Microsoft Dynamics SL

Object Model Reference Guide

Release 2015

Disclaimer

This document is provided “as-is”. Information and views expressed in this document, including URL and other Internet Web site references, may change without notice. You bear the risk of using it.

Some examples are for illustration only and are fictitious. No real association is intended or inferred.

This document does not provide you with any legal rights to any intellectual property in any Microsoft product. You may copy and use this document for your internal, reference purposes only.

Sample Code Warranty disclaimer

Microsoft Corporation disclaims any warranty regarding the sample code contained in this documentation, including the warranties of merchantability and fitness for a particular purpose.

License agreement

Use of this software is covered by a license agreement provided with it. If you have any questions, please call the Customer Assistance Department at 800-456-0025 (in the United States or Canada) or +1-701-281-6500.

Copyright

© 2014 Microsoft Corporation. All rights reserved.

Publication Date

September 2014

Contents

Introduction 1

Who Should Use this Information ... 1
What Programmers Should Know ... 1

Reference 3

Button Control (SIVControl Object) ... 3
Cancel Method ... 5
Controls Collection (SIVApplication Object) ... 6
Constants ... 7
ControlType Property (SIVProperty Object) ... 9
Count Property ... 11
Delete Method ... 12
DisableObjectModel Statement (Microsoft SL SDK Applications) ... 14
Dispose Method ... 15
DSLDate Control (SIVControl Object) ... 16
EventLog Property ... 18
DisableObjectModel Statement (Microsoft SL SDK Applications) ... 20
ExposeCustomObject Statement (Microsoft SL SDK Applications) ... 21
First Method ... 22
GetBusinessDate Method ... 25
GetCurrencyIDs Method ... 27
GetCurrencyView Method ... 28
GetCustomizationLevel Method ... 29
GetCustomObject Method ... 31
GetEntityStatus Method ... 32
GetStatusBarText Method ... 34
InitializeMode Property ... 35
Item Property ... 36
KeyControls Collection (SIVApplication Object) ... 37
Label Control (SIVControl Object) ... 38
Last Method ... 40
Level Property (SIVProperty Object) ... 42
LevelNumber Property (SIVProperty Object) ... 44
Login Method ... 45
Logout Method ... 48
Message Event ... 50
Name Property (SIVControl Object) ... 55
Name Property (SIVProperty Object) ... 56
New Method ... 57
Next Method ... 59
Notes/Attachments Icon (NoteButton Control) ... 61
Previous Method ... 62
Properties Collection (SIVControl Object) ... 64
Quit Method ... 65
Relative Date Dialog ... 67
SAFCheck Control (SIVControl Object) ... 68
SAFCombo Control (SIVControl Object) ... 70
SAFContainer Control ... 72
SAFFloat Control (SIVControl Object) ... 73
SAFGrid Control ... 75
SAFInteger Control (SIVControl Object) ... 76

SAFMaskedText Control (SIVControl Object) 78
SAFOption Control (SIVControl Object) 80
SAFUpdate Control 82
Save Method 83
SetBusinessDate Method 86
SetCurrencyIDs Method 88
SetCurrencyView Method 90
SetCustomizationLevel Method 92
SetStatusBarText Method 95
SetTranCurrencyAndRate Method 96
SIVApplication Object 98
SIVControl Object 99
SIVControls Collection 101
SIVProperties Collection 103
SIVProperty Object 105
SIVToolbar Object 109
StartAppAndAutomate Function (Microsoft SL SDK Applications) 111
StartApplication Method 113
StartApplication_2 Method 116
StatusBar Control (Microsoft SL SDK Application) 118
SubFormDisplay Event 119
Value Property (SIVControl Object) 120
Value Property (SIVProperty Object) 123
Visible Property 125

Appendix A: Integrating with Microsoft Office 127

Appendix B: Information for Microsoft SL SDK Programmers 129

Getting a Handle to a Custom Object 130
Manipulating a Custom Object 130

Appendix C: Limitations 131

Appendix D: Advanced Tasks 133

Working with Troublesome Controls 133

Appendix E: Error Numbers 135

Cross-Reference Between Various Error-Numbering Schemes 144

Appendix F: Application Browser 147

Logging on with the Browser 147
Choosing an Application to Browse 148
Left and Right Browser Panes 149
Exercise: Browsing the Sample Application 149

Appendix G: Visual Basic .NET-Related Changes 151

Programming Interface Changes 151
Programmatic References 151
Enumerations for Unmanaged (Visual Basic 6.0) Clients Only 152
Default Properties 153
SIVControl Default Property 153
Deterministic Freeing of SIVToolbar and SIVApplication Objects 154
Error Handling 154
Support for WinForm Controls 157
Label Properties 157

CommandButton Properties 158
Support for Remaining WinForm Controls 159
Caption Support 161
Tag Support 161
Color Support 161
Font support 161
Support for COM Controls 162
Custom Object Support 162

Glossary of Terms 165

Index 169

Introduction

This documentation provides Microsoft® Visual Basic® programmers, other Windows programmers, and their managers with a complete reference to the object model of Microsoft Dynamics® SL.

Who Should Use this Information

This information is of interest to:

- *Visual Basic programmers* looking to automate Microsoft Dynamics SL through Visual Basic client applications that you will write. Visual Basic client programmers are the main audience of this information. This group includes Microsoft Visual Basic for Applications (VBA) programmers working in the standard scripting environments of Microsoft® Office applications, as well as other VBA-enabled applications (such as Microsoft Visio®, AutoCAD® and many others).
- *Other Windows programmers* who need to write client applications that automate Microsoft Dynamics SL and who know a programming tool that can create COM automation clients. Such tools might include Microsoft Visual C++, Borland C++, Borland Delphi, PowerBuilder, and many other Windows programming tools. Although all of the examples that are offered are Visual Basic-based, the discussions apply to any COM client.
- *Programming managers* who want to understand what their programmers need to do and know in order to manipulate Microsoft Dynamics SL applications through automation.
- *Evaluators* of Microsoft Dynamics SL's accessibility as a server to other applications.

What Programmers Should Know

If you plan to use this documentation, you should:

- Know your way around the Visual Basic 2008 or Visual Basic .NET 2010 integrated development environment (IDE).
- Know how to design a form and place controls on it.
- Know how to instantiate objects programmatically.
- Know how to program with the properties and methods of objects.
- Know how to program with basic constructs such as looping and branching.
- Know how to program with procedures, including sub and function procedures, how to pass parameters, and how to return values from functions.
- Know how to write object model automation clients in Visual Basic.

If you are a programmer working in a language other than Visual Basic, you should have knowledge and skills that are equivalent to those mentioned above for your chosen development environment. You should also be comfortable with translating Visual Basic source code into your own programming language.

For more information on the differences using Object Model with Visual Basic 6 versus Visual Basic 2005 or Visual Basic 2008, see *Appendix A* in the Visual Basic 2005 Conversion Toolkit for Microsoft Dynamics SL help or user guide.

Reference

Button Control (SIVControl Object)

Refers to the standard Visual Basic **Button** control.

Applies To

SIVControl object

Syntax

Object(“*Button Name*”)

The **Button** control syntax has these parts:

Part	Description
object	An instance of SIVApplication.
Button Name	Required string. Corresponds to the name of a **Button** control exposed by the **SIVApplication** object.

Remarks

If you set the **Value** property of an instance of a **SIVControl** object that represents a **Button** to True, then you cause its **Click** event to fire. In other words, setting the **Value** property to True is the same as the interactive user clicking on the **Button**. As with all **SIVControl** objects, you can perform this manipulation either by setting the **Value** property of the **SIVControl** instance that holds the **Button**, or you can implicitly see the **Value** by simply referring to the **SIVControl** instance itself (see examples below).

Note: in the case of the **Button**, the underlying Visual Basic control’s default property is also named **Value**, so it would also be possible to refer explicitly to the control’s **Value** property by name in the Properties collection (see examples). This would not work for all standard Visual Basic controls, many of whose default properties are not named “Value.”

You can also manipulate the **Button**’s other standard Visual Basic properties through the Microsoft Dynamics SL Object Model by using the **SIVControl** instance’s Properties collection of **SIVProperty** objects (see examples).

Possible Exceptions

7556 — Attempt to set **Value** property of disabled **Button**

See Also

SIVControl Object

Example (Visual Basic 6.0 client)

```
'Following code assumes that sivMyApp
'is an instance of SIVApplication

'Following are four different ways
'to manipulate a Button control on
'the application screen:

'this way --
    Dim sivctrlCmdOK As SIVControl
```

```
    Set sivctrlCmdOK = sivMyApp.Controls("cmdOK")
    sivctrlCmdOK = True
    sivctrlCmdOK.Properties.Item("Enabled") = False
'or this way --
    sivMyApp.Controls("cmdOK") = True
    sivMyApp.Controls("cmdOK").Properties.Item("Enabled") = False
'or this way --
    sivMyApp.Controls("cmdOK").Value = True
    sivMyApp.Controls("cmdOK").Properties.Item("Enabled") = False
'or this way --
    sivMyApp.Controls("cmdOK").Properties.Item("Value") = True
    sivMyApp.Controls("cmdOK").Properties.Item("Enabled") = False
```

Example (Visual Basic 2005 Client)

```
'Following code assumes that sivMyApp
'is an instance of SIVApplication

'Following are three different ways to manipulate a Button control on
'the application screen:

'this way --
Dim sivctrlCmdOK As SIVControl
sivctrlCmdOK = sivMyApp.Controls("cmdOK")
sivctrlCmdOK.Value = True
sivctrlCmdOK.Properties("Enabled").Value = False
'or this way --
sivMyApp.Controls("cmdOK").Value = True
sivMyApp.Controls("cmdOK").Properties("Enabled").Value = False
'or this way --
sivMyApp.Controls("cmdOK").Properties.Item("Value").Value = True
sivMyApp.Controls("cmdOK").Properties.Item("Enabled").Value = False
sivMyApp.Controls("cmdOK").Properties.Item("Enabled").Value = False
```

Cancel Method

Cancels pending changes made to all application entities.

Applies To

SIVApplication object

Syntax

object.**Cancel**

The **Cancel** method syntax has these parts:

Part	Description
Object	An instance of **SIVApplication**.

Remarks

This method abandons any changes to all data entities in the current application since the last save. If the **SIVApplication** object is visible to the user, the screen will be refreshed with the contents of the current data entity, or it will refresh to the New state if no entity was retrieved prior to any changes.

The **Cancel** method is equivalent to the user action of clicking `Cancel` on the toolbar, or pressing `ESC`.

Example

```
'Assumes sivMyApp is an already-instantiated SIVApplication object
sivMyApp.Cancel
```

Controls Collection (SIVApplication Object)

Returns a reference to a collection of **SIVControl** objects.

Applies To

SIVApplication object

Syntax

object.**Controls**

where *object* represents an instance of **SIVApplication**.

Methods

Count method, **Item** method

Remarks

You can manipulate **SIVControl** objects using the reference returned by the **Controls** property (see examples).

Note that controls originally programmed in a Microsoft Dynamics SL application and controls added via Customization Manager have no difference in the Microsoft Dynamics SL Object Model.

See Also

SIVControl Object, **SIVControls Collection**, **KeyControls Collection (SIVApplication Object)**

Example (Visual Basic 6.0 Client)

```
'Set a control object variable
'to point to a control in a screen's Controls collection

    Dim ctrlCurr As SIVControl
    Set ctrlCurr = sivMyApp.Controls("cTotalBalance")
    ctrlCurr = -300
'OR...Change a control directly
    sivMyApp.Controls("cTotalBalance") = -300
```

Example (Visual Basic 2005 Client)

```
'Set a control object variable
'to point to a control in a screen's Controls collection

Dim ctrlCurr As SIVControl
ctrlCurr = sivMyApp.Controls("cTotalBalance")
ctrlCurr.Value = -300

'OR...Change a control directly
sivMyApp.Controls("cTotalBalance").Value = -300
```

Constants

sivCurrencyView

Constant	Value	Description
sivCurrencyViewBase	1	Indicates screen is showing Base Currency values.
sivCurrencyViewTransaction	2	Indicates screen is showing Transaction Currency values.

sivCustomizationLevel

Constant	Value	Description
sivCstLvlStandard	1	Represents the Standard customization level.
sivCstLvlSupplementalProduct	2	Represents the Supplemental Product customization level.
sivCstLvlAllUsers	3	Represents the All Users customization level.
sivCstLvlOneUser	4	Represents the One User customization level.
sivCstLvlSelf	5	Represents the Self customization level.

sivEntityStatus

Constant	Value	Description
sivEntStatusInserted	1	Indicates the entity's status is Inserted.
sivEntStatusUpdated	2	Indicates the entity's status is Updated.
sivEntStatusNotChanged	3	Indicates the entity's status is NotChanged.

sivMessageResponse

Constant	Value	Description
sivMsgRspOk	1	Instructs the Object Model to invoke the **OK** button on the message.
sivMsgRspCancel	2	Instructs the Object Model to invoke the **Cancel** button on the message.
sivMsgRspAbort	3	Instructs the Object Model to invoke the **Abort** button on the message.
sivMsgRspRetry	4	Instructs the Object Model to invoke the **Retry** button on the message.
sivMsgRspIgnore	5	Instructs the Object Model to invoke the **Ignore** button on the message.
sivMsgRspYes	6	Instructs the Object Model to invoke the **Yes** button on the message.
sivMsgRspNo	7	Instructs the Object Model to invoke the **No** button on the message.
sivMsgRspClose	8	Instructs the Object Model to invoke the **Close** button on the message.

sivMessageType

Constant	Value	Description
sivMsgOk	1	Indicates the message box contains an **OK** button.
sivMsgOkCancel	2	Indicates the message box contains an **OK** button and a **Cancel** button.
sivMsgAbortRetryIgnore	3	Indicates the message box contains an **Abort** button, a **Retry** button, and an **Ignore** button.
sivMsgYesNo	4	Indicates the message box contains a **Yes** button and a **No** button.
sivMsgYesNoCancel	5	Indicates the message box contains a **Yes** button, a **No** button, and a **Cancel** button.
sivMsgRetryCancel	6	Indicates the message box contains a **Retry** button and a **Cancel** button.

sivRecordFound

Constant	Value	Description
sivRecFndNotFound	1	Indicates a record was not found for the specified data entity.
sivRecFndFound	2	Indicates a record was found for the specified data entity.

ControlType Property (SIVProperty Object)

A string representing the type (class name) of a control in an **SIVApplication** object.

Applies To

SIVProperty Object

Syntax

Properties(“**ControlType**”)

where *Properties* represents an instance of **SIVProperties**.

Remarks

The **ControlType** property is display only.

It represents the control type, or class name to which a control on an SIVApplication screen belongs.

It is useful to know the type of each control that you are dealing with when performing mass adjustments to the controls belonging to an **SIVApplication** object’s Controls collection (see examples below), or when a control whose type may be indeterminate is passed as a parameter to a procedure.

Example (Visual Basic 6.0 Client)

```
'Make sure all Label controls have Unhighlighted text
'Assume SIVMyApp is an instance of SIVApplication.

Dim ctrlCurr As SIVControl

For Each ctrlCurr In sivMyApp.Controls

        If ctrlCurr.Properties.Item("ControlType").Value = "Label" Then
                Dim fontProperty As IFontDisp
                Set fontProperty = ctrlCurr.Properties.Item("Font").Value
                fontProperty.Bold = False
        End If

Next ctrlCurr
```

Example (Visual Basic 2005 Client)

```
'Make sure all Label controls have Unhighlighted text
'Assume SIVMyApp is an instance of SIVApplication.

Dim ctrlCurr As SIVControl

For Each ctrlCurr In sivMyApp.Controls

        If ctrlCurr.Properties("ControlType").Value = "Label" Then
                Dim fontProperty As sivFont
                fontProperty = ctrlCurr.Properties("Font").Value
                fontProperty.Bold = False
        End If

Next ctrlCurr
```

Count Property

Returns the number of objects in a collection.

Applies To

Controls collection (**SIVApplication** object), Properties collection (**SIVControl** object)

Syntax

object.**Count**

The *object* placeholder is an object expression that evaluates to an object in the Applies To list.

Remarks

You can use this property to create a **For...Next** loop that loops through all the elements of a collection (see the example below). However, **For...Each** loops are usually more practical for this purpose (see documentation on the **SIVControls** and **SIVProperties** collections for examples).

Count is one-based, as is the index of the Controls and Properties collections. This means that the highest index number of the Controls collection would be Controls.Count, and the highest index number of the Properties collection would be Properties.Count (see example).

See Also

Controls Collection (SIVApplication Object), **Properties Collection (SIVControl Object)**

Example

```
'Assumes sivMyApp is an instance of
'SIVApplication

    Dim i As Integer, j As Integer
    For i = 1 To sivMyApp.Controls.Count
        For j = 1 To sivMyApp.Controls(i).Properties.Count
            '...do something with the current property of
            'the current control
        Next j
    Next i
```

Delete Method

Deletes data for a specified entity and any dependent entities.

Applies To

SIVApplication

Syntax

RetVal = *object*.**Delete** *Entity*

The **Delete** method syntax has these parts:

Part	Description
object	An instance of **SIVApplication**.
Entity	Required string. Corresponds to one of the SIVApplication object's data entities. (**Note:** This is the Microsoft Dynamics SL data level.) See remarks below.
RetVal	A value of type **sivRecordFound**. Corresponds to one of the values described below under the Returns section.

Returns

Normal (header level)

- **sivRecFndFound - 2** — Delete call at all non-Detail (that is, normal or header) levels always returns **sivRecFndFound**.

Detail (transaction or grid level)

One of the two following values, of type **sivRecordFound**:

- **sivRecFndNotFound – 1** — Means that no record following the deleted record was located upon execution of the **Delete** method.
- **sivRecFndFound - 2** — Means that a record following the deleted record was successfully located upon execution of the **Delete** method.

Note: The value that ends up in *RetVal* has nothing to do with whether or not a record was successfully deleted. It simply describes the state of the record pointer for *Entity* after the call to the **Delete** method. To check the success of the deletion attempt, set an error trap as shown in the example.

Remarks

Deletes the data for the specified entity from the database. You can obtain the *Entity* string for a particular set of controls by getting the **Level** property from the Properties collection of one of the controls.

The **Delete** method is equivalent to the user action of pressing the `DELETE` key on the toolbar or keying `CTRL+D`.

For Detail (grid or transaction) levels, if the **Delete** method is successful (no error was raised), then the current record changes to one of the following:

- If a record exists after the deleted record in logical order, then the current record becomes the next record after the deleted record and *Retval* is equal to **sivRecFndFound**.
- If no record existed after the deleted record (that is, the record just deleted was the last record in logical order in the table), then the current record becomes blank and *Retval* is equal to **sivRecFndNotFound**.

If the **SIVApplication** object is visible to the user, the new record's contents display in the application screen on the user's desktop.

For non-detail (normal or header) levels, if the **Delete** method is successful (no error was raised), then the current record at that level is reset to nothing (that is, as if the user hit the New key (CTRL+N) on that level).

The **Delete** method, as well as all other methods and property set operations throughout the Microsoft Dynamics SL Object Model, is synchronous with respect to the automation client. Control does not return to the automation client until the completion of all Microsoft Dynamics SL processing caused by the call to this method, whether the processing comes from the core Microsoft Dynamics SL kernel, the original application, or any customization of the application.

Possible Exceptions

- 7525 — Invalid entity string
- 7526 — Deletion is disabled for this entity

Example

```
'Assumes sivMyApp is an already-instantiated
'SIVApplication object
    On Error GoTo DELETE_ERROR
    Dim sivrfResult As sivRecordFound
    sivrfResult = sivMyApp.Delete("Batch")
    If sivrfResult = sivRecFndNotFound Then
        MsgBox "No record found after delete"
    End If
    Exit Sub
DELETE_ERROR:
    Select Case Err.Number - vbObjectError
        Case 7525
            MsgBox "Entity 'Batch' does not exist"
            Resume Next
        Case 7526
            MsgBox "DELETE not permitted here"
            Resume Next
        Case Else
            MsgBox Err.Number & ": " & Err.Description, _
            "UNEXPECTED ERROR"
    End Select
```

DisableObjectModel Statement (Microsoft SL SDK Applications)

Disables the Object Model for a Microsoft SL SDK application.

Syntax

DisableObjectModel

Remarks

If the Microsoft SL SDK programmer calls this statement in the Microsoft SL SDK application, then no Object Model automation client will be able to automate this application using the Object Model. If an automation client attempts to automate this application, then the **StartApplication** method of the **SIVApplication** object raises error 7563.

For this statement to function properly, the Microsoft SL SDK programmer must call this statement between the calls to *Applinit* and *ScreenInit* in the *Form_Load* event of the Microsoft SL SDK application's *Form1*.

Possible Exceptions

None

Example

```
'Code within Form1 of a Microsoft SL SDK application
Sub Form_Load ()

    'Standard Application Initialization Call
    Call ApplInit

    'Standard Setaddr Calls
    .....

    'Standard sqlcursor Call
    .....

    DisableObjectModel

    'Standard Screen Initialization Call
    Call ScreenInit

End Sub
```

Dispose Method

Destroys an instance of an object synchronously.

Applies To

SIVToolbar, SIVApplication

Syntax

object.**Dispose**

The **Delete** method syntax has these parts:

Part	Description
object	An instance of **SIVApplication** or **SIVToolbar**.

Remarks

In Visual Basic 2005, the rules for deterministically freeing .NET objects conform to the behavior of the CLR garbage collection process. Simply setting a variable to Nothing is not enough to cause an immediate cleanup of that object's resources. Instead, it simply marks the object for garbage collection. The garbage collection process runs periodically to free and reorganize workstation memory. Until that occurs, a toolbar or application could remain active on the workstation indefinitely.

To allow Object Model clients the ability to immediately free the Toolbar or application, the Dispose method has been implemented on both the SIVToolbar and SIVApplication objects. Calling this method will immediately destroy that object and free its resources, and permit the same behavior present in prior versions of Microsoft Dynamics SL.

See Also

Deterministic Freeing of SIVToolbar and SIVApplication Objects

Example

```
'Close an application
sivApp.Quit()
sivApp.Dispose()
sivApp = Nothing

'Close the Toolbar
sivToolbar.Logout()
sivToolbar.Quit()
sivToolbar.Dispose()
sivToolbar = Nothing
```

DSLDate Control (SIVControl Object)

Refers to the Microsoft Dynamics SL **DSLDate** control.

Applies To

Controls collection of **SIVApplication** object

Syntax

Object.Controls.("*DSLDate Name*")

The **DSLDate** control syntax has these parts:

Part	Description
object	An instance of **SIVApplication**.
DSLDate Name	Required string. Corresponds to the name of the **DSLDate** control exposed by the **SIVApplication** object.

Remarks

The instance of an **SIVControl** object that contains the **DSLDate** control has a **Value** property that corresponds to the current contents of that control as seen by the interactive user.

As with all **SIVControl** objects, you can manipulate this value either by setting the **Value** property of the **SIVControl** instance that holds the **DSLDate**, or you can implicitly refer to the **Value** by simply referring to the **SIVControl** instance itself (see examples below).

Note about date formats: The only format allowed for dates throughout the Microsoft Dynamics SL Object Model is a String or String Variant of the format "MM/DD/YYYY." All reads will return values in this format. If you attempt to write the **DSLDate Value** property with an invalid date format, you will generate an exception. One way to make sure that dates of unknown format will not cause a problem when you try to assign them to the **DSLDate**, would be to use the standard Visual Basic **Format** function, as illustrated in the example.

You can also manipulate the **DSLDate**'s Microsoft Dynamics SL and standard Visual Basic properties through the Microsoft Dynamics SL Object Model by using the **SIVControl** instance's Properties collection of **SIVProperty** objects (see examples).

For further particulars about the behavior of the **DSLDate** control, see the Microsoft SL SDK documentation.

Possible Exceptions

7542 — Incorrect date format (must be MM/DD/YYYY)

See Also

SIVControl Object, **Value Property (SIVControl Object)**

Example (Visual Basic 2005 Client)

```
'Following code assumes that sivMyApp
'is an instance of SIVApplication.

'Following are two different ways
'to manipulate a DSLDate control on
'the application screen:
'this way --
Dim sivctrlcSellDate As SIVControl
```

```
sivctrlcSellDate = sivMyApp.Controls("cSellDate")
sivctrlcSellDate.Value = Today.Date.ToString("MM/dd/yyyy")

'or this way --
sivMyApp.Controls("cSellDate").Value = Today.Date.ToString("MM/dd/yyyy")

'NOTE the use of the formatting behavior of Date.ToString to ensure that
'the format of the assigned date is MM/DD/YYYY. This
'is the only format that a DSLDate will accept
'(all other formats raise an exception).
```

EventLog Property

Contains the fully-qualified path to the event log created by the last process run during the current session of the application.

Applies To

SIVApplication

Syntax

RetVal = *Object.*EventLog

The EventLog property syntax has these parts:

Part	Description
object	A SIVApplication object.
RetVal	A string expression containing the fully-qualified path and file name of the event log generated by the last process run during the current session of the SIVApplication object.

Remarks

The EventLog property is display only.

The application creates an event log file by starting a process that creates an event log.

If the application has not run any processes during the current session, then the EventLog property contains a blank string.

If the last process that ran during the current application session did not generate an event log, then the EventLog property also contains a blank string.

Attempts to read the EventLog property, as well as all other property get/set operations and method calls throughout the Microsoft Dynamics SL Object Model, are synchronous with respect to the automation client. That is, control does not return to the automation client until the completion of all Microsoft Dynamics SL processing caused by the attempt to set this property, whether the processing comes from the core Microsoft Dynamics SL kernel, the original application, or any customization of the application.

Possible Exceptions

None

Example (Visual Basic 6 Client):

```
'Assumes sivApp is an instantiated copy of
'SIVApplication

'METHOD 1 FOR FIRING A PROCESS
'Fire off a process behind the command button
sivApp.Controls("cBeginProcessing") = True


'...and open its EventLog (maybe we will print it)
Dim iHandle As Integer
iHandle = FreeFile
Open sivApp.EventLog For Input As iHandle


'Put contents of the file into a string variable
Dim strLogContents As String
```

```
strLogContents = Input(LOF(iHandle, #iHandle)

'... then do something with the string variable
'(not shown here: perhaps print, search, or parse it)

'METHOD 2 FOR FIRING A PROCESS
'Set the screen's Action field to "Release Now"
sivApp.Controls("cAction") = "Release Now"
'Then Save the screen's contents
sivApp.Save

'...and do similar things with its event log
'to what's shown above (not shown here)
```

Example (Visual Basic 2005 Client)

```
'Assumes sivApp is an instantiated copy of
'SIVApplication

'METHOD 1 FOR FIRING A PROCESS
'Fire off a process behind the command button
sivApp.Controls("cBeginProcessing") = True


'Open its EventLog and put contents of the file into a
'string variable (maybe we will print it)
Dim strLogContents As String
strLogContents = My.Computer.FileSystem.ReadAllText(sivApp.EventLog)

'... then do something with the string variable
'(not shown here: perhaps print, search, or parse it)

'METHOD 2 FOR FIRING A PROCESS
'Set the screen's Action field to "Release Now"
sivApp.Controls("cAction") = "Release Now"
'Then Save the screen's contents
sivApp.Save

'...and do similar things with its event log
'to what's shown above (not shown here)
```

DisableObjectModel Statement (Microsoft SL SDK Applications)

Disables the Object Model for a Microsoft SL SDK application.

Syntax

DisableObjectModel

Remarks

If the Microsoft SL SDK programmer calls this statement in the Microsoft SL SDK application, then no Object Model automation client will be able to automate this application using the Object Model. If an automation client attempts to automate this application, then the **StartApplication** method of the **SIVApplication** object raises error 7563.

For this statement to function properly, the Microsoft SL SDK programmer must call this statement between the calls to *Applinit* and *ScreenInit* in the *Form_Load* event of the Microsoft SL SDK application's *Form1*.

Possible Exceptions

None

Example

```
'Code within Form1 of a Microsoft SL SDK application
Sub Form_Load ()

    'Standard Application Initialization Call
    Call ApplInit

    'Standard Setaddr Calls
    .....

    'Standard sqlcursor Call
    .....

    DisableObjectModel

    'Standard Screen Initialization Call
    Call ScreenInit

End Sub
```

ExposeCustomObject Statement (Microsoft SL SDK Applications)

Allows a Microsoft SL SDK application to expose a custom object to automation clients.

