in that the object is not necessarily distorted to touch all boundaries of the control. In other words, the object may touch the horizontal edges of the control, but not necessarily the vertical edges of the control, and vice versa.
Example

The following example sets the size mode of the OLE control "Customer Picture" on the "Order Entry" form to zoomed.

Forms("Order Entry").Controls("Customer Picture").Scaling = 2
Screen Property

You can use the **Screen** property to return a reference the **Screen** object and its related properties. Read-only.

`expression.Screen`

`expression`  Required. An expression that returns one of the objects in the Applies To list.
Remarks

This property is available only by using Visual Basic and is read-only. Use the Screen object to refer to a particular form, report, or control that has the focus.
Example

The following example demonstrates how to change the cursor to an hourglass and back again to signify that some background activity is occurring.

Application.Screen.MousePointer = 11 ' Hourglass
' Do some background activity.
Application.Screen.MousePointer = 0 ' Back to normal
ScreenTip Property

You can use the ScreenTip property to specify or determine the text that is displayed when you move the cursor over a hyperlink control. Read/write String.

expression.ScreenTip

.expression Required. An expression that returns one of the objects in the Applies To list.
Remarks

The ScreenTip property is a string expression displayed as a screen tip.

You can set the HyperlinkSubAddress property with the SubAddress property by using Visual Basic.

Note  You can set the HyperlinkSubAddress property by using a control’s property sheet, a macro, or Visual Basic.

When you move the cursor over a hyperlink control whose HyperlinkSubAddress property is set, Microsoft Access changes the cursor to an upward-pointing hand and displays the text string defined by the ScreenTip property. Clicking the control displays the object or Web page specified by the link.

For more information about hyperlink addresses and their format, see the HyperlinkAddress and HyperlinkSubAddress property topics.
Example

The following example displays the message "Go to Home page" when the cursor hovers over the hyperlink named "HomePage" on the "Order Entry" form.

Forms("Order Entry").Controls("HomePage").Hyperlink.ScreenTip = "Go
ScrollBarAlign Property

You can use the **ScrollBarAlign** to specify or determine the alignment of a vertical scroll bar. Read/write **Byte**.

*expression*.ScrollBarAlign

*expression*    Required. An expression that returns one of the objects in the Applies To list.
**Remarks**

The `ScrollBarAlign` property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>System 0</td>
<td>Orientation</td>
<td>Vertical scroll bar is placed on the left if the form or report Orientation property is right to left; and on the right if the form or report Orientation property is left to right.</td>
</tr>
<tr>
<td>Right</td>
<td>1</td>
<td>Aligns vertical scroll bar on the right side of the control.</td>
</tr>
<tr>
<td>Left</td>
<td>2</td>
<td>Aligns vertical scroll bar on the left side of the control.</td>
</tr>
</tbody>
</table>

**Note** You can set this property by using the [property sheet](#) or [Visual Basic](#).

**Remarks**

For combo and list boxes, `ScrollBarAlign` also controls the placement of the box button above the scroll bar.
Example

The following example aligns the vertical scroll bar on the left side of the "Country" combo box in the "International Shipping" form.

Forms("International Shipping").Controls("Country").ScrollBarAlign =
**ScrollBars Property**

You can use the **ScrollBars** property to specify whether scroll bars appear on a **form** or in a **text box** control. Read/write **Byte**.

**expression.ScrollBars**

**expression** Required. An expression that returns one of the objects in the Applies To list.

**Setting**

The **ScrollBars** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Neither (forms)</td>
<td>0</td>
<td>(Default for text boxes) No scroll bars appear on the form or text box.</td>
</tr>
<tr>
<td>None (text boxes)</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Horizontal Only (forms)</td>
<td>1</td>
<td>Horizontal scroll bar appears on the form. Not applicable to text boxes.</td>
</tr>
<tr>
<td>Vertical Only (forms)</td>
<td>2</td>
<td>Vertical scroll bar appears on the form or text box.</td>
</tr>
<tr>
<td>Vertical (text boxes)</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Both (forms)</td>
<td>3</td>
<td>(Default for forms) Vertical and horizontal scroll bars appear on the form.</td>
</tr>
<tr>
<td>Vertical (text boxes)</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

You can set this property by using the form's or control's property sheet, a macro, or **Visual Basic**.

For a text box, you can set the default for this property by using the default control style or the **DefaultControl** method in Visual Basic.
**Remarks**

If your form is larger than the available display window, you can use the `ScrollBars` property to allow the user to view the entire form.

You can use the `NavigationButtons` property to control record navigation.
Section Property

Section property as it applies to controls on a form or report.

You can identify these controls by the section of a form or report where the control appears. Read/write Integer.

expression.Section

expression  Required. An expression that returns one of the above objects.
Remarks

For controls, you can use the **Section** property to determine which section of a form or report a control is in.

**Section property as it applies to the Form and Report objects.**

You can use the **Section** property to identify a **section** of a **form** or **report** and provide access to the properties of that section. Read-only **Section** object.

```vba
expression.Section(Index)
```

*expression* Required. An expression that returns one of the above objects.

*Index* Required **Variant**. The section number or name.
Remarks

The **Section** property corresponds to a particular section. You can use the following constants listed below. It is recommended that you use the constants to make your code easier to read.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>acDetail</td>
<td>Form detail section or report detail section</td>
</tr>
<tr>
<td>1</td>
<td>acHeader</td>
<td>Form or report header section</td>
</tr>
<tr>
<td>2</td>
<td>acFooter</td>
<td>Form or report footer section</td>
</tr>
<tr>
<td>3</td>
<td>acPageHeader</td>
<td>Form or report page header section</td>
</tr>
<tr>
<td>4</td>
<td>acPageFooter</td>
<td>Form or report page footer section</td>
</tr>
<tr>
<td>5</td>
<td>acGroupLevel1Header</td>
<td>Group-level 1 header section (reports only)</td>
</tr>
<tr>
<td>6</td>
<td>acGroupLevel1Footer</td>
<td>Group-level 1 footer section (reports only)</td>
</tr>
<tr>
<td>7</td>
<td>acGroupLevel2Header</td>
<td>Group-level 2 header section (reports only)</td>
</tr>
<tr>
<td>8</td>
<td>acGroupLevel2Footer</td>
<td>Group-level 2 footer section (reports only)</td>
</tr>
</tbody>
</table>

If a report has additional group-level sections, the header/footer pairs are numbered consecutively beginning with 9.

For forms and reports, the **Section** property is an array of all existing sections in the form or report specified by the section number. For example, `Section(0)` refers to a form's detail section and `Section(3)` refers to a form's page header section.

You can also refer to a section by name. The following statements refer to the Detail0 section for the Customers form and are equivalent.

Forms!Customers.Section(acDetail).Visible

Forms!Customers.Section(0).Visible

Forms!Customers.Detail0.Visible

For forms and reports, you must combine the **Section** property with other properties that apply to form or report sections.
**Example**

As it applies to controls on a form or report.

The following example uses the `Section` property to determine which section contains the `CustomerID` control.

```vba
Dim intSectionNumber As Integer
intSectionNumber = Forms!Customers!CustomerID.Section
```

As it applies to the Form and Report objects.

The following example shows how to refer to the `Visible` property of the page header section of the Customers form.

```
Forms!Customers.Section(acPageHeader).Visible
```

```
Forms!Customers.Section(3).Visible
```
Selected Property

You can use the **Selected** property in Visual Basic to determine if an item in a **list box** is selected. Read/write **Long**.

\[expression.Selected(IRow)\]

- **expression**  Required. An expression that returns one of the objects in the Applies To list.

- **IRow**  Required **Long**. The item in the list box. The first item is represented by a zero (0), the second by a one (1), and so on.
Remarks

The **Selected** property is a zero-based array that contains the selected state of each item in a list box.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>The list box item is selected.</td>
</tr>
<tr>
<td>False</td>
<td>The list box item isn't selected.</td>
</tr>
</tbody>
</table>

**Note**  You can get or set the **Selected** property by using **Visual Basic**.

This property is available only at run time.

When a list box control's **MultiSelect** property is set to None, only one item can have its **Selected** property set to **True**. When a list box control's **MultiSelect** property is set to Simple or Extended, any or all of the items can have their **Selected** property set to **True**. A multiple-selection list box bound to a field will always have a **Value** property equal to **Null**. You use the **Selected** property or the **ItemsSelected** collection to retrieve information about which items are selected.

You can use the **Selected** property to select items in a list box by using Visual Basic. For example, the following expression selects the fifth item in the list:

```vbnet
Me!Listbox.Selected(4) = True
```
Example

The following example uses the Selected property to move selected items in the lstSource list box to the lstDestination list box. The lstDestination list box's RowSourceType property is set to Value List and the control's RowSource property is constructed from all the selected items in the lstSource control. The lstSource list box's MultiSelect property is set to Extended. The CopySelected() procedure is called from the cmdCopyItem command button.

Private Sub cmdCopyItem_Click()
    CopySelected Me
End Sub

Public Sub CopySelected(ByRef frm As Form)

    Dim ctlSource As Control
    Dim ctlDest As Control
    Dim strItems As String
    Dim intCurrentRow As Integer

    Set ctlSource = frm!lstSource
    Set ctlDest = frm!lstDestination

    For intCurrentRow = 0 To ctlSource.ListCount - 1
        If ctlSource.Selected(intCurrentRow) Then
            strItems = strItems & ctlSource.Column(0, _
                                               intCurrentRow) & ";"
        End If
    Next intCurrentRow

    ' Reset destination control's RowSource property.
    ctlDest.RowSource = ""
    ctlDest.RowSource = strItems

    Set ctlSource = Nothing
    Set ctlDest = Nothing

End Sub
SelectionChange Property

Returns or sets a String indicating which macro, event procedure, or user-defined function runs when the SelectionChange event occurs. Read/write.

expression.SelectionChange

expression Required. An expression that returns one of the objects in the Applies To list.
**Remarks**

Valid values for this property are "*macroname*" where *macroname* is the name of macro, "*[Event Procedure]*" which indicates the event procedure associated with the SelectionChange event for the specified object, or "*=functionname()" where *functionname* is the name of a user-defined function. For a more detailed discussion of event properties, see "[Event Properties](#)."
Example

The following example specifies that when the SelectionChange event occurs on the first form of the current project, the associated event procedure should run.

\texttt{Forms(0).SelectionChange = "[Event Procedure]"}
SelHeight Property

You can use the SelHeight property to specify or determine the number of selected rows (records) in the current selection rectangle in a table, query, or form datasheet, or the number of selected records in a continuous form. The SelHeight property returns a Long Integer value between 0 and the number of records in the datasheet or continuous form. The setting of this property specifies or returns the number of selected rows in the selection rectangle or the number of selected records in the continuous form.

This property isn't available in Design view. This property is only available by using a macro or Visual Basic.
Remarks

If there's no selection, the value returned by this property will be zero. Setting this property to 0 removes the selection from the datasheet or form.

If you've selected one or more columns in a datasheet (using the column headings), you can't change the setting of the SelHeight property (except to set it to 0).

You can use these properties with the SelTop and SelLeft properties to specify or determine the actual position of the selection rectangle on the datasheet. If there's no selection, then the SelTop and SelLeft properties return the row number and column number of the cell with the focus.

The SelHeight and SelWidth properties contain the position of the lower-right corner of the selection rectangle. The SelTop and SelLeft property values determine the upper-left corner of the selection rectangle.
SelLeft Property

You can use the SelLeft property to specify or determine which column (field) is leftmost in the current selection rectangle. Read/write Long between 1 and the number of columns in the datasheet. The setting of this property specifies or returns the number of the leftmost column in the current selection rectangle.

expression.SelLeft

expression Required. An expression that returns one of the objects in the Applies To list.
Remarks

This property isn't available in Design view. This property is available only by using a macro or Visual Basic.

If there's no selection, the value returned by these properties is the row and column of the cell with the focus. If you've selected one or more records in the datasheet (using the record selectors), you can't change the setting of the SelLeft property.

You can use these properties with the SelHeight and SelWidth properties to specify or determine the actual size of the selection rectangle. The SelTop and SelLeft properties determine the position of the upper-left corner of the selection rectangle. The SelHeight and SelWidth properties determine the lower-right corner of the selection rectangle.
Example

The following example shows how to use the `SelHeight`, `SelWidth`, `SelTop`, and `SelLeft` properties to determine the position and size of a selection rectangle in datasheet view. The `SetHeightWidth` procedure assigns the height and width of the current selection rectangle to the variables `lngNumRows`, `lngNumColumns`, `lngTopRow`, and `lngLeftColumn`, and displays those values in a message box.

```vba
Public Sub SetHeightWidth(ByRef frm As Form)
    Dim lngNumRows As Long
    Dim lngNumColumns As Long
    Dim lngTopRow As Long
    Dim lngLeftColumn As Long
    Dim strMsg As String

    ' Form is inDatasheet view.
    If frm.CurrentView = 2 Then
        ' Number of rows selected.
        lngNumRows = frm.SelHeight
        strMsg = strMsg & vbCrLf & "Number of rows: " & lngNumRows

        ' Number of columns selected.
        lngNumColumns = frm.SelWidth
        strMsg = strMsg & vbCrLf & "Number of columns: " & lngNumColumns

        ' Topmost row selected.
        lngTopRow = frm.SelTop
        strMsg = strMsg & vbCrLf & "Top row: " & lngTopRow

        ' Leftmost column selected.
        lngLeftColumn = frm.SelLeft
        strMsg = strMsg & vbCrLf & "Left column: " & lngLeftColumn

        MsgBox strMsg, vbInformation
    End If

End Sub
```
SelLength Property

The **SelLength** property specifies or determines the number of characters selected in a text box or the text box portion of a combo box. The **SelLength** property uses an **Integer** in the range 0 to the total number of characters in a text box or text box portion of a combo box.
Remarks

You can set the **SelLength** property by using a [macro](#) or Visual Basic.

To set or return this property for a control, the control must have the [focus](#). To move the focus to a control, use the **SetFocus** method.

Setting the **SelLength** property to a number less than 0 produces a [run-time error](#).
Example

The following example uses two event procedures to search for text specified by a user. The text to search is set in the form's Load event procedure. The Click event procedure for the Find button (which the user clicks to start the search) prompts the user for the text to search for and selects the text in the text box if the search is successful.

Private Sub Form_Load()
    Dim ctlTextToSearch As Control
    Set ctlTextToSearch = Forms!Form1!Textbox1
    ' SetFocus to text box.
    ctlTextToSearch.SetFocus
    ctlTextToSearch.Text = "This company places large orders twice "
    "a year for garlic, oregano, chilies and cumin."
    Set ctlTextToSearch = Nothing
End Sub

Public Sub Find_Click()
    Dim strSearch As String
    Dim intWhere As Integer
    Dim ctlTextToSearch As Control
    ' Get search string from user.
    With Me!Textbox1
        strSearch = InputBox("Enter text to find:")
    End With
    ' Find string in text.
    intWhere = InStr(.Value, strSearch)
    If intWhere Then
        ' If found.
        .SetFocus
        .SelStart = intWhere - 1
        .SelLength = Len(strSearch)
    Else
        ' Notify user.
        MsgBox "String not found."
    End If
End With
End Sub
SelStart Property

The SelStart property specifies or determines the starting point of the selected text or the position of the insertion point if no text is selected. Read/write Integer.

expression.SelStart

expression精密 Required. An expression that returns one of the objects in the Applies To list.
Remarks

The SelStart property uses an Integer in the range 0 to the total number of characters in a text box or text box portion of a combo box. You can set the SelStart property by using a macro or Visual Basic.

To set or return this property for a control, the control must have the focus. To move the focus to a control, use the SetFocus method.

Changing the SelStart property cancels the selection, places an insertion point in the text, and sets the SelLength property to 0.
Example

The following example uses two event procedures to search for text specified by a user. The text to search is set in the form's Load event procedure. The Click event procedure for the Find button (which the user clicks to start the search) prompts the user for the text to search for and selects the text in the text box if the search is successful.

Private Sub Form_Load()
    Dim ctlTextToSearch As Control
    Set ctlTextToSearch = Forms!Form1!Textbox1

    ' SetFocus to text box.
    ctlTextToSearch.SetFocus
    ctlTextToSearch.Text = "This company places large orders twice "
    "a year for garlic, oregano, chilies and cumin."
    Set ctlTextToSearch = Nothing
End Sub

Public Sub Find_Click()
    Dim strSearch As String
    Dim intWhere As Integer
    Dim ctlTextToSearch As Control

    ' Get search string from user.
    With Me!Textbox1
        strSearch = InputBox("Enter text to find:")
    End With

    ' Find string in text.
    intWhere = InStr(.Value, strSearch)
    If intWhere Then
        ' If found.
        .SetFocus
        .SelStart = intWhere - 1
        .SelLength = Len(strSearch)
    Else
        ' Notify user.
        MsgBox "String not found."
    End If
End With
End Sub
SelText Property

The `SelText` property returns a string containing the selected text. If no text is selected, the `SelText` property contains a `Null` value. Read/write `String`.

`expression.SelText`

`expression` Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **SelText** property uses a **string expression** that contains the text selected in the control. If the control contains selected text when this property is set, the selected text is replaced by the new **SelText** setting.

To set or return this property for a control, the control must have the **focus**. To move the focus to a control, use the **SetFocus** method.
Example

The following example uses two event procedures to search for text specified by a user. The text to search is set in the form's Load event procedure. The Click event procedure for the Find button (which the user clicks to start the search) prompts the user for the text to search for and selects the text in the text box if the search is successful.

Sub Form_Load()
    Dim ctlTextToSearch As Control
    Set ctlTextToSearch = Forms!Form1!TextBox1
    ctlTextToSearch.SetFocus ' SetFocus to text box.
    ctlTextToSearch.SelText = "This company places large orders " _ & "twice a year for garlic, oregano, chilies and cumin."
End Sub

Sub Find_Click()
    Dim strSearch As String, intWhere As Integer
    Dim ctlTextToSearch As Control
    ' Get search string from user.
    With Me!Textbox1
        strSearch = InputBox("Enter text to find:")
        ' Find string in text.
        intWhere = InStr(.Value, strSearch)
        If intWhere Then
            ' If found.
            .SetFocus
            .SelStart = intWhere - 1
            .SelLength = Len(strSearch)
        Else
            ' Notify user.
            MsgBox "String not found."
        End If
    End With
End Sub
SelTop Property

You can use the **SelTop** property to specify or determine which row (record) is topmost in the current *selection rectangle* in a table, query, or form *datasheet*, or which selected record is topmost in a *continuous form*. The **SelTop** property returns a *Long Integer* value between 1 and the number of records in the datasheet or continuous form. The setting of this property specifies or returns the number of the topmost row in the current selection rectangle or the number of the topmost selected record in the continuous form.
Remarks

This property isn't available in Design view. This property is available only by using a macro or Visual Basic.

If there's no selection, the value returned by this property is the row and column of the cell with the focus.

If you've selected one or more columns (using the column headings), you can't change the setting of the SelTop property.

You can use these properties with the SelHeight and SelWidth properties to specify or determine the actual size of the selection rectangle. The SelTop and SelLeft properties determine the position of the upper-left corner of the selection rectangle. The SelHeight and SelWidth properties determine the lower-right corner of the selection rectangle.
SelWidth Property

You can use the **SelWidth** property to specify or determine the number of selected columns (fields) in the current selection rectangle. Read/write **Long** between 0 and the number of columns in the datasheet. The setting of this property specifies or returns the number of selected columns in the selection rectangle.

expression.SelWidth

*expression* Required. An expression that returns one of the objects in the Applies To list.
Remarks

This property isn't available in Design view. This property is only available by using a macro or Visual Basic.

If there's no selection, the value returned by this property will be zero. Setting this property to 0 removes the selection from the datasheet or form.

If you've selected one or more records in the datasheet (using the record selectors), you can't change the setting of the SelWidth property (except to set it to 0).

You can use these properties with the SelTop and SelLeft properties to specify or determine the actual position of the selection rectangle on the datasheet. If there's no selection, then the SelTop and SelLeft properties return the row number and column number of the cell with the focus.

The SelHeight and SelWidth properties contain the position of the lower-right corner of the selection rectangle. The SelTop and SelLeft property values determine the upper-left corner of the selection rectangle.
Example

The following example shows how to use the `SelHeight`, `SelWidth`, `SelTop`, and `SelLeft` properties to determine the position and size of a selection rectangle in datasheet view. The `SetHeightWidth` procedure assigns the height and width of the current selection rectangle to the variables `lngNumRows`, `lngNumColumns`, `lngTopRow`, and `lngLeftColumn`, and displays those values in a message box.

```vba
Public Sub SetHeightWidth(ByRef frm As Form)
    Dim lngNumRows As Long
    Dim lngNumColumns As Long
    Dim lngTopRow As Long
    Dim lngLeftColumn As Long
    Dim strMsg As String

    ' Form is in Datasheet view.
    If frm.CurrentView = 2 Then

        ' Number of rows selected.
        lngNumRows = frm.SelHeight

        ' Number of columns selected.
        lngNumColumns = frm.SelWidth

        ' Topmost row selected.
        lngTopRow = frm.SelTop

        ' Leftmost column selected.
        lngLeftColumn = frm.SelLeft

        ' Display message.
        strMsg = "Number of rows: " & lngNumRows & vbCrLf
        strMsg = strMsg & "Number of columns: " & lngNumColumns & vbCrLf
        strMsg = strMsg & "Top row: " & lngTopRow & vbCrLf
        strMsg = strMsg & "Left column: " & lngLeftColumn
        MsgBox strMsg, vbInformation
    End If

End Sub
```
ServerFilter Property

You can use the ServerFilter property to specify a subset of records to be displayed when a server filter is applied to a form or report within a Microsoft Access project (.adp) or a data access page in a Microsoft Access project (.adp) or database (.mdb). Read/write String.

expression.ServerFilter

expression  Required. An expression that returns one of the objects in the Applies To list.
Remarks

The ServerFilter property is a string expression consisting of a WHERE clause without the WHERE keyword. For example, the following Visual Basic code defines and applies a filter to show only customers from the USA:

```visualbasic
Me.ServerFilter = "Country = 'USA'"
Me.Refresh
```

The easiest way to set this property is by using a form or report's property sheet. You can also set this property on a form or report by using Visual Basic.

To set the ServerFilter property, you must first either:

- Set the property value in the form's property sheet.
- Set the property in Visual Basic by typing
  ```visualbasic
  Forms(0).ServerFilter = "fieldname = value"
  ```

Note Setting the ServerFilter property has no effect on the ADO Filter property.

You can use the ServerFilter property to save a filter and apply it at a later time. Filters are saved with the objects in which they are created. They are automatically loaded when the object is opened, but they aren't automatically applied.

To apply a saved filter to a form, you can click Apply Server Filter on the toolbar, click Apply Filter/Sort on the Records menu, or use a macro or Visual Basic to set the ServerFilterByForm property to True.

The Apply Server Filter button indicates the state of the ServerFilter and ServerFilterByForm properties. The button remains disabled until there is a filter to apply. If an existing filter is currently applied, the Apply Server Filter button appears pressed in.

To apply a filter automatically when a form is opened, specify in the OnOpen
event property setting of the form either a macro that uses the ApplyFilter action or an event procedure that uses the `ApplyFilter` method of the `DoCmd` object. In either case, the form opens in the Server Filter By Form window.

You can only remove a server filter by using Visual Basic to set the `ServerFilterByForm` property to `False` or clear all filter criteria in the Server Filter By Form window and then click **Apply Server Filter**.

When the `ServerFilter` property is set in form Design view, Microsoft Access does not attempt to validate the SQL expression. If the SQL expression is invalid, an error occurs when the filter is applied.

**Notes**

- When a new object is created, it inherits the `RecordSource`, `Filter`, `ServerFilter`, `OrderBy`, and `OrderByOn` properties of the table or query it was created from.

- The `ServerFilter` property setting is ignored if the form's record source is a stored procedure.
ServerFilterByForm Property

You can use the ServerFilterByForm property to specify or determine whether a form is opened in the Server Filter By Form window. Read/write Boolean.

expression.ServerFilterByForm

expression Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **ServerFilterByForm** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>True</td>
<td>The form is opened in the Server Filter By Form window.</td>
</tr>
<tr>
<td>No</td>
<td>False (Default)</td>
<td>The form is not opened in the Server Filter By Form window.</td>
</tr>
</tbody>
</table>

*Note* When the **ServerFilterByForm** property is **True**, the **Filter By Form** feature is disabled.

To set the **ServerFilterByForm** property, you must first either:

- Set the property in the form's property sheet.
- Set the property in Visual Basic by typing
  
  ```
  Forms(0).ServerFilterByForm = True
  ```

If the **ServerFilter** property has been set, you can also set this property by clicking **Apply Server Filter** on the **Form View** toolbar or the **Filter/Sort** toolbar.

To apply a saved filter to a form, press the **Apply Server Filter** button, or apply the filter by using a macro or Visual Basic by setting the **ServerFilterByForm** property to **True**.

The **Apply Server Filter** button indicates the state of the **ServerFilter** and **ServerFilterByForm** properties. The button remains disabled until there is a filter to apply. If an existing filter is currently applied, the **Apply Server Filter** button appears pressed in. To apply a filter automatically when a form opened,
specify in the **OnOpen** event property setting of the form either a macro that uses the ApplyFilter action or an event procedure that uses the **ApplyFilter** method of the **DoCmd** object.

You can remove a filter by using Visual Basic to set the **ServerFilterByForm** property to **False** or clear all filter criteria in the Server Filter By Form window and then click **Apply Server Filter**.

**Notes**

- When a new object is created, it inherits the **RecordSource**, **Filter**, **ServerFilter**, **OrderBy**, and **OrderByOn** properties of the table or query it was created from. For forms and reports, inherited filters aren't automatically applied when an object is opened.

- The **ServerFilterByForm** property setting is ignored if the form's record source is a stored procedure.
Example

The following example enables the "Order Lookup" form to be opened in a Microsoft Access Data Project in the Server Filter By Form window.

Forms("Order Lookup").ServerFilterByForm = True
Shape Property

Returns a String representing the shape command corresponding to the sorting and grouping of the specified report. Read-only.

expression.Shape

expression  Required. An expression that returns one of the objects in the Applies To list.
Remarks

Analyzing the shape command for a particular report can make it easier to create ADO recordsets that can be used with the report.
Example

The following example reads the shape command of the specified report and stores it to a string variable.

```vba
Dim strShape As String
strShape = Reports(0).Shape
```
ShortcutMenu Property

You can use the ShortcutMenu property to specify whether a shortcut menu is displayed when you right-click an object on a form. For example, you might want to disable a shortcut menu to prevent the user from changing the form's underlying record source by using one of the filtering commands on the form's shortcut menu. Read/write Boolean.

expression.ShortcutMenu

expression Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **ShortcutMenu** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td><strong>True</strong></td>
<td>(Default) Shortcut menus are displayed.</td>
</tr>
<tr>
<td>No</td>
<td><strong>False</strong></td>
<td>Shortcut menus aren't displayed.</td>
</tr>
</tbody>
</table>

You can set this property by using the form's [property sheet](#), a macro, or [Visual Basic](#).

This property controls the displaying of the shortcut menus for a form and for any of its [controls](#). If the **ShortcutMenu** property is set to No, shortcut menus aren't displayed when you right-click a form or any of its controls.

If you're developing a [wizard](#), you might want to hide shortcut menus on your wizard forms to prevent the user from viewing or using them. This is especially true for forms that display choices. For example, the **ShortcutMenu** property for the Startup form in the Northwind sample database is set to No. This prevents users from displaying shortcut menus for the form or controls on the form.
Example

The following example disables the shortcut menus for the Invoice form and its controls:

Forms!Invoice.ShortcutMenu = False
ShortcutMenuBar Property

You can use the ShortcutMenuBar property to specify the shortcut menu that will appear when you right-click on a form, report, or control on a form. Read/write String.

expression.ShortcutMenuBar

expression    Required. An expression that returns one of the objects in the Applies To list.
Remarks

You create these shortcut menus by pointing to Toolbars on the View menu and then clicking Customize. For more information about creating custom toolbars, see Create a custom shortcut menu for the current database.

Note The ShortcutMenuBar property applies only to controls on a form, not controls on a report.

You can also use the ShortcutMenuBar property to specify the menu bar macro that will be used to display a shortcut menu for a datasheet, form, form control, or report.

Note In versions of Microsoft Access prior to Microsoft Access 97, you created a custom shortcut menu by setting the ShortcutMenuBar property to the name of a menu bar macro. You then created a macro group containing the commands for this menu. This functionality is still supported in Microsoft Access. However, it is strongly recommended that you use the new Customize dialog box, available by pointing to Toolbars on the View menu, and then clicking Customize, to create custom shortcut menus.

Enter the name of the shortcut menu you want to display. A shortcut menu can be any command bar whose Type property is set to Popup. If you leave the ShortcutMenuBar property setting blank, Microsoft Access displays the built-in (default) shortcut menu or the application's global shortcut menu. If you set the ShortcutMenuBar property to a value that isn't the name of an existing shortcut menu or menu bar macro, the form, form control, or report won't have a shortcut menu (the default shortcut menu won't be shown).

You can set this property by using the object's property sheet, a macro, or Visual Basic.

In Visual Basic, you set this property by using a string expression that is the name of the shortcut menu you want to display.

To display the built-in shortcut menu for a database, form, form control, or report by using a macro or Visual Basic, set the property to a zero-length string ("").
You create a custom shortcut menu by first creating a toolbar that includes all the commands you want to appear on your custom shortcut menu. Then open the Toolbar Properties dialog box by selecting the toolbar in the Customize dialog box and clicking the Properties button. In the Toolbar Properties dialog box, set the Type property to Popup. This toolbar will now be available in the ShortcutMenuBar property box in the property sheet for a form, form control, or report.

When used with the Application object, the ShortcutMenuBar property enables you to display a custom shortcut menu as a global shortcut menu. However, if you've set the ShortcutMenuBar property for a form, form control, or report in the database, the custom shortcut menu of that object will be displayed in place of the database's global shortcut menu. You can display a different custom shortcut menu for a specific form, form control, or report by setting its ShortcutMenuBar property to a different shortcut menu. When the form, form control, or report has the focus, the custom shortcut menu for that object is displayed when the user clicks the right mouse button; otherwise, the global shortcut menu for the database is displayed.

Shortcut menus aren't available to any object if the AllowShortcutMenus property is set to False.
Example

The following example sets the "Suppliers_Toolbar" as the custom shortcut menu to display when the user clicks the right mouse button on the "Suppliers" form.

Forms("Suppliers").ShortcutMenuBar = "Suppliers_Toolbar"
SizeMode Property

You can use the **SizeMode** property to specify how to size a picture or other object in a bound object frame, an unbound object frame, or an image control.

```
expression.SizeMode
```

*expression* Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **SizeMode** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clip</td>
<td>acOLESizeClip</td>
<td>(Default) Displays the object at actual size. If the object is larger than the control, its image is clipped on the right and bottom by the control's borders.</td>
</tr>
<tr>
<td>Stretch</td>
<td>acOLESizeStretch</td>
<td>Sizes the object to fill the control. This setting may distort the proportions of the object.</td>
</tr>
<tr>
<td>Zoom</td>
<td>acOLESizeZoom</td>
<td>Displays the entire object, resizing it as necessary without distorting the proportions of the object. This setting may leave extra space in the control if the control is resized.</td>
</tr>
</tbody>
</table>

**Note** You can set the **SizeMode** property in a property sheet, in a macro, or by using Visual Basic. You can set the default for this property by using a control's default control style or the **DefaultControl** method in Visual Basic.

**Tip**

Use the Clip setting for the fastest display. You can use the Stretch setting for bar graphs and line graphs without concern for size adjustments. The Stretch setting can distort circles and photos.
Example

The following example creates a linked OLE object using an unbound object frame named OLE1 and sizes the control to display the object's entire contents when the user clicks a command button.

Sub Command1_Click
    OLE1.Class = "Excel.Sheet" ' Set class name.
    ' Specify type of object.
    OLE1.OLETypeAllowed = acOLELinked
    ' Specify source file.
    OLE1.SourceDoc = "C:\Excel\Oletext.xls"
    ' Specify data to create link to.
    OLE1.SourceItem = "R1C1:R5C5"
    ' Create linked object.
    OLE1.Action = acOLECreateLink
    ' Adjust control size.
    OLE1.SizeMode = acOLESizeZoom
End Sub
SmartTagActions Property

Returns a SmartTagActions collection that represents the actions available for a specific smart tag. Read-only.

Smart tag actions are processes that are programmed into smart tags that allow users to perform certain functions related to the smart tag. For example, one action for a smart tag might be to access a Web site, while another action inserts contact information from Microsoft Outlook, while yet another action displays a map and driving directions.

expression.SmartTagActions

expression Required. An expression that returns a SmartTag object.
SmartTags Property

Returns a `SmartTags` collection that represents the collection of `smart tags` that have been added to a control.

`expression.SmartTags`

`expression` Required. An expression that returns one of the objects in the Applies To list.
Remarks

Using the SmartTags property of the Control object will result in a run-time error if the control is not a ComboBox, Label, or TextBox object.

**Note**  Unlike the SmartTags collections in Microsoft Excel and Microsoft Word, the SmartTags collection in Microsoft Access is zero-based. Therefore, the code `control.SmartTags(0)` returns the first smart tag for the specified control.
SortOrder Property

You use the **SortOrder** property to specify the **sort order** for fields and **expressions** in a report. For example, if you're printing a list of suppliers, you can sort the records alphabetically by company name. Read/write **Boolean**.

**expression.SortOrder**

**expression**  Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **SortOrder** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ascending</td>
<td>False</td>
<td>(Default) Sorts values in ascending (A to Z, 0 to 9) order.</td>
</tr>
<tr>
<td>Descending</td>
<td>True</td>
<td>Sorts values in descending (Z to A, 9 to 0) order.</td>
</tr>
</tbody>
</table>

**Note** You can set the **SortOrder** property by using the **Sorting And Grouping** box, a macro, or **Visual Basic**.

In Visual Basic, you set the **SortOrder** property in report **Design view** or in the **Open** event procedure of a report by using the **GroupLevel** property.
Example

The following example sets the sort order to ascending for the first group level in the "Product Summary" report.

Reports("Product Summary").GroupLevel(0).SortOrder = False
SourceDoc Property

You can use the **SourceDoc** property to specify the file to create a link to or to embed when you create a linked object or embedded object by using the **Action** property in Visual Basic. Read/write **String**.

`expression.SourceDoc`

`expression` Required. An expression that returns one of the objects in the Applies To list.
Remarks

For an embedded object, enter the full path and file name for the file you want to use as a template and set the **Action** property to **acOLECreateEmbed**.

For a linked object, enter the full path and file name of the file to create a link to and set the **Action** property to **acOLECreateLink**.

You can set this property in a **property sheet**, in a **macro**, or by using **Visual Basic**.

**Note** While this property appears in the property sheet, it takes effect only after the **Action** property is set in a macro or by using Visual Basic.

You can use the **SourceDoc** property to specify the file to create a link to and the control's **SourceItem** property to specify the data within that file. If you want to create a link to the entire object, leave the **SourceItem** property blank.

When a linked unbound object is created, the control's **SourceItem** property setting is concatenated with its **SourceDoc** property setting. In **Form view**, **Datasheet view**, and **Print Preview**, the control's **SourceItem** property setting is a **zero-length string** (" "), and its **SourceDoc** property setting is the full path to the linked file, followed by an exclamation point (!) or a backslash (\) and the **SourceItem** property setting, as in the following example:

"C:\Work\Qtr1\Revenue.xls!R1C1:R30C15"
Example

The following example creates a linked OLE object using an unbound object frame named OLE1 and sizes the control to display the object's entire contents when the user clicks a command button.

Sub Command1_Click
    OLE1.Class = "Excel.Sheet" ' Set class name.
    ' Specify type of object.
    OLE1.OLETypeAllowed = acOLELinked
    ' Specify source file.
    OLE1.SourceDoc = "C:\Excel\Oletext.xls"
    ' Specify data to create link to.
    OLE1.SourceItem = "R1C1:R5C5"
    ' Create linked object.
    OLE1.Action = acOLECreateLink
    ' Adjust control size.
    OLE1.SizeMode = acOLESizeZoom
End Sub
**SourceItem Property**

You can use the **SourceItem** property to specify the data within a file to be linked when you create a linked **OLE object**. Read/write **String**.

*expression*.SourceItem

**expression**  Required. An expression that returns one of the objects in the Applies To list.
Remarks

You can set the **SourceItem** property by specifying data in units recognized by the application supplying the object. For example, when you link to Microsoft Excel, you specify the **SourceItem** property setting by using a cell or cell-range reference such as R1C1 or R3C4:R9C22 or a named range such as Revenues.

**Note** To determine the syntax to describe a unit of data for a particular object, see the documentation for the application that was used to create the object.

You can set this property by using the [control's property sheet](#), a [macro](#), or [Visual Basic](#).

In Visual Basic, you set this property by using a [string expression](#).

The control's [OLETypeAllowed](#) property must be set to Linked or Either when you use this property. Use the control's [SourceDoc](#) property to specify the file to link.
Example

The following example creates a linked OLE object using an unbound object frame named OLE1 and sizes the control to display the object's entire contents when the user clicks a command button.

Sub Command1_Click
    OLE1.Class = "Excel.Sheet"    ' Set class name.
    ' Specify type of object.
    OLE1.OLETypeAllowed = acOLELinked
    ' Specify source file.
    OLE1.SourceDoc = "C:\Excel\Oletext.xls"
    ' Specify data to create link to.
    OLE1.SourceItem = "R1C1:R5C5"
    ' Create linked object.
    OLE1.Action = acOLECreateLink
    ' Adjust control size.
    OLE1.SizeMode = acOLESizeZoom
End Sub
You can use the **SourceObject** property to identify the form or report that is the source of the subform or subreport on a form or report. You can also use this property for linked unbound object frames to determine the complete path and file name of the file that contains the data linked to the object frame. Read/write String.

**expression.SourceObject**

**expression** Required. An expression that returns one of the objects in the Applies To list.
Remarks

Enter the name of the form or report that is the source of the subform or subreport in the control's property sheet. If you add a subform or subreport to the form or report by dragging it from the Database window, the SourceObject property is set automatically in the property sheet.

For unbound object frames, the SourceObject property is set automatically when you use the Object command on the Insert menu to insert a linked OLE object.

For a subform or subreport, you can set this property by using the control's property sheet, a macro, or Visual Basic.

In Visual Basic, you set this property by using a string expression that is a name of a form or report.

For linked unbound object frames, the SourceObject property can't be set in any view.

Note You can't set or change the SourceObject property in the Open or Format events of a report.

If you delete the SourceObject property setting in the property sheet for a subform or subreport, the control remains on the form but is no longer bound to the source form or report.
Example

The following example displays the name of the form that is the source of the ProductList subform control in the Debug window.

Debug.Print Forms!Categories! _
 [Product List].SourceObject
SpecialEffect Property

You can use the **SpecialEffect** property to specify whether special formatting will apply to a [section](#) or [control](#).

`expression.SpecialEffect`

`expression` Required. An expression that returns one of the objects in the Applies To list.
Remarks

The SpecialEffect property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flat</td>
<td>0</td>
<td>The object appears flat and has the system's default colors or custom colors that were set in Design view.</td>
</tr>
<tr>
<td>Raised</td>
<td>1</td>
<td>The object has a highlight on the top and left and a shadow on the bottom and right.</td>
</tr>
<tr>
<td>Sunken</td>
<td>2</td>
<td>The object has a shadow on the top and left and a highlight on the bottom and right.</td>
</tr>
<tr>
<td>Etched</td>
<td>3</td>
<td>The object has a sunken line surrounding the control.</td>
</tr>
<tr>
<td>Shadowed</td>
<td>4</td>
<td>The object has a shadow below and to the right of the control.</td>
</tr>
<tr>
<td>Chiseled</td>
<td>5</td>
<td>The object has a sunken line below the control.</td>
</tr>
</tbody>
</table>

You can set this property by using the Special Effect button on the Formatting (Form/Report) toolbar, the object's property sheet, a macro, or Visual Basic.

For controls, you can set the default for this property by using the default control style or the DefaultControl method in Visual Basic.

The SpecialEffect property setting affects related property settings for the BorderStyle, BorderColor, and BorderWidth properties. For example, if the SpecialEffect property is set to Raised, the settings for the BorderStyle, BorderColor, and BorderWidth properties are ignored. In addition, changing or setting the BorderStyle, BorderColor, and BorderWidth properties may cause Microsoft Access to change the SpecialEffect property setting to Flat.

**Note** When you set the SpecialEffect property of a text box to Shadowed, the vertical height of the text display area is reduced. You can adjust the Height property of the text box to increase the size of the text display area.
Example

The following example sets the appearance of the text box "OrganizationName1" on the "Mailing List" form to raised.

Forms("Mailing List").Controls("OrganizationName1").SpecialEffect =
StatusBarText Property

You can use the StatusBarText property to specify the text that is displayed in the status bar when a control is selected. Read/write String.

expression.StatusBarText

expression Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **StatusBarText** property applies only to controls on a **form**, not controls on a [report](#).

You set the **StatusBarText** property by using a [string expression](#) up to 255 characters long.

You can set this property by using the control's [property sheet](#), a [macro](#), or [Visual Basic](#).

**Note** The length of the text you can display in the status bar depends on your computer hardware and video display.

You can use the **StatusBarText** property to provide specific information about a control. For example, when a text box has the [focus](#), a brief instruction can tell the user what kind of data to enter.

**Note** You can also use the [ControlTipText](#) property to display a ScreenTip for a control.

If you create a control by dragging a field from the [field list](#), the value in a field's [Description](#) property is copied to the **StatusBarText** property.
Example

The following example sets the status bar text to be displayed when the "Address_TextBox" control in the "Mailing List" form has the focus in Form View.

Forms("Mailing List").Controls("Address_TextBox"). _
   StatusBarText = "Enter the company's mailing address."
Style Property

You can use the **Style** property to specify or determine the appearance of tabs on a tab control.

*expression.Style*

*expression* Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **Style** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tabs</td>
<td>0</td>
<td>(Default) Tabs appear as tabs.</td>
</tr>
<tr>
<td>Buttons</td>
<td>1</td>
<td>Tabs appear as buttons.</td>
</tr>
<tr>
<td>None</td>
<td>2</td>
<td>No tabs appear in the control.</td>
</tr>
</tbody>
</table>

**Note**  You can set the **Style** property by using the tab control's **property sheet**, a **macro**, or **Visual Basic**.

You can also set the default for this property by using a control's **default control style** or the **DefaultControl** method in Visual Basic.

You can set the **Style** property in any view.

When the tab control's **Style** property is set to Tabs or Buttons, the appearance of the tabs is determined by the **TabFixedHeight**, **TabFixedWidth**, and **MultiRow** properties.

You could set the tab control's **Style** property to None if you wanted complete control over when a user could move between tabs. In prior versions of Microsoft Access, wizard dialogs were created by using multiple-page forms. You can now use a tab control create your own wizard with each page of the wizard contained on a separate page of a tab control with its **Style** property set to None.

**Example**

The following example causes tabs to appear as buttons on the tab control named "TabCtl1" on the "Mailing List" form.

```vba
Forms("Mailing List").Controls("TabCtl1").Style = 1
```
SubAddress Property

You can use the SubAddress property to specify or determine a location within the target document specified by the Address property. The SubAddress property can be an object within a Microsoft Access database, a bookmark within a Microsoft Word document, a named range within a Microsoft Excel spreadsheet, a slide within a Microsoft PowerPoint presentation, or a location within an HTML document. Read/write String.

expression/SubAddress

expression Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **SubAddress** property is a *string expression* representing the **HyperlinkSubAddress** property of a named location within the target document specified by the **HyperlinkAddress** property.

You can set the **HyperlinkSubAddress** property with the **SubAddress** property by using Visual Basic.

**Note**  You can set the **HyperlinkSubAddress** property by using a control’s **property sheet**, a **macro**, or Visual Basic.

When you move the cursor over a **command button**, **image control**, or **label control** whose **HyperlinkSubAddress** property is set, the cursor changes to an upward-pointing hand. Clicking the control displays the object or Web page specified by the link.

For more information about hyperlink addresses and their format, see the **HyperlinkAddress** and **HyperlinkSubAddress** property topics.
Example

The following example turns a label named "Label20" on the "Suppliers" form into an active hyperlink. When the user click the hyperlink, Access opens the "Mailing List" form in the "Postal Operations" database.

With Forms("Suppliers").Controls("Label20").Hyperlink
    .Address = "PostalOperations.mdb"
    .SubAddress = "Form Mailing List"
End With
SubdatasheetExpanded Property

You can use the **SubdatasheetExpanded** property to specify or determine the saved state of all subdatasheets within a table or query. Read/write **Boolean**.

*expression/SubdatasheetExpanded*

*expression*  Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **SubdatasheetExpanded** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>True</td>
<td>The saved state of all subdatasheets in the table is expanded.</td>
</tr>
<tr>
<td>No</td>
<td>False</td>
<td>(Default) The saved state of all subdatasheets in the table is closed.</td>
</tr>
</tbody>
</table>

**Note** The easiest way to set the **SubdatasheetExpanded** property is by using a [table's property sheet](https://example.com). You can set this property by using **Visual Basic**.

To set the **SubdatasheetExpanded** property by using Visual Basic, you must first either:

- Set the property in [table Design view](https://example.com) by pointing to **Properties** on the **View** menu.

- Create the property by using the DAO **CreateProperty** method.

The value of the **SubdatasheetExpanded** property is displayed in the **Table Properties** property sheet.

The **SubdatasheetExpanded** and **SubdatasheetHeight** properties take effect on the subform control when the form is in datasheet view.
### Example

The following example turns subdatasheet expansion on or off for the "Purchase Orders" form.

```vba
Dim strExpand As String

With Forms("Purchase Orders")
    strExpand = InputBox("Expand subdatasheets? Y/N")
    Select Case strExpand
        Case "Y"
            .SubdatasheetExpanded = True
        Case "N"
            .SubdatasheetExpanded = False
        Case Else
            MsgBox "Can't determine subdatasheet expansion state."
    End Select
End With
```

To try this example yourself, open a form (containing a subform) in Design view, click the Builder button next to the On Load property box in the form's property window, paste this code into the form's Form_Load event (removing the reference to the "Purchase Orders" form), and then open the form in Datasheet view.
SubdatasheetHeight Property

You can use the SubdatasheetHeight property to specify or determine the display height of a subdatasheet when expanded. Read/write Integer.

expression.SubdatasheetHeight

expression  Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **SubdatasheetHeight** property in a [numeric expression](#) representing the default height to display in the subdatasheet.

The easiest way to set the **SubdatasheetHeight** property is by using a [table's property sheet](#). You can set this property by using Visual Basic. In Visual Basic, the property's value is expressed in twips.

To set the **SubdatasheetHeight** property by using Visual Basic, you must first either:

- Set the property in table **Design view** by pointing to **Properties** on the **View** menu.

- Create the property by using the DAO [CreateProperty](#) method.

If the subdatasheet includes more records than the height setting can accommodate, a vertical scrollbar is displayed.

The **SubdatasheetHeight** property setting includes the New Record row if adding new records is supported. It does not include the column header row or scrollbar region.

The **SubdatasheetHeight** and **SubdatasheetExpanded** properties take effect on the subform control when the form is in datasheet view.
Example

The following example resizes the height of the subdatasheet in the "Purchase Orders" form (containing a subform) to show only one line of the subdatasheet at a time (measured at about 400 twips), accompanied by a vertical scrollbar. The number 400 is arbitrary, and will vary based on monitor resolution and default font size. This behavior can only be seen in Datasheet View.

Forms("Purchase Orders").SubdatasheetHeight = 400
**TabFixedHeight Property**

You can use the **TabFixedHeight** property to specify or determine the height of the tabs on a [tab control](#). Read/write **Integer**.

`expression.TabFixedHeight`

`expression` Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **TabFixedHeight** property setting is a value that represents the height of tabs in the unit of measurement specified in the **Regional Options** dialog box in Windows Control Panel. If you set this property to zero, the tabs automatically adjust to the height of the tab contents.

You can set this property by using the tab control's [property sheet](#), a [macro](#), or Visual Basic.

You can also set the default for this property by using a control's [default control style](#) or the **DefaultControl** method in Visual Basic.

In Visual Basic this property uses a **Long Integer** value representing the height of the tabs in **twips** and can be set in any view.

**Note**  To use a unit of measurement different from the setting in the **Regional Options** dialog box in Windows Control Panel, specify the unit, such as cm or in (for example, 5 cm or 3 in).

You can't change the color of a tab control. If the tabs don't cover the height of the tab control, the area behind the tabs is displayed. If you place a tab control on an object with a different color than the tab control, you should make sure that the tabs cover the control's background area.
**Example**

The following example sets the height of each tab in the tab control "TabCtl1" on the "Mailing List" form to 500 twips.

Forms("Mailing List").Controls("TabCtl1").TabFixedWidth = 500
TabFixedWidth Property

You can use the TabFixedWidth property to specify or determine the width of the tabs on a tab control. Read/write Integer.

expression.TabFixedWidth

expression Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **TabFixedWidth** property setting is a value that represents the width of tabs in the unit of measurement specified in the **Regional Options** dialog box in Windows Control Panel. If you set this property to zero, the tabs automatically adjust to the width of the tab contents.

You can set this property by using the tab control's **property sheet**, a **macro**, or Visual Basic.

You can also set the default for this property by using a control's **default control style** or the **DefaultControl** method in Visual Basic.

In Visual Basic this property uses a **Long Integer** value representing the width of the tabs in **twips** and can be set in any view.

**Note**  To use a unit of measurement different from the setting in the **Regional Options** dialog box in Windows Control Panel, specify the unit, such as cm or in (for example, 5 cm or 3 in).

You can't change the color of a tab control. If the tabs don't cover the width of the tab control, the area behind the tabs is displayed. If you place a tab control on an object with a different color than the tab control, you should make sure that the tabs cover the control's background area.
Example

The following example sets the width of each tab in the tab control "TabCtl1" on the "Mailing List" form to 2000 twips.

Forms("Mailing List").Controls("TabCtl1").TabFixedWidth = 2000
**TabIndex Property**

You can use the `TabIndex` property to specify a control’s place in the tab order on a form. Read/write Integer.

`expression.TabIndex`

`expression` Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **TabIndex** property applies only to controls on a form, not controls on a report.

You can set the **TabIndex** property to an integer representing the position of the control within the tab order of the form. Valid settings are 0 for the first tab position, up to the total number of controls minus 1 for the last tab position. For example, if a form has three controls that each have a **TabIndex** property, valid **TabIndex** property settings are 0, 1, and 2.

Setting the **TabIndex** property to an integer less than 0 produces an error.

You can set this property by using the control's property sheet, a macro, or Visual Basic.

**Note** You can also set the tab order of controls on a form by using the **Tab Order** command on the View menu. This also sets the **TabOrder** property for the controls.

By default, Microsoft Access assigns a tab order to controls in the order that you create them on a form. Each new control is placed last in the tab order. If you change the setting of a control's **TabIndex** property to adjust the tab order, Microsoft Access automatically renumbers the **TabIndex** property setting of other controls to reflect insertions and deletions.

In **Form view**, invisible or disabled controls remain in the tab order but are skipped when you press the TAB key.

Changing the tab order of other controls on the form doesn't affect what happens when you press a control's access key. For example, if you've created an access key for the label of a text box, the focus will move to the text box whenever you press the label's access key—even if you change the **TabIndex** property setting for the text box.

If you press an access key for a control such as a label that doesn't have a **TabIndex** property (and thus isn't in the tab order), the focus moves to the next control in the tab order that can receive the focus.
Example

The following example reverses the tab order of a command button and a text box. Because TextBox1 was created first, it has a TabIndex property setting of 0 and Command1 has a setting of 1.

Sub Form_Click()
    Me!Command1.TabIndex = 0
    Me!TextBox1.TabIndex = 1
End Sub
TabStop Property

You can use the TabStop property to specify whether you can use the TAB key to move the focus to a control in Form view. Read/write Boolean.

expression.TabStop

expression Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **TabStop** property applies only to controls on a form, not controls on a report.

This property doesn't apply to check box, option button, or toggle button controls when they appear in an option group. It applies only to the option group itself.

The **TabStop** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>True</td>
<td>(Default) You can move the focus to the control by pressing the TAB key.</td>
</tr>
<tr>
<td>No</td>
<td>False</td>
<td>You can't move the focus to the control by pressing the TAB key.</td>
</tr>
</tbody>
</table>

**Note** You can set this property by using the control's property sheet, a macro, or Visual Basic.

When you create a control on a form, Microsoft Access automatically assigns the control a position in the form's tab order. Each new control is placed last in the tab order. If you want to prevent a control from being available when you tab through the controls in a form, set the control's **TabStop** property to No.

In Form view, hidden or disabled controls remain in the tab order but are skipped when you move through the controls by pressing TAB, even if their **TabStop** properties are set to Yes.

As long as a control's **Enabled** property is set to Yes, you can click the control or use an access key to select it, regardless of its **TabStop** property setting. For example, you can set the **TabStop** property of a command button to No to prevent users from selecting the button by pressing TAB. However, they can still click the command button to choose it.
Example

The following example disables the ability to move the focus to the "City" text box on the "Suppliers" form by using the TAB key.

Forms("Suppliers").Controls("City").TabStop = False
Tag Property

Stores extra information about a form, report, data access page, section, or control needed by a Microsoft Access application. Read/write String.

expression.Tag

expression Required. An expression that returns one of the objects in the Applies To list.
Remarks

You can enter a string expression up to 2048 characters long. The default setting is a zero-length string (" ").

You can set this property by using the object's property sheet, a macro, or Visual Basic.

Unlike other properties, the Tag property setting doesn't affect any of an object's attributes.

You can use this property to assign an identification string to an object without affecting any of its other property settings or causing other side effects. The Tag property is useful when you need to check the identity of a form, report, data access page, section, or control that is passed as a variable to a procedure.
Example

The following example uses the Tag property to display custom messages about controls on a form. When a control has the focus, descriptive text is displayed in a label control called lblMessage. You specify the text for the message by setting the Tag property for each control to a short text string. When a control receives the focus, its Tag property is assigned to the label control's Caption property. This example displays the descriptive text for a text box named txtDescription and a command button named cmdButton on a form.

Sub Form_Load()
    Dim frmMessageForm As Form
    Set frmMessageForm = Forms!Form1
    frmMessageForm!lblMessage.Caption = "" ' Clear text.
    frmMessageForm!txtDescription.Tag = "Help text for the text box.
    frmMessageForm!cmdButton.Tag = "Help text for the command button
End Sub

Sub txtDescription_GotFocus()
    ' Tag property setting as caption.
    Me!lblMessage.Caption = Me!txtDescription.Tag
End Sub

Sub txtDescription_LostFocus()
    Me!lblMessage.Caption = ""
End Sub

Sub cmdButton_GotFocus()
    ' Tag property setting as caption.
    Me!lblMessage.Caption = Me!cmdButton.Tag
End Sub

Sub cmdButton_LostFocus()
    Me.lblMessage.Caption = " "
End Sub
TargetBrowser Property

Returns or sets an MsoTargetBrowser constant indicating which Web browser is the intended target for the specified data access page or for all data access pages. Read/write.

MsoTargetBrowser can be one of these MsoTargetBrowser constants.

- **msoTargetBrowserIE4** Microsoft Internet Explorer version 4.
- **msoTargetBrowserIE5** Internet Explorer version 5.
- **msoTargetBrowserIE6** Internet Explorer version 6.
- **msoTargetBrowserV3** Netscape Navigator version 3.
- **msoTargetBrowserV4** Netscape Navigator version 4.

expression.**TargetBrowser**

expression Required. An expression that returns one of the objects in the Applies To list.
Example

This example sets the target browser for the active data access page to Microsoft Internet Explorer 6 if the current target browser is an earlier version of Internet Explorer.

```
With Screen.ActiveDataAccessPage.WebOptions
    If .TargetBrowser = msoTargetBrowserIE4 Or _
        .TargetBrowser = msoTargetBrowserIE5 Then
        .TargetBrowser = msoTargetBrowserIE6
    End If
End With
```

This example sets the target browser for all data access pages to Internet Explorer 6.

```
Application.DefaultWebOptions._
    .TargetBrowser = msoTargetBrowserIE6
```
Text Property

You can use the Text property to set or return the text contained in a text box or in the text box portion of a combo box. Read/write String.

expression.Text

expression Required. An expression that returns one of the objects in the Applies To list.
Remarks

You can set the Text property to the text you want to display in the control. You can also use the Text property to read the text currently in the control.

You can set or read this property only by using a macro or Visual Basic.

Note To set or return a control's Text property, the control must have the focus, or an error occurs. To move the focus to a control, you can use the SetFocus method or GoToControl action.

While the control has the focus, the Text property contains the text data currently in the control; the Value property contains the last saved data for the control. When you move the focus to another control, the control's data is updated, and the Value property is set to this new value. The Text property setting is then unavailable until the control gets the focus again. If you use the Save Record command on the Records menu to save the data in the control without moving the focus, the Text property and Value property settings will be the same.
**Example**

The following example uses the `Text` property to enable a Next button named `btnNext` whenever the user enters text into a text box named `txtName`. Anytime the text box is empty, the Next button is disabled.

Sub txtName_Change()
    btnNext.Enabled = Len(Me!txtName.Text & ":")<>0
End Sub
**TextAlign Property**

The ** TextAlign** property specifies the text alignment in new controls. Read/write **Byte**.

`expression.TextAlign`

`expression` Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **TextAlign** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>General</td>
<td>0</td>
<td>(Default) The text aligns to the left; numbers and dates align to the right.</td>
</tr>
<tr>
<td>Left</td>
<td>1</td>
<td>The text, numbers, and dates align to the left.</td>
</tr>
<tr>
<td>Center</td>
<td>2</td>
<td>The text, numbers, and dates are centered.</td>
</tr>
<tr>
<td>Right</td>
<td>3</td>
<td>The text, numbers, and dates align to the right.</td>
</tr>
<tr>
<td>Distribute</td>
<td>4</td>
<td>The text, numbers, and dates are evenly distributed.</td>
</tr>
</tbody>
</table>

**Note** You can set the **TextAlign** property by using the control’s **property sheet**, a **macro**, or **Visual Basic**.

You can also set the **TextAlign** property by clicking **Align Left**, **Center**, and **Align Right** on the **Formatting (Form/Report)** toolbar.

You can set the default for the **TextAlign** property by using a control's default control style or the **DefaultControl** method in Visual Basic.
Example

The following example aligns the text in the "Address" text box on the "Suppliers" form to the right.

Forms("Suppliers").Controls("Address").** TextAlign** = 3
**TextToDisplay Property**

You can use the `TextToDisplay` property to specify or determine the display text for a hyperlink. Read/write `String`.

`expression.TextToDisplay`

`expression`  Required. An expression that returns one of the objects in the Applies To list.
Remarks

The `TextToDisplay` property is a `string expression` representing the text displayed as a hyperlink.

The `TextToDisplay` property can be set by using the form's `property sheet` or `Visual Basic`. 
Example

The following example displays the words "Go to Home page" as an active hyperlink in the label named "Label20" on the "Suppliers" form. Clicking the hyperlink takes the user to the address specified in the label's HyperlinkAddress property.

Forms.Item("Suppliers").Controls.Item("Label20").Hyperlink._
    TextToDisplay = "Go to Home page"
TimerInterval Property

You can use the TimerInterval property to specify the interval, in milliseconds, between Timer events on a form. Read/write Long.

expression.TimerInterval

expression Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **TimerInterval** property setting is a **Long Integer** value between 0 and 2,147,483,647.

You can set this property by using the form's property sheet, a macro, or Visual Basic.

**Note** When using Visual Basic, you set the **TimerInterval** property in the form's **Load** event.

To run Visual Basic code at intervals specified by the **TimerInterval** property, put the code in the form's Timer event procedure. For example, to requery records every 30 seconds, put the code to requery the records in the form's Timer event procedure, and then set the **TimerInterval** property to 30000.
Example

The following example shows how to create a flashing button on a form by displaying and hiding an icon on the button. The form's Load event procedure sets the form's **TimerInterval** property to 1000 so the icon display is toggled once every second.

```vba
Sub Form_Load()
    Me.TimerInterval = 1000
End Sub

Sub Form_Timer()
    Static intShowPicture As Integer
    If intShowPicture Then
        ' Show icon.
        Me!btnPicture.Picture = "C:\Icons\Flash.ico"
    Else
        ' Don't show icon.
        Me!btnPicture.Picture = ""
    End If
    intShowPicture = Not intShowPicture
End Sub
```
Toolbar Property

You can use the **Toolbar** property to specify the toolbar to use for a **form** or **report**. You create these toolbars by using the **Customize** subcommand of the **Toolbars** command on the **View** menu. For more information on creating customized toolbars, see Create a custom toolbar for the current database. Read/write **String**.

*expression*.**Toolbar*

*expression* Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **Toolbar** property setting is the name of the custom toolbar you want to display. If you leave the **Toolbar** property setting blank, Microsoft Access displays the form's or report's built-in (default) toolbar.

You can set the **Toolbar** property by using the form's or report's property sheet, a macro, or **Visual Basic**.

In Visual Basic, set this property by using a **string expression** that's the name of the toolbar you want to display.

To display the built-in toolbar for a form or report by using a macro or Visual Basic, set the **Toolbar** property to a **zero-length string** (" ").

When you set the **Toolbar** property, Microsoft Access displays the specified custom toolbar when the form or report is opened. This toolbar is displayed whenever the form or report has the **focus**.
Example

The following example displays the custom toolbar named "Suppliers_Toolbar" associated with the "Suppliers" form.

Forms.Item("Suppliers").Toolbar = "Suppliers_Toolbar"
Top Property

You can use the Top property to specify an object's location on a form or report. Read/write Integer for all of the objects in the Applies To list except for the Report object, which is read-only Long.

expression Required. An expression that returns one of the above objects.
Remarks

A control's location is the distance measured from its left or top border to the left or top edge of the section containing the control. Setting the Top property to 0 places the control's edge at the very top of the section. To use a unit of measurement different from the setting in the Regional Options dialog box in Windows Control Panel, specify the unit, such as cm or in (for example, 3 cm or 2 in).

In Visual Basic, use a numeric expression to set the value of this property. Values are expressed in twips.

For controls, you can set this property by using a control's property sheet, a macro, or Visual Basic.

When you move a control, its new Top property setting is automatically entered in the property sheet. When you view a form or report in Print Preview or when you print a form, a control's location is determined by its Top property setting along with the margin settings in the Page Setup dialog box, available by clicking Page Setup on the File menu.

For reports, the Top property setting is the amount the current section is offset from the top of the page. This property setting is expressed in twips. You can use this property to specify how far down the page you want a section to print in the section's Format event procedure.
Example

The following example checks the **Top** property setting for the current report. If the value is less than the minimum margin setting, the **NextRecord** and **PrintSection** properties are set to **False**. The section doesn't advance to the next record, and the next section isn't printed.

```vbnet
Sub Detail1_Format(Cancel As Integer, FormatCount As Integer)
Const conTopMargin = 1880
' Don't advance to next record or print next section
' if Top property setting is less than 1880 twips.
    If Me.Top < conTopMargin Then
        Me.NextRecord = False
        Me.PrintSection = False
    End If
End Sub
```
TopMargin Property

TopMargin property as it applies to the Label and TextBox objects.

Along with the LeftMargin, RightMargin, and BottomMargin properties, specifies the location of information displayed within a label or text box control. Read/write Integer.

expression.TopMargin

expression  Required. An expression that returns one of the above objects.
Remarks

A control's displayed information location is measured from the control's left, top, right, or bottom border to the left, top, right, or bottom edge of the displayed information. Setting the **LeftMargin** or **TopMargin** property to 0 places the displayed information's edge at the very left or top of the control. To use a unit of measurement different from the setting in the regional settings of Windows, specify the unit (for example, cm or in).

In Visual Basic, use a numeric expression to set the value of this property. Values are expressed in **twips**.

You can set these properties by using the **property sheet**, a **macro**, or **Visual Basic**.

**$expression.TopMargin**

$expression Required. An expression that returns a **Printer** object.
Example

As it applies to the Label and TextBox objects.

The following example offsets the caption in the label "EmployeeID_Label" in the "Purchase Orders" form by 100 twips from the top of the label's border.

With Forms.Item("Purchase Orders").Controls.Item("EmployeeID_Label")
  .TopMargin = 100
End With
Transparent Property

You can use the `Transparent` property to specify whether a command button is solid or transparent. Read/write `Boolean`.

`expression.Transparent`

`expression` Required. An expression that returns one of the objects in the Applies To list.
Remarks

**Tip** Use the **BackStyle** property to make other controls solid or transparent.

The **Transparent** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td><strong>True</strong></td>
<td>The button is transparent.</td>
</tr>
<tr>
<td>No</td>
<td><strong>False</strong></td>
<td>(Default) The button is solid.</td>
</tr>
</tbody>
</table>

**Note** You can set this property by using the command button's **property sheet**, a **macro**, or **Visual Basic**.

You can use this property to place a transparent command button over another control. For example, you could place several transparent buttons over a picture displayed in an image control and run various macros or Visual Basic event procedures depending on which part of the picture the user clicks.

**Note** To hide and disable a button, use the **Visible** property. To disable a button without hiding it, use the **Enabled** property. To hide a button only when a form or report is printed, use the **DisplayWhen** property.
Example

The following example makes the command button "Preview" on the "Purchase Orders" form transparent.

Forms.Item("Purchase Orders").Controls.Item("Preview"). _
  Transparent = True
TripleState Property

You can use the `TripleState` property to specify how a [check box], [toggle button], or [option button] will display `Null` values. Read/write [Boolean].

`expression.TripleState`

`expression`  Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **TripleState** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>True</td>
<td>The control will cycle through states for Yes, No, and Null values. The control appears dimmed (grayed) when its <strong>Value</strong> property is set to Null.</td>
</tr>
<tr>
<td>No</td>
<td>False</td>
<td>(Default) The control will cycle through states for Yes and No values. Null values display as if they were No values.</td>
</tr>
</tbody>
</table>

**Note** You can set the **TripleState** property by using a control's property sheet, a macro, or Visual Basic.

This property can be set in any view.
Example

The following example displays a message describing in detail the state of a check box named "Check1" on the form "frmOperations".

Dim strTripleState As String
strTripleState = Forms.Item("frmOperations").Controls.Item("Check1")

Select Case strTripleState
  Case True
    MsgBox "For Check1, TripleState = " & strTripleState & ". The control will cycle through states for Yes, No, " ", and Null values. The control appears dimmed (grayed) " ", when its Value property is set to Null."
  Case False
    MsgBox "For Check1, TripleState = " & strTripleState & ". The control will cycle through states for Yes and No " ", values. Null values display as if they were No values."
  Case Else
    MsgBox "Can't determine the TripleState property for Check1."
End Select
Type Property

Type property as it applies to the **AccessObject** object.

Returns the value of an **AccessObject** object type. Read-only **AcObjectType**.

AcObjectType can be one of these AcObjectType constants.

- **acDataAccessPage**
- **acDefault**
- **acDiagram**
- **acForm**
- **acFunction**
- **acMacro**
- **acModule**
- **acQuery**
- **acReport**
- **acServerView**
- **acStoredProcedure**
- **acTable**

<table>
<thead>
<tr>
<th>expression.Type</th>
<th></th>
</tr>
</thead>
</table>

expression Required. An expression that returns an **AccessObject** object.

Type property as it applies to the **FormatCondition** object.

Returns the value of a **FormatCondition** object type. Read-only **AcFormatConditionType**.

AcFormatConditionType can be one of these AcFormatConditionType constants.

- **acExpression**
- **acFieldHasFocus**
acFieldValue

expression.Type

expression Required. An expression that returns a FormatCondition object.

Type property as it applies to the Module object.

Indicates whether a module is a standard module or a class module. Read-only AcModuleType.

AcModuleType can be one of these AcModuleType constants.

acClassModule

acStandardModule

expression.Type

expression Required. An expression that returns a Module object.
Example

As it applies to the Module object.

The following example determines whether a Module object represents a standard module or a class module:

Function CheckModuleType(strModuleName As String) As Integer
    Dim mdl As Module
    
    ' Open module to include in Modules collection.
    DoCmd.OpenModule strModuleName
    
    ' Return reference to Module object.
    Set mdl = Modules(strModuleName)
    
    ' Check Type property.
    If mdl.Type = acClassModule Then
        ' Insert comment.
        mdl.InsertLines 1, "' Class module."
        CheckModuleType = acClassModule
    Else
        ' Insert comment.
        mdl.InsertLines 1, "' Standard module."
        CheckModuleType = acStandardModule
    End If
End Function
UnderlineHyperlinks Property

You can use the UnderlineHyperlink property to specify or determine if hyperlinks within the Application object should be underlined when displayed. Read/write Boolean.

expression.UnderlineHyperlinks

expression Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **UnderlineHyperlink** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>True</td>
<td>(Default) Hyperlinks will be underlined</td>
</tr>
<tr>
<td>No</td>
<td>False</td>
<td>Hyperlinks will not be underlined.</td>
</tr>
</tbody>
</table>

You can set the **UnderlineHyperlink** property through the **DefaultWebOptions** property or the **SetOption** method by using **Visual Basic**.

You can set or change the underline hyperlink setting in the **Web Options** dialog box. To display this dialog box, click **Options** on the **Tools** menu. Click the **General** tab and click the **Web Pages** button.

Use the **DefaultWebOptions** property to identify or set the **Application** object's **DefaultWebOptions** object properties.
Example

The following example displays a message indicating whether hyperlinks will be underlined.

MsgBox "Hyperlinks will be underlined: " & _
Application.DefaultWebOptions.UnderlineHyperlinks
UniqueTable Property

Specifies the table to be updateable when:

- A **form** or **data access page** is bound to a multiple table **view** or **stored procedure** within a **Microsoft Access project** (.adp).
- A **data access page** is bound to a multiple table **query** within a **Microsoft Access project** (.adp) or a Microsoft Access database (.mdb).

Read/write **String**.

*expression*.UniqueTable

*expression*    Required. An expression that returns a **Form** object.
Remarks

The `UniqueTable` property is a *string expression* representing the unique table to be updatable.

You can set this property by using the [property sheet](#) or [Visual Basic](#).

The `UniqueTable` property identifies the "most many" table of a join. If you don't set the `UniqueTable` property, a form that is bound to a view or stored proc or SQL String containing a join is read only. Also, the datasheet produced by View.Open or StoredProc.Run is read only in the case of a join (because there's no way to set the `UniqueTable` property). Once you set the `UniqueTable` property, only fields from that table are updatable, and inserts and deletes can only be made to that table.

A form or data access page based on a join cursor must have a `UniqueTable` property string in order for the recordset to be an *updatable snapshot*. The Unique Table is the table in the underlying query whose rows have a 1-to-1 correspondence with rows in the cursor. In a simple Patients - Doctors join, Patients is the unique table because every row of the cursor corresponds to one row of the Patients table. Note that a Many-to-Many join does not have a valid `UniqueTable` property, and is thus read-only. The `UniqueTable` property will be exposed as a RecordsetDef object in the case of a data access page. The purpose of the `UniqueTable` property is:

To enforce the correct updatability semantics:

- The key columns of the Unique Table must be present in the select list of the query that forms the cursor, even for SQL Server. (For other data sources, see the Remarks section in the [ResyncCommand](#) property topic.)

- Deletion of a row in a join cursor deletes the row from the Unique Table only.

- Insertion of a row in a join cursor is allowed for the Unique Table only

- Update of a row in a join cursor is allowed for fields in the Unique Table only.
To provide the right parameter values for the Resync Query. The **UniqueTable** property of a form or a RecordsetDef supports the catalog.owner.tablename notation to fully qualify a base table from others in the same cursor, if this is required. For example, for example, if dbo.authors were joined to user1.authors in a cursor, then the UniqueTable would need to be specified as dbo.authors or user1.authors.

For a join cursor, if the **UniqueTable** property is empty, the recordset reverts to read only and any attempt to edit results in a beep and a status message, "This Recordset is not updatable because the **UniqueTable** property is not set." If there is a non-empty **UniqueTable** property, set the **UniqueTable** property (and UniqueSchema, UniqueCatalog properties if necessary) on the underlying Recordset or Rowset. Then, go through and mark each column that does not match the **UniqueTable** property as read only.

On insert and update operations, only the fields from the Unique Table are available for editing. When the user tries to type into them he gets a beep and the message "Only fields from the Unique Table can be edited." If the **UniqueTable** property has been set incorrectly, this will happen for all columns.
UnsupportedObjects Property

Returns a DependencyObjects collection that contains objects that cannot be searched for dependency. Read-only.

expression.UnsupportedObjects

expression Required. An expression that returns a DependencyInfo object.
Remarks

The DependencyObjects collection returned by this property contains a collection of AccessObject objects.
UpdateOptions Property

You can use the UpdateOptions property to specify how a linked OLE object is updated. Read/write Integer.

expression.UpdateOptions

expression  Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **UpdateOptions** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Automatic</td>
<td>acOLEUpdateAutomatic</td>
<td>(Default) Updates the object each time the linked data changes.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Updates the object only when the control's <strong>Action</strong> property is set to</td>
</tr>
<tr>
<td></td>
<td></td>
<td>acOLEUpdate</td>
</tr>
<tr>
<td>Manual</td>
<td>acOLEUpdateManual</td>
<td>or the link is updated with the <strong>OLE/DDE Links</strong> command on the <strong>Edit</strong> menu.</td>
</tr>
</tbody>
</table>

You can set the **UpdateOptions** property in a **property sheet**, in a **macro**, or by using **Visual Basic**. You can set the default for this property by using a control's **default control style** or the **DefaultControl** method in Visual Basic.

The **UpdateOptions** property setting is also available through the **OLE/DDE Links** command on the **Edit** menu.

Normally, the object is updated automatically whenever the linked data changes, but you can tell Microsoft Access to update the data only when it receives a specific instruction to do so. For example, if other users or applications can access or change linked spreadsheet data on a form, you can use this property to specify that the linked data only be updated when the database is opened in single-user mode.

When the **UpdateOptions** property is set to Manual, updates don't occur based on the setting of the **Refresh interval** box on the **Advanced** tab of the **Options** dialog box, available by clicking **Options** on the **Tools** menu.

**Note**  When an object's data is changed, the **Updated** event occurs.
Example

The following example sets the **UpdateOptions** property for an unbound object frame named OLE1 to update manually, and then uses the **Action** property to force an update of the OLE object in the control.

OLE1.UpdateOptions = acOLEUpdateManual
OLE1.Action = acOLEUpdate
UseDefaultPrinter Property

Returns or sets a Boolean indicating whether the specified form or report uses the default printer for the system; True if the form or report uses the default printer. Read/write.

expression. UseDefaultPrinter

expression  Required. An expression that returns one of the objects in the Applies To list.
Remarks

This property is read/write in Design view and read-only in all other views.

When this property is True, the form or report inherits all of its printer settings from the settings of the default printer. Changing the printer associated with a form or report by assigning its Printer property to a Printer object sets the UseDefaultPrinter property to False.
Example

The following example checks to see if the specified form is using the default printer; if not, the user is asked if it should.

Function CheckPrinter(frmTemp As Form) As Boolean

    If frmTemp.UseDefaultPrinter = False Then
        If MsgBox("Should this form use " _
            & "the default printer?", _
            vbYesNo) = vbYes Then
            frmTemp.UseDefaultPrinter = True
        End If
    End If

End Function
UseLongFileNames Property

You can use the **UseLongFileNames** property to specify or determine whether long file names are used when a document is stored as a data access page. Read/write **Boolean**.

*expression*.UseLongFileNames

*expression* Required. An expression that returns one of the objects in the Applies To list.
Remarks

The UseLongFileNames property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>True</td>
<td>(Default) Use long file names whenever possible when you save the document as a data access page.</td>
</tr>
<tr>
<td>No</td>
<td>False</td>
<td>Use the 8.3 DOS filename format.</td>
</tr>
</tbody>
</table>

The UseLongFileNames property is available only by using Visual Basic.

If you don't use long file names and your data access page has supporting files, Microsoft Access automatically organizes those files in a separate folder. Otherwise, use the OrganizeInFolder property to determine whether supporting files are organized in a separate folder.
Example

This example cancels the use of long file names as the global default for the application.

Application.DefaultWebOptions.UseLongFileNames = False
**UserControl Property**

You can use the **UserControl** property to determine whether the current Microsoft Access application was started by the user or by another application with **Automation**, formerly called OLE Automation. Read/write **Boolean**.

*expression*.UserControl

*expression* Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **UserControl** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>The current application was started by the user.</td>
</tr>
<tr>
<td>False</td>
<td>The current application was started by another application with Automation.</td>
</tr>
</tbody>
</table>

You can determine the **UserControl** property setting only by using **Visual Basic**.

This property is read-only in all views when user starts the Access application. If Microsoft Access is started by OLE Automation, the **UserControl** property can be set in Visual Basic.

When an application is launched by the user, the **Visible** and **UserControl** properties of the **Application** object are both set to True. When the **UserControl** property is set to True, it isn't possible to set the **Visible** property of the object to False.

When an **Application** object is created by using Automation, the **Visible** and **UserControl** properties of the object are both set to False.
**Example**

The following example displays a message indicating whether Access was started by the user.

MsgBox "The user started Access: " & Application.**UserController**
ValidationRule Property

You can use the ValidationRule property to specify requirements for data entered into a record, field, or control. When data is entered that violates the ValidationRule setting, you can use the ValidationText property to specify the message to be displayed to the user.

Note  The ValidationRule and ValidationText properties don't apply to check box, option button, or toggle button controls when they are in an option group. They apply only to the option group itself.
Remarks

Enter an expression for the ValidationRule property setting and text for the ValidationText property setting. The maximum length for the ValidationRule property setting is 2048 characters. The maximum length for the ValidationText property setting is 255 characters.

For controls, you can set the ValidationRule property to any valid expression. For field and record validation rules, the expression can't contain user-defined functions, domain aggregate or aggregate functions, the Eval function, or CurrentUser method, or references to forms, queries, or tables. In addition, field validation rules can't contain references to other fields. For records, expressions can include references to fields in that table.

You can set the ValidationRule and ValidationText properties by using:

- The Field Properties section of table Design view (for a field validation rule).
- The property sheet for a table by clicking Properties on the View menu in table Design view (for a record validation rule).
- The property sheet for a control on a form.
- A macro or Visual Basic. In Visual Basic, use a string expression to set these properties.

For table fields and records, you can also set these properties in Visual Basic by using the DAO ValidationRule property.

Microsoft Access automatically validates values based on a field's data type; for example, Microsoft Access doesn't allow text in a numeric field. You can set rules that are more specific by using the ValidationRule property.

If you set the ValidationRule property but not the ValidationText property, Microsoft Access displays a standard error message when the validation rule is violated. If you set the ValidationText property, the text you enter is displayed as the error message.

For example, when a record is added for a new employee, you can enter a ValidationRule property requiring that the value in the employee's StartDate
field fall between the company's founding date and the current date. If the date entered isn't in this range, you can display the ValidationText property message: "Start date is incorrect."

If you create a control by dragging a field from the field list, the field's validation rule remains in effect, although it isn't displayed in the control's ValidationRule property box in the property sheet. This is because a field's validation rule is inherited by a control bound to that field.

Control, field, and record validation rules are applied as follows:

- Validation rules you set for fields and controls are applied when you edit the data and the focus leaves the field or control.
- Validation rules for records are applied when you move to another record.
- If you create validation rules for both a field and a control bound to the field, both validation rules are applied when you edit data and the focus leaves the control.

The following table contains expression examples for the ValidationRule and ValidationText properties.

<table>
<thead>
<tr>
<th>ValidationRule property</th>
<th>ValidationText property</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;&gt; 0</td>
<td>Entry must be a nonzero value.</td>
</tr>
<tr>
<td>&gt; 1000 Or Is Null</td>
<td>Entry must be blank or greater than 1000.</td>
</tr>
<tr>
<td>Like &quot;A????&quot;</td>
<td>Entry must be 5 characters and begin with the letter &quot;A&quot;.</td>
</tr>
<tr>
<td>&gt;= #1/1/96# And &lt;#1/1/97#</td>
<td>Entry must be a date in 1996.</td>
</tr>
<tr>
<td>DLookup(&quot;CustomerID&quot;, &quot;Customers&quot;, &quot;CustomerID = Forms!Customers!CustomerID&quot;) Is Null</td>
<td>Entry must be a unique CustomerID (domain aggregate functions are allowed only for form-level validation).</td>
</tr>
</tbody>
</table>

If you create a validation rule for a field, Microsoft Access doesn't normally allow a Null value to be stored in the field. If you want to allow a Null value, add "Is Null" to the validation rule, as in "<> 8 Or Is Null" and make sure the Required property is set to No.

You can't set field or record validation rules for tables created outside Microsoft
Access (for example, dBASE, Paradox, or SQL Server). For these kinds of tables, you can create validation rules for controls only.
Example

The following example creates a validation rule for a field that allows only values over 65 to be entered. If a number less than 65 is entered, a message is displayed. The properties are set by using the SetFieldValidation function.

Dim strTblName As String, strFldName As String
Dim strValidRule As String
Dim strValidText As String, intX As Integer

strTblName = "Customers"
strFldName = "Age"
strValidRule = ">= 65"
strValidText = "Enter a number greater than or equal to 65."
intX = SetFieldValidation(strTblName, strFldName, 
    strValidRule, strValidText)

Function SetFieldValidation(strTblName As String, 
    strFldName As String, strValidRule As String, 
    strValidText As String) As Integer

    Dim dbs As Database, tdf As TableDef, fld As Field

    Set dbs = CurrentDb
    Set tdf = dbs.TableDefs(strTblName)
    Set fld = tdf.Fields(strFldName)
    fld.ValidationRule = strValidRule
    fld.ValidationText = strValidText

End Function

The next example uses the SetTableValidation function to set record-level validation to ensure that the value in the EndDate field comes after the value in the StartDate field.

Dim strTblName As String, strValidRule As String
Dim strValidText As String
Dim intX As Integer

strTblName = "Employees"
strValidRule = "EndDate > StartDate"
strValidText = "Enter an EndDate that is later than the StartDate."
intX = SetTableValidation(strTblName, strValidRule, strValidText)

Function SetTableValidation(strTblName As String, 
    strValidRule As String, 
    strValidText As String) As Integer

End Function
strValidRule As String, strValidText As String) As Integer

Dim dbs As Database, tdf As TableDef

Set dbs = CurrentDb
Set tdf = dbs.TableDefs(strTblName)
tdf.ValidationRule = strValidRule
tdf.ValidationText = strValidText

End Function
ValidationText Property

Use the ValidationText property to specify a message to be displayed to the user when data is entered that violates a ValidationRule setting for a record, field, or control. Read/write String.

expression.ValidationText

expression Required. An expression that returns one of the objects in the Applies To list.
Remarks

The ValidationRule and ValidationText properties don't apply to check box, option button, or toggle button controls when they are in an option group. They apply only to the option group itself.

Enter an expression for the ValidationRule property setting and text for the ValidationText property setting. The maximum length for the ValidationRule property setting is 2048 characters. The maximum length for the ValidationText property setting is 255 characters.

For controls, you can set the ValidationRule property to any valid expression. For field and record validation rules, the expression can't contain user-defined functions, domain aggregate or aggregate functions, the Eval function, or CurrentUser method, or references to forms, queries, or tables. In addition, field validation rules can't contain references to other fields. For records, expressions can include references to fields in that table.

You can set the ValidationRule and ValidationText properties by using:

- The Field Properties section of table Design view (for a field validation rule).
- The property sheet for a table by clicking Properties on the View menu in table Design view (for a record validation rule).
- The property sheet for a control on a form.
- A macro or Visual Basic. In Visual Basic, use a string expression to set these properties.

For table fields and records, you can also set these properties in Visual Basic by using the DAO ValidationRule property.

Microsoft Access automatically validates values based on a field's data type; for example, Microsoft Access doesn't allow text in a numeric field. You can set rules that are more specific by using the ValidationRule property.
If you set the ValidationRule property but not the ValidationText property, Microsoft Access displays a standard error message when the validation rule is violated. If you set the ValidationText property, the text you enter is displayed as the error message.

For example, when a record is added for a new employee, you can enter a ValidationRule property requiring that the value in the employee's StartDate field fall between the company's founding date and the current date. If the date entered isn't in this range, you can display the ValidationText property message: "Start date is incorrect."

If you create a control by dragging a field from the field list, the field's validation rule remains in effect, although it isn't displayed in the control's ValidationRule property box in the property sheet. This is because a field's validation rule is inherited by a control bound to that field.

Control, field, and record validation rules are applied as follows:

- Validation rules you set for fields and controls are applied when you edit the data and the focus leaves the field or control.

- Validation rules for records are applied when you move to another record.

- If you create validation rules for both a field and a control bound to the field, both validation rules are applied when you edit data and the focus leaves the control.

The following table contains expression examples for the ValidationRule and ValidationText properties.

<table>
<thead>
<tr>
<th>ValidationRule property</th>
<th>ValidationText property</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;&gt; 0</td>
<td>Entry must be a nonzero value.</td>
</tr>
<tr>
<td>&gt; 1000 Or Is Null</td>
<td>Entry must be blank or greater than 1000.</td>
</tr>
<tr>
<td>Like &quot;A?????&quot;</td>
<td>Entry must be 5 characters and begin with the letter &quot;A&quot;.</td>
</tr>
<tr>
<td>&gt;= #1/1/96# And &lt;#1/1/97#</td>
<td>Entry must be a date in 1996.</td>
</tr>
</tbody>
</table>
| DLookup("CustomerID", "Customers", "CustomerID =" | Entry must be a unique CustomerID (domain aggregate functions are allowed


If you create a validation rule for a field, Microsoft Access doesn't normally allow a **Null** value to be stored in the field. If you want to allow a **Null** value, add "Is Null" to the validation rule, as in "<> 8 Or Is Null" and make sure the **Required** property is set to No.

You can't set field or record validation rules for tables created outside Microsoft Access (for example, dBASE, Paradox, or SQL Server). For these kinds of tables, you can create validation rules for controls only.
Example

The following example creates a validation rule for a field that allows only values over 65 to be entered. If a number less than 65 is entered, a message is displayed. The properties are set by using the SetFieldValidation function.

```vba
Dim strTblName As String, strFldName As String
Dim strValidRule As String
Dim strValidText As String, intX As Integer

strTblName = "Customers"
strFldName = "Age"
strValidRule = ">= 65"
strValidText = "Enter a number greater than or equal to 65."
intX = SetFieldValidation(strTblName, strFldName,_
strValidRule, strValidText)

Function SetFieldValidation(strTblName As String, _
    strFldName As String, strValidRule As String, _
    strValidText As String) As Integer
    Dim dbs As Database, tdf As TableDef, fld As Field
    Set dbs = CurrentDb
    Set tdf = dbs.TableDefs(strTblName)
    Set fld = tdf.Fields(strFldName)
    fld.ValidationRule = strValidRule
    fld.ValidationText = strValidText
End Function
```

The next example uses the SetTableValidation function to set record-level validation to ensure that the value in the EndDate field comes after the value in the StartDate field.

```vba
Dim strTblName As String, strValidRule As String
Dim strValidText As String
Dim intX As Integer

strTblName = "Employees"
strValidRule = "EndDate > StartDate"
strValidText = "Enter an EndDate that is later than the StartDate."
intX = SetTableValidation(strTblName, strValidRule, strValidText)

Function SetTableValidation(strTblName As String, _
    strValidRule As String, strValidText As String) As Integer
    Dim dbs As Database, tdf As TableDef, fld As Field
    Set dbs = CurrentDb
    Set tdf = dbs.TableDefs(strTblName)
    Set fld = tdf.Fields(strFldName)
    fld.ValidationRule = strValidRule
    fld.ValidationText = strValidText
```
strValidRule As String, strValidText As String) As Integer

Dim dbs As Database, tdf As TableDef

Set dbs = CurrentDb
Set tdf = dbs.TableDefs(strTblName)
tdf.ValidationRule = strValidRule
tdf.ValidationText = strValidText

End Function
Value Property

You can use the **Value** property to determine or specify if a control is selected, the selected value or option within the control, the text contained in a text box control, or the value of a custom property.

- Check box, option button, and toggle button controls. Determines or specifies whether or not the control is selected.

- Combo box, list box, and option group controls. Determines or specifies which value or option in the control is selected.

- Text box controls. Determines or specifies the text in the text box.

- Tab control. Determines or specifies the selected **Page** object.

- Built-in properties. Determines or specifies the value of a built-in property of an **AccessObject** object.

Read/write **Variant**.

`expression.Value`

**expression** Required. An expression that returns one of the objects in the Applies To list.

**Remarks**

The **Value** property uses the following setting depending on the specified control.

<table>
<thead>
<tr>
<th>Control</th>
<th>Setting</th>
<th>Description</th>
<th>Visual Basic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Check box</td>
<td><strong>True</strong></td>
<td>The check box is selected.</td>
<td>True</td>
</tr>
<tr>
<td></td>
<td><strong>False</strong></td>
<td>(Default) The check box is cleared.</td>
<td>False</td>
</tr>
<tr>
<td>Control Type</td>
<td>Description</td>
<td>Sample Setting</td>
<td></td>
</tr>
<tr>
<td>--------------</td>
<td>---------------------------------------------------------------------------------------------------</td>
<td>----------------</td>
<td></td>
</tr>
<tr>
<td>Combo box</td>
<td>This may or may not be the same as the setting for the <strong>Text</strong> property of the control. The current setting for the <strong>Text</strong> property is what is displayed in the text box portion of the combo box; the <strong>Value</strong> property is set to the <strong>Text</strong> property setting only after this text is saved.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>List box</td>
<td>The value in the bound column for the item selected in the list.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Option button</td>
<td>The option button is selected. <em>(Default) The option button isn't selected.</em></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Option group</td>
<td>The <strong>OptionValue</strong> property setting for the selected control in the group.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Text box</td>
<td>The <strong>Text</strong> property returns the formatted string. The <strong>Text</strong> property may be different than the <strong>Value</strong> property for a text box control. The <strong>Text</strong> property is the current contents of the control. The <strong>Value</strong> property is the saved value of the text box control. The <strong>Text</strong> property is always current while the control has the focus.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Toggle button</td>
<td>The toggle button is pressed in. <em>(Default) The toggle button isn't pressed in.</em></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tab control</td>
<td>The <strong>Value</strong> property of a tab control contains the index number of the current <strong>Page</strong> object. There is one <strong>Page</strong> object for each tab in a tab control. The first <strong>Page</strong> object always has an index number of 0, the second has an index number of 1, and so on.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bound object</td>
<td>The <strong>Value</strong> property for a bound object frame or a bound chart control is set to the value of the field that the control is bound to. Since these</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
frame or chart control fields normally contain OLE objects or chart objects, which are stored as binary data, this value is usually meaningless.

ActiveX control Some ActiveX controls support the Value property. For example, the Value property setting for a Calendar control is the currently selected date in the control. For more information, see the documentation for each ActiveX control.

A Long or string expression representing the value of the custom property. The Value property of a custom property contains the value of the specified custom property of an AccessObject object. For more information about custom properties, see the topics about the AccessObject object, AccessObjectProperty object, and the AccessObjectProperties collection.

Note You can set this property by using a macro or Visual Basic.

- The Value property returns or sets a control's default property, which is the property that is assumed when you don't explicitly specify a property name. In the following example, because the default value of the text box is the value of the Text property, you can refer to its Text property setting without explicitly specifying the name of the property.

Forms!frmCustomers!txtLastName = "Smith"

This means that the following two statements are equivalent.

Forms!frmCustomers!optCreditApproved.Value = True

Forms!frmCustomers!optCreditApproved = True

Note The Value property is not the same as the DefaultValue property, which specifies the value that a property is assigned when a new record is created.
- The **Value** property can also return or set a custom properties' default property, which is the property that is assumed when you don't explicitly specify a property name. This means that the following two statements are equivalent.

```plaintext
CurrentProject.AllForms!optCreditApproved.Value = True

CurrentProject.AllForms!optCreditApproved = True
```
Example

The following example shows how you can call one of two procedures, depending whether the Credit check box on the Customers form is selected or cleared.

Sub PaymentType()
    If Forms!Customers!Credit.Value = False Then
        ProcessCash
    ElseIf Forms!Customers!Credit.Value = True Then
        ProcessCredit
    End If
End Sub
VBE Property

You can use the VBE property to return a reference to the current VBE object and its related properties. The VBE property of the Application object represents the Microsoft Visual Basic for Applications editor. Read-only VBE object.

expression.VBE

description of a variable or VBE expression. Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **VBE** property is available only by using **Visual Basic**.

Once you establish a reference to the **VBE** object, you can access all the properties and methods of the object. You can set a reference to the **VBE** object by clicking **References** on the **Tools** menu while in module **Design view**. Then set a reference to the Microsoft Visual Basic for Applications Extensibility 5.3 Object Library in the **References** dialog box by selecting the appropriate check box. Microsoft Access can set this reference for you if you use a Microsoft Visual Basic for Applications Extensibility 5.3 Object Library constant to set a **VBE** object's property or as an argument to a **VBE** object's method.
Example

This example displays the number of references available for the active project.

MsgBox "Number of References = " & VBE.ActiveVBProject._References.Count
Verb Property

You can use the Verb property to specify the operation to perform when an OLE object is activated, which is permitted when the control's Action property is set to `acOLEActivate`. Read/write Long.

expression.Verb

expression Required. An expression that returns one of the objects in the Applies To list.
Remarks

You can set the **Verb** property by specifying an **Integer** data type value indicating the position of a **verb** in the list of verbs returned by the **ObjectVerbs** property. You can set the **Verb** property to 1 to specify the first verb in the list, you can set it to 2 to specify the second verb in the list, and so on.

You can set the **Verb** property by using the control's **property sheet**, a **macro**, or **Visual Basic**. You can set the default for this property by using the **default control style** or the **DefaultControl** method in Visual Basic.

If you don't use the **ObjectVerbs** property to identify a specific verb, you can set the **Verb** property to one of the following values to indicate the operation to perform. These values specify the standard verbs supported by all objects.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>acOLEVerbPrimary</td>
<td>Performs the default operation for the object.</td>
</tr>
<tr>
<td>acOLEVerbShow</td>
<td>Activates the object for editing.</td>
</tr>
<tr>
<td>acOLEVerbOpen</td>
<td>Opens the object in a separate application window.</td>
</tr>
<tr>
<td>acOLEVerbHide</td>
<td>For embedded objects, hides the application that was used to create the object.</td>
</tr>
</tbody>
</table>

With some applications' objects, you can use these additional values.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>acOLEVerbInPlaceUIActivate</td>
<td>Activates the object for editing within the control. The menus and toolbars of the OLE server become available in the OLE container.</td>
</tr>
<tr>
<td>acOLEVerbInPlaceActivate</td>
<td>Activates the object within the control. The menus and toolbars of the OLE server aren't available in the OLE container.</td>
</tr>
</tbody>
</table>

Each object supports its own set of verbs. For example, many objects support the verbs Edit and Play. You can use the **ObjectVerbs** and **ObjectVerbsCount** properties to find out which verbs are supported by an object.
Microsoft Access automatically uses an object's default verb if the user double-clicks an object for which the \texttt{AutoActivate} property is set to Double-Click.
Example

The following example activates the control "OLEUnbound0" in the form "frmOperations" by opening up the OLE object in its own application window for editing. In this case, "OLEUnbound0" contains a new bitmap image, which is linked to the Microsoft Paint program.

```vba
With Forms.Item("frmOperations").Controls.Item("OLEUnbound0")
    .Action = acOLEActivate
    .Verb = acOLEVerbOpen
End With
```
Version Property

Returns a **String** indicating the version number of the currently installed copy of Microsoft Access. Read-only.

*expression*. **Version**

*expression* Required. An expression that returns one of the objects in the Applies To list.
Example

The following example displays the version and build number of the currently-installed copy of Microsoft Access.

MsgBox "You are currently running Microsoft Access, " & " version " & Application.Version & ", build " & Application.Build & "."
Vertical Property

You can use the Vertical property to set a form control for vertical display and editing or set a report control for vertical display and printing. Read/write Boolean.

expression. Vertical

expression       Required. An expression that returns one of the objects in the Applies To list.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>True</td>
<td>Displays, edits, and prints vertical text.</td>
</tr>
<tr>
<td>No</td>
<td>False</td>
<td>Does not display, edit, or print vertical text.  (default)</td>
</tr>
</tbody>
</table>
Remarks

You can set the **Vertical** property by using the [property sheet](#), a [macro](#), or [Visual Basic](#).

You can specify how vertical text will be displayed, edited, or printed in the control by setting the **Vertical** property. If set to Yes, the starting point for inputting text is the upper right corner of the control (the ending point is the lower left corner of the control). If using full pitch characters, the display and print directions are the same as the control for horizontal text. If using half pitch characters, it shifts 90 degrees to the right. The cursor is also rotated 90 degrees to the right in a vertical text control.

**Note**  Text selection using key combinations is different for vertical text control and horizontal text control. Key combinations and their effect on range selection are described below.

<table>
<thead>
<tr>
<th>Key combination</th>
<th>Selected range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shift+Up</td>
<td>Vertical: One character before the cursor.</td>
</tr>
<tr>
<td></td>
<td>Horizontal: One line before the cursor.</td>
</tr>
<tr>
<td>Shift+Down</td>
<td>Vertical: One character after the cursor.</td>
</tr>
<tr>
<td></td>
<td>Horizontal: One line after the cursor.</td>
</tr>
<tr>
<td>Shift+Right</td>
<td>Vertical: One line after the cursor.</td>
</tr>
<tr>
<td></td>
<td>Horizontal: One character before the cursor.</td>
</tr>
<tr>
<td>Shift+Left</td>
<td>Vertical: One line before the cursor.</td>
</tr>
<tr>
<td></td>
<td>Horizontal: One character after the cursor.</td>
</tr>
</tbody>
</table>
Example

The following example prints a message in the Immediate window indicating whether the control "text13" in the "Product Summary" report displays, edits, or prints vertical text.

Debug.Print "Vertical = " & Reports.Item("Product Summary"). Controls.Item("text13"). Vertical
**VerticalDatasheetGridlineStyle Property**

Returns or sets a **Byte** indicating the line style to use for vertical gridlines on the specified datasheet. Read/write.

*expression*. **VerticalDatasheetGridlineStyle**

*expression*  Required. An expression that returns one of the objects in the Applies To list.
Remarks

Valid values are between zero and seven. Values greater than seven are ignored; negative values or values above 255 cause an error.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Transparent border</td>
</tr>
<tr>
<td>1</td>
<td>Solid</td>
</tr>
<tr>
<td>2</td>
<td>Dashes</td>
</tr>
<tr>
<td>3</td>
<td>Short dashes</td>
</tr>
<tr>
<td>4</td>
<td>Dots</td>
</tr>
<tr>
<td>5</td>
<td>Sparse dots</td>
</tr>
<tr>
<td>6</td>
<td>Dash-dot</td>
</tr>
<tr>
<td>7</td>
<td>Dash-dot-dot</td>
</tr>
</tbody>
</table>

This property is not supported when saving a form as a data access page.
Example

This example sets the vertical gridline style on the first open form to dashes. The form must be set to Datasheet View in order for you to see the change.

Forms(0).VerticalDatasheetGridlineStyle = 2
**ViewChange Property**

Returns or sets a **String** indicating which macro, event procedure, or user-defined function runs when the **ViewChange** event occurs. Read/write.

`expression.ViewChange`

*expression*   Required. An expression that returns one of the objects in the Applies To list.
Remarks

Valid values for this property are "macroname" where macroname is the name of macro, "[Event Procedure]" which indicates the event procedure associated with the ViewChange event for the specified object, or "+functionname()" where functionname is the name of a user-defined function. For a more detailed discussion of event properties, see "Event Properties."
Example

The following example specifies that when the ViewChange event occurs on the first form of the current project, the associated event procedure should run.

Forms(0).ViewChange = "[Event Procedure]"
ViewsAllowed Property

You can use the ViewsAllowed property to specify whether users can switch between Datasheet view and Form view by clicking the Form View or Datasheet View command on the View menu or by clicking the arrow next to the View button and clicking Form View or Datasheet View. Read/write Byte.

expression. ViewsAllowed

expression Required. An expression that returns one of the objects in the Applies To list.
Remarks

The ViewsAllowed property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Both</td>
<td>0</td>
<td>(Default) Users can switch between Form view and Datasheet view.</td>
</tr>
<tr>
<td>Form</td>
<td>1</td>
<td>Users can't switch to Datasheet view from Form view.</td>
</tr>
<tr>
<td>Datasheet</td>
<td>2</td>
<td>Users can't switch to Form view from Datasheet view.</td>
</tr>
</tbody>
</table>

**Note** Design view is always available (unless permissions are set otherwise).

You can set these properties by using the form's property sheet, a macro, or Visual Basic.

The views displayed in the View button list and on the View menu depend on the setting of the ViewsAllowed property. For example, if the ViewsAllowed property is set to Datasheet, Form View is disabled in the View button list and on the View menu.

The combination of these properties creates the following conditions.

<table>
<thead>
<tr>
<th>DefaultView</th>
<th>ViewsAllowed</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single, Continuous Forms, or Datasheet</td>
<td>Both</td>
<td>Users can switch between Form view and Datasheet view.</td>
</tr>
<tr>
<td>Single or Continuous Forms</td>
<td>Form</td>
<td>Users can't switch from Form view to Datasheet view.</td>
</tr>
<tr>
<td>Single or Continuous Forms</td>
<td>Datasheet</td>
<td>Users can switch from Form view to Datasheet view but not back again.</td>
</tr>
<tr>
<td>Datasheet</td>
<td>Form</td>
<td>Users can switch from Datasheet view to Form view but not back again.</td>
</tr>
<tr>
<td>Datasheet</td>
<td>Datasheet</td>
<td>Users can't switch from Datasheet view to Form view.</td>
</tr>
</tbody>
</table>
Example

The following example prints a message in the Immediate window indicating the state of how users can switch between Datasheet view and Form view for the "Switchboard" form.

Debug.Print "ViewsAllowed = " & Forms.Item("Switchboard").ViewsAllowed
Visible Property

When used with the Application object, returns or sets whether a Microsoft Access application is minimized. You can also use the Visible property to show or hide a form, report, form or report section, data access page, or control. This may be useful if you want to maintain access to information on a form without it being visible. For example, you could use the value of a control on a hidden form as the criteria for a query. Read/write Boolean; True if visible/minimized, False if not visible/not minimized.

expression.Visible

expression Required. An expression that returns one of the objects in the Applies To list.
Remarks

For the Application object:

- You can set the Visible property of the Application object only by using Visual Basic. You can set the default for this property by using a control’s default control style or the DefaultControl method in Visual Basic.

- When an application is launched by the user, the Visible and UserControl properties of the Application object are both set to True. When the UserControl property is set to True, it isn't possible to set the Visible property of the object to False.

- When an Application object is created by using Automation (formerly called OLE Automation), the Visible and UserControl properties of the object are both set to False.

For all other objects:

- You can set this property by using the object’s property sheet (for sections and all controls except page breaks), a macro, or Visual Basic.

- For forms, reports, and data access pages, you must set this property by using a macro or Visual Basic.

- For controls, you can set the default for this property by using the default control style or the DefaultControl method in Visual Basic.

- The Visible property has no effect on a column in Datasheet view. To specify whether a column is visible in Datasheet view, use the ColumnHidden property.

- To hide an object when printing, use the DisplayWhen property.

- You can use the Visible property to hide a control on a form or report by including the property in a macro or event procedure that runs when the Current event occurs. For example, you can show or hide a congratulatory message next to a salesperson's monthly sales total in a sales report,
depending on the sales total.
WebOptions Property

You can use the WebOptions property to reference a WebOptions object and its related properties.

expression.WebOptions

expression  Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **WebOptions** property is available by using Visual Basic and is read-only.

Use the **WebOptions** object's properties to set web options for the active data access page.
Example

The following example displays a message indicating whether long filenames will be used for the data access page named "Switchboard".

MsgBox DataAccessPages.Item("Switchboard").WebOptions.UseLongFileNames
**WhatsThisButton Property**

You can use the **WhatsThisButton** property to specify whether a **What's This** button is added to a form's **title bar**. Read/write **Boolean**.

*expression*.**WhatsThisButton**

*expression*  Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **WhatsThisButton** property uses the following values:

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>True</td>
<td>A <strong>What's This</strong> button appears on the title bar.</td>
</tr>
<tr>
<td>No</td>
<td>False</td>
<td>(Default) A <strong>What's This</strong> button doesn't appear on the title bar.</td>
</tr>
</tbody>
</table>

You can set the **WhatsThisButton** property by using the form's property sheet, a macro, or Visual Basic.

This property can be set only in form Design view.

You can't display the **What's This** button on the title bar of a form unless the **MinMaxButtons** property is set to None.

Clicking the **What's This** button on the title bar of a form causes the question-mark mouse pointer to appear. With the question-mark pointer, you can click any control to access its custom Help topic specified by the control's **HelpContextID** property. If the control doesn't have a custom Help topic, the form's custom Help topic is displayed. If neither the form or the control has a custom Help topic, Microsoft Access Help is displayed.
Example

The following example places a **What's This** button on the title bar of the "Switchboard" form. The form must be in form Design view, or else a run-time error will occur.

Forms.Item("Switchboard").MinMaxButtons = 0
Forms.Item("Switchboard").**WhatThisButton** = True
Width Property

You can use the **Height** and **Width** properties to size an object to specific dimensions. Read/write **Integer**.

`expression.Width`

`expression` Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **Width** property applies only to forms and reports, not to form sections and report sections.

Enter a number for the desired width in the current unit of measurement. To use a unit of measurement different from the setting in the **Regional Options** dialog box in Windows Control Panel, specify the unit, such as cm or in (for example, 5 cm or 3 in). The setting for the **Width** property must be a value from 0 to 22 inches (55.87 cm).

You can set this property by using the object's **property sheet**, a **macro**, or **Visual Basic**.

For controls, you can set the default for this property by using the **default control style** or the **DefaultControl** method in Visual Basic.

In Visual Basic, use a **numeric expression** to set the value of this property. Values are expressed in **twips**.

For report controls, you can set the **Width** property when you print or preview a report only by using a macro or an **event procedure** specified in a section's **OnFormat event property** setting.

You can't set this property for an object once the print process has started.

Microsoft Access automatically sets the **Width** property when you create or size a control or when you size a window in **form Design View** or **report Design view**.

The width of forms and reports is measured from the inside of their borders. The width of controls is measured from the center of their borders so controls with different border widths align correctly. The margins for forms and reports are set in the **Page Setup** dialog box, available by clicking **Page Setup** on the **File** menu.

**Note** To set the left and top location of an object, use the **Left** and **Top** properties.
Example

The following code resizes a command button to a 1-inch by 1-inch square button (the default unit of measurement in Visual Basic is twips; 1440 twips equals one inch):

```vbnet
Me!cmdSizeButton.Height = 1440    ' 1440 twips = 1 inch.
Me!cmdSizeButton.Width = 1440
```
WillContinue Property

Determines if the current section will continue on the following page. Read/write Boolean.

expression.WillContinue

expression Required. An expression that returns one of the objects in the Applies To list.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>The current section continues on the following page.</td>
</tr>
<tr>
<td>False</td>
<td>The current section doesn't continue on the following page.</td>
</tr>
</tbody>
</table>
Remarks

You can use this property to determine whether to show or hide certain controls, depending on the value of the property. For example, you may have a hidden label in a page header containing the text "Continued on next page". If the value of the `WillContinue` property is `True`, you can make the hidden label visible.

You can get or set the value of the `WillContinue` property by using a `macro` or `Visual Basic`. 
Example

The following example displays a message box indicating whether the page header for the report "Product Summary" will continue on the following page.

MsgBox Reports("Product Summary").Section("PageHeaderSection").WillContinue
WindowHeight Property

Returns or sets the height of a form, report, or data access page in twips, depending on the object. Read/write Integer for the Form and Report objects; read-only Long for the DataAccessPage object.

expression.WindowHeight

expression  Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **WindowHeight** property is measured from the upper-left corner of the form, report, or data access page to its lower-left corner.

This property setting is available only by using a [macro](#) or [Visual Basic](#).
Example

The following example displays a message box indicating the height of the window (in twips) for the "Switchboard" form.

MsgBox Forms.Item("Switchboard").WindowHeight
WindowLeft Property

Returns an **Integer** indicating the screen position in twips of the left edge of a form or report relative to the left edge of the Microsoft Access window. Read-only.

`expression.WindowLeft`

`expression`  Required. An expression that returns one of the objects in the Applies To list.
Remarks

Use the Move method to change the position of a form or report.
Example

The following example returns the screen position of the top and left edges of the first form in the current project.

With Forms(0)

    MsgBox "The form is " & .WindowLeft _
    & " twips from the left edge of the Access window and " _
    & .WindowTop _
    & " twips from the top edge of the Access window."

End With
WindowTop Property

Returns an Integer indicating the screen position in twips of the top edge of a form or report relative to the top of the Microsoft Access window. Read-only.

expression.WindowTop

expression  Required. An expression that returns one of the objects in the Applies To list.
Remarks

Use the Move method to change the position of a form or report.
Example

The following example returns the screen position of the top and left edges of the first form in the current project.

With Forms(0)

    MsgBox "The form is " & .WindowLeft _
    & " twips from the left edge of the Access window and " _
    & .WindowTop _
    & " twips from the top edge of the Access window."

End With
WindowWidth Property

Returns or sets the width of a form, report, or data access page in twips, depending on the object. Read/write Integer for the Form and Report objects; read-only Long for the DataAccessPage object.

expression.WindowWidth

expression Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **WindowWidth** property is measured from the upper left corner of the form, report, or data access page to its upper-right corner.

This property setting is available only by using a [macro](https://example.com) or [Visual Basic](https://example.com).
Example

The following example displays a message box indicating the width of the window (in twips) for the "Switchboard" form.

MsgBox Forms.Item("Switchboard").WindowWidth
XML Property

Returns a **String** that represents the related XML code for a `smart tag`. Read-only.

*expression*.XML

*expression* Required. An expression that returns a `SmartTag` object.
Activate Event

The Activate event occurs when a form or report receives the focus and becomes the active window.

Private Sub object_Activate()

Object The name of a Form or Report.
Remarks

Note  The Activate event doesn't occur when a form receives focus back from a dialog box, popup, or another form.

To run a macro or event procedure when these events occur, set the **OnActivate**, or **OnDeactivate** property to the name of the macro or to [Event Procedure].

You can make a form or report active by opening it, clicking it or a control on it, or by using the **SetFocus** method in Visual Basic (for forms only).

The Activate event can occur only when a form or report is visible.

The Activate event occurs before the **GotFocus** event; the Deactivate event occurs after the **LostFocus** event.

When you switch between two open forms, the Deactivate event occurs for the form being switched from, and the Activate event occurs for the form being switched to. If the forms contain no visible, enabled controls, the LostFocus event occurs for the first form before the Deactivate event, and the GotFocus event occurs for the second form after the Activate event.

When you first open a form, the following events occur in this order:

Open → Load → Resize → Activate → Current

When you close a form, the following events occur in this order:

Unload → Deactivate → Close
Macro

You can use the ShowToolbar action in an Activate or Deactivate macro to display or hide a custom toolbar. Use the Activate or Deactivate event to display or hide custom toolbars when you have more than one form loaded at a time. If you have only one form loaded at a time, you can use the Open and Close events.

If you use a Deactivate macro to hide a custom toolbar on a form, also make sure to hide the toolbar in response to the Unload event for the form, because the Deactivate event doesn't occur when a form is unloaded. Don't use the GotFocus or LostFocus event to display or hide a custom toolbar on a form, because these events aren't triggered on forms that contain enabled controls.

You can't use the CancelEvent action in an Activate or Deactivate macro.
Example

The following example shows how to display a custom toolbar named CustomToolbar when a form receives the focus and to hide it when the focus moves to a different window.

Private Sub Form_Activate()
    ' Display custom toolbar.
    DoCmd.ShowToolbar "CustomToolbar", acToolbarYes
End Sub

Private Sub Form_Deactivate()
    ' Hide custom toolbar. DoCmd.ShowToolbar "CustomToolbar", acToolbarNo
End Sub
AfterDelConfirm Event

The AfterDelConfirm event occurs after the user confirms the deletions and the records are actually deleted or when the deletions are canceled.

Private Sub Form_AfterDelConfirm(Status As Integer)

Status Indicates whether a record has been deleted. The Status argument can be any of the following intrinsic constants:

Constant Description

acDeleteOK Indicates the deletion was successful.

acDeleteCancel Indicates the deletion was canceled in Visual Basic.

acDeleteUserCancel Indicates the deletion was canceled by the user.

Remarks

To run a macro or event procedure when these events occur, set the AfterDelConfirm property to the name of the macro or to [Event Procedure].

After a record is deleted, it's stored in a temporary buffer.

The AfterDelConfirm event occurs after a record or records are actually deleted or after a deletion or deletions are canceled. If the BeforeDelConfirm event isn't canceled, the AfterDelConfirm event occurs after the Delete Confirm dialog box is displayed. The AfterDelConfirm event occurs even if the BeforeDelConfirm event is canceled. The AfterDelConfirm event procedure returns status information about the deletion. For example, you can use a macro or event procedure associated with the AfterDelConfirm event to recalculate totals affected by the deletion of records.

If you cancel the Delete event, the AfterDelConfirm event does not occur and the Delete Confirm dialog box isn't displayed.

Note The AfterDelConfirm event does not occur and the Delete Confirm dialog
box isn't displayed if you clear the **Record Changes** check box under **Confirm** on the **Edit/Find** tab of the **Options** dialog box, available by clicking **Options** on the **Tools** menu.

By running a macro or an event procedure when the Delete event occurs, you can prevent a record from being deleted or allow a record to be deleted only under certain conditions. You can also use a Delete event to display a dialog box asking whether the user wants to delete a record before it's deleted.

To delete a record, you can click **Delete Record** on the **Edit** menu. This deletes the current record (the record indicated by the **record selector**). You can also click the record selector or click **Select Record** on the **Edit** menu to select the record, and then press the DEL key to delete it. If you click **Delete Record**, the record selector of the current record, or **Select Record**, the **Exit** and **LostFocus** events for the control that has the focus occur. If you've changed any data in the record, the **BeforeUpdate** and **AfterUpdate** events for the record occur before the Exit and LostFocus events. If you click the record selector of a different record, the **Current** event for that record also occurs.

After you delete the record, the focus moves to the next record following the deleted record, and the **Current** event for that record occurs, followed by the **Enter** and **GotFocus** events for the first control in that record.

The **BeforeDelConfirm** event then occurs, just before Microsoft Access displays the **Delete Confirm** dialog box asking you to confirm the deletion. After you respond to the dialog box by confirming or canceling the deletion, the **AfterDelConfirm** event occurs.

You can delete one or more records at a time. The Delete event occurs after each record is deleted. This enables you to access the data in each record before it's actually deleted, and selectively confirm or cancel each deletion in the Delete macro or event procedure. When you delete more than one record, the **Current** event for the record following the last deleted record and the **Enter** and **GotFocus** events for the first control in this record don't occur until all the records are deleted. In other words, a Delete event occurs for each selected record, but no other events occur until all the selected records are deleted. The **AfterDelConfirm** event also does not occur until all the selected records are deleted.
**Macro**

You can use a Delete macro to display the Delete Confirm dialog box asking whether the user wants to delete a record before it's deleted.

You can use a BeforeDelConfirm or AfterDelConfirm macro to respond whenever the user deletes a record. However, the BeforeDelConfirm and AfterDelConfirm macros can't set the argument determining whether the Delete Confirm dialog box is displayed and can't return the argument indicating the status of the deletion, so you typically use event procedures with these events.

You can use the CancelEvent action in a Delete macro to cancel the deletion.

You can use the CancelEvent action in a BeforeDelConfirm macro to cancel the deletion of all the deleted records. If you do this, the Delete Confirm dialog box isn't displayed. However, the AfterDelConfirm event occurs even if you cancel the BeforeDelConfirm event.

You can't use the CancelEvent action in an AfterDelConfirm macro.
**Example**

The following example shows how you can use the BeforeDelConfirm event procedure to suppress the Delete Confirm dialog box and display a custom dialog box when a record is deleted. It also shows how you can use the AfterDelConfirm event procedure to display a message indicating whether the deletion progressed in the usual way or whether it was canceled in Visual Basic or by the user.

```vbnet
Private Sub Form_BeforeDelConfirm(Cancel As Integer, _
    Response As Integer)
    ' Suppress default Delete Confirm dialog box.
    Response = acDataErrContinue
    ' Display custom dialog box.
    If MsgBox("Delete this record?", vbOKCancel) = vbCancel Then
        Cancel = True
    End If
End Sub

Private Sub Form_AfterDelConfirm(Status As Integer)
    Select Case Status
        Case acDeleteOK
            MsgBox "Deletion occurred normally."
        Case acDeleteCancel
            MsgBox "Programmer canceled the deletion."
        Case acDeleteUserCancel
            MsgBox "User canceled the deletion."
    End Select
End Sub
```
AfterFinalRender Event

Occurs after all elements in the specified PivotChart view have been rendered.

Private Sub Form_AfterFinalRender(ByVal drawObject As Object)

drawObject A ChChartDraw object. Use the methods and properties of this object to draw objects on the chart.
Example

The following example demonstrates the syntax for a subroutine that traps the AfterFinalRender event.

Private Sub Form_AfterFinalRender(ByVal drawObject As Object)
    MsgBox "The PivotChart View has fully rendered."
End Sub
AfterInsert Event

The AfterInsert event occurs after a new record is added.

**Note** Setting the value of a [control](#) by using a [macro](#) or Visual Basic doesn't trigger these events.

**Private Sub Form_AfterInsert()**
Remarks

You can use an AfterInsert event procedure or macro to requery a recordset whenever a new record is added.

To run a macro or event procedure when the AfterInsert event occurs, set the OnAfterInsert property to the name of the macro or to [Event Procedure].
**Macro**

You can use these macros to display messages or useful information. For example, you can have the BeforeInsert macro display data the user needs when entering a new record.

You can use the CancelEvent action in a BeforeInsert macro to cancel the creation of the new record. When you use the CancelEvent action in a BeforeInsert macro, the focus returns to the new record, which is blank (the character that the user typed is deleted).

You can't use the CancelEvent action in an AfterInsert macro.
**Example**

This example shows how you can use a *BeforeInsert* event procedure to verify that the user wants to create a new record, and an *AfterInsert* event procedure to requery the record source for the Employees form after a record has been added.

To try the example, add the following event procedure to a form named Employees that is based on a table or query. Switch to form Datasheet view and try to insert a record.

```vba
Private Sub Form_BeforeInsert(Cancel As Integer)
    If MsgBox("Insert new record here?", _
        vbOKCancel) = vbCancel Then
        Cancel = True
    End If
End Sub

Private Sub Form_AfterInsert()
    Forms!Employees.Requery
End Sub
```
AfterLayout Event

Occurs after all charts in the specified PivotChart view have been laid out, but before they have been rendered.

Private Sub Form_AfterLayout(ByVal drawObject As Object)

drawObject  A ChChartDraw object. Use the methods and properties of this object to draw objects on the chart.
Remarks

During this event, you can reposition the ChTitle, ChLegend, ChChart, and ChAxis objects of each PivotChart view by changing their Left and Top properties. You can reposition the ChPlotArea object by changing its Left, Top, Right, and Bottom properties. These properties cannot be changed outside of this event.
Example

The following example demonstrates the syntax for a subroutine that traps the AfterLayout event.

Private Sub Form_AfterLayout(ByVal drawObject As Object)
    MsgBox "The PivotChart view has been laid out."
End Sub
**AfterRender Event**

Occurs after the object represented by the `chartObject` argument has been rendered.

**Private Sub Form_AfterRender(ByVal drawObject As Object, ByVal chartObject As Object)**

- **drawObject**  A `ChChartDraw` object. Use the methods and properties of this object to draw objects on the chart.

- **chartObject**  The object that has just been rendered. Use the `TypeName` function to determine what type of object has just been rendered.
Example

The following example demonstrates the syntax for a subroutine that traps the AfterRender event.

Private Sub Form_AfterRender(_
    ByVal drawObject As Object, ByVal chartObject As Object)
    MsgBox TypeName(chartObject) & " has been rendered."
End Sub
AfterUpdate Event

The AfterUpdate event occurs after changed data in a control or record is updated.

Private Sub object_AfterUpdate()

Object The name of a form or a control.
Remarks

Notes

- Changing data in a control by using Visual Basic or a macro containing the SetValue action doesn't trigger these events for the control. However, if you then move to another record or save the record, the form's AfterUpdate event does occur.
- The AfterUpdate event applies only to controls on a form, not controls on a report.
- This event does not apply to option buttons, check boxes, or toggle buttons in an option group. It applies only to the option group itself.

To run a macro or event procedure when this event occurs, set the AfterUpdate property to the name of the macro or to [Event Procedure].

The AfterUpdate event is triggered when a control or record is updated. Within a record, changed data in each control is updated when the control loses the focus or when the user presses ENTER or TAB. When the focus leaves the record or if the user clicks Save Record on the Records menu, the entire record is updated, and the data is saved in the database.

When you enter new or changed data in a control on a form and then move to another record or save the record by clicking Save Record on the Records menu, the AfterUpdate event for the form occur immediately after the AfterUpdate event for the control. When you move to a different record, the Exit and LostFocus events for the control occur, followed by the Current event for the record you moved to, and the Enter and GotFocus events for the first control in this record. To run the AfterUpdate macro or event procedure without running the Exit and LostFocus macros or event procedures, save the record by using the Save Record command on the Records menu.

AfterUpdate macros and event procedures run only if you change the data in a control. This event does not occur when a value changes in a calculated control. AfterUpdate macros and event procedures for a form run only if you change the data in one or more controls in the record.

For bound controls, the OldValue property isn't set to the updated value until
after the AfterUpdate event for the form occurs. Even if the user enters a new value in the control, the **OldValue** property setting isn't changed until the data is saved (the record is updated). If you cancel an update, the value of the **OldValue** property replaces the existing value in the control.

**Note** To perform simple validations, or more complex validations such as requiring a value in a field or validating more than one control on a form, you can use the **ValidationRule** property for controls and the **ValidationRule** and **Required** properties for fields and records in tables.
**Macro**

You can use BeforeUpdate macros if you want to store all the validations for a form in one macro group or if you use a validation expression on several forms.

For example, suppose you have a form on which the user enters the name of a country in a Country control and its corresponding postal code in a Postal Code control. You use conditional expressions in the macro to make sure that the data entered in the Country control corresponds to the data entered in the Postal Code control and to display a message if invalid data is entered in the Postal Code control.

You can use an AfterUpdate macro to display a different page in the form or move the focus to a particular control or record. For example, when the user enters a value in a Category control on a form, you can use the GoToPage action in an AfterUpdate macro to move the focus to the page of the form that contains the controls for that category.

You can use the CancelEvent action in a BeforeUpdate macro to cancel the update. If the BeforeUpdate macro is attached to a control, the CancelEvent action returns the focus to the control, and the data the user entered remains in the control.

If a BeforeUpdate macro is attached to a form (in which it runs each time a record is changed), the CancelEvent action returns the focus to the record.

You can't use the CancelEvent action in an AfterUpdate macro to cancel the AfterUpdate event.
ApplyFilter Event

The ApplyFilter event can occur within a Microsoft Access project (.adp) or Access database (.mdb).

Within an Access database, an ApplyFilter event occurs when the user does one of the following:

- Clicks **Apply Filter/Sort** on the **Records** menu in **Form view**, clicks **Apply Filter/Sort** on the **Filter** menu in the Filter window, or clicks **Apply Filter** on the toolbar. This applies the most recently created **filter** (created by using either the **Filter By Form** feature or the **Advanced Filter/Sort** window).

- On the **Records** menu in Form view, points to **Filter** and clicks **Filter By Selection**, or clicks **Filter By Selection** on the toolbar. This applies a filter based on the current selection in the form.

- On the **Records** menu in Form view, points to **Filter** and clicks **Filter Excluding Selection**. This applies a filter excluding the current selection in the form.

- Clicks **Remove Filter/Sort** on the **Records** menu in Form view, or clicks **Remove Filter** on the toolbar. This removes any filter (or sort) currently applied to the form.

  **Note** On subforms, the ApplyFilter event does not occur when the user removes a filter. In this case, the ApplyFilter event occurs on the parent form.

- Clicks **Filter By Selection**, **Filter Excluding Selection**, or **Remove Filter/Sort** or enters a value or **expression** in the **Filter For** box on the shortcut menu when a **bound control** has the **focus**.
• Closes the Advanced Filter/Sort window or the Filter By Form window.

• Clicks **Advanced Filter/Sort** on the **Filter** menu while the Filter By Form window is open, or clicks **Filter By Form** on the **Filter** menu while the Advanced Filter/Sort window is open. This causes the ApplyFilter event to occur when the open filter window is closed, and then the **Filter** event to occur when the other filter window is opened.