Syntax

ExposeCustomObject *CustomObject* **As Object**

The **ExposeCustomObject** statement syntax has these parts:

Part	Description
CustomObject	**Object** variable instantiated from a custom class of the Microsoft SL SDK application.

Remarks

When the Microsoft SL SDK application exposes the custom object pointed to by *Object*, then automation clients can use the **GetCustomObject** method of the **SIVApplication** object to get a pointer to the object. The automation clients can then use this object pointer to manipulate the custom object's properties and methods.

For this statement to function properly, the Microsoft SL SDK programmer should call this statement between the calls to *ApplInit* and *ScreenInit* in the *Form_Load* event of the Microsoft SL SDK application's *Form1*.

See "Custom Object Support" for comments on Visual Basic .NET changes necessary for this API.

Possible Exceptions

None

Example

```
'Code within a Microsoft SL SDK application:
'Assume that
'1) CustomObject is a public class containing the attribute ComVisible(True)
'[General Declarations section of a  BAS module]
Public oCustom As New CustomObject

'[in the project's Form1 ]
Sub Form_Load ()

    'Standard Application Initialization Call
    Call ApplInit

    'Standard Setaddr Calls
    .....

    'Standard sqlcursor Call
    .....

    ExposeCustomObject oCustom

    'Standard Screen Initialization Call
    Call ScreenInit

End Sub
```

First Method

Navigates to the first record of a specified entity for an **SIVApplication** object.

Applies To

SIVApplication

Syntax

object.**First** Entity

The **First** method syntax has these parts:

Part	Description
object	An instance of **SIVApplication**.
Entity	Required string. Corresponds to one of the **SIVApplication** object's data entities. (**Note:** This is the Microsoft Dynamics SL data level.) See remarks below.
RetVal	A value of type **sivRecordFound**. Corresponds to one of the values described below under the Returns section.

Returns

One of the two following values:

- **sivRecFndNotFound - 1** — Means that no record was located upon execution of the **First** method.
- **sivRecFndFound - 2** — Means that a record was successfully located upon execution of the **First** method.

Remarks

Navigates to the first record of the entity specified by *Entity* for the **SIVApplication** object. You can obtain the entity string for a particular set of controls by getting the **Level** property from the Properties collection of one of the controls.

If the current record was already the first record in Entity, the **First** method always returns **sivRecFndFound** even if it is already at the first record. The **First** method returns **sivRecFndNotFound** only if there are no entity records at all.

The **First** method is equivalent to the user action of clicking the **First** navigation button on the toolbar.

If the **First** method is successful (return value was **sivRecFndFound**) and the **SIVApplication** object is visible to the user, then the new record's contents will display in the application screen on the user's desktop.

The **First** method, as well as all other methods and property set operations throughout the Microsoft Dynamics SL Object Model, is synchronous with respect to the automation client. That is, control does not return to the automation client until the completion of all Microsoft Dynamics SL processing caused by the call to this method, whether the processing comes from the core Microsoft Dynamics SL kernel, the original application, or any customization of the application.

Possible Exceptions

- 7525 — Invalid entity string
- 7528 — **First** method is disabled for this entity

Example (Visual Basic 6 Client)

```
'Assumes sivMyApp is an already-instantiated
'SIVApplication object

    On Error GoTo NAVIGATE_ERROR
    Dim sivrfResult As sivRecordFound

    sivrfResult = sivMyApp.First("Batch")
    If sivrfResult = sivRecFndNotFound Then
        MsgBox "Navigate to a new Record"
    End If
    Exit Sub
NAVIGATE_ERROR:
  Select Case Err.Number - vbObjectError
      Case 7525
          MsgBox "Entity 'Batch' does not exist"
          Resume Next
      Case 7528
          MsgBox "FIRST not permitted here"
          Resume Next
      Case Else
          MsgBox Err.Number & ": " & Err.Description, _
          "UNEXPECTED ERROR"
  End Select
```

Example (Visual Basic 2005 Client)

```
'Assumes sivMyApp is an already-instantiated
'SIVApplication object

Dim sivrfResult As sivRecordFound


Try
    sivrfResult = sivMyApp.First("Batch")
    If sivrfResult = sivRecordFound.sivRecFndNotFound Then
        MsgBox("Navigate to a new Record")
    End If


Catch ex As System.Runtime.InteropServices.COMException


    Dim lErrNumber As Integer
    If ex.Source.Contains("Solomon") = True Or ex.ErrorCode - vbObjectError
> 0 Then
        lErrNumber = ex.ErrorCode - vbObjectError
    Else
        lErrNumber = ex.ErrorCode
    End If

    Select Case lErrNumber
        Case 7525
            MsgBox("Entity 'Batch' does not exist")
```

```
        Case 7528
            MsgBox("FIRST not permitted here")
        Case Else
            MsgBox(ex.Message, MsgBoxStyle.Exclamation, String.Format("{0}
Error {1}", ex.Source, lErrNumber.ToString()))

    End Select

Catch ex As Exception
    MsgBox(ex.Message, MsgBoxStyle.Exclamation, String.Format("{0}
Exception", ex.Source))
End Try
```

GetBusinessDate Method

Gives the current Microsoft Dynamics SL business date.

Applies To

SIVToolbar

Syntax

object.**GetBusinessDate** Month, Day, Year

The **GetBusinessDate** method syntax has these parts:

Part	Description
object	An instance of **SIVToolbar**.
Month	**ByRef Integer**. Represents the calendar month of the current business date. Values can be 1 - 12.
Day	**ByRef Integer**. Represents the day of the month of the current business date. Values can be 1 - 31.
Year	**ByRef Integer**. Represents the four-digit year of the current business date. Values can be any four-digit number.

Remarks

Returns the current Microsoft Dynamics SL business date in its three parameters. Before calling this method in your code, you should prepare three integer variables to hold the month, day, and year and pass these variables as the respective arguments to the method (see examples).

The **GetBusinessDate** method, as well as all other methods and property set operations throughout the Microsoft Dynamics SL Object Model, is synchronous with respect to the automation client. That is, control does not return to the automation client until the completion of all Microsoft Dynamics SL processing caused by the call to this method, whether the processing comes from the core Microsoft Dynamics SL kernel, the original application, or any customization of the application.

Possible Exceptions

7506 — Not Logged In.

See Also

SetBusinessDate Method

Example (Visual Basic 6 Client)

```
'Assumes sivTB is an already-instantiated
'SIVToolbar object
    Dim iMonth As Integer
    Dim iDay As Integer, iYear As Integer

    On Error GoTo GET_DATE_ERROR

    sivTB.GetBusinessDate iMonth, iDay, iYear
    '...do something with the date information here

    Exit Sub

GET_DATE_ERROR:
    Select Case Err.Number - vbObjectError
        Case 7506
            MsgBox "Not logged in"
            Resume Next
        Case Else
```

```
            MsgBox Err.Number & ": " _
                &  Err.Description, _
                  "UNEXPECTED ERROR"
    End Select
```

Example (Visual Basic 2005 Client)

```
'Assumes sivTB is an already-instantiated
'SIVToolbar object
Dim iMonth As Integer
Dim iDay As Integer, iYear As Integer

Try
    sivTB.GetBusinessDate(iMonth, iDay, iYear)
    '...do something with the date information here
Catch ex As System.Runtime.InteropServices.COMException

    Dim lErrNumber As Integer
    If ex.Source.Contains("Solomon") = True Or ex.ErrorCode - vbObjectError > 0
Then
        lErrNumber = ex.ErrorCode - vbObjectError
    Else
        lErrNumber = ex.ErrorCode
    End If

    Select Case lErrNumber
        Case 7506
            MsgBox("Not logged in")
        Case Else
            MsgBox(ex.Message, MsgBoxStyle.Exclamation, String.Format("{0}
Error {1}", ex.Source, lErrNumber.ToString()))
        End Select

Catch ex As Exception
    MsgBox(ex.Message, MsgBoxStyle.Exclamation, String.Format("{0} Exception",
ex.Source))
End Try
```

GetCurrencyIDs Method

Gets the transaction currency ID for the current screen.

Applies To

SIVApplication

Syntax

object.**GetCurrencyIDs** *ByRef TransactionCurrencyID As String, ByRef IntermediateCurrencyID As String*

The **GetCurrencyIDs** method syntax has these parts:

Part	Description
object	An instance of **SIVApplication**.
TransactionCurrencyID	**ByRef** string representing a valid **CuryID** code for the logged-in company and database.
IntermediateCurrencyID	**ByRef** string. Reserved for future use. Has no effect at this time.

Remarks

Allows you to get the current screen's transaction currency ID.

Before calling this method, you should prepare two String variables and pass them as arguments to the method. After the call to the method, the variable that you passed for the *TransactionCurrencyID* will contain the currency ID for the transaction of the current screen.

At this point, *IntermediateCurrencyID* should always return as a blank string. It is reserved for future use and has no significance in the current version of the Microsoft Dynamics SL Object Model.

The **GetCurrencyIDs** method, as well as all other methods and property set operations throughout the Microsoft Dynamics SL Object Model, is synchronous with respect to the automation client. That is, control does not return to the automation client until the completion of all Microsoft Dynamics SL processing caused by the call to this method, whether the processing comes from the core Microsoft Dynamics SL kernel, the original application, or any customization of the application.

Possible Exceptions

None

See Also

SetCurrencyIDs Method

Example

```
'Assumes sivApp is an already-instantiated
'SIVApp object
    Dim strTransCuryID As String
    Dim strIntCuryID As String
    sivApp.GetCurrencyIDs _
           strTransCuryID, strIntCuryID

    MsgBox strTransCuryID,,"Transaction Currency"
    'strIntCuryID will be blank
```

GetCurrencyView Method

Returns the display currency for currency amounts on the current screen.

Applies To

SIVApplication

Syntax

Retval = *object*.GetCurrencyView

The GetCurrencyView method syntax has these parts:

Part	Description
object	An instance of SIVApplication.
Retval	Integer of enumerated type sivCurrencyView representing the type of currency that amounts are currently displayed in (either system Base Currency, or screen Transaction Currency).

Remarks

Allows you to find out whether current screen's is displaying currency in the system Base Currency or in the screen Transaction Currency.

The return value from this method will be one of the enumerated constants in the sivCurrencyView type indicating either the system Base Currency or the screen's Transaction Currency.

The GetCurrencyView method, as well as all other methods and property set operations throughout the Microsoft Dynamics SL Object Model, is synchronous with respect to the automation client. That is, control does not return to the automation client until the completion of all Microsoft Dynamics SL processing caused by the call to this method, whether the processing comes from the core Microsoft Dynamics SL kernel, the original application, or any customization of the application.

Possible Exceptions

None

See Also

SetCurrencyView Method

Example

```
'Assumes sivApp is an already-instantiated
'SIVApp object
    Dim iCuryView As sivCurrencyView
    iCuryView = sivApp.GetCurrencyView
    If iCuryView = sivCurrencyViewBase Then
        MsgBox "Base Currency", , "Current Screen display"
    Else
        MsgBox "Tran Currency", , "Current Screen display"
    End If
```

GetCustomizationLevel Method

Gets the current customization level of the session.

Applies To

SIVApplication

Syntax

object.**GetCustomizationLevel** *CustomizationLevel, UserID, ExcludeMacroCode*

The **GetCustomizationLevel** method syntax has these parts:

Part	Description
object	An instance of **SIVToolbar**.
CustomizationLevel	**ByRef Integer** of type **sivCustomizationLevel**. Represents the current customization level of the toolbar session.
UserID	**String**. The user ID to apply when setting the customization level to **sivCstLvlOneUser**.
ExcludeMacroCode	**Boolean**. Setting to True excludes macro code, while setting to False includes macro code.

Remarks

Returns the current Microsoft Dynamics SL customization level in the ByRef argument customization level, as well as information about the user ID (only meaningful when customization level is SingleUser) and whether or not macro code is excluded from customizations for this session. Before calling this method in your code, you should prepare integer, string, and Boolean variables to hold the customization level, user ID, and ExcludeMacroCode values, respectively, and pass these variables as the arguments to the method (see example).

The **SIVToolbar** object must already be logged on to a database for this method call to work.

The **GetCustomizationLevel** method, as well as all other methods and property set operations throughout the Microsoft Dynamics SL Object Model, is synchronous with respect to the automation client. That is, control does not return to the automation client until the completion of all Microsoft Dynamics SL processing caused by the call to this method, whether the processing comes from the core Microsoft Dynamics SL kernel, the original application, or any customization of the application.

Possible Exceptions

7506 — Not logged in

See Also

SetCustomizationLevel Method

Example

```
'Assumes sivTB is an already-instantiated
'SIVToolbar object
  Dim iCustLevel As sivCustomizationLevel
  Dim strUserID As String
  Dim blnExcludeMacroCode As Boolean
  On Error GoTo GET_CUSTLEVEL_ERROR
  sivTB.GetCustomizationLevel iCustLevel, _
  strUserID, _
  blnExcludeMacroCode
```

```
  '...do something with the customization information here
  Exit Sub
GET_CUSTLEVEL_ERROR:
  Select Case Err.Number - vbObjectError
      Case 7506
          MsgBox "Not logged in"
          Resume Next
      Case Else
          MsgBox Err.Number & ": " & Err.Description, _
          "UNEXPECTED ERROR"
  End Select
```

GetCustomObject Method

Returns a handle to the custom object defined by the underlying Microsoft SL SDK application.

Applies To

SIVApplication

Syntax

Set *Retval* = *object*.GetCustomObject()

The **GetCustomObject** method syntax has these parts:

Part	Description
RetVal	A variable of type **Object**.
object	An instance of **SIVApplication**.

The **GetCustomObject** method returns a handle to an object that may be defined in the underlying Microsoft Dynamics SL application and that the underlying application has exposed with a call to **ExposeCustomObject**.

If the underlying application does not expose a custom object, then the value returned by **GetCustomObject** is equal to Nothing (see example below).

Calls to the **GetCustomObject** method, as well as all property set operations and other method calls throughout the Microsoft Dynamics SL Object Model, are synchronous with respect to the automation client. That is, control does not return to the automation client until the completion of all Microsoft Dynamics SL processing caused by the call to this method, whether the processing comes from the core Microsoft Dynamics SL kernel, the original application, or any customization of the application.

Possible Exceptions

None

Example

```
'Assumes that sivApp is an already-instantiated
'SIVApplication object
    Dim oCustom As Object
    Set oCustom = sivApp.GetCustomObject()
    If oCustom Is Nothing Then
        MsgBox "No Custom Object Available"
    Else
        '...do something with custom object
    End If
```

GetEntityStatus Method

Returns the state of the specified data entity.

Applies To

SIVApplication

Syntax

*RetVal = object.**GetEntityStatus(**Entity **As String)***

The **GetEntityStatus** method syntax has these parts:

Part	Description
RetVal	An integer of enumerated type **sivEntityStatus**.
object	An instance of **SIVApplication**.
Entity	A string representing the data entity whose status you are checking.

You can use the **GetEntityStatus** method to detect whether or not a specific data entity belonging to an **SIVApplication** object has changed since the beginning of the session with the current recordset. The exact value returned by **GetEntityStatus** indicates the type of change that has happened (update, insert, or no change).

The value of *Entity* is a string that names the particular data entity in the application that you are checking. If you are not certain of the exact name of the data entity, then you can get the name by checking the **Level** property of any control whose data belongs to that entity (see examples below).

Calls to the **GetEntityStatus** method, as well as all property set operations and other method calls throughout the Microsoft Dynamics SL Object Model, are synchronous with respect to the automation client. That is, control does not return to the automation client until the completion of all Microsoft Dynamics SL processing caused by the call to this method, whether the processing comes from the core Microsoft Dynamics SL kernel, the original application, or any customization of the application.

Possible Exceptions

None

Example (Visual Basic 6 Client)

```
'Assumes that sivApp is an instantiated copy of
'SIVApplication

'Set a control object variable to point
'to a control in the app's Controls collection
Dim ctrlCurr As SIVControl
Set ctrlCurr = sivApp.Controls("cTotalBalance")

'Get the name of that control's DataEntity
Dim strLevel as String
strLevel = ctrlCurr.Properties.Item("Level")

'Get the entity's status
Dim iStatus As sivEntityStatus
iStatus = sivApp.GetEntityStatus(strLevel)

'React to the status
Select Case iStatus
    Case sivEntStatusUpdated
        MsgBox "Updated"
    Case sivEntStatusInserted
        MsgBox "Inserted"
    Case sivEntStatusNotChanged
```

```
        MsgBox "No Change"
End Select
```

Example (Visual Basic 2005 Client)

```
'Assumes that sivApp is an instantiated copy of
'SIVApplication

'Set a control object variable to point
'to a control in the app's Controls collection
Dim ctrlCurr As SIVControl
ctrlCurr = sivApp.Controls("cctrltot")
'Get the name of that control's DataEntity
Dim strLevel As String
strLevel = ctrlCurr.Properties("Level").Value
'Get the entity's status
Dim iStatus As sivEntityStatus
iStatus = sivApp.GetEntityStatus(strLevel)

'React to the status
Select Case iStatus
    Case sivEntityStatus.sivEntStatusUpdated
        MsgBox("Updated")
    Case sivEntityStatus.sivEntStatusInserted
        MsgBox("Inserted")
    Case sivEntityStatus.sivEntStatusNotChanged
        MsgBox("No Change")
End Select
```

GetStatusBarText Method

Returns the text and the tooltip text for the text pane of the status bar.

Applies To

SIVApplication

Syntax

object.GetStatusBarText *Text, ToolTipText*

The SetStatusBarText method syntax has these parts:

Part	Description
object	An instance of SIVToolbar.
Text	ByRef string variable representing text currently displayed in the text pane of the application screen's status bar.
ToolTipText	ByRef string representing the current ToolTip text that will pop up when the user pauses the mouse over the text pane of the application screen's status bar.

Remarks

You can detect the text currently displayed on the application screen's status bar text panel as well as the ToolTip text for that panel with the GetStatusBarText method.

The status bar is the area at the bottom of a Microsoft Dynamics SL application screen that contains information in various panes. Panes include information about date and time as well as a text pane with variable information.

The GetStatusBarText method retrieves information only about the text pane in its *Text* argument and the ToolTip text for the same pane in its *ToolTipText* argument. *ToolTipText* represents the contents of the popup that appears when the user pauses the mouse over the text pane.

Calls to the GetStatusBarText method, as well as all property set operations and other method calls throughout the Microsoft Dynamics SL Object Model, are synchronous with respect to the automation client. That is, control does not return to the automation client until the completion of all Microsoft Dynamics SL processing caused by this method call, whether the processing comes from the core Microsoft Dynamics SL kernel, the original application, or any customization of the application.

Possible Exceptions

None

Example

```
'Assumes sivApp is an already-instantiated
'SIVApplication object

Dim strSBText As String
Dim strSBToolTip As String
    sivApp.GetStatusBarText strSBText, _
                            strSBToolTip

MsgBox "Status Bar Text = " & strSBText
MsgBox "ToolTip = " & strSBToolTip
```

InitializeMode Property

Controls whether or not the current toolbar session is in Initialize mode.

Applies To

SIVToolbar

Syntax

Object.**InitializeMode** = *Boolean*
Boolean = Object.**InitializeMode**

The **InitializeMode** property syntax has these parts:

Part	Description
object	A **SIVToolbar** object.
Boolean	A Boolean expression specifying whether Initialize mode is on or off.

Settings

The settings for *Boolean* are:

Setting	Description
True	Initialize mode is on.
False	(Default) Initialize mode is off.

Remarks

For the SIVToolbar, an **InitializeMode** property setting of True would toggle on the system's Initialize mode, allowing the currently logged-on user to set all fields on application screens.

An **InitializeMode** property setting of False (the default) would toggle off the system's Initialize mode, signifying that user could only set fields as specified by the underlying application and customizations.

Attempts to set the **InitializeMode** property, as well as all other property set operations and method calls throughout the Microsoft Dynamics SL Object Model, are synchronous with respect to the automation client. That is, control does not return to the automation client until the completion of all Microsoft Dynamics SL processing caused by the attempt to set this property, whether the processing comes from the core Microsoft Dynamics SL kernel, the original application, or any customization of the application.

Possible Exceptions

7506 — Not logged in

See Also

SIVToolbar Object

Example

```
Set sivtbCurrent = New SIVToolbar

sivtbCurrent.Login "CHOMPER", "MSINTL", _
"NWIND", "NDavolio", "yowsa"
dim sivMyApp As SIVApplication
Set sivMyApp = _
sivtbCurrent.StartApplication("tx\MyApp.exe")
sivtbCurrent.InitializeMode = True
```

Item Property

Returns a specific member of a collection, either by position or by key.

Applies To

Controls collection (**SIVApplication** object), **Properties** collection (**SIVControl** object)

Syntax

object.**Item**(*index*)
object(*index*)

The **Item** property syntax has the following parts:

Part	Description
object	Required. An object from the Applies To list.
index	Required string or integer expression. If an integer expression, index must be a number between 1 and the value of the collection's **Count** property. If a string expression, index must correspond to the item's name.

Remarks

If the value provided by *index* does not exist in the collection, then a runtime error occurs.

Item is the default property of a collection, so it is never necessary to specify **Item** in .NET code. All you really ever need is the collection name and the *index* value (see the second syntax example above and the code example below). The exception is that you need to specify **Item** when you code in Visual Basic 6.0.

Possible Exceptions

None

See Also

Controls Collection (SIVApplication Object), **Properties Collection (SIVControl Object)**

Example (Visual Basic 2005 Client)

```
'Assumes sivMyApp is an instance of
'SIVApplication

Dim currCtrl as SIVControl
Dim currProp as SIVProperty

'the longer, explicit syntactic form :
Set currCtrl = SIVApplication.Controls.Item("cTotal")
Set currProp = currCtrl.Properties.Item("BackColor")

'the shorter, implicit syntactic form:
Set currCtrl = SIVApplication.Controls("cTotal")
Set currProp = currCtrl.Properties("BackColor")
```

KeyControls Collection (SIVApplication Object)

Returns a reference to a collection of **SIVControl** objects.

Applies To

SIVApplication object

Syntax

object.KeyControls

where *object* represents an instance of **SIVApplication**.

Methods

Count method, **Item** method

Remarks

You can manipulate **SIVControl** objects using the reference returned by the **KeyControls** property (see examples). This collection contains only the key controls for the application. These controls are also contained in the Controls collection.

Note: Controls originally programmed in a Microsoft Dynamics SL application and controls added via Customization Manager have no difference in the Microsoft Dynamics SL Object Model.

See Also

Controls Collection (SIVApplication Object), **SIVControl Object**, **SIVControls Collection**

Example (Visual Basic 6 Client)

```
'Set a control object variable
'to point to a control in a screen's Controls collection

    Dim ctrlCurr As SIVControl
    Set ctrlCurr = sivMyApp.KeyControls("cTotalBalance")
    ctrlCurr = -300

'OR...Change a control directly

    sivMyApp.KeyControls("cTotalBalance") = -300
```

Example (Visual Basic 2005 Client)

```
'Set a control object variable
'to point to a control in a screen's Controls collection

    Dim ctrlCurr As SIVControl
    ctrlCurr = sivMyApp.KeyControls("cTotalBalance")
    ctrlCurr.Value = -300

'OR...Change a control directly

    sivMyApp.KeyControls("cTotalBalance").Value = -300
```

Label Control (SIVControl Object)

Refers to the standard Visual Basic **Label** control.

Applies To

SIVControl object

Syntax

Object.(“*Label Name*”)

The **Label** control syntax has these parts:

Part	Description
object	An instance of **SIVApplication**.
Label Name	Required string. Corresponds to the name of a **Label** control exposed by the **SIVApplication** object.

Remarks

If you set the **Value** property of an instance of an **SIVControl** object that represents a label to some string value, then you set the l**Label's Caption** (since **Caption** is the default property of the standard Visual Basic **Label** control). As with all **SIVControl** objects, you can perform this manipulation either by setting the **Value** property of the **SIVControl** instance that holds the **Label**, or you can implicitly refer to the **Value** by simply referring to the **SIVControl** instance itself (see examples below).

You can also manipulate the **Label**'s other standard Visual Basic properties through the Microsoft Dynamics SL Object Model by using the **SIVControl** instance's Properties collection of **SIVProperty** objects (see examples).

Possible Exceptions

None

See Also

SIVControl Object

Example (Visual Basic 6.0 Client)

```
'Following code assumes that sivMyApp
'is an instance of SIVApplication
'Following are three different ways
'to manipulate a Label control on
'the application screen:

'this way --
Dim sivctrlLblBalance As SIVControl
Set sivctrlLblBalance = sivMyApp.Controls("lblBalance")
sivctrlLblBalance = "Balance"
sivctrlLblBalance.Properties.Item("BackColor") = vbRed

'or this way --
sivMyApp.Controls("lblBalance") = "Balance"
sivMyApp.Controls("lblBalance").Properties.Item("BackColor") = vbRed

'or this way --
sivMyApp.Controls("lblBalance").Value = "Balance"
sivMyApp.Controls("lblBalance").Properties.Item("BackColor") = vbRed
```

Example (Visual Basic 2005 Client)

```
'Following code assumes that sivMyApp
'is an instance of SIVApplication
       'Following are two different ways
       'to manipulate a Label control on
       'the application screen:
       'this way --
       Dim sivctrlLblBalance As SIVControl
       sivctrlLblBalance = sivMyApp.Controls("lblBalance")
       sivctrlLblBalance.Value = "Balance"
       sivctrlLblBalance.Properties("BackColor").Value =
VBRUN.ColorConstants.vbRed

       'or this way --
       sivMyApp.Controls("lblBalance").Value = "Balance"
    sivMyApp.Controls("lblBalance").Properties("BackColor").Value =
VBRUN.ColorConstants.vbRed
```

Last Method

Navigates to the last record for a specified entity.

Applies To

SIVApplication

Syntax

object.**Last** Entity

The **Last** method syntax has these parts:

Part	Description
object	An instance of **SIVApplication**.
Entity	Required string. Corresponds to one of the **SIVApplication** object's data entities. (**Note:** This is the Microsoft Dynamics SL data level.) See remarks below.
RetVal	A value of type **sivRecordFound**. Corresponds to one of the values described below under the Returns section.

Returns

One of the two following values:

- **sivRecFndNotFound – 1** — Means that no record was located upon execution of the **Last** method.
- **sivRecFndFound - 2** — Means that a record was successfully located upon execution of the **Last** method.

Remarks

Navigates to the last record of the entity specified by *Entity* for the **SIVApplication** object. You can obtain the *Entity* string for a particular set of controls by getting the **Level** property from the Properties collection of one of the controls.

If the current record was already the last record in *Entity*, the **Last** method always returns **sivRecFndFound** even if it is already at the last record. The **Last** method returns **sivRecFndNotFound** only if there are no entity records at all.

The **Last** method is equivalent to the user action of clicking the **Last** navigation button on the toolbar.

If the **Last** method is successful (return value was **sivRecFndFound**) and the **SIVApplication** object is visible to the user, then the new record's contents display in the application screen on the user's desktop.

The **Last** method, as well as all other methods and property set operations throughout the Microsoft Dynamics SL Object Model, is synchronous with respect to the automation client. That is, control does not return to the automation client until the completion of all processing caused by the call to this method, whether the processing comes from the core Microsoft Dynamics SL kernel, the original application, or any customization of the application.