Within an Access project, an ApplyFilter event occurs when the user does one of the following:

• Clicks **Apply Filter/Sort** on the **Records** menu in Form view, clicks **Apply Filter/Sort** on the **Filter** menu in the Filter window, or clicks **Apply Filter** on the toolbar. This applies the most recently created filter (created by using the Filter By Form feature).

• Clicks **Apply Server Filter** on the **Records** menu in Form view, clicks **Apply Server Filter** on the **Filter** menu in the Filter window, or clicks **Apply Server Filter** on the toolbar. This applies the most recently created filter (created by using the Server Filter By Form feature).

• On the **Records** menu in Form view, points to **Filter** and clicks **Filter By Selection**, or clicks **Filter By Selection** on the toolbar. This applies a filter based on the current selection in the form.

• On the **Records** menu in Form view, points to **Filter** and clicks **Filter Excluding Selection**. This applies a filter excluding the current selection in the form.

• Clicks **Filter By Selection** or **Filter Excluding Selection** or enters a value or expression in the **Filter For** box on the shortcut menu when a bound control has the focus.
Remarks

To run a macro or event procedure when this event occurs, set the OnApplyFilter property to the name of the macro or to [Event Procedure].

You can use the ApplyFilter event to:

- Make sure the filter that is being applied is correct. For example, you may want to be sure that any filter applied to an Orders form includes criteria restricting the OrderDate field. To do this, check the form's Filter or ServerFilter property value to make sure this criteria is included in the WHERE clause expression.

- Change the display of the form before the filter is applied. For example, when you apply a certain filter, you may want to disable or hide some fields that aren't appropriate for the records displayed by this filter.

- Undo or change actions you took when the Filter event occurred. For example, you can disable or hide some controls on the form when the user is creating the filter, because you don't want these controls to be included in the filter criteria. You can then enable or show these controls after the filter is applied.

The actions in the ApplyFilter macro or event procedure occur before the filter is applied or removed; or after the Advanced Filter/Sort, Filter By Form, or Server Filter By Form window is closed, but before the form is redisplayed. The criteria you've entered in the newly created filter are available to the ApplyFilter macro or event procedure as the setting of the Filter or ServerFilter property.

Note The ApplyFilter event doesn't occur when the user does one of the following:

- Applies or removes a filter by using the ApplyFilter, OpenForm, or ShowAllRecords actions in a macro, or their corresponding methods of the DoCmd object in Visual Basic.

- Uses the Close action or the Close method of the DoCmd object to close the Advanced Filter/Sort, Filter By Form, or Server Filter By Form window.
• Sets the **Filter** or **ServerFilter** property or **FilterOn** or **ServerFilterByForm** property in a macro or Visual Basic (although you can set these properties in an ApplyFilter macro or event procedure).
Example

The following example shows how to hide the AmountDue, Tax, and TotalDue controls on an Orders form when the applied filter restricts the records to only those orders that have been paid for.

To try this example, add the following event procedure to an Orders form that contains AmountDue, Tax, and TotalDue controls. Run a filter that lists only those orders that have been paid for.

```vbscript
Private Sub Form_ApplyFilter(Cancel As Integer, ApplyType As Integer)
    If Not IsNull(Me.Filter) And (InStr(Me.Filter, "Orders.Paid = -1") > 0 Or InStr(Me.Filter, "Orders.Paid = True") > 0) Then
        If ApplyType = acApplyFilter Then
            Forms!Orders!AmountDue.Visible = False
            Forms!Orders!Tax.Visible = False
            Forms!Orders!TotalDue.Visible = False
        ElseIf ApplyType = acShowAllRecords Then
            Forms!Orders!AmountDue.Visible = True
            Forms!Orders!Tax.Visible = True
            Forms!Orders!TotalDue.Visible = True
        End If
    End If
End Sub
```
BeforeDelConfirm Event

The BeforeDelConfirm event occurs after the user deletes to the buffer one or more records, but before Microsoft Access displays a dialog box asking the user to confirm the deletions.

Private Sub Form_BeforeDeleteConfirm(Cancel As Integer, Response As Integer)

Cancel The setting determines if the BeforeDelConfirm event occurs. Setting the Cancel argument to True cancels the BeforeDelConfirm event and prevents the Delete Confirm dialog box from being displayed. If the event is canceled, the original records are restored, but the AfterDelConfirm event still occurs. If Cancel is set to True, the Response argument is ignored. If Cancel is set to False (0), which it is by default, the value in the Response argument is used by Microsoft Access to determine the type of response to the delete event.

Response The setting determines whether Microsoft Access displays the Delete Confirm dialog box asking if the record should be deleted. The Response argument can be set to one of the following intrinsic constants:

Constant Description

acDataErrContinue Continues without displaying the Delete Confirm dialog box. Setting the Cancel argument to False and the Response argument to acDataErrContinue enables Microsoft Access to delete records without prompting the user.

acDataErrDisplay (Default) Displays the Delete Confirm dialog box.
Remarks

To run a macro or event procedure when these events occur, set the **BeforeDelConfirm** property to the name of the macro or to [Event Procedure].

After a record is deleted, it's stored in a temporary buffer. The BeforeDelConfirm event occurs after the Delete event (or if you've deleted more than one record, after all the records are deleted, with a Delete event occurring for each record), but before the **Delete Confirm** dialog box is displayed. Canceling the BeforeDelConfirm event restores the record or records from the buffer and prevents the **Delete Confirm** dialog box from being displayed.

The AfterDelConfirm event occurs after a record or records are actually deleted or after a deletion or deletions are canceled. If the BeforeDelConfirm event isn't canceled, the AfterDelConfirm event occurs after the **Delete Confirm** dialog box is displayed. The AfterDelConfirm event occurs even if the BeforeDelConfirm event is canceled.

If you cancel the Delete event, the BeforeDelConfirm event does not occur and the **Delete Confirm** dialog box isn't displayed.

**Note** The BeforeDelConfirm event does not occur and the **Delete Confirm** dialog box isn't displayed if you clear the **Record Changes** check box under **Confirm** on the **Edit/Find** tab of the **Options** dialog box, available by clicking **Options** on the **Tools** menu.

By running a macro or an event procedure when the Delete event occurs, you can prevent a record from being deleted or allow a record to be deleted only under certain conditions. You can also use a Delete event to display a dialog box asking whether the user wants to delete a record before it's deleted.

To delete a record, you can click **Delete Record** on the **Edit** menu. This deletes the current record (the record indicated by the **record selector**). You can also click the record selector or click **Select Record** on the **Edit** menu to select the record, and then press the DEL key to delete it. If you click **Delete Record**, the record selector of the current record, or **Select Record**, the **Exit** and **LostFocus** events for the control that has the focus occur. If you've changed any data in the record, the **BeforeUpdate** and **AfterUpdate** events for the record occur before the
Exit and LostFocus events. If you click the record selector of a different record, the Current event for that record also occurs.

After you delete the record, the focus moves to the next record following the deleted record, and the Current event for that record occurs, followed by the Enter and GotFocus events for the first control in that record.

The BeforeDelConfirm event then occurs, just before Microsoft Access displays the Delete Confirm dialog box asking you to confirm the deletion. After you respond to the dialog box by confirming or canceling the deletion, the AfterDelConfirm event occurs.

You can delete one or more records at a time. The Delete event occurs after each record is deleted. This enables you to access the data in each record before it's actually deleted, and selectively confirm or cancel each deletion in the Delete macro or event procedure. When you delete more than one record, the Current event for the record following the last deleted record and the Enter and GotFocus events for the first control in this record don’t occur until all the records are deleted. In other words, a Delete event occurs for each selected record, but no other events occur until all the selected records are deleted. The BeforeDelConfirm event does not occur until all the selected records are deleted.
**Macro**

You can use a Delete macro to display the Delete Confirm dialog box asking whether the user wants to delete a record before it's deleted.

You can use a BeforeDelConfirm or AfterDelConfirm macro to respond whenever the user deletes a record. However, the BeforeDelConfirm and AfterDelConfirm macros can't set the argument determining whether the Delete Confirm dialog box is displayed and can't return the argument indicating the status of the deletion, so you typically use event procedures with these events.

You can use the CancelEvent action in a Delete macro to cancel the deletion.

You can use the CancelEvent action in a BeforeDelConfirm macro to cancel the deletion of all the deleted records. If you do this, the Delete Confirm dialog box isn't displayed. However, the AfterDelConfirm event occurs even if you cancel the BeforeDelConfirm event.

You can't use the CancelEvent action in an AfterDelConfirm macro.
Example

The following example shows how you can use the BeforeDelConfirm event procedure to suppress the Delete Confirm dialog box and display a custom dialog box when a record is deleted. It also shows how you can use the AfterDelConfirm event procedure to display a message indicating whether the deletion progressed in the usual way or whether it was canceled in Visual Basic or by the user.

Private Sub Form_BeforeDelConfirm(Cancel As Integer, _
    Response As Integer)
    ' Suppress default Delete Confirm dialog box.
    Response = acDataErrContinue
    ' Display custom dialog box.
    If MsgBox("Delete this record?", vbOKCancel) = vbCancel Then
        Cancel = True
    End If
End Sub

Private Sub Form_AfterDelConfirm(Status As Integer)
Select Case Status
    Case acDeleteOK
        MsgBox "Deletion occurred normally."
    Case acDeleteCancel
        MsgBox "Programmer canceled the deletion."
    Case acDeleteUserCancel
        MsgBox "User canceled the deletion."
End Select
End Sub
BeforeInsert Event

The BeforeInsert event occurs when the user types the first character in a new record, but before the record is actually created.

Private Sub Form_BeforeInsert(Cancel As Integer)

Cancel The setting determines if the BeforeInsert event occurs. Setting the Cancel argument to True (–1) cancels the BeforeInsert event.
Remarks

**Note** Setting the value of a control by using a macro or Visual Basic doesn't trigger these events.

To run a macro or event procedure when these events occur, set the **BeforeInsert** or **AfterInsert** property to the name of the macro or to [Event Procedure].

You can use an AfterInsert event procedure or macro to query a recordset whenever a new record is added.

The BeforeInsert and AfterInsert events are similar to the **BeforeUpdate** and **AfterUpdate** events. These events occur in the following order:

BeforeInsert → BeforeUpdate → AfterUpdate → AfterInsert.

The following table summarizes the interaction between these events.

<table>
<thead>
<tr>
<th>Event</th>
<th>Occurs when</th>
</tr>
</thead>
<tbody>
<tr>
<td>BeforeInsert</td>
<td>User types the first character in a new record.</td>
</tr>
<tr>
<td>BeforeUpdate</td>
<td>User updates the record.</td>
</tr>
<tr>
<td>AfterUpdate</td>
<td>Record is updated.</td>
</tr>
<tr>
<td>AfterInsert</td>
<td>Record updated is a new record.</td>
</tr>
</tbody>
</table>

If the first character in a new record is typed into a text box or combo box, the BeforeInsert event occurs before the Change event.
**Macro**

You can use these macros to display messages or useful information. For example, you can have the BeforeInsert macro display data the user needs when entering a new record.

You can use the CancelEvent action in a BeforeInsert macro to cancel the creation of the new record. When you use the CancelEvent action in a BeforeInsert macro, the focus returns to the new record, which is blank (the character that the user typed is deleted).

You can't use the CancelEvent action in an AfterInsert macro.
Example

This example shows how you can use a **BeforeInsert** event procedure to verify that the user wants to create a new record, and an **AfterInsert** event procedure to requery the record source for the Employees form after a record has been added.

To try the example, add the following event procedure to a form named Employees that is based on a table or query. Switch to form Datasheet view and try to insert a record.

```vbnet
Private Sub Form_BeforeInsert(Cancel As Integer)
    If MsgBox("Insert new record here?", _
        vbOKCancel) = vbCancel Then
        Cancel = True
    End If
End Sub

Private Sub Form_AfterInsert()
    Forms!Employees.Requery
End Sub
```
BeforeQuery Event

Occurs when the specified PivotTable view queries its data source.

Private Sub Form_BeforeQuery()
Remarks

This event occurs quite frequently. Some examples of actions that trigger this event include adding fields to the PivotTable view, moving fields, sorting, or filtering data.
Example

The following example demonstrates the syntax for a subroutine that traps the BeforeQuery event.

Private Sub Form_BeforeQuery()
    MsgBox "The PivotTable view is about to query its data source."
End Sub
BeforeRender Event

Occurs before any object in the specified PivotChart view has been rendered.

Private Sub Form_BeforeRender(ByVal drawObject As Object, ByVal chartObject As Object, ByVal Cancel As Object)

drawObject    A reference to the ChChartDraw object. Use the DrawType property of the returned object to determine what type of rendering is about to occur.

chartObject   The object that is to be rendered. Use the TypeName function to determine the type of the object.

Cancel        Set the Value property of this object to True to cancel the rendering of the PivotChart View object.
Example

The following example demonstrates the syntax for a subroutine that traps the BeforeRender event.

Private Sub Form_BeforeRender(_
    ByVal drawObject As Object, _
    ByVal chartObject As Object, _
    ByVal Cancel As Object)
Dim intResponse As Integer
Dim strPrompt As String

strPrompt = "Cancel render of current object?"

intResponse = MsgBox(strPrompt, vbYesNo)

If intResponse = vbYes Then
    Cancel.Value = True
Else
    Cancel.Value = False
End If
End Sub
**BeforeScreenTip Event**

Occurs before a ScreenTip is displayed for an element in a PivotChart view or PivotTable view.

**Private Sub Form_BeforeScreenTip(ScreenTipText As Object, ByVal SourceObject As Object)**

*ScreenTipText*  Set the Value property of this object to the ScreenTip that you want to display. Changing this argument to an empty string effectively hides the ScreenTip.

*SourceObject*  The object that generates the ScreenTip.
BeforeUpdate Event

The BeforeUpdate event occurs before changed data in a control or record is updated.

Private Sub object_BeforeUpdate(Cancel As Integer)

Object  The name of a form or a control.

Cancel   The setting determines if the BeforeUpdate event occurs. Setting the Cancel argument to True (–1) cancels the BeforeUpdate event.
Remarks

Changing data in a control by using Visual Basic or a macro containing the SetValue action doesn't trigger these events for the control. However, if you then move to another record or save the record, the form's BeforeUpdate event does occur.

The BeforeUpdate event applies only to controls on a form, not controls on a report.

The BeforeUpdate event does not apply to option buttons, check boxes, or toggle buttons in an option group. It applies only to the option group itself.

Remarks

To run a macro or event procedure when these events occur, set the BeforeUpdate property to the name of the macro or to [Event Procedure].

The BeforeUpdate event is triggered when a control or record is updated. Within a record, changed data in each control is updated when the control loses the focus or when the user presses ENTER or TAB. When the focus leaves the record or if the user clicks Save Record on the Records menu, the entire record is updated, and the data is saved in the database.

When you enter new or changed data in a control on a form and then move to another record or save the record by clicking Save Record on the Records menu, the BeforeUpdate event for the form occurs immediately after the BeforeUpdate event for the control. When you move to a different record, the Exit and LostFocus events for the control occur, followed by the Current event for the record you moved to, and the Enter and GotFocus events for the first control in this record. To run the BeforeUpdate macros or event procedures without running the Exit and LostFocus macros or event procedures, save the record by using the Save Record command on the Records menu.

BeforeUpdate macro and event procedures run only if you change the data in a control. These events don't occur when a value changes in a calculated control. BeforeUpdate macro and event procedures for a form run only if you change the data in one or more controls in the record.
For forms, you can use the BeforeUpdate event to cancel updating of a record before moving to another record.

If the user enters a new value in the control, the OldValue property setting isn't changed until the data is saved (the record is updated). If you cancel an update, the value of the OldValue property replaces the existing value in the control.

You often use the BeforeUpdate event to validate data, especially when you perform complex validations, such as those that:

- Involve conditions for more than one value on a form.
- Display different error messages for different data entered.
- Can be overridden by the user.
- Contain references to controls on other forms or contain user-defined functions.

Note To perform simple validations, or more complex validations such as requiring a value in a field or validating more than one control on a form, you can use the ValidationRule property for controls and the ValidationRule and Required properties for fields and records in tables.

A run-time error will occur if you attempt to modify the data contained in the control that fired the BeforeUpdate event in the event's procedure.
**Macro**

You can use BeforeUpdate macros if you want to store all the validations for a form in one macro group or if you use a validation expression on several forms.

For example, suppose you have a form on which the user enters the name of a country in a Country control and its corresponding postal code in a Postal Code control. You use conditional expressions in the macro to make sure that the data entered in the Country control corresponds to the data entered in the Postal Code control and to display a message if invalid data is entered in the Postal Code control.

You can use an AfterUpdate macro to display a different page in the form or move the focus to a particular control or record. For example, when the user enters a value in a Category control on a form, you can use the GoToPage action in an AfterUpdate macro to move the focus to the page of the form that contains the controls for that category.

You can use the CancelEvent action in a BeforeUpdate macro to cancel the update. If the BeforeUpdate macro is attached to a control, the CancelEvent action returns the focus to the control, and the data the user entered remains in the control.

If a BeforeUpdate macro is attached to a form (in which it runs each time a record is changed), the CancelEvent action returns the focus to the record.

You can't use the CancelEvent action in an AfterUpdate macro to cancel the AfterUpdate event.
Example
The following example shows how you can use a BeforeUpdate event procedure
to check whether a product name has already been entered in the database. After
the user types a product name in the ProductName box, the value is compared to
the ProductName field in the Products table. If there is a matching value in the
Products table, a message is displayed that informs the user that the product has
already been entered.
To try the example, add the following event procedure to a form named Products
that contains a text box called ProductName.
Private Sub ProductName_BeforeUpdate(Cancel As Integer)
If(Not IsNull(DLookup("[ProductName]", _
"Products", "[ProductName] ='" _
& Me!ProductName & "'"))) Then
MsgBox "Product has already been entered in the database."
Cancel = True
Me!ProductName.Undo
End If
End Sub


Change Event

The Change event occurs when the contents of a text box or the text portion of a combo box changes. It also occurs when you move from one page to another page in a tab control.

Private Sub object_Change()

Object The name of a control.
**Remarks**

Examples of this event include entering a character directly in the text box or combo box or changing the control's **Text** property setting by using a **macro** or Visual Basic.

**Notes**

- Setting the value of a **control** by using a macro or Visual Basic doesn't trigger this event for the control. You must type the data directly into the control, or set the control's **Text** property.
- This event applies only to controls on a **form**, not controls on a **report**.

To run a macro or **event procedure** when this event occurs, set the **OnChange** property to the name of the macro or to [Event Procedure].

By running a macro or event procedure when a Change event occurs, you can coordinate data display among controls. You can also display data or a formula in one control and the results in another control.

The Change event doesn't occur when a value changes in a calculated control.

**Note** A Change event can cause a **cascading event**. This occurs when a macro or event procedure that runs in response to the control's Change event alters the control's contents— for example, by changing a property setting that determines the control's value, such as the **Text** property for a text box. To prevent a cascading event:

- If possible, avoid attaching a Change macro or event procedure to a control that alters the control's contents.

- Avoid creating two or more controls having Change events that affect each other— for example, two text boxes that update each other.

Changing the data in a text box or combo box by using the keyboard causes keyboard events to occur in addition to control events like the Change event. For example, if you move to a new record and type an **ANSI** character in a text box in the record, the following events occur in this order:
KeyDown → KeyPress → BeforeInsert → Change → KeyUp

The **BeforeUpdate** and **AfterUpdate** events for the text box or combo box control occur after you have entered the new or changed data in the control and moved to another control (or clicked **Save Record** on the **Records** menu), and therefore after all of the Change events for the control.

In combo boxes for which the **LimitToList** property is set to Yes, the **NotInList** event occurs after you enter a value that isn't in the list and attempt to move to another control or save the record. It occurs after all the Change events for the combo box. In this case, the BeforeUpdate and AfterUpdate events for the combo box don't occur, because Microsoft Access doesn't accept a value that is not in the list.
Click Event

The Click event occurs when the user presses and then releases a mouse button over an object.

Private Sub object_Change()

Object The name of a Form or control.
Remarks

- The Click event applies only to forms, form sections, and controls on a form, not controls on a report.
- This event doesn't apply to check boxes, option buttons, or toggle buttons in an option group. It applies only to the option group itself.
- This event doesn't apply to a label attached to another control, such as the label for a text box. It applies only to "freestanding" labels. Clicking an attached label has the same effect as clicking the associated control. The normal events for the control occur, not any events for the attached label.
- This event applies to a control containing a hyperlink.

To run a macro or event procedure when this event occurs, set the OnClick property to the name of the macro or to [Event Procedure].

On a form, this event occurs when the user clicks a blank area or record selector on the form.

For a control, this event occurs when the user:

- Clicks a control with the left mouse button. Clicking a control with the right or middle mouse button does not trigger this event.

- Clicks a control containing hyperlink data with the left mouse button. Clicking a control with the right or middle mouse button does not trigger this event. When the user moves the mouse pointer over a control containing hyperlink data, the mouse pointer changes to a "hand" icon. When the user clicks the mouse button, the hyperlink is activated, and then the Click event occurs.

- Selects an item in a combo box or list box, either by pressing the arrow keys and then pressing the ENTER key or by clicking the mouse button.

- Presses SPACEBAR when a command button, check box, option button, or toggle button has the focus.

- Presses the ENTER key on a form that has a command button whose Default property is set to Yes.
• Presses the ESC key on a form that has a command button whose **Cancel** property is set to Yes.

• Presses a control's **access key**. For example, if a command button's **Caption** property is set to &Go, pressing ALT+G triggers the event.

Typically, you attach a Click **event procedure** or **macro** to a command button to carry out commands and command-like actions. For the other applicable controls, use this event to trigger actions in response to one of the occurrences discussed earlier in this topic.

For a command button only, Microsoft Access runs the macro or event procedure specified by the **OnClick** property when the user chooses the command button by pressing the ENTER key or an access key. The macro or event procedure runs once. If you want the macro or event procedure to run repeatedly while the command button is pressed, set its **AutoRepeat** property to Yes. For other types of controls, you must click the control by using the mouse button to trigger the Click event.

The Click event for a command button occurs when you choose the command button. In addition, if the command button doesn't already have the focus when you choose it, the **Enter** and **GotFocus** events for the command button occur before the Click event.

Double-clicking a control causes both the **DblClick** and Click events to occur. For command buttons, double-clicking triggers the following events, in this order:

MouseDown → MouseUp → Click → DblClick → Click

You can use a **CancelEvent** action in a DblClick macro to cancel the second Click event. For more information, see the DblClick event topic.

The Click event for an option group occurs after you change the value of one of the controls in the option group by clicking the control. For example, if you click a toggle button, option button, or check box in an option group, the Click event for the option group occurs after the **BeforeUpdate** and **AfterUpdate** events for the option group.
Tip

To distinguish between the left, right, and middle mouse buttons, use the MouseDown and MouseUp events.
Example

In the following example, the Click event procedure is attached to the ReadOnly check box. The event procedure sets the Enabled and Locked properties of another control on the form, the Amount text box. When the check box is clicked, the event procedure checks whether the check box is being selected or cleared and then sets the text box's properties to enable or disable editing accordingly.

To try the example, add the following event procedure to a form that contains a check box called ReadOnly and a text box named Amount.

```vba
Private Sub ReadOnly_Click()
    With Me!Amount
        If Me!ReadOnly = True Then
            .Enabled = False
            .Locked = True
        Else
            .Enabled = True
            .Locked = False
        End If
    End With
End Sub
```
Close Event

The Close event occurs when a form or report is closed and removed from the screen.

Private Sub object_Close()

Object    The name of a Form or Report.
Remarks

To run a macro or event procedure when this event occurs, set the OnClose property to the name of the macro or to [Event Procedure].

The Close event occurs after the Unload event, which is triggered after the form is closed but before it is removed from the screen.

When you close a form, the following events occur in this order:

Unload → Deactivate → Close

When the Close event occurs, you can open another window or request the user's name to make a log entry indicating who used the form or report.

The Unload event can be canceled, but the Close event can't.
CommandBeforeExecute Event

Occurs before a specified command is executed. Use this event when you want to impose certain restrictions before a particular command is executed.

Private Sub Form_CommandBeforeExecute(ByVal Command As Variant, ByVal Cancel As Object)

Command  The command that is going to be executed.

Cancel   Set the Value property of this object to True to cancel the command.
Remarks

The **OCCommandId**, **ChartCommandIdEnum**, and **PivotCommandId** constants contain lists of the supported commands for each of the Microsoft Office Web Components.
Example

The following example demonstrates the syntax for a subroutine that traps the CommandBeforeExecute event.

Private Sub Form_CommandBeforeExecute(ByVal Command As Variant, ByVal Cancel As Object)
    Dim intResponse As Integer
    Dim strPrompt As String

    strPrompt = "Cancel the command?"

    intResponse = MsgBox(strPrompt, vbYesNo)

    If intResponse = vbYes Then
        Cancel.Value = True
    Else
        Cancel.Value = False
    End If
End Sub
CommandChecked Event

Occurs when the specified Microsoft Office Web Component determines whether the specified command is checked.

Private Sub Form_CommandChecked(ByVal Command As Variant, ByVal Checked As Object)

Command    The command that has been verified as being checked.

Checked    Set the Value property of this object to False to uncheck the command.
Remarks

The **OCCommandId**, **ChartCommandIdEnum**, and **PivotCommandId** constants contain lists of the supported commands for each Web component.
Example

The following example demonstrates the syntax for a subroutine that traps the CommandChecked event.

Private Sub Form_CommandChecked( _
   ByVal Command As Variant, ByVal Checked As Object)
   Dim intResponse As Integer
   Dim strPrompt As String

   strPrompt = "Uncheck the command?"

   intResponse = MsgBox(strPrompt, vbYesNo)

   If intResponse = vbYes Then
      Checked.Value = False
   Else
      Checked.Value = True
   End If
End Sub
CommandEnabled Event

Occurs when the specified Microsoft Office Web Component determines whether the specified command is enabled.

Private Sub Form_CommandEnabled(ByVal Command As Variant, ByVal Enabled As Object)

Command  The command that has been verified as being enabled.

Enabled  Set the Value property of this object to False to disable the command.
Remarks

The **OCCommandId**, **ChartCommandIdEnum**, and **PivotCommandId** constants contain lists of the supported commands for each Web component.
Example

The following example demonstrates the syntax for a subroutine that traps the CommandEnabled event.

Private Sub Form_CommandEnabled( _
    ByVal Command As Variant, ByVal Enabled As Object)
    Dim intResponse As Integer
    Dim strPrompt As String

    strPrompt = "Disable the command?"

    intResponse = MsgBox(strPrompt, vbYesNo)

    If intResponse = vbYes Then
        Enabled.Value = False
    Else
        Enabled.Value = True
    End If
End Sub
CommandExecute Event

Occurs after the specified command is executed. Use this event when you want to execute a set of commands after a particular command is executed.

Private Sub Form_CommandBeforeExecute(ByVal Command As Variant)

Command  The command that is executed.
Remarks

The OCCommandId, ChartCommandIdEnum, and PivotCommandId constants contain lists of the supported commands for each of the Microsoft Office Web Components.
Example

The following example demonstrates the syntax for a subroutine that traps the CommandExecute event.

Private Sub Form_CommandExecute(ByVal Command As Variant)
    MsgBox "The command specified by " & Command.Name & " has been executed."
End Sub
Show All
Current Event

The Current event occurs when the focus moves to a record, making it the current record, or when the form is refreshed or requiered.

Private Sub object_Current()

Object The name of a Form.


**Remarks**

To run a [macro](#) or [event procedure](#) when this event occurs, set the [OnCurrent](#) property to the name of the macro or to [Event Procedure](#).

This event occurs both when a form is opened and whenever the focus leaves one record and moves to another. Microsoft Access runs the Current macro or event procedure before the first or next record is displayed.

By running a macro or event procedure when a form's Current event occurs, you can display a message or synchronize records in another form related to the current record. For example, when a customer record becomes current, you can display the customer's previous order. When a supplier record becomes current, you can display the products manufactured by the supplier in a Suppliers form. You can also perform calculations based on the current record or change the form in response to data in the current record.

If your macro or event procedure runs a [GoToControl](#) or [GoToRecord](#) action or the corresponding method of the [DoCmd](#) object in response to an [Open](#) event, the Current event occurs.

The Current event also occurs when you refresh a form or requery the form's underlying table or query— for example, when you click [Remove Filter/Sort](#) on the [Records](#) menu or use the [Requery](#) action in a macro or the [Requery](#) method in Visual Basic code.

When you first open a form, the following events occur in this order:

Open → Load → Resize → Activate → Current
Example

In the following example, a Current event procedure checks the status of an option button called Discontinued. If the button is selected, the example sets the background color of the ProductName field to red to indicate that the product has been discontinued.

To try the example, add the following event procedure to a form that contains an option called Discontinued and a text box called ProductName.

Private Sub Form_Current()
    If Me!Discontinued Then
        Me!ProductName.BackColor = 255
    EndIf
End Sub
DataChange Event

Occurs when certain properties are changed or when certain methods are executed in the specified PivotTable view.

Private Sub Form_DataChange(ByVal Reason As Long)

Reason Use the value of the PivotDataReasonEnum constant to determine the reason that this event was triggered.

PivotDataReasonEnum can be one of these PivotDataReasonEnum constants.
plDataReasonAdhocFieldAdded
plDataReasonAdhocFieldDeleted
plDataReasonAdhocMemberChanged
plDataReasonAllIncludeExcludeChange
plDataReasonAllowDetailsChange
plDataReasonAllowMultiFilterChange
plDataReasonAlwaysIncludeInCubeChange
plDataReasonCommandTextChange
plDataReasonConnectionStringChange
plDataReasonDataMemberChange
plDataReasonDataSourceChange
plDataReasonDisplayCalculatedMembersChange
plDataReasonDisplayCellColorChange
plDataReasonDisplayEmptyMembersChange
plDataReasonExcludedMembersChange
plDataReasonExpressionChange
plDataReasonFieldNameChange
plDataReasonFieldSetDeleted
plDataReasonFieldSetNameChange
plDataReasonFilterContextChange
plDataReasonFilterCrossJoinsChange
plDataReasonFilterFunctionChange
plDataReasonFilterFunctionValueChange
plDataReasonFilterOnChange
plDataReasonFilterOnScopeChange
plDataReasonGroupEndChange
plDataReasonGroupIntervalChange
plDataReasonGroupOnChange
plDataReasonGroupStartChange
plDataReasonIncludedMembersChange
plDataReasonInsertFieldSet
plDataReasonInsertTotal
plDataReasonIsFilteredChange
plDataReasonIsIncludedChange
plDataReasonMemberPropertyDisplayInChange
plDataReasonMemberPropertyIsIncludedChange
plDataReasonOrderedMembersChange
plDataReasonRecordChanged
plDataReasonRefreshDataSource
plDataReasonRemoveFieldSet
plDataReasonRemoveTotal
plDataReasonSortDirectionChange
plDataReasonSortOnChange
plDataReasonSortOnScopeChange
plDataReasonSubtotalsChange
plDataReasonTotalAllMembersChange
plDataReasonTotalDeleted
plDataReasonTotalExpressionChange
plDataReasonTotalFunctionChange
plDataReasonTotalNameChange
plDataReasonTotalSolverOrderChange
plDataReasonUnknown
plDataReasonUser
Example

The following example demonstrates the syntax for a subroutine that traps the DataChange event. For this example to work, a reference must be set to the Microsoft Office Web Components 10.0 type library.

Private Sub Form_DataChange(Reason As Long)
    If Reason = OWC.plDataReasonDisplayCellColorChange Then
        MsgBox "The cell display color was changed."
    End If
End Sub
**DataSetChange Event**

Occurs whenever the specified PivotTable view is data-bound and the data set changes— for example, when a filter operation takes place. This event also occurs when initial data is available from the data source.

**Private Sub Form_DataSetChange()**
Example

The following example demonstrates the syntax for a subroutine that traps the DataSetChange event.

Private Sub Form_DataSetChange()
    MsgBox "The data set for the PivotChart view has changed."
End Sub
**DblClick Event**

The DblClick event occurs when the user presses and releases the left mouse button twice over an `object` within the double-click time limit of the system.

**Private Sub object_DblClick(Cancel As Integer)**

*Object*  The name of a Form or control.

*Cancel*  The setting determines if the DblClick event occurs. Setting the Cancel argument to True (–1) cancels the DblClick event.
Remarks

On a form, the DblClick event occurs when the user double-clicks a blank area or record selector on the form. For a control, it occurs when the user double-clicks a control or its label in Form view. The DblClick event occurs when the user double-clicks the form or control but before the result of the double-click action occurs (for example, before Microsoft Access selects the word that the insertion point is on in a text box).

- The DblClick event applies only to forms, form sections, and controls on a form, not controls on a report.
- This event doesn't apply to check boxes, option buttons, or toggle buttons in an option group. It applies only to the option group itself.
- This event doesn't apply to a label attached to another control, such as the label for a text box. It applies only to "freestanding" labels. Double-clicking an attached label has the same effect as double-clicking the associated control. The normal events for the control occur, not any events for the attached label.

To run a macro or event procedure when this event occurs, set the OnDblClick property to the name of the macro or to [Event Procedure].

For controls, the result of double-clicking depends on the control. For example, double-clicking a word in a text box selects the entire word. Double-clicking a control containing an OLE object starts the application used to create the object, allowing it to be edited.

If the DblClick event doesn't occur within the double-click time limit of the system, the form, form section, or control recognizes two Click events instead of a single DblClick event. The double-click time limit depends on the setting under Double-Click Speed on the Buttons tab of the Mouse option of Windows Control Panel.

By running a macro or an event procedure when the DblClick event occurs, you can open a window or document when an icon is double-clicked.

Double-clicking a control causes both Click and DblClick events to occur. If the control doesn't already have the focus when you double-click it, the Enter and
GotFocus events for the control occur before the Click and DblClick events.

For objects that receive mouse events, the events occur in this order:

MouseDown → MouseUp → Click → DblClick

When you double-click a command button, the following events occur in this order:

MouseDown → MouseUp → Click → DblClick → MouseUp → Click

The second click may have no effect (for example, if the Click macro or event procedure opens a modal dialog box in response to the first Click event). To prevent the second Click macro or event procedure from running, put a CancelEvent action in the DblClick macro or use the Cancel argument in the DblClick event procedure. Note that, generally speaking, double-clicking a command button should be discouraged.

If you double-click any other control besides a command button, the second Click event doesn't occur.
Macro

By running a macro when the DblClick event occurs, you can modify a control's default double-click behavior. For example, you can run a macro before an OLE server is opened in response to double-clicking an embedded object. The macro could display a message asking if the user wants to print or edit the object. You could use a user-defined function to print the object.

You can also use the DblClick event to enable a dialog box to be closed by double-clicking a control instead of clicking the OK button. The DblClick macro would run a Close action for the dialog box. You could also have the macro hide the dialog box by using the SetValue action to set the dialog box's Visible property to No.

You use the CancelEvent action in a DblClick macro to cancel the default double-click behavior of the control. If the user wanted to print the OLE object mentioned earlier in this topic instead of editing it, you would use a user-defined function to print the object, and then use a CancelEvent action to cancel opening of the OLE application.
Example

The following example shows how you can use a DblClick event procedure to open a form that displays records from the table that is the row source of a combo box. When the user double-clicks the Salesperson combo box in an Orders form, the Employees form is displayed, showing the record for the employee selected in the combo box.

To try the example, add the following event procedure to a form named Orders that contains a combo box named EmployeeID. The combo box should have as its row source the same table that is the source for the Employees form (or a query based on that table).

```vba
Private Sub EmployeeID_DblClick(Cancel As Integer)
    DoCmd.OpenForm "Employees", , , _
    "EmployeeID = Forms!Orders!EmployeeID"
End Sub
```
Deactivate Event

The **Deactivate** event occurs when a form or report loses the focus to a Table, Query, Form, Report, Macro, or Module window, or to the Database window.

**Private Sub object_Deactivate()**

**Object**  The name of a Form or Report.
Remarks

When you switch between two open forms, the **Deactivate** event occurs for the form being switched from, and the **Activate** event occurs for the form being switched to. If the forms contain no visible, enabled controls, the **LostFocus** event occurs for the first form before the **Deactivate** event, and the **GotFocus** event occurs for the second form after the **Activate** event.

When you first open a form, the following events occur in this order:

Open → Load → Resize → Activate → Current

When you close a form, the following events occur in this order:

Unload → Deactivate → Close
**Macro**

You can use the ShowToolbar action in an Activate or Deactivate macro to display or hide a custom toolbar. Use the Activate or Deactivate event to display or hide custom toolbars when you have more than one form loaded at a time. If you have only one form loaded at a time, you can use the Open and Close events.

If you use a Deactivate macro to hide a custom toolbar on a form, also make sure to hide the toolbar in response to the Unload event for the form, because the Deactivate event doesn't occur when a form is unloaded. Don't use the GotFocus or LostFocus event to display or hide a custom toolbar on a form, because these events aren't triggered on forms that contain enabled controls.

You can't use the CancelEvent action in an Activate or Deactivate macro.
Example

The following example shows how to hide a toolbar when the focus moves to a different window.

Private Sub Form_Deactivate()
' Hide custom toolbar.
    DoCmd.ShowToolbar "CustomToolbar", acToolbarNo
End Sub
Delete Event

Occurs when the user performs some action, such as pressing the DEL key, to delete a record, but before the record is actually deleted.

Private Sub Form_Delete(Cancel As Integer)

Cancel  The setting determines if the Delete event occurs. Setting the Cancel argument to True (–1) cancels the Delete event.
Remarks

To run a macro or event procedure when these events occur, set the OnDelete, BeforeDelConfirm, or AfterDelConfirm property to the name of the macro or to [Event Procedure].

After a record is deleted, it's stored in a temporary buffer. The BeforeDelConfirm event occurs after the Delete event (or if you've deleted more than one record, after all the records are deleted, with a Delete event occurring for each record), but before the Delete Confirm dialog box is displayed. Canceling the BeforeDelConfirm event restores the record or records from the buffer and prevents the Delete Confirm dialog box from being displayed.

The AfterDelConfirm event occurs after a record or records are actually deleted or after a deletion or deletions are canceled. If the BeforeDelConfirm event isn't canceled, the AfterDelConfirm event occurs after the Delete Confirm dialog box is displayed. The AfterDelConfirm event occurs even if the BeforeDelConfirm event is canceled. The AfterDelConfirm event procedure returns status information about the deletion. For example, you can use a macro or event procedure associated with the AfterDelConfirm event to recalculate totals affected by the deletion of records.

If you cancel the Delete event, the BeforeDelConfirm and AfterDelConfirm events don't occur and the Delete Confirm dialog box isn't displayed.

Note The BeforeDelConfirm and AfterDelConfirm events don't occur and the Delete Confirm dialog box isn't displayed if you clear the Record Changes check box under Confirm on the Edit/Find tab of the Options dialog box, available by clicking Options on the Tools menu.

By running a macro or an event procedure when the Delete event occurs, you can prevent a record from being deleted or allow a record to be deleted only under certain conditions. You can also use a Delete event to display a dialog box asking whether the user wants to delete a record before it's deleted.

To delete a record, you can click Delete Record on the Edit menu. This deletes the current record (the record indicated by the record selector). You can also click the record selector or click Select Record on the Edit menu to select the
record, and then press the DEL key to delete it. If you click **Delete Record**, the record selector of the current record, or **Select Record**, the **Exit** and **LostFocus** events for the control that has the focus occur. If you've changed any data in the record, the **BeforeUpdate** and **AfterUpdate** events for the record occur before the Exit and LostFocus events. If you click the record selector of a different record, the **Current** event for that record also occurs.

After you delete the record, the focus moves to the next record following the deleted record, and the **Current** event for that record occurs, followed by the **Enter** and **GotFocus** events for the first control in that record.

The **BeforeDelConfirm** event then occurs, just before Microsoft Access displays the **Delete Confirm** dialog box asking you to confirm the deletion. After you respond to the dialog box by confirming or canceling the deletion, the **AfterDelConfirm** event occurs.

You can delete one or more records at a time. The **Delete** event occurs after each record is deleted. This enables you to access the data in each record before it's actually deleted, and selectively confirm or cancel each deletion in the Delete macro or event procedure. When you delete more than one record, the **Current** event for the record following the last deleted record and the **Enter** and **GotFocus** events for the first control in this record don't occur until all the records are deleted. In other words, a Delete event occurs for each selected record, but no other events occur until all the selected records are deleted. The **BeforeDelConfirm** and **AfterDelConfirm** events also don't occur until all the selected records are deleted.
Macro

You can use a Delete macro to display the Delete Confirm dialog box asking whether the user wants to delete a record before it's deleted.

You can use a BeforeDelConfirm or AfterDelConfirm macro to respond whenever the user deletes a record. However, the BeforeDelConfirm and AfterDelConfirm macros can't set the argument determining whether the Delete Confirm dialog box is displayed and can't return the argument indicating the status of the deletion, so you typically use event procedures with these events.

You can use the CancelEvent action in a Delete macro to cancel the deletion.

You can use the CancelEvent action in a BeforeDelConfirm macro to cancel the deletion of all the deleted records. If you do this, the Delete Confirm dialog box isn't displayed. However, the AfterDelConfirm event occurs even if you cancel the BeforeDelConfirm event.

You can't use the CancelEvent action in an AfterDelConfirm macro.
Example

The following example shows how you can prevent a user from deleting records from a table.

To try this example, add the following event procedure to a form that is based on a table. Switch to form Datasheet view and try to delete a record.

Private Sub Form_Delete(Cancel As Integer)
    Cancel = True
    MsgBox "This record can't be deleted."
End Sub
Show All
Dirty Event

The Dirty event occurs when the contents of a form or the text portion of a combo box changes. It also occurs when you move from one page to another page in a tab control.

Private Sub Form_Dirty(Cancel As Integer)

The Dirty event procedure has the following argument.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cancel</td>
<td>The setting determines if the Dirty event occurs. Setting the Cancel argument to True cancels the Dirty event. You can also use the CancelEvent method of the DoCmd object to cancel the event.</td>
</tr>
</tbody>
</table>
Remarks

Examples of this event include entering a character directly in the text box or combo box or changing the control's `Text` property setting by using a `macro` or Visual Basic.

- Modifying a record within a form by using a macro or Visual Basic doesn't trigger this event. You must type the data directly into the record or set the control's `Text` property.
- This event applies only to bound forms, not an `unbound form` or report.

To run a macro or `event procedure` when this event occurs, set the `OnDirty` property to the name of the macro or to `[Event Procedure].`

By running a macro or event procedure when a Dirty event occurs, you can determine if the record can be changed. You can also display a message and ask for edit permission.

Changing the data in a record by using the keyboard causes keyboard events to occur in addition to control events like the Dirty event. For example, if you move to a new record and type an ANSI character in a text box in the record, the following events occur in this order:

```
KeyDown > KeyPress > BeforeInsert > Dirty > KeyUp
```

The `BeforeUpdate` and `AfterUpdate` events for a record occur after you have entered the new or changed data in the record and moved to another record (or clicked `Save Record` on the `Records` menu), and therefore after the Dirty event for the record.

Canceling the Dirty event will cause the changes to the current record to be rolled back. It is equivalent to pressing the ESC key.
Example

The following example enables the btnUndo button when data is changed. The UndoEdits() subroutine is called from the Dirty event of text box controls. Clicking the enabled btnUndo button restores the original value of the control by using the OldValue property.

Private Sub Form_Dirty()
    If Me.Dirty Then
        Me!btnUndo.Enabled = True ' Enable button.
    Else
        Me!btnUndo.Enabled = False ' Disable button.
    End If
End Sub

Sub btnUndo_Click()
    Dim ctlC As Control
    ' For each control.
    For Each ctlC In Me.Controls
        If ctlC.ControlType = acTextBox Then
            ' Restore Old Value.
            ctlC.Value = ctlC.OldValue
        End If
    Next ctlC
End Sub
Enter Event

The Enter event occurs before a control actually receives the focus from a control on the same form.

Private Sub object_Enter()

Object The name of a control on a Form.
Remarks

The Enter event applies only to controls on a form, not controls on a report. This event does not apply to check boxes, option buttons, or toggle buttons in an option group. It applies only to the option group itself.

To run a macro or event procedure when these events occur, set the OnEnter or OnExit property to the name of the macro or to [Event Procedure].

Because the Enter event occurs before the focus moves to a particular control, you can use an Enter macro or event procedure to display instructions; for example, you could use a macro or event procedure to display a small form or message box identifying the type of data the control typically contains, or giving instructions on how to use the control.

The Enter event occurs before the GotFocus event. The Exit event occurs before the LostFocus event.

Unlike the GotFocus and LostFocus events, the Enter and Exit events don't occur when a form receives or loses the focus. For example, suppose you select a check box on a form, and then click a report. The Enter and GotFocus events occur when you select the check box. Only the LostFocus event occurs when you click the report. The Exit event doesn't occur (because the focus is moving to a different window). If you select the check box on the form again to bring it to the foreground, the GotFocus event occurs, but not the Enter event (because the control had the focus when the form was last active). The Exit event occurs only when you click another control on the form.

If you move the focus to a control on a form, and that control doesn't have the focus on that form, the Exit and LostFocus events for the control that does have the focus on the form occur before the Enter and GotFocus events for the control you moved to.

If you use the mouse to move the focus from a control on a main form to a control on a subform of that form (a control that doesn't already have the focus on the subform), the following events occur:

Exit (for the control on the main form)
LostFocus (for the control on the main form)

Enter (for the subform control)

Exit (for the control on the subform that had the focus)

LostFocus (for the control on the subform that had the focus)

Enter (for the control on the subform that the focus moved to)

GotFocus (for the control on the subform that the focus moved to)

If the control you move to on the subform previously had the focus, neither its Enter event nor its GotFocus event occurs, but the Enter event for the subform control does occur. If you move the focus from a control on a subform to a control on the main form, the Exit and LostFocus events for the control on the subform don't occur, just the Exit event for the subform control and the Enter and GotFocus events for the control on the main form.

Note  You often use the mouse or a key such as TAB to move the focus to another control. This causes mouse or keyboard events to occur in addition to the events discussed in this topic.
Macros

When the Enter or Exit event occurs, you can run a macro that uses a GoToControl action to change the tab order based on current data in a control. For example, if the user enters Single in the MaritalStatus field, you can ensure that the focus doesn't move to the SpouseName field.

You can use the CancelEvent action in an Exit macro to cancel exiting the control. You can't use the CancelEvent action in an Enter macro to cancel the Enter event. However, you can use the GoToControl action in an Enter macro to move the focus to a different control.
Example

In the following example, two event procedures are attached to the LastName text box. The Enter event procedure displays a message specifying what type of data the user can enter in the text box. The Exit event procedure displays a dialog box asking the user if changes should be saved before the focus moves to another control. If the user clicks the Cancel button, the Cancel argument is set to True (–1), which moves the focus to the text box without saving changes. If the user chooses the OK button, the changes are saved, and the focus moves to another control.

To try the example, add the following event procedure to a form that contains a text box named LastName.

```vbscript
Private Sub LastName_Enter()
    MsgBox "Enter your last name."
End Sub

Private Sub LastName_Exit(Cancel As Integer)
    Dim strMsg As String

    strMsg = "You entered '" & Me!LastName _ & "' as your last name." & vbCrLf & "Is this correct?"
    If MsgBox(strMsg, vbYesNo) = vbNo Then
        Cancel = True ' Cancel exit.
    Else
        Exit Sub ' Save changes and exit.
    End If
End Sub
```
Error Event

The Error event occurs when a run-time error is produced in Microsoft Access when a form or report has the focus.

Private Sub object_Error(DataErr As Integer, Response As Integer)

Object  The name of a Form or Report.

DataErr  The error code returned by the Err object when an error occurs. You can use the DataErr argument with the Error function to map the number to the corresponding error message.

Response  The setting determines whether or not an error message is displayed. The Response argument can be one of the following intrinsic constants.

Constant Description

acDataErrContinue Ignore the error and continue without displaying the default Microsoft Access error message. You can supply a custom error message in place of the default error message.

acDataErrDisplay (Default) Display the default Microsoft Access error message.
Remarks

This includes Microsoft Jet database engine errors, but not run-time errors in Visual Basic or errors from ADO.

To run a macro or event procedure when this event occurs, set the OnError property to the name of the macro or to [Event Procedure].

By running an event procedure or a macro when an Error event occurs, you can intercept a Microsoft Access error message and display a custom message that conveys a more specific meaning for your application.
Example

The following example shows how you can replace a default error message with a custom error message. When Microsoft Access returns an error message indicating it has found a duplicate key (error code 3022), this event procedure displays a message that gives more application-specific information to users.

To try the example, add the following event procedure to a form that is based on a table with a unique employee ID number as the key for each record.

```vba
Private Sub Form_Error(DataErr As Integer, Response As Integer)
    Const conDuplicateKey = 3022
    Dim strMsg As String

    If DataErr = conDuplicateKey Then
        Response = acDataErrContinue
        strMsg = "Each employee record must have a unique " _
                 & "employee ID number. Please recheck your data."
        MsgBox strMsg
    End If
End Sub
```
Exit Event

The Exit event occurs just before a control loses the focus to another control on the same form.

Private Sub object_Exit()

Object The name of a control on a Form.
Remarks

The Exit event applies only to controls on a form, not controls on a report. This event does not apply to check boxes, option buttons, or toggle buttons in an option group. It applies only to the option group itself.

To run a macro or event procedure when this event occurs, set the OnExit property to the name of the macro or to [Event Procedure].

Because the Enter event occurs before the focus moves to a particular control, you can use an Enter macro or event procedure to display instructions; for example, you could use a macro or event procedure to display a small form or message box identifying the type of data the control typically contains, or giving instructions on how to use the control.

The Exit event occurs before the LostFocus event.

Unlike the LostFocus event, the Exit event does not occur when a form loses the focus. For example, suppose you select a check box on a form, and then click a report. The Enter and GotFocus events occur when you select the check box. Only the LostFocus event occurs when you click the report. The Exit event doesn't occur (because the focus is moving to a different window). If you select the check box on the form again to bring it to the foreground, the GotFocus event occurs, but not the Enter event (because the control had the focus when the form was last active). The Exit event occurs only when you click another control on the form.

If you move the focus to a control on a form, and that control doesn't have the focus on that form, the Exit and LostFocus events for the control that does have the focus on the form occur before the Enter and GotFocus events for the control you moved to.

If you use the mouse to move the focus from a control on a main form to a control on a subform of that form (a control that doesn't already have the focus on the subform), the following events occur:

Exit (for the control on the main form)
LostFocus (for the control on the main form)

Enter (for the subform control)

Exit (for the control on the subform that had the focus)

LostFocus (for the control on the subform that had the focus)

Enter (for the control on the subform that the focus moved to)

GotFocus (for the control on the subform that the focus moved to)

If the control you move to on the subform previously had the focus, neither its Enter event nor its GotFocus event occurs, but the Enter event for the subform control does occur. If you move the focus from a control on a subform to a control on the main form, the Exit and LostFocus events for the control on the subform don't occur, just the Exit event for the subform control and the Enter and GotFocus events for the control on the main form.

**Note** You often use the mouse or a key such as TAB to move the focus to another control. This causes mouse or keyboard events to occur in addition to the events discussed in this topic.
Macro

When the Enter or Exit event occurs, you can run a macro that uses a GoToControl action to change the tab order based on current data in a control. For example, if the user enters Single in the MaritalStatus field, you can ensure that the focus doesn't move to the SpouseName field.

You can use the CancelEvent action in an Exit macro to cancel exiting the control. You can't use the CancelEvent action in an Enter macro to cancel the Enter event. However, you can use the GoToControl action in an Enter macro to move the focus to a different control.
Example

In the following example, two event procedures are attached to the LastName text box. The Enter event procedure displays a message specifying what type of data the user can enter in the text box. The Exit event procedure displays a dialog box asking the user if changes should be saved before the focus moves to another control. If the user clicks the Cancel button, the Cancel argument is set to **True** (−1), which moves the focus to the text box without saving changes. If the user chooses the OK button, the changes are saved, and the focus moves to another control.

To try the example, add the following event procedure to a form that contains a text box named LastName.

```vbnet
Private Sub LastName_Enter()
    MsgBox "Enter your last name."
End Sub

Private Sub LastName_Exit(Cancel As Integer)
    Dim strMsg As String

        strMsg = "You entered '" & Me!LastName & "' as your last name." & 
                vbCrLf & "Is this correct?"

    If MsgBox(strMsg, vbYesNo) = vbNo Then
        Cancel = True  ' Cancel exit.
    Else
        Exit Sub  ' Save changes and exit.
    End If
End Sub
```
Filter Event

The Filter event can occur within a Microsoft Access project (.adp) or Access database (.mdb).

Private Sub Form_Filter(Cancel As Integer, FilterType As Integer)

Cancel The setting determines whether to open the filter window. Setting the Cancel argument to True (–1) prevents the filter window from opening. You can also use the CancelEvent method of the DoCmd object to cancel opening the filter window.

FilterType The filter window the user is trying to open. The FilterType argument can be one of the following intrinsic constants:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>acFilterByForm</td>
<td>0</td>
</tr>
<tr>
<td>acFilterAdvanced</td>
<td>1</td>
</tr>
<tr>
<td>acServerFilterByForm</td>
<td>2</td>
</tr>
</tbody>
</table>

Within an Access database, a Filter event occurs when the user does one of the following:

- On the Records menu in Form view, points to Filter and then clicks Filter By Form, or clicks Filter By Form on the toolbar. This opens the Filter By Form window, where you can create a filter based on the fields in the form.
- On the Records menu in Form view, points to Filter and then clicks Advanced Filter/Sort. This opens the Advanced Filter/Sort window, where you can create complex filters for the form.
- Clicks Advanced Filter/Sort on the Filter menu while the Filter By Form window is open, or clicks Filter By Form on the Filter menu while the Advanced Filter/Sort window is open. This causes the ApplyFilter event to occur when the open filter window is closed, and then the Filter event to
occur when the other filter window is opened.

Within an Access project, a Filter event occurs when the user does one of the following:

- On the **Records** menu in Form view, points to **Filter** and then clicks **Filter By Form**, or clicks **Filter By Form** on the toolbar. This opens the Filter By Form window, where you can create a filter based on the fields in the form.
- On the **Records** menu in Form view, points to **Filter** and then clicks **Server Filter By Form** on the toolbar. This opens the **Server Filter By Form** window, where you can quickly create a server filter based on the fields in the form.
- The Advanced Filter/Sort window is not available in an Access project.
Remarks

To run a macro or event procedure when this event occurs, set the OnFilter property to the name of the macro or to [Event Procedure].

You can use the Filter event to:

- Remove any previous filter for the form. To do this, set the Filter or ServerFilter property of the form to a zero-length string (" ") in the Filter macro or event procedure. This is especially useful if you want to make sure extraneous criteria don't appear in the new filter. For example, when you use the Filter By Selection feature, the criteria you use (the selected text in the form) is added to the Filter or ServerFilter property WHERE clause expression, and appears in both the Filter By Form window and the Advanced Filter/Sort window or the Server Filter By Form window. You can remove these old criteria by using the Filter event.

- Enter default settings for the new filter. To do this, set the Filter or ServerFilter property to include these criteria. For example, you may want all filters for a Products form to display only current products (products for which the Discontinued control in the Products form isn't selected).

- Use your own custom filter window instead of one of the Microsoft Access filter windows. When the Filter event occurs, you can open your own custom form and use the entries on this form to set the Filter or ServerFilter property and filter the original form. When the user closes this custom form, set the FilterOn or ServerFilterByForm property of the original form to True (–1) to apply the filter. Canceling the Filter event prevents the Microsoft Access filter window from opening.

- Prevent certain controls on the form from appearing or being used in the Filter By Form or Server Filter By Form window. If you hide or disable a control in the Filter macro or event procedure, the control is hidden or disabled in the Filter By Form or Server Filter By Form window, and can't be used to set filter criteria. You can then use the ApplyFilter event to show or enable this control after the filter is applied, or when the filter is removed from the form.
**Macro**

You can set the Filter or ServerFilter property in a Filter macro by using the SetValue action. This enables you to customize the filter criteria that the user sees when the Filter By Form, Advanced Filter/Sort, or the Server Filter By Form window is displayed.

You can use the CancelEvent action in a Filter macro to cancel opening the filter window.
Example

The following example shows how to disable the TotalDue control on an Orders form when the user tries to create a filter, so that the user can't filter on this field. Any records that have a TotalDue value and meet the other filter criteria will always be shown on the filtered form. This example also forces the user to use the Filter By Form window and not the Advanced Filter/Sort window.

To try this example, add the following event procedure to an Orders form that contains a TotalDue control. Try to create a filter by using the Advanced Filter/Sort window that uses the TotalDue control. Also try creating the same filter by using the Filter By Form window.

Private Sub Form_Filter(Cancel As Integer, FilterType As Integer)
    If FilterType = acFilterByForm Then
        Forms!Orders!TotalDue.Enabled = False
    ElseIf FilterType = acFilterAdvanced Then
        MsgBox "The best way to filter this form is to use the " & "Filter By Form command or toolbar button.", vbOKOnly
        Cancel = True
    End If
End Sub
GotFocus Event

The GotFocus event occurs when a form or control receives the focus.

Private Sub object_GotFocus()

Object The name of a Form or a control on a Form.
Remarks

**Note** The GotFocus event applies only to forms and controls on a form, not controls on a report.

To run a macro or event procedure when these events occur, set the **OnGotFocus** property to the name of the macro or to [Event Procedure].

These events occur when the focus moves in response to a user action, such as pressing the TAB key or clicking the object, or when you use the **SetFocus** method in Visual Basic or the **SelectObject**, **GoToRecord**, **GoToControl**, or **GoToPage** action in a macro.

A control can receive the focus only if its **Visible** and **Enabled** properties are set to Yes. A form can receive the focus only if it has no controls or if all visible controls are disabled. If a form contains any visible, enabled controls, the GotFocus event for the form doesn't occur.

You can specify what happens when a form or control receives the focus by running a macro or an event procedure when the GotFocus event occurs. For example, by attaching a GotFocus event procedure to each control on a form, you can guide the user through your application by displaying brief instructions or messages in a text box. You can also provide visual cues by enabling, disabling, or displaying controls that depend on the control with the focus.

**Note** To customize the order in which the focus moves from control to control on a form when you press the TAB key, set the **tab order** or specify **access keys** for the controls.

The GotFocus event differs from the **Enter** event in that the GotFocus event occurs every time a control receives the focus. For example, suppose the user clicks a check box on a form, then clicks a report, and finally clicks the check box on the form to bring it to the foreground. The GotFocus event occurs both times the check box receives the focus. In contrast, the Enter event occurs only the first time the user clicks the check box. The GotFocus event occurs after the Enter event.

If you move the focus to a control on a form, and that control doesn't have the
focus on that form, the Exit and LostFocus events for the control that does have the focus on the form occur before the Enter and GotFocus events for the control you moved to.

If you use the mouse to move the focus from a control on a main form to a control on a subform of that form, the following events occur:

Exit (for the control on the main form)

β

LostFocus (for the control on the main form)

β

Enter (for the subform control)

β

Exit (for the control on the subform that had the focus)

β

LostFocus (for the control on the subform that had the focus)

β

Enter (for the control on the subform that the focus moved to)

β

GotFocus (for the control on the subform that the focus moved to)

If the control you move to on the subform previously had the focus, neither its Enter event nor its GotFocus event occurs, but the Enter event for the subform control does occur. If you move the focus from a control on a subform to a control on the main form, the Exit and LostFocus events for the control on the subform don't occur, just the Exit event for the subform control and the Enter and GotFocus events for the control on the main form.

Note  You often use the mouse or a key such as TAB to move the focus to
another control. This causes mouse or keyboard events to occur in addition to the events discussed in this topic.

When you switch between two open forms, the Deactivate event occurs for the first form, and the Activate event occurs for the second form. If the forms contain no visible, enabled controls, the LostFocus event occurs for the first form before the Deactivate event, and the GotFocus event occurs for the second form after the Activate event.
Macro

You can use a LostFocus macro to validate entered data as the user moves the focus from a control. You can also reverse or change conditions that you set up in the object's GotFocus macro.

Other uses for LostFocus and GotFocus macros are enabling, disabling, displaying, and hiding other objects.

You can't use the CancelEvent action in a GotFocus or LostFocus macro.
**Example**

The following example displays a message in a label when the focus moves to an option button.

To try the example, add the following event procedures to a form named Contacts that contains an option button named OptionYes and a label named LabelYes.

```vbnet
Private Sub OptionYes_GotFocus()
    Me!LabelYes.Caption = "Option button 'Yes' has the focus."
End Sub

Private Sub OptionYes_LostFocus()
    Me!LabelYes.Caption = "" ' Clear caption.
End Sub
```
ItemAdded Event

Some of the content in this topic may not be applicable to some languages.

The ItemAdded event occurs when a reference is added to the project from Visual Basic.
Remarks

- The ItemAdded event applies to the References collection. It isn't associated with a control, form, or report, as are most other events. Therefore, in order to create a procedure definition for the ItemAdded event procedure, you must use a special syntax.
- The ItemAdded event can run only an event procedure when it occurs, it cannot run a macro.

This event occurs only when you add a reference from code. It doesn't occur when you add a reference from the References dialog box, available by clicking References on the Tools menu when the Module window is the active window.
The following example includes event procedures for the ItemAdded and ItemRemoved events. To try this example, first create a new class module by clicking **Class Module** on the **Insert** menu. Paste the following code into the class module and save the module as RefEvents:

' Declare object variable to represent References collection.
Public WithEvents evtReferences As References

' When instance of class is created, initialize evtReferences variable.
Private Sub Class_Initialize()
    Set evtReferences = Application.References
End Sub

' When instance is removed, set evtReferences to Nothing.
Private Sub Class_Terminate()
    Set evtReferences = Nothing
End Sub

' Display message when reference is added.
Private Sub evtReferences_ItemAdded(ByVal Reference As Access.Reference)
    MsgBox "Reference to " & Reference.Name & " added."
End Sub

' Display message when reference is removed.
Private Sub evtReferences_ItemRemoved(ByVal Reference As Access.Reference)
    MsgBox "Reference to " & Reference.Name & " removed."
End Sub

The following **Function** procedure adds a specified reference. When a reference is added, the ItemAdded event procedure defined in the RefEvents class runs.

For example, to set a reference to the calendar control, you could pass the string "C:\Windows\System\Mscal.ocx", if this is the correct location for the calendar control on your computer.

' Create new instance of RefEvents class.
Dim objRefEvents As New RefEvents
' Pass file name and path of type library to this procedure.
Function AddReference(strFileName As String) As Boolean
    Dim ref As Reference

    On Error GoTo Error_AddReference

    ' Create new reference on References object variable.
    Set ref = objRefEvents.evtReferences.AddFromFile(strFileName)
    AddReference = True

Exit_AddReference:
    Exit Function

Error_AddReference:
    MsgBox Err & ": " & Err.Description
    AddReference = False
    Resume Exit_AddReference

End Function
ItemRemoved Event

Some of the content in this topic may not be applicable to some languages.

The ItemRemoved event occurs when a reference is removed from the project.
Remarks

- The ItemRemoved event applies to the References collection. It isn't associated with a control, form, or report, as are most other events. Therefore, in order to create a procedure definition for the ItemRemoved event procedure, you must use a special syntax.
- The ItemRemoved event can run only an event procedure when it occurs, it cannot run a macro.

This event occurs only when you remove a reference from code. It doesn't occur when you remove a reference from the References dialog box, available by clicking References on the Tools menu when the Module window is the active window.
Example

The following example includes event procedures for the ItemAdded and ItemRemoved events. To try this example, first create a new class module by clicking **Class Module** on the **Insert** menu. Paste the following code into the class module and save the module as RefEvents:

```
' Declare object variable to represent References collection.
Public WithEvents evtReferences As References
'
' When instance of class is created, initialize evtReferences variable.
Private Sub Class_Initialize()
    Set evtReferences = Application.References
End Sub
'
' When instance is removed, set evtReferences to Nothing.
Private Sub Class_Terminate()
    Set evtReferences = Nothing
End Sub
'
' Display message when reference is added.
Private Sub evtReferences_ItemAdded(ByVal Reference As Access.Reference)
    MsgBox "Reference to " & Reference.Name & " added."
End Sub
'
' Display message when reference is removed.
Private Sub evtReferences_ItemRemoved(ByVal Reference As Access.Reference)
    MsgBox "Reference to " & Reference.Name & " removed."
End Sub
```

The next **Function** procedure removes a specified reference. When a reference is removed, the ItemRemoved event procedure defined in the RefEvents class runs.

For example, to remove a reference to the calendar control, you could pass the string "MSACAL", which is the name of the **Reference** object that represents the calendar control.

```
Function RemoveReference(strRefName As String) As Boolean
    Dim ref As Reference
```
On Error GoTo Error_RemoveReference
' Return object representing existing reference.
Set ref = objRefEvents.evtReferences(strRefName)
' Remove reference from collection.
objRefEvents.evtReferences.Remove ref
RemoveReference = True

Exit_RemoveReference:
   Exit Function

Error_RemoveReference:
   MsgBox Err & ": " & Err.Description
   RemoveReference = False
   Resume Exit_RemoveReference

End Function
KeyDown Event

The KeyDown event occurs when the user presses a key while a form or control has the focus. This event also occurs if you send a keystroke to a form or control by using the SendKeys action in a macro or the SendKeys statement in Visual Basic.

Private Sub object_KeyDown(KeyCode As Integer, Shift As Integer)

Object  The name of a Form or control on a Form.

KeyCode  A key code, such as vbKeyF1 (the F1 key) or vbKeyHome (the HOME key). To specify key codes, use the intrinsic constants shown in the Object Browser. You can prevent an object from receiving a keystroke by setting KeyCode to 0.

Shift  The state of the SHIFT, CTRL, and ALT keys at the time of the event. If you need to test for the Shift argument, you can use one of the following intrinsic constants as bit masks:

Constant Description

acShiftMask The bit mask for the SHIFT key.

acCtrlMask The bit mask for the CTRL key.

acAltMask The bit mask for the ALT key.
Remarks

**Note** The KeyDown event applies only to forms and controls on a form, not controls on a report.

To run a macro or event procedure when these events occur, set the OnKeyDown property to the name of the macro or to [Event Procedure].

For both events, the object with the focus receives all keystrokes. A form can have the focus only if it has no controls or all its visible controls are disabled.

A form will also receive all keyboard events, even those that occur for controls, if you set the KeyPreview property of the form to Yes. With this property setting, all keyboard events occur first for the form, and then for the control that has the focus. You can respond to specific keys pressed in the form, regardless of which control has the focus. For example, you may want the key combination CTRL+X to always perform the same action on a form.

If you press and hold down a key, the KeyDown and_KeyPress events alternate repeatedly (KeyDown, KeyPress, KeyDown, KeyPress, and so on) until you release the key, then the KeyUp event occurs.

Although the KeyDown event occurs when most keys are pressed, it is typically used to recognize or distinguish between:

- Extended character keys, such as function keys.
- Navigation keys, such as HOME, END, PAGE UP, PAGE DOWN, UP ARROW, DOWN ARROW, RIGHT ARROW, LEFT ARROW, and TAB.
- Combinations of keys and standard keyboard modifiers (SHIFT, CTRL, or ALT keys).
- The numeric keypad and keyboard number keys.

The KeyDown event does not occur when you press:

- The ENTER key if the form has a command button for which the Default property is set to Yes.
- The ESC key if the form has a command button for which the **Cancel** property is set to Yes.
Tip

To find out the ANSI character corresponding to the key pressed, use the **KeyPress** event.

The KeyDown event occurs when you press or send an ANSI key. The KeyUp event occurs after any event for a control caused by pressing or sending the key. If a keystroke causes the focus to move from one control to another control, the KeyDown event occurs for the first control, while the KeyPress and KeyUp events occur for the second control.

For more information, see [Order of events for keystrokes and mouse clicks](#).

If a **modal** dialog box is displayed as a result of pressing or sending a key, the KeyDown and KeyPress events occur, but the KeyUp event doesn't occur.
Macro

You can use a KeyDown or KeyUp macro to respond whenever the user presses or releases a key while a form or control has the focus. However, macros can't return the key code and determine which key was pressed, so you typically use event procedures with these events.

You can't use the CancelEvent action in a KeyDown or KeyUp macro.
Example

The following example determines whether you have pressed the SHIFT, CTRL, or ALT key.

To try the example, add the following event procedure to a form containing a text box named KeyHandler.

```vba
Private Sub KeyHandler_KeyDown(KeyCode As Integer, _
    Shift As Integer)
    Dim intShiftDown As Integer, intAltDown As Integer
    Dim intCtrlDown As Integer

    ' Use bit masks to determine which key was pressed.
    intShiftDown = (Shift And acShiftMask) > 0
    intAltDown = (Shift And acAltMask) > 0
    intCtrlDown = (Shift And acCtrlMask) > 0

    ' Display message telling user which key was pressed.
    If intShiftDown Then MsgBox "You pressed the SHIFT key."
    If intAltDown Then MsgBox "You pressed the ALT key."
    If intCtrlDown Then MsgBox "You pressed the CTRL key."

End Sub
```
KeyPress Event

The KeyPress event occurs when the user presses and releases a key or key combination that corresponds to an ANSI code while a form or control has the focus. This event also occurs if you send an ANSI keystroke to a form or control by using the SendKeys action in a macro or the SendKeys statement in Visual Basic.

Private Sub object_KeyPress(KeyAscii As Integer)

Object The name of a Form or control on a Form.

KeyAscii Returns a numeric ANSI key code. The KeyAscii argument is passed by reference; changing it sends a different character to the object. Setting the KeyAscii argument to 0 cancels the keystroke so that the object doesn't recognize that a key was pressed.
Remarks

**Note** The KeyPress event applies only to forms and controls on a form, not controls on a report.

To run a macro or event procedure when this event occurs, set the OnKeyPress property to the name of the macro or to [Event Procedure].

The object with the focus receives all keystrokes. A form can have the focus only if it has no controls or all its visible controls are disabled.

A form will also receive all keyboard events, even those that occur for controls, if you set the KeyPreview property of the form to Yes. With this property setting, all keyboard events occur first for the form, and then for the control that has the focus. You can respond to specific keys pressed in the form, regardless of which control has the focus. For example, you may want the key combination CTRL+X to always perform the same action on a form.

If you press and hold down an ANSI key, the KeyDown and KeyPress events alternate repeatedly (KeyDown, KeyPress, KeyDown, KeyPress, and so on) until you release the key, then the KeyUp event occurs.

A KeyPress event can involve any printable keyboard character, the CTRL key combined with a character from the standard alphabet or a special character, and the ENTER or BACKSPACE key. You can use the KeyDown and KeyUp event procedures to handle any keystroke not recognized by the KeyPress event, such as function keys, navigation keys, and any combinations of these with keyboard modifiers (ALT, SHIFT, or CTRL keys). Unlike the KeyDown and KeyUp events, the KeyPress event doesn't indicate the physical state of the keyboard; instead, it indicates the ANSI character that corresponds to the pressed key or key combinations.

KeyPress interprets the uppercase and lowercase of each character as separate key codes and, therefore, as two separate characters.

**Note** The BACKSPACE key is part of the ANSI character set, but the DEL key isn't. If you delete a character in a control by using the BACKSPACE key, you cause a KeyPress event; if you use the DEL key, you don't.
The KeyDown and KeyPress events occur when you press or send an ANSI key. The KeyUp event occurs after any event for a control caused by pressing or sending the key. If a keystroke causes the focus to move from one control to another control, the KeyDown event occurs for the first control, while the KeyPress and KeyUp events occur for the second control.

For example, if you go to a new record and type a character in the first control in the record, the following events occur:

Current (for the new record)

Enter (for the first control in the new record)

GotFocus (for the control)

KeyDown (for the control)

KeyPress (for the control)

BeforeInsert (for the new record in the form)

Change (for the control if it's a text box or combo box)

KeyUp (for the control)

For more information, see Order of events for keystrokes and mouse clicks.
Macro

You can use a KeyPress macro to respond whenever the user presses an ANSI key while the focus is on a form or control. However, macros can't return the ANSI key code and determine which key was pressed, so you typically use event procedures with this event.

You can use the CancelEvent action in a KeyPress macro to cancel the key or key combination. For example, you may want to disallow the entry of ANSI keystrokes in a control under certain conditions.
Example

The following example converts text entered in a text box to uppercase as the text is typed in, one character at a time.

To try the example, add the following event procedure to a form that contains a text box named ShipRegion.

Private Sub ShipRegion_KeyPress(KeyAscii As Integer)
    Dim strCharacter As String

    ' Convert ANSI value to character string.
    strCharacter = Chr(KeyAscii)
    ' Convert character to upper case, then to ANSI value.
    KeyAscii = Asc(UCase(strCharacter))
End Sub
KeyUp Event

The KeyUp event occurs when the user releases a key while a form or control has the focus. This event also occurs if you send a keystroke to a form or control by using the SendKeys action in a macro or the `SendKeys` statement in Visual Basic.

**Private Sub** `object_KeyUp(KeyCode As Integer, Shift As Integer)`

*Object*  The name of a Form or control on a Form.

*KeyCode*  A key code, such as `vbKeyF1` (the F1 key) or `vbKeyHome` (the HOME key). To specify key codes, use the intrinsic constants shown in the Object Browser. You can prevent an object from receiving a keystroke by setting KeyCode to 0.

*Shift*  The state of the SHIFT, CTRL, and ALT keys at the time of the event. If you need to test for the Shift argument, you can use one of the following intrinsic constants as bit masks:
Remarks

**Note** The KeyUp event applies only to forms and controls on a form, not controls on a report.

To run a macro or event procedure when these events occur, set the **OnKeyUp** property to the name of the macro or to [Event Procedure].

For this event, the object with the focus receives all keystrokes. A form can have the focus only if it has no controls or all its visible controls are disabled.

A form will also receive all keyboard events, even those that occur for controls, if you set the **KeyPreview** property of the form to Yes. With this property setting, all keyboard events occur first for the form, and then for the control that has the focus. You can respond to specific keys pressed in the form, regardless of which control has the focus. For example, you may want the key combination CTRL+X to always perform the same action on a form.

If you press and hold down a key, the KeyDown and **KeyPress** events alternate repeatedly (KeyDown, KeyPress, KeyDown, KeyPress, and so on) until you release the key, then the KeyUp event occurs.

Although the KeyUp event occurs when most keys are pressed, they are typically used to recognize or distinguish between:

- Extended character keys, such as function keys.

- Navigation keys, such as HOME, END, PAGE UP, PAGE DOWN, UP ARROW, DOWN ARROW, RIGHT ARROW, LEFT ARROW, and TAB.

- Combinations of keys and standard keyboard modifiers (SHIFT, CTRL, or ALT keys).

- The numeric keypad and keyboard number keys.

The KeyUp event does not occur when you press:

- The ENTER key if the form has a command button for which the **Default**
property is set to Yes.

- The ESC key if the form has a command button for which the Cancel property is set to Yes.

**Tip**

To find out the ANSI character corresponding to the key pressed, use the KeyPress event.

The KeyUp event occurs after any event for a control caused by pressing or sending the key. If a keystroke causes the focus to move from one control to another control, the KeyDown event occurs for the first control, while the KeyPress and KeyUp events occur for the second control.

For more information, see Order of events for keystrokes and mouse clicks.

If a modal dialog box is displayed as a result of pressing or sending a key, the KeyDown and KeyPress events occur, but the KeyUp event doesn't occur.
Macro

You can use a KeyDown or KeyUp macro to respond whenever the user presses or releases a key while a form or control has the focus. However, macros can't return the key code and determine which key was pressed, so you typically use event procedures with these events.

You can't use the CancelEvent action in a KeyDown or KeyUp macro.
Load Event

The Load event occurs when a form is opened and its records are displayed.

Private Sub Form_Load()
Remarks

To run a macro or event procedure when these events occur, set the OnLoad property to the name of the macro or to [Event Procedure].

The Load event is caused by user actions such as:

- Starting an application.
- Opening a form by clicking Open in the Database window.
- Running the OpenForm action in a macro.

By running a macro or an event procedure when a form's Load event occurs, you can specify default settings for controls, or display calculated data that depends on the data in the form's records.

By running a macro or an event procedure when a form's Unload event occurs, you can verify that the form should be unloaded or specify actions that should take place when the form is unloaded. You can also open another form or display a dialog box requesting the user's name to make a log entry indicating who used the form.

When you first open a form, the following events occur in this order:

Open → Load → Resize → Activate → Current

If you're trying to decide whether to use the Open or Load event for your macro or event procedure, one significant difference is that the Open event can be canceled, but the Load event can't. For example, if you're dynamically building a record source for a form in an event procedure for the form's Open event, you can cancel opening the form if there are no records to display.

When you close a form, the following events occur in this order:

Unload → Deactivate → Close

The Unload event occurs before the Close event. The Unload event can be canceled, but the Close event can't.

Note When you create macros or event procedures for events related to the
Load event, such as **Activate** and **GotFocus**, be sure that they don't conflict (for example, make sure you don't cause something to happen in one macro or procedure that is canceled in another) and that they don't cause **cascading events**.
Macro

You can use a Load macro to carry out actions for the controls or records on a form after the form opens. For example, you can specify default settings for controls.

You can't use the CancelEvent action in a Load macro.

You can use the CancelEvent action in an Unload macro to cancel unloading of the records. This also cancels closing of the form.

If a form's Unload macro carries out a CancelEvent action, you won't be able to close the form. You must either correct the condition that caused the CancelEvent action to be carried out or open the macro and delete the CancelEvent action. If the form is modal, you won't be able to open the macro or work in any other windows in the application.
**Example**

The following example displays the current date in the form's caption when the form is loaded.

To try the example, add the following event procedure to a form:

```vbnet
Private Sub Form_Load()
    Me.Caption = Date
End Sub
```
LostFocus Event

The LostFocus event occurs when a form or control loses the focus.

Private Sub object_LostFocus()

Object The name of a Form or a control on a Form.
Remarks

To run a macro or event procedure when these events occur, set the OnLostFocus property to the name of the macro or to [Event Procedure].

This event occurs when the focus moves in response to a user action, such as pressing the TAB key or clicking the object, or when you use the SetFocus method in Visual Basic or the SelectObject, GoToRecord, GoToControl, or GoToPage action in a macro.

A control can receive the focus only if its Visible and Enabled properties are set to Yes. A form can receive the focus only if it has no controls or if all visible controls are disabled. If a form contains any visible, enabled controls, the GotFocus event for the form doesn't occur.

Note  To customize the order in which the focus moves from control to control on a form when you press the TAB key, set the tab order or specify access keys for the controls.

The LostFocus event differs from the Exit event in that the LostFocus event occurs every time a control loses the focus. The Exit event occurs only before a control loses the focus to another control on the same form. The LostFocus event occurs after the Exit event.

If you move the focus to a control on a form, and that control doesn't have the focus on that form, the Exit and LostFocus events for the control that does have the focus on the form occur before the Enter and GotFocus events for the control you moved to.

If you use the mouse to move the focus from a control on a main form to a control on a subform of that form, the following events occur:

Exit (for the control on the main form)

LostFocus (for the control on the main form)
Enter (for the subform control)

Exit (for the control on the subform that had the focus)

LostFocus (for the control on the subform that had the focus)

Enter (for the control on the subform that the focus moved to)

GotFocus (for the control on the subform that the focus moved to)

If the control you move to on the subform previously had the focus, neither its Enter event nor its GotFocus event occurs, but the Enter event for the subform control does occur. If you move the focus from a control on a subform to a control on the main form, the Exit and LostFocus events for the control on the subform don't occur, just the Exit event for the subform control and the Enter and GotFocus events for the control on the main form.

**Note** You often use the mouse or a key such as TAB to move the focus to another control. This causes mouse or keyboard events to occur in addition to the events discussed in this topic.

When you switch between two open forms, the **Deactivate** event occurs for the first form, and the **Activate** event occurs for the second form. If the forms contain no visible, enabled controls, the LostFocus event occurs for the first form before the Deactivate event, and the GotFocus event occurs for the second form after the Activate event.
Macro

You can use a LostFocus macro to validate entered data as the user moves the focus from a control. You can also reverse or change conditions that you set up in the object's GotFocus macro.

Other uses for LostFocus and GotFocus macros are enabling, disabling, displaying, and hiding other objects.

You can't use the CancelEvent action in a GotFocus or LostFocus macro.
**Example**

The following example displays a message in a label when the focus moves to an option button.

To try the example, add the following event procedures to a form named Contacts that contains an option button named OptionYes and a label named LabelYes.

```vbnet
Private Sub OptionYes_GotFocus()
    Me!LabelYes.Caption = "Option button 'Yes' has the focus."
End Sub

Private Sub OptionYes_LostFocus()
    Me!LabelYes.Caption = "" ' Clear caption.
End Sub
```
MouseDown Event

The MouseDown event occurs when the user presses a mouse button.

Private Sub object_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As Single)

Object    The name of a Form, Form section, or control on a Form.

Button    The button that was pressed (MouseDown) or released (MouseUp) to trigger the event. If you need to test for the Button argument, you can use one of the following intrinsic constants as bit masks:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>acLeftButton</td>
<td>The bit mask for the left mouse button.</td>
</tr>
<tr>
<td>acRightButton</td>
<td>The bit mask for the right mouse button.</td>
</tr>
<tr>
<td>acMiddleButton</td>
<td>The bit mask for the middle mouse button.</td>
</tr>
</tbody>
</table>

Shift    The state of the SHIFT, CTRL, and ALT keys when the button specified by the Button argument was pressed or released. If you need to test for the Shift argument, you can use one of the following intrinsic constants as bit masks:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>acShiftMask</td>
<td>The bit mask for the SHIFT key.</td>
</tr>
<tr>
<td>acCtrlMask</td>
<td>The bit mask for the CTRL key.</td>
</tr>
<tr>
<td>acAltMask</td>
<td>The bit mask for the ALT key.</td>
</tr>
</tbody>
</table>

X, Y      The x and y coordinates for the current location of the mouse pointer. The X and Y arguments are always expressed in twips.
Remarks

- The MouseDown event applies only to forms, form sections, and controls on a form, not controls on a report.
- This event does not apply to a label attached to another control, such as the label for a text box. It applies only to "freestanding" labels. Pressing and releasing a mouse button in an attached label has the same effect as pressing and releasing the button in the associated control. The normal events for the control occur; no separate events occur for the attached label.

To run a macro or event procedure when these events occur, set the OnMouseDown property to the name of the macro or to [Event Procedure].

You can use a MouseDown event to specify what happens when a particular mouse button is pressed or released. Unlike the Click and DblClick events, the MouseDown event enables you to distinguish between the left, right, and middle mouse buttons. You can also write code for mouse-keyboard combinations that use the SHIFT, CTRL, and ALT keys.

To cause a MouseDown event for a form to occur, press the mouse button in a blank area or record selector on the form. To cause a MouseDown event for a form section to occur, press the mouse button in a blank area of the form section.

The following apply to MouseDown events:

- If a mouse button is pressed while the pointer is over a form or control, that object receives all mouse events up to and including the last MouseUp event.
- If mouse buttons are pressed in succession, the object that receives the mouse event after the first press receives all mouse events until all buttons are released.

To respond to an event caused by moving the mouse, you use a MouseMove event.
Macro

You can use a MouseDown or MouseUp macro to respond whenever the user presses or releases a mouse button in a form or control. However, macros can't return the button code and determine which mouse button was pressed, so you typically use event procedures with this event.

You can't use the CancelEvent action in a MouseDown or MouseUp macro, with one exception: You can use the CancelEvent action in a MouseDown macro to cancel the event that occurs when you press the right mouse button. For example, you can cancel the display of a shortcut menu and display your own custom shortcut menu.
**Example**

The following example shows how you can find out which mouse button caused a MouseDown event.

To try the example, add the following event procedure to a form:

```vbnet
Private Sub Form_MouseDown(Button As Integer, _
                     Shift As Integer, X As Single, _
                     Y As Single)
    If Button = acLeftButton Then
        MsgBox "You pressed the left button."
    End If
    If Button = acRightButton Then
        MsgBox "You pressed the right button."
    End If
    If Button = acMiddleButton Then
        MsgBox "You pressed the middle button."
    End If
End Sub
```
**MouseMove Event**

The MouseMove event occurs when the user moves the mouse.

**Private Sub object_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As Single)**

*Object*  The name of a Form, Form section, or control on a Form.

*Button*  The button that was pressed (MouseDown) or released (MouseUp) to trigger the event. If you need to test for the Button argument, you can use one of the following intrinsic constants as bit masks:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>acLeftButton</td>
<td>The bit mask for the left mouse button.</td>
</tr>
<tr>
<td>acRightButton</td>
<td>The bit mask for the right mouse button.</td>
</tr>
<tr>
<td>acMiddleButton</td>
<td>The bit mask for the middle mouse button.</td>
</tr>
</tbody>
</table>

*Shift*  The state of the SHIFT, CTRL, and ALT keys when the button specified by the Button argument was pressed or released. If you need to test for the Shift argument, you can use one of the following intrinsic constants as bit masks:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>acShiftMask</td>
<td>The bit mask for the SHIFT key.</td>
</tr>
<tr>
<td>acCtrlMask</td>
<td>The bit mask for the CTRL key.</td>
</tr>
<tr>
<td>acAltMask</td>
<td>The bit mask for the ALT key.</td>
</tr>
</tbody>
</table>

*X, Y*  The x and y coordinates for the current location of the mouse pointer. The X and Y arguments are always expressed in twips.
Remarks

The MouseMove event applies only to forms, form sections, and controls on a form, not controls on a report.

This event doesn't apply to a label attached to another control, such as the label for a text box. It applies only to "freestanding" labels. Moving the mouse pointer over an attached label has the same effect as moving the pointer over the associated control. The normal events for the control occur; no separate events occur for the attached label.

To run a macro or event procedure when this event occurs, set the OnMouseMove property to the name of the macro or to [Event Procedure].

The MouseMove event is generated continually as the mouse pointer moves over objects. Unless another object generates a mouse event, an object recognizes a MouseMove event whenever the mouse pointer is positioned within its borders.

To cause a MouseMove event for a form to occur, move the mouse pointer over a blank area, record selector, or scroll bar on the form. To cause a MouseMove event for a form section to occur, move the mouse pointer over a blank area of the form section.

Notes

- Moving a form can trigger a MouseMove event even if the mouse is stationary. MouseMove events are generated when the form moves underneath the pointer. If a macro or event procedure moves a form in response to a MouseMove event, the event can cascade (that is, continually generate MouseMove events).

- If two controls are very close together, and you move the mouse pointer quickly over the space between them, the MouseMove event may not occur for the space (for example, this might be the MouseMove event for the form section). In such cases, you may need to respond to the MouseMove event in the contiguous control, as well as in the form section.

To run a macro or event procedure in response to pressing and releasing the
mouse buttons, you use the **MouseDown** and **MouseUp** events.
**Macro**

You can use a MouseMove macro to respond whenever the user moves the mouse over a form or control. However, macros can't return the button code and determine the state of the mouse buttons or determine the coordinates of the mouse pointer when the movement occurs, so you typically use event procedures with this event.

You can't use the CancelEvent action in a MouseMove macro.
Example

The following example determines where the mouse is and whether the left mouse button and/or the SHIFT key is pressed. The x and y coordinates of the mouse pointer position are displayed in a label control as you move the mouse.

To try the example, add the following event procedure to a form that contains a label named Coordinates:

Private Sub Detail_MouseMove(Button As Integer, _
    Shift As Integer, X As Single, Y As Single)
    Dim intShiftDown As Integer, intLeftButton As Integer
    Me!Coordinates.Caption = X & ", " & Y
    ' Use bit masks to determine state of
    ' SHIFT key and left button.
    intShiftDown = Shift And acShiftMask
    intLeftButton = Button And acLeftButton
    ' Check that SHIFT key and left button
    ' are both pressed.
    If intShiftDown And intLeftButton > 0 Then
        MsgBox "Shift key and left mouse button were pressed."
    End If
End Sub
MouseUp Event

The MouseUp event occurs when the user releases a mouse button.

Private Sub object_MouseUp(Button As Integer, Shift As Integer, X As Single, Y As Single)

Object The name of a Form, Form section, or control on a Form.

Button The button that was pressed (MouseDown) or released (MouseUp) to trigger the event. If you need to test for the Button argument, you can use one of the following intrinsic constants as bit masks:

Constant Description

acLeftButton The bit mask for the left mouse button.

acRightButton The bit mask for the right mouse button.

acMiddleButton The bit mask for the middle mouse button.

Shift The state of the SHIFT, CTRL, and ALT keys when the button specified by the Button argument was pressed or released. If you need to test for the Shift argument, you can use one of the following intrinsic constants as bit masks:

Constant Description

acShiftMask The bit mask for the SHIFT key.

acCtrlMask The bit mask for the CTRL key.

acAltMask The bit mask for the ALT key.

X, Y The x and y coordinates for the current location of the mouse pointer. The X and Y arguments are always expressed in twips.
Remarks

- The MouseUp events apply only to forms, form sections, and controls on a form, not controls on a report.
- These events don't apply to a label attached to another control, such as the label for a text box. They apply only to "freestanding" labels. Pressing and releasing a mouse button in an attached label has the same effect as pressing and releasing the button in the associated control. The normal events for the control occur; no separate events occur for the attached label.
**Remarks**

To run a macro or event procedure when these events occur, set the **OnMouseUp** property to the name of the macro or to [Event Procedure].

You can use a MouseUp event to specify what happens when a particular mouse button is pressed or released. Unlike the **Click** and **DblClick** events, the MouseUp event enables you to distinguish between the left, right, and middle mouse buttons. You can also write code for mouse-keyboard combinations that use the SHIFT, CTRL, and ALT keys.

To cause a MouseUp event for a form to occur, press the mouse button in a blank area or **record selector** on the form. To cause a MouseUp event for a form section to occur, press the mouse button in a blank area of the form section.

The following applies to MouseUp events:

- If a mouse button is pressed while the pointer is over a form or control, that object receives all mouse events up to and including the last MouseUp event.
- If mouse buttons are pressed in succession, the object that receives the mouse event after the first press receives all mouse events until all buttons are released.

To respond to an event caused by moving the mouse, you use a **MouseMove** event.
Macro

You can use a MouseDown or MouseUp macro to respond whenever the user presses or releases a mouse button in a form or control. However, macros can't return the button code and determine which mouse button was pressed, so you typically use event procedures with this event.

You can't use the CancelEvent action in a MouseDown or MouseUp macro, with one exception: You can use the CancelEvent action in a MouseDown macro to cancel the event that occurs when you press the right mouse button. For example, you can cancel the display of a shortcut menu and display your own custom shortcut menu.
MouseWheel Event

Occurs when the user rolls the mouse wheel in Form View, Datasheet View, PivotChart View, or PivotTable View.

Private Sub Form_MouseWheel(ByVal Page As Boolean, ByVal Count As Long)

*Page True* if the page was changed.

*Count*  The number of lines by which the view was scrolled with the mouse wheel.
Example

The following example demonstrates the syntax for a subroutine that traps the MouseWheel event.

Private Sub Form_MouseWheel(_
    ByVal Page As Boolean, ByVal Count As Long)
    If Page = True Then
        MsgBox "You've moved to another page."
    End If
End Sub
NoData Event

The NoData event occurs after Microsoft Access formats a report for printing that has no data (the report is bound to an empty recordset), but before the report is printed. You can use this event to cancel printing of a blank report.

Private Sub Report_NoData(Cancel As Integer)

    Cancel  The setting determines whether to print the report. Setting the Cancel argument to True (–1) prevents the report from printing. You can also use the CancelEvent method of the DoCmd object to cancel printing the report.
**Remarks**

To run a [macro](#) or [event procedure](#) when this event occurs, set the [OnNoData](#) property to the name of the macro or to [Event Procedure](#).

If the report isn't bound to a [table](#) or [query](#) (by using the report's [RecordSource](#) property), the NoData event doesn't occur.

This event occurs after the [Format](#) events for the report, but before the first [Print](#) event.

This event doesn't occur for [subreports](#). If you want to hide [controls](#) on a subreport when the subreport has no data, so that the controls don't print in this case, you can use the [HasData](#) property in a macro or event procedure that runs when the Format or Print event occurs.

The NoData event occurs before the first [Page](#) event for the report.
**Example**

The following example shows how to cancel printing a report when it has no data. A message box notifying the user that the printing has been canceled is also displayed.

To try this example, add the following event procedure to a report. Try running the report when it contains no data.

```vba
Private Sub Report_NoData(Cancel As Integer)
    MsgBox "The report has no data." & vbCrLf &
        "Printing is canceled. " & vbCrLf &
        "Check the data source for the " & vbCrLf &
        "report. Make sure you entered " & vbCrLf &
        "the correct criteria (for " & vbCrLf &
        "example, a valid range of " & vbCrLf &
        "dates),." vbOKOnly + vbInformation
    Cancel = True
End Sub
```
NotInList Event

The NotInList event occurs when the user enters a value in the text box portion of a combo box that isn't in the combo box list.

Private Sub object_NotInList(NewData As String, Response As Integer)

Object The name of a combo box control.

NewData A string that Microsoft Access uses to pass the text the user entered in the text box portion of the combo box to the event procedure.

Response The setting indicates how the NotInList event was handled. The Response argument can be one of the following intrinsic constants:

Constant Description

acDataErrDisplay (Default) Displays the default message to the user. You can use this when you don't want to allow the user to add a new value to the combo box list.

acDataErrContinue Doesn't display the default message to the user. You can use this when you want to display a custom message to the user. For example, the event procedure could display a custom dialog box asking if the user wanted to save the new entry. If the response is Yes, the event procedure would add the new entry to the list and set the Response argument to acDataErrAdded. If the response is No, the event procedure would set the Response argument to acDataErrContinue.

acDataErrAdded Doesn't display a message to the user but enables you to add the entry to the combo box list in the NotInList event procedure. After the entry is added, Microsoft Access updates the list by requerying the combo box. Microsoft Access then rechecks the string against the combo box list, and saves the value in the NewData argument in the field the combo box is bound to. If the string is not in the list, then Microsoft Access displays an error message.
Remarks

The NotInList event applies only to controls on a form, not controls on a report.

To run a macro or event procedure when this event occurs, set the OnNotInList property to the name of the macro or to [Event Procedure].

This event enables the user to add a new value to the combo box list.

The LimitToList property must be set to Yes for the NotInList event to occur.

The NotInList event doesn't trigger the Error event.

The NotInList event occurs for combo boxes whose LimitToList property is set to Yes, after you enter a value that isn't in the list and attempt to move to another control or save the record. The event occurs after all the Change events for the combo box.

When the AutoExpand property is set to Yes, Microsoft Access selects matching values in the list as the user enters characters in the text box portion of the combo box. If the characters the user types match the first characters of a value in the list (for example, the user types "Smith" and "Smithson" is a value in the list), the NotInList event will not occur when the user moves to another control or saves the record. However, the characters that Microsoft Access adds to the characters the user types (in the example, "son") are selected in the text box portion of the combo box. If the user wants the NotInList event to fire in such cases (for example, the user wants to add the new name "Smith" to the combo box list), the user can enter a SPACE, BACKSPACE, or DEL character after the last character in the new value.

When the LimitToList property is set to Yes and the combo box list is dropped down, Microsoft Access selects matching values in the list as the user enters characters in the text box portion of the combo box, even if the AutoExpand property is set to No. If the user presses ENTER or moves to another control or record, the selected value appears in the combo box. In this case, the NotInList event will not fire. To allow the NotInList event to fire, the user should not drop down the combo box list.
**Macro**

You can use a NotInList macro to add a new value to the combo box list. When the NotInList event occurs, open a custom dialog box and set the value of one of its controls to the new value. This control should be bound to the field in the table or query that is the source for the combo box. Save the record in the custom dialog box, then requery the combo box.

You can't use the CancelEvent action in a NotInList macro.
Example

The following example uses the NotInList event to add an item to a combo box.

To try this example, create a combo box called Colors on a form. Set the combo box's LimitToList property to Yes. To populate the combo box, set the combo box's RowSourceType property to Value List, and supply a list of values separated by semicolons as the setting for the RowSource property. For example, you might supply the following values as the setting for this property: Red; Green; Blue.

Next add the following event procedure to the form. Switch to Form view and enter a new value in the text portion of the combo box.

Private Sub Colors_NotInList(NewData As String, _
    Response As Integer)
    Dim ctl As Control
    ' Return Control object that points to combo box.
    Set ctl = Me!Colors
    ' Prompt user to verify they wish to add new value.
    If MsgBox("Value is not in list. Add it?", _
        vbOKCancel) = vbOK Then
      ' Set Response argument to indicate that data
      ' is being added.
      Response = acDataErrAdded
      ' Add string in NewData argument to row source.
      ctl.RowSource = ctl.RowSource & ";" & NewData
    Else
      ' If user chooses Cancel, suppress error message
      ' and undo changes.
      Response = acDataErrContinue
      ctl.Undo
    End If
End Sub

Note The above example adds an item to an unbound combo box. When you add an item to a bound combo box, you add a value to a field in the underlying data source. In most cases you can't simply add one field in a new record — depending on the structure of data in the table, you probably will need to add one or more fields to fulfill data requirements. For instance, a new record must
include values for any fields comprising the primary key. If you need to add items to a bound combo box dynamically, you must prompt the user to enter data for all required fields, save the new record, and then requery the combo box to display the new value.
OnConnect Event

Occurs when the specified PivotTable view connects to a data source.

Private Sub Form_OnConnect()
Example

The following example demonstrates the syntax for a subroutine that traps the OnConnect event.

Private Sub Form_OnConnect()
    MsgBox "The PivotTable View has " & "connected to its data source!"
End Sub
OnDisconnect Event

Occurs when the specified PivotTable view disconnects from a data source.

Private Sub Form_OnDisconnect()
Example

The following example demonstrates the syntax for a subroutine that traps the OnDisconnect event.

Private Sub Form_{OnDisconnect}()
    MsgBox "The PivotTable View has " _
         & "disconnected from its data source!"
End Sub
Open Event

The Open event occurs when a form is opened, but before the first record is displayed. For reports, the event occurs before a report is previewed or printed.

Private Sub object_Open(Cancel As Integer)

Object  The name of a Form or Report.

Cancel  The setting determines if the opening of the form or report occurs. Setting the Cancel argument to True (–1) cancels the opening of the form or report.
Remarks

To run a macro or event procedure when these events occur, set the OnOpen property to the name of the macro or to [Event Procedure].

By running a macro or an event procedure when a form's Open event occurs, you can close another window or move the focus to a particular control on a form. You can also run a macro or an event procedure that asks for information needed before the form or report is opened or printed. For example, an Open macro or event procedure can open a custom dialog box in which the user enters the criteria for the set of records to display on a form or the date range to include for a report.

The Open event doesn't occur when you activate a form that's already open— for example, when you switch to the form from another window in Microsoft Access or use the OpenForm action in a macro to bring the open form to the top. However, the Activate event does occur in these situations.

When you open a form based on an underlying query, Microsoft Access runs the underlying query for the form before it runs the Open macro or event procedure. However, when you open a report based on an underlying query, Microsoft Access runs the Open macro or event procedure before it runs the underlying query for the report. This enables the user to specify criteria for the report before it opens— for example, in a custom dialog box you display when the Open event occurs.

If your application can have more than one form loaded at a time, use the Activate and Deactivate events instead of the Open event to display and hide custom toolbars when the focus moves to a different form.

The Open event occurs before the Load event, which is triggered when a form is opened and its records are displayed.

When you first open a form, the following events occur in this order:

Open → Load → Resize → Activate → Current

The Close event occurs after the Unload event, which is triggered after the form
is closed but before it is removed from the screen.

When you close a form, the following events occur in this order:

Unload → Deactivate → Close

When the Close event occurs, you can open another window or request the user's name to make a log entry indicating who used the form or report.

If you're trying to decide whether to use the Open or Load event for your macro or event procedure, one significant difference is that the Open event can be canceled, but the Load event can't. For example, if you're dynamically building a record source for a form in an event procedure for the form's Open event, you can cancel opening the form if there are no records to display. Similarly, the Unload event can be canceled, but the Close event can't.
Macro

You can use the CancelEvent action in an Open macro to cancel opening of the form or report. You can't use the CancelEvent action in a Close macro to cancel the Close event. However, you can use the CancelEvent action in a macro that runs when the Unload event occurs on a form. This cancels closing of the form.

If you want to refer to controls from an Open macro, you must first move the focus to the appropriate control or record. For example, to use a SetValue action in an Open macro to set the value of a control on a form, you must first use the GoToControl or GoToRecord action to access the control or the record containing the control.
Example

The following example shows how you can cancel the opening of a form when the user clicks a No button. A message box prompts the user to enter order details. If the user clicks No, the Order Details form isn't opened.

To try the example, add the following event procedure to a form.

```vba
Private Sub Form_Open(Cancel As Integer)
    Dim intReturn As Integer
    intReturn = MsgBox("Enter order details now?", vbYesNo)
    Select Case intReturn
        Case vbYes
            ' Open Order Details form.
            DoCmd.OpenForm "Order Details"
        Case vbNo
            MsgBox "Remember to enter order details by 5 P.M."
            Cancel = True  ' Cancel Open event.
    End Select
End Sub
```
Show All
Page Event

The Page event occurs after Microsoft Access formats a page of a report for printing, but before the page is printed. You can use this event to draw a border around the page, or add other graphic elements to the page.

Private Sub Report_Page()
Remarks

To run a macro or event procedure when this event occurs, set the OnPage property to the name of the macro or to [Event Procedure].

This event occurs after all the Format events for the report, and after all the Print events for the page, but before the page is actually printed.

You normally use the Line, Circle, or PSet methods in the Page event procedure to create the desired graphics for the page.

The NoData event occurs before the first Page event for the report.
Example

The following example shows how to draw a rectangle around a report page by using the Line method. The ScaleWidth and ScaleHeight properties by default return the internal width and height of the report.

Private Sub Report_Page()
    Me.Line(0, 0)-(Me.ScaleWidth, Me.ScaleHeight), , B
End Sub
PivotTableChange Event

Occurs whenever the specified PivotTable view field, field set, or total is added or deleted.

Private Sub Form_PivotTableChange(ByVal Reason As Long)

  *Reason* Specifies how the PivotTable list changed. Can be one of the *PivotTableReasonEnum* constants.

  plPivotTableReasonTotalAdded
  plPivotTableReasonFieldSetAdded
  plPivotTableReasonFieldAdded
Example

The following example demonstrates the syntax for a subroutine that traps the PivotTableChange event. For this example to work, a reference must be set to the Microsoft Office Web Components 10.0 type library.

Private Sub Form_PivotTableChange(Reason As Long)
    Select Case Reason
        Case OWC.plPivotTableReasonTotalAdded
            MsgBox "A total was added!"
        Case OWC.plPivotTableReasonFieldSetAdded
            MsgBox "A field set was added!"
        Case OWC.plPivotTableReasonFieldAdded
            MsgBox "A field was added!"
    End Select
End Sub
Query Event

Occurs whenever the specified PivotTable view query becomes necessary. The query may not occur immediately; it may be delayed until the new data is displayed.

Private Sub Form_Query()
Example

The following example demonstrates the syntax for a subroutine that traps the Query event.

Private Sub Form_Query()
    MsgBox "A query has become necessary."
End Sub
**Resize Event**

The Resize event occurs when a *form* is opened and whenever the size of a form changes.

```vbnet
Private Sub Form_Resize()
```

**Remarks**

To run a [macro](https://www.example.com) or [event procedure](https://www.example.com) when this event occurs, set the [OnResize](https://www.example.com) property to the name of the macro or to [Event Procedure].

This event occurs if you change the size of the form in a macro or event procedure— for example, if you use the [MoveSize](https://www.example.com) action in a macro to resize the form.

By running a macro or an event procedure when a Resize event occurs, you can move or resize a [control](https://www.example.com) when the form it's on is resized. You can also use a Resize event to recalculate variables or reset properties that may depend on the size of the form.

When you first open a form, the following events occur in this order:

- Open → Load → Resize → Activate → Current

**Note** You need to be careful if you use a [MoveSize](https://www.example.com), [Maximize](https://www.example.com), [Minimize](https://www.example.com), or [Restore](https://www.example.com) action (or the corresponding methods of the [DoCmd](https://www.example.com) object) in a Resize macro or event procedure. These actions can trigger a Resize event for the form, and thus cause a [cascading event](https://www.example.com).
Example

The following example shows how a Resize event procedure can be used to repaint a form when it is maximized. When the user clicks a command button labeled "Maximize," the form is maximized and the Resize event is triggered.

To try the example, add the following event procedures to a form named Contacts that contains a command button named Maximize:

Private Sub Maximize_Click()
    DoCmd.Maximize
End Sub

Private Sub Form_Resize()
    Forms!Contacts.Repaint
End Sub
SelectionChange Event

Occurs whenever the user makes a new selection in a PivotChart view or PivotTable view.

Private Sub Form_SelectionChange()
Remarks

The user cannot cancel this event.
Example

The following example demonstrates the syntax for a subroutine that traps the SelectionChange event.

```vbnet
Private Sub Form_SelectionChange()
    MsgBox "The selection has changed!"
End Sub
```
Timer Event

The Timer event occurs for a form at regular intervals as specified by the form's TimerInterval property.

Private Sub Form_Timer()
Remarks

To run a macro or event procedure when this event occurs, set the OnTimer property to the name of the macro or to [Event Procedure].

By running a macro or event procedure when a Timer event occurs, you can control what Microsoft Access does at every timer interval. For example, you might want to requery underlying records or repaint the screen at specified intervals.

The TimerInterval property setting of the form specifies the interval, in milliseconds, between Timer events. The interval can be between 0 and 2,147,483,647 milliseconds. Setting the TimerInterval property to 0 prevents the Timer event from occurring.
Example

The following example demonstrates a digital clock you can display on a form. A label control displays the current time according to your computer's system clock.

To try the example, add the following event procedure to a form that contains a label named Clock. Set the form's TimerInterval property to 1000 milliseconds to update the clock every second.

```vba
Private Sub Form_Timer()
    Clock.Caption = Time ' Update time display.
End Sub
```
Undo Event

Occurs when the user undoes a change to a combo box control, a form, or a text box control.

Private Sub object_Undo(Cancel As Integer)

object A variable which references an object of one of the types in the Applies To list.

Cancel Set this argument to True to cancel the undo operation and leave the control or form in its edited state.
Remarks

The Undo event for controls occurs whenever the user returns a control to its original state by clicking the **Undo Field/Record** button on the command bar, clicking the **Undo** button, pressing the ESC key, or calling the **Undo** method of the specified control. The control needs to have focus in all three cases. The event does not occur if the user clicks the **Undo Typing** button on the command bar.

The Undo event for forms occurs whenever the user returns a form to its original state by clicking the **Undo** button, pressing the ESC key, or calling the **Undo** method of the specified form.
Example

The following example demonstrates the syntax for a subroutine that traps the Undo event for a form.

Private Sub Form_Undo(Cancel As Integer)
    Dim intResponse As Integer
    Dim strPrompt As String

    strPrompt = "Cancel the undo operation?"

    intResponse = MsgBox(strPrompt, vbYesNo)

    If intResponse = vbYes Then
        Cancel = True
    Else
        Cancel = False
    End If
End Sub
Unload Event

The Unload event occurs after a form is closed but before it's removed from the screen. When the form is reloaded, Microsoft Access redisplaysthe form and reinitializes the contents of all its controls.

Private Sub Form_Unload()
Remarks

To run a macro or event procedure when these events occur, set the **OnUnload** property to the name of the macro or to [Event Procedure].

The Unload event is caused by user actions such as:

- Clicking a Form window's **Close** button or clicking **Close** on the **File** menu or a form's **Control** menu.

- Running the **Close** action in a macro.

- Quitting an application by right-clicking the application's taskbar button and then clicking **Close**.

- Quitting Windows while an application is running.

By running a macro or an event procedure when a form's Unload event occurs, you can verify that the form should be unloaded or specify actions that should take place when the form is unloaded. You can also open another form or display a dialog box requesting the user's name to make a log entry indicating who used the form.

When you close a form, the following events occur in this order:

**Unload** → **Deactivate** → **Close**

The Unload event occurs before the **Close** event. The Unload event can be canceled, but the Close event can't.

**Note**  When you create macros or event procedures for events related to the Load event, such as **Activate** and **GotFocus**, be sure that they don't conflict (for example, make sure you don't cause something to happen in one macro or procedure that is canceled in another) and that they don't cause cascading events.
Example

This example prompts the user to verify that the form should close.

To try the example, add the following event procedure to a form. In Form view, close the form to display the dialog box, and then click Yes or No.

Private Sub Form_Unload(Cancel As Integer)
    If MsgBox("Close form?", vbYesNo) = vbYes Then
        Exit Sub
    Else
        Cancel = True
    End If
End Sub
Show All
Updated Event

Some of the content in this topic may not be applicable to some languages.

The Updated event occurs when an OLE object's data has been modified.
Remarks

The Updated event applies only to controls on a form, not controls on a report.

To run a macro or event procedure when this event occurs, set the OnUpdated property to the name of the macro or to [Event Procedure].

You can use this event to determine if an object's data has been changed since it was last saved.

The Updated event occurs when the data in an OLE object has been modified. This update can come from the application in which the object was created or from one of the linked copies of this object. As a result, this event is asynchronous with other Microsoft Access control events.

Notes

- The Updated event and the BeforeUpdate and AfterUpdate events for bound and unbound object frames are not related. The Updated event occurs when an OLE object's data is changed, and the BeforeUpdate and AfterUpdate events occur when data is updated. Although not related, all three events usually occur when an OLE object's data is changed. The Updated event generally occurs before the BeforeUpdate and AfterUpdate events; however, this may not happen every time.
- The Calendar control included with Microsoft Access 97 no longer supports the Updated event. If you convert a database that includes the Calendar control from a previous version of Microsoft Access to Microsoft Access 97, you should move any code in the Updated event of the Calendar control to the AfterUpdate event.
ViewChange Event

Occurs whenever the specified PivotChart view or PivotTable view is redrawn.

Private Sub Form_ViewChange(ByVal Reason As Long)

Reason  The PivotViewReasonEnum constant that indicates how the view was changed. Reason always returns –1 for PivotChart Views.
Example

The following example demonstrates the syntax for a subroutine that traps the ViewChange event. For this example to work, a reference must be set to the Microsoft Office Web Components 10.0 type library.

Private Sub Form_ViewChange(ByVal Reason As Long)
    If Reason = OWC.plViewReasonShowDetails Then
        MsgBox "You've opted to show details."
    End If
End Sub
Microsoft Access Constants

This topic lists all the constants in the Microsoft Access object model.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>acSaveNo</td>
<td>2</td>
</tr>
<tr>
<td>acSavePrompt</td>
<td>0</td>
</tr>
<tr>
<td>acSaveYes</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>acColorIndexAqua</td>
<td>14</td>
</tr>
<tr>
<td>acColorIndexBlack</td>
<td>0</td>
</tr>
<tr>
<td>acColorIndexBlue</td>
<td>12</td>
</tr>
<tr>
<td>acColorIndexBrightGreen</td>
<td>10</td>
</tr>
<tr>
<td>acColorIndexDarkBlue</td>
<td>4</td>
</tr>
<tr>
<td>acColorIndexFuschia</td>
<td>13</td>
</tr>
<tr>
<td>acColorIndexGray</td>
<td>7</td>
</tr>
<tr>
<td>acColorIndexGreen</td>
<td>2</td>
</tr>
<tr>
<td>acColorIndexMaroon</td>
<td>1</td>
</tr>
<tr>
<td>acColorIndexOlive</td>
<td>3</td>
</tr>
<tr>
<td>acColorIndexRed</td>
<td>9</td>
</tr>
<tr>
<td>acColorIndexSilver</td>
<td>8</td>
</tr>
<tr>
<td>acColorIndexTeal</td>
<td>6</td>
</tr>
<tr>
<td>acColorIndexViolet</td>
<td>5</td>
</tr>
<tr>
<td>acColorIndexWhite</td>
<td>15</td>
</tr>
<tr>
<td>acColorIndexYellow</td>
<td>11</td>
</tr>
</tbody>
</table>