Possible Exceptions

- 7525 — Invalid entity string
- 7530 — **Last** method is disabled for this entity

Example (Visual Basic 6.0 Client)

```
'Assumes sivMyApp is an already-instantiated
'SIVApplication object

On Error GoTo NAVIGATE_ERROR

Dim sivrfResult As sivRecordFound
sivrfResult = sivMyApp.Last("Batch")

If sivrfResult = sivRecFndNotFound Then
  MsgBox "Navigate to a new Record"
End If

Exit Sub

NAVIGATE_ERROR:

  Select Case Err.Number - vbObjectError
      Case 7525
          MsgBox "Entity 'Batch' does not exist"
          Resume Next
      Case 7530
          MsgBox "LAST not permitted here"
          Resume Next
      Case Else
          MsgBox Err.Number & ": " & Err.Description, _
          "UNEXPECTED ERROR"
  End Select
```

Example (Visual Basic 2005 Client)

```
'Assumes sivMyApp is an already-instantiated
'SIVApplication object

Dim sivrfResult As sivRecordFound

Try
    sivrfResult = sivMyApp.Last("Batch")
    If sivrfResult = sivRecordFound.sivRecFndNotFound Then
        MsgBox("Navigate to a new Record")
    End If

Catch ex As System.Runtime.InteropServices.COMException

    Dim lErrNumber As Integer
    If ex.Source.Contains("Solomon") = True Or ex.ErrorCode - vbObjectError > 0
Then
        lErrNumber = ex.ErrorCode - vbObjectError
    Else
        lErrNumber = ex.ErrorCode
    End If

    Select Case lErrNumber
        Case 7525
            MsgBox("Entity ’Batch’ does not exist")
        Case 7530
            MsgBox("LAST not permitted here")
        Case Else
            MsgBox(ex.Message, MsgBoxStyle.Exclamation, String.Format("{0}
Error {1}", ex.Source, lErrNumber.ToString()))
    End Select

Catch ex As Exception
    MsgBox(ex.Message, MsgBoxStyle.Exclamation, String.Format("{0} Exception",
ex.Source))
End Try
```

Level Property (SIVProperty Object)

A String representing the entity name (Microsoft Dynamics SL data level) of a control in an **SIVApplication** object.

Applies To

SIVProperty object

Syntax

Properties(“**Level**”)

where *Properties* represents an instance of **SIVProperties**.

Remarks

The **Level** property is display only.

It represents the Microsoft Dynamics SL data entity (or Microsoft Dynamics SL data level) to which a control on an SIVApplication screen belongs.

You need to know the **Level** names of controls on a Microsoft Dynamics SL application screen in order to navigate and manipulate the various data entities with such **SIVApplication** methods as **Delete**, **First**, **Last**, **New**, **Next**, and **Previous,** because the *Entity* string is a required argument to these methods. You can use an **SIVControl**’s **Level** property to determine the *Entity* name of a particular control’s data level (see the examples below).

Note: Though Microsoft Dynamics SL data entity levels have traditionally been represented by integers, starting with Level 0 and proceeding upward, the Object Model uses the actual level name from the update control instead of the level number. For methods in the object model that deal with levels (for instance, new, delete, first, last, next, and previous) you must use the string value returned by the **Level** property of a control. The Object Model provides the **LevelNumber** property for reference, but you cannot use it as the level parameter in one of those methods. Using the string value will make it easier to debug your code because you will not have to keep track of which number corresponds to the batch, document or detail level on the screen.

The word *Level* is strictly reserved to the context of this property. It is not used anywhere else in the Microsoft Dynamics SL Object Model.

Possible Exceptions

None

See Also

Delete Method, **GetEntityStatus Method**, **New Method**, **Next Method**, **SIVControl Object**, **SIVControls Collection**, **SIVProperty Object**, **LevelNumber Property (SIVProperty Object)**

Example (Visual Basic 6.0 Client)

```
'Set a control object variable
'to point to a control in a screen's Controls
'collection

Dim ctrlCurr As SIVControl
Set ctrlCurr = sivMyApp.Controls("cTotalBalance")

'Get the Level of that control's Entity
Dim strLevel as String
strLevel = ctrlCurr.Properties.Item("Level")
```

Example (Visual Basic 2005 Client)

```
'Set a control object variable
'to point to a control in a screen's Controls
'collection

Dim ctrlCurr As SIVControl
ctrlCurr = sivMyApp.Controls("cTotalBalance")

'Get the Level of that control's Entity
Dim strLevel as String
strLevel = ctrlCurr.Properties("Level").Value
```

LevelNumber Property (SIVProperty Object)

An integer representing the entity level (Microsoft Dynamics SL data level) of a control in an **SIVApplication** object.

Applies To

SIVProperty object

Syntax

Properties("**LevelNumber**")

where *Properties* represents an instance of **SIVProperties**.

Remarks

The **LevelNumber** property is display only.

It represents the Microsoft Dynamics SL data entity (or data level) to which a control on an SIVApplication screen belongs.

This integer cannot be used when specifying an entity as a parameter for the First, Next, Previous, Last, New and Delete methods of the SIVApplication object. The level property that returns a string needs to be used as a parameter value for those functions.

See Also

Delete Method, **GetEntityStatus Method**, **New Method**, **Next Method**, **SIVControl Object**, **SIVControls Collection**, **SIVProperty Object**, **LevelNumber Property (SIVProperty Object)**

Example (Visual Basic 6.0 Client)

```
'Set a control object variable
'to point to a control in a screen's Controls
'collection

Dim ctrlCurr As SIVControl
Set ctrlCurr = sivMyApp.Controls("cTotalBalance")

'Get the Level of that control's Entity
Dim LevelNumber as Integer
LevelNumber = ctrlCurr.Properties.Item("LevelNumber")
```

Example (Visual Basic 2005 client)

```
'Set a control object variable
'to point to a control in a screen's Controls
'collection

Dim ctrlCurr As SIVControl
ctrlCurr = sivMyApp.Controls("cTotalBalance")

'Get the Level of that control's Entity
Dim LevelNumber as Integer
LevelNumber = ctrlCurr.Properties("LevelNumber").Value
```

Login Method

Sets a database context for the automation server. When this function returns without exceptions, the automation client is logged on to a company and ready to start an application or call other methods on the toolbar.

Applies To

SIVToolbar

Syntax

object.**Login** *SystemDatabaseServerName*, *SystemDatabaseName, CompanyID, UserID, Password*

The **Login** method syntax has these parts:

Part	Description
object	An instance of **SIVToolbar**.
SystemDatabase ServerName	Required. String value indicating name of database server where the target Microsoft Dynamics SL database resides.
SystemDatabase Name	Required. String value giving name of target system Microsoft Dynamics SL database.
CompanyID	Required. String parameter giving the company ID that the session will use.
UserID	Required for SQL Authentication. Microsoft Dynamics SL user ID to use during this session with the Microsoft Dynamics SL database. This parameter is ignored if Windows Authentication is being utilized.
Password	Required for SQL Authentication. Microsoft Dynamics SL password for the user ID. This parameter is ignored if Windows Authentication is being utilized.

Remarks

This method cannot be called if the automation client is already logged on to a company.

This method must always be called at the beginning of an automation session, even if the interactive user is already logged on. If this logon differs from the one the interactive user is using, an exception is raised, because Microsoft Dynamics SL only supports one login context (that is, user ID and company ID) per desktop. In general, the first user to log on (whether the interactive user or the automation client) sets the context. Subsequent concurrent logins must use the same context, or they will receive an error.

To maintain system security, the interactive user must always log on manually, regardless of whether the automation client is already logged on.

This method does not enable any menus on the toolbar application as the interactive login would. This ensures that the interactive user has to log on even if the automation client is already logged on.

The **Login** method, as well as all other methods and property set operations throughout the Microsoft Dynamics SL Object Model, is synchronous with respect to the automation client. That is, control does not return to the automation client until the completion of all Microsoft Dynamics SL processing caused by the call to this method, whether the processing comes from the core Microsoft Dynamics SL kernel, the original application, or any customization of the application.

You might wish to create a unique user ID that only an automation client on a workstation uses and then give that user ID access rights that are sufficient for carrying out the tasks that the automation client needs to perform.

Use of Windows Authentication vs. SQL Authentication

If the Microsoft Dynamics SL installation has been configured to use Windows Authentication for user logons, the user ID and password are ignored by the **Login** method. For SQL Server Authentication, the user ID and password are required and should be passed to this method.

Possible Exceptions

- 7500 — Login Error: Already logged in
- 7501 — Login Error: System Database Name is Greater than 20 Characters
- 7502 — Login Error: Incorrect System Database Server Name or System Database Name
- 7503 — Login Error: Incorrect Company ID, User ID, or Password
- 7504 — Login Error: System Database Does Not Exist
- 7505 — Login Error: Interactive User id Different from Client Login

See Also

Logout Method

Example (Visual Basic 6.0 Client)

```
Dim sivtbCurrent As New SIVToolbar
On Error GoTo LOGIN_ERR
sivtbCurrent.Login "CHOMPER", "MSINTL", _
  "NWIND", "NDavolio", "yowsa"

Exit Sub

LOGIN_ERR:

Select Case Err.Number - vbobjecterror
  Case 7500 'Already logged in
  Case 7501 'System Database Name > 20 Characters
  Case 7502 'Wrong System Database Server Database Name
  Case 7503 'Incorrect Company ID, User ID, or Password
  Case 7504 'System Database Does Not Exist
  Case 7505 'Interactive User ID <> Client Login
  Case Else
End Select
```

Example (Visual Basic 2005 Client)

```
Dim sivtbCurrent As New SIVToolbar
Try
    sivtbCurrent.Login("CHOMPER", "MSINTL", "NWIND", "NDavolio", "yowsa")
Catch ex As System.Runtime.InteropServices.COMException

    Dim lErrNumber As Integer
    If ex.Source.Contains("Solomon") = True Or ex.ErrorCode - vbObjectError > 0
Then
        lErrNumber = ex.ErrorCode - vbObjectError
    Else
        lErrNumber = ex.ErrorCode
    End If

    Select Case lErrNumber
                Case 7500 'Already logged in
                Case 7501 'System Database Name > 20 Characters
                Case 7502 'Wrong System Database Server Database Name
                Case 7503 'Incorrect Company ID, User ID, or Password
                Case 7504 'System Database Does Not Exist
                Case 7505 'Interactive User ID <> Client Login
                Case Else
```

```
                    MsgBox(ex.Message, MsgBoxStyle.Exclamation,
String.Format("{0} Error {1}", ex.Source, lErrNumber.ToString()))
            End Select

Catch ex As Exception
    MsgBox(ex.Message, MsgBoxStyle.Exclamation, String.Format("{0} Exception",
ex.Source))
End Try
```

Logout Method

Logs off the automation client from the company.

Applies To

SIVToolbar object

Syntax

object.**Logout**

The **Logout** method syntax has these parts:

Part	Description
object	An instance of **SIVToolbar**.

Remarks

After this method completes, the **Toolbar** object does not have a database context. Therefore, no applications can be started until a successful login has been completed again.

If any applications are open when the **Logout** method is called, an exception is raised.

The **Logout** method, as well as all other methods and property set operations throughout the Microsoft Dynamics SL Object Model, is synchronous with respect to the automation client. That is, control does not return to the automation client until the completion of all Microsoft Dynamics SL processing caused by the call to this method, whether the processing comes from the core Microsoft Dynamics SL kernel, the original application, or any customization of the application.

Possible Exceptions

7507 — Logout Error: Applications are still open

Example (Visual Basic 6.0 Client)

```
On Error GoTo LOGOUT_ERR
sivToolbar.Logout
Exit Sub

LOGOUT_ERR:

Select Case Err.Number
    Case 7507
        MsgBox "Can't Log out. Applications are still open."
    Case Else
        MsgBox "Unexpected Error"
End Select
Resume Next
```

Example (Visual Basic 2005 Client)

```
Try
    sivToolbar.Logout()
Catch ex As System.Runtime.InteropServices.COMException

    Dim lErrNumber As Integer
    If ex.Source.Contains("Solomon") = True Or ex.ErrorCode - vbObjectError > 0
Then
        lErrNumber = ex.ErrorCode - vbObjectError
    Else
        lErrNumber = ex.ErrorCode
    End If
```

```
    Select Case lErrNumber
        Case 7507
            MsgBox("Can't Log out. Applications are still open.")
        Case Else
            MsgBox(ex.Message, MsgBoxStyle.Exclamation, String.Format("{0}
Error {1}", ex.Source, lErrNumber.ToString()))
    End Select

    Catch ex As Exception
        MsgBox(ex.Message, MsgBoxStyle.Exclamation, String.Format("{0}
Exception", ex.Source))
    End Try
```

Message Event

Notifies the automation client of any non-fatal exceptions that the interactive user would see in the form of message boxes and allows the automation client to choose a response.

Applies To

SIVApplication

Syntax (Visual Basic 6.0 Client)

Private Sub *Object*_Message(ByVal *MessageNumber* As Integer, ByVal *MessageText* As String, ByVal *MessageType* As sivMessageType, ByRef *MessageResponse* As sivMessageResponse)

Syntax (Visual Basic 2005 Client)

```
    Private Sub sivApp_Message( _
        ByVal MessageNumber As Integer, _
        ByVal MessageText As String, _
        ByVal MessageType As
Microsoft.Dynamics.SL.ObjectModel.sivMessageType, _
        ByRef MessageResponse As
Microsoft.Dynamics.SL.ObjectModel.sivMessageResponse) _
    Handles sivApp.Message
```

The **Message** event syntax has these parts:

Part	Description
object	An instance of **SIVApplication**.
MessageNumber	An integer value indicating the number of a Microsoft Dynamics SL message as defined in the Messages.csv file.
MessageText	A string value giving the corresponding message text from Messages.csv for the Microsoft Dynamics SL message indicated by **MessageNumber**.
MessageType	A value of type **sivMessageType** indicating the type of message box (the set of standard message box buttons) that would display to the interactive user, as indicated below in Settings.
MessageResponse	A value of type **sivMessageResponse** indicating the action that the system should take in response to this message. You can set this value in the event procedure code of the Message event.

Settings

The settings for *MessageType* as are follows:

Constant	Description
sivMsgOK	Only **OK** is displayed.
sivMsgOKCancel	**OK** and **Cancel** are displayed.
sivMsgAbortRetryIgnore	**Abort**, **Retry**, and **Ignore** are displayed.
sivMsgYesNo	**Yes** and **No** are displayed.
sivMsgYesNoCancel	**Yes**, **No**, and **Cancel** are displayed.
sivMsgRetryCancel	**Retry** and **Cancel** are displayed.

The settings for *MessageResponse* are as follows:

Constant	Description
sivMsgRspOK	OK is pressed.
sivMsgRspCancel	Cancel is pressed.
sivMsgRspAbort	Abort is pressed.
sivMsgRspRetry	Retry is pressed.
sivMsgRspIgnore	Ignore is pressed.
sivMsgRspYes	Yes is pressed.
sivMsgRspNo	No is pressed.
sivMsgRspClose	Close in the message box is pressed (like the user pressing `ESC`).

Remarks

The **Message** event notifies your automation client of non-fatal messages. The **Message** event stands in for the display of non-fatal messages to the interactive user in the Microsoft Dynamics SL user interface. In other words, the **Message** event fires in the automation client at every point in the application that the interactive user would see a non-fatal message.

Whenever the **Message** event fires, the calling code that triggered the message event (typically an attempt to set a control value or to call an **SIVApplication** method) also receives runtime error 2048 from the Object Model in addition to the firing of the **Message** event.

Fatal exceptions (that is, unrecoverable runtime errors) are raised as trappable errors to the automation client by the Object Model and so are not handled by the **Message** event. Instead, the calling code receives runtime error 2049 from the Object Model.

The purpose of the *MessageNumber* and *MessageText* parameters is to tell you which message has occurred. All available *MessageNumber* and *MessageText* values for the Microsoft Dynamics SL environment are furnished in the Messages.csv file, which is a comma-delimited text file residing in the same directory as the Microsoft Dynamics SL executable. Any unidentified messages have a *MessageNumber* with value *Not found* and a *MessageText* of “Message not found.”

The purpose of the *MessageType* parameter is to tell you which buttons are available for the interactive user (in this case, the automation client) to press.

The purpose of the *MessageResponse* parameter is to let you communicate your chosen response (in the form of one of the available buttons) back to the **SIVApplication** object. If you set MessageResponse to an invalid value for the particular value of MessageType, then the value of MessageResponse defaults to the default button and runtime error 7558 is raised.

If there is no code in the **Message** event procedure to set the value of *MessageResponse*, then the system assumes the default response for that particular message.

The **Message** event is *modal* with respect to the automation object and client code. In other words, no more code in your automation client runs until the **Message** event returns. In effect, this means that the last instruction in the server application to run within the application before this event fired was the instruction that caused the event. No other server code runs until you have had a chance to respond to the message in the event procedure and the event procedure terminates.

The **Message** event is also *not re-entrant* with respect to the Microsoft Dynamics SL Object Model. This means that you cannot perform any manipulation of the Microsoft Dynamics SL Object Model, such as instantiating new object variables or calling their methods during the **Message** event’s event procedure. In other words, the **Message** event procedure must finish completely before any further Microsoft Dynamics SL Object Model processing happens in your code. If you attempt to re-enter the Microsoft Dynamics SL Object Model from within the **Message** event procedure, you receive runtime error 7554.

Note:

- This event is not available to the internal automation client.
- This event is only triggered by the use of Microsoft Dynamics SL's **MessBox** or **Mess** procedures. If you use the standard Visual Basic **MsgBox** function, then your server code does not fire the **Message** event. Furthermore, the user interface receives a modal message box, which requires interactive user response. This message box may, however, not be visible to the user and so effectively freezes your application. In addition, the use of **MsgBox** in a Microsoft SL SDK application may crash the application. If you use the Microsoft Dynamics SL Code Inspector, it warns you when you use **MsgBox** in your Microsoft SL SDK programs and offers to automatically replace it with **MessBox**.

Possible Exceptions

- 2048 — XXXX: Non-fatal Microsoft Dynamics SL exception (**Note**: This error does not occur in the **Message** event procedure itself, but instead occurs in the calling routine after the message event runs.)
- 7554 — Cannot call back into the Object Model from within the **Message** event.
- 7558 — Invalid message response for message number XXXX (**Note**: This error does not occur in the **Message** event procedure itself, but instead occurs in the calling routine after the **Message** event runs.)

Example

```
'This example first shows a calling routine that might fire
'the Message event, and then the Message event itself

'The following routine makes a call to the
'Object Model that triggers a recoverable exception
Private Sub DoSomething()
    'Assume that sivApp is an instantiated copy of
    'of SIVApplication
    sivApp.Controls("BigKey") = NonExistentValue
DoSomething_Exit:
    Exit Sub
DoSomething_Error:
    If Err.Source = "Solomon" Then
        Dim iErrNumber As Integer
        Dim iSolErr As Integer
        iErrNumber = Err.Number - vbObjectError
        Select Case iErrNumber

            Case 7558 'Message event procedure gave invalid response
```

```
            Case 2048 'non-fatal exception
            '(Message event has just fired before this)
                iSolErr = Val(Err.Description)
                Select Case iSolErr
                    Case 9 'Not found
                        'fix problem or give up
                        Resume Next
                    Case 15 'value was too small
                        'fix problem or give up
                        Resume Next
                    Case 19 'value was too large
                        'fix problem or give up
                        Resume Next
                End Select
            Case 2049 'fatal exception (no Message event)
                iSolErr = Val(Err.Description)
                'Perhaps react to iSolErr
                MsgBox "Fatal Error:" _
                       & vbCRLF & Err.Description _
                       & vbCRLF & "Application will terminate."
                'Clean up environment and exit

            Case 2050 'object model error
                MsgBox "Fatal internal object model Error:" _
                       & vbCRLF & Err.Description _
                       & vbCRLF & "Report to Microsoft tech support."
                'Clean up environment and exit
            'Cases for other object model-generated errors
            Case 7500 'etc...
        End Select
    Else
        'Non-object model error
    End If
End Sub

Private Sub sivMyApp_Message(ByVal MessageNumber As Integer, _
  ByVal MessageText As String, _
  ByVal MessageType As sivMessageType, _
  ByRef MessageResponse As _
  sivMessageResponse)

  On Error GoTo MESSAGE_ERROR:

  Select Case MessageType

      Case sivMsgOK
          sivMessageResponse = sivMsgRspOK

      Case sivMsgAbortRetryIgnore
          Select Case MessageNumber
              Case x
                  MessageResponse = sivMsgRspRetry
              Case y
                  MessageResponse = sivMsgRspIgnore
              Case Else
                  MessageResponse = sivMsgRspAbort
          End Select

      Case sivMsgYesNo
          '....etc.
```

```
      Case sivMsgYesNoCancel
          '....etc.

      Case sivMsgRetryCancel
          '....etc.

      End Select

EXIT_MESSAGE_EVENT:
  Exit Sub

MESSAGE_ERROR:

  Select Case Err.Number - vbObjectError

      Case 7554 'can't re-enter object model from Message event
          'you were trying to do something
          'with the Object Model, which is forbidden
          'here, because Message is non-re-entrant
          Resume Next

      Case Else
          MsgBox "Unexpected error #" & Err.Number & _
          ": " & Err.Description, "Message Event"
          Resume EXIT_MESSAGE_EVENT

      End Select

End Sub
```

Name Property (SIVControl Object)

The name of an instance of an **SIVControl** object.

Applies To

SIVControl Object

Syntax

Object.**Name**
Properties("**Name**")

The **Name** property syntax has these parts:

Part	Description
object	An instance of an **SIVControl** object.

Remarks

The **Name** property is display only.

You will not usually refer to the **Name** property specifically, except when trying to detect a particular control when looping through the Controls collection, or in a situation where an unknown **Control** may have been passed to a **Sub** or **Function** procedure.

Possible Exceptions

None

See Also

Controls Collection (SIVApplication Object), **SIVControl Object**

Example

```
'Using For...Each to Traverse the Controls collection and display
'the control name using the Name property of the SIVControl object

Dim ctrlCurr As SIVControl

For Each ctrlCurr in sivMyApp.Controls
    MsgBox ctrlCurr.Name
Next ctrlCurr
```

Name Property (SIVProperty Object)

The name of an instance of a **SIVProperty** object.

Applies To

SIVProperty object

Syntax

Object.**Name**
Properties("**Name**")

The **Name** property syntax has these parts:

Part	Description
object	An instance of a **Property** object.

Remarks

The **Name** property is also the same as the contents of the string-type key that you use to refer to a control in the Properties collection.

The **Name** property is display only.

You will not usually refer to the **Name** property specifically, except when trying to detect a particular property when looping through the Properties collection, or in a situation where an unknown **Property** may have been passed to a **Sub** or **Function** procedure.

Possible Exceptions

None

See Also

Controls Collection (SIVApplication Object), **Properties Collection (SIVControl Object)**, **SIVControl Object**, **SIVProperty Object**

Example

```
'Assume that sivCtrlCurr is an
'instance of an SIVControl

Dim blnSupportsLobsterProperty As Boolean
Dim sivPropCurr As SIVProperty

For Each sivPropCurr In sivCtrlCurr.Properties.Item
    If sivPropCurr.Name = "Lobster" Then
        blnSupportsLobsterProperty = True
        Exit For
    End If
Next sivPropCurr

If blnSupportsLobsterProperty Then
    sivCtrlCurr.Properties.Item("Lobster").Value = True
End If
```

New Method

Performs a **New** action for a specified entity and all its dependent entities on a particular screen.

Applies To

SIVApplication

Syntax

object.**New** *Entity*

The **New** method syntax has these parts:

Part	Description
object	An instance of **SIVApplication**.
Entity	Required string. Corresponds to one of the **SIVApplication** object's data entities. (**Note:** This is the Microsoft Dynamics SL data level.) See remarks below.

Remarks

The **New** method is analogous to the interactive user clicking **New** or pressing `CTRL+N`.

When the **New** method executes successfully, then the controls bound to the specified *Entity* are set to their default starting values, and so are all the controls bound to all entities that are dependent on the specified *Entity*.

If the **SIVApplication** object is visible to the user, then the controls bound to the *Entity* and its subentities display with their default initialization values on the screen.

The **New** method, as well as all other methods and property set operations throughout the Microsoft Dynamics SL Object Model, is *synchronous* with respect to the automation client. That is, control does not return to the automation client until the completion of all Microsoft Dynamics SL processing caused by the call to this method, whether the processing comes from the core Microsoft Dynamics SL kernel, the original application, or any customization of the application.

Possible Exceptions

- 7524 — **New** method is disabled for the specified entity, either by the application or because of access rights
- 7525 — Invalid entity string

See Also

SIVApplication Object

Example (Visual Basic 6.0 Client)

```
'Assume that sivMyApp is an instance of
'SIVApplication

On Error GoTo NEW_ERROR
sivMyApp.New "Batch"
Exit Sub

NEW_ERROR:

Select Case Err.Number - vbObjectError
    Case 7524
        MsgBox "New method disabled"
        Resume Next
    Case 7525
```

```
        MsgBox "Invalid entity 'Batch'"
        Resume Next
    Case Else
        MsgBox Err.Number & ": " & Err.Description, _
        "UNEXPECTED ERROR"
End Select
```

Example (Visual Basic 2005 Client)

```
'Assume that sivMyApp is an instance of
'SIVApplication
Try
    sivMyApp.[New]("Batch")
Catch ex As System.Runtime.InteropServices.COMException

    Dim lErrNumber As Integer
    If ex.Source.Contains("Solomon") = True Or ex.ErrorCode - vbObjectError > 0
Then
        lErrNumber = ex.ErrorCode - vbObjectError
    Else
        lErrNumber = ex.ErrorCode
    End If

    Select Case lErrNumber
        Case 7524
            MsgBox("New method disabled")
        Case 7525
            MsgBox("Invalid entity 'Batch'")
        Case Else
            MsgBox(ex.Message, MsgBoxStyle.Exclamation, String.Format("{0}
Error {1}", ex.Source, lErrNumber.ToString()))
    End Select

Catch ex As Exception
    MsgBox(ex.Message, MsgBoxStyle.Exclamation, String.Format("{0} Exception",
ex.Source))
End Try
```

Next Method

Navigates to the next record for a specified entity.

Applies To

SIVApplication

Syntax

object.Next *Entity*

The Next method syntax has these parts:

Part	Description
object	An instance of SIVApplication.
Entity	Required string. Corresponds to one of the SIVApplication object's data entities. (Note: This is the Microsoft Dynamics SL data level.) See remarks below.
RetVal	A value of type sivRecordFound. Corresponds to one of the values described below under the Returns section.

Returns

One of the two following values:

- sivRecFndNotFound – 1 — Means that no record was located upon execution of the Next method.
- sivRecFndFound - 2 — Means that a record was successfully located upon execution of the Next method.

Remarks

Navigates to the next record of the entity specified by *Entity* for the SIVApplication object. You can obtain the *Entity* string for a particular set of controls by getting the Level property from the Properties collection of one of the controls.

If the current record was already the last record in *Entity*, then the record pointer changes to point to a new record, the return value is sivRecFndNotFound, and no exception is generated.

The Next method is equivalent to the user action of clicking the Next navigation button on the toolbar.

If the Next method is successful (return value was sivRecFndFound) and the SIVApplication object is visible to the user, then the new record's contents display in the application screen on the user's desktop.

The Next method, as well as all other methods and property set operations throughout the Microsoft Dynamics SL Object Model, is *synchronous* with respect to the automation client. That is, control does not return to the automation client until the completion of all Microsoft Dynamics SL processing caused by the call to this method, whether the processing comes from the core Microsoft Dynamics SL kernel, the original application, or any customization of the application.

Possible Exceptions

- 7525 — Invalid entity string
- 7532 — Next method is disabled for this entity

Example (Visual Basic 6.0 Client)

```
'Assumes sivMyApp is an already-instantiated
'SIVApplication object
On Error GoTo NAVIGATE_ERROR
```

```
Dim sivrfResult As sivRecordFound
sivrfResult = sivMyApp.Next ("Batch")
If sivrfResult = sivRecFndNotFound Then
    MsgBox "Navigated to a new Record"
End If
Exit Sub
NAVIGATE_ERROR:
Select Case Err.Number - vbObjectError
    Case 7525
        MsgBox "Invalid entity 'Batch'"
        Resume Next
    Case 7524
        MsgBox "Next method disabled"
        Resume Next
    Case Else
        MsgBox Err.Number & ": " & Err.Description, _
        "UNEXPECTED ERROR"
End Select
```

Example (Visual Basic 2005 Client)

```
'Assumes sivMyApp is an already-instantiated
'SIVApplication object
Dim sivrfResult As sivRecordFound

Try
    sivrfResult = sivMyApp.Next("Batch")
    If sivrfResult = sivRecordFound.sivRecFndNotFound Then
        MsgBox("Navigated to a new Record")
    End If

Catch ex As System.Runtime.InteropServices.COMException

    Dim lErrNumber As Integer
    If ex.Source.Contains("Solomon") = True Or ex.ErrorCode - vbObjectError > 0
Then
        lErrNumber = ex.ErrorCode - vbObjectError
    Else
        lErrNumber = ex.ErrorCode
    End If

    Select Case lErrNumber
        Case 7525
            MsgBox("Invalid entity 'Batch'")
        Case 7524
            MsgBox("Next method disabled")
        Case Else
            MsgBox(ex.Message, MsgBoxStyle.Exclamation, String.Format("{0}
Error {1}", ex.Source, lErrNumber.ToString()))
    End Select

Catch ex As Exception
    MsgBox(ex.Message, MsgBoxStyle.Exclamation, String.Format("{0} Exception",
ex.Source))
End Try
```

Notes/Attachments Icon (NoteButton Control)

Remarks

Not supported in this version of the Microsoft Dynamics SL Object Model.