AcCloseSave

AcCommand
<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>acCmdAboutMicrosoftAccess</td>
<td>35</td>
</tr>
<tr>
<td>acCmdAddInManager</td>
<td>526</td>
</tr>
<tr>
<td>acCmdAddToNewGroup</td>
<td>494</td>
</tr>
<tr>
<td>acCmdAddWatch</td>
<td>201</td>
</tr>
<tr>
<td>acCmdAdvancedFilterSort</td>
<td>99</td>
</tr>
<tr>
<td>acCmdAlignBottom</td>
<td>46</td>
</tr>
<tr>
<td>acCmdAlignCenter</td>
<td>477</td>
</tr>
<tr>
<td>acCmdAlignLeft</td>
<td>43</td>
</tr>
<tr>
<td>acCmdAlignmentAndSizing</td>
<td>478</td>
</tr>
<tr>
<td>acCmdAlignMiddle</td>
<td>476</td>
</tr>
<tr>
<td>acCmdAlignRight</td>
<td>44</td>
</tr>
<tr>
<td>acCmdAlignToGrid</td>
<td>47</td>
</tr>
<tr>
<td>acCmdAlignTop</td>
<td>45</td>
</tr>
<tr>
<td>acCmdAlignToShortest</td>
<td>153</td>
</tr>
<tr>
<td>acCmdAlignToTallest</td>
<td>154</td>
</tr>
<tr>
<td>acCmdAnalyzePerformance</td>
<td>283</td>
</tr>
<tr>
<td>acCmdAnalyzeTable</td>
<td>284</td>
</tr>
<tr>
<td>acCmdAnswerWizard</td>
<td>235</td>
</tr>
<tr>
<td>acCmdApplyDefault</td>
<td>55</td>
</tr>
<tr>
<td>acCmdApplyFilterSort</td>
<td>93</td>
</tr>
<tr>
<td>acCmdAppMaximize</td>
<td>10</td>
</tr>
<tr>
<td>acCmdAppMinimize</td>
<td>11</td>
</tr>
<tr>
<td>acCmdAppMove</td>
<td>12</td>
</tr>
<tr>
<td>acCmdAppRestore</td>
<td>9</td>
</tr>
<tr>
<td>acCmdAppSize</td>
<td>13</td>
</tr>
<tr>
<td>acCmdArrangeIconsAuto</td>
<td>218</td>
</tr>
<tr>
<td>acCmdArrangeIconsByCreated</td>
<td>216</td>
</tr>
<tr>
<td>acCmdArrangeIconsByModified</td>
<td>217</td>
</tr>
<tr>
<td>acCmdArrangeIconsByName</td>
<td>214</td>
</tr>
<tr>
<td>acCmdArrangeIconsByType</td>
<td>215</td>
</tr>
<tr>
<td>acCmdAutoCorrect</td>
<td>261</td>
</tr>
<tr>
<td>acCmdAutoDial</td>
<td>192</td>
</tr>
<tr>
<td>acCmdAutoFormat</td>
<td>270</td>
</tr>
<tr>
<td>--------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>acCmdBackgroundPicture</td>
<td>474</td>
</tr>
<tr>
<td>acCmdBackgroundSound</td>
<td>475</td>
</tr>
<tr>
<td>acCmdBackup</td>
<td>513</td>
</tr>
<tr>
<td>acCmdBookmarksClearAll</td>
<td>310</td>
</tr>
<tr>
<td>acCmdBookmarksNext</td>
<td>308</td>
</tr>
<tr>
<td>acCmdBookmarksPrevious</td>
<td>309</td>
</tr>
<tr>
<td>acCmdBookmarksToggle</td>
<td>307</td>
</tr>
<tr>
<td>acCmdBringToFront</td>
<td>52</td>
</tr>
<tr>
<td>acCmdCallStack</td>
<td>172</td>
</tr>
<tr>
<td>acCmdChangeToCheckBox</td>
<td>231</td>
</tr>
<tr>
<td>acCmdChangeToComboBox</td>
<td>230</td>
</tr>
<tr>
<td>acCmdChangeToCommandButton</td>
<td>501</td>
</tr>
<tr>
<td>acCmdChangeToImage</td>
<td>234</td>
</tr>
<tr>
<td>acCmdChangeToLabel</td>
<td>228</td>
</tr>
<tr>
<td>acCmdChangeToListBox</td>
<td>229</td>
</tr>
<tr>
<td>acCmdChangeToOptionButton</td>
<td>233</td>
</tr>
<tr>
<td>acCmdChangeToTextBox</td>
<td>227</td>
</tr>
<tr>
<td>acCmdChangeToToggleButton</td>
<td>232</td>
</tr>
<tr>
<td>acCmdClearAll</td>
<td>146</td>
</tr>
<tr>
<td>acCmdClearAllBreakpoints</td>
<td>132</td>
</tr>
<tr>
<td>acCmdClearGrid</td>
<td>71</td>
</tr>
<tr>
<td>acCmdClearHyperlink</td>
<td>343</td>
</tr>
<tr>
<td>acCmdClearItemDefaults</td>
<td>237</td>
</tr>
<tr>
<td>acCmdClose</td>
<td>58</td>
</tr>
<tr>
<td>acCmdCloseWindow</td>
<td>186</td>
</tr>
<tr>
<td>acCmdColumnWidth</td>
<td>117</td>
</tr>
<tr>
<td>acCmdCompactDatabase</td>
<td>4</td>
</tr>
<tr>
<td>acCmdCompileAllModules</td>
<td>125</td>
</tr>
<tr>
<td>acCmdCompileAndSaveAllModules</td>
<td>126</td>
</tr>
<tr>
<td>acCmdCompileLoadedModules</td>
<td>290</td>
</tr>
<tr>
<td>acCmdCompleteWord</td>
<td>306</td>
</tr>
<tr>
<td>acCmdConditionalFormatting</td>
<td>500</td>
</tr>
</tbody>
</table>
acCmdConnection 383
acCmdControlWizardsToggle 197
acCmdConvertDatabase 171
acCmdConvertMacrosToVisualBasic 279
acCmdCopy 190
acCmdCopyDatabaseFile 516
acCmdCopyHyperlink 328
acCmdCreateMenuFromMacro 334
acCmdCreateRelationship 150
acCmdCreateReplica 263
acCmdCreateShortcut 219
acCmdCreateShortcutMenuFromMacro 336
acCmdCreateToolbarFromMacro 335
acCmdCut 189
acCmdDataAccessPageAddToPage 480
acCmdDataAccessPageBrowse 344
acCmdDataAccessPageDesignView 385
acCmdDataAccessPageFieldListRefresh 479
acCmdDatabaseProperties 256
acCmdDatabaseSplitter 520
acCmdDataEntry 78
acCmdDataOutline 468
acCmdDatasheetView 282
acCmdDateAndTime 226
acCmdDebugWindow 123
acCmdDelete 337
acCmdDeleteGroup 493
acCmdDeletePage 332
acCmdDeleteQueryColumn 81
acCmdDeleteRecord 223
acCmdDeleteRows 188
acCmdDeleteTab 255
acCmdDeleteTable 489
acCmdDeleteTableColumn 271
acCmdDeleteWatch 267
acCmdDemote 388
acCmdDesignView 183
acCmdDiagramAddRelatedTables 373
acCmdDiagramAutosizeSelectedTables 378
acCmdDiagramDeleteRelationship 382
acCmdDiagramLayoutDiagram 380
acCmdDiagramLayoutSelection 379
acCmdDiagramModifyUserDefinedView 375
acCmdDiagramNewLabel 372
acCmdDiagramNewTable 381
acCmdDiagramRecalculatePageBreaks 377
acCmdDiagramShowRelationshipLabels 374
acCmdDiagramViewPageBreaks 376
acCmdDocMaximize 15
acCmdDocMinimize 60
acCmdDocMove 16
acCmdDocRestore 14
acCmdDocSize 17
acCmdDocumenter 285
acCmdDropSQLDatabase 517
acCmdDuplicate 34
acCmdEditHyperlink 325
acCmdEditingAllowed 70
acCmdEditRelationship 151
acCmdEditTriggers 384
acCmdEditWatch 202
acCmdEncryptDecryptDatabase 5
acCmdEnd 198
acCmdExit 3
acCmdExport 487
acCmdFavoritesAddTo 299
<table>
<thead>
<tr>
<th>.Ribbon Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>acCmdFavoritesOpen</td>
<td>298</td>
</tr>
<tr>
<td>acCmdFieldList</td>
<td>42</td>
</tr>
<tr>
<td>acCmdFilterByForm</td>
<td>207</td>
</tr>
<tr>
<td>acCmdFilterBySelection</td>
<td>208</td>
</tr>
<tr>
<td>acCmdFilterExcludingSelection</td>
<td>277</td>
</tr>
<tr>
<td>acCmdFind</td>
<td>30</td>
</tr>
<tr>
<td>acCmdFindNext</td>
<td>341</td>
</tr>
<tr>
<td>acCmdFindNextWordUnderCursor</td>
<td>313</td>
</tr>
<tr>
<td>acCmdFindPrevious</td>
<td>120</td>
</tr>
<tr>
<td>acCmdFindPrevWordUnderCursor</td>
<td>312</td>
</tr>
<tr>
<td>acCmdFitToWindow</td>
<td>245</td>
</tr>
<tr>
<td>acCmdFont</td>
<td>19</td>
</tr>
<tr>
<td>acCmdFormatCells</td>
<td>77</td>
</tr>
<tr>
<td>acCmdFormHdrFtr</td>
<td>36</td>
</tr>
<tr>
<td>acCmdFormView</td>
<td>281</td>
</tr>
<tr>
<td>acCmdFreezeColumn</td>
<td>105</td>
</tr>
<tr>
<td>acCmdGoBack</td>
<td>294</td>
</tr>
<tr>
<td>acCmdGoContinue</td>
<td>127</td>
</tr>
<tr>
<td>acCmdGoForward</td>
<td>295</td>
</tr>
<tr>
<td>acCmdGroupByTable</td>
<td>387</td>
</tr>
<tr>
<td>acCmdGroupControls</td>
<td>484</td>
</tr>
<tr>
<td>acCmdHideColumns</td>
<td>79</td>
</tr>
<tr>
<td>acCmdHidePane</td>
<td>365</td>
</tr>
<tr>
<td>acCmdHideTable</td>
<td>147</td>
</tr>
<tr>
<td>acCmdHorizontalSpacingDecrease</td>
<td>158</td>
</tr>
<tr>
<td>acCmdHorizontalSpacingIncrease</td>
<td>159</td>
</tr>
<tr>
<td>acCmdHorizontalSpacingMakeEqual</td>
<td>157</td>
</tr>
<tr>
<td>acCmdHyperlinkDisplayText</td>
<td>329</td>
</tr>
<tr>
<td>acCmdImport</td>
<td>257</td>
</tr>
<tr>
<td>acCmdIndent</td>
<td>205</td>
</tr>
<tr>
<td>acCmdIndexes</td>
<td>152</td>
</tr>
<tr>
<td>acCmdInsertActiveXControl</td>
<td>258</td>
</tr>
<tr>
<td>acCmdInsertChart</td>
<td>293</td>
</tr>
<tr>
<td>Command</td>
<td>Count</td>
</tr>
<tr>
<td>--------------------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>acCmdInsertFile</td>
<td>39</td>
</tr>
<tr>
<td>acCmdInsertFileIntoModule</td>
<td>118</td>
</tr>
<tr>
<td>acCmdInsertHyperlink</td>
<td>259</td>
</tr>
<tr>
<td>acCmdInsertLookupColumn</td>
<td>273</td>
</tr>
<tr>
<td>acCmdInsertLookupField</td>
<td>291</td>
</tr>
<tr>
<td>acCmdInsertMovieFromFile</td>
<td>469</td>
</tr>
<tr>
<td>acCmdInsertObject</td>
<td>33</td>
</tr>
<tr>
<td>acCmdInsertPage</td>
<td>331</td>
</tr>
<tr>
<td>acCmdInsertPicture</td>
<td>222</td>
</tr>
<tr>
<td>acCmdInsertPivotTable</td>
<td>470</td>
</tr>
<tr>
<td>acCmdInsertProcedure</td>
<td>262</td>
</tr>
<tr>
<td>acCmdInsertQueryColumn</td>
<td>82</td>
</tr>
<tr>
<td>acCmdInsertRows</td>
<td>187</td>
</tr>
<tr>
<td>acCmdInsertSpreadsheet</td>
<td>471</td>
</tr>
<tr>
<td>acCmdInsertSubdatasheet</td>
<td>499</td>
</tr>
<tr>
<td>acCmdInsertTableColumn</td>
<td>272</td>
</tr>
<tr>
<td>acCmdInsertUnboundSection</td>
<td>472</td>
</tr>
<tr>
<td>acCmdInvokeBuilder</td>
<td>178</td>
</tr>
<tr>
<td>acCmdJoinProperties</td>
<td>72</td>
</tr>
<tr>
<td>acCmdLastPosition</td>
<td>339</td>
</tr>
<tr>
<td>acCmdLayoutPreview</td>
<td>141</td>
</tr>
<tr>
<td>acCmdLineUpIcons</td>
<td>213</td>
</tr>
<tr>
<td>acCmdLinkedTableManager</td>
<td>519</td>
</tr>
<tr>
<td>acCmdLinkTables</td>
<td>102</td>
</tr>
<tr>
<td>acCmdListConstants</td>
<td>303</td>
</tr>
<tr>
<td>acCmdLoadFromQuery</td>
<td>95</td>
</tr>
<tr>
<td>acCmdMacroConditions</td>
<td>87</td>
</tr>
<tr>
<td>acCmdMacroNames</td>
<td>86</td>
</tr>
<tr>
<td>acCmdMakeMDEFile</td>
<td>7</td>
</tr>
<tr>
<td>acCmdMaximumRecords</td>
<td>508</td>
</tr>
<tr>
<td>acCmdMicrosoftAccessHelpTopics</td>
<td>100</td>
</tr>
<tr>
<td>acCmdMicrosoftOnTheWeb</td>
<td>236</td>
</tr>
<tr>
<td>acCmdMicrosoftScriptEditor</td>
<td>390</td>
</tr>
<tr>
<td>acCmdMoreWindows</td>
<td>8</td>
</tr>
<tr>
<td>---------------------------</td>
<td>--</td>
</tr>
<tr>
<td>acCmdNewDatabase</td>
<td>26</td>
</tr>
<tr>
<td>acCmdNewGroup</td>
<td>491</td>
</tr>
<tr>
<td>acCmdNewObjectAutoForm</td>
<td>193</td>
</tr>
<tr>
<td>acCmdNewObjectAutoReport</td>
<td>194</td>
</tr>
<tr>
<td>acCmdNewObjectClassModule</td>
<td>140</td>
</tr>
<tr>
<td>acCmdNewObjectDataAccessPage</td>
<td>346</td>
</tr>
<tr>
<td>acCmdNewObjectDiagram</td>
<td>352</td>
</tr>
<tr>
<td>acCmdNewObjectForm</td>
<td>136</td>
</tr>
<tr>
<td>acCmdNewObjectFunction</td>
<td>394</td>
</tr>
<tr>
<td>acCmdNewObjectMacro</td>
<td>138</td>
</tr>
<tr>
<td>acCmdNewObjectModule</td>
<td>139</td>
</tr>
<tr>
<td>acCmdNewObjectQuery</td>
<td>135</td>
</tr>
<tr>
<td>acCmdNewObjectReport</td>
<td>137</td>
</tr>
<tr>
<td>acCmdNewObjectStoreProcedure</td>
<td>351</td>
</tr>
<tr>
<td>acCmdNewObjectTable</td>
<td>134</td>
</tr>
<tr>
<td>acCmdNewObjectView</td>
<td>350</td>
</tr>
<tr>
<td>acCmdObjBrwFindWholeWordOnly</td>
<td>314</td>
</tr>
<tr>
<td>acCmdObjBrwGroupMembers</td>
<td>318</td>
</tr>
<tr>
<td>acCmdObjBrwHelp</td>
<td>316</td>
</tr>
<tr>
<td>acCmdObjBrwShowHiddenMembers</td>
<td>315</td>
</tr>
<tr>
<td>acCmdObjBrwViewDefinition</td>
<td>317</td>
</tr>
<tr>
<td>acCmdObjectBrowser</td>
<td>200</td>
</tr>
<tr>
<td>acCmdOfficeClipboard</td>
<td>488</td>
</tr>
<tr>
<td>acCmdOLEDDELinks</td>
<td>27</td>
</tr>
<tr>
<td>acCmdOLEObjectConvert</td>
<td>167</td>
</tr>
<tr>
<td>acCmdOLEObjectDefaultVerb</td>
<td>57</td>
</tr>
<tr>
<td>acCmdOpenDatabase</td>
<td>25</td>
</tr>
<tr>
<td>acCmdOpenHyperlink</td>
<td>326</td>
</tr>
<tr>
<td>acCmdOpenNewHyperlink</td>
<td>327</td>
</tr>
<tr>
<td>acCmdOpenSearchPage</td>
<td>253</td>
</tr>
<tr>
<td>acCmdOpenStartPage</td>
<td>252</td>
</tr>
<tr>
<td>acCmdOpenTable</td>
<td>221</td>
</tr>
</tbody>
</table>
acCmdOpenURL 251
acCmdOptions 49
acCmdOutdent 206
acCmdOutputToExcel 175
acCmdOutputToRTF 176
acCmdOutputToText 177
acCmdPageHdrFtr 182
acCmdPageNumber 225
acCmdPageProperties 467
acCmdPageSetup 32
acCmdParameterInfo 305
acCmdPartialReplicaWizard 524
acCmdPaste 191
acCmdPasteAppend 38
acCmdPasteAsHyperlink 490
acCmdPasteSpecial 64
acCmdPivotAutoAverage 416
acCmdPivotAutoCount 413
acCmdPivotAutoFilter 398
acCmdPivotAutoMax 415
acCmdPivotAutoMin 414
acCmdPivotAutoStdDev 417
acCmdPivotAutoStdDevP 419
acCmdPivotAutoSum 412
acCmdPivotAutoVar 418
acCmdPivotAutoVarP 420
acCmdPivotChartByRowByColumn 456
acCmdPivotChartDrillInto 457
acCmdPivotChartDrillOut 532
acCmdPivotChartMultiplePlots 458
acCmdPivotChartMultiplePlotsUnifiedScale 459
acCmdPivotChartShowLegend 455
acCmdPivotChartSortAscByTotal 534
acCmdPivotChartSortDescByTotal 535
acCmdPivotChartType 453
acCmdPivotChartUndo 460
acCmdPivotChartView 397
acCmdPivotCollapse 400
acCmdPivotDelete 454
acCmdPivotDropAreas 452
acCmdPivotExpand 401
acCmdPivotRefresh 404
acCmdPivotShowAll 461
acCmdPivotShowBottom1 432
acCmdPivotShowBottom10 435
acCmdPivotShowBottom10Percent 440
acCmdPivotShowBottom1Percent 437
acCmdPivotShowBottom2 433
acCmdPivotShowBottom25 436
acCmdPivotShowBottom25Percent 441
acCmdPivotShowBottom2Percent 438
acCmdPivotShowBottom5 434
acCmdPivotShowBottom5Percent 439
acCmdPivotShowBottomOther 442
acCmdPivotShowTop1 421
acCmdPivotShowTop10 424
acCmdPivotShowTop10Percent 429
acCmdPivotShowTop1Percent 426
acCmdPivotShowTop2 422
acCmdPivotShowTop25 425
acCmdPivotShowTop25Percent 430
acCmdPivotShowTop2Percent 427
acCmdPivotShowTop5 423
acCmdPivotShowTop5Percent 428
acCmdPivotShowTopOther 431
acCmdPivotTableClearCustomOrdering 527
acCmdPivotTableCreateCalcField 444
acCmdPivotTableCreateCalcTotal 443
acCmdPivotTableDemote 411
acCmdPivotTableExpandIndicators 451
acCmdPivotTableExportToExcel 405
acCmdPivotTableFilterBySelection 528
acCmdPivotTableGroupItems 530
acCmdPivotTableHideDetails 402
acCmdPivotTableMoveToColumnArea 407
acCmdPivotTableMoveToDetailArea 409
acCmdPivotTableMoveToFilterArea 408
acCmdPivotTableMoveToRowArea 406
acCmdPivotTablePercentColumnTotal 447
acCmdPivotTablePercentGrandTotal 450
acCmdPivotTablePercentParentColumnItem 449
acCmdPivotTablePercentParentRowItem 448
acCmdPivotTablePercentRowTotal 446
acCmdPivotTablePromote 410
acCmdPivotTableRemove 529
acCmdPivotTableShowAsNormal 445
acCmdPivotTableShowDetails 403
acCmdPivotTableSubtotal 399
acCmdPivotTableUngroupItems 531
acCmdPivotTableView 396
acCmdPreviewEightPages 249
acCmdPreviewFourPages 248
acCmdPreviewOnePage 246
acCmdPreviewTwelvePages 250
acCmdPreviewTwoPages 247
acCmdPrimaryKey 107
acCmdPrint 340
acCmdPrintPreview 54
acCmdPrintRelationships 483
acCmdProcedureDefinition   122
acCmdPromote               386
acCmdProperties            287
acCmdPublish               320
acCmdPublishDefaults       324
acCmdQueryaddToOutput      362
acCmdQueryGroupBy          361
acCmdQueryParameters       76
acCmdQueryTotals           73
acCmdQueryTypeAppend       91
acCmdQueryTypeCrosstab     74
acCmdQueryTypeDelete       92
acCmdQueryTypeMakeTable    94
acCmdQueryTypeSelect       89
acCmdQueryTypeSQLDataDefinition 168
acCmdQueryTypeSQLPassThrough 169
acCmdQueryTypeSQLUnion     180
acCmdQueryTypeUpdate       90
acCmdQuickInfo             304
acCmdQuickPrint            278
acCmdQuickWatch            203
acCmdRecordsGoToFirst      67
acCmdRecordsGoToLast       68
acCmdRecordsGoToNew        28
acCmdRecordsGoToNext       65
acCmdRecordsGoToPrevious   66
acCmdRecoverDesignMaster   265
acCmdRedo                  199
acCmdReferences            260
acCmdRefresh               18
acCmdRefreshPage           297
acCmdRegisterActiveXControls 254
acCmdRelationships         133
acCmdRemove 366
acCmdRemoveFilterSort 144
acCmdRemoveTable 84
acCmdRename 143
acCmdRenameColumn 274
acCmdRenameGroup 492
acCmdRepairDatabase 6
acCmdReplace 29
acCmdReportHdrFtr 37
acCmdReset 124
acCmdResolveConflicts 266
acCmdRestore 514
acCmdRowHeight 116
acCmdRun 181
acCmdRunMacro 31
acCmdRunOpenMacro 338
acCmdSave 20
acCmdSaveAllModules 280
acCmdSaveAllRecords 392
acCmdSaveAs 21
acCmdSaveAsASP 323
acCmdSaveAsDataAccessPage 389
acCmdSaveAsHTML 321
acCmdSaveAsIDC 322
acCmdSaveAsQuery 96
acCmdSaveAsReport 142
acCmdSaveLayout 145
acCmdSaveModuleAsText 119
acCmdSaveRecord 97
acCmdSelectAll 333
acCmdSelectAllRecords 109
acCmdSelectDataAccessPage 347
acCmdSelectForm 40
acCmdSelectRecord 50
acCmdSelectReport 319
acCmdSend 173
acCmdSendToBack 53
acCmdServerFilterByForm 507
acCmdServerProperties 496
acCmdSetControlDefaults 56
acCmdSetDatabasePassword 275
acCmdSetNextStatement 129
acCmdShowAllRelationships 149
acCmdShowDirectRelationships 148
acCmdShowEnvelope 533
acCmdShowMembers 302
acCmdShowNextStatement 130
acCmdShowOnlyWebToolbar 300
acCmdShowTable 185
acCmdSingleStep 88
acCmdSizeToFit 59
acCmdSizeToFitForm 69
acCmdSizeToGrid 48
acCmdSizeToNarrowest 155
acCmdSizeToWidest 156
acCmdSnapToGrid 62
acCmdSortAscending 163
acCmdSortDescending 164
acCmdSortingAndGrouping 51
acCmdSpeech 511
acCmdSpelling 269
acCmdSQLView 184
acCmdStartupProperties 224
acCmdStepInto 342
acCmdStepOut 311
acCmdStepOver 128
| acCmdStepToCursor           | 204 |
| acCmdStopLoadingPage       | 296 |
| acCmdSubdatasheetCollapseAll| 505 |
| acCmdSubdatasheetExpandAll  | 504 |
| acCmdSubdatasheetRemove    | 506 |
| acCmdSubformDatasheet      | 108 |
| acCmdSubformDatasheetView  | 463 |
| acCmdSubformFormView       | 462 |
| acCmdSubformInNewWindow    | 495 |
| acCmdSubformPivotChartView | 465 |
| acCmdSubformPivotTableView | 464 |
| acCmdSwitchboardManager    | 521 |
| acCmdSynchronizeNow        | 264 |
| acCmdTabControlPageOrder   | 330 |
| acCmdTableAddTable         | 498 |
| acCmdTableCustomView       | 497 |
| acCmdTableNames            | 75  |
| acCmdTabOrder              | 41  |
| acCmdTestValidationRules   | 196 |
| acCmdTileHorizontally      | 286 |
| acCmdTileVertically        | 23  |
| acCmdToggleBreakpoint      | 131 |
| acCmdToggleFilter          | 220 |
| acCmdToolBarControlProperties| 301 |
| acCmdToolbarsCustomize     | 165 |
| acCmdTransferSQLDatabase   | 515 |
| acCmdTransparentBackground | 288 |
| acCmdTransparentBorder     | 289 |
| acCmdUndo                  | 292 |
| acCmdUndoAllRecords        | 393 |
| acCmdUnfreezeAllColumns    | 106 |
| acCmdUngroupControls       | 485 |
| acCmdUnhideColumns         | 80  |
acCmdUpsizingWizard          522
acCmdUserAndGroupAccounts    104
acCmdUserAndGroupPermissions 103
acCmdUserLevelSecurityWizard 276
acCmdVerticalSpacingDecrease 161
acCmdVerticalSpacingIncrease 162
acCmdVerticalSpacingMakeEqual 160
acCmdViewCode                 170
acCmdViewDataAccessPages     349
acCmdViewDetails              210
acCmdViewDiagrams             354
acCmdViewFieldList            353
acCmdViewForms                112
acCmdViewFunctions            395
acCmdViewGrid                 63
acCmdViewLargeIcons           209
acCmdViewList                 212
acCmdViewMacros               114
acCmdViewModules              115
acCmdViewObjectDependencies   536
acCmdViewQueries              111
acCmdViewReports              113
acCmdViewRuler                61
acCmdViewShowPaneDiagram      358
acCmdViewShowPaneGrid         359
acCmdViewShowPaneSQL          357
acCmdViewSmallIcons           211
acCmdViewStoredProcedures     355
acCmdViewTableColumnNames     363
acCmdViewTableColumnProperties 368
acCmdViewTableKeys            369
acCmdViewTableNameOnly        364
acCmdViewTables               110
acCmdViewTableUserView 370
acCmdViewToolbox 85
acCmdViewVerifySQL 360
acCmdViewViews 356
acCmdVisualBasicEditor 525
acCmdWebPagePreview 466
acCmdWebPageProperties 486
acCmdWebTheme 473
acCmdWindowArrangeIcons 24
acCmdWindowCascade 22
acCmdWindowHide 2
acCmdWindowSplit 121
acCmdWindowUnhide 1
acCmdWordMailMerge 195
acCmdWorkgroupAdministrator 391
acCmdZoom10 244
acCmdZoom100 240
acCmdZoom1000 482
acCmdZoom150 239
acCmdZoom200 238
acCmdZoom25 243
acCmdZoom50 242
acCmdZoom500 481
acCmdZoom75 241
acCmdZoomBox 179
acCmdZoomSelection 371

**AcControlType**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>acBoundObjectFrame</td>
<td>108</td>
</tr>
<tr>
<td>acCheckBox</td>
<td>106</td>
</tr>
<tr>
<td>acComboBox</td>
<td>111</td>
</tr>
</tbody>
</table>
acCommandButton 104
acCustomControl 119
acImage 103
acLabel 100
acLine 102
acListBox 110
acObjectFrame 114
acOptionButton 105
acOptionGroup 107
acPage 124
acPageBreak 118
acRectangle 101
acSubform 112
acTabCtl 123
acTextBox 109
acToggleButton 122

**AcCurrentView**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>acCurViewDatasheet</td>
<td>2</td>
</tr>
<tr>
<td>acCurViewDesign</td>
<td>0</td>
</tr>
<tr>
<td>acCurViewFormBrowse</td>
<td>1</td>
</tr>
<tr>
<td>acCurViewPivotChart</td>
<td>4</td>
</tr>
<tr>
<td>acCurViewPivotTable</td>
<td>3</td>
</tr>
<tr>
<td>acCurViewPreview</td>
<td>5</td>
</tr>
</tbody>
</table>

**AcDataAccessPageView**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>acDataAccessPageBrowse</td>
<td>0</td>
</tr>
<tr>
<td>acDataAccessPageDesign</td>
<td>1</td>
</tr>
</tbody>
</table>

**AcDataObjectType**
<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>acActiveDataObject</td>
<td>-1</td>
</tr>
<tr>
<td>acDataForm</td>
<td>2</td>
</tr>
<tr>
<td>acDataFunction</td>
<td>10</td>
</tr>
<tr>
<td>acDataQuery</td>
<td>1</td>
</tr>
<tr>
<td>acDataServerView</td>
<td>7</td>
</tr>
<tr>
<td>acDataStoredProcedure</td>
<td>9</td>
</tr>
<tr>
<td>acDataTable</td>
<td>0</td>
</tr>
</tbody>
</table>

**AcDataTransferType**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>acExport</td>
<td>1</td>
</tr>
<tr>
<td>acImport</td>
<td>0</td>
</tr>
<tr>
<td>acLink</td>
<td>2</td>
</tr>
</tbody>
</table>

**AcDefView**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>acDefViewContinuous</td>
<td>1</td>
</tr>
<tr>
<td>acDefViewDatasheet</td>
<td>2</td>
</tr>
<tr>
<td>acDefViewPivotChart</td>
<td>4</td>
</tr>
<tr>
<td>acDefViewPivotTable</td>
<td>3</td>
</tr>
<tr>
<td>acDefViewSingle</td>
<td>0</td>
</tr>
</tbody>
</table>

**AcExportXMLEncoding**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>acEUCJ</td>
<td>4</td>
</tr>
<tr>
<td>acUCS2</td>
<td>2</td>
</tr>
<tr>
<td>acUCS4</td>
<td>3</td>
</tr>
<tr>
<td>acUTF16</td>
<td>1</td>
</tr>
<tr>
<td>acUTF8</td>
<td>0</td>
</tr>
</tbody>
</table>
### AcExportXMLObjectType

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>acExportDataAccessPage</td>
<td>6</td>
</tr>
<tr>
<td>acExportForm</td>
<td>2</td>
</tr>
<tr>
<td>acExportFunction</td>
<td>10</td>
</tr>
<tr>
<td>acExportQuery</td>
<td>1</td>
</tr>
<tr>
<td>acExportReport</td>
<td>3</td>
</tr>
<tr>
<td>acExportServerView</td>
<td>7</td>
</tr>
<tr>
<td>acExport.StoredProcedure</td>
<td>9</td>
</tr>
<tr>
<td>acExportTable</td>
<td>0</td>
</tr>
</tbody>
</table>

### AcExportXMLOtherFlags

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>acEmbedSchema</td>
<td>1</td>
</tr>
<tr>
<td>acExcludePrimaryKeyAndIndexes</td>
<td>2</td>
</tr>
<tr>
<td>acLiveReportSource</td>
<td>8</td>
</tr>
<tr>
<td>acPersistReportML</td>
<td>16</td>
</tr>
<tr>
<td>acRunFromServer</td>
<td>4</td>
</tr>
</tbody>
</table>

### AcExportXMLSchemaFormat

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>acSchemaNone</td>
<td>0</td>
</tr>
<tr>
<td>acSchemaXSD</td>
<td>1</td>
</tr>
</tbody>
</table>

### AcFileFormat

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>acFileFormatAccess2</td>
<td>2</td>
</tr>
<tr>
<td>acFileFormatAccess2000</td>
<td>9</td>
</tr>
<tr>
<td>acFileFormatAccess2002</td>
<td>10</td>
</tr>
<tr>
<td>acFileFormatAccess95</td>
<td>7</td>
</tr>
<tr>
<td>AcFileFormatAccess97</td>
<td>8</td>
</tr>
<tr>
<td>---------------------</td>
<td>---</td>
</tr>
<tr>
<td><strong>AcFilterType</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Constant</strong></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td>acFilterNormal</td>
<td>0</td>
</tr>
<tr>
<td>acServerFilter</td>
<td>1</td>
</tr>
<tr>
<td><strong>AcFindField</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Constant</strong></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td>acAll</td>
<td>0</td>
</tr>
<tr>
<td>acCurrent</td>
<td>-1</td>
</tr>
<tr>
<td><strong>AcFindMatch</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Constant</strong></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td>acAnywhere</td>
<td>0</td>
</tr>
<tr>
<td>acEntire</td>
<td>1</td>
</tr>
<tr>
<td>acStart</td>
<td>2</td>
</tr>
<tr>
<td><strong>AcFormatConditionOperator</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Constant</strong></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td>acBetween</td>
<td>0</td>
</tr>
<tr>
<td>acEqual</td>
<td>2</td>
</tr>
<tr>
<td>acGreaterThan</td>
<td>4</td>
</tr>
<tr>
<td>acGreaterThanOrEqual</td>
<td>6</td>
</tr>
<tr>
<td>acLessThan</td>
<td>5</td>
</tr>
<tr>
<td>acLessThanOrEqual</td>
<td>7</td>
</tr>
<tr>
<td>acNotBetween</td>
<td>1</td>
</tr>
<tr>
<td>acNotEqual</td>
<td>3</td>
</tr>
<tr>
<td><strong>AcFormatConditionType</strong></td>
<td></td>
</tr>
</tbody>
</table>
### AcExpression

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>acExpression</td>
<td>1</td>
</tr>
<tr>
<td>acFieldHasFocus</td>
<td>2</td>
</tr>
<tr>
<td>acFieldValue</td>
<td>0</td>
</tr>
</tbody>
</table>

### AcFormOpenDataMode

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>acFormAdd</td>
<td>0</td>
</tr>
<tr>
<td>acFormEdit</td>
<td>1</td>
</tr>
<tr>
<td>acFormPropertySettings</td>
<td>-1</td>
</tr>
<tr>
<td>acFormReadOnly</td>
<td>2</td>
</tr>
</tbody>
</table>

### AcFormView

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>acDesign</td>
<td>1</td>
</tr>
<tr>
<td>acFormDS</td>
<td>3</td>
</tr>
<tr>
<td>acFormPivotChart</td>
<td>5</td>
</tr>
<tr>
<td>acFormPivotTable</td>
<td>4</td>
</tr>
<tr>
<td>acNormal</td>
<td>0</td>
</tr>
<tr>
<td>acPreview</td>
<td>2</td>
</tr>
</tbody>
</table>

### AcHyperlinkPart

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>acAddress</td>
<td>2</td>
</tr>
<tr>
<td>acDisplayedValue</td>
<td>0</td>
</tr>
<tr>
<td>acDisplayText</td>
<td>1</td>
</tr>
<tr>
<td>acFullAddress</td>
<td>5</td>
</tr>
<tr>
<td>acScreenTip</td>
<td>4</td>
</tr>
<tr>
<td>acSubAddress</td>
<td>3</td>
</tr>
</tbody>
</table>

### AcImeMode

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>acAddress</td>
<td>2</td>
</tr>
<tr>
<td>acDisplayedValue</td>
<td>0</td>
</tr>
<tr>
<td>acDisplayText</td>
<td>1</td>
</tr>
<tr>
<td>acFullAddress</td>
<td>5</td>
</tr>
<tr>
<td>acScreenTip</td>
<td>4</td>
</tr>
<tr>
<td>acSubAddress</td>
<td>3</td>
</tr>
</tbody>
</table>
### Constant

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>acImeModeAlpha</td>
<td>8</td>
</tr>
<tr>
<td>acImeModeAlphaFull</td>
<td>7</td>
</tr>
<tr>
<td>acImeModeDisable</td>
<td>3</td>
</tr>
<tr>
<td>acImeModeHangul</td>
<td>10</td>
</tr>
<tr>
<td>acImeModeHangulFull</td>
<td>9</td>
</tr>
<tr>
<td>acImeModeHiragana</td>
<td>4</td>
</tr>
<tr>
<td>acImeModeKatakana</td>
<td>5</td>
</tr>
<tr>
<td>acImeModeKatakanaHalf</td>
<td>6</td>
</tr>
<tr>
<td>acImeModeNoControl</td>
<td>0</td>
</tr>
<tr>
<td>acImeModeOff</td>
<td>2</td>
</tr>
<tr>
<td>acImeModeOn</td>
<td>1</td>
</tr>
</tbody>
</table>

### AcImeSentenceMode

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>acImeSentenceModeConversation</td>
<td>2</td>
</tr>
<tr>
<td>acImeSentenceModeNone</td>
<td>3</td>
</tr>
<tr>
<td>acImeSentenceModePhrasePredict</td>
<td>0</td>
</tr>
<tr>
<td>acImeSentenceModePluralClause</td>
<td>1</td>
</tr>
</tbody>
</table>

### AcImportXMLObjectType

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>acImportQuery</td>
<td>1</td>
</tr>
<tr>
<td>acImportServerView</td>
<td>7</td>
</tr>
<tr>
<td>acImportTable</td>
<td>0</td>
</tr>
</tbody>
</table>

### AcImportXMLOption

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>acAppendData</td>
<td>2</td>
</tr>
<tr>
<td>acStructureAndData</td>
<td>1</td>
</tr>
<tr>
<td>acStructureOnly</td>
<td>0</td>
</tr>
</tbody>
</table>
### AcModuleType

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>acClassModule</td>
<td>1</td>
</tr>
<tr>
<td>acStandardModule</td>
<td>0</td>
</tr>
</tbody>
</table>

### AcObjectType

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>acDataAccessPage</td>
<td>6</td>
</tr>
<tr>
<td>acDefault</td>
<td>-1</td>
</tr>
<tr>
<td>acDiagram</td>
<td>8</td>
</tr>
<tr>
<td>acForm</td>
<td>2</td>
</tr>
<tr>
<td>acFunction</td>
<td>10</td>
</tr>
<tr>
<td>acMacro</td>
<td>4</td>
</tr>
<tr>
<td>acModule</td>
<td>5</td>
</tr>
<tr>
<td>acQuery</td>
<td>1</td>
</tr>
<tr>
<td>acReport</td>
<td>3</td>
</tr>
<tr>
<td>acServerView</td>
<td>7</td>
</tr>
<tr>
<td>acStoredProcedure</td>
<td>9</td>
</tr>
<tr>
<td>acTable</td>
<td>0</td>
</tr>
</tbody>
</table>

### AcOpenDataMode

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>acAdd</td>
<td>0</td>
</tr>
<tr>
<td>acEdit</td>
<td>1</td>
</tr>
<tr>
<td>acReadOnly</td>
<td>2</td>
</tr>
</tbody>
</table>

### AcOutputObjectType

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>acOutputDataAccessPage</td>
<td>6</td>
</tr>
<tr>
<td>acOutputForm</td>
<td>2</td>
</tr>
</tbody>
</table>
acOutputFunction 10
acOutputModule 5
acOutputQuery 1
acOutputReport 3
acOutputServerView 7
acOutputStoredProcedure 9
acOutputTable 0

**AcPrintColor**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>acPRCMColor</td>
<td>2</td>
</tr>
<tr>
<td>acPRCMMonochrome</td>
<td>1</td>
</tr>
</tbody>
</table>

**AcPrintDuplex**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>acPRDPHorizontal</td>
<td>2</td>
</tr>
<tr>
<td>acPRDPSimplex</td>
<td>1</td>
</tr>
<tr>
<td>acPRDPVertical</td>
<td>3</td>
</tr>
</tbody>
</table>

**AcPrintItemLayout**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>acPRHorizontalColumnLayout</td>
<td>1953</td>
</tr>
<tr>
<td>acPRVerticalColumnLayout</td>
<td>1954</td>
</tr>
</tbody>
</table>

**AcPrintObjQuality**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>acPRPQDraft</td>
<td>-1</td>
</tr>
<tr>
<td>acPRPQHigh</td>
<td>-4</td>
</tr>
<tr>
<td>acPRPQLow</td>
<td>-2</td>
</tr>
<tr>
<td>acPRPQMedium</td>
<td>-3</td>
</tr>
</tbody>
</table>
### AcPrintOrientation

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>acPRORLandscape</td>
<td>2</td>
</tr>
<tr>
<td>acPRORPortrait</td>
<td>1</td>
</tr>
</tbody>
</table>

### AcPrintPaperBin

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>acPRBNAuto</td>
<td>7</td>
</tr>
<tr>
<td>acPRBNCassette</td>
<td>14</td>
</tr>
<tr>
<td>acPRBNEnvelope</td>
<td>5</td>
</tr>
<tr>
<td>acPRBNEnvManual</td>
<td>6</td>
</tr>
<tr>
<td>acPRBNFormSource</td>
<td>15</td>
</tr>
<tr>
<td>acPRBNLargeCapacity</td>
<td>11</td>
</tr>
<tr>
<td>acPRBNLargeFmt</td>
<td>10</td>
</tr>
<tr>
<td>acPRBNLower</td>
<td>2</td>
</tr>
<tr>
<td>acPRBNManual</td>
<td>4</td>
</tr>
<tr>
<td>acPRBNMiddle</td>
<td>3</td>
</tr>
<tr>
<td>acPRBNSmallFmt</td>
<td>9</td>
</tr>
<tr>
<td>acPRBNTractor</td>
<td>8</td>
</tr>
<tr>
<td>acPRBNUpper</td>
<td>1</td>
</tr>
</tbody>
</table>

### AcPrintPaperSize

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>acPRPS10x14</td>
<td>16</td>
</tr>
<tr>
<td>acPRPS11x17</td>
<td>17</td>
</tr>
<tr>
<td>acPRPSA3</td>
<td>8</td>
</tr>
<tr>
<td>acPRPSA4</td>
<td>9</td>
</tr>
<tr>
<td>acPRPSA4Small</td>
<td>10</td>
</tr>
<tr>
<td>acPRPSA5</td>
<td>11</td>
</tr>
<tr>
<td>acPRPSB4</td>
<td>12</td>
</tr>
<tr>
<td>acPRPSB5</td>
<td>13</td>
</tr>
</tbody>
</table>


**AcPRPSUser**  
256

**AcPrintQuality**

**Constant**  **Value**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>acDraft</td>
<td>3</td>
</tr>
<tr>
<td>acHigh</td>
<td>0</td>
</tr>
<tr>
<td>acLow</td>
<td>2</td>
</tr>
<tr>
<td>acMedium</td>
<td>1</td>
</tr>
</tbody>
</table>

**AcPrintRange**

**Constant**  **Value**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>acPages</td>
<td>2</td>
</tr>
<tr>
<td>acPrintAll</td>
<td>0</td>
</tr>
<tr>
<td>acSelection</td>
<td>1</td>
</tr>
</tbody>
</table>

**AcProjectType**

**Constant**  **Value**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>acADP</td>
<td>1</td>
</tr>
<tr>
<td>acMDB</td>
<td>2</td>
</tr>
<tr>
<td>acNull</td>
<td>0</td>
</tr>
</tbody>
</table>

**AcQuitOption**

**Constant**  **Value**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>acQuitPrompt</td>
<td>0</td>
</tr>
<tr>
<td>acQuitSaveAll</td>
<td>1</td>
</tr>
<tr>
<td>acQuitSaveNone</td>
<td>2</td>
</tr>
</tbody>
</table>

**AcRecord**

**Constant**  **Value**
### AcFirst
2

### AcGoTo
4

### AcLast
3

### AcNewRec
5

### AcNext
1

### AcPrevious
0

### AcSearchDirection
- **Constant**  
  - acDown: 1  
  - acSearchAll: 2  
  - acUp: 0

### AcSection
- **Constant**  
  - acDetail: 0  
  - acFooter: 2  
  - acGroupLevel1Footer: 6  
  - acGroupLevel1Header: 5  
  - acGroupLevel2Footer: 8  
  - acGroupLevel2Header: 7  
  - acHeader: 1  
  - acPageFooter: 4  
  - acPageHeader: 3

### AcSendObjectType
- **Constant**  
  - acSendDataAccessPage: 6  
  - acSendForm: 2  
  - acSendModule: 5  
  - acSendNoObject: -1
<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>acSendQuery</td>
<td>1</td>
</tr>
<tr>
<td>acSendReport</td>
<td>3</td>
</tr>
<tr>
<td>acSendTable</td>
<td>0</td>
</tr>
</tbody>
</table>

### AcShowToolbar

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>acToolbarNo</td>
<td>2</td>
</tr>
<tr>
<td>acToolbarWhereApprop</td>
<td>1</td>
</tr>
<tr>
<td>acToolbarYes</td>
<td>0</td>
</tr>
</tbody>
</table>

### AcSpreadSheetType

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>acSpreadsheetTypeExcel3</td>
<td>0</td>
</tr>
<tr>
<td>acSpreadsheetTypeExcel4</td>
<td>6</td>
</tr>
<tr>
<td>acSpreadsheetTypeExcel5</td>
<td>5</td>
</tr>
<tr>
<td>acSpreadsheetTypeExcel7</td>
<td>5</td>
</tr>
<tr>
<td>acSpreadsheetTypeExcel8</td>
<td>8</td>
</tr>
<tr>
<td>acSpreadsheetTypeExcel9</td>
<td>8</td>
</tr>
<tr>
<td>acSpreadsheetTypeLotusWJ2</td>
<td>4</td>
</tr>
<tr>
<td>acSpreadsheetTypeLotusWK1</td>
<td>2</td>
</tr>
<tr>
<td>acSpreadsheetTypeLotusWK3</td>
<td>3</td>
</tr>
<tr>
<td>acSpreadsheetTypeLotusWK4</td>
<td>7</td>
</tr>
</tbody>
</table>

### AcSysCmdAction

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>acSysCmdAccessDir</td>
<td>9</td>
</tr>
<tr>
<td>acSysCmdAccessVer</td>
<td>7</td>
</tr>
<tr>
<td>acSysCmdClearHelpTopic</td>
<td>11</td>
</tr>
<tr>
<td>acSysCmdClearStatus</td>
<td>5</td>
</tr>
<tr>
<td>acSysCmdGetObjectState</td>
<td>10</td>
</tr>
<tr>
<td>acSysCmdGetWorkgroupFile</td>
<td>13</td>
</tr>
</tbody>
</table>
### AcTextTransferType

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>acExportDelim</td>
<td>2</td>
</tr>
<tr>
<td>acExportFixed</td>
<td>3</td>
</tr>
<tr>
<td>acExportHTML</td>
<td>8</td>
</tr>
<tr>
<td>acExportMerge</td>
<td>4</td>
</tr>
<tr>
<td>acImportDelim</td>
<td>0</td>
</tr>
<tr>
<td>acImportFixed</td>
<td>1</td>
</tr>
<tr>
<td>acImportHTML</td>
<td>7</td>
</tr>
<tr>
<td>acLinkDelim</td>
<td>5</td>
</tr>
<tr>
<td>acLinkFixed</td>
<td>6</td>
</tr>
<tr>
<td>acLinkHTML</td>
<td>9</td>
</tr>
</tbody>
</table>

### AcTransformXMLScriptOption

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>acDisableObjects</td>
<td>2</td>
</tr>
<tr>
<td>acDisableScript</td>
<td>2</td>
</tr>
<tr>
<td>acEnableObjects</td>
<td>0</td>
</tr>
<tr>
<td>acEnableScript</td>
<td>0</td>
</tr>
<tr>
<td>acPromptObjects</td>
<td>1</td>
</tr>
<tr>
<td>acPromptScript</td>
<td>1</td>
</tr>
</tbody>
</table>

### AcView
<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>acViewDesign</td>
<td>1</td>
</tr>
<tr>
<td>acViewNormal</td>
<td>0</td>
</tr>
<tr>
<td>acViewPivotChart</td>
<td>4</td>
</tr>
<tr>
<td>acViewPivotTable</td>
<td>3</td>
</tr>
<tr>
<td>acViewPreview</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AcWindowMode</td>
<td></td>
</tr>
<tr>
<td>acDialog</td>
<td>3</td>
</tr>
<tr>
<td>acHidden</td>
<td>1</td>
</tr>
<tr>
<td>acIcon</td>
<td>2</td>
</tr>
<tr>
<td>acWindowNormal</td>
<td>0</td>
</tr>
</tbody>
</table>
Set Form, Report, and Control Properties

Each form, report, section, and control has property settings that you can change to alter the look or behavior of that particular item. You view and change properties by using the property sheet, macro, or Visual Basic.

To set properties

1. In form Design view or report Design view, select the control, section, form, or report for which you want to set the property. You can select:
   - One or more controls. To select multiple controls, hold down the SHIFT key and click the controls, or drag the mouse pointer over the controls you wish to select. If you select multiple controls, the property sheet will display only those properties that the selected controls have in common.
   - One section. Click the section selector of the section you wish to select.
   - The entire form or report. Click the form selector or report selector in the upper-left corner of the form or report.
2. Display the property sheet by right-clicking the object or section and then clicking Properties on the shortcut menu, or by clicking Properties on the toolbar.
3. Click the property for which you want to set the value, and then do one of the following:
   - In the property box, type the appropriate setting or expression.
   - If the property box contains an arrow, click the arrow and then click a value in the list.
   - If a Build button appears to the right of the property box, click it to display a builder or to display a dialog box giving you a choice of builders. For example, you can use the Code Builder, Macro Builder, or Query Builder to set some properties.

Tips

- Microsoft Access provides a Zoom box for typing and viewing expressions
or other long property settings. To display the **Zoom** box, click a property
box in the property sheet. Then press SHIFT+F2, or right-click and then
click **Zoom** on the shortcut menu.

- You can set the **ControlSource** property for some controls by typing the
  property setting in the control itself.
- You can change the default property settings for a type of control so that
  future controls you create will have the new default settings.
- The property settings of a **bound control** might not match the corresponding
  settings in the field in the underlying table or query to which the control is
  bound. If the settings are different, the form or report settings typically
  override those in the table or query. For more information about how
  properties are inherited, see [How control properties relate to properties in
  their underlying fields](#).
Unbound Object Frame Control

The unbound object frame control displays a picture, chart, or any OLE object not stored in a table. For example, you can use an unbound object frame to display a chart that you created and stored in Microsoft Graph.
Remarks

This control allows you to create or edit the object from within a Microsoft Access form or report by using the application in which the object was originally created.

To display objects that are stored in a Microsoft Access database, use a bound object frame control.

The object in an unbound object frame is the same for every record.

The unbound object frame can display linked or embedded objects.

Tip

You can use the unbound object frame or an image control to display unbound pictures in a form or report. The advantage of using the unbound object frame is that you can edit the object directly from the form or report. The advantage of using the image control is that it's faster to display.
Image Control (Forms and Reports)

The image control can add a picture to a form or report. For example, you could include an image control for a logo on an Invoice report.

Note  This control should not be confused with the Dynamic HTML image control used on a data access page. For information about a image control on a data access page, see Image Control (Data Access Page).
Remarks

You can use the image control or an unbound object frame for unbound pictures. The advantage of using the image control is that it's faster to display. The advantage of using the unbound object frame is that you can edit the object directly from the form or report.
Check Box Control (Data Access Pages)

A check box on a data access page is a bound control that displays a Yes/No value from an underlying record source.

Control: Tool:
☑️ Address Change ☑️
Remarks

When you select or clear a check box that's bound to a Yes/No field, Microsoft Access displays the value in the underlying table according to the field's **Format** property (Yes/No, **True/False**, or On/Off).

You can also use check boxes in an **option group** to display values to choose from.

It's also possible to use an **unbound** check box in a **custom dialog box** to accept user input.
Drop-down List Box Control (Data Access Pages)

The drop-down list box control on a data access page combines some of the features of a text box and a list box. Use a drop-down list box control when you want the option of displaying and selecting a value from a predefined list.

Control: Tool:

Canada ▼
Remarks

In Page view, Microsoft Access doesn't display the list until you click the drop-down list box control's arrow.

If you have Control Wizards on before you select the drop-down list box control tool, you can create a drop-down list box with a wizard. To turn Control Wizards on or off, click the Control Wizards tool in the toolbox.

You cannot enter values that aren't in the list.

The list can be single- or multiple-column, and the columns can appear with or without headings.
Event Properties

Event properties cause a macro or the associated Visual Basic event procedure to run when a particular event occurs. For example, if you enter the name of a macro in a command button's OnClick property, that macro runs when the command button is clicked.
Setting

To run a macro, enter the name of the macro. You can choose an existing macro in the list. If the macro is in a macro group, it will be listed under the macro group name, as `macrogroupName.macroname`.

To run the event procedure associated with the event, select [Event Procedure] in the list.

**Note** Although using an event procedure is the recommended method for running Visual Basic code in response to an event, you can also run a user-defined function when an event occurs. To run a user-defined function, place an equal sign (=) before the function name and parentheses after it, as in `=functionname()`.

You can set event properties in the property sheet for an object, in a macro, or by using Visual Basic. Note that you can't set any event properties while you're formatting or printing a form or report.

**Tip**

You can use builders to help you set an event property. To use them, click the Build button to the right of the property box, or right-click the property box and then click Build on the shortcut menu. In the Choose Builder dialog box, select:

- The Macro Builder to create and specify a macro for this event property. You can also use the Macro Builder to edit a macro already specified by the property.
- The Code Builder to create and specify an event procedure for this event property. You can also use the Code Builder to edit an event procedure already specified by the property.
- In a Microsoft Access database (.mdb), the Expression Builder to choose and specify a user-defined function for this event property.

In Visual Basic, set the property to a string expression.

<table>
<thead>
<tr>
<th>To run this</th>
<th>Use this syntax</th>
<th>Example</th>
</tr>
</thead>
</table>


Macro     "macroname"    Button1.OnClick = "MyMacro"

Event procedure    "[Event Procedure]"    Button1.OnClick = "[Event Procedure]"

User-defined "=functionname(    Button1.OnClick = "=MyFunction()"
function    )"
Example

The following example shows how you can use the value entered in the Country control to determine which of two different macros to run when you click the Print Country Report button.

Private Sub Country_AfterUpdate()
    If Country = "Canada" Then
        [Print Country Report].OnClick = "PrintCanadaReport"
    ElseIf Country = "USA" Then
        [Print Country Report].OnClick = "PrintUSAReport"
    End If
End Sub
Command Button Control (Data Access Pages)

A command button on a data access page can start an action or a set of actions. On a data access page, code written in either JavaScript or Visual Basic Script is attached to a command button using the Microsoft Visual Script Editor.
Remarks

You can display text on a command button by setting its **Value** property.
KeepTogether Property - Groups

You can use the KeepTogether property for a group in a report to keep parts of a group— including the group header, detail section, and group footer— together on the same page. For example, you might want a group header to always be printed on the same page with the first detail section.
Setting

The **KeepTogether** property for a group uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>0</td>
<td>(Default) Prints the group without keeping the group header, detail section, and group footer on the same page.</td>
</tr>
<tr>
<td>Whole Group</td>
<td>1</td>
<td>Prints the group header, detail section, and group footer on the same page.</td>
</tr>
<tr>
<td>With First Detail</td>
<td>2</td>
<td>Prints the group header on a page only if it can also print the first detail record.</td>
</tr>
</tbody>
</table>

You can set the **KeepTogether** property for a group by using the **Sorting And Grouping** box, a **macro**, or **Visual Basic**.

In Visual Basic, you set the **KeepTogether** property for a group in **report Design view** or the **Open** event procedure of a report by using the **GroupLevel** property.
Remarks

To set the KeepTogether property for a group to a value other than No, you must set the GroupHeader or GroupFooter property or both to Yes for the selected field or expression.

A group includes the group header, detail section, and group footer. If you set the KeepTogether property for a group to Whole Group and the group is too large to fit on one page, Microsoft Access will ignore the setting for that group. Similarly, if you set this property to With First Detail and either the group header or detail record is too large to fit on one page, the setting will be ignored.

If the KeepTogether property for a section is set to No and the KeepTogether property for a group is set to Whole Group or With First Detail, the KeepTogether property setting for the section is ignored.
Image Control (Data Access Pages)

The image control can add a picture to a data access page. For example, you could include an image control for a logo on a Sales entry page.

Control: Tool:
Remarks

You can use the image control or an unbound object frame for unbound pictures. The advantage of using the image control is that it's faster to display.
Label Control (Data Access Pages)

Labels on an data access page display descriptive text such as titles, captions, or brief instructions. Labels have certain characteristics:

- Labels don't display values from fields or expressions.
- Labels don't change as you move from record to record.

Control: Tool:
Remarks

A label can be attached to another control. When you create a text box, for example, it has an attached label that displays a caption for that text box.

When you create a label by using the **Label** tool, the label stands on its own it isn't attached to any other control. You use stand-alone labels for information such as the title, or for other descriptive text.
Line Control (Data Access Pages)

The line control displays a horizontal, vertical, or diagonal line on a data access page.

Control: Tool:

![Graph showing data access over quarters](image)
Remarks

You can use the **Height** property to change the line width. You can use **Line/Border Color** to change the color of the border or make it transparent.
List Box Control (Data Access Pages)

The list box control displays a list of values or alternatives. In many cases, it's quicker and easier to select a value from a list than to remember a value to type. A list of choices also helps ensure that the value that's entered in a field is correct.

**Control:**

<table>
<thead>
<tr>
<th>Supplier:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aux joyeux ecclésiastique</td>
</tr>
<tr>
<td>Bigfoot Breweries</td>
</tr>
<tr>
<td>Coopérativa de Queso Ltd.</td>
</tr>
</tbody>
</table>

**Tool:**

The list in a list box consists of rows of data. Rows can have one or more columns, which can appear with or without headings.

---

<table>
<thead>
<tr>
<th>Supplier:</th>
<th>Supplier ID</th>
<th>Company Name</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>18</td>
<td>Aux joyeux ecclésiastique</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>Bigfoot Breweries</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>Coopérativa de Queso Ltd.</td>
</tr>
<tr>
<td></td>
<td>27</td>
<td>Escargots Nouveaux</td>
</tr>
</tbody>
</table>

Two-column list box with headings
Remarks

If a multiple-column list box is **bound**, Microsoft Access stores the values from one of the columns.

You can use an **unbound** list box to store a value that you can use with another control. For example, you could use an unbound list box to limit the values in another list box or in a **custom dialog box**. You could also use an unbound list box to find a record based on the value you select in the list box.

If you don't have room on your form to display a list box, use a **dropdown box** instead of a list box.
Bound Object Frame Control

A bound object frame control displays a picture, chart, or any OLE object stored in a table in a Microsoft Access database. For example, if you store pictures of your employees in a table in Microsoft Access, you can use a bound object frame to display these pictures on a form or report.
Remarks

This control type allows you to create or edit the object from within the form or report by using the OLE server.

A bound object frame is bound to a field in an underlying table.

The field in the underlying table to which the bound object frame is bound must be of the OLE Object data type.

The object in a bound object frame is different for each record. The bound object frame can display linked or embedded objects. If you want to display objects not stored in an underlying table, use an unbound object frame or an image control.
Option Button Control (Data Access Pages)

An option button on a data access page is a stand-alone control used to display a Yes/No value from an underlying record source.

Control: Tool:

Discontinued
Remarks

When you select or clear an option button that's bound to a Yes/No field, Microsoft Access displays the value in the underlying table according to the field's **Format** property (Yes/No, **True/False**, or On/Off).

You can also use option buttons in an **option group** to display values to choose from.

It's also possible to use an unbound option button in a **custom dialog box** to accept user input.
Option Group Control (Data Access Pages)

An option group on a data access page displays a limited set of alternatives. An option group makes selecting a value easy since you can just click the value you want. Only one option in an option group can be selected at a time.

An option group consists of a group frame and a set of check boxes, toggle buttons, or option buttons.

Control: Tool:
Remarks

If an option group is bound to a field, only the group frame itself is bound to the field, not the check boxes, toggle buttons, or option buttons inside the frame. Instead of setting the **ControlSource** property for each control in the option group, you set the **Value** property of each check box, toggle button, or option button to a number that's meaningful for the field to which the group frame is bound. When you select an option in an option group, Microsoft Access sets the value of the field to which the option group is bound to the value of the selected option's **Value** property.

Note  The **Value** property is set to a number because the value of an option group can only be a number, not text. Microsoft Access stores this number in the underlying table. In the preceding example, if you want to display the name of the shipper instead of a number in the Orders table, you can create a separate table called Shippers that stores shipper names, and then make the ShipVia field in the Orders table a Lookup field that looks up data in the Shippers table.

An option group can also be set to an **expression**, or it can be unbound. You can use an unbound option group in a **custom dialog box** to accept user input and then carry out an action based on that input.
Rectangle Control (Data Access Pages)

The rectangle control displays a rectangle on a data access page.

Control: Tool:

[ ]  [ ]
Remarks

You can move a rectangle and the controls in it as a single unit by dragging the mouse pointer diagonally across the entire rectangle to select all of the controls. The entire selection can then be moved to a new position.
Text Box Control (Data Access Pages)

Text boxes on a data access page display data from a record source. This type of text box is called a bound text box because it's bound to data in a field. Text boxes can also be unbound. For example, you can create an unbound text box to display the results of a calculation, or to accept input from a user. Data in an unbound text box isn't saved with the database.

Control: Tool:

Extension: 5467 abl
Guid Property

The GUID property of a Reference object returns a GUID that identifies a type library in the Windows Registry. Read-only String.

expression.Guid

expression Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **GUID** property is available only by using Visual Basic.

Every type library has an associated GUID which is stored in the Registry. When you set a reference to a type library, Microsoft Access uses the type library’s GUID to identify the type library.

You can use the `AddFromGUID` method to create a **Reference** object from a type library's GUID.
Example

The following example prints the value of the **FullPath**, **GUID**, **IsBroken**, **Major**, and **Minor** properties for each **Reference** object in the **References** collection:

```vbnet
Sub ReferenceProperties()
    Dim ref As Reference

    ' Enumerate through References collection.
    For Each ref In References
        ' Check IsBroken property.
        If ref.IsBroken = False Then
            Debug.Print "Name: ", ref.Name
            Debug.Print "FullPath: ", ref.FullPath
        Else
            Debug.Print "GUIDs of broken references:"
            Debug.Print ref.GUID
        EndIf
    Next ref
End Sub
```
Set Options from Visual Basic

You can use the `SetOption` and `GetOption` methods to set and return option values in the `Options` dialog box from code. To view the `Options` dialog box, click `Options` on the `Tools` menu.

The following tables list the names of all options that can be set or returned from code and the tabs on which they can be found in the `Options` dialog box, followed by the corresponding string argument that you must pass to the `SetOption` or `GetOption` method.

**Notes**

If your database may run on a version of Microsoft Access for a language other than the one in which you created it, then you must supply the arguments for the `GetOption` and `SetOption` methods in English.

Some options are available only within a Microsoft Access database (.mdb) or Access project (.adp).
## View Tab

<table>
<thead>
<tr>
<th>Option text</th>
<th>String argument</th>
</tr>
</thead>
<tbody>
<tr>
<td>Show, Status bar</td>
<td>Show Status Bar</td>
</tr>
<tr>
<td>Show, Startup Task Pane</td>
<td>Show Startup Dialog Box</td>
</tr>
<tr>
<td>Show, New object shortcuts</td>
<td>Show New Object Shortcuts</td>
</tr>
<tr>
<td>Show, Hidden objects</td>
<td>Show Hidden Objects</td>
</tr>
<tr>
<td>Show, System objects</td>
<td>Show System Objects</td>
</tr>
<tr>
<td>Show, Windows in Taskbar</td>
<td>ShowWindowsInTaskbar</td>
</tr>
<tr>
<td>Show in Macro Design, Names column</td>
<td>Show Macro Names Column</td>
</tr>
<tr>
<td>Show in Macro Design, Conditions column</td>
<td>Show Conditions Column</td>
</tr>
<tr>
<td>Click options in database window</td>
<td>Database Explorer Click Behavior</td>
</tr>
</tbody>
</table>
## General Tab

<table>
<thead>
<tr>
<th>Option text</th>
<th>String argument</th>
</tr>
</thead>
<tbody>
<tr>
<td>Print margins, Left margin</td>
<td>Left Margin</td>
</tr>
<tr>
<td>Print margins, Right margin</td>
<td>Right Margin</td>
</tr>
<tr>
<td>Print margins, Top margin</td>
<td>Top Margin</td>
</tr>
<tr>
<td>Print margins, Bottom margin</td>
<td>Bottom Margin</td>
</tr>
<tr>
<td>Use four-year digit year formatting, This database</td>
<td>Four-Digit Year Formatting</td>
</tr>
<tr>
<td>Use four-year digit year formatting, All databases</td>
<td>Four-Digit Year Formatting All Databases</td>
</tr>
<tr>
<td>Name AutoCorrect, Track name AutoCorrect info</td>
<td>Track Name AutoCorrect Info</td>
</tr>
<tr>
<td>Name AutoCorrect, Perform name AutoCorrect</td>
<td>Perform Name AutoCorrect</td>
</tr>
<tr>
<td>Name AutoCorrect, Log name AutoCorrect changes</td>
<td>Log Name AutoCorrect Changes</td>
</tr>
<tr>
<td>Recently used file list</td>
<td>Enable MRU File List</td>
</tr>
<tr>
<td>Recently used file list, (number of files)</td>
<td>Size of MRU File List</td>
</tr>
<tr>
<td>Provide feedback with sound</td>
<td>Provide Feedback with Sound</td>
</tr>
<tr>
<td>Compact on Close</td>
<td>Auto Compact</td>
</tr>
<tr>
<td>New database sort order</td>
<td>New Database Sort Order</td>
</tr>
<tr>
<td>Remove personal information from file properties on save</td>
<td>Remove Personal Information</td>
</tr>
<tr>
<td>Default database folder</td>
<td>Default Database Directory</td>
</tr>
</tbody>
</table>
## Edit/Find Tab

<table>
<thead>
<tr>
<th>Option text</th>
<th>String argument</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default find/replace behavior</td>
<td>Default Find/Replace Behavior</td>
</tr>
<tr>
<td>Confirm, Record changes</td>
<td>Confirm Record Changes</td>
</tr>
<tr>
<td>Confirm, Document deletions</td>
<td>Confirm Document Deletions</td>
</tr>
<tr>
<td>Confirm, Action queries</td>
<td>Confirm Action Queries</td>
</tr>
<tr>
<td>Show list of values in, Local indexed fields</td>
<td>Show Values in Indexed</td>
</tr>
<tr>
<td>Show list of values in, Local nonindexed fields</td>
<td>Show Values in Non-Indexed</td>
</tr>
<tr>
<td>Show list of values in, ODBC fields</td>
<td>Show Values in Remote</td>
</tr>
<tr>
<td>Show list of values in, Records in local snapshot</td>
<td>Show Values in Snapshot</td>
</tr>
<tr>
<td>Show list of values in, Records at server</td>
<td>Show Values in Server</td>
</tr>
<tr>
<td>Don't display lists where more than this number of records read</td>
<td>Show Values Limit</td>
</tr>
</tbody>
</table>
## Datasheet Tab

<table>
<thead>
<tr>
<th>Option text</th>
<th>String argument</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default colors, Font</td>
<td>Default Font Color</td>
</tr>
<tr>
<td>Default colors, Background</td>
<td>Default Background Color</td>
</tr>
<tr>
<td>Default colors, Gridlines</td>
<td>Default Gridlines Color</td>
</tr>
<tr>
<td>Default gridlines showing, Horizontal</td>
<td>Default Gridlines Horizontal</td>
</tr>
<tr>
<td>Default gridlines showing, Vertical</td>
<td>Default Gridlines Vertical</td>
</tr>
<tr>
<td>Default column width</td>
<td>Default Column Width</td>
</tr>
<tr>
<td>Default font, Font</td>
<td>Default Font Name</td>
</tr>
<tr>
<td>Default font, Weight</td>
<td>Default Font Weight</td>
</tr>
<tr>
<td>Default font, Size</td>
<td>Default Font Size</td>
</tr>
<tr>
<td>Default font, Underline</td>
<td>Default Font Underline</td>
</tr>
<tr>
<td>Default font, Italic</td>
<td>Default Font Italic</td>
</tr>
<tr>
<td>Default cell effect</td>
<td>Default Cell Effect</td>
</tr>
<tr>
<td>Show animations</td>
<td>Show Animations</td>
</tr>
<tr>
<td>Show Smart Tags on Datasheets</td>
<td>Show Smart Tags on Datasheets</td>
</tr>
</tbody>
</table>
# Keyboard Tab

<table>
<thead>
<tr>
<th>Option text</th>
<th>String argument</th>
</tr>
</thead>
<tbody>
<tr>
<td>Move after enter</td>
<td>Move After Enter</td>
</tr>
<tr>
<td>Behavior entering field</td>
<td>Behavior Entering Field</td>
</tr>
<tr>
<td>Arrow key behavior</td>
<td>Arrow Key Behavior</td>
</tr>
<tr>
<td>Cursor stops at first/last field</td>
<td>Cursor Stops at First/Last Field</td>
</tr>
<tr>
<td>Auto commit</td>
<td>Ime Autocommit</td>
</tr>
<tr>
<td>Datasheet IME control</td>
<td>Datasheet Ime Control</td>
</tr>
</tbody>
</table>
## Tables/Queries Tab

<table>
<thead>
<tr>
<th>Option text</th>
<th>String argument</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table design, Default field sizes - Text</td>
<td>Default Text Field Size</td>
</tr>
<tr>
<td>Table design, Default field sizes - Number</td>
<td>Default Number Field Size</td>
</tr>
<tr>
<td>Table design, Default field type</td>
<td>Default Field Type</td>
</tr>
<tr>
<td>Table design, AutoIndex on Import/Create</td>
<td>AutoIndex on Import/Create</td>
</tr>
<tr>
<td>Query design, Show table names</td>
<td>Show Table Names</td>
</tr>
<tr>
<td>Query design, Output all fields</td>
<td>Output All Fields</td>
</tr>
<tr>
<td>Query design, Enable AutoJoin</td>
<td>Enable AutoJoin</td>
</tr>
<tr>
<td>Query design, Run permissions</td>
<td>Run Permissions</td>
</tr>
<tr>
<td>Query design, SQL Server Compatible Syntax (ANSI 92) - This database</td>
<td>ANSI Query Mode</td>
</tr>
<tr>
<td>Query design, SQL Server Compatible Syntax (ANSI 92) - Default for new databases</td>
<td>ANSI Query Mode Default</td>
</tr>
<tr>
<td>Query design, Query design font, Font</td>
<td>Query Design Font Name</td>
</tr>
<tr>
<td>Query design, Query design font, Size</td>
<td>Query Design Font Size</td>
</tr>
<tr>
<td>Show Property Update Options buttons</td>
<td>Show Property Update Options buttons</td>
</tr>
</tbody>
</table>
# Forms/Reports Tab

<table>
<thead>
<tr>
<th>Option text</th>
<th>String argument</th>
</tr>
</thead>
<tbody>
<tr>
<td>Selection behavior</td>
<td>Selection Behavior</td>
</tr>
<tr>
<td>Form template</td>
<td>Form Template</td>
</tr>
<tr>
<td>Report template</td>
<td>Report Template</td>
</tr>
<tr>
<td>Always use event procedures</td>
<td>Always Use Event Procedures</td>
</tr>
<tr>
<td>Show Smart Tags on Forms</td>
<td>Show Smart Tags on Forms</td>
</tr>
<tr>
<td>Show Windows Themed Controls on Forms</td>
<td>Themed Form Controls</td>
</tr>
</tbody>
</table>
## Advanced Tab

<table>
<thead>
<tr>
<th><strong>Option text</strong></th>
<th><strong>String argument</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>DDE operations, Ignore DDE requests</td>
<td>Ignore DDE Requests</td>
</tr>
<tr>
<td>DDE operations, Enable DDE refresh</td>
<td>Enable DDE Refresh</td>
</tr>
<tr>
<td>Default File Format</td>
<td>Default File Format</td>
</tr>
<tr>
<td>Default open mode</td>
<td>Default Open Mode for Databases</td>
</tr>
<tr>
<td>Command-line arguments</td>
<td>Command-Line Arguments</td>
</tr>
<tr>
<td>OLE/DDE timeout (sec)</td>
<td>OLE/DDE Timeout (sec)</td>
</tr>
<tr>
<td>Default record locking</td>
<td>Default Record Locking</td>
</tr>
<tr>
<td>Refresh interval (sec)</td>
<td>Refresh Interval (sec)</td>
</tr>
<tr>
<td>Number of update retries</td>
<td>Number of Update Retries</td>
</tr>
<tr>
<td>ODBC refresh interval (sec)</td>
<td>ODBC Refresh Interval (sec)</td>
</tr>
<tr>
<td>Update retry interval (msec)</td>
<td>Update Retry Interval (msec)</td>
</tr>
<tr>
<td>Open databases using record-level locking</td>
<td>Use Row Level Locking</td>
</tr>
</tbody>
</table>
## Pages Tab

<table>
<thead>
<tr>
<th>Option text</th>
<th>String argument</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default Designer Properties, Section Indent</td>
<td>Section Indent</td>
</tr>
<tr>
<td>Default Designer Properties, Alternate Row Color</td>
<td>Alternate Row Color</td>
</tr>
<tr>
<td>Default Designer Properties, Caption Section Style</td>
<td>Caption Section Style</td>
</tr>
<tr>
<td>Default Designer Properties, Footer Section Style</td>
<td>Footer Section Style</td>
</tr>
<tr>
<td>Default Database/Project Properties, Use Default Page Folder</td>
<td>Use Default Page Folder</td>
</tr>
<tr>
<td>Default Database/Project Properties, Default Page Folder</td>
<td>Default Page Folder</td>
</tr>
<tr>
<td>Default Database/Project Properties, Use Default Connection File</td>
<td>Use Default Connection File</td>
</tr>
<tr>
<td>Default Database/Project Properties, Default Connection File</td>
<td>Default Connection File</td>
</tr>
</tbody>
</table>
## Spelling Tab

<table>
<thead>
<tr>
<th><strong>Option text</strong></th>
<th><strong>String argument</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Dictionary Language</td>
<td>Spelling dictionary language</td>
</tr>
<tr>
<td>Add words to</td>
<td>Spelling add words to</td>
</tr>
<tr>
<td>Suggest from main dictionary only</td>
<td>Spelling suggest from main dictionary only</td>
</tr>
<tr>
<td>Ignore words in UPPERCASE</td>
<td>Spelling ignore words in UPPERCASE</td>
</tr>
<tr>
<td>Ignore words with numbers</td>
<td>Spelling ignore words with number</td>
</tr>
<tr>
<td>Ignore Internet and file addresses</td>
<td>Spelling ignore Internet and file addresses</td>
</tr>
<tr>
<td>Language-specific, German: Use post-reform rules</td>
<td>Spelling use German post-reform rules</td>
</tr>
<tr>
<td>Language-specific, Korean: Combine aux verb/adj.</td>
<td>Spelling combine aux verb/adj</td>
</tr>
<tr>
<td>Language-specific, Korean: Search misused word list</td>
<td>Spelling use auto-change list</td>
</tr>
<tr>
<td>Language-specific, Korean: Process compound nouns</td>
<td>Spelling process compound nouns</td>
</tr>
<tr>
<td>Language-specific, Hebrew modes</td>
<td>Spelling Hebrew modes</td>
</tr>
<tr>
<td>Language-specific, Arabic modes</td>
<td>Spelling Arabic modes</td>
</tr>
</tbody>
</table>
### International Tab

<table>
<thead>
<tr>
<th>Option text</th>
<th>String argument</th>
</tr>
</thead>
<tbody>
<tr>
<td>Right-to-Left, Default direction</td>
<td>Default direction</td>
</tr>
<tr>
<td>Right-to-Left, General alignment</td>
<td>General alignment</td>
</tr>
<tr>
<td>Right-to-Left, Cursor movement</td>
<td>Cursor movement</td>
</tr>
<tr>
<td>Use Hijri Calendar</td>
<td>Use Hijri Calendar</td>
</tr>
</tbody>
</table>
## Error Checking Tab

<table>
<thead>
<tr>
<th>Option text</th>
<th>String argument</th>
</tr>
</thead>
<tbody>
<tr>
<td>Settings, Enable error checking</td>
<td>Enable Error Checking</td>
</tr>
<tr>
<td>Settings, Error indicator color</td>
<td>Error Checking Indicator Color</td>
</tr>
<tr>
<td>Form/Report Design Rules, Unassociated label and control</td>
<td>Unassociated Label and Control Error Checking</td>
</tr>
<tr>
<td>Form/Report Design Rules, New unassociated labels</td>
<td>New Unassociated Label Error Checking</td>
</tr>
<tr>
<td>Form/Report Design Rules, Keyboard shortcut errors</td>
<td>Keyboard Shortcut Errors Error Checking</td>
</tr>
<tr>
<td>Form/Report Design Rules, Invalid control properties</td>
<td>Invalid Control Properties Error Checking</td>
</tr>
<tr>
<td>Form/Report Design Rules, Common report errors</td>
<td>Common Report Errors Error Checking</td>
</tr>
</tbody>
</table>

**Note** If you are developing a database application, add-in, library database, or referenced database, make sure that the Error Trapping option is set to 2 (Break On Unhandled Errors) when you have finished debugging your code.

The value that you pass to the **SetOption** method as the *setting* argument depends on which type of option you are setting. The following table establishes some guidelines for setting options.

<table>
<thead>
<tr>
<th>If the option is</th>
<th>Then the <em>setting</em> argument is</th>
</tr>
</thead>
<tbody>
<tr>
<td>A text box</td>
<td>A string</td>
</tr>
<tr>
<td>A check box</td>
<td>A Boolean value— <strong>True</strong> (–1) or <strong>False</strong> (0)</td>
</tr>
<tr>
<td>An option button in an option group, or an option in a combo box or a list box</td>
<td>An integer corresponding to the option's position in the option group or list (starting with zero [0])</td>
</tr>
</tbody>
</table>
**Print Event**

The Print event occurs after data in a report section is formatted for printing, but before the section is printed.

**Private Sub object_Print(Cancel As Integer, PrintCount As Integer)**

*Object*  The name of a Report section.

*Cancel*  The setting determines if the printing of the section occurs. Setting the Cancel argument to True (–1) cancels printing of the section.

*PrintCount*  An Integer value that specifies whether the Print event has occurred more than once for a record. For example, if part of a record is printed on one page and the rest is printed on the next page, the Print event occurs twice, and Microsoft Access sets the PrintCount argument to 2.
Remarks

To run a macro or event procedure when this event occurs, set the OnPrint property to the name of the macro or to [Event Procedure].

For a report detail section, the Print event occurs for each record in the section just before Microsoft Access prints the data in the record. A Print event procedure or macro has access to the data in the current record.

For report group headers, the Print event occurs for each new group, and a Print macro or event procedure has access to the data in the group header and the data in the first record in the detail section. For report group footers, the Print event occurs for each new group, and a Print macro or event procedure has access to the data in the group footer and the data in the last record in the detail section.

You can use the Print event to run a macro or event procedure only after Microsoft Access has prepared data for printing on a page. For example, you can calculate running page totals that are printed in the page header or footer.

For changes that affect page layout, such as displaying or hiding controls, use the Format event.

The Print event occurs only for sections that are actually printed. If you need access to data from sections that aren't printed (for example, you are keeping a running sum, but are only printing certain pages), use the Format event instead.

The Page event occurs after all the Format events for the report, and after all the Print events for a page, but before the page is actually printed.
Macro

You can use the PrintCount property to check whether the Print event has occurred more than once for a record. For example, if part of a record is printed on one page and the rest is printed on the next page, the Print event occurs twice, and the PrintCount property is set to 2. You can check the PrintCount property value for Print macros that should run only once for a record, such as a macro that shows totals for the records on a page in a page footer.

You can use the CancelEvent action in a Print macro to cancel printing of a section. Microsoft Access prints a blank area on the page where the section would have been when the report is printed. You can use the CancelEvent action to skip a record in a report and leave a blank area on the page in its place when the report is printed.

A Print macro runs only for sections that are actually printed.
**Example**

The following example checks the PrintCount argument to determine the number of times the Print event has occurred. If it has occurred more than once, the Print event is canceled for the section.

To try the example, add the following event procedure to a report that contains a section named CustomerDetail.

```vba
Private Sub CustomerDetail_Print(Cancel As Integer, _
    PrintCount As Integer)
    If PrintCount > 1 Then Cancel = True
End Sub
```
Line Control (Forms and Reports)

The line control displays a horizontal, vertical, or diagonal line on a form or report.

**Note**  This control should not be confused with the Dynamic HTML line control used on a data access page. For information about a line control on a data access page, see Line Control (Data Access Pages).
Remarks

You can use **Border Width** to change the line width. You can use **Border Color** to change the color of the border or make it transparent. You can change the line style (dots, dashes, and so on) of the border by using the **BorderStyle** property.
Page Break Control

The page break control marks the start of a new screen or printed page on a form or report.
Remarks

In a form, a page break is active only when you set the form's `DefaultView` property to Single Form. Page breaks don't affect a form's datasheet.

In **Form view**, press the PAGE UP or PAGE DOWN key to move to the previous or next page break.

Position page breaks above or below other controls. Placing a page break on the same line as another control splits that control's data.
Rectangle Control (Forms and Reports)

The rectangle control displays a rectangle on a form or report.

**Note** This control should not be confused with the Dynamic HTML rectangle control used on a data access page. For information about a rectangle control on a data access page, see Rectangle Control (Data Access Page).
Remarks

You can move a rectangle and the controls in it as a single unit by dragging the mouse pointer diagonally across the entire rectangle to select all of the controls. The entire selection can then be moved to a new position.
Form Section

A form section is part of a form such as a header, footer, or detail section.

You can set section properties which are attributes of a form that affect the appearance or behavior of that section. For example, you can set the CanGrow property to specify whether the section will increase vertically to print all the data the section contains. Section properties are set in form Design view.
Report Section

A report section is part of a report such as a header, footer, or detail section.

You can set report section properties that are attributes of a report which affect the appearance or behavior of a specific section. For example, you can set the CanGrow property to specify whether the section will increase vertically to print all the data the section contains. Section properties are set in report Design view.
Use Microsoft Access as a DDE Server

Microsoft Access supports **dynamic data exchange (DDE)** as either a destination (client) application or a source (server) application. For example, an application such as Microsoft Word, acting as a client, can request data through DDE from a Microsoft Access database that's acting as a server.

**Tip**

If you need to manipulate Microsoft Access objects from another application, you may want to consider using Automation.

A DDE conversation between a client and server is established on a particular **topic**. A topic can be either a data file in the format supported by the server application, or it can be the System topic, which supplies information about the server application itself. Once a conversation has begun on a particular topic, only a **data item** associated with that topic can be transferred.

For example, suppose you are running Microsoft Word and want to insert data from a particular Microsoft Access database into a document. You begin a DDE conversation with Microsoft Access by opening a DDE channel with the **DDEInitiate** function and specifying the database file name as the topic. You can then transfer data from that database to Microsoft Word through that channel.

As a DDE server, Microsoft Access supports the following topics:

- The System topic
- The name of a database (**database topic**)
- The name of a table (**tablename topic**)
- The name of a query (**queryname topic**)
- A Microsoft Access **SQL string** (**sqlstring topic**)

Once you've established a DDE conversation, you can use the **DDEExecute** statement to send a command from the client to the server application. When
used as a DDE server, Microsoft Access recognizes any of the following as a valid command:

- The name of a macro in the current database.
- Any action that you can carry out in Visual Basic by using one of the methods of the `DoCmd` object.
- The OpenDatabase and CloseDatabase actions, which are used only for DDE operations. (For an example of how to use these actions, see the example later in this topic.)

**Note** When you specify a `macro` action as a `DDEExecute` statement, the action and any arguments follow the `DoCmd` object syntax and must be enclosed in brackets ([ ]). However, applications that support DDE don't recognize intrinsic constants in DDE operations. Also, string arguments must be enclosed in quotation marks (" ") if the string contains a comma. Otherwise, quotation marks aren't required.

The client application can use the `DDERequest` function to request text data from the server application over an open DDE channel. Or the client can use the `DDEPoke` statement to send data to the server application. Once the data transfer is complete, the client can use the `DDETerminate` statement to close the DDE channel, or the `DDETerminateAll` statement to close all open channels.

**Note** When your client application has finished receiving data over a DDE channel, it should close that channel to conserve memory resources.

The following example demonstrates how to create a Microsoft Word procedure with Visual Basic that uses Microsoft Access as a DDE server. (For this example to work, Microsoft Access must be running.)

```vb
Sub AccessDDE()  
    Dim intChan1 As Integer, intChan2 As Integer  
    Dim strQueryData As String
    
    ' Use System topic to open Northwind sample database.  
    ' Database must be open before using other DDE topics.  
    intChan1 = DDEInitiate("MSAccess", "System")  
    ' You may need to change this path to point to actual location  
    ' of Northwind sample database.  
    DDEExecute intChan1, "[OpenDatabase C:\Access\Samples\Northwind."  

    ' Get all data from Ten Most Expensive Products query.
```
intChan2 = DDEInitiate("MSAccess", "Northwind.mdb;" _
    & "QUERY Ten Most Expensive Products")
strQueryData = DDERequest(intChan2, "All")
DDETerminate intChan2

' Close database.
DDEExecute intChan1, "[CloseDatabase]"
DDETerminate intChan1

' Print retrieved data to Debug Window.
Debug.Print strQueryData
End Sub

The following sections provide information about the valid DDE topics supported by Microsoft Access.
The System Topic

The System topic is a standard topic for all Microsoft Windows–based applications. It supplies information about the other topics supported by the application. To access this information, your code must first call the **DDEInitiate** function with "System" as the *topic* argument, and then execute the **DDERequest** statement with one of the following supplied for the *item* argument.

<table>
<thead>
<tr>
<th>Item</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>SysItems</td>
<td>A list of items supported by the System topic in Microsoft Access.</td>
</tr>
<tr>
<td>Formats</td>
<td>A list of the formats Microsoft Access can copy onto the Clipboard.</td>
</tr>
<tr>
<td>Status</td>
<td>&quot;Busy&quot; or &quot;Ready&quot;.</td>
</tr>
<tr>
<td>Topics</td>
<td>A list of all open databases.</td>
</tr>
</tbody>
</table>

The following example demonstrates the use of the **DDEInitiate** and **DDERequest** functions with the System topic:

```vbs
' In Visual Basic, initiate DDE conversation with Microsoft Access.
Dim intChan1 As Integer, strResults As String
intChan1 = DDEInitiate("MSAccess", "System")
' Request list of topics supported by System topic.
strResults = DDERequest(intChan1, "SysItems")
' Run OpenDatabase action to open Northwind.mdb.
' You may need to change this path to point to actual location
' of Northwind sample database.
DDEExecute intChan1, "[OpenDatabase C:\Access\Samples\Northwind.mdb]"
```
The **database** Topic

The *database* topic is the file name of an existing database. You can type either just the base name (Northwind), or its path and .mdb extension (C:\Access\Samples\Northwind.mdb). After you start a DDE conversation with the database, you can request a list of the objects in that database.

**Note** You can't use DDE to query the Microsoft Access workgroup information file.

The *database* topic supports the following items.

<table>
<thead>
<tr>
<th>Item</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>TableList</td>
<td>A list of tables.</td>
</tr>
<tr>
<td>QueryList</td>
<td>A list of queries.</td>
</tr>
<tr>
<td>FormList</td>
<td>A list of forms.</td>
</tr>
<tr>
<td>ReportList</td>
<td>A list of reports.</td>
</tr>
<tr>
<td>MacroList</td>
<td>A list of macros.</td>
</tr>
<tr>
<td>ModuleList</td>
<td>A list of modules.</td>
</tr>
<tr>
<td>ViewList</td>
<td>A list of views</td>
</tr>
<tr>
<td>StoredProcedureList</td>
<td>A list of stored procedures</td>
</tr>
<tr>
<td>DatabaseDiagramList</td>
<td>A list of database diagrams</td>
</tr>
</tbody>
</table>

The following example shows how you can open the Employees form in the Northwind sample database from a Visual Basic procedure:

```vbnet
' In Visual Basic, initiate DDE conversation with
' Northwind sample database.
' Make sure database is open.
intChan2 = DDEInitiate("MSAccess", "Northwind")
' Request list of forms in Northwind sample database.
strResponse = DDERequest(intChan2, "FormList")
' Run OpenForm action and arguments to open Employees form.
DDEExecute intChan2, ":[OpenForm Employees,0,,,1,0]"
```
The **TABLE** `tablename`, **QUERY** `queryname`, and **SQL** `sqlstring` Topics

These topics use the following syntax:

```
databasename; TABLE tablename

databasename; QUERY queryname

databasename; SQL [sqlstring]
```

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>databasename</code></td>
<td>The name of the database that the table or query is in or that the SQL statement applies to, followed by a semicolon (;). The database name can be just the base name (Northwind) or its full path and .mdb extension (C:\Access\Samples\Northwind.mdb).</td>
</tr>
<tr>
<td><code>tablename</code></td>
<td>The name of an existing table.</td>
</tr>
<tr>
<td><code>queryname</code></td>
<td>The name of an existing query. A valid SQL statement up to 256 characters long, ending with a semicolon. To exchange more than 256 characters, omit this argument and instead use successive <strong>DDEPoke</strong> statements to build an SQL statement.</td>
</tr>
<tr>
<td><code>sqlstring</code></td>
<td>For example, the following Visual Basic code uses the <strong>DDEPoke</strong> statement to build an SQL statement and then request the results of the query.</td>
</tr>
</tbody>
</table>

```vba
intChan1 = DDEInitiate("MSAccess", "Northwind;SQL")
DDEPoke intChan1, "SQLText", "SELECT *
DDEPoke intChan1, "SQLText", " FROM Orders"
DDEPoke intChan1, "SQLText", " WHERE [Freight] > 100;"
strResponse = DDERequest(intChan1, "NextRow")
DDETerminate intChan1
```

The following table lists the valid items for the **TABLE** `tablename`, **QUERY** `queryname`, and **SQL** `sqlstring` topics:
*queryname*, and SQL *sqlstring* topics.

<table>
<thead>
<tr>
<th>Item</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>All the data in the table, including field names.</td>
</tr>
<tr>
<td>Data</td>
<td>All rows of data, without field names.</td>
</tr>
<tr>
<td>FieldNames</td>
<td>A single-row list of field names.</td>
</tr>
<tr>
<td>FieldNames:T</td>
<td>A two-row list of field names (first row) and their data types (second row).</td>
</tr>
</tbody>
</table>

These are the values returned and the data types they represent:

<table>
<thead>
<tr>
<th>Value</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Invalid</td>
</tr>
<tr>
<td>1</td>
<td><strong>True/False</strong></td>
</tr>
<tr>
<td></td>
<td>(non-Null)</td>
</tr>
<tr>
<td>2</td>
<td>Unsigned byte</td>
</tr>
<tr>
<td>3</td>
<td>2-byte signed integer (Integer)</td>
</tr>
<tr>
<td>4</td>
<td>4-byte signed integer (Long)</td>
</tr>
<tr>
<td>5</td>
<td>8-byte signed integer (Currency)</td>
</tr>
<tr>
<td>6</td>
<td>4-byte single-precision floating-point (Single)</td>
</tr>
<tr>
<td>8</td>
<td>8-byte double-precision floating-point (Double)</td>
</tr>
</tbody>
</table>
NextRow

The data in the next row in the table or query.
When you open a channel, NextRow returns the data in the first row. If the current row is the last record and you run NextRow, the request fails.

PrevRow

The data in the previous row in the table or query. If PrevRow is the first request on a new channel, the data in the last row of the table or query is returned. If the first record is the current row, the request for PrevRow fails.

FirstRow

The data in the first row of the table or query.

LastRow

The data in the last row of the table or query.

FieldCount

The number of fields in the table or query.

SQLText

An SQL statement representing the table or query. For tables, this item returns an SQL statement in the form "SELECT * FROM table;". An SQL statement, in n-character chunks, representing the table or query, where n is an integer up to 256. For example, suppose a query is
represented by the following SQL statement:
"SELECT * FROM Orders;"

The item "SQLText;7" returns the following tab-delimited chunks:

"SELECT \\
"*	FROM \\
"Orders;"

The following example shows how you can use DDE in a Visual Basic procedure to request data from a table in the Northwind sample database and insert that data into a text file:

Sub NorthwindDDE
    Dim intChan1 As Integer, intChan2 As Integer, intChan3 As Integer
    Dim strResp1 As Variant, strResp2 As Variant, strResp3 As Variant

    ' In a Visual Basic module, get data from Categories table, 
    ' Catalog query, and Orders table in Northwind.mdb. 
    ' Make sure database is open first.
    intChan1 = DDEInitiate("MSAccess", "Northwind;TABLE Shippers")
    intChan2 = DDEInitiate("MSAccess", "Northwind;QUERY Catalog")
    intChan3 = DDEInitiate("MSAccess", "Northwind;SQL SELECT * " _
        & "FROM Orders " _
        & "WHERE OrderID > 10050;")

    strResp1 = DDERequest(intChan1, "All")
    strResp2 = DDERequest(intChan2, "FieldNames;T")
    strResp3 = DDERequest(intChan3, "FieldNames;T")
    DDETerminate intChan1
    DDETerminate intChan2
    DDETerminate intChan3

    ' Insert data into text file.
    Open "C:\DATA.TXT" For Append As #1
    Print #1, strResp1
    Print #1, strResp2
Print #1, strResp3
Close #1
End Sub
Automation with Microsoft Access

Microsoft Access is a **COM component** that supports **Automation**, formerly called OLE Automation. Microsoft Access supports Automation in two ways. From Microsoft Access, you can work with objects supplied by another component. Microsoft Access also supplies its objects to other COM components.

In previous versions of Microsoft Access, you could use the **CreateObject** function or the **GetObject** function to point a variable to an instance of a component. In Microsoft Access 97 and above, you can also use the **New** keyword to create a new instance of some components.

In Microsoft Access, you can set a reference to a component's **type library** to improve performance when you work with that component through Automation. Microsoft Access also includes the **Object Browser**, a tool that enables you to view objects in another component's type library, as well as their methods and properties.

The Microsoft Access type library provides information about **Microsoft Access objects** to other components. You can **set a reference** to the Microsoft Access type library from a component and view its objects in the Object Browser.

To work with Microsoft Access objects through Automation, you must create an instance of the Microsoft Access **Application** object. For example, suppose you want to display data from Microsoft Excel in a Microsoft Access form or report. To launch Microsoft Access from Microsoft Excel, you can use the **New** keyword to create an instance of the Microsoft Access **Application** object. You can also use the **CreateObject** function to create a new instance of the Microsoft Access **Application** object, or you can use the **GetObject** function to point an object variable to an existing instance of Microsoft Access. Check your component's documentation to determine which syntax it supports.

Once you've launched an instance of Microsoft Access, if you want to control any Microsoft Access objects, you must open a **database** (.mdb) or **project** (.adp) in the Microsoft Access window by using either the **OpenCurrentDatabase** or the **NewCurrentDatabase** method for a database or by using the
OpenAccessProject or the NewAccessProject method for a project.

If you've opened Microsoft Access only as a means of using the Data Access Objects provided by Microsoft DAO, then you don't need to open a database in the Microsoft Access window. You can use the DBEngine property of the Microsoft Access Application object to access objects in the Microsoft DAO 3.6 object library during an Automation operation.
Show All
**DDETerminate Statement**

You can use the **DDETerminate** statement to close a specified dynamic data exchange (DDE) channel.

For example, if you've opened a DDE channel to transfer data between Microsoft Excel and Microsoft Access, you can use the **DDETerminate** statement to close that channel once the transfer is complete.

**DDETerminate (channum)**
Remarks

The *channum* argument specifies the channel number to close. It refers to a channel opened by the [DDEInitiate](function) function. If the *channum* argument isn't an integer corresponding to an open channel, a run-time error occurs.

Once a channel is closed, any subsequent DDE functions or statements performed on that channel cause a run-time error.

The [DDETerminate](statement) statement has no effect on active DDE link expressions in fields on forms or reports.

Tip

If you need to manipulate another application's objects from Microsoft Access, you may want to consider using Automation.
DeleteObject Action

You can use the DeleteObject action to delete a specified database object.
Setting

The DeleteObject action has the following arguments.

<table>
<thead>
<tr>
<th>Action argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object Type</td>
<td>The type of object to delete. Click Table, Query, Form, Report, Macro, Module, Data Access Page, Server View, Diagram, Stored Procedure, or Function in the Object Type box in the Action Arguments section of the Macro window. To delete the object selected in the Database window, leave this argument blank. The name of the object to delete. The Object Name box shows all objects in the database of the type selected by the Object Type argument. If you leave the Object Type box blank, leave this box blank also.</td>
</tr>
<tr>
<td>Object Name</td>
<td>If you run a macro containing the DeleteObject action in a library database, Microsoft Access looks for the object with this name first in the library database, then in the current database.</td>
</tr>
</tbody>
</table>

**Warning** If you leave the Object Type and Object Name boxes blank, Access deletes the object selected in the Database window without displaying a warning message when it encounters the DeleteObject action.
Remarks

You can use the DeleteObject action to delete temporary objects you've created while running the macro. For example, you could use the OpenQuery action to run a make-table query that creates a temporary table. When you're finished using the temporary table, you can use the DeleteObject action to delete it.

This action has the same effect as selecting an object in the Database window and then pressing the DEL key or clicking Delete on the Edit menu.

To run the DeleteObject action in Visual Basic, you can use the DeleteObject method of the DoCmd object.
Calculate Fields in Domain Aggregate Functions

You can use the string expression argument (the expr argument) in a domain aggregate function to perform a calculation on values in a field. For example, you can calculate a percentage (such as a surcharge or sales tax) by dividing a field value by a number.

The following table provides examples of calculations on fields from an Orders table and an Order Details table.

<table>
<thead>
<tr>
<th>Calculation</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add a number to a field</td>
<td>&quot;[Freight] + 5&quot;</td>
</tr>
<tr>
<td>Subtract a number from a field</td>
<td>&quot;[Freight] - 5&quot;</td>
</tr>
<tr>
<td>Multiply a field by a number</td>
<td>&quot;[Freight] * 2&quot;</td>
</tr>
<tr>
<td>Divide a field by a number</td>
<td>&quot;[Freight] / 2&quot;</td>
</tr>
<tr>
<td>Add one field to another</td>
<td>&quot;[UnitsInStock] + [UnitsOnOrder]&quot;</td>
</tr>
<tr>
<td>Subtract one field from another</td>
<td>&quot;[ReorderLevel] - [UnitsInStock]&quot;</td>
</tr>
</tbody>
</table>

You would most likely use a domain aggregate function in a macro or module, in a calculated control on a form or report, or in a criteria expression in a query.

For example, you can calculate the average discount amount for all orders in an Order Details table. Multiply the Unit Price and Discount fields to determine the discount for each order, then calculate the average. Enter the following example in a procedure in a module.

```
DimdblX As Double
dblX = DAvg("[UnitPrice] * [Discount]", "[Order Details]")
```
Show All
TopValues Property

You can use the TopValues property to return a specified number of records or a percentage of records that meet the criteria you specify. For example, you might want to return the top 10 values or the top 25 percent of all values in a field.

Note The TopValues property applies only to append, make-table, and select queries.
Setting

The TopValues property setting is an Integer value representing the exact number of values to return or a number followed by a percent sign (%) representing the percentage of records to return. For example, to return the top 10 values, set the TopValues property to 10; to return the top 10 percent of values, set the TopValues property to 10%.

You can't set this property in code directly. It's set in SQL view of the Query window by using a TOP n or TOP n PERCENT clause in the SQL statement.

You can also set the TopValues property by using the query's property sheet or the Top Values box on the Query Design toolbar.

Note  The TopValues property in the query's property sheet and on the Query Design toolbar is a combo box that contains a list of values and percentage values. You can select one of these values or you can type any valid setting in the text box portion of this control.
Remarks

Typically, you use the **TopValues** property setting together with sorted fields. The field you want to display top values for should be the leftmost field that has the **Sort** box selected in the *query design grid*. An ascending sort returns the bottommost records, and a descending sort returns the topmost records. If you specify that a specific number of records be returned, all records with values that match the value in the last record are also returned.

For example, suppose a set of employees has the following sales totals.

<table>
<thead>
<tr>
<th>Sales</th>
<th>Salesperson</th>
</tr>
</thead>
<tbody>
<tr>
<td>90,000</td>
<td>Leverling</td>
</tr>
<tr>
<td>80,000</td>
<td>Peacock</td>
</tr>
<tr>
<td>70,000</td>
<td>Davolio</td>
</tr>
<tr>
<td>70,000</td>
<td>King</td>
</tr>
<tr>
<td>60,000</td>
<td>Suyama</td>
</tr>
<tr>
<td>50,000</td>
<td>Buchanan</td>
</tr>
</tbody>
</table>

If you set the **TopValues** property to 3 with a descending sort on the Sales field, Microsoft Access returns the following four records.

<table>
<thead>
<tr>
<th>Sales</th>
<th>Salesperson</th>
</tr>
</thead>
<tbody>
<tr>
<td>90,000</td>
<td>Leverling</td>
</tr>
<tr>
<td>80,000</td>
<td>Peacock</td>
</tr>
<tr>
<td>70,000</td>
<td>Davolio</td>
</tr>
<tr>
<td>70,000</td>
<td>King</td>
</tr>
</tbody>
</table>

**Note** To return the topmost or bottommost values without displaying duplicate values, set the **UniqueValues** property in the query's property sheet to Yes.
Example

The following example assigns an SQL string that returns the top 10 most expensive products to the **RecordSource** property for a form that will display the ten most expensive products.

```
Dim strGetSQL As String
strGetSQL = "SELECT TOP 10 Products.[ProductName] " 
& "AS TenMostExpensiveProducts, Products.UnitPrice FROM Products " 
& "ORDER BY Products.[UnitPrice] DESC;"
Me.RecordSource = strGetSQL
```
Set Properties by Using Visual Basic

You can set most properties from Visual Basic code. How you set a property depends on whether you are setting it for a **Form**, **Report**, or **Control** object, an ActiveX Data Object (ADO), a Data Access Object (DAO), or for a **DataAccessPage** object. The following topics outline the steps involved for setting properties of each type of object:

- [Set Form, Report, and Control Properties in Visual Basic](#)
- [Set Properties of ActiveX Data Objects in Visual Basic](#)
- [Set Properties of Data Access Objects in Visual Basic](#)
- [Set Properties of Data Access Pages in Visual Basic](#)
Hwnd Property

You can use the hWnd property to determine the handle (a unique Long Integer value) assigned by Microsoft Windows to the current window. Read/write Long.

expression.Hwnd

expression Required. An expression that returns one of the objects in the Applies To list.
Remarks

This property is available only by using a macro or Visual Basic.

You can use this property in Visual Basic when making calls to Windows application programming interface (API) functions or other external routines that require the hWnd property as an argument. Many Windows functions require the hWnd property value of the current window as one of the arguments.

**Caution**  Because the value of this property can change while a program is running, don't store the hWnd property value in a public variable.
Example

The following example uses the hWnd property with the Windows API IsZoomed function to determine if a window is maximized.

' Enter on single line in Declarations section of Module window. Declare Function IsZoomed Lib "user32" (ByVal hWnd As Long) As Long

Sub Form_Activate()
    Dim intWindowHandle As Long
    intWindowHandle = Screen.ActiveForm.hWnd
    If Not IsZoomed(intWindowHandle) Then
        DoCmd.Maximize
    End If
End Sub
ReturnsRecords Property

You can use the ReturnsRecords property in an SQL pass-through query to specify whether the query returns records. For example, if your pass-through query is `SELECT * FROM EMPLOYEES`, you can set the ReturnsRecords property to Yes to display all employee records in the query's datasheet.

Note  The ReturnsRecords property applies only to pass-through queries.
Setting

The **ReturnsRecords** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>True</td>
<td>(Default) The query returns records (as a select query does).</td>
</tr>
<tr>
<td>No</td>
<td>False</td>
<td>The query doesn't return records (as an action query does).</td>
</tr>
</tbody>
</table>

You can set this property by using the query's property sheet or **Visual Basic**.

Tip

You can access a pass-through query's **ReturnsRecords** property in Visual Basic, but the preferred method is to use the DAO **ReturnsRecords** property.
Remarks

A pass-through query can return records, or it can be used to change data, create a database object, or perform an action as an action query does.

When you set the ReturnsRecords property to Yes, you can use the pass-through query in another query or as the basis for a combo box, list box, form, or report. If you set the ReturnsRecords property to No, you can't use the query as the basis for an object or control because no records are returned.
AppTitle Property

You can use the AppTitle property to specify the text that appears in the application database's title bar. For example, you can use the AppTitle property to specify that the string "Inventory Control" appear in the title bar of your Inventory Control database application.
Setting

The **AppTitle** property is a *string expression* containing the text to appear in the title bar.

The easiest way to set this property is by using the **Application Title** option in the **Startup** dialog box, available by clicking **Startup** on the **Tools** menu. You can also set this property by using a **macro** or **Visual Basic**.

To set the **AppTitle** property by using a macro or Visual Basic, you must first either set the property in the **Startup** dialog box once or create the property in the following ways:

- In a **Microsoft Access database** (.mdb), you can add it by using the **CreateProperty** method and append it to the **Properties** collection of the **Database** object.
- In a **Microsoft Access project** (.adp), you can add it to the **AccessObjectProperties** collection of the **CurrentProject** object by using the **Add** method.

You must also use the **RefreshTitleBar** method to make any changes visible immediately.
Remarks

If this property isn't set, the string "Microsoft Access" appears in the title bar.

This property's setting takes effect immediately after setting the property in code (as long as the code includes the RefreshTitleBar method) or closing the Startup dialog box.
AppIcon Property

You can use the AppIcon property to specify the name of the bitmap (.bmp) or icon (.ico) file that contains the application's icon. For example, you can use the AppIcon property to specify a .bmp file that contains a picture of an automobile to represent an automotive parts application.
Setting

The **AppIcon** property is a *string expression* that's a valid bitmap or icon file name (including the path).

The easiest way to set this property is by using the **Application Icon** option in the **Startup** dialog box, available by clicking **Startup** on the **Tools** menu. You can also set this property by using a **macro** or **Visual Basic**.

To set the **AppIcon** property by using a macro or Visual Basic, you must first either set the property in the **Startup** dialog box once or create the property in the following ways:

- In a **Microsoft Access database** (.mdb), you can add it by using the **CreateProperty** method and append it to the **Properties** collection of the **Database** object.
- In a **Microsoft Access project** (.adp), you can add it to the **AccessObjectProperties** collection of the **CurrentProject** object by using the **Add** method.

You must also use the **RefreshTitleBar** method to make any changes visible immediately.
Remarks

If you are distributing your application, it's recommended that the .bmp or .ico file containing the icon reside in the same directory as your Microsoft Access application.

If the **AppIcon** property isn't set or is invalid, the Microsoft Access icon is displayed.

This property setting takes effect immediately after it's set in code (as long as the code includes the **RefreshTitleBar** method) or the **Startup** dialog box is closed.
Example

The following example shows how to change the **AppIcon** and **AppTitle** properties in a Microsoft Access database (.mdb). If the properties haven't already been set or created, you must create them and append them to the **Properties** collection by using the **CreateProperty** method.

```vba
Sub cmdAddProp_Click()
    Dim intX As Integer
    Const DB_Text As Long = 10
    intX = AddAppProperty("AppTitle", DB_Text, "My Custom Application")
    intX = AddAppProperty("AppIcon", DB_Text, "C:WindowsCars.bmp")
    CurrentDb.Properties("UseAppIconFor FrmRpt") = 1
    Application.RefreshTitleBar
End Sub

Function AddAppProperty(strName As String, _
    varType As Variant, varValue As Variant) As Integer
    Dim dbs As Object, prp As Variant
    Const conPropNotFoundError = 3270

    Set dbs = CurrentDb
    On Error GoTo AddProp_Err
    dbs.Properties(strName) = varValue
    AddAppProperty = True

AddProp_Bye:
    Exit Function

AddProp_Err:
    If Err = conPropNotFoundError Then
        Set prp = dbs.CreateProperty(strName, varType, varValue)
        dbs.Properties.Append prp
        Resume
    Else
        AddAppProperty = False
        Resume AddProp_Bye
    End If
End Function
```
Chart Control (Forms and Reports)

You can use the chart control to embed a chart that displays Microsoft Access data from a form or report. You can then edit the chart by using Microsoft Graph from within the form or report.

**Note** This control should not be confused with the Dynamic HTML image, hotspot image, or Office chart, or Office Spreadsheet control used on a data access page. For information about these control on a data access page, see [Image](#), [Image Hyperlink](#), [Office Chart](#), or [Office Spreadsheet](#) Control (Data Access Pages).
Remarks

When you place a chart control on a form or report, Microsoft Access displays the Chart Wizard to help you create the chart.

Note  To embed or link a chart containing data from other applications, use an unbound object frame or bound object frame control.
OLETypeAllowed Property

You can use the **OLETypeAllowed** property to specify the type of OLE object a control can contain. Read/write **Byte**.

expression.OLETypeAllowed

*expression*  Required. An expression that returns one of the objects in the Applies To list.
Remarks

The **OLETypeAllowed** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linked</td>
<td><code>acOLELinked</code></td>
<td>The control can contain only a linked object.</td>
</tr>
<tr>
<td>Embedded</td>
<td><code>acOLEEmbedded</code></td>
<td>The control can contain only an embedded object.</td>
</tr>
<tr>
<td>Either</td>
<td><code>acOLEEither</code></td>
<td>(Default) The control can contain either a linked or an embedded object.</td>
</tr>
</tbody>
</table>

**Note**  You can set the **OLETypeAllowed** property by using the control’s property sheet, a macro, or Visual Basic. You can set the default for this property by using a control's default control style or the `DefaultControl` method in Visual Basic.

**Note**  For unbound object frames and charts, you can't change the **OLETypeAllowed** setting after an object is created. For bound object frames, you can change the setting after the object is created. Changing the **OLETypeAllowed** property setting only affects new objects that you add to the control.

To determine the type of OLE object a control already contains, you can use the **OLEType** property.
Example

The following example creates a linked OLE object using an unbound object frame named OLE1 and sizes the control to display the object's entire contents when the user clicks a command button.

Sub Command1_Click
  OLE1.Class = "Excel.Sheet"  ' Set class name.
  ' Specify type of object.
  OLE1.OLETypeAllowed = acOLELinked
  ' Specify source file.
  OLE1.SourceDoc = "C:\Excel\Oletext.xls"
  ' Specify data to create link to.
  OLE1.SourceItem = "R1C1:R5C5"
  ' Create linked object.
  OLE1.Action = acOLECreateLink
  ' Adjust control size.
  OLE1.SizeMode = acOLESizeZoom
End Sub
OLEType Property

You can use the OLEType property to determine if a control contains an OLE object, and, if so, whether the object is linked or embedded. Read/write Byte.

expression.OLEType

expression  Required. An expression that returns one of the objects in the Applies To list.
**Remarks**

The **OLEType** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linked</td>
<td>acOLELinked</td>
<td>The control contains a linked object. All the object's data is managed by the application that created it.</td>
</tr>
<tr>
<td>Embedded</td>
<td>acOLEEmbedded</td>
<td>The control contains an embedded object. All the object's data is managed by Microsoft Access.</td>
</tr>
<tr>
<td>None</td>
<td>acOLENone</td>
<td>The control doesn't contain an OLE object.</td>
</tr>
</tbody>
</table>

**Note** When creating an OLE object, use the **OLETypeAllowed** property to determine what type of object a control can contain.
Example

The following example illustrates how to display the Insert Object dialog box and how to display an error message if the Cancel button in the Insert Object dialog box is clicked.

Sub InsertObject_Click()
    Dim conUserCancelled As Integer

    ' Error message returned when user cancels.
    conUserCancelled = 2001
    On Error GoTo ButtonErr
    If OLE1.OLEType = acOLENone Then
        ' No OLE object created.
        ' Display Insert Object dialog box.
        OLE1.Action = acOLEInsertObjDlg
    End If
    Exit Sub

ButtonErr:
    If Err = conUserCancelled Then ' Display message.
        MsgBox "You clicked the Cancel button; " _
            & "no object was created."
    End If
    Resume Next
End Sub
Set Properties by Using Macros

From a macro, you can set properties for **Form**, **Report**, and **Control** objects, as well as form and report **sections**. You use the `SetValue` action from a macro to set the value of a property.

You can't use a macro to set properties of other **Microsoft Access objects** or **ActiveX Data Objects (ADO)**, or to set the default properties of a control, but you can set them either by using Visual Basic or an object's **property sheet** in **Design view**.

**To set a form, report, or control property by using a macro**

1. In a macro, add a `SetValue` action.
2. Set the `Item` action argument of the `SetValue` action to an expression that refers to the property you want to set:
   - To set a form or report property, use the syntax `Forms!formname.propertyname` or `Reports!reportname.propertyname`. For example, the following expression refers to the **Visible** property of the Customers form:
     
     `Forms!Customers.Visible`
   - To set a property of a **control** on a form or report, use the syntax `Forms!formname!controlname.propertyname` or `Reports!reportname!controlname.propertyname`. For example, the following expression refers to the **Visible** property of the HiddenPageBreak control on the Invoices report:
     
     `Reports!Invoices!HiddenPageBreak.Visible`

**Tip**

If the macro containing the `SetValue` action runs from the form or report with the property you want to set, you can refer to the property by using just the syntax **propertyname**. However, it's a good idea to use the full syntax to
refer to the property to avoid conflicts with names of controls or Visual Basic keywords. For example, Name is a Microsoft Access property; if you also have a control on your form called Name, you should use the full syntax to refer to both the control and the property.

3. Set the Expression action argument of the SetValue action to the value you want to set the property to. If the setting is a string, be sure to enclose it in double (") quotation marks. For example, to set the Caption property of a form to Orders, you would enter "Orders" in the Expression argument.

To set a section property by using a macro

1. In a macro, add a SetValue action.
2. In the Item action argument, use the syntax `Forms!formname.Section(constant).propertyname` to refer to the property you want to set. The constant argument refers to a particular section on the form or report, as described in the Section property. For example, the following expression refers to the Visible property of the page header section of the Customers form:

   `Forms!Customers.Section(acPageHeader).Visible`

3. Set the Expression action argument as described above.

Note  For each property you want to set, you can look up the property in the Help index to find information about:

- Whether you can set the property from a macro.
- Views in which you can set the property. Not all properties can be set in all views. For example, you can set a form's BorderStyle property only in form Design view.
- Which values you can use to set the property. Some properties require settings that don't correspond to values in the property sheet but to numeric values. You may need to set the property by using the settings you would use in Visual Basic rather than those you would use in the property sheet. For example, if the property settings are selections from a list, you must use the value or numeric equivalent for each selection.
OpenForm Action

You can use the OpenForm action to open a form in Form view, form Design view, Print Preview, or Datasheet view. You can select data entry and window modes for the form and restrict the records that the form displays.
## Setting

The OpenForm action has the following arguments.

<table>
<thead>
<tr>
<th>Action argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Form Name</td>
<td>The name of the form to open. The Form Name box in the Action Arguments section of the Macro window shows all forms in the current database. This is a required argument.</td>
</tr>
<tr>
<td>View</td>
<td>The view in which the form will open. Click Form, Design, Print Preview, Datasheet, PivotTable, or PivotChart in the View box. The default is Form.</td>
</tr>
<tr>
<td>Filter Name</td>
<td>A filter that restricts or sorts the form's records. You can enter the name of either an existing query or a filter that was saved as a query. However, the query must include all the fields in the form you are opening or have its OutputAllFields property set to Yes. A valid SQL WHERE clause (without the word WHERE) or expression that Access uses to select records from the form's underlying table or query. If you select a filter with the Filter Name argument, Access applies this WHERE clause to the results of the filter.</td>
</tr>
</tbody>
</table>
| Where            | To open a form and restrict its records to those specified by the value of a control on another form, use the following expression: 

\[ \text{[fieldname]} = \text{Forms! [formname]! [controlname on other form]} \]
Condition  The *fieldname* argument is the name of a field in the underlying table or query of the form you want to open. The *controlname on other form* argument is the name of the control on the other form that contains the value you want records in the first form to match.

**Note**  The maximum length of the Where Condition argument is 255 characters. If you need to enter a more complex SQL WHERE clause longer than this, use the **OpenForm** method of the **DoCmd** object in Microsoft Visual Basic instead. You can enter SQL WHERE clause statements of up to 32,768 characters in Visual Basic.

The data entry mode for the form. This applies only to forms opened in Form view or Datasheet view. Click **Add** (the user can add new records but can't edit existing records), **Edit** (the user can edit existing records and add new records), or **Read Only** (the user can only view records). The default is **Edit**.

**Notes**

- The Data Mode argument setting overrides the settings of the form's **AllowEdits**, **AllowDeletions**, **AllowAdditions**, and **DataEntry** properties. For example, if a form's **AllowEdits** property is set to **No**, you can still use the OpenForm action to open the form in Edit mode.
- If you leave this argument blank, Access opens the form in the data entry mode set by the form's **AllowEdits**, **AllowDeletions**, **AllowAdditions**, and **DataEntry** properties.

Window Mode

The window mode in which the form opens. Click **Normal** so the form is in the mode set by its **properties**, **Hidden** (the form is hidden), **Icon** (the form opens minimized as a small title bar at the bottom of the screen), or **Dialog** (the form's **Modal** and **PopUp** properties are set to **Yes**). The default is **Normal**.
Remarks

This action is similar to clicking the **Open** button or **Design** button after clicking **Forms** under **Objects**, and then selecting a form in the **Database window**.

A form can be **modal** (it must be closed or hidden before the user can perform any other action) or **modeless** (the user can move to other windows while the form is open). It can also be a **pop-up form** (a form used to collect or display information that remains on top of all other Access windows). You set the **Modal** and **PopUp** properties when you design the form. If you use **Normal** for the Window Mode argument, the form opens in the mode specified by these property settings. If you use **Dialog** for the Window Mode argument, these properties are both set to **Yes**. A form opened as hidden or as an icon returns to the mode specified by its property settings when you show or restore it.

When you open a form with the Window Mode argument set to **Dialog**, Access suspends the macro until the form is closed or hidden. You can hide a form by setting its **Visible** property to **No** by using the SetValue action.

**Tip**

You can select a form in the Database window and drag it to a macro **action row**. This automatically creates an OpenForm action that opens the form in Form view.

The filter and WHERE condition you apply become the setting of the form's **Filter** property.
Examples

Set the value of a control by using a macro

The following macro opens the Add Products form from a button on the Suppliers form. It shows the use of the Echo, Close, OpenForm, SetValue, and GoToControl actions. The SetValue action sets the Supplier ID control on the Products form to the current supplier on the Suppliers form. Then the GoToControl action moves the focus to the Category ID field, where you can begin to enter data for the new product. This macro should be attached to the Add Products button on the Suppliers form.

<table>
<thead>
<tr>
<th>Action</th>
<th>Arguments: Setting</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Echo</td>
<td>Echo On: No</td>
<td>Stop screen updating while the macro is running.</td>
</tr>
<tr>
<td></td>
<td>Object Type: Form</td>
<td>Object Name: Product List</td>
</tr>
<tr>
<td>Close</td>
<td>Save: No</td>
<td>Close Product List form.</td>
</tr>
<tr>
<td></td>
<td>Form Name: Products</td>
<td></td>
</tr>
<tr>
<td></td>
<td>View: Form</td>
<td></td>
</tr>
<tr>
<td>OpenForm</td>
<td>Data Mode: Add</td>
<td>Open the Products form.</td>
</tr>
<tr>
<td></td>
<td>Window Mode: Normal</td>
<td></td>
</tr>
<tr>
<td>SetValue</td>
<td>Item: [Forms]!</td>
<td>Set the Supplier ID control to the current supplier on the Suppliers form.</td>
</tr>
<tr>
<td></td>
<td>[Products]!</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[SupplierID]</td>
<td></td>
</tr>
</tbody>
</table>
Synchronize forms by using a macro

The following macro opens a Product List form in the lower-right corner of the Suppliers form, displaying the current supplier's products. It shows the use of the Echo, MsgBox, GoToControl, StopMacro, OpenForm, and MoveSize actions. It also shows the use of a conditional expression with the MsgBox, GoToControl, and StopMacro actions. This macro should be attached to the Review Products button on the Suppliers form.

<table>
<thead>
<tr>
<th>Condition</th>
<th>Action</th>
<th>Arguments: Setting</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Echo</td>
<td>Echo On: No</td>
<td>Stop screen updating while the macro is running.</td>
</tr>
<tr>
<td></td>
<td>MsgBox</td>
<td>Beep: Yes, Type: None, Title: Select a Supplier</td>
<td>If there is no current supplier on the Suppliers form, display a message.</td>
</tr>
<tr>
<td>IsNull([SupplierID])</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

... GoToControl Control Name: CompanyName
... StopMacro

Form Name: Product List
View: Datasheet
OpenForm

Filter Name: 
Where Condition: 
[SupplierID] = [Forms]![Suppliers]![SupplierID]

Data Mode: **Read Only**

Window Mode: **Normal**

MoveSize

Right: 0.7799"
Down: 1.8"

Position the Product List form in the lower right of the Suppliers form.

Open the Product List form and show the current supplier's products.
ApplyFilter Action

You can use the ApplyFilter action to apply a filter, a query, or an SQL WHERE clause to a table, form, or report to restrict or sort the records in the table, or the records from the underlying table or query of the form or report. For reports, you can use this action only in a macro specified by the report's OnOpen event property.

Note  You can use this action to apply an SQL WHERE clause only when applying a server filter. A server filter cannot be applied to a stored procedure's record source.
### Setting

The ApplyFilter action has the following arguments.

<table>
<thead>
<tr>
<th>Action argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Filter Name</td>
<td>The name of a filter or query that restricts or sorts the records of the table, form, or report. You can enter the name of either an existing query or a filter that has been saved as a query in the <strong>Filter Name</strong> box in the <strong>Action Arguments</strong> section of the <strong>Macro window</strong>. Note When you are using this action to apply a server filter, the Filter Name argument must be blank.</td>
</tr>
<tr>
<td>Where Condition</td>
<td>A valid SQL WHERE clause (without the word WHERE) or an expression that restricts the records of the table, form, or report. Note In a Where Condition argument expression, the left side of the expression typically contains a field name from the underlying table or query for the form or report. The right side of the expression typically contains the criteria you want to apply to this field in order to restrict or sort the records. For example, the criteria can be the name of a control on another form that contains the value you want the records in the first form to match. The name of the control should be fully qualified, for example: <strong>Forms!formname!controlname</strong></td>
</tr>
</tbody>
</table>

Field names should be surrounded by double quotation marks and string literals should be surrounded by single quotation marks.

The maximum length of the Where Condition argument is 255 characters. If you need to enter a longer SQL WHERE clause, use the **ApplyFilter** method of the **DoCmd** object in Microsoft Visual Basic. You can enter SQL WHERE clause statements of up to 32,768 characters in Visual Basic.

Note You can use the Filter Name argument if you've already defined a filter.
that provides the appropriate data. You can use the Where Condition argument to enter the restriction criteria directly. If you use both arguments, Microsoft Access applies the WHERE clause to the results of the filter. You must use one or both arguments.
Remarks

You can apply a filter or query to a form in Form view or Datasheet view.

The filter and WHERE condition you apply become the setting of the form's or report's Filter or ServerFilter property.

For tables and forms, this action is similar to clicking Apply Filter/Sort or Apply Server Filter on the Records menu. The menu command applies the most recently created filter to the table or form, whereas the ApplyFilter action applies a specified filter or query.

In an Access database, if you point to Filter on the Records menu and then click Advanced Filter/Sort after running the ApplyFilter action, the Advanced Filter/Sort window shows the filter criteria you've selected with this action.

To remove a filter and display all of the records for a table or form in an Access database, you can use the ShowAllRecords action or the Remove Filter/Sort command on the Records menu. To remove a filter in an Access project (.adp), you can return to the Server Filter By Form window and remove all filter criteria and then click Apply Server Filter on the Records menu on the toolbar, or set the ServerFilterByForm property to False (0).

When you save a table or form, Access saves any filter currently defined in that object, but won't apply the filter automatically the next time the object is opened (although it will automatically apply any sort you applied to the object before it was saved). If you want to apply a filter automatically when a form is first opened, specify a macro containing the ApplyFilter action or an event procedure containing the ApplyFilter method of the DoCmd object as the OnOpen event property setting of the form. You can also apply a filter by using the OpenForm or OpenReport action, or their corresponding methods. To apply a filter automatically when a table is first opened, you can open the table by using a macro containing the OpenTable action, followed immediately by the ApplyFilter action.
**Example**

**Apply a filter by using a macro**

The following macro contains a set of actions, each of which filters the records for a Customer Phone List form. It shows the use of the ApplyFilter, ShowAllRecords, and GoToControl actions. It also shows the use of conditions to determine which toggle button in an option group has been selected on the form. Each action row is associated with a toggle button that selects the set of records starting with A, B, C, and so on, or all records. This macro should be attached to the **AfterUpdate** event of the CompanyNameFilter option group.

<table>
<thead>
<tr>
<th>Condition</th>
<th>Action</th>
<th>Arguments: Setting</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>[CompanyNameFilters]=1</td>
<td>ApplyFilter</td>
<td>Where Condition: [CompanyName] Like &quot;[AÀÁÂÃÄ]*&quot;</td>
<td>Filter for company names that start with A, À, Á, Â, Ã, or Ä.</td>
</tr>
<tr>
<td>[CompanyNameFilters]=2</td>
<td>ApplyFilter</td>
<td>Where Condition: [CompanyName] Like &quot;B*&quot;</td>
<td>Filter for company names that start with B.</td>
</tr>
<tr>
<td>[CompanyNameFilters]=3</td>
<td>ApplyFilter</td>
<td>Where Condition: [CompanyName] Like &quot;[CÇ]*&quot;</td>
<td>Filter for company names that start with C or Ç.</td>
</tr>
</tbody>
</table>

... Action rows for D through Y have the same format as A through C ...

| [CompanyNameFilters]=26 | ApplyFilter | Where Condition: [CompanyName] Like "" | Filter for company names that start with Z, |
Show all records.

If records are returned for the selected letter, move focus to the CompanyName control.
ShowAllRecords Action

You can use the ShowAllRecords action to remove any applied filter from the active table, query result set, or form, and display all records in the table or result set or all records in the form's underlying table or query.
Setting

The ShowAllRecords action doesn't have any arguments.
Remarks

You can use this action to ensure that all records (including any changed or new records) are displayed for a table, query result set, or form. This action causes a requery of the records for a form or subform.

You can also use this action to remove any filter that was applied with the ApplyFilter action, the Apply Filter/Sort command on the Records menu, or the Filter Name or Where Condition argument of the OpenForm action.

This action has the same effect as clicking Remove Filter/Sort on the Records menu or clicking Remove Filter on the toolbar in Form view or Datasheet view.

To run the ShowAllRecords action in Visual Basic, use the ShowAllRecords method of the DoCmd object.
Example

Apply a filter by using a macro

The following macro contains a set of actions, each of which filters the records for a Customer Phone List form. It shows the use of the ApplyFilter, ShowAllRecords, and GoToControl actions. It also shows the use of conditions to determine which toggle button in an option group has been selected on the form. Each action row is associated with a toggle button that selects the set of records starting with A, B, C, and so on, or all records. This macro should be attached to the AfterUpdate event of the CompanyNameFilter option group.

<table>
<thead>
<tr>
<th>Condition</th>
<th>Action</th>
<th>Arguments: Setting</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>[CompanyName Filters] =1</td>
<td>ApplyFilter</td>
<td>Where [CompanyName Name] Like &quot;A, Å, Â, Ã, Ä, or Å. [AAÅÄÄÄ]&quot;</td>
<td>Filter for company names that start with A, Å, Â, Ã, Ä, or Å.</td>
</tr>
<tr>
<td>[CompanyName Filters] =2</td>
<td>ApplyFilter</td>
<td>Where [CompanyName Name] Like &quot;B*&quot;</td>
<td>Filter for company names that start with B.</td>
</tr>
<tr>
<td>[CompanyName Filters] =3</td>
<td>ApplyFilter</td>
<td>Where [CompanyName Name] Like &quot;CÇ*&quot;</td>
<td>Filter for company names that start with C or Ç.</td>
</tr>
</tbody>
</table>

... Action rows for D through Y have the same format as A through C ...

Where
<table>
<thead>
<tr>
<th>[Company Name Filters] =26</th>
<th>ApplyFilter</th>
<th>Condition: [Company Name] Like &quot;[ZÆØÅ]*&quot;</th>
<th>Filter for company names that start with Z, Æ, Ø, or Å.</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Company Name Filters] =27</td>
<td>ShowAllRecords</td>
<td></td>
<td>Show all records.</td>
</tr>
<tr>
<td>[RecordsetClone]. [RecordCount]&gt;0</td>
<td>GoToControl</td>
<td>Control Name: CompanyName</td>
<td>If records are returned for the selected letter, move focus to the CompanyName control.</td>
</tr>
</tbody>
</table>
Close Action

You can use the Close action to close either a specified Microsoft Access window or the active window if none is specified.
### Setting

The Close action has the following arguments.

<table>
<thead>
<tr>
<th>Action argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object Type</td>
<td>The type of object whose window you want to close. Click Table, Query, Form, Report, Macro, Module, Data Access Page, Server View, Diagram, Stored Procedure, or Function in the Object Type box in the Action Arguments section of the Macro window. To select the active window, leave this argument blank.</td>
</tr>
<tr>
<td>Object Name</td>
<td>The name of the object to be closed. The Object Name box shows all objects in the database of the type selected by the Object Type argument. Click the object to close. If you leave the Object Type argument blank, leave this argument blank also.</td>
</tr>
<tr>
<td>Save</td>
<td>Whether to save changes to the object when it's closed. Click Yes (save the object), No (close the object without saving it), or Prompt (prompt the user whether or not to save the object). The default is Prompt.</td>
</tr>
</tbody>
</table>

**Note** If you are closing a module in the Visual Basic Editor, you must use Module in the Object Type argument.
Remarks

The Close action works on all database objects that the user can explicitly open or close. This action has the same effect as selecting an object and then closing it by clicking Close on the File menu, clicking Close on the Control menu for the object's window, or clicking the Close button ❌ for the object.

If the Save argument is set to Prompt and the object hasn't already been saved before the Close action is carried out, a dialog box prompts the user to save the object before the macro closes it. If you've set the Warnings On argument of the SetWarnings action to No, the dialog box isn't displayed and the object is automatically saved.

To run the Close action in Microsoft Visual Basic, use the Close method of the DoCmd object.
Table Field

Table fields are separate pieces of information that make up a record within a table. You can control the appearance of data, specify default values, and speed up searching and sorting by setting field properties in the Field Properties section of *table Design view*.

Microsoft Access uses field properties when you view or edit data. For example, the **Format**, **InputMask**, and **Caption** properties that you set affect the appearance of table and query databases. The controls on new forms and reports that are based on the table inherit these properties by default. You can use other properties to set rules for data or to require data entry in your fields, which Microsoft Access enforces whenever you add or edit data in a table.

To open a table in Design view, go to the [Database window](#), click the **Tables** tab, select the table you want to open, and then click **Design**.
DisplayControl Property

You can use the DisplayControl property in table Design view to specify the default control you want to use for displaying a field.
Setting

You can set the **DisplayControl** property in the table's property sheet in table Design view by clicking the **Lookup** tab in the Field Properties section.

This property contains a drop-down list of the available controls for the selected field. For fields with a **Text** or **Number** data type, this property can be set to Text Box, List Box, or Combo Box. For fields with a **Yes/No** data type, this property can be set to Check Box, Text Box, or Combo Box.
Remarks

When you select a control for this property, any additional properties needed to configure the control are also displayed on the Lookup tab.

Tip

You can let Microsoft Access set the DisplayControl property and any related properties for you when you select the Lookup Wizard as the data type for a field.

Setting this property and any related control type properties will affect the field display in both Datasheet view and Form view. The field is displayed by using the control and control property settings set in table Design view. If a field had its DisplayControl property set in table Design view and you drag it from the field list in form Design view, Microsoft Access copies the appropriate properties to the control's property sheet.
**FieldName Property**

You can use the **FieldName** property to specify the name of a field within a table. For example, you can specify "Last Name" for a field that stores employees' last names within the Employees table.
Setting

Enter a field name, following Microsoft Access object naming rules. The name can't duplicate any other field name in the table.

**Note** Avoid specifying a name for a field that could cause a conflict with built-in Microsoft Access function or property names, such as the *Name* property.

You can set this property in the upper portion of table Design view or by using Visual Basic.

In Visual Basic, use the ADO *Name* property to read and set a table's field name.
Remarks

Microsoft Access identifies a field by its field name. Once you have specified a field name in table Design view, you can use that name in expressions, Visual Basic procedures, and SQL statements.
Check Box Control (Forms and Reports)

A check box on a form or report is a stand-alone control that display a Yes/No value from an underlying record source.

**Note** This control should not be confused with the Dynamic HTML check box control used on a data access page. For information about a check box control on a data access page, see Check Box Control (Data Access Pages).
Remarks

When you select or clear a check box that's bound to a Yes/No field, Microsoft Access displays the value in the underlying table according to the field's Format property (Yes/No, True/False, or On/Off).

You can also use check boxes in an option group to display values to choose from.

It's also possible to use an unbound check box in a custom dialog box to accept user input.
Combo Box Control (Forms)

The combo box control combines the features of a text box and a list box. Use a combo box when you want the option of either typing a value or selecting a value from a predefined list.

**Note** This control should not be confused with the Dynamic HTML drop-down list box control used on a data access page. For information about a drop-down list box control on a data access page, see [Drop-down List Box Control (Data Access Page)](#).
Remarks

In **Form view**, Microsoft Access doesn't display the list until you click the combo box's arrow.

If you have Control Wizards on before you select the combo box tool, you can create a combo box with a wizard. To turn Control Wizards on or off, click the Control Wizards button in the toolbox.

The setting of the **LimitToList** property determines whether you can enter values that aren't in the list.

The list can be single- or multiple-column, and the columns can appear with or without headings.
**Command Button Control (Forms)**

A command button on a form can start an action or a set of actions. For example, you could create a command button that opens another form. To make a command button do something, you write a [macro](#) or [event procedure](#) and attach it to the button's [OnClick](#) property.

**Note** This control should not be confused with the Dynamic HTML command button control used on a data access page. For information about a command button control on a data access page, see [Command Button Control](#) (Data Access Pages).
Remarks

You can display text on a command button by setting its **Caption** property, or you can display a picture by setting its **Picture** property.

Tip

You can create over 30 different types of command buttons with the Command Button Wizard. When you use the Command Button Wizard, Microsoft Access creates the button and the event procedure for you.
ActiveX Control (Form)

Some of the content in this topic may not be applicable to some languages.

In addition to the built-in controls that appear in the toolbox, Microsoft Access supports ActiveX controls (formerly known as custom or OLE controls). An ActiveX control, like a built-in control, is an object that you place on a form to enable or enhance a user's interaction with an application. ActiveX controls have events and can be incorporated into other controls. These controls have an .ocx file name extension. The Calendar control is an example of an ActiveX control.

Control: 

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>29</td>
<td>30</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
</tr>
<tr>
<td>20</td>
<td>21</td>
<td>22</td>
<td>23</td>
<td>24</td>
<td>25</td>
<td>26</td>
</tr>
<tr>
<td>27</td>
<td>28</td>
<td>29</td>
<td>30</td>
<td>31</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>

Tool:

Note This control should not be confused with the Dynamic HTML ActiveX control used on a data access page. For information about a ActiveX control on a data access page, see ActiveX Control (Data Access Pages).
Label Control (Forms and Reports)

Labels on a form or report display descriptive text such as titles, captions, or brief instructions. Labels have certain characteristics:

- Labels don't display values from fields or expressions.
- Labels are always unbound.
- Labels don't change as you move from record to record.

Note This control should not be confused with the Dynamic HTML label control used on a data access page. For information about a label control on a data access page, see Label Control (Data Access Pages).
Remarks

A label can be attached to another control. When you create a text box, for example, it has an attached label that displays a caption for that text box. This label appears as a column heading in the Datasheet view of a form.

When you create a label by using the Label tool, the label stands on its own— it isn't attached to any other control. You use stand-alone labels for information such as the title of a form or report, or for other descriptive text. Stand-alone labels don't appear in Datasheet view.
List Box Control (Forms)

The list box control displays a list of values or alternatives. In many cases, it's quicker and easier to select a value from a list than to remember a value to type. A list of choices also helps ensure that the value that's entered in a field is correct.

The list in a list box consists of rows of data. Rows can have one or more columns, which can appear with or without headings.

Note This control should not be confused with the Dynamic HTML list box control used on a data access page. For information about a list box control on a data access page, see List Box Control (Data Access Pages).
Remarks

If a multiple-column list box is bound, Microsoft Access stores the values from one of the columns.