Previous Method

Navigates to the previous record for a specified entity.

Applies To

SIVApplication
object.**Previous** Entity

The **Previous** method syntax has these parts:

Part	Description
object	An instance of **SIVApplication**.
Entity	Required string. Corresponds to one of the **SIVApplication** object's data entities. (**Note:** this is the Microsoft Dynamics SL data level.) See remarks below.
RetVal	A value of type **sivRecordFound**. Corresponds to one of the values described below under the Returns section.

Returns

One of the two following values:

- **sivRecFndNotFound - 1** — Means that no record was located upon execution of the **Previous** method.
- **sivRecFndFound - 2** — Means that a record was successfully located upon execution of the **Previous** method.

Remarks

Navigates to the previous record of the entity specified by *Entity* for the **SIVApplication** object. You can obtain the *Entity* string for a particular set of controls by getting the **Level** property of one of the controls.

If the current record was already the first record in *Entity*, then the record pointer changes to point to a new record, the return value is **sivRecFndNotFound**, and no exception is generated.

The **Previous** method is equivalent to the user action of clicking the **Previous** navigation button on the toolbar.

If the **Previous** method is successful (return value was **sivRecFndFound**) and the **SIVApplication** object is visible to the user, then the new record's contents display in the application screen on the user's desktop.

The **Previous** method, as well as all other methods and property set operations throughout the Microsoft Dynamics SL Object Model, is synchronous with respect to the automation client. That is, control does not return to the automation client until the completion of all Microsoft Dynamics SL processing caused by the call to this method, whether the processing comes from the core Microsoft Dynamics SL kernel, the original application, or any customization of the application.

Possible Exceptions

- 7525 — Invalid entity string
- 7534 — **Previous** method (and other table navigation) is disabled for this entity

Example (Visual Basic 6.0 Client)

```
'Assumes sivMyApp is an already-instantiated
'SIVApplication object

On Error GoTo NAVIGATE_ERROR
```

```
Dim sivrfResult As sivRecordFound
sivrfResult = sivMyApp.Previous("Batch")
If sivrfResult = sivRecFndNotFound Then
    MsgBox "Navigated to a new Record"
End If
Exit Sub
NAVIGATE_ERROR:
Select Case Err.Number - vbObjectError
    Case 7525
        MsgBox "Invalid entity 'Batch'"
        Resume Next
    Case 7534
        MsgBox "Previous method disabled"
        Resume Next
    Case Else
        MsgBox Err.Number & ": " & Err.Description, _
        "UNEXPECTED ERROR"
End Select
```

Example (Visual Basic 2005 Client)

```
'Assumes sivMyApp is an already-instantiated
'SIVApplication object

Dim sivrfResult As sivRecordFound

Try
    sivrfResult = sivMyApp.Previous("Batch")
    If sivrfResult = sivRecordFound.sivRecFndNotFound Then
        MsgBox("Navigated to a new Record")
    End If

Catch ex As System.Runtime.InteropServices.COMException

    Dim lErrNumber As Integer
    If ex.Source.Contains("Solomon") = True Or ex.ErrorCode - vbObjectError > 0
Then
        lErrNumber = ex.ErrorCode - vbObjectError
    Else
        lErrNumber = ex.ErrorCode
    End If

    Select Case lErrNumber
        Case 7525
            MsgBox("Invalid entity 'Batch'")
        Case 7534
            MsgBox("Previous method disabled")
        Case Else
            MsgBox(ex.Message, MsgBoxStyle.Exclamation, String.Format("{0}
Error {1}", ex.Source, lErrNumber.ToString()))
    End Select

Catch ex As Exception
    MsgBox(ex.Message, MsgBoxStyle.Exclamation, String.Format("{0} Exception",
ex.Source))
End Try
```

Properties Collection (SIVControl Object)

Returns a reference to a collection of **SIVProperty** objects.

Applies To

SIVControl Object

Syntax

object.**Properties**

where *object* represents an instance of **SIVControl**.

Methods

Count method, **Item** method

Remarks

You can manipulate **SIVProperty** objects using the reference returned by the **Properties** property (see the following examples).

Possible Exceptions

None

See Also

SIVControl Object

Example (Visual Basic 6.0 Client)

```
'Set a property object variable
'to point to a property in a SIVControl's Properties collection
Dim propCurr As SIVProperty
Set propCurr = currCtrl.Properties.Item("BackColor")
propCurr.Value = vbRed

'Change a control directly
sivMyApp.Controls("cTotalBalance").Properties.Item("BackColor") = vbRed
```

Example (Visual Basic 2005 Client)

```
'This example uses a constant from VBRUN.DLL.  Dynamics SL
'provides an interop for VBRUN.DLL called Interop.VBRUN.DLL.
'This program requires a reference to this assembly to use
'the “vbRed” constant.

'Set a property object variable
'to point to a property in a SIVControl's Properties collection
Dim propCurr As SIVProperty
propCurr = currCtrl.Properties("BackColor")
propCurr.Value = VBRUN.ColorConstants.vbRed

'Change a control directly
sivMyApp.Controls("cTotalBalance").Properties("BackColor").Value =
VBRUN.ColorConstants.vbRed
```

Quit Method

Causes either the toolbar or the Microsoft SL SDK application to shut down.

Applies To

SIVToolbar, SIVApplication

Syntax

object.**Quit**

The **Quit** method syntax has these parts:

Part	Description
object	An instance of **SIVToolbar** or **SIVApplication**.

Remarks: SIVToolbar

Calling the **Quit** method on the **SIVToolbar** object is equivalent to the user clicking the toolbar's close box (the box with the "X" in the upper right corner of the toolbar's window) or choosing **File | Exit** from the toolbar menu.

The **SIVToolbar** object must be logged off before its **Quit** method is called. This ensures that all applications are closed before the toolbar is shut down (one of the requirements for the **Logout** method is that all applications must be closed).

After the **Quit** method has been called, the **Toolbar** object is destroyed when its **Dispose** method is called. The toolbar shuts down only after **Quit** has been called, and the toolbar object has been destroyed.

If you destroy the instance of the **SIVToolbar** object but do not call the **Quit** method, the connection between the toolbar and the automation client is broken, and the toolbar automatically becomes visible to the interactive user.

If an automation client holds a valid reference to a visible instance of **SIVToolbar**, then the interactive user will be unable to close it.

Remarks: SIVApplication

Calling the **Quit** method on the **SIVApplication** object is equivalent to the interactive user clicking **Close** on the toolbar.

After the **Quit** method has been called, the **SIVApplication** object will be destroyed by calling its **Dispose** method. The application only shuts down after **Quit** has been called and the **SIVApplication** object has been destroyed.

If you destroy the instance of the **SIVApplication** object but do not call the **Quit** method, the connection between the application and the automation client is broken, and the application automatically becomes visible to the interactive user.

Any exceptions raised during the shutdown of the application, such as prompts to save changes or warnings about incorrect data settings, prevent the application from shutting down.

If an automation client holds a valid reference to a visible instance of **SIVApplication**, then the interactive user is unable to close it.

Remarks: Common

The **Quit** method, as well as all other methods and property set operations throughout the Microsoft Dynamics SL Object Model, is synchronous with respect to the automation client. That is, control does not return to the automation client until the completion of all Microsoft Dynamics SL processing caused by the call to this method, whether the processing comes from the core Microsoft Dynamics SL kernel, the original application, or any customization of the application.

Possible Exceptions

7522 – Automation client is not logged out.

Example (Visual Basic 6.0 Client)

```
'following code assumes that sivtbCurrent
'is a SIVToolbar object that has already
'successfully logged in
    On Error GoTo QUIT_ERR
    sivToolbar.Quit
    Set SIVToolbar = Nothing
    Exit Sub
QUIT_ERR:
    Select Case Err.Number - vbObjectError
        Case 7522
    End Select
```

Example (Visual Basic 2005 Client)

```
'following code assumes that sivToolbar
'is a SIVToolbar object that has already
'successfully logged in
Try

    sivToolbar.Quit()
    sivToolbar.Dispose()
    sivToolbar = Nothing

Catch ex As System.Runtime.InteropServices.COMException

    Dim lErrNumber As Integer
    If ex.Source.Contains("Solomon") = True Or ex.ErrorCode - vbObjectError > 0
Then
        lErrNumber = ex.ErrorCode - vbObjectError
    Else
        lErrNumber = ex.ErrorCode
    End If

    Select Case lErrNumber
        Case 7522
            MsgBox("Automation client is not logged out")
        Case Else
            MsgBox(ex.Message, MsgBoxStyle.Exclamation, String.Format("{0}
Error {1}", ex.Source, lErrNumber.ToString()))
    End Select

Catch ex As Exception
    MsgBox(ex.Message, MsgBoxStyle.Exclamation, String.Format("{0} Exception",
ex.Source))
End Try
```

Relative Date Dialog

Remarks

Not supported in this version of the Microsoft Dynamics SL Object Model.

SAFCheck Control (SIVControl Object)

Refers to the Microsoft Dynamics SL **SAFCheck** control.

Applies To

Controls collection of **SIVApplication** object

Syntax

Object.Controls.(“*SAFCheck Name*”)

The **SAFCheck** control syntax has these parts:

Part	Description
object	An instance of **SIVApplication**.
SAFCheck Name	Required string. Corresponds to the name of an **SAFCheck** control exposed by the **SIVApplication** object.

Remarks

The instance of an **SIVControl** object that contains an **SAFCheck** control has a **Value** property that can be either True for a checked box or False for an unchecked box. Note that **SAFCheck** only supports these two values as opposed to the standard Visual Basic **CheckBox** control’s *vbChecked*, *vbUnChecked*, and *vbGrayed*.

Note: Value can only be True or False, regardless of the TrueText or FalseText setting of the control.

As with all **SIVControl** objects, you can perform this manipulation either by setting the **Value** property of the **SIVControl** instance that holds the **SAFCheck**, or you can implicitly refer to the **Value** by simply referring to the **SIVControl** instance itself (see the examples below).

You can also manipulate the **SAFCheck**’s Microsoft Dynamics SL and standard Visual Basic properties through the Object Model by using the **SIVControl** instance’s Properties collection of **SIVProperty** objects (see examples).

For further information about the behavior of the **SAFCheck** control, see Microsoft SL SDK documentation.

Possible Exceptions

None

See Also

SIVControl Object, **Value Property (SIVControl Object)**

Example (Visual Basic 6.0 Client)

```
'Following code assumes that sivMyApp
'is an instance of SIVApplication
'Following are three different ways
'to manipulate a SAFCheck control on
'the application screen:

'this way --
Dim sivctrlc1099 As SIVControl
Set sivctrlc1099 = sivMyApp.Controls("c1099")
sivctrlc1099 = True
sivctrlc1099.Properties.Item("Font.Bold") = True
```

```
'or this way --
sivMyApp.Controls("c1099") = True
sivMyApp.Controls("c1099").Properties.Item("Font.Bold") = True
'or this way --
sivMyApp.Controls("c1099").Value = True
sivMyApp.Controls("c1099").Properties.Item("Font.Bold") = True
```

Example (Visual Basic 2005 Client)

```
'Following code assumes that sivMyApp
'is an instance of SIVApplication
'Following are two different ways
'to manipulate a SAFCheck control on
'the application screen:

'this way --
Dim sivctrlc1099 As SIVControl
sivctrlc1099 = sivMyApp.Controls("c1099")
sivctrlc1099.Value = True
sivctrlc1099.Properties("Font").Value.Bold = True

'or this way --
sivMyApp.Controls("c1099").Value = True
sivMyApp.Controls("c1099").Properties("Font").Value.Bold = True
```

SAFCombo Control (SIVControl Object)

Refers to the Microsoft Dynamics SL **SAFCombo** control.

Applies To

Controls collection of **SIVApplication** object

Syntax

Object.Controls.(“*SAFCombo Name*”)

The **SAFCombo** control syntax has these parts:

Part	Description
object	An instance of **SIVApplication**.
SAFCombo Name	Required string. Corresponds to the name of an **SAFCombo** control exposed by the **SIVApplication** object.

Remarks

The instance of an **SIVControl** object that contains an **SAFCombo** control has a **Value** property that corresponds to the current contents of the selected list item text in the **ComboBox**. You cannot add text that does not already exist as an option in the **ComboBox** list. Beware that **SAFCombo** lists do not generally correspond to the exact contents of underlying data fields.

If a Microsoft Dynamics SL customization has disabled a list entry, then you will not be able to set the **Value** property to that list entry and an error is raised. If you attempt to set an **SAFCombo** list entry that is disabled, then you receive runtime error 7537.

As with all **SIVControl** objects, you can perform this manipulation by setting the **Value** property of the **SIVControl** instance that holds the **SAFCombo**, or you can implicitly refer to the **Value** by simply referring to the **SIVControl** instance itself (see the examples below).

Note about customized list values: Since you must refer to the list values available to the interactive user in order to manipulate an **SAFCombo**’s current setting, you might be concerned about what will happen to your client automation code if someone customizes the Microsoft Dynamics SL application to display different list values. Fortunately, the Microsoft Dynamics SL Object Model is intelligent enough to check underlying, un-customized values of **SAFCombo** list items as well as current, visible values. For example, consider an **SAFCombo** named *cOrdType* with standard list values “Credit Memo,” “Debit Memo,” “Invoice,” and “Quote.” You have written automation client code that changes the value of *cOrdType* to “Invoice” in a particular situation. If someone customizes their Microsoft Dynamics SL installation so that the list values now display as “Nota de Credito,” “Nota de Debito,” “Factura,” and “Cotizacion,” your original automation code still works with “Invoice” because the Microsoft Dynamics SL Object Model examines both the customized and the original values of the list whenever you attempt to assign a list value in your automation client code.

The **List** property is display only and consists of a delimited string that contains the entries visible to the user in the combo box, as well as the underlying values that are stored in the data for each visible list entry. The format of the string is *Data Value*;*Visible Value*[, *Data Value*;*Visible Value*[...]]. In the previous example for *cOrdType*, the original uncustomized list would look like: “C;Credit Memo,D;Debit Memo,I;Invoice,Q;Quote”.

You can also manipulate the **SAFCombo**’s Microsoft Dynamics SL and standard Visual Basic properties through the Microsoft Dynamics SL Object Model by using the **SIVControl** instance’s Properties collection of **SIVProperty** objects (see examples).

For further information about the behavior of the **SAFCombo** control, see Microsoft SL SDK documentation.

Possible Exceptions

- 7537 — Specified list value is disabled
- 7538 — Specified list value does not exist

See Also

SIVControl Object, Value Property (SIVControl Object)

Example (Visual Basic 6.0 Client)

```
'Following code assumes that sivMyApp
'is an instance of SIVApplication

'Following are three different ways
'to manipulate a SAFCombo control on
'the application screen:
'this way --
Dim sivctrlcEmpType As SIVControl
Set sivctrlcEmpType = sivMyApp.Controls("cEmpType")
sivctrlcEmpType = "Exempt"
'or this way --
sivMyApp.Controls("cEmpType") = "Exempt"

'or this way --
sivMyApp.Controls("cEmpType").Value = "Exempt"
'NOTE: The value "Exempt" will work even if the screen
'has been customized to use a different string for the
'"Exempt" option.
```

Example (Visual Basic 2005 Client)

```
'Following code assumes that sivMyApp
'is an instance of SIVApplication

'Following are two different ways
'to manipulate a SAFCombo control on
'the application screen:
'this way --
Dim sivctrlcEmpType As SIVControl
sivctrlcEmpType = sivMyApp.Controls("cEmpType")
sivctrlcEmpType.Value = "Exempt"

'or this way --
sivMyApp.Controls("cEmpType").Value = "Exempt"

'NOTE: The value "Exempt" will work even if the screen
'has been customized to use a different string for the
'"Exempt" option.
```

SAFContainer Control

Remarks

Not supported in this version of the Microsoft Dynamics SL Object Model.

SAFFloat Control (SIVControl Object)

Refers to the Microsoft Dynamics SL **SAFFloat** control.

Applies To

Controls collection of **SIVApplication** object

Syntax

Object.Controls.(“*SAFFloat Name*”)

The **SAFFloat** control syntax has these parts:

Part	Description
object	An instance of **SIVApplication**.
SAFFloat Name	Required string. Corresponds to the name of an **SAFFloat** control exposed by the **SIVApplication** object.

Remarks

The instance of an **SIVControl** object that contains an **SAFFloat** control has a **Value** property that corresponds to the current contents of the control as seen by the interactive user.

You can assign to the **Value** property any numeric type compatible with a **Double** type, or you can assign a **String Variant** type. If it is a string, however, then the string cannot contain any currency symbols or commas and, if it contains a decimal point, then the decimal point must be US-format decimal float (“.”). In other words, the only legal characters are the digits, the positive and negative signs (“+” and “-”), and the US-format decimal point. Attempting to assign other characters will generate an exception.

The **Value** property’s return type is always a **Variant** that contains a **Double**.

As with all **SIVControl** objects, you can manipulate this value either by setting the **Value** property of the **SIVControl** instance that holds the **SAFFloat**, or you can implicitly refer to the **Value** by simply referring to the **SIVControl** instance itself (see examples below).

You can also manipulate the **SAFFloat**’s Microsoft Dynamics SL and standard Visual Basic properties through the Microsoft Dynamics SL Object Model by using the **SIVControl** instance’s Properties collection of **SIVProperty** objects (see examples).

For further details about the behavior of the **SAFFloat** control, see the Microsoft SL SDK documentation.

Possible Exceptions

- 7539 — Incorrect data format for SAFFloat control
- 7561 — Attempt to set number of decimal places to be greater than the number of decimal places allowed by the control

See Also

SIVControl Object, **Value Property (SIVControl Object)**

Example (Visual Basic 6.0 Client)

```
'Following code assumes that sivMyApp
'is an instance of SIVApplication.

'Following are three different ways
'to manipulate a SAFFloat control on
'the application screen:
'this way (notice use of string) --
Dim sivctrlcTotal As SIVControl
Set sivctrlcTotal = sivMyApp.Controls("cTotal")
sivctrlcTotal = "123456789.32"
'or this way (notice use of numeric literal) --
sivMyApp.Controls("cTotal") = 123456789.32
'or this way --
sivMyApp.Controls("cTotal").Value = 123456789.32

'NOTE: each of the following examples would raise an exception,
'because the numeric strings are incorrectly formatted:
sivctrlcTotal = "123,456,789.32" 'must NOT use thousands separator
sivctrlcTotal = "123456789,32" 'MUST use US decimal point
sivctrlcTotal = "$123456789.32" 'must NOT use currency symbol
sivctrlcTotal = "BAL=123456789.32" 'must NOT use non-numeric chars
```

Example (Visual Basic 2005 Client)

```
'Following code assumes that sivMyApp
'is an instance of SIVApplication.

'Following are two different ways
'to manipulate a SAFFloat control on
'the application screen:
'this way (notice use of string) --
Dim sivctrlcTotal As SIVControl
sivctrlcTotal = sivMyApp.Controls("cTotal")
sivctrlcTotal.Value = "123456789.32"

'or this way (notice use of numeric literal) --
sivMyApp.Controls("cTotal").Value = 123456789.32

'NOTE: each of the following examples would raise an exception,
'because the numeric strings are incorrectly formatted:
sivctrlcTotal.Value = "123,456,789.32" 'must NOT use thousands separator
sivctrlcTotal.Value = "123456789,32" 'MUST use US decimal point
sivctrlcTotal.Value = "$123456789.32" 'must NOT use currency symbol
sivctrlcTotal.Value = "BAL=123456789.32" 'must NOT use non-numeric chars
```

SAFGrid Control

Remarks

Not available through the Microsoft Dynamics SL Object Model.

However, it certainly is possible to access data exposed in grids on Microsoft SL SDK application screens.

SAFInteger Control (SIVControl Object)

Refers to the Microsoft Dynamics SL **SAFInteger** control.

Applies To

Controls collection of **SIVApplication** object

Syntax

Object.Controls.("*SAFInteger Name*")

The **SAFInteger** control syntax has these parts:

Part	Description
object	An instance of **SIVApplication**.
SAFInteger Name	Required string. Corresponds to the name of an **SAFInteger** control exposed by the **SIVApplication** object.

Remarks

The instance of an **SIVControl** object that contains an **SAFInteger** control has a **Value** property that corresponds to the current contents of the control as seen by the interactive user.

You can assign to the **Value** property any numeric type compatible with an **Integer** type, or you can assign a **String Variant** type. If it is a string, however, then the string cannot contain any currency symbols, commas or decimal points. In other words, the only legal characters are the digits and positive and negative signs ("+" and "-"). Attempting to assign other characters will generate an exception. Attempting to assign numeric types or values that do not fit in the range of an integer also will generate an exception.

The **Value** property's return type is always a **Variant** that contains an integer.

As with all **SIVControl** objects, you can manipulate this value either by setting the **Value** property of the **SIVControl** instance that holds the **SAFInteger**, or you can implicitly refer to the **Value** by simply referring to the **SIVControl** instance itself (see examples below).

You can also manipulate the **SAFInteger**'s Microsoft Dynamics SL and standard Visual Basic properties through the Microsoft Dynamics SL Object Model by using the **SIVControl** instance's Properties collection of **SIVProperty** objects (see examples).

For further particulars about the behavior of the **SAFInteger** control, see the Microsoft SL SDK documentation.

Possible Exceptions

7540 — Incorrect data format for SAFInteger control

See Also

SIVControl Object, **Value Property (SIVControl Object)**

Example (Visual Basic 6.0 Client)

```
'Following code assumes that sivMyApp
'is an instance of SIVApplication.

'Following are three different ways
'to manipulate a SAFInteger control on
'the application screen:

'this way (notice use of string) --
Dim sivctrlcCount As SIVControl
```

```
Set sivctrlcCount = sivMyApp.Controls("cCount")
sivctrlcCount = "12345"
'or this way (notice use of numeric literal) --
sivMyApp.Controls("cCount") = 12345
'or this way --
sivMyApp.Controls("cCount").Value = 12345
'NOTE: each of the following examples would raise an exception,
'because the numeric strings are incorrectly formatted or outside
'the range allowed for an Integer:
sivctrlcCount = "12,345" 'must NOT use thousands separator
sivctrlcTotal = "12345.32" 'MUST NOT use decimals of any kind
sivctrlcTotal = "$12345" 'must NOT use currency symbol
sivctrlcTotal = "BAL=12345" 'must NOT use non-numeric chars
sivctrlcTotal = "123456789" 'must NOT use incompatible numeric types
```

Example (Visual Basic 2005 Client)

```
'Following code assumes that sivMyApp
'is an instance of SIVApplication.

'Following are two different ways
'to manipulate a SAFInteger control on
'the application screen:
'this way (notice use of string) --
Dim sivctrlcCount As SIVControl
sivctrlcCount = sivMyApp.Controls("ccycle")
sivctrlcCount.Value = "12345"

'or this way (notice use of numeric literal) --
sivMyApp.Controls("ccycle").Value = 12345

'NOTE: each of the following examples would raise an exception,
'because the numeric strings are incorrectly formatted or outside
'the range allowed for an Integer:
sivctrlcCount.Value = "12,345" 'must NOT use thousands separator
sivctrlcCount.Value = "12345.32" 'MUST NOT use decimals of any kind
sivctrlcCount.Value = "$12345" 'must NOT use currency symbol
sivctrlcCount.Value = "BAL=12345" 'must NOT use non-numeric chars
sivctrlcCount.Value = 123456789" 'must NOT use incompatible numeric types
```

SAFMaskedText Control (SIVControl Object)

Refers to the Microsoft Dynamics SL **SAFMaskedText** control.

Applies To

Controls collection of **SIVApplication** object

Syntax

Object.Controls.(“*SAFMaskedText Name*”)

The **SAFMaskedText** control syntax has these parts:

Part	Description
object	An instance of **SIVApplication**.
SAFMaskedText Name	Required string. Corresponds to the name of an **SAFMaskedText** control exposed by the **SIVApplication** object.

Remarks

The instance of an **SIVControl** object that contains an **SAFMaskedText** control has a **Value** property that corresponds to the current contents of the control as seen by the interactive user.

The **Value** property of an **SAFMaskedText** is a **Variant** that contains a string. If the contents of the string that you attempt to assign to the **SAFMaskedText** do not agree with the **Mask** property of the **SAFMaskedText** control, then an exception will be generated. When you assign a character string to the **Mask**, then you must include in the string non-significant formatting characters that are part of the **Mask**, such as dashes or slashes.

You can also manipulate the **SAFMaskedText**’s Microsoft Dynamics SL and standard Visual Basic properties through the Microsoft Dynamics SL Object Model by using the **SIVControl** instance’s Properties collection of **SIVProperty** objects (see examples).

For further particulars about the behavior of the **SAFMaskedText** control, see the Microsoft SL SDK documentation.

Possible Exceptions

- 7541 — Data does not match mask
- 7553 — Data is too long to fit in field

See Also

SIVControl Object, **Value Property (SIVControl Object)**

Example (Visual Basic 6.0 Client)

```
'Following code assumes that sivMyApp
'is an instance of SIVApplication.
'Assume a SAFMaskedText control for Social Security Number.
'Its mask would be "999-99-9999"

'Note that when assigning characters to fit the mask,
'it is not necessary to insert formatting characters
'(in this case, the dashes "-").

'Following are three different ways
'to manipulate a SAFMaskedText control on
'the application screen:

'this way --
Dim sivctrlcSSN As SIVControl
```

```
Set sivctrlcSSN = sivMyApp.Controls("cSSN")
sivctrlcSSN = "596708972"

'or this way --
sivMyApp.Controls("cSSN") = "596708972"

'or this way --
sivMyApp.Controls("cSSN").Value = "596708972"

'NOTE: each of the following examples would raise an exception,
'because the numeric strings are incorrectly formatted or outside
'the range allowed for an Integer:

sivctrlcSSN = "5967089723" 'too long for Mask

sivctrlcSSN = "596AA8972" 'alpha characters where Mask is numeric
```

Example (Visual Basic 2005 Client)

```
'Following code assumes that sivMyApp
'is an instance of SIVApplication.
'Assume a SAFMaskedText control for Social Security Number.
'Its mask would be "999-99-9999"
'Note that when assigning characters to fit the mask,
'it is not necessary to insert formatting characters
'(in this case, the dashes "-").

'Following are two different ways
'to manipulate a SAFMaskedText control on
'the application screen:

'this way --
Dim sivctrlcSSN As SIVControl
sivctrlcSSN = sivMyApp.Controls("cSSN")
sivctrlcSSN.Value = "596708972"

'or this way --
sivMyApp.Controls("cSSN").Value = "596708972"

'NOTE: each of the following examples would raise an exception,
'because the numeric strings are incorrectly formatted or outside
'the range allowed for an Integer:
sivctrlcSSN.Value = "5967089723" 'too long for Mask
sivctrlcSSN.Value = "596AA8972"  'aplha characters where Mask is numeric
```

SAFOption Control (SIVControl Object)

Refers to the Microsoft Dynamics SL **SAFOption** control.

Applies To

Controls collection of **SIVApplication** object

Syntax

Object.Controls.(“*SAFOption Name*”)

The **SAFOption** control syntax has these parts:

Part	Description
object	An instance of **SIVApplication**.
SAFOption Name	Required string. Corresponds to the name of an **SAFOption** control exposed by the **SIVApplication** object.

Remarks

The instance of an **SIVControl** object that contains an **SAFOption** control has a **Value** property that can be either True for a selected **SAFOption** or False for an unselected **SAFOption**.

Note: Value can only be True or False, regardless of the **TrueText** or **FalseText** setting of the control.

SAFOption controls are commonly found in control arrays.

When you need to refer to an **SIVControl** that points to a member of a control array in the underlying application, then you must use the control array index as part of the index element that you pass as an argument to the Controls collection. For example, you would refer to element 2 of a control array named optCreditCard of **SAFOption** controls as follows:

```
sivMyApp.Controls(“optCreditCard(2)”)
```

As with all **SIVControl** objects, you can perform this manipulation either by setting the **Value** property of the **SIVControl** instance that holds the **SAFOption**, or you can implicitly refer to the **Value** by simply referring to the **SIVControl** instance itself (see the examples below).