You can use an unbound list box to store a value that you can use with another control. For example, you could use an unbound list box to limit the values in another list box or in a custom dialog box. You could also use an unbound list box to find a record based on the value you select in the list box.

If you don't have room on your form to display a list box, or if you want to be able to type new values as well as select values from a list, use a combo box instead of a list box.
Option Button Control (Forms and Reports)

An option button on a form or report is a stand-alone control used to display a Yes/No value from an underlying record source.

**Note** This control should not be confused with the Dynamic HTML option button control used on a data access page. For information about a option button control on a data access page, see Option Button Control (Data Access Pages).
Remarks

When you select or clear an option button that's bound to a Yes/No field, Microsoft Access displays the value in the underlying table according to the field's **Format** property (Yes/No, **True/False**, or On/Off).

You can also use option buttons in an **option group** to display values to choose from.

It's also possible to use an unbound option button in a **custom dialog box** to accept user input.
Option Group Control (Forms and Reports)

An option group on a form or report displays a limited set of alternatives. An option group makes selecting a value easy since you can just click the value you want. Only one option in an option group can be selected at a time.

An option group consists of a group frame and a set of check boxes, toggle buttons, or option buttons.

**Note** This control should not be confused with the Dynamic HTML option group control used on a data access page. For information about a option group control on a data access page, see [Option Group Control (Data Access Pages)](#).
Remarks

If an option group is bound to a field, only the group frame itself is bound to the field, not the check boxes, toggle buttons, or option buttons inside the frame. Instead of setting the ControlSource property for each control in the option group, you set the OptionValue property of each check box, toggle button, or option button to a number that's meaningful for the field to which the group frame is bound. When you select an option in an option group, Microsoft Access sets the value of the field to which the option group is bound to the value of the selected option's OptionValue property.

Note  The OptionValue property is set to a number because the value of an option group can only be a number, not text. Microsoft Access stores this number in the underlying table. In the preceding example, if you want to display the name of the shipper instead of a number in the Orders table, you can create a separate table called Shippers that stores shipper names, and then make the ShipVia field in the Orders table a Lookup field that looks up data in the Shippers table.

An option group can also be set to an expression, or it can be unbound. You can use an unbound option group in a custom dialog box to accept user input and then carry out an action based on that input.
Subform/Subreport Control

The subform/subreport control embeds a form in a form or a report in a report.

For example, you can use a form with a subform to present one-to-many relationships, such as one product category with the items that fall into that category. In this case, the main form can display the category ID, name, and description; the subform can display the available products in that category.

Tip

Instead of creating the main form, and then adding the subform control to it, you can simultaneously create the main form and subform with a wizard. You can also create a subform or subreport by dragging an existing form or report from the Database window to the main form or report.
Tab Control

A tab control contains multiple pages on which you can place other controls, such as text boxes or option buttons. When a user clicks the corresponding tab, that page becomes active.

With the tab control, you can construct a single form or dialog box that contains several different tabs, and you can group similar options or data on each tab's page. For example, you might use a tab control on an Employees form to separate general and personal information.
Text Box Control (Forms and Reports)

Text boxes on a form or report display data from a record source. This type of text box is called a bound text box because it's bound to data in a field. Text boxes can also be unbound. For example, you can create an unbound text box to display the results of a calculation, or to accept input from a user. Data in an unbound text box isn't saved with the database.

**Note** This control should not be confused with the Dynamic HTML text box control used on a data access page. For information about a text box control on a data access page, see Text Box Control (Data Access Pages).
Toggle Button Control

A toggle button on a form is a stand-alone control used to display a Yes/No value from an underlying record source.
Remarks

When you click a toggle button that's bound to a Yes/No field, Microsoft Access displays the value in the underlying table according to the field's **Format** property (Yes/No, **True/False**, or On/Off).

Toggle buttons are most useful when used in an **option group** with other buttons.

You can also use a toggle button in a **custom dialog box** to accept user input.
DatasheetGridlinesBehavior Property

You can use the `DatasheetGridlinesBehavior` property to specify which gridlines will appear in Datasheet view. Read/write Byte.

`expression.DatasheetGridlinesBehavior`

`expression` Required. An expression that returns one of the objects in the Applies To list.
Remarks

This **DatasheetGridlinesBehavior** property applies only to objects in Datasheet view.


The **DatasheetGridlinesBehavior** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>acGridlinesNone</td>
<td>No gridlines are displayed.</td>
</tr>
<tr>
<td>Horizontal</td>
<td>acGridlinesHoriz</td>
<td>Only horizontal gridlines are displayed.</td>
</tr>
<tr>
<td>Vertical</td>
<td>acGridlinesVert</td>
<td>Only vertical gridlines are displayed.</td>
</tr>
<tr>
<td>Both</td>
<td>acGridlinesBoth</td>
<td>(Default) Horizontal and vertical gridlines are displayed.</td>
</tr>
</tbody>
</table>

**Note** You can set this property by using the **Gridlines** button on the **Formatting (Datasheet)** [toolbar](https://developer.microsoft.com/en-us/visualbasic), and in an Access database (.mdb), by using a [macro](https://msdn.microsoft.com/en-us/library/office/jj118206.aspx), or by using Visual Basic.

The following setting information applies to both Access databases (.mdb) and Access projects (.adp):

You can also set this property by selecting the settings displayed under **Gridlines Shown** in the **Cells Effects** dialog box, available by clicking **Cells** on the **Format** menu.

You can set the default **DatasheetGridlinesBehavior** property by using the settings under **Default Gridlines Showing** on the **Datasheet** tab of the **Options** dialog box, available by clicking **Options** on the **Tools** menu.

Changes to this property will be visible only if the **DatasheetCellsEffect** property is set to Flat.

The following table contains the properties that don't exist in the DAO **Properties** collection of until you set them by using the **Formatting**
(Datasheet) toolbar or you can add them in an Access database (.mdb) by using the CreateProperty method and append it to the DAO Properties collection.

DatasheetFontItalic *  DatasheetForeColor *
DatasheetFontHeight *  DatasheetBackColor
DatasheetFontName *  DatasheetGridlinesColor
DatasheetFontUnderline *  DatasheetGridlinesBehavior
DatasheetFontWeight *  DatasheetCellsEffect

Note  When you add or set any property listed with an asterisk, Microsoft Access automatically adds all the properties listed with an asterisk to the Properties collection in the database.
DatasheetGridlinesColor Property

You can use the **DatasheetGridlinesColor** property to specify the color of gridlines in a datasheet. Read/write **Long**.

`expression.DatasheetGridlinesColor`

`expression`  Required. An expression that returns one of the objects in the Applies To list.
Remarks

The `DatasheetGridlinesColor` property applies only to objects in `Datasheet view`.

This property is only available in `Visual Basic` within a `Microsoft Access database` (.mdb).

The `DatasheetGridlinesColor` property setting is a `Long Integer` value. In Visual Basic, you can also use the `RGB` or `QBColor` functions to set this property.

You can set this property by using `Line Color` on the `Formatting (Datasheet)` toolbar, in an Access database (.mdb), you can add the property by using a `macro`, or by using Visual Basic.

The following setting information applies to both Access databases (.mdb) and `Access projects` (.adp):

You can also set this property by clicking `Cells` on the `Format` menu and displaying the `Cells Effects` dialog box. You can then select an available color from the drop-down list under `Gridline color`.

You can set the default `DatasheetGridlinesColor` property by using the settings under `Default Colors` on the `Datasheet` tab of the `Options` dialog box, available by clicking `Options` on the `Tools` menu.

This property setting affects the gridline color for the entire datasheet. It's not possible to set the gridline color of individual cells in Datasheet view.

The following table contains the properties that don't exist in the DAO Properties collection of until you set them by using the `Formatting (Datasheet)` toolbar or you can add them in an Access database (.mdb) by using the `CreateProperty` method and append it to the DAO Properties collection.

- `DatasheetFontItalic` *
- `DatasheetFontHeight` *
- `DatasheetFontName` *
- `DatasheetForeColor` *
- `DatasheetBackColor`
- `DatasheetFontColor` *
- `DatasheetGridlinesColor`
DatasheetFontUnderline * DatasheetGridlinesBehavior
DatasheetFontWeight * DatasheetCellsEffect

**Note**  When you add or set any property listed with an asterisk, Microsoft Access automatically adds all the properties listed with an asterisk to the Properties collection in the database.
FieldSize Property

You can use the **FieldSize** property to set the maximum size for data stored in a field set to the **Text**, **Number**, or **AutoNumber** data type.
Setting

If the **DataType** property is set to Text, enter a number from 0 to 255. The default setting is 50.

If the **DataType** property is set to AutoNumber, the **FieldSize** property can be set to Long Integer or Replication ID.

If the **DataType** property is set to Number, the **FieldSize** property settings and their values are related in the following way.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
<th>Decimal precision</th>
<th>Storage size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Byte</td>
<td>Stores numbers from 0 to 255 (no fractions).</td>
<td>None</td>
<td>1 byte</td>
</tr>
<tr>
<td></td>
<td>Stores numbers from $-10^{38}$–1 through $10^{38}$–1 (.adp)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Stores numbers from $-10^{28}$–1 through $10^{28}$–1 (.mdb)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Decimal</td>
<td>Stores numbers from $-32,768$ to $32,767$ (no fractions).</td>
<td>28</td>
<td>12 bytes</td>
</tr>
<tr>
<td>Integer</td>
<td>(Default) Stores numbers from $-2,147,483,648$ to $2,147,483,647$ (no fractions).</td>
<td>None</td>
<td>2 bytes</td>
</tr>
<tr>
<td></td>
<td>Stores numbers from $-3.402823E38$ to $-1.401298E-45$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Long Integer</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Single</td>
<td>for negative values and from $1.401298E-45$ to $3.402823E38$ for positive values.</td>
<td>7</td>
<td>4 bytes</td>
</tr>
<tr>
<td></td>
<td>Stores numbers from $-1.79769313486231E308$ to $-4.94065645841247E-324$ for negative values and from $4.94065645841247E-324$ to $1.79769313486231E308$ for positive values.</td>
<td>15</td>
<td>8 bytes</td>
</tr>
<tr>
<td>Replication ID</td>
<td><strong>Globally unique identifier (GUID)</strong></td>
<td>N/A</td>
<td>16 bytes</td>
</tr>
</tbody>
</table>
You can set this property only from the table's property sheet.

To set the size of a field from Visual Basic, use the DAO Size property to read and set the maximum size of Text fields (for data types other than Text, the ADO Type property setting automatically determines the Size property setting).

**Note**  You can specify the default field sizes for Text and Number fields by changing the values under Default Field Sizes on the Tables/Queries tab, available by clicking Options on the Tools menu.
Remarks

You should use the smallest possible FieldSize property setting because smaller data sizes can be processed faster and require less memory.

Caution  If you convert a large FieldSize setting to a smaller one in a field that already contains data, you might lose data. For example, if you change the FieldSize setting for a Text data type field from 255 to 50, data beyond the new 50-character setting will be discarded.

If the data in a Number data type field doesn't fit in a new FieldSize setting, fractional numbers may be rounded or you might get a Null value. For example, if you change from a Single to an Integer field size, fractional values will be rounded to the nearest whole number and values greater than 32,767 or less than −32,768 will result in null fields.

You can't undo changes to data that result from a change to the FieldSize property after saving those changes in table Design view.

Tip

You can use the Currency data type if you plan to perform many calculations on a field that contains data with one to four decimal places. Single and Double data type fields require floating-point calculation. Currency data type fields use a faster fixed-point calculation.
Format Event

The Format event occurs when Microsoft Access determines which data belongs in a report section, but before Microsoft Access formats the section for previewing or printing.

Private Sub sectionname_Format(Cancel As Integer, FormatCount As Integer)

SectionName    The name of a Report section.

Cancel    The setting determines if the formatting of the section occurs. Setting the Cancel argument to True (–1) cancels formatting of the section.

FormatCount    An Integer value that specifies whether the Format event has occurred more than once for a section. For example, if a section doesn't fit on one page and part of it moves to the next page of the report, Microsoft Access sets the FormatCount argument to 2.
Remarks

To run a macro or event procedure when this event occurs, set the OnFormat property to the name of the macro or to [Event Procedure].

A Format event occurs for each section in a report. This allows you to create complex running calculations by using data from each section, including sections that aren't printed.

For report detail sections, the Format event occurs for each record in the section just before Microsoft Access formats the data in the record. A Format macro or event procedure has access to the data in the current record.

For report group headers, the Format event occurs for each new group, and a Format macro or event procedure has access to the data in the group header and the data in the first record in the detail section. For report group footers, the Format event occurs for each new group, and a Format macro or event procedure has access to the data in the group footer and the data in the last record in the detail section.

By running a macro or an event procedure when the Format event occurs, you can use data in the current record to make changes to the report that affect page layout. For example, you can display or hide a congratulatory message next to a salesperson's monthly sales total in a sales report, depending on the sales total. After the control is displayed or hidden, Microsoft Access formats the section by using the values of format properties, such as CanGrow, CanShrink, HideDuplicates, KeepTogether, and Visible.

For changes that don't affect page layout or for event procedures or macros that should run only after the data on a page has been formatted, such as a macro that prints page totals, use the Print event for the report section.

There are times when Microsoft Access must return to previous sections on a report to perform multiple formatting passes. When this happens, the Retreat event occurs as the report returns to each previous section, and the Format event occurs more than once for each section. You can run a macro or event procedure when the Retreat event occurs to undo any changes that you made when the Format event occurred for the section. This is useful when your Format macro or
event procedure carries out actions, such as calculating page totals or controlling the size of a section, that you want to perform only once for each section.
Macro

You can use the FormatCount property to check whether the Format event has occurred more than once for a section. For example, if a section doesn't fit on one page and part of it moves to the next page of the report, the Format event occurs once for each page, and the FormatCount property is set to 2. You can check the FormatCount property setting to determine how many times a Format macro has run per section.

You can use the CancelEvent action in a Format macro to cancel formatting of a section. If you cancel formatting, Microsoft Access doesn't format the section for printing and prints the next section instead. You can use the CancelEvent action to skip a section in a report without leaving a blank space on the page when the report is printed.
Example

The following example displays or hides a congratulatory message next to a calculated control that shows sales totals by salesperson. In report sections where the sales total is greater than the sales goal, a label named Message displays the message "Congratulations! You have met your sales goal" when the section is printed; in sections where the sales total is less than the sales goal, the label is hidden.

To try the example, add the following event procedure to a report that contains a label called Message, a text box called GrandTotal (a calculated control that displays the sales total), and a detail section called SalesDetail.

```vba
Private Sub SalesDetail_Format(Cancel As Integer, _
    FormatCount As Integer)
    Const conSalesGoal = 1000
    If Me!GrandTotal > conSalesGoal Then
        Me!Message.Visible = True
    Else
        Me!Message.Visible = False
    End If
End Sub
```
Retreat Event

The Retreat event occurs when Microsoft Access returns to a previous report section during report formatting.

Private Sub object_Retreat()

Object The name of a Report section.
Remarks

The Retreat event applies to all report sections except page headers and footers.

To run a macro or event procedure when this event occurs, set the OnRetreat property to the name of the macro or to [Event Procedure].

Under certain circumstances, Microsoft Access must return to a previous report section to determine where certain controls and sections are on a report and whether they will fit in a given space. Examples include:

- Groups (except for page headers and footers) for which the KeepTogether property is set to either Whole Group or With First Detail.
- Subreports or subforms for which the CanGrow and/or CanShrink property is set to Yes.
- Sections on the last page of a report.

In these situations, the Format event occurs as Microsoft Access determines how the sections will fit on the printed page (however, a Print event doesn't occur because the sections aren't being printed yet). If the sections can't be printed, Microsoft Access backs up to the necessary location so that the sections can actually be printed on the following page. The Retreat event occurs for each section past which Microsoft Access retreats. The Format event for each section then occurs again as Microsoft Access prepares to actually print the sections.

For example, whenever Microsoft Access reaches the end of the last page while formatting a report, the Retreat event occurs for each previous section until Microsoft Access reaches the first section at the top of the last page. Then the Format event occurs again for each section on the page, followed by the Print events.

You can run an event procedure or macro when the Retreat event occurs to undo any changes that you made when the Format event occurred for the section. This is useful when your Format event procedure or macro carries out actions, such as calculating page totals or controlling the size of a section, that you want to perform only once for each section.

The Retreat event is also useful for maintaining the positioning of report items.
For example, the Sales By Year report in the Northwind sample database uses this event to determine if a page header is printed or not (the page header is printed on pages following a page where the group header has been printed, and the page header isn't printed on the page following a page where the group footer has been printed).
Format Property - Date/Time Data Type

You can set the **Format** property to predefined date and time formats or use custom formats for the Date/Time data type.
## Setting

### Predefined Formats

The following table shows the predefined **Format** property settings for the Date/Time data type.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>General Date</td>
<td><em>(Default)</em> If the value is a date only, no time is displayed; if the value is a time only, no date is displayed. This setting is a combination of the Short Date and Long Time settings. Examples: 4/3/93, 05:34:00 PM, and 4/3/93 05:34:00 PM.</td>
</tr>
<tr>
<td>Long Date</td>
<td>Same as the Long Date setting in the regional settings of Windows. Example: Saturday, April 3, 1993.</td>
</tr>
<tr>
<td>Medium Date</td>
<td>Example: 3-Apr-93. Same as the Short Date setting in the regional settings of Windows. Example: 4/3/93.</td>
</tr>
<tr>
<td>Short Date</td>
<td><strong>Warning</strong> The Short Date setting assumes that dates between 1/1/00 and 12/31/29 are twenty-first century dates (that is, the years are assumed to be 2000 to 2029). Dates between 1/1/30 and 12/31/99 are assumed to be twentieth century dates (that is, the years are assumed to be 1930 to 1999). Same as the setting on the <strong>Time</strong> tab in the regional settings of Windows. Example: 5:34:23 PM.</td>
</tr>
<tr>
<td>Long Time</td>
<td>Example: 5:34 PM.</td>
</tr>
<tr>
<td>Medium Time</td>
<td></td>
</tr>
<tr>
<td>Short Time</td>
<td>Example: 17:34.</td>
</tr>
</tbody>
</table>
# Custom Formats

You can create custom date and time formats by using the following symbols.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>:</td>
<td>Time separator. Separators are set in the regional settings of Windows. (colon)</td>
</tr>
<tr>
<td>/</td>
<td>Date separator.</td>
</tr>
<tr>
<td>c</td>
<td>Same as the General Date predefined format.</td>
</tr>
<tr>
<td>d</td>
<td>Day of the month in one or two numeric digits, as needed (1 to 31).</td>
</tr>
<tr>
<td>dd</td>
<td>Day of the month in two numeric digits (01 to 31).</td>
</tr>
<tr>
<td>dddd</td>
<td>First three letters of the weekday (Sun to Sat).</td>
</tr>
<tr>
<td>ddddd</td>
<td>Full name of the weekday (Sunday to Saturday).</td>
</tr>
<tr>
<td>dddddd</td>
<td>Same as the Short Date predefined format.</td>
</tr>
<tr>
<td>ddddddd</td>
<td>Same as the Long Date predefined format.</td>
</tr>
<tr>
<td>w</td>
<td>Day of the week (1 to 7).</td>
</tr>
<tr>
<td>ww</td>
<td>Week of the year (1 to 53).</td>
</tr>
<tr>
<td>m</td>
<td>Month of the year in one or two numeric digits, as needed (1 to 12).</td>
</tr>
<tr>
<td>mm</td>
<td>Month of the year in two numeric digits (01 to 12).</td>
</tr>
<tr>
<td>mmm</td>
<td>First three letters of the month (Jan to Dec).</td>
</tr>
<tr>
<td>mmmm</td>
<td>Full name of the month (January to December).</td>
</tr>
<tr>
<td>q</td>
<td>Date displayed as the quarter of the year (1 to 4).</td>
</tr>
<tr>
<td>y</td>
<td>Number of the day of the year (1 to 366).</td>
</tr>
<tr>
<td>yy</td>
<td>Last two digits of the year (01 to 99).</td>
</tr>
<tr>
<td>yyyy</td>
<td>Full year (0100 to 9999).</td>
</tr>
<tr>
<td>h</td>
<td>Hour in one or two digits, as needed (0 to 23).</td>
</tr>
<tr>
<td>hh</td>
<td>Hour in two digits (00 to 23).</td>
</tr>
<tr>
<td>n</td>
<td>Minute in one or two digits, as needed (0 to 59).</td>
</tr>
<tr>
<td>nn</td>
<td>Minute in two digits (00 to 59).</td>
</tr>
<tr>
<td>s</td>
<td>Second in one or two digits, as needed (0 to 59).</td>
</tr>
<tr>
<td>ss</td>
<td>Second in two digits (00 to 59).</td>
</tr>
<tr>
<td>ttttt</td>
<td>Same as the Long Time predefined format.</td>
</tr>
</tbody>
</table>
AM/PM Twelve-hour clock with the uppercase letters "AM" or "PM", as appropriate.

am/pm Twelve-hour clock with the lowercase letters "am" or "pm", as appropriate.

A/P Twelve-hour clock with the uppercase letter "A" or "P", as appropriate.

a/p Twelve-hour clock with the lowercase letter "a" or "p", as appropriate.

AMPM Twelve-hour clock with the appropriate morning/afternoon designator as defined in the regional settings of Windows.

Custom formats are displayed according to the settings specified in the regional settings of Windows. Custom formats inconsistent with the settings specified in the regional settings of Windows are ignored.

Note If you want to add a comma or other separator to a custom format, enclose the separator in quotation marks as follows: mmm d", "yyyy.
Example

The following are examples of custom date/time formats.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Display</th>
</tr>
</thead>
<tbody>
<tr>
<td>ddd&quot;, &quot;mmm d&quot;, &quot;yyy</td>
<td>Mon, Jun 2, 1997</td>
</tr>
<tr>
<td>mmmm dd&quot;, &quot;yyy</td>
<td>June 02, 1997</td>
</tr>
<tr>
<td>&quot;This is week number &quot;ww This is week number 22</td>
<td></td>
</tr>
<tr>
<td>&quot;Today is &quot;dddd</td>
<td>Today is Tuesday</td>
</tr>
</tbody>
</table>

You could use a custom format to display "A.D." before or "B.C." after a year depending on whether a positive or negative number is entered. To see this custom format work, create a new table field, set its data type to Number, and enter a format as follows:

"A.D. " #;# " B.C."

Positive numbers are displayed as years with an "A.D." before the year. Negative numbers are displayed as years with a "B.C." after the year.
Format Property - Number and Currency Data Types

You can set the **Format** property to predefined number formats or custom number formats for the Number and Currency data types.
**Setting**

**Predefined Formats**

The following table shows the predefined **Format** property settings for numbers.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>General</td>
<td>(Default) Display the number as entered.</td>
</tr>
<tr>
<td>Number</td>
<td>Use the thousand separator; follow the settings specified in the regional</td>
</tr>
<tr>
<td></td>
<td>settings of Windows for negative amounts, decimal and currency symbols, and</td>
</tr>
<tr>
<td></td>
<td>decimal places.</td>
</tr>
<tr>
<td>Currency</td>
<td>Use the euro symbol (€), regardless of the currency symbol specified in the</td>
</tr>
<tr>
<td></td>
<td>regional settings of Windows.</td>
</tr>
<tr>
<td></td>
<td>Display at least one digit; follow the settings specified in the regional</td>
</tr>
<tr>
<td></td>
<td>settings of Windows for negative amounts, decimal and currency symbols, and</td>
</tr>
<tr>
<td></td>
<td>decimal places.</td>
</tr>
<tr>
<td>Fixed</td>
<td>Use the thousand separator; follow the settings specified in the regional</td>
</tr>
<tr>
<td></td>
<td>settings of Windows for negative amounts, decimal symbols, and decimal places.</td>
</tr>
<tr>
<td>Standard</td>
<td>Multiply the value by 100 and append a percent sign (%) ; follow the settings</td>
</tr>
<tr>
<td></td>
<td>specified in the regional settings of Windows for negative amounts, decimal</td>
</tr>
<tr>
<td></td>
<td>symbols, and decimal places.</td>
</tr>
<tr>
<td>Percent</td>
<td>Use standard scientific notation.</td>
</tr>
</tbody>
</table>
Custom Formats

Custom number formats can have one to four sections with semicolons (;) as the list separator. Each section contains the format specification for a different type of number.

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>First</td>
<td>The format for positive numbers.</td>
</tr>
<tr>
<td>Second</td>
<td>The format for negative numbers.</td>
</tr>
<tr>
<td>Third</td>
<td>The format for zero values.</td>
</tr>
<tr>
<td>Fourth</td>
<td>The format for Null values.</td>
</tr>
</tbody>
</table>

For example, you could use the following custom Currency format:

```
$#,##0.00[Green];($#,##0.00)[Red];"Zero";'Null'
```

This number format contains four sections separated by semicolons and uses a different format for each section.

If you use multiple sections but don't specify a format for each section, entries for which there is no format will either display nothing or default to the formatting of the first section.

You can create custom number formats by using the following symbols.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.</td>
<td>Decimal separator. Separators are set in the regional settings in Windows.</td>
</tr>
<tr>
<td>,</td>
<td>Thousand separator.</td>
</tr>
<tr>
<td>0</td>
<td>Digit placeholder. Display a digit or 0.</td>
</tr>
<tr>
<td>#</td>
<td>Digit placeholder. Display a digit or nothing.</td>
</tr>
</tbody>
</table>
| $      | Display the literal character "$". Percentage. The value is multiplied by 100 and a percent sign is
% appended.

Scientific notation with a minus sign (−) next to negative exponents E− or e− and nothing next to positive exponents. This symbol must be used with other symbols, as in 0.00E−00 or 0.00E00.

Scientific notation with a minus sign (−) next to negative exponents E+ or e+ and a plus sign (+) next to positive exponents. This symbol must be used with other symbols, as in 0.00E+00.
Remarks

You can use the **DecimalPlaces** property to override the default number of decimal places for the predefined format specified for the **Format** property.

The predefined currency and euro formats follow the settings in the regional settings of Windows. You can override these by entering your own currency format.
Example

The following are examples of the predefined number formats.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Data</th>
<th>Display</th>
</tr>
</thead>
<tbody>
<tr>
<td>General Number</td>
<td>3456.789</td>
<td>3456.789</td>
</tr>
<tr>
<td></td>
<td>–3456.789</td>
<td>–3456.789</td>
</tr>
<tr>
<td></td>
<td>$213.21</td>
<td>$213.21</td>
</tr>
<tr>
<td>Currency</td>
<td>3456.789</td>
<td>$3,456.79</td>
</tr>
<tr>
<td></td>
<td>–3456.789</td>
<td>($3,456.79)</td>
</tr>
<tr>
<td></td>
<td>3456.789</td>
<td>3456.79</td>
</tr>
<tr>
<td>Fixed</td>
<td>–3456.789</td>
<td>–3456.79</td>
</tr>
<tr>
<td></td>
<td>3.56645</td>
<td>3.57</td>
</tr>
<tr>
<td>Standard</td>
<td>3456.789</td>
<td>3,456.79</td>
</tr>
<tr>
<td>Percent</td>
<td>3</td>
<td>300%</td>
</tr>
<tr>
<td></td>
<td>0.45</td>
<td>45%</td>
</tr>
<tr>
<td>Scientific</td>
<td>3456.789</td>
<td>3.46E+03</td>
</tr>
<tr>
<td></td>
<td>–3456.789</td>
<td>–3.46E+03</td>
</tr>
</tbody>
</table>

The following are examples of custom number formats.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0; (0);&quot;Null&quot;</td>
<td>Display positive values normally; display negative values in parentheses; display the word &quot;Null&quot; if the value is Null.</td>
</tr>
<tr>
<td>+0.0;–</td>
<td>Display a plus (+) or minus (−) sign with positive or negative numbers; display 0.0 if the value is zero.</td>
</tr>
</tbody>
</table>
Format Property - Text and Memo Data Types

You can use special symbols in the setting for the **Format** property to create custom formats for Text and Memo fields.
Setting

You can create custom text and memo formats by using the following symbols.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@</td>
<td>Text character (either a character or a space) is required.</td>
</tr>
<tr>
<td>&amp;</td>
<td>Text character is not required.</td>
</tr>
<tr>
<td>&lt;</td>
<td>Force all characters to lowercase.</td>
</tr>
<tr>
<td>&gt;</td>
<td>Force all characters to uppercase.</td>
</tr>
</tbody>
</table>

Custom formats for Text and Memo fields can have up to two sections. Each section contains the format specification for different data in a field.

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>First</td>
<td>Format for fields with text.</td>
</tr>
<tr>
<td>Second</td>
<td>Format for fields with zero-length strings and Null values.</td>
</tr>
</tbody>
</table>

For example, if you have a text box control in which you want the word "None" to appear when there is no string in the field, you could type the custom format @;"None" as the control's Format property setting. The @ symbol causes the text from the field to be displayed; the second section causes the word "None" to appear when there is a zero-length string or Null value in the field.

**Note** You can use the Format function to return one value for a zero-length string and another for a Null value, and you can similarly use the Format property to automatically format fields in table Datasheet view or controls on a form or report.
Example

The following are examples of text and memo custom formats.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Data</th>
<th>Display</th>
</tr>
</thead>
<tbody>
<tr>
<td>@@@@-@@-@@@@ @@@@-@@-@@@@</td>
<td>465043799</td>
<td>465-04-3799</td>
</tr>
<tr>
<td>@@@@@@@@@@@@@</td>
<td>465043799</td>
<td>465043799</td>
</tr>
<tr>
<td>davolio</td>
<td>DAVOLIO</td>
<td>Davolio</td>
</tr>
<tr>
<td>&gt;</td>
<td>DAVOLIO</td>
<td>DAVOLIO</td>
</tr>
<tr>
<td>&lt;</td>
<td>DAVOLIO</td>
<td>davolio</td>
</tr>
<tr>
<td>@;&quot;Unknown&quot;</td>
<td>Null value</td>
<td>Unknown</td>
</tr>
<tr>
<td>Zero-length string</td>
<td>Unknown</td>
<td></td>
</tr>
<tr>
<td>Any text</td>
<td>Same text as entered is displayed</td>
<td></td>
</tr>
</tbody>
</table>
Format Property - Yes/No Data Type

You can set the **Format** property to the Yes/No, **True/False**, or On/Off predefined formats or to a custom format for the Yes/No data type.
Setting

Microsoft Access uses a check box control as the default control for the Yes/No data type. Predefined and custom formats are ignored when a check box control is used. Therefore, these formats apply only to data that is displayed in a text box control.
Predefined Formats

Yes, **True**, and On are equivalent, as are No, **False**, and Off. If you specify one predefined format and then enter an equivalent value, the predefined format of equivalent value will be displayed. For example, if you enter **True** or On in a text box control with its **Format** property set to Yes/No, the value is automatically converted to Yes.
Custom Formats

The Yes/No data type can use custom formats containing up to three sections.

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>First</td>
<td>This section has no effect on the Yes/No data type. However, a semicolon (;) is required as a placeholder.</td>
</tr>
<tr>
<td>Second</td>
<td>The text to display in place of Yes, <strong>True</strong>, or On values.</td>
</tr>
<tr>
<td>Third</td>
<td>The text to display in place of No, <strong>False</strong>, or Off values.</td>
</tr>
</tbody>
</table>
Example

The following example shows a custom yes/no format for a text box control. The control displays the word "Always" in blue text for Yes, True, or On, and the word "Never" in red text for No, False, or Off.

;"Always"[Blue];"Never"[Red]
KeepTogether Property - Sections

You can use the **KeepTogether** property for a section to print a form or report section all on one page. For example, you might have a group of related information that you don't want printed across two pages.

**Note** The **KeepTogether** property applies only to form and report sections (except page headers and page footers).
### Setting

The **KeepTogether** property for a section uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>True</td>
<td>Microsoft Access starts printing the section at the top of the next page if it can't print the entire section on the current page.</td>
</tr>
<tr>
<td>No</td>
<td>False</td>
<td>(Default) Microsoft Access prints as much of the section as possible on the current page and prints the rest on the next page.</td>
</tr>
</tbody>
</table>

You can set this property by using the section's property sheet, a macro, or Visual Basic.

The **KeepTogether** property for a section can only be set in form Design view or report Design view.
Remarks

Usually, when a page break occurs while a section is being printed, Microsoft Access continues printing the section on the next page. By using the section's **KeepTogether** property, you can print the section all on one page. If a section is longer than one page, Microsoft Access starts printing it on the next page and continues on the following page.

If the **KeepTogether** property for a group is set to Whole Group or With First Detail and the **KeepTogether** property for a section is set to No, the **KeepTogether** property setting for the section is ignored.
Example

The following example returns the `KeepTogether` property setting for a report's detail section and assigns the value to the variable `intGetVal`.

```vba
Dim intGetVal As Integer
intGetVal = Me.Section(acDetail).KeepTogether
```
Learn about language-specific information

Language-specific Help topics apply only if the language-specific feature is available. Learn about working in another language or installing the proofing tools for another language, or see your system administrator for more information.
Show All
SetValue Action

You can use the SetValue action to set the value of a Microsoft Access field, control, or property on a form, a form datasheet, or a report.

Note  You cannot use the SetValue action to set the value of an Access property that returns an object.
Setting

The SetValue action has the following arguments.

<table>
<thead>
<tr>
<th>Action argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Item</td>
<td>The name of the field, control, or property whose value you want to set. Enter the field, control, or property name in the Item box in the Action Arguments section of the Macro window. You must use the full syntax to refer to this item, such as controlname (for a control on the form or report from which the macro was called) or Forms!formname!controlname. This is a required argument. The expression Access uses to set the value for this item. You must always use the full syntax to refer to any objects in the expression. For example, to increase the value in a Salary control on an Employees form by 10 percent, use Forms!Employees!Salary*1.1. This is a required argument.</td>
</tr>
<tr>
<td>Expression</td>
<td>Note You shouldn't use an equal sign (=) before the expression in this argument. If you do, Access evaluates the expression and then uses this value as the expression in this argument. This can produce unexpected results if the expression is a string. For example, if you type =&quot;String1&quot; for this argument, Access first evaluates the expression as String1. Then it uses String1 as the expression in this argument, expecting to find a control or property named String1 on the form or report that called the macro.</td>
</tr>
</tbody>
</table>

Note In an Access database (.mdb), click the Build button to the right of the Item or Expression box to use the Expression Builder to create an expression for either of these arguments.
Remarks

You can use this action to set a value for a field or control on a form, a form datasheet, or a report. You can also set the value for almost all control, form, and report properties in any view. To find out whether a particular property can be set by using a macro and which views it can be set in, see the Help topic for that property.

You can also set the value for a field in a form's underlying table even if the form doesn't contain a control bound to the field. Use the syntax `Forms!formname!fieldname` in the Item box to set the value for such a field. You can also refer to a field in a report's underlying table by using the syntax `Reports!reportname!fieldname`, but there must be a control on the report bound to this field, or the field must be referred to in a calculated control on the report.

If you set the value of a control on a form, the SetValue action doesn't trigger the control's form-level validation rules, but it does trigger the underlying field's table-level validation rules if the control is a bound control. The SetValue action also triggers recalculation, but the recalculation may not happen immediately. To trigger immediate repainting and force the recalculation to completion, use the RepaintObject action. The value you set in a control by using the SetValue action is also not affected by an input mask set in the control's or underlying field's InputMask property.

To change the value of a control, you can use the SetValue action in a macro specified by the control's AfterUpdate event property. However, you can't use the SetValue action in a macro specified by a control's BeforeUpdate event property to change the value of the control (although you can use SetValue to change the value of other controls). You can also use the SetValue action in a macro specified by the BeforeUpdate or AfterUpdate property of a form to change the value of any controls in the current record.

Note  You can't use the SetValue action to set the value of the following controls:

- Bound controls and calculated controls on reports.
- Calculated controls on forms.

Tip
You can use the SetValue action to hide or show a form in Form view. Enter **Forms!formname.Visible** in the **Item** box and **No** or **Yes** in the **Expression** box. Setting a **modal** form's **Visible** property to **No** hides the form and makes it modeless. Setting the property to **Yes** displays the form and makes it modal again.

Changing the value of or adding new data in a control by using the SetValue action in a macro doesn't trigger events such as **BeforeUpdate**, **BeforeInsert**, or **Change** that occur when you change or enter data in these controls in the user interface. These events also don't occur if you set the value of the control by using Visual Basic.

This action isn't available in Microsoft Visual Basic. Set the value directly in Visual Basic.
# Example

Set the value of a control by using a macro

The following macro opens the Add Products form from a button on the Suppliers form. It shows the use of the Echo, Close, OpenForm, SetValue, and GoToControl actions. The SetValue action sets the SupplierID control on the Products form to the current supplier on the Suppliers form. Then the GoToControl action moves the focus to the CategoryID field, where you can begin to enter data for the new product. This macro should be attached to the Add Products button on the Suppliers form.

<table>
<thead>
<tr>
<th>Action</th>
<th>Arguments: Setting</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Echo</td>
<td>Echo On: No</td>
<td>Stop screen updating while the macro is running.</td>
</tr>
<tr>
<td>Close</td>
<td>Object Type: Form</td>
<td>Object Name: Product List</td>
</tr>
<tr>
<td></td>
<td>Save: No</td>
<td>Close Product List form.</td>
</tr>
<tr>
<td>OpenForm</td>
<td>Data Mode: Add</td>
<td>Form Name: Products</td>
</tr>
<tr>
<td></td>
<td>Window Mode: Normal</td>
<td>View: Form</td>
</tr>
<tr>
<td>SetValue</td>
<td>Item: [Forms]!</td>
<td>Set the SupplierID control to the current supplier on the Suppliers form.</td>
</tr>
<tr>
<td></td>
<td>[Products]!</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[SupplierID]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Expression:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SupplierID</td>
<td></td>
</tr>
</tbody>
</table>
Go to the CategoryID control.
### Required Property

You can use the **Required** property to specify whether a value is required in a field. If this property is set to Yes, when you enter data in a record, you must enter a value in the field or in any **control** bound to the field, and the value cannot be **Null**. For example, you might want to be sure that a LastName control has a value for each record. When you want to permit **Null** values in a field, you must not only set the **Required** property to No but, if there is a **ValidationRule** property setting, it must also explicitly state "validationrule Or Is Null".

**Note**  The **Required** property doesn't apply to AutoNumber fields.
Setting

The **Required** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>True (–1)</td>
<td>The field requires a value.</td>
</tr>
<tr>
<td>No</td>
<td>False (0)</td>
<td>(Default) The field doesn't require a value.</td>
</tr>
</tbody>
</table>

You can set this property for all table fields (except **AutoNumber** data type fields) by using the table's property sheet or **Visual Basic**.

**Note** To access a field's **Required** property in Visual Basic, use the DAO **Required** property.
Remarks

The Required property is enforced at the table level by the Microsoft Jet database engine. If you set this property to Yes, the field must receive or already contain a value when it has the focus—when a user enters data in a table (or in a form or datasheet based on the table), when a macro or Visual Basic sets the value of the field, or when data is imported into the table.

You can use the Required and AllowZeroLength properties to differentiate between information that doesn't exist (stored as a zero-length string (" ") in the field) and information that may exist but is unknown (stored as a Null value in the field). If you set the AllowZeroLength property to Yes, a zero-length string will be a valid entry in the field regardless of the Required property setting. If you set Required to Yes and AllowZeroLength to No, you must enter a value in the field, and a zero-length string won't be a valid entry.

Tip

You can use an input mask when data is entered in a field to distinguish between the display of a Null value and a zero-length string. For example, the string "None" could be displayed when a zero-length string is entered.

The following table shows the results you can expect when you combine the settings of the Required and AllowZeroLength properties.

<table>
<thead>
<tr>
<th>Required</th>
<th>AllowZeroLength</th>
<th>User's action</th>
<th>Value stored</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>No</td>
<td>Presses ENTER</td>
<td>Null</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Presses SPACEBAR</td>
<td>Null</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Enters a zero-length string (not allowed)</td>
<td></td>
</tr>
<tr>
<td>No</td>
<td>Yes</td>
<td>Presses ENTER</td>
<td>Null</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Presses SPACEBAR</td>
<td>Null</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Enters a zero-length string Zero-length string</td>
<td></td>
</tr>
<tr>
<td>Yes</td>
<td>No</td>
<td>Presses ENTER</td>
<td>(not allowed)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Presses SPACEBAR</td>
<td>(not allowed)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Enters a zero-length string (not allowed)</td>
<td></td>
</tr>
<tr>
<td>Yes</td>
<td>Yes</td>
<td>Presses ENTER</td>
<td>(not allowed)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Presses SPACEBAR</td>
<td>Zero-length string</td>
</tr>
</tbody>
</table>
If you set the **Required** property to Yes for a field in a table that already contains data, Microsoft Access gives you the option of checking whether the field has a value in all existing records. However, you can require that a value be entered in this field in all new records even if there are existing records with **Null** values in the field.

**Note** To enforce a relationship between related tables that don't allow **Null** values, set the **Required** property of the **foreign key** field in the related table to Yes. The Jet database engine then ensures that you have a related record in the parent table before you can create a record in the child table. If the foreign key field is part of the **primary key** of the child table, this is unnecessary, because a primary key field can't contain a **Null** value.
FindRecord Action

You can use the FindRecord action to find the first instance of data that meets the criteria specified by the FindRecord arguments. This data can be in the current record, in a succeeding or prior record, or in the first record. You can find records in the active table datasheet, query datasheet, form datasheet, or form.
Setting

The FindRecord action has the following arguments.

<table>
<thead>
<tr>
<th>Action argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Find What</td>
<td>Specifies the data you want to find in the record. Enter the text, number, or date you want to find or type an expression, which is preceded by an equal sign (=), in the <strong>Find What</strong> box in the <strong>Action Arguments</strong> section of the Macro window. You can use wildcard characters. This is a required argument. Specifies where the data is located in the field. You can specify a search for data in any part of the field (<strong>Any Part Of Field</strong>), for data that fills the entire field (<strong>Whole Field</strong>), or for data located at the beginning of the field (<strong>Start Of Field</strong>). The default is <strong>Whole Field</strong>.</td>
</tr>
<tr>
<td>Match</td>
<td>Specifies whether the search is case-sensitive. Click <strong>Yes</strong> (conduct a case-sensitive search) or <strong>No</strong> (search without matching uppercase and lowercase letters exactly). The default is <strong>No</strong>. Specifies whether the search proceeds from the current record up to the beginning of the records (<strong>Up</strong>); down to the end of the records (<strong>Down</strong>); or down to the end of the records and then from the beginning of the records to the current record, so all records are searched (<strong>All</strong>). The default is <strong>All</strong>. Specifies whether the search includes formatted data. Click <strong>Yes</strong> (Microsoft Access searches for the data as it's formatted and displayed in the field) or <strong>No</strong> (Access searches for the data as it's stored in the database, which isn't always the same as it's displayed). The default is <strong>No</strong>.</td>
</tr>
<tr>
<td>Match Case</td>
<td>You can use this feature to restrict the search to data in a particular format. For example, click <strong>Yes</strong> and type <strong>1,234</strong> in the Find What argument to find a value of 1,234 in a field formatted to include commas. Click <strong>No</strong> if you want to type <strong>1234</strong> to search for the data in this field.</td>
</tr>
<tr>
<td>Search</td>
<td>To search for dates, click <strong>Yes</strong> to find a date exactly as it's formatted, such as 09-March-2001. If you click <strong>No</strong>, enter the date for the Find</td>
</tr>
</tbody>
</table>
Search As  What argument in the format that is set in the regional settings in Formatted Windows Control Panel. This format is shown in the **Short date format** box found on the **Date** tab in the regional settings. For example, if the **Short date format** box is set to **M/d/yy**, you can enter 3/9/01, and Access will find all entries in a Date field that correspond to March 9, 2001, regardless of how this field is formatted.

**Note**  The Search As Formatted argument takes effect only if the current field is a **bound control**, the Match argument is set to **Whole Field**, the Only Current Field argument is set to **Yes**, and the Match Case argument is set to **No**.

If you set Match Case to **Yes** or Only Current Field to **No**, you must also set Search As Formatted to **Yes**.

**Only Current Field**  Specifies whether the search is confined to the current field in each record or includes all fields in each record. The current field search is faster. Click **Yes** (confine the search to the current field) or **No** (search in all fields in each record). The default is **Yes**.

**Find First**  Specifies whether the search starts at the first record or at the current record. Click **Yes** (start at the first record) or **No** (start at the current record). The default is **Yes**.
Remarks

When a macro runs the FindRecord action, Access searches for the specified data in the records (the order of the search is determined by the setting of the Search argument). When Access finds the specified data, the data is selected in the record.

The FindRecord action is equivalent to clicking Find on the Edit menu, and its arguments are the same as the options in the Find In Field dialog box, available by clicking Find on the Edit menu. If you set the FindRecord arguments in the Macro window and then run the macro, you'll see the corresponding options selected in the Find In Field dialog box when you click Find.

Access retains the most recent FindRecord arguments during a database session so that you don't need to enter the same criteria repeatedly as you perform subsequent FindRecord operations. If you leave an argument blank, Access uses the most recent setting for the argument, as set either by a previous FindRecord action or in the Find In Field dialog box.

When you want to find a record by using a macro, use the FindRecord action, not the RunCommand action with its argument set to run the Find command.

**Note** While the FindRecord action corresponds to Find on the Edit menu for tables, queries, and forms, it doesn't correspond to Find on the Edit menu in the Code window. You can't use the FindRecord action to search for text in modules.

If the currently selected text is the same as the search text at the time the FindRecord macro action is carried out, the search begins immediately following the selection in the same field as the selection, and in the same record. Otherwise the search begins at the start of the current record. This enables you to find multiple instances of the same search criteria that might appear in a single record.

However, note that if you use a command button to run a macro containing the FindRecord action, the first instance of the search criteria will be found repeatedly. This behavior occurs because clicking the command button removes the focus from the field containing the matching value. The FindRecord action will then begin searching from the start of the record. To avoid this problem, run
the macro by using a technique that doesn't change the focus, such as a custom toolbar button or a key combination defined in an AutoKeys macro, or set the focus in the macro to the field containing the search criteria before you carry out the FindRecord action.

**Security**  Avoid using the `SendKeys` statement or an AutoKeys macro with sensitive or confidential information. A malicious user could intercept the keystrokes and compromise the security of your computer and data.

The same behavior also occurs if you use a command button to run a macro containing the FindNext action.

To run the FindRecord action in Visual Basic, use the `FindRecord` method of the `DoCmd` object.
Echo Action

You can use the Echo action to specify whether `echo` is turned on. For example, you can use this action to hide or show the results of a `macro` while it runs.
## Setting

The Echo action has the following arguments.

<table>
<thead>
<tr>
<th><strong>Action argument</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Echo On</td>
<td>Click <strong>Yes</strong> (turn echo on) or <strong>No</strong> (turn echo off) in the <strong>Echo On</strong> box in the <strong>Action Arguments</strong> section of the <strong>Macro window</strong>. The default is <strong>Yes</strong>.</td>
</tr>
<tr>
<td>Status Bar Text</td>
<td>The text to display in the <strong>status bar</strong> when echo is turned off. For example, when echo is turned off, the status bar can display &quot;The macro is running.&quot;</td>
</tr>
</tbody>
</table>
Remarks

When Microsoft Access runs a macro, screen updating often shows information not essential to the functioning of the macro. When you set the Echo On argument to No, the macro runs without updating the screen. When the macro finishes, Microsoft Access automatically turns echo back on and repaints the window. The No setting for the Echo On argument doesn't affect the macro or its results.

The Echo action doesn't suppress the display of modal dialog boxes, such as error messages, or pop-up forms, such as property sheets. You can use these to gather or display information even if echo is turned off. To turn off all message or dialog boxes except error message boxes and dialog boxes that require the user to enter information, use the SetWarnings action.

You can run the Echo action more than once in a macro. This allows you to change the status bar text while the macro runs.

If you turn echo off, you can use the Hourglass action to change the mouse pointer into an hourglass icon (or whatever mouse pointer icon you've set for "Busy" in Microsoft Windows 2000 or Microsoft Windows XP) to provide a visual indication that the macro is running.

To run the Echo action in Visual Basic, use the Echo method of the DoCmd object.
Examples

Set the value of a control by using a macro

The following macro opens the Add Products form from a button on the Suppliers form. It shows the use of the Echo, Close, OpenForm, SetValue, and GoToControl actions. The SetValue action sets the Supplier ID control on the Products form to the current supplier on the Suppliers form. Then the GoToControl action moves the focus to the Category ID field, where you can begin to enter data for the new product. This macro should be attached to the Add Products button on the Suppliers form.

<table>
<thead>
<tr>
<th>Action</th>
<th>Arguments: Setting</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Echo</td>
<td>Echo On: No</td>
<td>Stop screen updating while the macro is running.</td>
</tr>
<tr>
<td></td>
<td>Object Type: Form</td>
<td></td>
</tr>
<tr>
<td>Close</td>
<td>Object Name: Product List</td>
<td>Close Product List form.</td>
</tr>
<tr>
<td></td>
<td>Save: No</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Form Name: Products</td>
<td></td>
</tr>
<tr>
<td></td>
<td>View: Form</td>
<td></td>
</tr>
<tr>
<td>OpenForm</td>
<td>Data Mode: Add</td>
<td>Open the Products form.</td>
</tr>
<tr>
<td></td>
<td>Window Mode: Normal</td>
<td></td>
</tr>
<tr>
<td>SetValue</td>
<td>Item: [Forms]! [Products]! [SupplierID]</td>
<td>Set the Supplier ID control to the current supplier on the Suppliers form.</td>
</tr>
<tr>
<td></td>
<td>Expression: SupplierID</td>
<td></td>
</tr>
</tbody>
</table>
Synchronize forms by using a macro

The following macro opens a Product List form in the lower-right corner of the Suppliers form, displaying the current supplier's products. It shows the use of the Echo, MsgBox, GoToControl, StopMacro, OpenForm, and MoveSize actions. It also shows the use of a conditional expression with the MsgBox, GoToControl, and StopMacro actions. This macro should be attached to the Review Products button on the Suppliers form.

<table>
<thead>
<tr>
<th>Condition</th>
<th>Action</th>
<th>Arguments: Setting</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>IsNull([Supplier ID])</td>
<td>MsgBox</td>
<td>Message: Move to the supplier record whose products you want to see, then click the Review Products button again.</td>
<td>If there is no current supplier on the Suppliers form, display a message.</td>
</tr>
<tr>
<td>Echo</td>
<td>Echo</td>
<td>Echo On: No</td>
<td>Stop screen updating while the macro is running.</td>
</tr>
<tr>
<td></td>
<td>MsgBox</td>
<td>Beep: Yes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MsgBox</td>
<td>Type: None</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MsgBox</td>
<td>Title: Select a Supplier</td>
<td></td>
</tr>
<tr>
<td>GoToControl</td>
<td>Control Name: CompanyName</td>
<td></td>
<td>Move focus to the CompanyName control.</td>
</tr>
<tr>
<td>...</td>
<td>StopMacro</td>
<td>Form Name: Product List</td>
<td>Stop the macro.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>View: Datasheet</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Filter Name:</td>
<td></td>
</tr>
</tbody>
</table>
OpenForm

Where Condition: [Supplier ID] = [Forms]!Suppliers![SupplierID]
show the current supplier's products.

Data Mode: **Read Only**

Window Mode: **Normal**

MoveSize

Right: 0.7799" Position the Product List form in the lower right of the Suppliers form.

Down: 1.8"
### Values for the PaperSize Member

<table>
<thead>
<tr>
<th>Value</th>
<th>Paper size</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Letter (8.5 x 11 in.)</td>
</tr>
<tr>
<td>2</td>
<td>Letter Small (8.5 x 11 in.)</td>
</tr>
<tr>
<td>3</td>
<td>Tabloid (11 x 17 in.)</td>
</tr>
<tr>
<td>4</td>
<td>Ledger (17 x 11 in.)</td>
</tr>
<tr>
<td>5</td>
<td>Legal (8.5 x 14 in.)</td>
</tr>
<tr>
<td>6</td>
<td>Statement (5.5 x 8.5 in.)</td>
</tr>
<tr>
<td>7</td>
<td>Executive (7.25 x 10.5 in.)</td>
</tr>
<tr>
<td>8</td>
<td>A3 (297 x 420 mm)</td>
</tr>
<tr>
<td>9</td>
<td>A4 (210 x 297 mm)</td>
</tr>
<tr>
<td>10</td>
<td>A4 Small (210 x 297 mm)</td>
</tr>
<tr>
<td>11</td>
<td>A5 (148 x 210 mm)</td>
</tr>
<tr>
<td>12</td>
<td>B4 (250 x 354 mm)</td>
</tr>
<tr>
<td>13</td>
<td>B5 (182 x 257 mm)</td>
</tr>
<tr>
<td>14</td>
<td>Folio (8.5 x 13 in.)</td>
</tr>
<tr>
<td>15</td>
<td>Quarto (215 x 275 mm)</td>
</tr>
<tr>
<td>16</td>
<td>11 x 17 in.</td>
</tr>
<tr>
<td>17</td>
<td>Note (8.5 x 11 in.)</td>
</tr>
<tr>
<td>18</td>
<td>Envelope #9 (3.875 x 8.875 in.)</td>
</tr>
<tr>
<td>19</td>
<td>Envelope #10 (4.125 x 9.5 in.)</td>
</tr>
<tr>
<td>20</td>
<td>Envelope #11 (4.5 x 10.375 in.)</td>
</tr>
<tr>
<td>21</td>
<td>Envelope #12 (4.25 x 11 in.)</td>
</tr>
<tr>
<td>22</td>
<td>Envelope #14 (5 x 11.5 in.)</td>
</tr>
<tr>
<td>23</td>
<td>C size sheet (17 x 22 in.)</td>
</tr>
<tr>
<td>24</td>
<td>D size sheet (22 x 34 in.)</td>
</tr>
<tr>
<td>25</td>
<td>E size sheet (34 x 44 in.)</td>
</tr>
<tr>
<td>26</td>
<td>Envelope DL (110 x 220 mm)</td>
</tr>
<tr>
<td>27</td>
<td>Envelope C5 (162 x 229 mm)</td>
</tr>
<tr>
<td>28</td>
<td>Envelope C3 (324 x 458 mm)</td>
</tr>
</tbody>
</table>
30   Envelope C4 (229 x 324 mm)
31   Envelope C6 (114 x 162 mm)
32   Envelope C65 (114 x 229 mm)
33   Envelope B4 (250 x 353 mm)
34   Envelope B5 (176 x 250 mm)
35   Envelope B6 (176 x 125 mm)
36   Envelope (110 x 230 mm)
37   Envelope Monarch (3.875 x 7.5 in.)
38   6-3/4 Envelope (3.625 x 6.5 in.)
39   US Std Fanfold (14.875 x 11 in.)
40   German Std Fanfold (8.5 x 12 in.)
41   German Legal Fanfold (8.5 x 13 in.)
256  User-defined
## Values for the DefaultSource Member

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Upper or only one bin</td>
</tr>
<tr>
<td>2</td>
<td>Lower bin</td>
</tr>
<tr>
<td>3</td>
<td>Middle bin</td>
</tr>
<tr>
<td>4</td>
<td>Manual bin</td>
</tr>
<tr>
<td>5</td>
<td>Envelope bin</td>
</tr>
<tr>
<td>6</td>
<td>Envelope manual bin</td>
</tr>
<tr>
<td>7</td>
<td>Automatic bin</td>
</tr>
<tr>
<td>8</td>
<td>Tractor bin</td>
</tr>
<tr>
<td>9</td>
<td>Small-format bin</td>
</tr>
<tr>
<td>10</td>
<td>Large-format bin</td>
</tr>
<tr>
<td>11</td>
<td>Large-capacity bin</td>
</tr>
<tr>
<td>14</td>
<td>Cassette bin</td>
</tr>
<tr>
<td>256</td>
<td>Device-specific bins start here</td>
</tr>
</tbody>
</table>
## Values for the TTOption Member

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Print TrueType fonts as graphics. This is the default for dot-matrix printers. Download TrueType fonts as soft fonts (fonts that are loaded into the printer's memory for rendering). This is the default for Hewlett-Packard printers that use Printer Control Language (PCL). Substitute device fonts for TrueType fonts. This is the default for PostScript printers.</td>
</tr>
</tbody>
</table>
RowSourceType Property (User-Defined Function) - Code Argument Values

The Visual Basic function you create must accept five arguments. The first argument must be declared as a control and the remaining arguments as Variants. The function itself must return a Variant.

Function functionname (fld As Control, id As Variant, row As Variant, col As Variant, code As Variant) As Variant

The Function procedure has the following five required arguments.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fld</td>
<td>A control variable that refers to the list box or combo box being filled.</td>
</tr>
<tr>
<td>id</td>
<td>A unique value that identifies the control being filled. This is useful when you want to use the same user-defined function for more than one list box or combo box and must distinguish between them. (The example sets this variable to the value of the Timer function.)</td>
</tr>
<tr>
<td>row</td>
<td>The row being filled (zero-based).</td>
</tr>
<tr>
<td>col</td>
<td>The column being filled (zero-based).</td>
</tr>
<tr>
<td>code</td>
<td>An intrinsic constant that specifies the kind of information being requested.</td>
</tr>
</tbody>
</table>

Note Because Microsoft Access calls a user-defined function several times to insert items into a list, often you must preserve information from call to call. The best way to do this is to use Static variables.

Microsoft Access calls the user-defined function by repeatedly using different values in the code argument to specify the information it needs. The code argument can use the following intrinsic constants.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Meaning</th>
<th>Function returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>-------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>acLBInitialize</td>
<td>Initialize</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Nonzero if the function can fill the list; <strong>False</strong> (0) or <strong>Null</strong> otherwise.</td>
<td></td>
</tr>
<tr>
<td>acLBOpen</td>
<td>Open</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Nonzero ID value if the function can fill the list; <strong>False</strong> or <strong>Null</strong> otherwise.</td>
<td></td>
</tr>
<tr>
<td>acLBGetRowCount</td>
<td>Number of rows</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Number of rows in the list (can be zero); –1 if unknown.</td>
<td></td>
</tr>
<tr>
<td>acLBGetColumnCount</td>
<td>Number of columns</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Number of columns in the list (can't be zero); must match the property sheet value.</td>
<td></td>
</tr>
<tr>
<td>acLBGetColumnWidth</td>
<td>Column width</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Width (in <strong>twips</strong>) of the column specified by the col argument; –1 to use the default width.</td>
<td></td>
</tr>
<tr>
<td>acLBGetValue</td>
<td>List entry</td>
<td></td>
</tr>
<tr>
<td></td>
<td>List entry to be displayed in the row and column specified by the row and col arguments.</td>
<td></td>
</tr>
<tr>
<td>acLBGetFormat</td>
<td>Format string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Format string to be used to format the list entry displayed in the row and column specified by the row and col arguments; –1 to use the default format.</td>
<td></td>
</tr>
<tr>
<td>acLBEnd</td>
<td>End (the last call to a user-defined function)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Nothing. End always uses this value</td>
<td></td>
</tr>
<tr>
<td>acLBClose</td>
<td>(Not used)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Not used.</td>
<td></td>
</tr>
</tbody>
</table>

Microsoft Access calls your user-defined function once for **acLBInitialize**, **acLBOpen**, **acLBGetRowCount**, and **acLBGetColumnCount**. It initializes the user-defined function, opens the query, and determines the number of rows and columns.

Microsoft Access calls your user-defined function twice for **acLBGetColumnWidth**— once to determine the total width of the list box or combo box and a second time to set the column width.
The number of times your user-defined function is called for `acLBGetValue` and `acLBGetFormat` to get list entries and to format strings varies depending on the number of entries, the user's scrolling, and other factors.

Microsoft Access calls the user-defined function for `acLBEnd` when the form is closed or each time the list box or combo box is queried.

Whenever a particular value (such as the number of columns) is required, returning `Null` or any invalid value causes Microsoft Access to stop calling the user-defined function with that code.

**Tip**

You can use the Select Case code structure from the example as a template for your own `RowSourceType` property user-defined functions.
Example

The following user-defined function returns a list of the next four Mondays following today's date. To call this function from a list box control, enter **ListMondays** as the **RowSourceType** property setting and leave the **RowSource** property setting blank.

```vba
Function ListMondays(fld As Control, id As Variant, _
    row As Variant, col As Variant, code As Variant) As Variant
    Dim intOffset As Integer
    Select Case code
        Case acLBInitialize    ' Initialize.
            ListMondays = True
        Case acLBOpen    ' Open.
            ListMondays = Timer    ' Unique ID.
        Case acLBGetRowCount    ' Get rows.
            ListMondays = 4
        Case acLBGetColumnCount    ' Get columns.
            ListMondays = 1
        Case acLBGetColumnWidth    ' Get column width.
            ListMondays = -1    ' Use default width.
        Case acLBGetValue    ' Get the data.
            intOffset = Abs((9 - Weekday(Now)) Mod 7)
            ListMondays = Format(Now() + _
                intOffset + 7 * row, "mmm d")
    End Select
End Function
```

The next example uses a static array to store the names of the databases in the current directory. To call this function, enter **ListMDBs** as the **RowSourceType** property setting and leave the **RowSource** property setting blank.

```vba
Function ListMDBs(fld As Control, id As Variant, _
    row As Variant, col As Variant, code As Variant) As Variant
    Static dbs(127) As String, Entries As Integer
    Dim ReturnVal As Variant
    ReturnVal = Null
    Select Case code
        Case acLBInitialize    ' Initialize.
            Entries = 0
            dbs(Entries) = Dir("*.MDB")
            Do Until dbs(Entries) = "" Or Entries >= 127
```
Entries = Entries+1
        dbs(Entries) = Dir
    Loop
    ReturnVal = Entries
    Case acLBOpen          ' Open.
        ' Generate unique ID for control.
        ReturnVal = Timer
    Case acLBGetRowCount   ' Get number of rows.
        ReturnVal = Entries
    Case acLBGetColumnCount ' Get number of columns.
        ReturnVal = 1
    Case acLBGetColumnWidth ' Column width.
        ' -1 forces use of default width.
        ReturnVal = -1
    Case acLBGetValue       ' Get data.
        ReturnVal = dbs(row)
    Case acLBEnd            ' End.
    Erase dbs
End Select
ListMDBs = ReturnVal
End Function
AllowShortcutMenus Property

You can use the AllowShortcutMenus property to specify whether or not your application allows Microsoft Access to display built-in shortcut menus. For example, you can use the AllowShortcutMenus property in conjunction with the AllowFullMenus property in your application to prevent users from using any built-in menu bar, toolbars, or shortcut menu commands that enable users to change the design of database objects.
Setting

The `AllowShortcutMenus` property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>True (–1)</td>
<td>Allow Microsoft Access built-in shortcut menus to be displayed.</td>
</tr>
<tr>
<td>False (0)</td>
<td>Don't allow Microsoft Access built-in shortcut menus to be displayed.</td>
</tr>
</tbody>
</table>

The easiest way to set this property is by using the **Allow Default Shortcut Menus** option in the **Startup** dialog box, available by clicking **Startup** on the **Tools** menu. You can also set this property by using a **macro** or **Visual Basic**.