You can also manipulate the **SAFOption**’s Microsoft Dynamics SL and standard Visual Basic properties through the Microsoft Dynamics SL Object Model by using the **SIVControl** instance’s Properties collection of **SIVProperty** objects (see examples).

For further particulars about the behavior of the **SAFOption** control, see the Microsoft SL SDK documentation.

Possible Exceptions

None

See Also

SIVControl Object, **Value Property (SIVControl Object)**

Example (Visual Basic 6.0 Client)

```
'Following code assumes that sivMyApp
'is an instance of SIVApplication
'Following are three different ways
'to manipulate a SAFOption control on
'the application screen (note that this
'control is element 2 of a control array):

'this way --
Dim sivctrlcCreditType2 As SIVControl
```

```
Set sivctrlcCreditType2 = sivMyApp.Controls("cCreditType(2)")
sivctrlcCreditType2 = True
'or this way --
sivMyApp.Controls("cCreditType(2)") = True

'or this way --
sivMyApp.Controls("cCreditType(2)").Value = True
```

Example (Visual Basic 2005 Client)

```
'Following code assumes that sivMyApp
'is an instance of SIVApplication
'Following are two different ways
'to manipulate a SAFOption control on
'the application screen (note that this
'control is element 2 of a control array):
'this way --
Dim sivctrlcCreditType2 As SIVControl
Set sivctrlcCreditType2 = sivMyApp.Controls("cCreditType(2)")
sivctrlcCreditType2.Value = True
'or this way --
sivMyApp.Controls("cCreditType(2)").Value = True
```

SAFUpdate Control

Remarks

Not available through the Microsoft Dynamics SL Object Model.

Save Method

Saves all the data in the current screen.

Applies To

SIVApplication

Syntax

object.**Save**

The **Save** method syntax has these parts:

Part	Description
object	An instance of **SIVApplication**.

Remarks

When you call the **Save** method, you perform the same action as interactive users perform when they click the **Save** button on the toolbar or key `CTRL+S`.

The **Save** method causes the contents of all bound controls on the application screen to be written to their underlying data fields.

It is possible that the underlying application or a customization may have disabled the **Save** button, or the current user ID may not have rights to save for the current application. In such cases, runtime error 7562 occurs.

Although there are no other specific runtime error codes or messages related to **Save** method exceptions, the attempt to save a screen's information can generate many validation exceptions due to the fact that the data is not ready to be saved.

You should trap for such exceptions, either in the **SIVApplication**'s **Message** event procedure, or as runtime errors in the procedure where you call the **Save** method.

In the example, note the use of a **Public** flag variable, *gblnSaving*, that indicates whether the system is in the midst of a Save operation. Other code, such as that in the **Message** event procedure, can check the flag to modify its behavior.

The **Save** method, as well as all other methods and property set operations throughout the Microsoft Dynamics SL Object Model, is *synchronous* with respect to the automation client. That is, control does not return to the automation client until the completion of all Microsoft Dynamics SL processing caused by the call to this method, whether the processing comes from the core Microsoft Dynamics SL kernel, the original application, or any customization of the application.

Possible Exceptions

7562 — Save may be disabled, either by the application or because of access rights

Example (Visual Basic 6.0 Client)

```
'following code assumes that sivMyApp
'is an instance of SIVApplication.
Public gblnSaving As Boolean

Private Sub SaveMe()
  On Error GoTo SAVE_ERROR
  gblnSaving = True
  sivMyApp.Save

SAVE_EXIT:
  gblnSaving = False
  Exit Sub
```

```
SAVE_ERROR:
  MsgBox "Unable to save. Error #" & Err.Number & _
  ": " & Err.Description

  Resume SAVE_EXIT
End Sub

'....And in the Message Event procedure

Private Sub sivMyApp_Message(ByVal MessageNumber As Integer, _
ByVal MessageText As String, _
ByVal MessageType As sivMessageType, _
ByRef MessageResponse As sivMessageResponse)

Select Case MessageType
    Case sivMsgOK
        sivMessageResponse = sivMsgRspOK
    Case sivMsgAbortRetryIgnore
        Select Case MessageNumber
            Case x
                If gblnSaving Then
                    '...code to try to fix problem
                    MessageResponse = sivMsgRspRetry
                Else
                    MessageResponse = sivMsgRspIgnore
                End If
            Case y
                    MessageResponse = sivMsgRspIgnore
            Case Else
                    MessageResponse = sivMsgRspAbort
        End Select

    Case sivMsgYesNo

        '....etc.

    Case sivMsgYesNoCancel

        '....etc.

    Case sivMsgRetryCancel

        '....etc.

End Select

End Sub
```

Example (Visual Basic 2005 Client)

```
Try
    gblnSaving = True
    sivMyApp.Save()
    gblnSaving = False

Catch ex As System.Runtime.InteropServices.COMException

    Dim lErrNumber As Integer
    If ex.Source.Contains("Solomon") = True Or ex.ErrorCode - vbObjectError > 0
Then
        lErrNumber = ex.ErrorCode - vbObjectError
    Else
        lErrNumber = ex.ErrorCode
    End If

    Select Case lErrNumber
        Case Else
            MsgBox(ex.Message, MsgBoxStyle.Exclamation, String.Format("{0}
Error {1}", ex.Source, lErrNumber.ToString()))
    End Select
```

```
Catch ex As Exception
    MsgBox(ex.Message, MsgBoxStyle.Exclamation, String.Format("{0} Exception",
ex.Source))
End Try


'....And in the Message Event procedure
Private Sub sivMyApp_Message( _
    ByVal MessageNumber As Integer, _
    ByVal MessageText As String, _
    ByVal MessageType As sivMessageType, _
    ByRef MessageResponse As sivMessageResponse) _
Handles sivMyApp.Message

    Select Case MessageType
        Case sivMessageType.sivMsgOk
            MessageResponse = sivMessageResponse.sivMsgRspOk
        Case sivMessageType.sivMsgAbortRetryIgnore
            Select Case MessageNumber
               Case x
                    If gblnSaving Then
                        '...code to try to fix problem
                        MessageResponse = sivMessageResponse.sivMsgRspRetry
                    Else
                        MessageResponse = sivMessageResponse.sivMsgRspIgnore
                    End If
                Case y
                   MessageResponse = sivMessageResponse.sivMsgRspIgnore
                Case Else
                    MessageResponse = sivMessageResponse.sivMsgRspAbort
            End Select
        Case sivMessageType.sivMsgYesNo
            '....etc.
        Case sivMessageType.sivMsgYesNoCancel
            '....etc.
        Case sivMessageType.sivMsgRetryCancel
            '....etc.
    End Select

End Sub
```

SetBusinessDate Method

Sets the current Microsoft Dynamics SL business date.

Applies To

SIVApplication

Syntax

object.SetBusinessDate *Month, Day, Year*

The SetBusinessDate method syntax has these parts:

Part	Description
object	An instance of SIVApplication.
Month	ByRef integer. Represents the calendar month of the current business date. Values can be 1 - 12.
Day	ByRef integer. Represents the day of the month of the current business date. Values can be between 1 and the last day of the month specified in the Month argument.
Year	ByRef integer. Represents the four-digit year of the current business date. Values can be any four-digit number.

Remarks

Allows you to set the current Microsoft Dynamics SL business date. Before calling this method in your code, you should prepare three integer variables to hold the month, day, and year, assign the desired values to the variable, and pass them variables as the respective arguments to the method (see examples).

The SetBusinessDate method, as well as all other methods and property set operations throughout the Microsoft Dynamics SL Object Model, is synchronous with respect to the automation client. That is, control does not return to the automation client until the completion of all Microsoft Dynamics SL processing caused by the call to this method, whether the processing comes from the core Microsoft Dynamics SL kernel, the original application, or any customization of the application.

Possible Exceptions

- 7506 — Not logged in
- 7514 — Business Date Error: Invalid Date

See Also

GetBusinessDate Method

Example (Visual Basic 6.0 Client)

```
'Assumes sivTB is an already-instantiated
'SIVToolbar object
    Dim iMonth As Integer, iDay As Integer, iYear As Integer
    iMonth = 12
    iDay = 31
    iYear = 2001
    On Error GoTo SET_DATE_ERROR
    sivTB.SetBusinessDate iMonth, iDay, iYear
    '...do something with the date information here
    Exit Sub
```

```
SET_DATE_ERROR:
    Select Case Err.Number - vbObjectError
        Case 7506
            MsgBox "Not logged in"
            Resume Next
        Case 7514
            MsgBox "Invalid date (M/D/Y): " _
                   & iMonth & "/" _
                   & iDay & "/"
                   & iYear
            Resume Next
        Case Else
            MsgBox Err.Number & ": " & Err.Description, _
            "UNEXPECTED ERROR"
    End Select
```

Example (Visual Basic 2005 Client)

```
Dim iMonth As Integer, iDay As Integer, iYear As Integer
Try

    'Assumes sivTB is an already-instantiated
    'SIVToolbar object
    iMonth = 12
    iDay = 31
    iYear = 2001
    sivTB.SetBusinessDate(iMonth, iDay, iYear)
    '...do something with the date information here

Catch ex As System.Runtime.InteropServices.COMException

    Dim lErrNumber As Integer
    If ex.Source.Contains("Solomon") = True Or ex.ErrorCode - vbObjectError > 0
Then
        lErrNumber = ex.ErrorCode - vbObjectError
    Else
        lErrNumber = ex.ErrorCode
    End If

    Select Case lErrNumber
        Case 7506
            MsgBox("Not logged in")

        Case 7514
            MsgBox(String.Format("Invalid date: {0}/{1}/{2}", iMonth.ToString,
iDay.ToString, iYear.ToString))

        Case Else
            MsgBox(String.Format("{0}: {1}", lErrNumber.ToString(),
ex.Message))
    End Select

Catch ex As Exception
    MsgBox("Exception: " + ex.Message, MsgBoxStyle.Exclamation, ex.Source)
End Try
```

SetCurrencyIDs Method

Sets the transaction currency ID for the current screen.

Applies To

SIVApplication

Syntax

object.**SetCurrencyIDs** *TransactionCurrencyID As String, IntermediateCurrencyID As String*

The **SetCurrencyIDs** method syntax has these parts:

Part	Description
object	An instance of **SIVApplication**.
TransactionCurrencyID	String representing a valid currency ID code for the logged-in company and database.
IntermediateCurrencyID	String. Reserved for future use. Has no effect at this time.

Remarks

Allows you to set the current screen's transaction currency ID.

The value that you pass for *TransactionCurrencyID* should correspond to a valid currency ID for the current logged-in company and database. If you supply an invalid currency ID, then the Object Model raises runtime error 7545.

The underlying application or a customization may disable the ability to change the transaction currency. When that is the case, then calls to this method raise runtime error 7543.

At this point, *IntermediateCurrencyID* should always be a blank string. It is reserved for future use and has no effect in the current version of the Microsoft Dynamics SL Object Model.

The **SetCurrencyIDs** method, as well as all other methods and property set operations throughout the Microsoft Dynamics SL Object Model, is synchronous with respect to the automation client. That is, control does not return to the automation client until the completion of all Microsoft Dynamics SL processing caused by the call to this method, whether the processing comes from the core Microsoft Dynamics SL kernel, the original application, or any customization of the application.

Possible Exceptions

- 7543 — Transaction Currency ID cannot be set at this time
- 7545 — Transaction Currency ID is invalid
- 7547 — Intermediate Currency ID is invalid

See Also

GetCurrencyIDs Method

Example

```
'Assumes sivApp is an already-instantiated
'SIVApp object
    On Error GoTo SET_CURYIDS_ERROR
    sivApp.SetCurrencyIDs "USD", ""
    Exit Sub
SET_CURYIDS_ERROR:
    Select Case Err.Number - vbObjectError
        Case 7543
```

```
        MsgBox "Transaction IDs can't be set now"
        Resume Next
    Case 7545
        MsgBox "Invalid Transaction Currency ID"
        Resume Next
    Case 7547
        'Shouldn't happen in current version
        'of Object Model
        MsgBox "Invalid Intermediate Currency ID"
    Case Else
        MsgBox Err.Number & ": " & Err.Description, _
        "UNEXPECTED ERROR"
End Select
```

SetCurrencyView Method

Toggles the display currency for currency amounts on the current screen.

Applies To

SIVApplication

Syntax

object.SetCurrencyView *CurrencyView* As sivCurrencyView

The SetCurrencyView method syntax has these parts:

Part	Description
object	An instance of SIVApplication.
CurrencyView	Integer of enumerated type sivCurrencyView representing the type of currency to display amounts in (either system Base currency, or screen transaction currency).

Remarks

Allows you to toggle the current screen's display currency between the system base currency and the screen transaction currency.

The value that you pass for *CurrencyViewID* should be one of the enumerated constants in the sivCurrencyView type indicating either the system base currency or the screen's transaction currency.

If the underlying application has blocked the toggling of currency view, then the Object Model raises runtime error 7549.

The SetCurrencyView method, as well as all other methods and property set operations throughout the Microsoft Dynamics SL Object Model, is synchronous with respect to the automation client. That is, control does not return to the automation client until the completion of all Microsoft Dynamics SL processing caused by the call to this method, whether the processing comes from the core Microsoft Dynamics SL kernel, the original application, or any customization of the application.

Possible Exceptions

- 7549 — Currency view cannot be toggled
- 7568 — Invalid value for sivCurrencyView enumeration

See Also

GetCurrencyView Method

Example (Visual Basic 6.0 Client)

```
'Assumes sivApp is an already-instantiated
'SIVApp object
    On Error GoTo SET_CURYVIEW_ERROR
    sivApp.SetCurrencyView sivCurrencyViewBase
    Exit Sub
SET_CURYVIEW_ERROR:
    Select Case Err.Number - vbObjectError
        Case 7549
            MsgBox "Can't toggle currency view"
            Resume Next
        Case Else
            MsgBox Err.Number & ": " & Err.Description, _
```

```
            "UNEXPECTED ERROR"
    End Select
```

Example (Visual Basic 2005 Client)

```
Try
    'Assumes sivApp is an already-instantiated
    'SIVApp object
    sivApp.SetCurrencyView(sivCurrencyView.sivCurrencyViewBase)

Catch ex As System.Runtime.InteropServices.COMException

    Dim lErrNumber As Integer
    If ex.Source.Contains("Solomon") = True Or ex.ErrorCode - vbObjectError > 0
Then
        lErrNumber = ex.ErrorCode - vbObjectError
    Else
        lErrNumber = ex.ErrorCode
    End If

    Select Case lErrNumber
        Case 7549
            MsgBox("Can't toggle currency view")
        Case Else
            MsgBox(String.Format("{0}: {1}", lErrNumber.ToString(),
ex.Message))
        End Select

Catch ex As Exception
    MsgBox("Exception: " + ex.Message, MsgBoxStyle.Exclamation, ex.Source)
End Try
```

SetCustomizationLevel Method

Sets the Microsoft Dynamics SL customization level for applications during the current session.

Applies To

SIVApplication

Syntax

object.**SetCustomizationLevel** *CustomizationLevel, UserID, ExcludeMacroCode*

The **SetCustomizationLevel** method syntax has these parts:

Part	Description
object	An instance of **SIVToolbar**.
CustomizationLevel	Integer of type **sivCustomizationLevel**. Represents the current customization level of the toolbar session.
UserID	String. The user ID to apply when setting the customization level to **sivCstLvlOneUser**.
ExcludeMacroCode	Boolean. Setting to True excludes macro code, while setting to False includes macro code.

Remarks

Sets the current Microsoft Dynamics SL customization level and other global customization information. Provides the functionality of s*elect customization level* in the user interface.

The *UserID* argument only has significance when *CustomizationLevel* is *sivCstLvlOneUser*. If *UserID* is not a valid user id for the current system, then the Object Model returns error 7512.

When you set *ExcludeMacroCode* to True, then customizations will not run any of their underlying script code. Only “cosmetic” customizations will be applied. This setting corresponds to the check box labeled **Exclude Event Code** that the interactive user sees on Microsoft Dynamics SL’s *Customization Level* dialog.

In order to call this method successfully, the **sivToolbar** object must already be logged on to the database, the current user ID must have sufficient rights to set the customization level, and no **SIVApplication** objects can be running.

The **SetCustomizationLevel** method, as well as all other methods and property set operations throughout the Microsoft Dynamics SL Object Model, is synchronous with respect to the automation client. That is, control does not return to the automation client until the completion of all Microsoft Dynamics SL processing caused by the call to this method, whether the processing comes from the core Microsoft Dynamics SL kernel, the original application, or any customization of the application.

Possible Exceptions

- 7506 — Not logged in
- 7509 — Customization Level Error: Applications Running
- 7510 — Customization Level Error: Enumeration Selection Error
- 7511 — Customization Level Error: No Access Rights
- 7512 — Customization Level Error: Specified Userid is invalid

See Also

GetCustomizationLevel Method

Example (Visual Basic 6.0 Client)

```
'Assumes sivTB is an already-instantiated
'SIVToolbar object

    On Error GoTo SET_CUSTLEVEL_ERROR
    sivTB.SetCustomizationLevel sivCstLvlOneUser, "JONES", False
    Exit Sub

SET_CUSTLEVEL_ERROR:
    Select Case Err.Number - vbObjectError
       Case 7506
            MsgBox "Not logged in"
            Resume Next
       Case 7509
            MsgBox _
            "Can't set Customization Level with Applications running"
            Resume Next
       Case 7510
            MsgBox "Invalid Customization Level specified"
            Resume Next
       Case 7511
            MsgBox "No Access rights to set customization level"
            Resume Next
       Case 7512
            MsgBox "Invalid UserID for SingleUser customization"
            Resume Next
        Case Else
            MsgBox Err.Number & ": " & Err.Description, _
            "UNEXPECTED ERROR"
    End Select
```

Example (Visual Basic 2005 Client)

```
Try

    'Assumes sivTB is an already-instantiated
    'SIVToolbar object
    sivTB.SetCustomizationLevel(sivCustomizationLevel.sivCstLvlOneUser,
"JONES", False)

Catch ex As System.Runtime.InteropServices.COMException

    Dim lErrNumber As Integer
    If ex.Source.Contains("Solomon") = True Or ex.ErrorCode - vbObjectError > 0
Then
        lErrNumber = ex.ErrorCode - vbObjectError
    Else
        lErrNumber = ex.ErrorCode
    End If

    Select Case lErrNumber
        Case 7506
            MsgBox("Not logged in")
        Case 7509
            MsgBox("Can't set Customization Level with Applications running")
        Case 7510
            MsgBox("Invalid Customization Level specified")
        Case 7511
            MsgBox("No Access rights to set customization level")
        Case 7512
            MsgBox("Invalid UserID for SingleUser customization")
        Case Else
```

```
            MsgBox(ex.Message, MsgBoxStyle.Exclamation, String.Format("{0}
Error {1}", ex.Source, lErrNumber.ToString()))
    End Select

Catch ex As Exception
    MsgBox(ex.Message, MsgBoxStyle.Exclamation, String.Format("{0} Exception",
ex.Source))
End Try
```

SetStatusBarText Method

Sets the text and the ToolTip text for the text pane of the status bar.

Applies To

SIVApplication

Syntax

object.**SetStatusBarText** *Text, ToolTipText*

The **SetStatusBarText** method syntax has these parts:

Part	Description
object	An instance of **SIVToolbar**.
Text	**String** representing text to display in the text pane of the application screen's status bar.
ToolTipText	**String** representing the ToolTip text that will pop up when the user pauses the mouse over the text pane of the application screen's status bar. You may leave this string blank to default the ToolTipText to be the same as the Text.

Remarks

You can change the text on the application screen's status bar text panel as well as the ToolTip text for that panel with the **SetStatusBarText** method.

The status bar is the area at the bottom of a Microsoft Dynamics SL application screen that contains information in various panes. Panes include information about date and time as well as a text pane with variable information.

The **SetStatusBarText** method affects only the text pane with its *Text* argument and the ToolTip text for the same pane with its *ToolTipText* argument. ToolTip text represents the contents of the popup that appears when the user pauses the mouse over the text pane.

If you pass a blank string as *ToolTipText*, then the text pane's ToolTip contents will default to be the same as the text pane itself.

Calls to the **SetStatusBarText** method, as well as all property set operations and other method calls throughout the Microsoft Dynamics SL Object Model, are synchronous with respect to the automation client. That is, control does not return to the automation client until the completion of all Microsoft Dynamics SL processing caused by this method call, whether the processing comes from the core Microsoft Dynamics SL kernel, the original application, or any customization of the application.

Possible Exceptions

None

Example

```
'Assumes sivApp is an already-instantiated
'SIVApplication object
    sivApp.SetStatusBarText "Waiting", _
                            "Be Patient"
```

SetTranCurrencyAndRate Method

Sets the transaction currency ID for the current screen based on the rate type and effective date.

Applies To

SIVApplication

Syntax

object.**SetTranCurrencyAndRate** *TransactionCurrencyID As String*, *RateType As String*, *EffectiveDate As String*

The **SetTranCurrencyAndRate** method syntax has these parts:

Part	Description
object	An instance of **SIVApplication**.
TransactionCurrencyID	**String** representing the valid currency ID code for the company and database currently in use.
RateType	**String** representing the rate type to use to set currency rate.
EffectiveDate	**String** representing the effective date to use to set currency rate. This value should be passed in the Date format used by the Microsoft Dynamics SDK.

Remarks

The value that you pass for *TransactionCurrencyID* should correspond to a valid currency ID for the company and database the user is currently accessing. If you supply a currency ID that is not valid, runtime error 7545 occurs.

An underlying application or a customization may affect the ability to change the transaction currency. If this occurs, calls to this method result in runtime error 7543.

The **SetTranCurrencyAndRate** method, as well as all other methods and set property operations throughout the Microsoft Dynamics SL Object Model, are synchronous with respect to the automation client. That is, control does not return to the automation client until completion of all Microsoft Dynamics SL processing resulting from the call to this method. This is regardless of whether the processing comes from the Microsoft Dynamics SL kernel, the original application, or any customization of the application.

Possible Exceptions

- Runtime error 7543 — Transaction Currency ID cannot be set at this time.
- Runtime error 7545 — Transaction Currency ID is invalid.
- Runtime error 7546 — Rate Type is invalid.

See Also

GetCurrencyIDs Method, **SetCurrencyIDs Method**

Example

```
'Assumes sivApp is an SIVApp object that is
'already instantiated
    On Error GoTo SET_CURYIDS_ERROR
    sivApp.SetTranCurrencyAndRate "CAD", "A", "09/30/2009"
    Exit Sub
SET_CURYIDS_ERROR:
```

```
Select Case Err.Number - vbObjectError
    Case 7543
        MsgBox "Transaction IDs can't be set now"
        Resume Next
    Case 7545
        MsgBox "Invalid Transaction Currency ID"
        Resume Next
    Case 7546
        MsgBox "Invalid Rate Type ID"
    Case Else
        MsgBox Err.Number & ": " & Err.Description, _
        "UNEXPECTED ERROR"
End Select
```

SIVApplication Object

Corresponds to an instance of a Microsoft Dynamics SL application screen.

Remarks

The **SIVApplication** object represents a running instance of a Microsoft SL SDK application. You can use such an object to access the functionality of the application that is available to an interactive user, including reading and altering data through the on-screen controls and navigating through various record sets on the screen. See the references to this object's various properties, methods, and events listed below for more details.

You initialize an **SIVApplication** object by setting an **SIVApplication** object variable to point to the return value of the **SIVToolbar's StartApplication** method (see the example below).

An automation client may not be able to start an **SIVApplication** for various reasons. See documentation for the **StartApplication** method of the **SIVToolbar** object for more details.

By default the **SIVApplication** object is invisible (**Visible** property = False) when it is initialized by the **StartApplication** method of the **SIVToolbar** object. When the **SIVApplication** object is invisible, it does not show up on the Windows Taskbar.

After calling the **Quit** method for the **SIVApplication** object, the object will be destroyed by calling its **Dispose** method. If the **SIVApplication** object is destroyed before calling the **Quit** method, then the application will automatically become visible to the user.

Any exceptions raised during the shutdown of the application, such as prompts to save changes or warnings about incorrect data settings, prevent the application from shutting down.

If an automation client holds a valid reference to a visible instance of **SIVApplication**, then the interactive user will be unable to close it.

Properties

Controls property, **Currency** property, **EventLog** property, **Value** property, **Visible** property.

Methods

GetCurrencyView method, **GetCurrencyIDs** method, **GetCustomObject** method, **GetEntityStatus** method, **GetStatusBarText** method, **GridView** method, **Cancel** method, **Delete** method, **Dispose** method, **First** method, **Last** method, **New** method, **Next** method, **Previous** method, **Quit** method, **Save** method, **SetStatusBarText** method, **SetCurrencyIds** method, **SetCurrencyView** method.

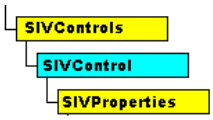
Events

Message event, **SubFormDisplay** event.

Example

```
'Assume that sivTB is an instance
'of SIVToolbar and that MyApp.exe is a
'Microsoft SL SDK application in the Microsoft Dynamics SL
'executable directory

Dim sivMyApp As SIVApplication
Set sivMyApp = sivTB.StartApplication("MyApp.exe")
```

SIVControl Object



Corresponds to a control on the Microsoft Dynamics SL application screen represented by a **SIVApplication** object.

Syntax

SIVControl

Remarks

An **SIVControl** object is only accessible as a member the Controls collection of an **SIVApplication** object. Once you have located in the collection an **SIVControl** object that you want to work with, you can set an object variable of type **SIVControl** to point to that collection member. Then, you can program with the object variable. You can also use the **With...End With** construct to program with a member of the Controls collection.

Index can be an integer value referring to the object's position within the SIVApplication's Controls collection. The use of a numeric index is generally most practical when you need to get and save a reference to a control in your code whose name you did not know beforehand.

When you know the name of the control beforehand, it is usually more practical to use a string index that gives the name of the control.

Unlike a standard Visual Basic application's control names (which are only unique to their containing form), control names in a Microsoft SL SDK application are unique in the entire Microsoft SL SDK application, even if the Microsoft SL SDK application has multiple forms. Thus, it is never necessary to know the name of the individual form where a control resides in order to locate it with its name. This is because the control name is unique to the entire Microsoft SL SDK application, so there is no danger of confusing the control with a control of the same name on a different form in the same Microsoft SL SDK application.

When you need to refer to an **SIVControl** that points to a member of a control array in the underlying application, then you must use the control array index as part of the control name that you pass as an argument to the Controls collection. For example, you would refer to element 2 of a control array of **SAFOption** controls as follows:

```
sivMyApp.Controls("optCreditCard(2)")
```

SAFOption is the most common type of control found in control arrays.

You can refer to the individual standard Visual Basic property settings of a control and to special Microsoft Dynamics SL properties through the control's Properties collection. You can read or write to a control's default property (also known as the control's *value*) by either simply referring to the control object itself in code (the default property is implicitly understood) or by referring the control's **Value** property.

Note: Not all ActiveX controls and no .NET WinForm controls support a default property, so this technique must be used with caution.

The Microsoft Dynamics SL Object Model does not expose any of a control's methods or events.

Note that controls originally programmed in a Microsoft Dynamics SL application and controls added via Customization Manager have no difference in the Microsoft Dynamics SL Object Model.

Properties

Name property, **Properties** property, **Value** property

See Also

SIVControls Collection

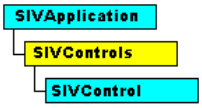
Example (Visual Basic 6.0 Client)

```
'Technique that uses a control object variable
'to point to a control in a screen's Controls collection
Dim ctrlCurr As SIVControl
Set ctrlCurr = sivMyApp.Controls("cTotalBalance")
'set the Value property by default:
ctrlCurr = -300
'directly set Value property of a control
sivMyApp.Controls("cCustID") = "C300"
'set a property that is not the default
sivMyApp.Controls("cCustID").BackColor = vbRed
```

Example (Visual Basic 2005 Client)

```
'Technique that uses a control object variable
'to point to a control in a screen's Controls collection
Dim ctrlCurr As SIVControl
ctrlCurr = sivMyApp.Controls("cTotalBalance")

'set the Value property by default:
ctrlCurr.Value = -300

'directly set Value property of a control
sivMyApp.Controls("cCustID").Value = "C300"

'set a property that is not the default
sivMyApp.Controls("cCustID").Properties("BackColor").Value =
VBRUN.ColorConstants.vbRed
```

SIVControls Collection



A collection whose elements represent the controls available on a Microsoft SL SDK application screen. Each element of the collection is an object of type **SIVControl**.

Applies To

SIVApplication object

Syntax

object.**Controls.Count**
object.**Controls.Item**
object.**Controls**(*index*)

The **SIVControls** collection syntax has these parts:

Part	Description
object	An instance of **SIVApplication**.
index	Either an integer between 1 and *object*.**Controls.Count** or a unique string key that corresponds to the name of the control that you are trying to access.

Methods

Count method, **Item** method

Remarks

You can traverse the collection of **SIVControl** objects by using a **For...Each** loop through the Controls collection of the **SIVApplication** object (see the example below).

Index can be an integer value referring to the Controls collection. The use of a numeric index is generally most practical when you need to get and save a reference to a control in your code whose name you did not know beforehand.

When you know the name of the control beforehand, it is usually more practical to use a string index that gives the name of the control.

The following control types are exposed in the Controls collection:

- SAFCheck
- SAFCombo
- DSLDate
- SAFFloat
- SAFInteger
- SAFMaskedText
- SAFOption
- Standard Visual Basic and third-party controls that are available to the user interface of the Microsoft SL SDK application represented by the current instance of **SIVApplication**. Note that problems caused by ill-behaved controls (that is, non-Microsoft Dynamics SL controls with features that, in some way, could “break” the proper functioning of Microsoft Dynamics SL or of the Microsoft Dynamics SL Object Model) are the responsibility of the automation programmer or

of the Microsoft SL SDK programmer who created the application. See “Working with Troublesome Controls” for further discussion of this topic.

The following control types are specifically *not* exposed in the Controls collection:

- Notes/Attachments Icon
- SAFContainer
- SAFDesigner
- SAFGrid
- SAFNewFrame
- SAFNewTab
- SAFUpdate
- StatusBar (Microsoft SL SDK)—see the **SetStatusBarText** method

Note that there is no difference for the Microsoft Dynamics SL Object Model between controls originally programmed in a Microsoft Dynamics SL application and controls added later with Customization Manager.

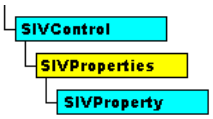
See Also

SIVControl Object

Example

```
'Using For...Each to Traverse the Controls collection

Dim ctrlCurr As SIVControl

For Each ctrlCurr in sivMyApp.Controls

    MsgBox ctrlCurr.Name

Next ctrlCurr
```

SIVProperties Collection



A collection whose elements represent the properties available for a control on a Microsoft SL SDK application screen. Each element of the collection is an object of type **SIVProperty**.

Applies To

SIVControl object

Syntax

object.**Properties.Count**
object.**Properties.Item**
object.**Properties**(*index*)

The **SIVProperties** collection syntax has these parts:

Part	Description
object	An instance of **SIVControl**.
index	Either an integer between 1 and *object*.**Properties.Count** or a unique string key that corresponds to the name of the control property that you are trying to access.

Methods

Count method, **Item** method

Remarks

You can traverse the collection of **SIVProperty** objects by using a **For...Each** loop through the Properties collection of an **SIVControl** object (see the example below).

Index can be an integer value referring to the Properties collection. The use of a numeric index is generally most practical when you need to get and save a reference to a property in your code whose name you did not know beforehand.

When you know the name of the property beforehand, it is usually more practical to use a string index that gives the name of the property.

A number of control properties are not exposed in the **SIVProperties** collection:

- Container property
- Hidden properties
- Non-browse able properties
- Object property
- Parent property
- Properties that take parameters
- Visible property (not exposed for forms, exposed for other controls)

See Also

Appendix G: Visual Basic .NET-Related Changes, **Count Property**, **Properties Collection (SIVControl Object)**, **SIVProperty Object**

Example

```
'Using For...Each to Traverse the Properties collection
'Assume that sivctrlCurr is an existing instance
'of an SIVControl object

Dim prCurr As SIVProperty

For Each prCurr in sivctrlCurr.Properties.Item
    'Show property name in MsgBox Caption and
    'its current setting in the MsgBox itself
    MsgBox prCurr.Value , prCurr.Name
Next prCurr
```

SIVProperty Object

Corresponds to a property of a control on a Microsoft Dynamics SL application screen.

Applies To

Properties object

Remarks

Apart from a control's value (see the following paragraph), all control properties in a Microsoft Dynamics SL automation client objects are only accessible as an **SIVProperty** member of the Properties collection of the **SIVControl** object. Once you have located in the Properties collection a **SIVProperty** object that you want to work with, you can set an object variable of type **SIVProperty** to point to that collection member. Then, you can program with the object variable (see the example below). You can also use the **With...End With** construct to program with a member of the Properties collection.

For each Microsoft Dynamics SL control type (SAF control) and for most non-Microsoft Dynamics SL Visual Basic controls, there is a default property that you do not need to refer to by name. This property is known in Visual Basic terminology as the control's value and it corresponds to the **SIVControl** object's **Value** property. You can refer directly to the default property without the use of the control's Properties collection by simply referring to the control directly, as in the following example:

```
sivMyApp.Controls("cTotal").Value = 1000
```

See this documentation's discussion of each of the SAF controls for a description of each control's default property. You can find information about the default properties of standard Visual Basic and third-party controls in their respective documentation.

Note: Not all ActiveX controls and no .NET WinForm controls support a default property, so this technique must be used with caution.

Note that your automation client's attempt to set a property value is subject to any restrictions on the control or on the specific property's value imposed by either the original control itself, the original Microsoft Dynamics SL application, or existing customizations that are currently in force (based on the current value of the **SIVToolbar** object's **CustomizationLevel** property). The Microsoft Dynamics SL Object Model raises an error to the automation client for all violations of data type, masking, and data-length rules caused by the client's attempt to set a property.

See this documentation's discussion of each of the SAF controls for a description of data validation issues for particular controls. You can also find information about property restrictions in the Microsoft SL SDK documentation for Microsoft Dynamics SL controls, or in the respective documentation for standard Visual Basic and third-party controls.

Index can be an integer value referring to the object's position within the SIVControl's Properties collection. The use of a numeric index is generally most practical when you need to get and save a reference to a property in your code whose name you did not know beforehand.

When you know the name of the property beforehand, it is usually more practical to use a string index that gives the name of the property.

A property setting is only specific to the current instance of the **SIVApplication** object where you set it. Changes that you make to a control's property do not persist (carry over) to other instances of the same application, even if those instances are running at the same time as the instance where you change the property.

Listed below are Microsoft Dynamics SL-specific properties that you can use in automation client code. Not all Microsoft Dynamics SL controls support all the properties listed. Note that when a Microsoft

Dynamics SL property name for an SAF control is the same as a standard Visual Basic property name, then the Microsoft Dynamics SL property name takes precedence.

Note: Microsoft supports only the properties of Microsoft Dynamics SL Object Model given in the following two lists. While it is possible to access other properties than those listed here, Microsoft reserves the right to change or withdraw its support of those unlisted properties.

Commonly Used Microsoft Dynamics SL-specific properties exposed as SIVProperty objects

Property Name	Description
BlankErr	**Boolean** (default is False). If True, then an entry is required in this control's contents, and the entity to which this control belongs cannot be saved if the control is blank (an error is raised). Suggested use: check the **BlankErr** property before attempting to save or navigate data. If it is True, then check the control's contents.
DBNav	Used to facilitate navigation through all database records in the result set of a SQL statement.
FalseText	Determines the value of the underlying data field whenever the control is not checked.
FieldClass	Associates a control with a particular class of data items having global display and/or operational characteristics.
FieldName	Facilitates proper runtime binding between the control and its underlying Visual Basic data variable by operating in conjunction with the SetAddr statement.
Level	Associates the control with a logical group of information contained within the application.
List	Determines the fixed list of valid data values for the underlying field along with corresponding descriptions.
Mask	Determines the type and number of characters that can be entered for a particular field.
Max	Determines the maximum valid value for the control.
Min	Determines the minimum valid value for the control.
PV	Determines all possible values currently existing in the database.
TrueText	Determines the value of the underlying data field whenever the check box or option button is selected.
Value	Represents the value of the data in the control.

The list below contains the most commonly-used Visual Basic standard properties. Only properties that Microsoft supports are listed. Many other properties are available, but Microsoft does not support them.

Commonly Used Standard Visual Basic properties exposed as SIVProperty objects

Property Name	Description
Align	**Integer** (enumerated constant, default is **vbLeft**).
BackColor	Background color of an object.
Caption	Text displayed next to the control.
Font	Identifies a specific font for an object.
ForeColor	Foreground color used to display data in an object.
Name	Name of the control. Display only.
Height	Height of the control, usually in twips (but unit of measure depends on ScaleMode setting of the container object, which is not available in the Microsoft Dynamics SL Object Model).

Property Name	Description
Left	Distance of leftmost edge of the control from the inner left boundary of the container object, usually in twips (but unit of measure depends on ScaleMode setting of the container object, which is not available in the Microsoft Dynamics SL Object Model).
Top	Distance of topmost edge of the control from the inner top boundary of the container object, usually in twips (but unit of measure depends on ScaleMode setting of the container object, which is not available in the Microsoft Dynamics SL Object Model).
Width	Width of the control, usually in twips (but unit of measure depends on ScaleMode setting of the container object, which is not available in the Microsoft Dynamics SL Object Model).
Value	Where this property exists, it is the default property of the control. Not all controls have a **Value** property, even some controls that do support a default property.
Visible	**Boolean** value determining whether or not the control is visible to the interactive user. Of course, no controls are visible if the parent **SIVApplication** object is invisible.

The following standard Visual Basic properties are not available as **SIVProperty** objects:

- Container
- Object
- Parent

Attempts to set any property's value, as well as any method calls to the Microsoft Dynamics SL Object Model, are synchronous with respect to the automation client. That is, control does not return to the automation client until the completion of all Microsoft Dynamics SL processing caused by the attempt to set the property, whether the processing comes from the core Microsoft Dynamics SL kernel, the original application, or any customization of the application.

See Also

Appendix G: Visual Basic .NET-Related Changes, **Properties Collection (SIVControl Object)**, **SIVProperty Object**, **Value Property (SIVProperty Object)**

Example (Visual Basic 6.0 Client)

```
'set a control object variable
'to point to a control in a screen's Controls collection
Dim ctrlCurr As SIVControl
Set ctrlCurr = sivMyApp.Controls("cTotalBalance")

'set the Value property by default:
ctrlCurr = -300

'directly set Value property of a control
sivMyApp.Controls("cCustID") = "C300"

'set a property that is not the default
sivMyApp.Controls("cCustID").Properties.Item("BackColor") = vbRed
```

Example (Visual Basic 2005 Client)

```
'Technique that uses a control object variable

'to point to a control in a screen's Controls collection

Dim ctrlCurr As SIVControl

ctrlCurr = sivMyApp.Controls("cTotalBalance")


'set the Value property by default:
```

```
ctrlCurr.Value = -300

'directly set Value property of a control
sivMyApp.Controls("cCustID").Value = "C300"

'set a property that is not the default
   sivMyApp.Controls("cCustID").Properties("BackColor").Value =
   VBRUN.ColorConstants.vbRed
```

SIVToolbar Object

Corresponds to an instance of the toolbar.

Remarks

The **SIVToolbar** object represents a running instance of the toolbar. Just as the interactive user cannot run any Microsoft Dynamics SL application screens without first running the toolbar, so it is with the Microsoft Dynamics SL automation client. The Microsoft Dynamics SL automation client cannot run any Microsoft Dynamics SL applications without first obtaining a logged-on instance of the **SIVToolbar** object.

You will use the **SIVToolbar** object at the following points in a Microsoft Dynamics SL automation session:

- *Begin* an automation session by instantiating the **SIVToolbar** object.
- *Log on* to a database context with the **Login** method of **SIVToolbar**.
- *Run* one or more Microsoft Dynamics SL applications by instantiating **SIVApplication** objects through the **StartApplication** method.
- *End* an automation session by calling the **Quit** method and destroying the **SIVToolbar** instance by callings its **Dispose** method.

There can be only one instance of **SIVToolbar** running on any given workstation. This includes both automation clients and interactive copies of the toolbar.

For considerations about using the **SIVToolbar** object concurrently with an interactive user session on the same workstation, see the documentation on the **Login** method of the **SIVToolbar** object.

By default the **SIVToolbar** object is invisible (**Visible** property = False) when it is initialized by your code. When the **SIVToolbar** object is invisible, it does not show up on the Windows Taskbar. If the interactive user is logged on to the workstation, then the **SIVToolbar** is visible and you cannot set its **Visible** property to False. See the documentation on the **Visible** property for more information.

If an automation client holds a valid reference to a visible instance of **SIVToolbar**, then the interactive user will be unable to close it.

If you destroy all automation client instances of the **SIVToolbar** object but do not call the **Quit** method, then the toolbar automatically becomes visible to the interactive user.

Properties

BusinessDate property**, CustomizationLevel** property, **InitializeMode** property, **Visible** property.

Methods

Dispose method**, Login** method, **Logout** method, **StartApplication** method, **Quit** method.

See Also

DSLDate Control (SIVControl Object), **InitializeMode Property**, **Login Method**, **Logout Method**, **Quit Method**, **StartApplication Method**, **Visible Property**

Example (Visual Basic 6.0 Client)

```
'Assume that MyApp.exe is a Microsoft SL SDK application
'in the Microsoft Dynamics SL executable folder

'STEP 1: Declare variables for Toolbar and Application
Dim sivtbCurrent As SIVToolbar
Dim sivMYApp As SIVApplication

'STEP 2: Initialize Toolbar and Log in
'(use pretend login information)

Set sivtbCurrent = New SIVToolbar
sivtbCurrent.Login "MyServer", "MySystemDB", _
"MyCompany", "SYSADMIN", "MyPass"

'STEP 3: Start application under new instance of Toolbar
Set sivMyApp = sivtbCurrent.StartApplication "MyApp.exe"

'STEP 4: do some stuff with the application
'...

'STEP 5: Quit app and Toolbar
'and free memory used by Toolbar
sivMyApp.Quit

sivMyApp.Dispose()

Set sivMyApp = Nothing

sivtbCurrent.Logout()
sivtbCurrent.Quit

sivtbCurrent.Dispose()

Set sivtbCurrent = Nothing
```

Example (Visual Basic 2005 Client)

```
'Assume that MyApp.exe is a Microsoft SL SDK application

'in the Microsoft Dynamics SL executable folder

'STEP 1: Declare variables for Toolbar and Application

Dim sivtbCurrent As SIVToolbar

Dim sivMYApp As SIVApplication

'STEP 2: Initialize Toolbar and Log in

'(use pretend login information)

sivtbCurrent = New SIVToolbar

sivtbCurrent.Login("MyServer", "MySystemDB", _

"MyCompany", "SYSADMIN", "MyPass")

'STEP 3: Start application under new instance of Toolbar

sivMYApp = sivtbCurrent.StartApplication("MyApp.exe")

'STEP 4: do some stuff with the application

'...

'STEP 5: Quit app and Toolbar

'and free memory used by Toolbar

sivMYApp.Quit()

sivMYApp.Dispose()

sivMYApp = Nothing

sivtbCurrent.Logout()

sivtbCurrent.Quit()

sivtbCurrent.Dispose()

sivtbCurrent = Nothing
```

StartAppAndAutomate Function (Microsoft SL SDK Applications)

Allows a Microsoft SL SDK application to become an Object Model client.

Syntax

Set *Retval* = **StartAppAndAutomate** *ExeName* as String, *SolomonErr* As Integer, *OSErr* As Long

The **StartAppAndAutomate** method syntax has these parts:

Part	Description
Retval	An instance of **SIVApplication**.
ExeName	**String**. The relative path and name of the executable to be automated.
SolomonErr	**ByRef** integer. Represents any Microsoft Dynamics SL error that the underlying application returned while loading.
OSErr	**ByRef** long. Represents any operating system error underlying Microsoft Dynamics SL error 7519.

Remarks

This statement allows the Microsoft SL SDK project to obtain a pointer to an **SIVApplication** object. The Microsoft SL SDK project can then automate the object. You can create an **SIVApplication** object, set control values to navigate to a particular recordset, and then destroy the reference to the **SIVApplication** object to make the application available to the interactive user.

The *ExeName* parameter must give the name of a valid Microsoft Dynamics SL executable, including the exe extension. If the executable is a standard Microsoft Dynamics SL screen, only the executable name is needed:

```
'standard Customer maintenance screen
Set SIVAppCust = _
    StartAppAndAutomate("0826000.exe", _
                        iSolErr, iOSErr)
```

If the executable is a non-standard Microsoft SL SDK application, the path to the executable is required, assuming the main Microsoft Dynamics SL directory as the root:

```
'custom app in file at \Microsoft Dynamics SL\tx\MyApp.exe
Set SIVAppCust = _
    StartAppAndAutomate("tx\MyApp.exe", _
                        iSolErr, iOSErr)
```

You need to pass an integer variable as the *SolErr* parameter. If there was an error starting the application, SolErr will contain the Microsoft Dynamics SL error number.

You need to pass a long variable as the *OSErr* parameter. This parameter usually contains 0 after the call to StartAppAndAutomate, unless the value of SolErr is returned as 7519 (problem starting application).

To use this method, the Microsoft SL SDK Project must set a reference to Microsoft.Dynamics.SL.ObjectModel.dll.

Possible Exceptions

None (All exceptions handled as return values in SolomonError and OSError parameters.)

Example

```
'Code within a Microsoft SL SDK application
'Note that a reference to Microsoft.Dynamics.SL.ObjectModel is required for
this example

Dim SIVApp As SIVApplication
Dim iSolErr As Integer
Dim iOSErr As Long
Set SIVApp = StartAppAndAutomate _
             ("MyApp.exe", iSolErr, iOSErr)

'If OS Error loading app
If iSolErr = 7519 Then
    'Handle the OS error
    Select Case iOSErr
        Case 5
        '....etc.
    End Select

'Or if other Microsoft Dynamics SL Error
ElseIf iSolErr <> 0 Then
    MsgBox "Error when loading app"

'Or if app loaded successfully
Else
    'put automation code here
End If
```

StartApplication Method

Starts a Microsoft SL SDK application and returns an instantiated **SIVApplication** object.

Applies To

SIVToolbar object

Syntax

Set objvariable = object.**StartApplication**(exename)

The **StartApplication** method syntax has these parts:

Part	Description
objvariable	A new instance of **SIVApplication** exposing the functionality in exename.
object	An instance of **SIVToolbar**.
exename	Required. String giving the name of the executable to run, including the EXE extension. See more detailed requirements in further discussion below.
cpnyid	Optional. Unique company ID.

Return Value

A pointer to a **SIVApplication** object that refers to the application.

Remarks

The **SIVToolbar** object must already be logged on to the appropriate company with a login context (user ID) that has sufficient rights to run the application. See Possible Exceptions below.

This instance of the application counts toward the total number of concurrent users allowed by the site license. It is therefore possible that attempting to run this instance of the application will cause the system to exceed the maximum number of users allowed. See Possible Exceptions below.

The **StartApplication** method may fail because the original developer of the underlying Microsoft Dynamics SL application has disabled the use of the Microsoft Dynamics SL Object Model for this application.

By default, the application is invisible when it starts.

If the interactive user has the same application already running, the Microsoft Dynamics SL Object Model will start another instance of the application. The Microsoft Dynamics SL Object Model is unable to create an object from an already-running instance of an application.

The *exename* parameter must give the name of a valid Microsoft Dynamics SL executable, including the exe extension. If the executable is a standard Microsoft Dynamics SL screen, only the executable name is needed:

```
'standard Customer maintenance screen
Set sivMyScreen = sivToolbar.StartApplication("0826000.exe")
```

If the program is being run against a specific company, the company id must be passed as the second parameter:

```
'standard Customer maintenance screen in company "0010"
Set sivMyScreen = sivToolbar.StartApplication("0826000.exe","0010")
```

If the executable is a non-standard Microsoft SL SDK application, then the path to the executable is required, assuming the main Microsoft Dynamics SL directory as the root:

```
'custom app in file at \Microsoft Dynamics SL\tx\MyApp.exe
Set sivMyScreen = sivToolbar.StartApplication("tx\MyApp.exe")
```

The **StartApplication** method, as well as all other methods and property set operations throughout the Microsoft Dynamics SL Object Model, is synchronous with respect to the automation client. That is, control does not return to the automation client until the completion of all Microsoft Dynamics SL processing caused by the call to this method, whether the processing comes from the core Microsoft Dynamics SL kernel, the original application, or any customization of the application.

A number of original Microsoft Dynamics SL executables do not support the Microsoft Dynamics SL Object Model. None of these are actual end user accounting applications, but are rather various types of utilities. See "Appendix C: Limitations" for more specific information.

Possible Exceptions

- 7506 — Not Logged In
- 7516 — Start Application Error: Login context may not have rights to run the application
- 7517 — Start Application Error: User Limit Exceeded
- 7519 — Start Application Error: Unable to start application due to operating system error: ***specific error number***
- 7520 — Start Application Error: Unable to start application due to fatal error during form load: ***specific error number***
- 7521 — Start Application Error: Initialize mode is on, but user does not have rights to run application in Initialize mode
- 7555 — Start Application Error: Application did not return an object handle
- 7563 — Object Model disabled in application

Example (Visual Basic 6.0 Client)

```
'following code assumes that sivtbCurrent
'is a SIVToolbar object that has already
'successfully logged in

On Error GoTo STARTAPP_ERR
dim sivMyApp As SIVApplication
'TWO POSSIBLE FORMS FOR THE STRING ARGUMENT:
'A) for a custom Microsoft SL SDK app in a directory named tx
'under the main Microsoft Dynamics SL directory:
Set sivMyApp = sivtbCurrent.StartApplication("tx\MyApp.exe")
'B) for a standard Microsoft Dynamics SL screen:
Set sivMyApp = sivtbCurrent.StartApplication("0826000.exe")
'C) for a standard Microsoft Dynamics SL screen specifically in
Company database '0010':
Set sivMyScreen = sivToolbar.StartApplication("0826000.exe", "0010")
Exit Sub
STARTAPP_ERR:
Select Case Err.Number - vbObjectError
    Case 7506
    Case 7516
    Case 7517
    Case 7519
    Case 7520
    Case 7521
    Case 7563
```

```
    Case Else
End Select
```

Example (Visual Basic 2005 Client)

```
Try
    'TWO POSSIBLE FORMS FOR THE STRING ARGUMENT:
    'A) for a custom Microsoft SL SDK app in a directory named tx
    'under the main Microsoft Dynamics SL directory:
    sivMyApp = sivtbCurrent.StartApplication("tx\MyApp.exe")
    'B) for a standard Microsoft Dynamics SL screen:
    sivMyApp = sivtbCurrent.StartApplication("0826000.exe")

Catch ex As System.Runtime.InteropServices.COMException

    Dim lErrNumber As Integer
    If ex.Source.Contains("Solomon") = True Then
        lErrNumber = ex.ErrorCode - vbObjectError
    ElseIf ex.ErrorCode - vbObjectError > 0 Then
        lErrNumber = ex.ErrorCode - vbObjectError
    Else
        lErrNumber = ex.ErrorCode
    End If

    Select Case lErrNumber
        Case 7506
        Case 7516
        Case 7517
        Case 7519
        Case 7520
        Case 7521
        Case 7563
        Case Else
            MsgBox(ex.Message, _
                   MsgBoxStyle.Exclamation, _
                   String.Format("{0} Error {1}", ex.Source,
lErrNumber.ToString()))
    End Select

Catch ex As Exception
    MsgBox("Exception: " + ex.Message, MsgBoxStyle.Exclamation, ex.Source)
End Try
```

StartApplication_2 Method

Starts a Microsoft SL SDK application and returns an instantiated **SIVApplication** object. This method may be used when developing with Visual Basic 6.

Applies To

SIVToolbar object

Syntax

Set objvariable = object.**StartApplication_2**(exename,cpnyid)

The **StartApplication_2 m**ethod syntax has these parts:

Part	Description
objvariable	A new instance of **SIVApplication** exposing the functionality in exename.
object	An instance of **SIVToolbar**.
exename	Required. String giving the name of the executable to run, including the EXE extension. See more detailed requirements in further discussion below.
cpnyid	Unique company ID.

Return Value

A pointer to a **SIVApplication** object that refers to the application.

Remarks

The **SIVToolbar** object must already be logged on to the appropriate company with a login context (user ID) that has sufficient rights to run the application. See Possible Exceptions below.

This instance of the application counts toward the total number of concurrent users allowed by the site license. It is therefore possible that attempting to run this instance of the application will cause the system to exceed the maximum number of users allowed. See Possible Exceptions below.

The **StartApplication_2** method may fail because the original developer of the underlying Microsoft Dynamics SL application has disabled the use of the Microsoft Dynamics SL Object Model for this application.

By default, the application is invisible when it starts.

If the interactive user has the same application already running, the Microsoft Dynamics SL Object Model will start another instance of the application. The Microsoft Dynamics SL Object Model is unable to create an object from an already-running instance of an application.

The *exename* parameter must give the name of a valid Microsoft Dynamics SL executable, including the exe extension. If the executable is a standard Microsoft Dynamics SL screen, only the executable name is needed:

```
'standard Customer maintenance screen
Set sivMyScreen = sivToolbar.StartApplication_2("0826000.exe", "0010")
```

If the executable is a non-standard Microsoft SL SDK application, then the path to the executable is required, assuming the main Microsoft Dynamics SL directory as the root:

```
'custom app in file at \Microsoft Dynamics SL\tx\MyApp.exe
Set sivMyScreen = sivToolbar.StartApplication_2("tx\MyApp.exe", "0010")
```

The **StartApplication_2** method, as well as all other methods and property set operations throughout the Microsoft Dynamics SL Object Model, is synchronous with respect to the automation client. That is, control does not return to the automation client until the completion of all Microsoft Dynamics SL

processing caused by the call to this method, whether the processing comes from the core Microsoft Dynamics SL kernel, the original application, or any customization of the application.

A number of original Microsoft Dynamics SL executables do not support the Microsoft Dynamics SL Object Model. None of these are actual end user accounting applications, but are rather various types of utilities. See “Appendix C: Limitations” for more specific information.

Possible Exceptions

- 7506 — Not Logged In
- 7516 — Start Application Error: Login context may not have rights to run the application
- 7517 — Start Application Error: User Limit Exceeded
- 7519 — Start Application Error: Unable to start application due to operating system error: ***specific error number***
- 7520 — Start Application Error: Unable to start application due to fatal error during form load: ***specific error number***
- 7521 — Start Application Error: Initialize mode is on, but user does not have rights to run application in Initialize mode
- 7555 — Start Application Error: Application did not return an object handle
- 7563 — Object Model disabled in application

Example (Visual Basic 6.0 Client)

```
'following code assumes that sivtbCurrent
'is a SIVToolbar object that has already
'successfully logged in

On Error GoTo STARTAPP_ERR
dim sivMyApp As SIVApplication
'TWO POSSIBLE FORMS FOR THE STRING ARGUMENT:
'A) for a custom Microsoft SL SDK app in a directory named tx
'under the main Microsoft Dynamics SL directory:
Set sivMyApp = sivtbCurrent.StartApplication_2("tx\MyApp.exe", “0010”)
'B) for a standard Microsoft Dynamics SL screen:
Set sivMyApp = sivtbCurrent.StartApplication_2("0826000.exe", “0010”)
Exit Sub
STARTAPP_ERR:
Select Case Err.Number - vbObjectError
    Case 7506
    Case 7516
    Case 7517
    Case 7519
    Case 7520
    Case 7521
    Case 7563
    Case Else
End Select
```

StatusBar Control (Microsoft SL SDK Application)

Remarks

Not directly exposed by the Microsoft Dynamics SL Object Model.

However, it is possible to manipulate the status bar on the Microsoft SL SDK application screen through the Microsoft Dynamics SL Object Model by using the **SetStatusBar** method of the **SIVApplication** object.

SubFormDisplay Event

Gives the automation client control over what occurs while a subform is open. Allows the automation client to call any Object Model methods from within an open subform and close the open subform before an event has finished processing.