To set the `AllowShortcutMenus` property by using a macro or Visual Basic, you must first either set the property in the **Startup** dialog box once or create the property in the following ways:

- In a **Microsoft Access database** (.mdb), you can add it by using the **CreateProperty** method and append it to the **Properties** collection of the **Database** object.
- In a **Microsoft Access project** (.adp), you can add it to the **AccessObjectProperties** collection of the **CurrentProject** object by using the **Add** method.
Remarks

The setting of this property doesn't affect custom shortcut menus and global shortcut menus. You can use the ShortcutMenuBar property to display custom shortcut menus for forms, form controls, and reports, and the StartupShortcutMenuBar property or the ShortcutMenuBar property of the Application object to display a global shortcut menu.

If you want to display built-in shortcut menus in your application, but don't want the user to be able to change them, set the AllowShortcutMenus property to True and set the AllowToolbarChanges property to False.

This property's setting doesn't take effect until the next time the database opens.
Description Property

You can use the **Description** property to provide information about objects contained in the [Database window](#) as well as about individual table or query fields.
Setting

For a database object, click **Properties** on the **View** menu and enter the description text in the **Description** box. For tables or queries, you can also enter the description in the table's or query's **property sheet**. An object's description appears next to the object's name in the Database window when you click **Details** on the **View** menu.

For individual table or query fields, enter the field description in the upper portion of **table Design view** or in the Field Properties property sheet in the **Query window**. The maximum length is 255 characters.

In **Visual Basic**, to set this property for the first time in a Microsoft Access **project** (.adp), you must create an application-defined property by using the **Add** method. In a Microsoft Access **database** (.mdb), you must use the DAO **CreateProperty** method.
Remarks

An object's description is displayed in the Description column in the Details view of the Database window.

If you create controls by dragging a field from the field list, Microsoft Access copies the field's Description property to the control's StatusBarText property.

Note  For a linked table, Microsoft Access displays the connection information in the Description property.
GoToControl Action

You can use the GoToControl action to move the focus to the specified field or control in the current record of the open form, form datasheet, table datasheet, or query datasheet. You can use this action when you want a particular field or control to have the focus. This field or control can then be used for comparisons or FindRecord actions. You can also use this action to navigate in a form according to certain conditions. For example, if the user enters No in a Married control on a health insurance form, the focus can automatically skip the Spouse Name control and move to the next control.

Note This action is not available for use with data access pages.
Setting

The GoToControl action has the following argument.

<table>
<thead>
<tr>
<th>Action argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control Name</td>
<td>The name of the field or control where you want the focus. Enter the field or control name in the <strong>Control Name</strong> box in the <strong>Action Arguments</strong> section of the <strong>Macro window</strong>. This is a required argument.</td>
</tr>
</tbody>
</table>

**Note**  Enter only the name of the field or control in the Control Name argument, not the fully qualified identifier, such as Forms!Products![Product ID].
Remarks

You can't use the GoToControl action to move the focus to a control on a hidden form.

--- Tip ---

You can use the GoToControl action to move to a subform, which is a type of control. You can then use the GoToRecord action to move to a particular record in the subform. You can also move to a control on a subform by using the GoToControl action to move first to the subform and then to the control on the subform.

To run the GoToControl action in Visual Basic, use the GoToControl method of the DoCmd object. You can also use the SetFocus method to move the focus to a control on a form or any of its subforms, or to a field in an open table, query, or form datasheet.
Examples

Set the value of a control by using a macro

The following macro opens the Add Products form from a button on the Suppliers form. It shows the use of the Echo, Close, OpenForm, SetValue, and GoToControl actions. The SetValue action sets the Supplier ID control on the Products form to the current supplier on the Suppliers form. Then the GoToControl action moves the focus to the Category ID field, where you can begin to enter data for the new product. This macro should be attached to the Add Products button on the Suppliers form.

<table>
<thead>
<tr>
<th>Action</th>
<th>Arguments: Setting</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Echo</td>
<td>Echo On: No</td>
<td>Stop screen updating while the macro is running.</td>
</tr>
<tr>
<td></td>
<td>Object Type: Form</td>
<td></td>
</tr>
<tr>
<td>Close</td>
<td>Object Name: Product List</td>
<td>Close Product List form.</td>
</tr>
<tr>
<td></td>
<td>Save: No</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Form Name: Products</td>
<td></td>
</tr>
<tr>
<td></td>
<td>View: Form</td>
<td></td>
</tr>
<tr>
<td>OpenForm</td>
<td>Data Mode: Add</td>
<td>Open the Products form.</td>
</tr>
<tr>
<td></td>
<td>Window Mode: Normal</td>
<td></td>
</tr>
<tr>
<td>SetValue</td>
<td>Item: [Forms]![Products]![SupplierID]</td>
<td>Set the Supplier ID control to the current supplier on the Suppliers form.</td>
</tr>
<tr>
<td></td>
<td>Expression: SupplierID</td>
<td></td>
</tr>
</tbody>
</table>
GoToControl Control Name: CategoryID

Go to the Category ID control.

**Validate data by using a macro**

The following validation macro checks the postal codes entered in a Suppliers form. It shows the use of the StopMacro, MsgBox, CancelEvent, and GoToControl actions. A conditional expression checks the country/region and postal code entered in a record on the form. If the postal code is not in the right format for the country/region, the macro displays a message box and cancels saving the record. It then returns you to the Postal Code control, where you can correct the error. This macro should be attached to the BeforeUpdate property of the Suppliers form.

<table>
<thead>
<tr>
<th>Condition</th>
<th>Action</th>
<th>Arguments: Setting</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>IsNull([Country])</td>
<td>StopMacro</td>
<td>Message: The postal code must be 5 characters.</td>
<td>If Country is <strong>Null</strong>, postal code can't be validated.</td>
</tr>
<tr>
<td>[Country] In (&quot;France&quot;,&quot;Italy&quot;,&quot;Spain&quot;) And Len([Postal Code]) &lt;&gt; 5</td>
<td>MsgBox</td>
<td>Beep: <strong>Yes</strong> Type: <strong>Information</strong> Title: Postal Code Error</td>
<td>If the postal code isn't 5 characters, display a message.</td>
</tr>
<tr>
<td>...</td>
<td>CancelEvent</td>
<td></td>
<td>Cancel the event.</td>
</tr>
<tr>
<td></td>
<td>GoToControl</td>
<td>Control Name: PostalCode Message: The postal code must be 4 characters.</td>
<td></td>
</tr>
</tbody>
</table>
[Country] In ("Australia","Singapore") And Len([Postal Code]) <> 4

MsgBox Beep: Yes
Type: Information
Title: Postal Code Error
If the postal code isn't 4 characters, display a message.

... CancelEvent
GoToControl Control Name: PostalCode
Message: The postal code is not valid.
Example of Canadian code: H1J 1C3

MsgBox Beep: Yes
Type: Information
Title: Postal Code Error
If the postal code isn't correct for Canada, display a message.
(Example of Canadian code: H1J 1C3)

... CancelEvent
Cancel the event.
CancelEvent Action

You can use the CancelEvent action to cancel the event that caused Microsoft Access to run the macro containing this action. The macro name is the setting of an event property such as BeforeUpdate, OnOpen, OnUnload, or OnPrint.
Setting

The CancelEvent action doesn't have any arguments.
Remarks

In a form, you typically use the CancelEvent action in a validation macro with the BeforeUpdate event property. When a user enters data in a control or record, Access runs the macro before adding the data to the database. If the data fails the validation conditions in the macro, the CancelEvent action cancels the update process before it starts.

Often, you use this action with the MsgBox action to indicate that the data has failed the validation conditions and to provide helpful information about the kind of data that should be entered.

The following events can be canceled by the CancelEvent action.

<table>
<thead>
<tr>
<th>Event</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>ApplyFilter</td>
<td>Dirty</td>
</tr>
<tr>
<td>BeforeDelConfirm</td>
<td>Exit</td>
</tr>
<tr>
<td>BeforeInsert</td>
<td>NoData</td>
</tr>
<tr>
<td>BeforeUpdate</td>
<td>Open</td>
</tr>
<tr>
<td>DblClick</td>
<td>Format</td>
</tr>
<tr>
<td>Delete</td>
<td>KeyPress</td>
</tr>
<tr>
<td></td>
<td>Unload</td>
</tr>
</tbody>
</table>

Note  You can use the CancelEvent action with the MouseDown event only to cancel the event that occurs when you right-click an object.

If a control's OnDblClick event property setting specifies a macro containing the CancelEvent action, the action cancels the DblClick event.

For events that can be canceled, the default behavior for the event (that is, what Access typically does when the event occurs) occurs after the macro for the event runs. This enables you to cancel the default behavior. For example, when you double-click a word that the insertion point is on in a text box, Access normally selects the word. You can cancel this default behavior in the macro for the DblClick event and perform some other action, such as opening a form containing information about the data in the text box. For events that can't be canceled, the default behavior occurs before the macro runs.

Note  If a form's OnUnload event property specifies a macro that carries out a
CancelEvent action, you won't be able to close the form. You must either correct the condition that caused the CancelEvent action to be carried out or open the macro and delete the CancelEvent action. If the form is a **modal** form, you won't be able to open the macro.

To carry out the CancelEvent action in Visual Basic, use the **DoCmd** method of the **DoCmd** object.
Validate data by using a macro

The following validation macro checks the postal codes entered in a Suppliers form. It shows the use of the StopMacro, MsgBox, CancelEvent, and GoToControl actions. A conditional expression checks the country/region and postal code entered in a record on the form. If the postal code is not in the right format for the country/region, the macro displays a message box and cancels saving the record. It then returns you to the Postal Code control, where you can correct the error. This macro should be attached to the BeforeUpdate property of the Suppliers form.

<table>
<thead>
<tr>
<th>Condition</th>
<th>Action</th>
<th>Arguments: Setting</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>IsNull([Country])</td>
<td>StopMacro</td>
<td></td>
<td>If Country is Null, postal code can't be validated.</td>
</tr>
<tr>
<td>[Country] In (&quot;France&quot;,&quot;Italy&quot;,&quot;Spain&quot;) And Len([Postal Code]) &lt;&gt; 5</td>
<td>MsgBox</td>
<td>Beep: Yes, Type: Information</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Title: Postal Code Error</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>CancelEvent</td>
<td></td>
<td>Cancel the event.</td>
</tr>
<tr>
<td></td>
<td>GoToControl</td>
<td>Control Name: PostalCode, Message: The postal code must be 4 characters.</td>
<td></td>
</tr>
</tbody>
</table>
[Country] In
("Australia","Singapore") MsgBox
And Len([Postal Code])
<> 4

Beep: Yes
Type: Information
Title: Postal Code Error

If the postal code isn't
4 characters, display a
message.

CancelEvent
Cancel the event.

GoToControl
Control Name: PostalCode
Message: The postal code is not valid.
Example of Canadian code: H1J 1C3

Beep: Yes
Type: Information
Title: Postal Code Error

If the postal code isn't
correct for Canada, display a
message.
(Example of Canadian
code: H1J 1C3)

CancelEvent
Cancel the event.
GoToRecord Action

You can use the GoToRecord action to make the specified record the current record in an open table, form, or query result set.
## Setting

The GoToRecord action has the following arguments.

<table>
<thead>
<tr>
<th>Action argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Object Type</strong></td>
<td>The type of object that contains the record you want to make current. Click <strong>Table</strong>, <strong>Query</strong>, <strong>Form</strong>, <strong>Server View</strong>, <strong>Stored Procedure</strong>, or <strong>Function</strong> in the <strong>Object Type</strong> box in the <strong>Action Arguments</strong> section of the <strong>Macro window</strong>. Leave this argument blank to select the active object.</td>
</tr>
<tr>
<td><strong>Object Name</strong></td>
<td>The name of the object that contains the record you want to make the current record. The <strong>Object Name</strong> box shows all objects in the current database of the type selected by the Object Type argument. If you leave the Object Type argument blank, leave this argument blank also.</td>
</tr>
<tr>
<td><strong>Record</strong></td>
<td>The record to make the current record. Click <strong>Previous</strong>, <strong>Next</strong>, <strong>First</strong>, <strong>Last</strong>, <strong>Go To</strong>, or <strong>New</strong> in the <strong>Record</strong> box. The default is <strong>Next</strong>. An integer or <strong>expression</strong> that evaluates to an integer. An expression must be preceded by an equal sign (=). This argument specifies the record to make the current record. You can use the Offset argument in two ways:</td>
</tr>
</tbody>
</table>
| **Offset** | • When the Record argument is **Next** or **Previous**, Microsoft Access moves the number of records forward or backward specified in the Offset argument.  
• When the Record argument is **Go To**, Access moves to the record with the number equal to the Offset argument. The record number is shown in the **record number box** at the bottom of the window. |

**Note** If you use the **First**, **Last**, or **New** setting for the Record argument, Access ignores the Offset argument. If you enter an Offset argument that's too large, Access displays an error message. You can't enter negative numbers for the Offset argument.
Remarks

If the focus is in a particular control in a record, this action leaves it in the same control for the new record.

You can use the New setting for the Record argument to move to the blank record at the end of a form or table so you can enter new data.

This action is similar to clicking Go To on the Edit menu. The First, Last, Next, Previous, and New Record subcommands of the Go To command have the same effect on the selected object as the First, Last, Next, Previous, and New settings for the Record argument. You can also move to records by using the navigation buttons at the bottom of the window.

You can use the GoToRecord action to make a record on a hidden form the current record if you specify the hidden form in the Object Type and Object Name arguments.

To run the GoToRecord action in Microsoft Visual Basic, use the GoToRecord method of the DoCmd object.
Requery Action

You can use the Requery action to update the data in a specified control on the active object by requeries the source of the control. If no control is specified, this action requeries the source of the object itself. Use this action to ensure that the active object or one of its controls displays the most current data.
Setting

The Requery action has the following argument.

<table>
<thead>
<tr>
<th>Action argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control Name</td>
<td>The name of the control you want to update. Enter the control name in the <strong>Control Name</strong> box in the <strong>Action Arguments</strong> section of the <strong>Macro window</strong>. You should use only the name of the control, not the fully qualified identifier (such as <strong>Forms!formname!controlname</strong>). Leave this argument blank to requery the source of the active object. If the active object is a <strong>datasheet</strong> or a query <strong>result set</strong>, you must leave this argument blank.</td>
</tr>
</tbody>
</table>
Remarks

The Requery action does one of the following:

- Reruns the query on which the control or object is based.
- Displays any new or changed records, and removes any deleted records from the table on which the control or object is based.

**Note** The Requery action does not affect the position of the record pointer.

Controls based on a query or table include:

- List boxes and combo boxes.
- Subform controls.
- OLE objects, such as charts.
- Controls containing domain aggregate functions, such as DSum.

If the specified control isn't based on a query or table, this action forces a recalculation of the control.

If you leave the Control Name argument blank, the Requery action has the same effect as pressing SHIFT+F9 when the object has the focus. If a subform control has the focus, this action requeries only the source of the subform (just as pressing SHIFT+F9 does).

**Note** The Requery action requeries the source of the control or object. In contrast, the RepaintObject action repaints controls in the specified object but doesn't requery the database or display new records. The ShowAllRecords action not only requeries the active object, but it also removes any applied filters, which the Requery action doesn't do.

If you want to requery a control that isn't on the active object, you must use the Requery method in Visual Basic, not the Requery action or its corresponding Requery method of the DoCmd object. The Requery method in Visual Basic is faster than the Requery action or the DoCmd.Requery method. In addition, when you use the Requery action or the DoCmd.Requery method, Microsoft Access closes the query and reloads it from the database, but when you use the Requery method, Access reruns the query without closing and reloading it. Note
that the ActiveX Data Object (ADO) \texttt{Requery} method works the same way as the Access \texttt{Requery} method.
SelectObject Action

You can use the SelectObject action to select a specified database object.
**Setting**

The SelectObject action has the following arguments.

<table>
<thead>
<tr>
<th>Action argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object Type</td>
<td>The type of database object to select. Click <em>Table, Query, Form, Report, Macro, Module, Data Access Pages, Server View, Diagram, Stored Procedure</em>, or <em>Function</em> in the <strong>Object Type</strong> box in the <strong>Action Arguments</strong> section of the <strong>Macro window</strong>. This is a required argument.</td>
</tr>
<tr>
<td>Object Name</td>
<td>The name of the object to select. The <strong>Object Name</strong> box shows all objects in the database of the type selected by the <strong>Object Type</strong> argument. This is a required argument, unless you set the In Database Window argument to <strong>Yes</strong>. If you set the In Database Window argument to <strong>Yes</strong> and leave the Object Name argument blank, Microsoft Access selects the tab in the <strong>Database window</strong> that corresponds to the database object you specify in the <strong>Object Type</strong> argument.</td>
</tr>
</tbody>
</table>

**Note** The object names for **Server View, Diagram, or Stored Procedure** objects are not displayed in the **Object Name** box of an **Access project** (.adp).

| In Database Window | Specifies whether Access selects the object in the Database window. Click **Yes** (to select the object in the Database window) or **No** (not to select the object in the Database window). The default is **No**. |
Remarks

The SelectObject action works with any Access object that can receive the focus. This action gives the specified object the focus and shows the object if it's hidden. If the object is a form, the SelectObject action sets the form's Visible property to Yes and returns the form to the mode set by its form properties (for example, as a modal or pop-up form).

If the object isn't open in one of the other Access windows, you can select it in the Database window by setting the In Database Window argument to Yes. If you set the In Database Window argument to No, an error message appears when you try to select an object that isn't open.

Often, you might use this action to select an object on which you want to perform additional actions. For example, you may want to restore an object that has been minimized (by using the Restore action) or maximize a window that contains an object you want to work with (by using the Maximize action).

If you select a form, you can use the GoToControl, GoToRecord, and GoToPage actions to move to specific areas on the form. The GoToRecord action also works for datasheets.

To run the SelectObject action in Visual Basic, use the SelectObject method of the DoCmd object.
GoToPage Action

You can use the GoToPage action to move the focus in the active form to the first control on a specified page. You can use this action if you've created a form with page breaks that contains groups of related information. For example, you might have an Employees form with personal information on one page, office information on another page, and sales information on a third page. You can use the GoToPage action to move to the desired page. You can also present multiple pages of information on a single form by using tab controls.
Setting

The GoToPage action has the following arguments.

<table>
<thead>
<tr>
<th>Action argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Page Number</td>
<td>The number of the page to which you want to move the focus. Enter the page number in the <strong>Page Number</strong> box in the <strong>Action Arguments</strong> section of the <a href="#">Macro window</a>. If you leave this argument blank, the focus stays on the current page. You can use the Right and Down arguments to display the part of the page you want to see.</td>
</tr>
<tr>
<td>Right</td>
<td>The horizontal position of the spot on the page, measured from the left edge of its containing window, that's to appear at the left edge of the window. This is required if you specify a Down argument.</td>
</tr>
<tr>
<td>Down</td>
<td>The vertical position of the spot on the page, measured from the top edge of its containing window, that's to appear at the top edge of the window. This is required if you specify a Right argument.</td>
</tr>
</tbody>
</table>

**Note** The Right and Down arguments are measured in inches or centimeters, depending on the regional settings in Windows Control Panel.
Remarks

You can use this action to select the first control (as defined by the form's tab order) on the specified page. Use the GoToControl action to move to a particular control on the form.

You can use the Right and Down arguments for forms with pages larger than the Microsoft Access window. Use the Page Number argument to move to the desired page, and then use the Right and Down arguments to display the part of the page you want to see. Access displays the part of the page whose upper-left corner is offset the specified distance from the upper-left corner of the page.

You can't use the GoToPage action in the following cases:

- To move the focus to a page on a hidden form.
- To move the focus from one page to another within the tab control.

To run the GoToPage action in Visual Basic, use the GoToPage method of the DoCmd object.
SendKeys Action

**Security**  Avoid using the **SendKeys** statement or an AutoKeys macro with sensitive or confidential information. A malicious user could intercept the keystrokes and compromise the security of your computer and data.

You can use the SendKeys action to send keystrokes directly to Microsoft Access or to an active Windows-based application.
**Setting**

The SendKeys action has the following arguments.

<table>
<thead>
<tr>
<th>Action argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keystrokes</td>
<td>The keystrokes you want Access or the application to process. Enter the keystrokes in the <strong>Keystrokes</strong> box in the <strong>Action Arguments</strong> section of the <a href="#">Macro window</a>. You can type up to 255 characters. This is a required argument.</td>
</tr>
<tr>
<td>Wait</td>
<td>Specifies whether the <strong>macro</strong> should pause until the keystrokes have been processed. Click <strong>Yes</strong> (to pause) or <strong>No</strong> (to not pause). The default is <strong>No</strong>.</td>
</tr>
</tbody>
</table>
Remarks

Access processes the keystrokes it receives through the SendKeys action exactly as if you had typed them directly in an Access window.

To specify the keystrokes, use the same syntax as you would for the **SendKeys** statement.

**Note**  An error can occur if the Keystrokes argument contains incorrect syntax, misspelled text, or other values that aren't appropriate for the window the keystrokes are sent to.

You can use this action to enter information in a dialog box, particularly if you don't want to interrupt the macro to respond manually to the dialog box. Some Access actions, such as PrintOut and FindRecord, automatically select the options in certain frequently used dialog boxes. You can use the SendKeys action to select the options in less commonly used dialog boxes.

Notes

- Because the dialog box suspends the macro, you must put the SendKeys action before the action that causes the dialog box to open and set the Wait argument to **No**.
- The timing of the keystrokes reaching Access or another application can be tricky. As a result, it's recommended that if there's some other method (such as the FindRecord action) you can use to achieve a desired task, use that method rather than using the SendKeys action to fill in the options in a dialog box.

If you want to send more than 255 characters to Access or another Windows-based application, you can use several SendKeys actions in succession in a macro.

Using the SendKeys action to send keystrokes triggers the appropriate KeyDown, KeyUp, and KeyPress events. Sending non-**ANSI** keystrokes (such as a function key) doesn't trigger the KeyPress event.

This action isn't available from Microsoft Visual Basic. Use the **SendKeys**
statement instead.
MoveSize Action

You can use the MoveSize action to move or resize the active window.
## Setting

The MoveSize action has the following arguments.

<table>
<thead>
<tr>
<th>Action argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Right</td>
<td>The new horizontal position of the window's upper-left corner, measured from the left edge of its containing window. Enter the position in the <strong>Right</strong> box in the <strong>Action Arguments</strong> section of the <strong>Macro window</strong>.</td>
</tr>
<tr>
<td>Down</td>
<td>The new vertical position of the window's upper-left corner, measured from the top edge of its containing window.</td>
</tr>
<tr>
<td>Width</td>
<td>The window's new width.</td>
</tr>
<tr>
<td>Height</td>
<td>The window's new height.</td>
</tr>
</tbody>
</table>

If you leave an argument blank, Microsoft Access uses the window's current setting.

You must enter a value for at least one argument.

**Note** Each measurement is in inches or centimeters, depending on the regional settings in **Windows Control Panel**.
Remarks

This action is similar to clicking **Move** or **Size** on the window's **Control** menu. With the menu commands, you use the keyboard's arrow keys to move or resize the window. With the MoveSize action, you enter the position and size measurements directly. You can also use the mouse to move and size windows.

You can use this action on any window, in any view.

Tips

- To move a window without resizing it, enter values for the Right and Down arguments but leave the Width and Height arguments blank.
- To resize a window without moving it, enter values for the Width and Height arguments but leave the Right and Down arguments blank.

To run the MoveSize action in Visual Basic, use the **MoveSize** method of the **DoCmd** object.
**Example**

Synchronize forms by using a macro

The following macro opens a Product List form in the lower-right corner of the Suppliers form, displaying the current supplier's products. It shows the use of the Echo, MsgBox, GoToControl, StopMacro, OpenForm, and MoveSize actions. It also shows the use of a conditional expression with the MsgBox, GoToControl, and StopMacro actions. This macro should be attached to the Review Products button on the Suppliers form.

<table>
<thead>
<tr>
<th>Condition</th>
<th>Action</th>
<th>Arguments: Setting</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL([Supplier ID])</td>
<td>MsgBox</td>
<td>Message: Move to the supplier record whose products you want to see, then click the Review Products button again.</td>
<td>Stop screen updating while the macro is running.</td>
</tr>
<tr>
<td></td>
<td>MsgBox</td>
<td>Beep: Yes</td>
<td>If there is no current supplier on the Suppliers form, display a message.</td>
</tr>
<tr>
<td></td>
<td>MsgBox</td>
<td>Type: None</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MsgBox</td>
<td>Title: Select a Supplier</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GoToControl</td>
<td>Control Name: CompanyName</td>
<td>Move focus to the CompanyName control.</td>
</tr>
<tr>
<td></td>
<td>StopMacro</td>
<td>Form Name: Product List</td>
<td>Stop the macro.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>View: Datasheet</td>
<td>Open the Product</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Filter Name:</td>
<td></td>
</tr>
</tbody>
</table>
OpenForm

Where Condition: [SupplierID] = [Forms]![Suppliers]![SupplierID]

Data Mode: **Read Only**

Window Mode: **Normal**

MoveSize

Right: 0.7799"  
Down: 1.8"

List form and show the current supplier's products.

Position the Product List form in the lower right of the Suppliers form.
Maximize Action

You can use the Maximize action to enlarge the active window so that it fills the Microsoft Access window. This action will allow you to see as much of the object in the active window as possible.

**Note** This action cannot be applied to code windows in the Visual Basic Editor. For information about how to affect code windows, see the `WindowState` property topic.
Setting

The Maximize action doesn't have any arguments.
Remarks

This action has the same effect as clicking the **Maximize** button in the window's upper-right corner or clicking **Maximize** on the window's **Control** menu.

You can use the Restore action to restore a maximized window to its previous size.

Tips

- You may need to use the SelectObject action if the window you want to maximize isn't the active window.
- When you maximize a window in Access, all other windows are also maximized when you open them or switch to them. However, pop-up forms aren't maximized. If you want a form to maintain its size when other windows are maximized, set its **PopUp** property to **Yes**.

To run the Maximize action in Visual Basic, use the **Maximize** method of the **DoCmd** object.
Minimize Action

You can use the Minimize action to reduce the active window to a small title bar at the bottom of the Microsoft Access window.

**Note** This action cannot be applied to code windows in the Visual Basic Editor. For information about how to affect code windows, see the *WindowState* property topic.
Setting

The Minimize action doesn't have any arguments.
Remarks

You can use this action to remove a window from the screen while leaving the object open. You can also use this action to open an object without displaying its window. To display the object, use the SelectObject action with either the Maximize or Restore action. The Restore action restores a minimized window to its previous size.

The Minimize action has the same effect as clicking the Minimize button in the window's upper-right corner or clicking Minimize on the window's Control menu.

Tips

- You may first need to use the SelectObject action if the window you want to minimize isn't the active window.
- To minimize the Database window, use the SelectObject action with the In Database Window argument set to Yes and then use the Minimize action.
- You can use the Hide command on the Window menu to hide the active window. Instead of being reduced to an icon, the window becomes invisible. The Unhide command on the Window menu will make the window reappear. You can use the RunCommand action to carry out either of these commands from a macro.
- You can also use the SetValue action to set a form's Visible property to hide or show the form's window.

To run the Minimize action in Visual Basic, use the Minimize method of the DoCmd object.
**Restore Action**

You can use the Restore action to restore a maximized or minimized window to its previous size.

**Note** This action cannot be applied to code windows in the Visual Basic Editor. For information about how to affect code windows, see the `WindowState` property topic.
Setting

The Restore action doesn't have any arguments.
Remarks

This action works on the selected object. If an object has been minimized, you can first select it by using the SelectObject action and then restore it to its previous size by using the Restore action.

You can use the MoveSize action to move or size a window that you've restored.

The Restore action has the same effect as clicking the Restore button in the window's upper-right corner or clicking the Restore command on the window's Control menu.

To run the Restore action in Visual Basic, use the Restore method of the DoCmd object.
DEExecute Statement

You can use the **DDEExecute** statement to send a command from a client application to a server application over an open **dynamic data exchange (DDE)** channel.

For example, suppose you've opened a DDE channel in Microsoft Access to transfer text data from a Microsoft Excel spreadsheet into a Microsoft Access database. Use the **DDEExecute** statement to send the **New** command to Microsoft Excel to specify that you wish to open a new spreadsheet. In this example, Microsoft Access acts as the client application, and Microsoft Excel acts as the server application.

**Syntax**

**DDEExecute**(channum, command)

The **DDEExecute** statement has the following arguments.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>channum</td>
<td>A channel number, the long integer returned by the <strong>DDEInitiate</strong> function.</td>
</tr>
<tr>
<td>command</td>
<td>A string expression specifying a command recognized by the server application. Check the server application's documentation for a list of these commands.</td>
</tr>
</tbody>
</table>

**Remarks**

The value of the **command** argument depends on the application and topic specified when the channel indicated by the **channum** argument is opened. An error occurs if the **channum** argument isn't an integer corresponding to an open channel or if the other application can't carry out the specified command.

From Visual Basic, you can use the **DDEExecute** statement only to send commands to another application. For information on sending commands to Microsoft Access from another application, see **Use Microsoft Access as a DDE Server**.
Tip

If you need to manipulate another application's objects from Microsoft Access, you may want to consider using Automation.
DDEPoke Statement

You can use the **DDEPoke** statement to supply text data from a client application to a server application over an open **dynamic data exchange (DDE)** channel.

For example, if you have an open DDE channel between Microsoft Access and Microsoft Excel, you can use the **DDEPoke** statement to transfer text from a Microsoft Access database to a Microsoft Excel spreadsheet. In this example, Microsoft Access acts as the client application, and Microsoft Excel acts as the server application.

**DDEPoke** *(channum, item, data)*

The **DDEPoke** statement has the following arguments.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>channum</td>
<td>A channel number, an integer returned by the <strong>DDEInitiate</strong> function. A string expression that's the name of a data item recognized by the application specified by the <strong>DDEInitiate</strong> function. Check the application's documentation for a list of possible items.</td>
</tr>
<tr>
<td>item</td>
<td>A string containing the data to be supplied to the other application.</td>
</tr>
</tbody>
</table>
Remarks

The value of the *item* argument depends on the application and *topic* specified when the channel indicated by the *channum* argument is opened. For example, the *item* argument may be a range of cells in a Microsoft Excel spreadsheet.

The string contained in the *data* argument must be an alphanumeric text string. No other formats are supported. For example, the *data* argument could be a number to fill a cell in a specified range in an Excel worksheet.

If the *channum* argument isn't an integer corresponding to an open channel or if the other application doesn't recognize or accept the specified data, a run-time error occurs.

Tip

If you need to manipulate another application's objects from Microsoft Access, you may want to consider using Automation.
DDETerminateAll Statement

You can use the DDETerminateAll statement to close all open dynamic data exchange (DDE) channels.

For example, suppose you've opened two DDE channels between Microsoft Excel and Microsoft Access, one to retrieve system information about Microsoft Excel and one to transfer data. You can use the DDETerminateAll statement to close both channels simultaneously.

DDETerminateAll
Remarks

Using the **DDETerminateAll** statement is equivalent to executing a **DDETerminate** statement for each open **channel number**. Like the **DDETerminate** statement, the **DDETerminateAll** statement has no effect on active DDE link **expressions** in fields on forms or reports.

If there are no DDE channels open, the **DDETerminateAll** statement runs without causing a **run-time error**.

Tips

- If you interrupt a procedure that performs DDE, you may inadvertently leave channels open. To avoid exhausting system resources, use the **DDETerminateAll** statement in your code or from the Immediate (lower) pane of the Debug window while debugging code that performs DDE.
- If you need to manipulate another application's objects from Microsoft Access, you may want to consider using Automation.
Set References to Type Libraries

When you set a reference to another application's type library, you can use the objects supplied by that application in your code. For example, if you set a reference from Microsoft Access to the Microsoft Excel library, you can then use Microsoft Excel objects through Automation (formerly called OLE Automation). If you set a reference to a Visual Basic project in another Microsoft Access database, you can call its public procedures. If you set a reference to an ActiveX control, you can use that control on Microsoft Access forms.

You can set a reference from Microsoft Access while the Microsoft Visual Basic Editor is open, or you can set a reference in Visual Basic code.
Setting a Reference from Microsoft Access

To set a reference to an application's type library:

1. On the **Tools** menu, click **References**. The **References** command on the **Tools** menu is available only when a Module window is open and active in **Design view**.
2. Select the check boxes for those applications whose type libraries you want to reference.
Setting a Reference from Visual Basic

To set a reference from Visual Basic, you create a new Reference object representing the desired reference. The References collection contains all currently set references.

To create a new Reference object, use either the AddFromFile or AddFromGUID method of the References collection. To remove a Reference object, use the Remove method.
Advantages of Setting References

Your Automation code will run faster if you set a reference to another application's type library before you work with its objects. If you've set a reference, you can declare an object variable representing an object in the other application as its most specific type. For example, if you're writing code to work with Microsoft Excel objects, you can declare an object variable of type `Excel.Application` by using the following syntax only if you've created a reference to the Microsoft Excel type library:

```
Dim appXL As New Excel.Application
```

If you haven't set a reference to the Microsoft Excel type library, you must declare the variable as a generic variable of type `Object`. The following code runs more slowly:

```
Dim appXL As Object
```

Additionally, if you set a reference to an application's type library, all of its objects, as well as their methods and properties, are listed in the `Object Browser`. This makes it easy to determine what properties and methods are available to each object.

Since Microsoft Access is an `COM component` that supports Automation, you can also set a reference to its type library from another application and work with Microsoft Access objects from that application.
Domain Aggregate Functions

Aggregate functions provide statistical information about sets of records (a domain). For example, you can use an aggregate function to count the number of records in a particular set of records or to determine the average of values in a particular field.

The two types of aggregate functions, domain aggregate functions and SQL aggregate functions, provide similar functionality but are used in different situations. The SQL aggregate functions can be included in the syntax of an SQL statement but can't be called directly from Visual Basic. Conversely, the domain aggregate functions can be called directly from Visual Basic code. They can also be included in an SQL statement, but an SQL aggregate function is generally more efficient.

If you are performing statistical calculations from within code, you must use the domain aggregate functions. You can also use the domain aggregate functions to specify criteria, update values, or create calculated fields in a query expression. You can use either the SQL aggregate or domain aggregate functions in a calculated control on a form or report.

The domain aggregate functions include:

- **DAvg Function**
- **DCount Function**
- **DLookup Function**
- **DFirst, DLast Functions**
- **DMin, DMax Functions**
- **DStDev, DStDevP Functions**
- **DSum Function**
- **DVar, DVarP Functions**
**UniqueValues Property**

You can use the **UniqueValues** property when you want to omit records that contain duplicate data in the fields displayed in **Datasheet view**. For example, if a query's output includes more than one field, the combination of values from all fields must be unique for a given record to be included in the results.

**Note** The **UniqueValues** property applies only to append and make-table action queries and select queries.
Setting

The **UniqueValues** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>Displays only the records in which the values of all fields displayed in Datasheet view are unique.</td>
</tr>
<tr>
<td>No</td>
<td>(Default) Displays all records.</td>
</tr>
</tbody>
</table>

You can set the **UniqueValues** property in the query's property sheet or in SQL view of the **Query window**.

**Note** You can set this property when you create a new query by using an SQL statement. The DISTINCT predicate corresponds to the **UniqueValues** property setting. The DISTINCTROW predicate corresponds to the **UniqueRecords** property setting.
Remarks

When you set the **UniqueValues** property to Yes, the results of the query aren't updatable and won't reflect subsequent changes made by other users.

The **UniqueValues** and **UniqueRecords** properties are related in that only one of them can be set to Yes at a time. When you set the **UniqueValues** property to Yes, for example, Microsoft Access automatically sets the **UniqueRecords** property to No. You can, however, set both of them to No. When both properties are set to No, all records are returned.

**Tip**

If you want to count the number of instances of a value in a field, create a [totals query](#).
Example

The SELECT statement in this example returns a list of the countries/regions in which there are customers. Because there may be many customers in each country/region, many records could have the same country/region in the Customers table. However, each country/region is represented only once in the query results.

This example uses the Customers table, which contains the following data.

<table>
<thead>
<tr>
<th>Country/Region</th>
<th>Company name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brazil</td>
<td>Familia Arquibaldo</td>
</tr>
<tr>
<td>Brazil</td>
<td>Gourmet Lanchonettes</td>
</tr>
<tr>
<td>Brazil</td>
<td>Hanari Carnes</td>
</tr>
<tr>
<td>France</td>
<td>Du monde entier</td>
</tr>
<tr>
<td>France</td>
<td>Folies gourmandes</td>
</tr>
<tr>
<td>Germany</td>
<td>Frankenversand</td>
</tr>
<tr>
<td>Ireland</td>
<td>Hungry Owl All-Night Grocers</td>
</tr>
</tbody>
</table>

This SQL statement returns the countries/regions in the following table:

```
SELECT DISTINCT Customers.Country
FROM Customers;
```

**Countries/Regions returned**

Brazil
France
Germany
Ireland
Set Form, Report, and Control Properties in Visual Basic

Form, Report, and Control objects are Microsoft Access objects. You can set properties for these objects from within a Sub, Function, or event procedure. You can also set properties for form and report sections.
To set a property of a form or report

Refer to the individual form or report within the **Forms** or **Reports** collection, followed by the name of the property and its value. For example, to set the **Visible** property of the Customers form to **True** (−1), use the following line of code:

```vba
Forms!Customers.Visible = True
```

You can also set a property of a form or report from within the object's module by using the object's **Me** property. Code that uses the **Me** property executes faster than code that uses a fully qualified object name. For example, to set the **RecordSource** property of the Customers form to an **SQL statement** that returns all records with a CompanyName field entry beginning with "A" from within the Customers form module, use the following line of code:

```vba
Me.RecordSource = "SELECT * FROM Customers " 
& "WHERE CompanyName Like 'A*'
```
To set a property of a control

Refer to the control in the Controls collection of the Form or Report object on which it resides. You can refer to the Controls collection either implicitly or explicitly, but the code executes faster if you use an implicit reference. The following examples set the Visible property of a text box called CustomerID on the Customers form:

' Faster method.
Me!CustomerID.Visible = True

' Slower method.
Forms!Customers.Controls!CustomerID.Visible = True

The fastest way to set a property of a control is from within an object's module by using the object's Me property. For example, you can use the following code to toggle the Visible property of a text box called CustomerID on the Customers form:

With Me!CustomerID
    .Visible = Not .Visible
End With
To set a property of a form or report section

Refer to the form or report within the Forms or Reports collection, followed by the Section property and the integer or constant that identifies the section. The following examples set the Visible property of the page header section of the Customers form to False:

Forms!Customers.Section(3).Visible = False

Me!Section(acPageHeader).Visible = False

Notes

- For each property you want to set, you can look up the property in the Help index to find information about:
  - Whether you can set the property from Visual Basic.
  - Views in which you can set the property. Not all properties can be set in all views. For example, you can set a form's BorderStyle property only in form Design view.
  - Which values you should use to set the property. You often use different settings when you set a property in Visual Basic instead of in the property sheet. For example, if the property settings are selections from a list, you must use the value or numeric equivalent for each selection.
- To set default properties for controls from Visual Basic, use the DefaultControl property.
Set Properties of ActiveX Data Objects in Visual Basic

ActiveX Data Objects (ADO) enable you to manipulate the structure of your database and the data it contains from Visual Basic. Many ADO objects correspond to objects that you see in your database— for example, a Table object corresponds to a Microsoft Access table. A Field object corresponds to a field in a table.

Most of the properties you can set for ADO objects are ADO properties. These properties are defined by the Microsoft Jet database engine and are set the same way in any application that includes the Jet database engine. Some properties that you can set for ADO objects are defined by Microsoft Access, and aren't automatically recognized by the Jet database engine. How you set properties for ADO objects depends on whether a property is defined by the Jet database engine or by Microsoft Access.
Setting ADO Properties for ADO Objects

To set a property that's defined by the Jet database engine, refer to the object in the ADO hierarchy. The easiest and fastest way to do this is to create object variables that represent the different objects you need to work with, and refer to the object variables in subsequent steps in your code. For example, the following code creates a new TableDef object and sets its Name property:

```vbnet
Dim tbl As New ADOX.Table
Dim cnn As ADODB.Connection
Set cnn = CurrentProject.Connection
tbl.Name = "Contacts"
```
Setting Microsoft Access Properties for ADO Objects

When you set a property that's defined by Microsoft Access, but applies to a ADO object, the Jet database engine doesn't automatically recognize the property as a valid property. The first time you set the property, you must create the property and append it to the Properties collection of the object to which it applies. Once the property is in the Properties collection, it can be set in the same manner as any ADO property.

If the property is set for the first time in the user interface, it's automatically added to the Properties collection, and you can set it normally.

When writing procedures to set properties defined by Microsoft Access, you should include error-handling code to verify that the property you are setting already exists in the Properties collection. See the Help topic about the Add method or the individual property topic for more information.

Keep in mind that when you create the property, you must correctly specify its Type property before you append it to the Properties collection. You can determine the Type property based on the information in the Settings section of the Help topic for the individual property. The following table provides some guidelines for determining the setting of the Type property.

<table>
<thead>
<tr>
<th>If the property setting is</th>
<th>Then the Type property setting should be</th>
</tr>
</thead>
<tbody>
<tr>
<td>A string</td>
<td>adLongVarWChar or adVarWChar</td>
</tr>
<tr>
<td>True/False</td>
<td>adBoolean</td>
</tr>
<tr>
<td>An integer</td>
<td>adInteger</td>
</tr>
</tbody>
</table>

The following table lists some Microsoft Access–defined properties that apply to ADO objects.

<table>
<thead>
<tr>
<th>ADO object</th>
<th>Microsoft Access–defined properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>AppTitle, AppIcon,</td>
<td></td>
</tr>
<tr>
<td>StartupShowDBWindow,</td>
<td></td>
</tr>
<tr>
<td>StartupShowStatusBar,</td>
<td></td>
</tr>
<tr>
<td>AllowShortcutMenus,</td>
<td></td>
</tr>
<tr>
<td>AllowFullMenus,</td>
<td></td>
</tr>
<tr>
<td>AllowBuiltInToolbars,</td>
<td></td>
</tr>
<tr>
<td>AllowToolbarChanges,</td>
<td></td>
</tr>
<tr>
<td>AllowBreakIntoCode,</td>
<td></td>
</tr>
</tbody>
</table>
Table
AllowSpecialKeys, Replicable, ReplicationConflictFunction
DatasheetBackColor, DatasheetCellsEffect,
DatasheetFontHeight, DatasheetFontItalic,
DatasheetFontName, DatasheetFontUnderline,
DatasheetFontWeight, DatasheetForeColor,
DatasheetGridlinesBehavior, DatasheetGridlinesColor,
Description, FrozenColumns, RowHeight, ShowGrid
QueryDef
DatasheetBackColor, DatasheetCellsEffect,
DatasheetFontHeight, DatasheetFontItalic,
DatasheetFontName, DatasheetFontUnderline,
DatasheetFontWeight, DatasheetForeColor,
DatasheetGridlinesBehavior, DatasheetGridlinesColor,
Description, FailOnError, FrozenColumns, LogMessages,
MaxRecords, RecordLocks, RowHeight, ShowGrid,
UseTransaction
Field
Caption, ColumnHidden, ColumnOrder, ColumnWidth,
DecimalPlaces, Description, Format, InputMask
Set Properties of Data Access Objects in Visual Basic

Data Access Objects (DAO) enable you to manipulate the structure of your database and the data it contains from Visual Basic. Many DAO objects correspond to objects that you see in your database— for example, a TableDef object corresponds to a Microsoft Access table. A Field object corresponds to a field in a table.

Most of the properties you can set for DAO objects are DAO properties. These properties are defined by the Microsoft Jet database engine and are set the same way in any application that includes the Jet database engine. Some properties that you can set for DAO objects are defined by Microsoft Access, and aren't automatically recognized by the Jet database engine. How you set properties for DAO objects depends on whether a property is defined by the Jet database engine or by Microsoft Access.
Setting DAO Properties for DAO Objects

To set a property that's defined by the Jet database engine, refer to the object in the DAO hierarchy. The easiest and fastest way to do this is to create object variables that represent the different objects you need to work with, and refer to the object variables in subsequent steps in your code. For example, the following code creates a new `TableDef` object and sets its `Name` property:

```vbs
Dim dbs As DAO.Database
Dim tdf As DAO.TableDef
Set dbs = CurrentDb
Set tdf = dbs.CreateTableDef
tdf.Name = "Contacts"
```
Setting Microsoft Access Properties for DAO Objects

When you set a property that's defined by Microsoft Access, but applies to a DAO object, the Jet database engine doesn't automatically recognize the property as a valid property. The first time you set the property, you must create the property and append it to the Properties collection of the object to which it applies. Once the property is in the Properties collection, it can be set in the same manner as any DAO property.

If the property is set for the first time in the user interface, it's automatically added to the Properties collection, and you can set it normally.

When writing procedures to set properties defined by Microsoft Access, you should include error-handling code to verify that the property you are setting already exists in the Properties collection. See the Help topic about the CreateProperty method or the individual property topic for more information.

Keep in mind that when you create the property, you must correctly specify its Type property before you append it to the Properties collection. You can determine the Type property based on the information in the Settings section of the Help topic for the individual property. The following table provides some guidelines for determining the setting of the Type property.

If the property setting is Then the Type property setting should be
A string dbText
True/False dbBoolean
An integer dbInteger

The following table lists some Microsoft Access–defined properties that apply to DAO objects.

<table>
<thead>
<tr>
<th>DAO object</th>
<th>Microsoft Access–defined properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database</td>
<td>AppTitle, AppIcon, StartupShowDBWindow, StartupShowStatusBar, AllowShortcutMenus, AllowFullMenus, AllowBuiltInToolbars, AllowToolbarChanges, AllowBreakIntoCode, AllowSpecialKeys, Replicable, ReplicationConflictFunction</td>
</tr>
</tbody>
</table>
Title, Subject, Author, Manager, Company, Category, Keywords, Comments, Hyperlink Base

SummaryInfo Container
(See the Summary tab of the DatabaseName Properties dialog box, available by clicking Database Properties on the File menu.)

UserDefined Container
(See the Summary tab of the DatabaseName Properties dialog box, available by clicking Database Properties on the File menu.)

TableDef
DatasheetBackColor, DatasheetCellsEffect, DatasheetFontHeight, DatasheetFontItalic, DatasheetFontName, DatasheetFontUnderline, DatasheetFontWeight, DatasheetForeColor, DatasheetGridlinesBehavior, DatasheetGridlinesColor, Description, FrozenColumns, RowHeight, ShowGrid

QueryDef

Field
Caption, ColumnHidden, ColumnOrder, ColumnWidth, DecimalPlaces, Description, Format, InputMask
Set Data Access Page Properties in Visual Basic

**DataAccessPage** objects are [Microsoft Access objects](https://docs.microsoft.com/en-us/previous-versions/dotnet-framework/library/ctzg897f). You can set properties for these objects from within a **Sub**, **Function**, or **event** procedure.
To set a property of a data access page

Refer to the individual data access page within the **DataAccessPages** collection, followed by the name of the property and its value. For example, to set the **Visible** property of the Customers data access page to **True** (–1), use the following line of code:

```
DataAccessPages!Customers.Visible = True
```

**Notes**

- For each property you want to set, you can look up the property in the Help index to find information about:
  - Whether you can set the property from Visual Basic.
  - Which values you should use to set the property. You often use different settings when you set a property in Visual Basic instead of in the property sheet. For example, if the property settings are selections from a list, you must use the value or numeric equivalent for each selection.
Image Hyperlink Control

The image hyperlink control can add an unbound image to a data access page.

Tool:
Remarks

Use the image hyperlink control to add an image to a data access page that, when clicked, displays another Web page from your hard drive, the Web, or another location. When you create an image hyperlink control, you select the image to display on the page and the address of the file to jump to.

- The **Insert Picture** dialog box displays a list of pictures that can be selected as the image.
- The **Insert Hyperlink** dialog box allows the user to select the file or web page that will be linked to the image control.

In **Page view**, as the pointer passes over the image, the pointer changes to a hand, indicating that the image is a link that you can click to go to another page. You can also define ScreenTips and alternate text for the image.
Office Chart Control

The Office chart control can add a chart to a data access page.

Tool:
Remarks

You can use Microsoft Office Chart, a Microsoft Office Web Component, to create dynamic, interactive charts in a data access page that you can make available on Web sites for viewing in a browser. For example, you can create a chart using data in a table that stores sales figures. When you update the sales figures, the chart updates—so users can always see the latest information on their Web pages.

Microsoft Internet Explorer 5 or later is required to create a chart on a data access page, but users with Internet Explorer version 4.x or later can view the chart and see the chart updated in response to changes to data.
Office Spreadsheet Control

The Office spreadsheet control can add a spreadsheet with some of the features of a Microsoft Excel spreadsheet to a data access page.

Tool:
**Remarks**

You can add a spreadsheet control to a data access page to provide some of the same capabilities you have in a Microsoft Excel worksheet. You can enter values, add formulas, apply filters, and so on. Use the procedure below to create a spreadsheet in which you enter raw data, or import data from a Web page or text file. If within the spreadsheet you want to use data from other controls on the data access page, you need to refer to those controls in the appropriate spreadsheet cells.
Understanding ActiveX Controls
(Data Access Pages)

Some of the content in this topic may not be applicable to some languages.

In addition to the built-in controls that appear in the toolbox, Microsoft Access supports ActiveX controls (formerly known as custom or OLE controls). An ActiveX control, like a built-in control, is an object that you place on a form to enable or enhance a user's interaction with an application. ActiveX controls have events and can be incorporated into other controls. These controls have an .ocx file name extension. The Calendar control is an example of an ActiveX control.
**DataType Property**

You can use the **DataType** property to specify the type of data stored in a table field. Each field can store data consisting of only a single data type.
Setting

The **DataType** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Type of data</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text</td>
<td>(Default) Text or combinations of text and numbers, as well as numbers that don't require calculations, such as phone numbers.</td>
<td>Up to 255 characters or the length set by the <strong>FieldSize</strong> property, whichever is less. Microsoft Access does not reserve space for unused portions of a text field.</td>
</tr>
<tr>
<td>Memo</td>
<td>Lengthy text or combinations of text and numbers.</td>
<td>Up to 65,535 characters. (If the Memo field is manipulated through DAO and only text and numbers [not binary data] will be stored in it, then the size of the Memo field is limited by the size of the database.)</td>
</tr>
<tr>
<td>Number</td>
<td>Numeric data used in mathematical calculations. For more information on how to set the specific Number type, see the <strong>FieldSize</strong> property topic.</td>
<td>1, 2, 4, or 8 bytes (16 bytes if the <strong>FieldSize</strong> property is set to Replication ID).</td>
</tr>
<tr>
<td>Date/Time</td>
<td>Date and time values for the years 100 through 9999.</td>
<td>8 bytes.</td>
</tr>
<tr>
<td>Currency</td>
<td>Currency values and numeric data used in mathematical calculations involving data with one to four decimal places. Accurate to 15 digits on the left side of the decimal separator and to 4 digits on the right side. A unique sequential (incremented by 1) number or random number assigned by</td>
<td>8 bytes.</td>
</tr>
</tbody>
</table>
AutoNumber
Microsoft Access whenever a new record is added to a table. AutoNumber fields can't be updated. For more information, see the NewValues property topic.

Yes/No
Yes and No values and fields that contain only one of two values (Yes/No, True/False, or On/Off).

OLE Object
An object (such as a Microsoft Excel spreadsheet, a Microsoft Word document, graphics, sounds, or other binary data) linked to or embedded in a Microsoft Access table.

Hyperlink
Text or combinations of text and numbers stored as text and used as a hyperlink address. A hyperlink address can have up to three parts:

- text to display— the text that appears in a field or control.
- address— the path to a file (UNC path) or page (URL).
- subaddress— a location within the file or page.

screentip— the text displayed as a tooltip.

The easiest way to insert a hyperlink address in a field or control is to click Hyperlink on the Insert menu.

Lookups
Creates a field that allows you to choose a value from another table or from a list of values by using a list box or combo box. Clicking this option generates a value for the field. The same size as the primary key field used to...
Wizard starts the Lookup Wizard, which creates a Lookup field. After you complete the wizard, Microsoft Access sets the data type based on the values selected in the wizard.

You can set this property only in the upper portion of table Design view.

In Visual Basic, you can use the ADO Type property to set a field's data type before appending it to the Fields collection.
Remarks

Memo, Hyperlink, and OLE Object fields can't be indexed.

Tip

Use the Currency data type for a field requiring many calculations involving data with one to four decimal places. **Single** and **Double** data type fields require floating-point calculation. The Currency data type uses a faster fixed-point calculation.

**Caution** Changing a field's data type after you enter data in a table causes a potentially lengthy process of data conversion when you save the table. If the data type in a field conflicts with a changed **DataType** property setting, you may lose some data.

Set the **Format** property to specify a predefined **display format** for Number, Date/Time, Currency, and Yes/No data types.
AllowZeroLength Property

You can use the **AllowZeroLength** property to specify whether a zero-length string ("") is a valid entry in a table field.

**Note** The **AllowZeroLength** property applies only to Text, Memo, and Hyperlink table fields.
Setting

The **AllowZeroLength** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Visual Basic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>True</td>
<td>A zero-length string is a valid entry.</td>
</tr>
<tr>
<td>No</td>
<td>False</td>
<td>(Default) A zero-length string is an invalid entry.</td>
</tr>
</tbody>
</table>

You can set this property by using the table's property sheet or [Visual Basic](#).

**Note**  To access a field's **AllowZeroLength** property by using Visual Basic, use the DAO **AllowZeroLength** property or the ADO **Column.Properties**("Set OLEDB:Allow Zero Length") property.
Remarks

If you want Microsoft Access to store a zero-length string instead of a Null value when you leave a field blank, set both the AllowZeroLength and Required properties to Yes.

The following table shows the results of combining the settings of the AllowZeroLength and Required properties.

<table>
<thead>
<tr>
<th>AllowZeroLength</th>
<th>Required</th>
<th>User’s action</th>
<th>Value stored</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>No</td>
<td>Presses ENTER</td>
<td>Null</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Presses SPACEBAR</td>
<td>Null</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Enters a zero-length string (not allowed)</td>
<td></td>
</tr>
<tr>
<td>Yes</td>
<td>No</td>
<td>Presses ENTER</td>
<td>Null</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Presses SPACEBAR</td>
<td>Null</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Enters a zero-length string Zero-length string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Yes</td>
<td>Presses ENTER</td>
<td>(not allowed)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Presses SPACEBAR</td>
<td>(not allowed)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Enters a zero-length string (not allowed)</td>
<td></td>
</tr>
<tr>
<td>Yes</td>
<td>Yes</td>
<td>Presses ENTER</td>
<td>Zero-length string</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Presses SPACEBAR</td>
<td>Zero-length string</td>
</tr>
</tbody>
</table>

Tip

You can use the Format property to distinguish between the display of a Null value and a zero-length string. For example, the string "None" can be displayed when a zero-length string is entered.

The AllowZeroLength property works independently of the Required property. The Required property determines only whether a Null value is valid for the field. If the AllowZeroLength property is set to Yes, a zero-length string will be a valid value for the field regardless of the setting of the Required property.
AllowFullMenus Property

You can use the **AllowFullMenus** property to specify whether or not full Microsoft Access built-in menus will be available when the application database opens. For example, you can use the **AllowFullMenus** property to disable menu items that give users the ability to modify table, form, query, or report structures.
Setting

The AllowFullMenus property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>Display the full built-in menus at startup.</td>
</tr>
<tr>
<td>False</td>
<td>Don't display full built-in menus at startup.</td>
</tr>
</tbody>
</table>

The easiest way to set this property is by using the Allow Full Menus option in the Startup dialog box, available by clicking Startup on the Tools menu. You can also set this property by using a macro or Visual Basic.

To set the AllowFullMenus property by using a macro or Visual Basic, you must first either set the property in the Startup dialog box once or create the property in the following ways:

- In a Microsoft Access database (.mdb), you can add it by using the CreateProperty method and append it to the Properties collection of the Database object.
- In a Microsoft Access project (.adp), you can add it to the AccessObjectProperties collection of the CurrentProject object by using the Add method.
Remarks

If you set this property to **False**, a predefined subset of the full built-in menus is displayed in your database. This set of menus doesn't include menus and commands that enable users to change the design of database objects.

Setting this property to **False** also disables the toolbar buttons that correspond to the disabled menu items. However, **shortcut menus** aren't affected (you can still change some design features by using shortcut menu commands). If you don't want users to have access to the commands on the shortcut menus, you can set the **AllowShortcutMenus** property to **False**.

This property's setting doesn't take effect until the next time the database opens.
StartupShortcutMenuBar Property

You can use the StartupShortcutMenuBar property to specify a custom shortcut menu to use as the global shortcut menu for your application. You can also specify a menu bar macro that displays the custom shortcut menu you want to use as the global shortcut menu.
Setting

The `StartupShortcutMenuBar` property is a string expression that's the name of a custom shortcut menu or a menu bar macro that displays a custom shortcut menu.

The easiest way to set this property is by using the Shortcut Menu Bar option in the Startup dialog box, available by clicking Startup on the Tools menu. You can also set this property by using a macro or Visual Basic.

To set the `StartupShortcutMenuBar` property by using a macro or Visual Basic, you must first either set the property in the Startup dialog box once or create the property in the following ways:

- In a Microsoft Access database (.mdb), you can add it by using the CreateProperty method and append it to the Properties collection of the Database object.
- In a Microsoft Access project (.adp), you can add it to the AccessObjectProperties collection of the CurrentProject object by using the Add method.
Remarks

If you are setting the `StartupShortcutMenuBar` property, you shouldn't use a SetValue action in the AutoExec macro to set the `ShortcutMenuBar` property for the `Application` object. Since Microsoft Access runs the AutoExec macro after it parses the startup properties, the global shortcut menu set in the AutoExec macro would replace the shortcut menu set in the `StartupShortcutMenuBar` property.

You can also create custom shortcut menus by using the `ShortcutMenuBar` property for forms, reports, and form controls. These custom shortcut menus are displayed when you right-click a specific form, report, or form control, and replace the global shortcut menu in those cases.

If this property is blank, Microsoft Access displays the built-in shortcut menus.

Setting this property has the same effect as setting the `ShortcutMenuBar` property of the `Application` object (except the `ShortcutMenuBar` property setting takes effect immediately).

This property's setting doesn't take effect until the next time the application database opens.
UniqueRecords Property

You can use the **UniqueRecords** property to specify whether to return only unique records based on all fields in the underlying data source, not just those fields present in the query itself.

**Note**  The **UniqueRecords** property applies only to [append](#) and [make-table](#) action queries and [select](#) queries.
Setting

The **UniqueRecords** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>Doesn't return duplicate records.</td>
</tr>
<tr>
<td>No</td>
<td>(Default) Returns duplicate records.</td>
</tr>
</tbody>
</table>

You can set the **UniqueRecords** property in the query's property sheet or in **SQL view** of the **Query window**.

**Note** You set this property when you create a new query by using an **SQL statement**. The DISTINCTROW predicate corresponds to the **UniqueRecords** property setting. The DISTINCT predicate corresponds to the **UniqueValues** property setting.
Remarks

You can use the **UniqueRecords** property when you want to omit data based on entire duplicate records, not just duplicate fields. Microsoft Access considers a record to be unique as long as the value in one field in the record differs from the value in the same field in another record.

The **UniqueRecords** property has an effect only when you use more than one table in the query and select fields from the tables used in the query. The **UniqueRecords** property is ignored if the query includes only one table.

The **UniqueRecords** and **UniqueValues** properties are related in that only one of them can be set to Yes at a time. When you set **UniqueRecords** to Yes, for example, Microsoft Access automatically sets **UniqueValues** to No. You can, however, set both of them to No. When both properties are set to No, all records are returned.
Example

The query in this example returns a list of customers from the Customers table who have at least one order in the Orders table.
# Customers Table

<table>
<thead>
<tr>
<th>Company name</th>
<th>Customer ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ernst Handel</td>
<td>ERNSH</td>
</tr>
<tr>
<td>Familia Arquibaldo</td>
<td>FAMIA</td>
</tr>
<tr>
<td>FISSA Fabrica Inter. Salchichas S.A. FISSA</td>
<td>FISSA</td>
</tr>
<tr>
<td>Folies gourmandes</td>
<td>FOLIG</td>
</tr>
</tbody>
</table>
Orders Table

<table>
<thead>
<tr>
<th>Customer ID</th>
<th>Order ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERNSH</td>
<td>10698</td>
</tr>
<tr>
<td>FAMIA</td>
<td>10512</td>
</tr>
<tr>
<td>FAMIA</td>
<td>10725</td>
</tr>
<tr>
<td>FOLIG</td>
<td>10763</td>
</tr>
<tr>
<td>FOLIG</td>
<td>10408</td>
</tr>
</tbody>
</table>

The following SQL statement returns the customer names in the following table:

```
SELECT DISTINCTROW Customers.CompanyName, Customers.CustomerID
FROM Customers INNER JOIN Orders
ON Customers.CustomerID = Orders.CustomerID;
```

Customers returned Customer ID

- Ernst Handel       ERNSH
- Familia Arquibaldo FAMIA
- Folies gourmandes  FOLIG
**NewValues Property**

You can use the `NewValues` property to specify how `AutoNumber` fields increment when new records are added to a table.

**Note** The `NewValues` property applies only to `AutoNumber` fields.
Setting

The **NewValues** property uses the following settings.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Increment</td>
<td>(Default) AutoNumber field values increment by 1 for new records.</td>
</tr>
<tr>
<td>Random</td>
<td>AutoNumber field values are assigned a random <strong>Long Integer</strong> value for new records.</td>
</tr>
</tbody>
</table>

You can set this property in the table's property sheet in table Design view by clicking the **General** tab in the Field Properties section.
Remarks

When you replicate a database, AutoNumber field settings are set to Random to ensure that new records entered in different replicas will have unique values.