Applies To

SIVApplication

Syntax

FormName - (ByVal string)

The **SubFormDisplay** event syntax has these parts:

- Part
- Description
- string

The name of the subform being displayed.

Remarks

Normally when an automation client clicks a button to display a subform, control is not returned to the client until the subform is closed. The **SubFormDisplay** event allows an automation client to open a subform, perform an action (that is, exercise control), and then close the subform.

The **SubFormDisplay** event is raised to the automation client whenever a subform is going to be displayed, even if the application being automated is invisible. **SubFormDisplay** supports recursion: a subform can contain a button that opens another subform. Clicking the button within the current event (that is, the first subform) immediately calls the event again for the second subform. The second event contains the name of the second subform in the FormName parameter. This enables the automation client to identify the subform for which the event has been called.

Whenever the automation client is running code in the **SubFormDisplay** event, **SubFormDisplay** should be used only with the subform for which the **SubFormDisplay** event has been raised. For example, if the frmAddress subform is open, the Object Model raises the **SubFormDisplay** event with frmAddress as the FormName parameter. In this case, only controls and entities existing on frmAddress should be manipulated. The resource on any other form should not be accessed.

Note: Technically, the automation client can access the controls and entities on other subforms from within **SubFormDisplay**. However, the results are unpredictable and probably not what is expected.

Value Property (SIVControl Object)

Represents the on-screen contents of a control represented by an **SIVControl** object. The **Value** property is the default property for a **SIVControl** object.

Applies To

SIVControl object

Syntax

object.**Value** = *newvalue*

object = newvalue (Visual Basic 6.0 Clients only)

Remarks

The **Value** property represents what the user sees as the contents of the on-screen control and (unless the control is disabled) can change.

For Visual Basic 6.0 clients, as illustrated in the syntax above, **Value** is the default property for its control, so an explicit reference is not required. In fact, using the implicit reference to the control's **Value** yields better performance. Visual Basic .NET does not permit this type of shortcut, and therefore, **.Value** must be explicitly used.

If there are restrictions on the end user's data entry into the control, then the Microsoft Dynamics SL Object Model enforces those restrictions.

Attempts to set the **Value** property, as well as all other property set operations and method calls throughout the Microsoft Dynamics SL Object Model, are synchronous with respect to the automation client. That is, control does not return to the automation client until the completion of all Microsoft Dynamics SL processing caused by the attempt to set this property, whether the processing comes from the core Microsoft Dynamics SL kernel, the original application, or any customization of the application.

Note that the **Value** property does not necessarily represent the underlying contents of the data field that the control is bound to, since it always represents what the user can see and change. The following list discusses the **Value** property for various types of **SIVControl**:

- **Button** — The **Value** property of the **Button** functions as it does in standard Visual Basic. It can be either True or False. Setting it to True fires the **Button's Click** event. If the **Button** is disabled (**Enabled** property is False) and the automation client sets its value to True, the button is not clicked and you will generate an exception.
- **Label** — The **Value** property corresponds to the label's caption.
- **SAFCheck** — The **Value** property is True for a checked box and False for an unchecked box. Note that **SAFCheck** only supports these two values as opposed to the standard Visual Basic **CheckBox** control's *vbChecked*, *vbUnChecked*, and *vbGrayed*.

Note: Value can only be True or False, regardless of the TrueText or FalseText setting of the control.

- **SAFCombo** — The **Value** property corresponds to the contents of the text box in the **ComboBox**. You cannot add text that does not already exist as an option in the **ComboBox** list. Beware that **ComboBox** lists do not generally correspond to the exact contents of underlying data fields. If a Microsoft Dynamics SL customization has disabled a list entry, you will not be able to set the **Value** property to that list entry.
- **DSLDate** — The only format allowed for dates throughout the Microsoft Dynamics SL Object Model is a **String** or **Variant** of the format "MM/DD/YYYY." All reads will return values in this format. If you attempt to write the **DSLDate Value** property with an invalid date format, you will generate an exception.

- **SAFFloat** — You can assign to the **Value** property any numeric type compatible with a **Double** type, or you can assign a **String Variant** type. However, if it is a string, the string cannot contain any currency symbols or commas, and if it contains a decimal point, the decimal point must be US-format decimal float (“.”). In other words, the only legal characters are the digits, the negative sign (“-”), and the US-format decimal point. Attempting to assign other characters will generate an exception.
- **SAFInteger** — You can assign to the **Value** property any numeric type compatible with an **Integer** type, or you can assign a **String Variant** type. However, if it is a string, the string cannot contain any currency symbols, commas, or decimal points. In other words, the only legal characters are the digits and the negative sign (“-”). Attempting to assign other characters will generate an exception.
- **SAFMaskedText** — The **Value** property is a string. If the contents of the string that you attempt to assign to the **SAFMaskedText** do not agree with **Mask** property of the SAFMaskedText control, then an exception is generated.
- **SAFOption** — The **Value** property is Boolean True for selected **OptionButtons** and False for unselected **OptionButtons**.

Note: Value can only be True or False, regardless of the TrueText or FalseText setting of the control.

- **Standard Visual Basic and Third-party controls** — The **Value** property exposes the object’s default property. Not all third-party ActiveX controls and no .Net WinForm controls have a default property. If Microsoft Dynamics SL Object Model automation code attempts to set the value of a control that does not support a default property, an exception is raised.

Possible Exceptions

See each control’s reference page for specific error numbers.

See Also

“SIVControl Default Property” topic in **Appendix G: Visual Basic .NET-Related Changes, SIVControl Object**

Example (Visual Basic 2005 Client)

```
Try
    'Assumes that sivApp is an already-instantiated
    'SIVApplication object
    'Note use of visible text of element, not
    'underlying data value
    sivApp.Controls("cTerms").Value = "Net 30"
    'Selecting an Option control from among various
    'members of a Control Array
    sivApp.Controls("cCreditType(1)").Value = True
    'Attempt to assign an SAFFloat control's value
    'using a string-type variant
    'The following line will cause an error, because
    'the string includes forbidden characters --
    'currency symbol, thousands separator, and decimal
    'is not in US format:
    sivApp.Controls("cTotal").Value = "¥1.000.000,00"
    'Attempt to assign a DSLDate control's value
    'using an invalid date format.
    'The following line will cause an error, because
    'the month and day are in the wrong order, and because
    'the year is not four digits long:
    sivApp.Controls("cTerms").Value = "27/03/99"

Catch ex As System.Runtime.InteropServices.COMException
```

```
    Dim lErrNumber As Integer
    If ex.Source.Contains("Solomon") = True Then
        lErrNumber = ex.ErrorCode - vbObjectError
    ElseIf ex.ErrorCode - vbObjectError > 0 Then
        lErrNumber = ex.ErrorCode - vbObjectError
    Else
        lErrNumber = ex.ErrorCode
    End If

    Select Case lErrNumber
        Case Else
            MsgBox(ex.Message, MsgBoxStyle.Exclamation, String.Format("{0}
Error {1}", ex.Source, lErrNumber.ToString()))
    End Select

Catch ex As Exception
    MsgBox("Exception: " + ex.Message, MsgBoxStyle.Exclamation, ex.Source)
End Try
```

Value Property (SIVProperty Object)

Represents the contents of a property represented by a **SIVProperty** object. The **Value** property is the default property for a **SIVProperty** object.

Applies To

SIVProperty object

Syntax

object.**Value** = *newvalue*
object = *newvalue* (Visual Basic 6.0 clients only)

Remarks

The **Value** property represents the assigned value of a control's property.

As illustrated above in the syntax for Visual Basic 6.0 clients, the **Value** property is the default property for its **SIVProperty** object, so an explicit reference is not required. In fact, using the implicit reference to the **SIVProperty**'s **Value** yields better performance. Visual Basic 2005 does not permit this type of shortcut, and therefore, .**Value** must be explicitly used.

If there are restrictions on the assignment of this particular property's value, either from the core **SIVApplication** or from any customizations, the Microsoft Dynamics SL Object Model enforces those restrictions.

The **Value** of a property that you set in code does not persist beyond the lifetime of the parent **SIVApplication** object. In other words, property values are not remembered across sessions.

Attempts to set the **Value** property, as well as all other property set operations and method calls throughout the Microsoft Dynamics SL Object Model, are synchronous with respect to the automation client. That is, control does not return to the automation client until the completion of all Microsoft Dynamics SL processing caused by the attempt to set this property, whether the processing comes from the core Microsoft Dynamics SL kernel, the original application, or any customization of the application.

Possible Exceptions

An infinite number of possible exceptions can occur when setting a property value, and each control will have different properties rules. Microsoft strongly recommends careful error trapping around code that sets property values.

See Also

SIVProperty Object

Example (Visual Basic 6.0 Client)

```
'Set a property object variable
'to point to a property in a SIVControl's Properties collection
Dim propCurr As SIVProperty
Set propCurr = currCtrl.Properties.Item("BackColor")
propCurr.Value = vbRed

'Or it can be set like this since value is the default property
propCurr = vbRed


'Change a control directly
sivMyApp.Controls("cTotalBalance").Properties.Item("BackColor").Value = vbRed
```

```
 'Or it can be set like this since value is the default property
sivMyApp.Controls("cTotalBalance").Properties.Item("BackColor") =
vbRed
```

Example (Visual Basic 2005 Client)

```
Try

    'Set a property object variable
    'to point to a property in a SIVControl's Properties collection
    Dim propCurr As SIVProperty
    propCurr = currCtrl.Properties("BackColor")
    propCurr.Value = VBRUN.ColorConstants.vbRed

    'Change a control directly
    sivMyApp.Controls("cTotalBalance").Properties("BackColor").Value =
VBRUN.ColorConstants.vbRed

Catch ex As System.Runtime.InteropServices.COMException

    Dim lErrNumber As Integer
    If ex.Source.Contains("Solomon") = True Then
        lErrNumber = ex.ErrorCode - vbObjectError
    ElseIf ex.ErrorCode - vbObjectError > 0 Then
        lErrNumber = ex.ErrorCode - vbObjectError
    Else
        lErrNumber = ex.ErrorCode
    End If

    Select Case lErrNumber
        Case Else
            MsgBox(ex.Message, MsgBoxStyle.Exclamation, String.Format("{0}
Error {1}", ex.Source, lErrNumber.ToString()))
    End Select

Catch ex As Exception
    MsgBox("Exception: " + ex.Message, MsgBoxStyle.Exclamation, ex.Source)
End Try
```

Visible Property

Controls the physical visibility of the toolbar or the application to the interactive user.

Applies To

SIVToolbar, SIVApplication

Syntax

*Object.*Visible = *Boolean*
Boolean = Object.Visible

The Visible property syntax has these parts:

Part	Description
object	A SIVApplication or SIVToolbar object.
Boolean	A Boolean expression specifying whether the object is visible to the user.

Settings

The settings for *Boolean* are:

Setting	Description
True	Object is visible.
False	(Default) object is invisible.

Remarks

For the SIVToolbar, a Visible property of True would allow the interactive user to use the toolbar. If you need to run a completely automated task with no user interaction, the best choice would be to leave the Visible property at its default value of False.

If you destroy all automation client instances of the SIVToolbar object but without calling the Quit method of SIVToolbar, then the toolbar automatically becomes visible to the interactive user.

If an interactive user starts Microsoft Dynamics SL while a client automation session is already running with an invisible instance of SIVToolbar, then the SIVToolbar automatically becomes visible.

Note: Each instance of the SIVApplication object has its own Visible property. This means that you can control the visibility of the toolbar and the visibility of a Microsoft Dynamics SL screen independently of each other.

By default the SIVApplication object is invisible (Visible property = False) when it is initialized by the StartApplication method of the SIVToolbar object. When the SIVApplication object or SIVToolbar object is invisible, it does not appear on the Windows Taskbar.

If the automation client destroys an SIVToolbar or SIVApplication object before calling the object's Quit method, then that object remains running and becomes visible to the user.

Attempts to set the Visible property, as well as all other property set operations and method calls throughout the Microsoft Dynamics SL Object Model, are synchronous with respect to the automation client. That is, control does not return to the automation client until the completion of all Microsoft Dynamics SL processing caused by the attempt to set this property, whether the processing comes from the core Microsoft Dynamics SL kernel, the original application, or any customization of the application.

Possible Exceptions

See Also

SIVApplication Object, SIVToolbar Object

Example

```
Set sivtbCurrent = New SIVToolbar
sivtbCurrent.Login "CHOMPER", "MSINTL", _
"NWIND", "NDavolio", "yowsa"

dim sivMyApp As SIVApplication
Set sivMyApp = sivtbCurrent.StartApplication("tx\MyApp.exe")
sivtbCurrent.Visible = True
sivMyApp.Visible = True
```

Appendix A: Integrating with Microsoft Office

Since both Microsoft Dynamics SL and the various features of Microsoft Office (such as Microsoft Word, Excel, Outlook, or PowerPoint) provide COM servers, it is possible to automate Microsoft Dynamics SL and Microsoft Office in the same COM client code. This makes it possible to tightly integrate Microsoft Dynamics SL with the features of Microsoft Office.

There are three ways that you can integrate Microsoft Dynamics SL with Microsoft Office:

- Automate Microsoft Dynamics SL and Microsoft Office features from a single stand-alone Visual Basic client application.
- Automate Microsoft Dynamics SL from within a VBA script in a Microsoft Office feature.
- **Microsoft SL SDK programmers only**: Automate Microsoft Office features from within a Microsoft SL SDK application.

The exercise that accompanies this appendix illustrates how to automate the Microsoft Dynamics SL Object Model from within an Excel VBA script.

For further examples of the integration of Microsoft Office features with standard Microsoft Dynamics SL screens, see the Microsoft Web site.

Exercise: Integrating Microsoft Dynamics SL with Microsoft Office

The Excel file distributed with this documentation (OMSolomon NWind Customers.xls) contains the material for this exercise. You will not write code for this exercise. Instead, you will examine the VBA code and the Excel spreadsheets as described here.

The spreadsheets and VBA script in this file have the following features:

- A main spreadsheet contains named ranges in the form of spreadsheet columns that will hold customer IDs and company names provided by the sample application 9359600.exe distributed with this document.
- A secondary spreadsheet uses single-cell named ranges to hold Microsoft Dynamics SL login context information (server name, system database name, company ID, and user ID) that can be “remembered” from one spreadsheet session to the next.
- In Excel’s Macro programming environment, note in the **Tools | References** menu option that a reference has been set to both the Microsoft Dynamics SL **Toolbar** and **Application** object libraries.
- The main VBA script routine is found under the macro named ImportCustomers. Your task in this exercise is to identify each of the following main tasks performed by this routine:
  - Calls a subroutine with accompanying dialog form to log on to the toolbar. The dialog gets initial information and also stored successful login contexts in the secondary login context spreadsheet mentioned above.
  - Starts the 9350000 application.
  - Clears the named spreadsheet ranges for customer ID and company name.
  - Loops through all customer records on the application’s main data entity, populating the spreadsheet ranges with customer IDs and company names respectively.
  - Closes down and destroys **Application** and **Toolbar** objects.

Appendix B: Information for Microsoft SL SDK Programmers

Note: This appendix will be of most interest to those who plan to create Microsoft Dynamics SL applications with the Microsoft SL SDK.

This appendix presents information that will be useful to Microsoft SL SDK programmers when they create applications that might be automated through the Object Model, or when they want to do Object Model automation from within a Microsoft SL SDK application.

The appendix discusses:

- How to disable a Microsoft SL SDK application so that it cannot be automated through the Object Model.
- How to make a Microsoft SL SDK application be an Object Model client (how to automate other Microsoft Dynamics SL applications from within a Microsoft SL SDK application).
- How to use a custom object to enhance the basic functionality of the Object Model in a Microsoft SL SDK application.

Disabling the Object Model for a Microsoft SL SDK Application

If the specifications for the Microsoft SL SDK application that you are writing require that the application not be automatable, then you can disable the Object Model for client manipulation by calling the DisableObjectModel statement in the Form_Load of the Microsoft SL SDK application's main form. You must make this statement call between the AppInit and ScreenInit calls in Form_Load. The documentation for the DisableObjectModel statement contains further details and an example.

Making a Microsoft SL SDK Application Behave as an Object Model Client

You can make your Microsoft SL SDK application behave as a Microsoft Dynamics SL automation client to another Microsoft Dynamics SL application by following these steps in your Microsoft SL SDK project:

1. Set a reference (in the **Visual Basic Project | References** dialog) to the Microsoft Dynamics SL Application Objects Library.
2. Declare a variable of type **SIVApplication**. Make sure to use the **WithEvents** keyword if you want access to the object's **Message** event (note that you cannot use WithEvents in a local declaration, so in this case you have to declare the object with form-wide or application-wide scope).
3. Set the **SIVApplication** variable equal to the result of the **StartAppAndAutomate** function call. The **StartAppAndAutomate** function takes arguments that give the name (and, if necessary, the path) of the application to be automated, as well as any Microsoft Dynamics SL or operating system errors that were raised in the attempt to initialize the application.
4. Manipulate the newly-instantiated **SIVApplication** object according to the rules of Microsoft Dynamics SL Object Model programming.

This technique is useful for drilldowns. If your Microsoft SL SDK application needs to display another Microsoft Dynamics SL application to the interactive user, you can instantiate an **SIVApplication** object for the called application, and then, depending on how much control your original application needs to keep over the called application, either:

- Set the **SIVApplication** object's **Visible** property to True (which keeps control of the called application inside your Microsoft SL SDK application) or
- Set the **SIVApplication** object variable to Nothing, thus exposing the object to the interactive user's full control and releasing any control from your original Microsoft SL SDK application.

Explanation of Custom Objects

Microsoft SL SDK programmers can use a **Custom** object to provide special functionality to automation clients that would not normally be available through the Object Model.

A Microsoft SL SDK application must first expose a **Custom** object to potential automation clients through a custom class of the application. Automation clients of that particular Microsoft SL SDK application can then use the **Custom** object within their code.

Using a Custom Object in an Automation Client

When a Microsoft SL SDK application exposes a custom object through the ExposeCustomObject function call, then clients of that application can automate the custom object through the **GetCustomObject** method of the application's **SIVApplication** object.

Getting a Handle to a Custom Object

Before you can work with an application's custom object, you must obtain a handle to the object by calling the **GetCustomObject** method of the **SIVApplication** object representing the application.

The method returns a variable of type Object. You can then use that object variable to manipulate the application's custom object.

If the object returned by the **GetCustomObject** method is equal to Nothing, then that means that the particular application does not have a custom object.

Manipulating a Custom Object

You can manipulate an **SIVApplication** object's custom object after you get a handle to the custom object.

In order to effectively manipulate the object, you must know its object structure (methods and properties). Since you are manipulating a variable of type object, then the custom object is late bound. That is, the Visual Basic design environment and the compiler do not have the information about the object's structure available at design- or compile-time. You therefore have to rely completely on the documentation of the custom object provided by the Microsoft SL SDK programmer.

Because of the late binding of the custom object in the application client, any syntax errors with the custom object will therefore not be caught by the Visual Basic compiler and will be generated as runtime errors in the application. You must therefore thoroughly debug your automation client code for errors with the custom object syntax.

The following is an example of code that manipulates an **SIVApplication** object's custom object.

Example

```
'Assumes that sivApp is an already-instantiated
'SIVApplication object
    Dim oCustom As Object
    Set oCustom = sivApp.GetCustomObject()
    If oCustom Is Nothing Then
        MsgBox "No Custom Object Available"
    Else
        'manipulate object model members
        oCustom.LoadInfo
        Dim iStatus As Integer
        iStatus = oCustom.Status
    End If
```

Appendix C: Limitations

The following Microsoft Dynamics SL features are either not available or are not fully supported in the Object Model:

- Customization Manager — It is not possible to create or edit a customization through the Object Model. However, it is possible to specify customization levels and other options for a toolbar session from the Object Model.
- DBCreate — This feature is not available through the Object Model.
- DBUpdate — This feature is not available through the Object Model.
- DB Validate and Repair — This feature is not available through the Object Model.
- Events — The Object Model does not handle Microsoft Dynamics SL events, such as **Chk**, **Default**, **LineGotFocus**, **LineChk**, **Update**, **New**, and others.
- Integrity Checkers — This feature is not available through the Object Model.
- Notes — The **Notes/Attachments** icon's functionality for key fields of tables is not accessible through the Object Model.
- **PV** property — The **PV** (possible values) property is accessible through the Object Model but is not available in the dropdown *PV* dialog box. However, messages returned to the user interface by PV validation (such as "item not found").
- Relative Date/Period — This functionality is not available through the Object Model.
- ROI — You cannot call Report Options Interpreter directly or indirectly through the Object Model. This prohibition includes using any user interface feature that would trigger the printing of a report, such as releasing a batch or firing a command button that runs a report. This restriction is because of the fact that the underlying report engine uses dialogs and other user interface features that the Object Model client cannot detect.
- Templates — You cannot create or manage templates from the Object Model.
- Transaction Import — This feature is not available from the Object Model. However, you can use Object Model processing to replace transaction import functionality.

Appendix D: Advanced Tasks

Working with Troublesome Controls

Microsoft Dynamics SL supports the behavior of all of its SAF-type controls, plus the standard WinForm Button and Label controls, as advertised in this documentation.

This does not mean that you cannot program with other controls that appear on an application screen (such as third-party controls or Microsoft controls other than the Button and Label). However, Microsoft technical support cannot answer questions about these other types of controls or guarantee their behavior.

Perhaps the thorniest issue that you will run into when programming with non-Microsoft Dynamics SL controls is the idea of the control's default property. Recall that your automation client can access a control's contents with the syntax:

```
Myapp.Controls("ControlName").Properties("PropertyName")
```

Where of course "ControlName" is the name of the specific control that you are interested in and "PropertyName" is the name of the property representing the control's contents. For SAF controls, the default property of a control is named "Value" and you can always refer to it by default with the following more abbreviated syntax:

```
Myapp.Controls("ControlName")
```

This syntax works with many controls besides the SAF controls, as long as each control type has a default property. For instance, the Visual Basic Label control's default property is its caption, the Visual Basic Button's default is the **Value** property, and the Visual Basic Scrollbar control's default is also its **Value** property.

However, you cannot be certain that every non-Microsoft Dynamics SL control type that you find in a Microsoft SL SDK application will have a default property. If you do run into such a control and forget to use the longer syntax listed in the first example above, then your attempt to use its default property will generate Object Model error 7560.

When in doubt about whether or not a control has a default property or not, always refer to the desired property explicitly, as in the first syntax model above.

Appendix E: Error Numbers

Note: The values given in the main table for each error number are the decimal value, less the value of the constant vbObjectError. For a cross-reference with values in hex and decimal and hex values that include vbObjectError, see the supplementary table at the end of this section.

Error Number	Description
2048	XXXX: Non-fatal exception (all Object Model). *Most Likely Cause*: A call to the Object Model has triggered a recoverable Microsoft Dynamics SL error that would normally be displayed to the interactive user as a message box. The error number is given in XXXX and can be found in the Messages.csv file. The **Message** event fires before this event is raised. *Possible Remedy*: You may trap for and ignore this error, since the **Message** event either handles the error or allows the default user response. You may also write logic to react to the condition brought about by the underlying Microsoft Dynamics SL error whose code is given by XXXX.
2049	XXXX: Fatal exception (all Object Model). *Most Likely Cause*: A call to the Object Model has triggered a non-recoverable Microsoft Dynamics SL error that would normally be displayed to the interactive user as a message box. The error number is given in XXXX and can be found in the Messages.csv file. *Possible Remedy*: During development, identify the error given by XXXX and, if possible, rectify your code so that the error does not happen. If the error happens in the production application, then you cannot gracefully recover from the error once it has been triggered. The only possible reaction when the error occurs at runtime in your compiled application is to destroy the **SIVApplication** object (set it to Nothing) and discontinue processing.
2050	[Internal COM exception] *Text of error message* (Object Model). *Most Likely Cause*: An error unrelated to the Microsoft Dynamics SL environment has been raised in the internal code of the Object Model. *Possible Remedy*: You may attempt to modify your application to react to this error based on text of error message. However, it is most likely that the best way to get a resolution to the error is to notify Microsoft technical support, reporting text of error message.
7500	Login Error: Already Logged In (**Login** method). *Most Likely Cause*: Attempt to call the **SIVToolbar's Login** method a second time. *Possible Remedy*: Use error trapping to ignore the second call to login, or debug your code's logic to avoid a second call to login.
7501	Login Error: System Database Name is greater than 20 (**Login** method). *Most Likely Cause*: You supplied a SystemDatabaseName argument to the **SIVToolbar's Login** method that was too long. *Possible Remedy*: Verify and use the correct system database name. You might have forgotten to trim a string that had many blanks in it.
7502	Login Error: Incorrect System Database Server Name or System Database Name (**Login** method). *Most Likely Cause*: You supplied an incorrect SystemDatabaseServerName or SystemDatabaseName argument to the **SIVToolbar's Login** method, or an incorrect combination of these two arguments. *Possible Remedy*: Verify and use the correct system database server

Error Number	Description
	name for the system database that you want to use. One possible situation in which this error might occur is when you test code in a particular database context with hard-coded login context values and you forget to change them when you move the code to a different environment. Also, make sure that any string value that you pass as system database or system database server name has any extra blanks trimmed out of it.
7503	Login Error: Incorrect Company Id, User Id, or Password (**Login** method). *Most Likely Cause*: One of the above arguments to the **SIVToolbar's Login** is incorrect, or the combination is incorrect. You might have a correct user ID, but the incorrect password, or you might be using a user ID that only exists in one of several companies, and you are trying to log on to a company where the user does not exist or the password is different. You might have forgotten to trim a string that had extra blanks in it. *Possible Remedy*: Verify and use a valid login context. Consider creating a special user ID that only your automation client uses and that has sufficient rights to perform all necessary actions. Make sure that any string value that you pass as company ID, user ID, or password is correctly trimmed.
7504	Login Error: System Database Does Not Exist (**Login** method). *Most Likely Cause*: You supplied a SystemDatabaseName argument to the **SIVToolbar's Login** method that was incorrect. You may be mistaken about which system database belongs with which system database server. *Possible Remedy*: Verify and use correct system database server name for the system database that you want to use. One possible situation in which this error might occur if you are testing code in a particular database context with hard-coded login context values and you forget to change them when you move the code to a different environment. Also you might have forgotten to trim a string that had blanks in it.
7505	Login Error: Interactive User is Different from Client Login (**Login** method). *Most Likely Cause*: The interactive user is already logged on to the workstation, and your automation client is trying to log on with a different Microsoft Dynamics SL login context. *Possible Remedy*: Your automation client's login context must agree with the login context of any interactive user who is already logged on to the same workstation when the automation client attempts to log on.
7506	Not Logged In (**Logout** method, **StartApplication** method, **SetBusinessDate** method, **SetCustomizationLevel** method, **InitializeMode** property). *Most Likely Cause*: You have attempted to manipulate an **SIVToolbar** property or call its **StartApplication** method without first successfully calling its **Login** method. *Possible Remedy*: Make sure your code calls the **Login** method. If it does call the **Login** method, then you should put error trapping in the code for errors 7501 through 7505 to detect if there is a problem with the **Login** method.
7507	Logout Error: Applications Still Running (**Logout** method). *Most Likely Cause*: You tried to set the call the **SIVToolbar** object's **Logout** method, but there were **SIVApplication** objects still running. *Possible Remedy*: Make sure all **SIVApplication** objects have been closed (with the **Quit** method) before attempting to call the **SIVToolbar's Logout** method.
7509	Customization Level Error: Applications Running (**SetCustomizationLevel**

Error Number	Description
	method). *Most Likely Cause*: You tried to call the **SetCustomizationLevel** method of the **SIVToolbar** object, but there were **SIVApplication** objects running at the same time. *Possible Remedy*: Make sure all **SIVApplication** objects have been closed (with the **Quit** method) before attempting to call the **SIVToolbar's SetCustomizationLevel** method.
7510	Customization Level Error: Enumeration Selection Error (**SetCustomizationLevel** method). *Most Likely Cause*: You tried to call the **SetCustomizationLevel** method of the **SIVToolbar** object, but you passed it an invalid level value as the first argument. *Possible Remedy*: Verify the customization level that you passed as the first argument to the **SetCustomizationLevel** method. Make sure that you only use integer values that belong to the enumerated type **sivCustomizationLevel**.
7511	Customization Level Error: No Access Rights (**SetCustomizationLevel** method). *Most Likely Cause*: You tried to call the **SetCustomizationLevel** method of the **SIVToolbar** object, but the current user ID does not have access rights for setting the customization level. *Possible Remedy*: Create or find a user ID with sufficient access rights to set the customization level, and then have the **SIVToolbar** object log on to the database with that user ID before attempting to call the **SetCustomizationLevel** method.
7512	Customization Level Error: Specified UserID is invalid (**SetCustomizationLevel** method). *Most Likely Cause*: You tried to call the **SetCustomizationLevel** method of the **SIVToolbar** object to sivCstLvlOneUser, but the user ID you specified in the second argument does not exist. *Possible Remedy*: Verify the valid user IDs for the current database (or create a new user ID) and make sure that you are using one of those user IDs as the second argument to the **SetCustomizationLevel** method.
7514	Business Date Error: Invalid Date (**SetBusinessDate** method). *Most Likely Cause*: The combination of arguments that you passed to the **SetBusinessDate** method of the **SIVToolbar** object does not specify a valid date. *Possible Remedy*: Verify that you have correctly specified the three arguments to **SetBusinessDate** so that they yield a valid date. Some possible common mistakes in specifying the date are: specifying February 29th on a year that is not a leap year; specifying the 31st for months with only 30 days, or the 30th for February; only specifying two digits for the year (years must always be specified with four digits). Also note that, regardless of the system's date settings, the order of the arguments to **SetBusinessDate** is always month, day, year.
7516	Start Application Error: Login context may not have rights to run the application (**StartApplication** method). *Most Likely Cause*: The user ID argument that you supplied when you called the **Login** method of the **SIVToolbar** object at the beginning of the current session represented a user that did not have sufficient rights to run this application. *Possible Remedy*: Get the system administrator to assign sufficient rights to the user ID, or create a special user ID that only your automation client uses and that has sufficient rights.
7517	Start Application Error: User Limit Exceeded (**StartApplication** method). *Most Likely Cause*: The current installation of Microsoft Dynamics SL is

Error Number	Description
	licensed for a number of users that is below the logged-on user count that would obtain if this application were to run. *Possible Remedy*: Make sure that fewer users are running applications when you re-try, or convince management to buy more user licenses for Microsoft Dynamics SL.
7519	Start Application Error: Unable to start application due to operating system error: Error Number (**StartApplication** method). *Most Likely Cause*: This error could be raised due to many causes, such as a temporary problem on the network or some other unforeseen problem. That is why this particular error contains features a further error number embedded at the end of the **Description** property of the **Err** object. A very common possible cause of this error would be that you have passed an invalid path or file name as the argument to the **StartApplication** method. *Possible Remedy*: If you want to write code that is as robust as possible, you should have code in your error trap parse out this second error code and then decide on the appropriate action to take.
7520	Start Application Error: Unable to start application due to fatal error during form load: Error Number (**StartApplication** method). *Most Likely Cause*: This error can have many causes and so has a more specific error number embedded at the beginning of the **Err.Description** property. Unlike error 7519, however, this error is caused by something that happens in the Microsoft Dynamics SL application as the application begins to run, but before the application is fully loaded in memory. For this type of error, the Microsoft Dynamics SL user interface would probably display a message box to the user (thus firing the **Message** event) or a runtime error would be raised. However, a special situation occurs before the application has fully loaded in memory, because the Microsoft Dynamics SL Object Model is not completely initialized at that point. Therefore, the environment is not yet able to raise every type of runtime error or fire the **Message** event. *Possible Remedy*: You can check the beginning of the **Err.Description** property for the specific error number that would have been raised as a runtime error or in the **Message** event, if it were possible to raise the **Message** event in this environment.
7521	Start Application Error: Initialize mode is on, but user does not have rights to run application in Initialize mode (**StartApplication** method). Most Likely Cause: The particular user ID with which the toolbar has logged on does not have Initialize mode rights for the screen that you are trying to run in the **SIVApplication** object. Possible Remedy: Get the system administrator to assign sufficient rights to the user ID, or create a special user ID that only your automation client uses and that has sufficient rights.
7522	Quit Error: Automation client is not logged out (**Quit** method of **SIVToolbar** object). *Most Likely Cause*: You tried to call the **Quit** method of the **SIVToolbar** object, but the **SIVToolbar** object is still logged on. *Possible Remedy*: Make sure to call the **Logout** method of the **SIVToolbar** object first.
7524	New may be disabled for specified entity, either by the application or because of access rights (**New** method). *Most Likely Cause*: The underlying Microsoft Dynamics SL application or a customization has disabled New on the toolbar. *Possible Remedy*: Trap for this error and take appropriate steps.
7525	Invalid Entity String (**New**, **Delete**, **First Last**, **Next**, **Previous**, **GetEntityStatus** methods).

Error Number	Description
	Most Likely Cause: You used an entity string as an argument to one of the above methods, but the application does not recognize an entity of that name. *Possible Remedy*: Verify the available entity strings for the application. It may be a simple misspelling, or you may have an untrimmed string containing extra blanks.
7526	Delete may be disabled for specified entity, either by the application or because of access rights (**Delete** method). *Most Likely Cause*: The underlying Microsoft Dynamics SL application or a customization has disabled **Delete** on the toolbar. *Possible Remedy*: Trap for this error and take appropriate steps.
7528	First may be disabled for specified entity by the application (**First** method). *Most Likely Cause*: The underlying Microsoft Dynamics SL application or a customization has disabled **First** on the toolbar. Also, the underlying application may represent control information (such as *GL Setup* (01.950.00)) that does not support the navigation buttons. *Possible Remedy*: Trap for this error and take appropriate steps.
7530	Last may be disabled for specified entity by the application (**Last** method). *Most Likely Cause*: The underlying Microsoft Dynamics SL application or a customization has disabled **Last** on the toolbar. Also, the underlying application may represent control information (such as *GL Setup* (01.950.00)) that does not support the navigation buttons. *Possible Remedy*: Trap for this error and take appropriate steps.
7532	Next may be disabled for specified entity by the application (**Next** method). *Most Likely Cause*: The underlying Microsoft Dynamics SL application or a customization has disabled **Next** on the toolbar. Also, the underlying application may represent control information (such as *GL Setup* (01.950.00)) that does not support the navigation buttons. *Possible Remedy*: Trap for this error and take appropriate steps.
7534	Previous may be disabled for specified entity by the application. (**Previous** method). *Most Likely Cause*: The underlying Microsoft Dynamics SL application or a customization has disabled **Previous** on the toolbar. Also, the underlying application may represent control information (such as *GL Setup* (01.950.00)) that does not support the navigation buttons. *Possible Remedy*: Trap for this error and take appropriate steps.
7537	Value specified has been disabled (**Value** property of **SIVControl** object of type SAFCombo). *Most Likely Cause*: Even though the list value that you specified is an option on this SAFCombo control's list, that value has been disabled by the underlying application or customization logic, and you cannot set the control to that value. *Possible Remedy*: Verify that the list value chosen by the automation client is not disabled. For most robust code, trap for this error in procedures that assign or read an SAFCombo's value.
7538	Value specified does not exist (**Value** property of **SIVControl** object of type SAFCombo). *Most Likely Cause*: The list value that you are trying to assign or read is not an option on this SAFCombo control's list, either in the original version of the application or in any customization of the list. Both original and customized list values provide valid values. It is possible, however, that the automation client is assuming that a customized value is

Error Number	Description
	available and either the **CustomizationLevel** property is incorrect for the customization to apply, or the assumed customization is not available. It is also possible that you are trying to assign or read a value that corresponds to the underlying value in the data field. Instead, you should always assign a value that corresponds to a list choice that is visible to the user in the original or customized screen. *Possible Remedy*: Verify that the value you are trying to specify exists as an original value of the list or as a customized value. For most robust code, always use original, uncustomized list values, which will work regardless of which customizations are currently applied. Verify that the value you use in your code for an SAFCombo corresponds to the list entry seen by the user (such as "Invoice"), and not to the value of the underlying data field (such as "I").
7539	Data does not conform to Float format (**Value** property of **SIVControl** object of type SAFFloat). *Most Likely Cause*: Automation code has attempted to assign a value to an SAFFloat control that was either too large or small for a Float value or that contained string data not formatted for the Float value. If you assign a string, you can only include numeric characters or the US decimal place holder (".") in the string. Not even thousands separators or currency symbols are acceptable. Decimal place holders other than US are not acceptable, regardless of the workstation's international settings. *Possible Remedy*: Use the Visual Basic Format function (or its equivalent in other programming environments) wherever possible when you assign the value of an SAFFloat control. The format string should be similar to "########.##" without any other formatting characters.
7540	Data does not conform to Integer format (**Value** property of **SIVControl** object of type SAFInteger). *Most Likely Cause*: Automation code has attempted to assign a value to an SAFInteger control that was either too large or small for an Integer value or that contained string data not formatted for the integer value. If you assign a string, you can only include numeric characters in the string. Not even thousands separators, decimal place holders, or currency symbols are acceptable. *Possible Remedy*: Use the Visual Basic Format function (or its equivalent in other programming environments) wherever possible when you assign the value of an SAFInteger control. The format string should be similar to "########" without any other formatting characters.
7541	Data does not match the mask specified (**Value** property of **SIVControl** object). *Most Likely Cause*: The **SIVControl** object's **Mask** property has a different format from the format of the data that you are trying to assign to the control. You may be trying to assign invalid data types (as specified by the mask) at one or particular character positions, or an invalid number of characters. You may be trying to include mask formatting characters (such as slashes, commas, or dashes) in your assignment. *Possible Remedy*: Verify the control's **Mask** property and make sure that it matches the data types of the individual characters that you are trying to assign and the valid data length specified by the mask. Also make sure that your assignment does not attempt to include mask formatting characters. For instance, if a mask specified "###-##-####" then a correct assignment would be "543681287" and an incorrect assignment (that would cause this error) would be "543-68-1287" because the dashes are non-significant formatting characters.
7542	Date format is incorrect or the date is invalid (**Value** property of **SIVControl** object of type DSLDate).

Error Number	Description
	Most Likely Cause: Automation code has attempted to assign a value to a DSLDate control that is either not in US date format, which is MM/DD/YYYY, or that is an incorrect date, such as 02/29/1999 or 04/31/2004. Only US date format is allowed throughout the Microsoft Dynamics SL Object Model, regardless of the date format settings of the workstation. *Possible Remedy*: Use the Visual Basic Format function with the template "MM/DD/YYYY" (or an equivalent solution in other programming environments) wherever possible when you assign the value of a DSLDate control. Also check the validity of the date before attempting to assign it.
7543	Currency ID cannot be set at this time (**SetCurrencyIDs** method). *Most Likely Cause*: You have called the **SetCurrencyIDs** method, but the underlying application or customization has disabled the ability to change the currency ID. *Possible Remedy*: Always trap for this error when calling the **SetCurrencyIDs** method, and abandon the attempt to set the currency ID if this error occurs.
7545	Transaction Currency ID is invalid (**SetCurrencyIDs** method). *Most Likely Cause*: The currency ID specified in the first argument of the **SetCurrencyIDs** method (TransactionCurrencyID) does not exist in the currency information of the company where the **sivToolbar** object is currently logged on. *Possible Remedy*: Either: Change the currency ID that you pass as the first argument for the **TransactionCurrencyID** method to a currency ID that exists in the company information, or Make sure that someone adds that currency ID for the companies that you plan to work with. To guard against this error happening, devise a way to check for the existence of a currency ID in the database before calling the **SetCurrencyIDs** method.
7547	Intermediate Currency ID is invalid (**SetCurrencyIDs** method). Most Likely Cause: The currency ID specified in the second argument of the **SetCurrencyIDs** method (IntermediateCurrencyID) does not exist in the currency information of the company where the **sivToolbar** object is currently logged on. Possible Remedy: Same as remedy for Error 7545. **Note:** At present, the IntermediateCurrencyID feature is not implemented in the Microsoft Dynamics SL Object Model.
7549	Currency View cannot be toggled (**SetCurrencyView** method). *Most Likely Cause*: You called the **SetCurrencyView** method, but the underlying application has disabled currency changes. *Possible Remedy*: Make sure to trap for this error whenever you call SetCurrencyView. If the error trap detects this error, then ignore the attempt to set the currency view.
7553	Data is too long to fit in the field (**Value** property of any SIVControl). *Most Likely Cause*: Data is too long for the format specified through the control's **Mask** property. *Possible Remedy*: Verify the control's **Mask** property and shorten your data as needed.
7554	Cannot call back into the Object Model from within the **Message** event (**Message** event). *Most Likely Cause*: There is automation code inside an **SIVApplication** object's **Message** event procedure that makes reference to some

Error Number	Description
	member of the Microsoft Dynamics SL Object Model's **SIVApplication** object, such as one of its properties or methods. Any such reference causes this error, because the **Message** event is strictly non-re-entrant with respect to the **SIVApplication** object, meaning that you cannot make any calls to the **SIVApplication** object while the **Message** event is running. *Possible Remedy*: Remove the offending code that calls the SIVApplication object. You will have to use logic that performs the intended action elsewhere in your code (not in the **Message** event). The best alternate place to perform such activity would be in a Visual Basic error trap.
7555	Application is Not Returning an Object Handle, Application May Not Be a Microsoft SL SDK Application (**StartApplication** method). *Most Likely Cause*: The Microsoft Dynamics SL Object Model was unable to run and instantiate an object for the application whose path and file name were given as the argument to the **SIVToolbar** object's **StartApplication** method. *Possible Remedy*: This error might occur because the application was not created with Microsoft SL SDK. You need to talk to the programmer who created the Microsoft Dynamics SL application that you are trying to automate.
7556	**Button** is disabled (**Value** property of an SIVControl whose type is Button). *Most Likely Cause*: You have attempted to set the **Value** property of a Button, but the Button's **Enabled** property is False. *Possible Remedy*: Always verify in your code that the Button's **Enabled** property is True before attempting to set its **Value** property.
7557	Cannot set Option Button to False (SAFOption). *Most Likely Cause*: You have assigned a value of False to an SAFOption control. You can only assign a value of True to SAFOption controls. *Possible Remedy*: The way to un-select an **Option** button is to select a different **Option** button in the same **Option** button group. Therefore, do not use logic that unselects an SAFOption control, but instead determine which SAFOption should be selected in place of the one you wish to de-select. Then assign a value of True to the SAFButton to be selected.
7558	Invalid message response for message number XXXX (any call to Object Model that fires a **Message** event). *Most Likely Cause*: The code in the **Message** event procedure has set a value for the MessageResponse parameter that is not allowed for this particular MessageNumber. *Possible Remedy*: Find out what MessageNumbers (in the **Message** event) the invalid value is getting set for, and change your code in the **Message** event procedure to be aware of those specific MessageNumbers so that it either sets a valid MessageResponse value for them or else ignores them, allowing the default response.
7559	Toolbar cannot be made invisible because the interactive user is logged in (**Visible** property of **sivToolbar** object). *Most Likely Cause*: Your code has attempted to set the **Visible** property of the **sivToolbar** object to False while the interactive user is logged on. *Possible Remedy*: Trap for this error whenever your code sets the **Visible** property of the **sivToolbar** to False. If you detect this error, then recover from it by ignoring the attempt to set the **Visible** property to False.
7560	Control's default property cannot be accessed (third-party controls). *Most Likely Cause*: The third party control whose value you are attempting to read or set has no **Default** property (although most third-party controls have a default property, only Microsoft Dynamics SL's SAF-

Error Number	Description
	type controls are guaranteed to have a **Default** property). *Possible Remedy*: Explicitly name the property when you read or assign its value, using the following syntactic format: `ControlObject.Properties("property name").`
7561	Attempt to set number of decimal places greater than number of decimal places allowed by the control (SAFFloat control). *Most Likely Cause*: You have attempted to set the **Value** property of an SAFFloat control with a number having a greater precision than the number of decimals places specified in the **SAFFloat's Decimals** property. *Possible Remedy*: Either reduce the number of decimal places that you are specifying in your value assignment, or make sure that the **Decimals** property in the underlying SAFFloat control is large enough to hold a number of the precision that you are specifying.
7562	Save may be disabled, either by the application or because of access rights (**Save** method). *Most Likely Cause*: The underlying application or a customization has disabled the save action. *Possible Remedy*: Trap for this error condition, and skip your attempt to save if you encounter this error.
7563	Object Model disabled by application (**StartApplication** method of the **SIVApplication** object). Most Likely Cause: The underlying Microsoft SL SDK application has disabled the Object Model by calling the DisableObjectModel api statement. Possible Remedy: The underlying application must be re-compiled without the call to DisableObjectModel.
7568	Currency Error: Invalid sivCurrencyView enumeration value.
7569	The value of disabled SAF controls cannot be set.

Cross-Reference Between Various Error-Numbering Schemes

The first two columns list the constant value vbObjectError (-2147221504 decimal or 80040000 hex) stripped out. The third and fourth columns contain decimal and hex versions of the same two values, but without vbObjectError stripped out.

Decimal - vbObjectError	Hex - vbObjectError	Decimal (includes vbObjectError)	Hex (includes vbObjectError)
2048	800	-2147219456	80040800
2049	801	-2147219455	80040801
2050	802	-2147219454	80040802
7500	1D4C	-2147214004	80041D4C
7501	1D4D	-2147214003	80041D4D
7502	1D4E	-2147214002	80041D4E
7503	1D4F	-2147214001	80041D4F
7504	1D50	-2147214000	80041D50
7505	1D51	-2147213999	80041D51
7506	1D52	-2147213998	80041D52
7507	1D53	-2147213997	80041D53
7508	1D54	-2147213996	80041D54
7509	1D55	-2147213995	80041D55
7510	1D56	-2147213994	80041D56
7511	1D57	-2147213993	80041D57
7512	1D58	-2147213992	80041D58
7513	1D59	-2147213991	80041D59
7514	1D5A	-2147213990	80041D5A
7515	1D5B	-2147213989	80041D5B
7516	1D5C	-2147213988	80041D5C
7517	1D5D	-2147213987	80041D5D
7518	1D5E	-2147213986	80041D5E
7519	1D5F	-2147213985	80041D5F
7520	1D60	-2147213984	80041D60
7521	1D61	-2147213983	80041D61
7522	1D62	-2147213982	80041D62
7523	1D63	-2147213981	80041D63
7524	1D64	-2147213980	80041D64
7525	1D65	-2147213979	80041D65
7526	1D66	-2147213978	80041D66
7527	1D67	-2147213977	80041D67
7528	1D68	-2147213976	80041D68
7529	1D69	-2147213975	80041D69
7530	1D6A	-2147213974	80041D6A
7531	1D6B	-2147213973	80041D6B
7532	1D6C	-2147213972	80041D6C
7533	1D6D	-2147213971	80041D6D

Decimal - vbObjectError	Hex - vbObjectError	Decimal (includes vbObjectError)	Hex (includes vbObjectError)
7534	1D6E	-2147213970	80041D6E
7535	1D6F	-2147213969	80041D6F
7536	1D70	-2147213968	80041D70
7537	1D71	-2147213967	80041D71
7538	1D72	-2147213966	80041D72
7539	1D73	-2147213965	80041D73
7540	1D74	-2147213964	80041D74
7541	1D75	-2147213963	80041D75
7542	1D76	-2147213962	80041D76
7543	1D77	-2147213961	80041D77
7544	1D78	-2147213960	80041D78
7545	1D79	-2147213959	80041D79
7546	1D7A	-2147213958	80041D7A
7547	1D7B	-2147213957	80041D7B
7548	1D7C	-2147213956	80041D7C
7549	1D7D	-2147213955	80041D7D
7550	1D7E	-2147213954	80041D7E
7551	1D7F	-2147213953	80041D7F
7552	1D80	-2147213952	80041D80
7553	1D81	-2147213951	80041D81
7554	1D82	-2147213950	80041D82
7555	1D83	-2147213949	80041D83
7556	1D84	-2147213948	80041D84
7557	1D85	-2147213947	80041D85
7558	1D86	-2147213946	80041D86
7559	1D87	-2147213945	80041D87
7560	1D88	-2147213944	80041D88
7561	1D89	-2147213943	80041D89
7562	1D8A	-2147213942	80041D8A
7563	1D8B	-2147213941	80041D8B
7564	1D8C	-2147213940	80041D8C
7565	1D8D	-2147213939	80041D8D
7566	1D8E	-2147213938	80041D8E
7567	1D8F	-2147213937	80041D8F
7568	1D90	-2147213936	80041D90
7569	1D91	-2147213935	80041D91
7570	1D92	-2147213934	80041D92
7571	1D93	-2147213933	80041D93
7572	1D94	-2147213932	80041D94
7573	1D95	-2147213931	80041D95
7574	1D96	-2147213930	80041D96
7575	1D97	-2147213929	80041D97
7576	1D98	-2147213928	80041D98

Decimal - vbObjectError	Hex - vbObjectError	Decimal (includes vbObjectError)	Hex (includes vbObjectError)
7577	1D99	-2147213927	80041D99
7578	1D9A	-2147213926	80041D9A
7579	1D9B	-2147213925	80041D9B
7580	1D9C	-2147213924	80041D9C
7581	1D9D	-2147213923	80041D9D
7582	1D9E	-2147213922	80041D9E
7583	1D9F	-2147213921	80041D9F
7584	1DA0	-2147213920	80041DA0
7585	1DA1	-2147213919	80041DA1
7586	1DA2	-2147213918	80041DA2
7587	1DA3	-2147213917	80041DA3
7588	1DA4	-2147213916	80041DA4
7589	1DA5	-2147213915	80041DA5
7590	1DA6	-2147213914	80041DA6
7591	1DA7	-2147213913	80041DA7
7592	1DA8	-2147213912	80041DA8
7593	1DA9	-2147213911	80041DA9
7594	1DAA	-2147213910	80041DAA
7595	1DAB	-2147213909	80041DAB
7596	1DAC	-2147213908	80041DAC
7597	1DAD	-2147213907	80041DAD
7598	1DAE	-2147213906	80041DAE
7599	1DAF	-2147213905	80041DAF
7600	1DB0	-2147213904	80041DB0

Appendix F: Application Browser

You can use the Microsoft Dynamics SL Application Browser to quickly get information about control and property names as you develop scripts to automate applications. To open the browser, double-click SIVAPPBROWSER.exe in \Program Files\Microsoft Dynamics\SL\Applications.

The application browser uses the Microsoft Dynamics SL Object Model for all interaction with the application or the Microsoft Dynamics SL system. This helps ensure that the application browser will always "see" application information from the point of view of your automation clients.

Logging on with the Browser

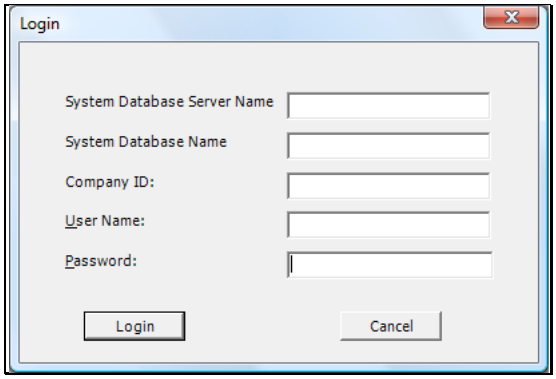


Figure 1: Login window

When the application browser starts, the first thing that you see is the *Login* dialog box.

You must enter the necessary information to log on to the database needed to run the application that you want to examine. This information consists of the same parameters needed for the **Login** method of the **SIVToolbar** object.

Login (Button)

Click **Login** when you are ready to attempt to log on. If the logon fails, a message appears informing you of the logon failure. After dismissing the failure message, you can try again.

If the logon is successful, then you are able to proceed to browse Microsoft Dynamics SL applications.

Cancel (Button)

Cancel shuts down the logon dialog and close the application browser.

Choosing an Application to Browse

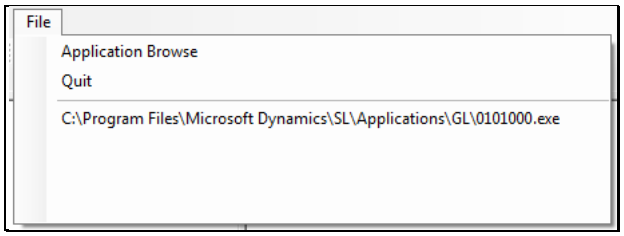


Figure 2: File window

When you first open the browser, it is blank. You must choose an application to browse before you can view any information.

To browse an application, you can click **Browse Application** ( ) on the toolbar or select the Browse Application option from the file menu. A dialog box opens, asking you to find the executable that you want to browse. Enter the directory and file name (this is the same format as required by the argument to the **StartApplication** method of the **SIVToolbar** object.) You can also choose the executable by pressing **Browse,** which opens the *File Open* dialog box.

Once you have chosen the executable to browse, you also specify the customization level to start the application in. When you then click **Ok** after choosing the customization level, the browser starts the application and fills in the left pane with all of the application's control names. If you decide not to proceed with the selected executable after you choose the customization level, click **Cancel**.

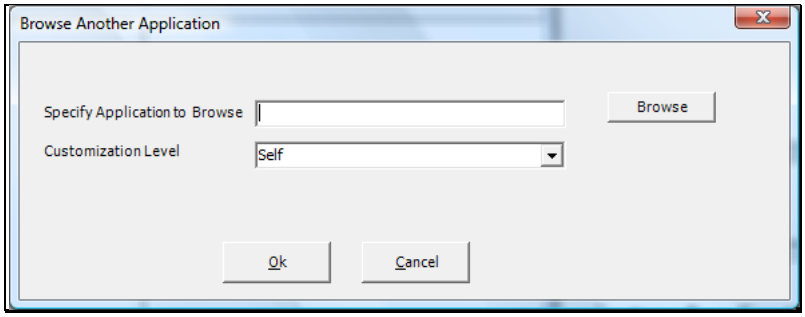


Figure 3: Browse another application window

You can also select previously browsed applications from the file menu. The file menu shows the last four applications browsed.

Once you have selected an application to browse, its executable name appears in the title bar of the browser.

Left and Right Browser Panes

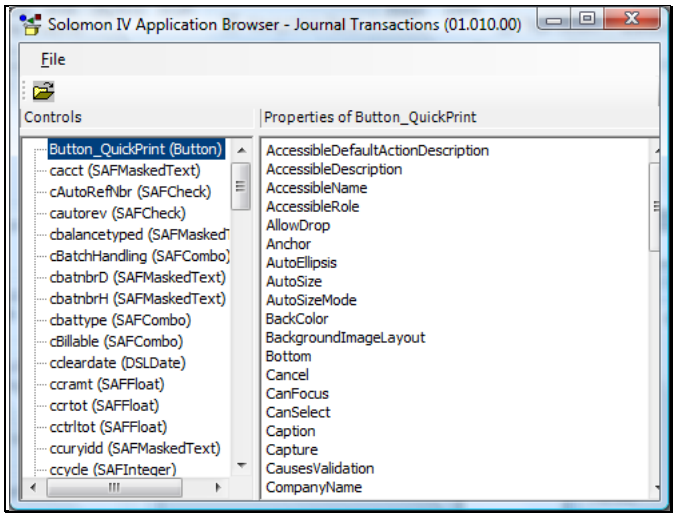


Figure 4: Application Browser window

The pane on the left side of the *Application Browser* window contains the names of all of the controls that are available to the automation client. The control type (identical to the value of the control's **ControlType** property) is in parenthesis after the control name.

When the user selects one of the control names, properties of the control that are available to the automation client are displayed in the pane on the right. This information is for display only.

Exercise: Browsing the Sample Application

For this exercise, you do not need to run the sample application.

1. Run the application SIVAppBrowser.exe under the main Microsoft Dynamics SL directory (the directory where Parent is installed). Complete the necessary login information to log on to Microsoft Dynamics SL.
2. Under the application's File menu, choose an application to browse.
3. When the target application runs, examine the two panes in the browser.
4. Also notice that the application screen becomes visible. If you pause the mouse over a control on the application screen, you will see the control's **FieldName** property in the ToolTip that appears.

Appendix G: Visual Basic .NET-Related Changes

Programming Interface Changes

Programmatic References

The public interface to the Microsoft Dynamics SL Object Model is now contained in the assembly Microsoft.Dynamics.SL.ObjectModel.dll. In the past, Object Model client code typically needed two references in order to access the Microsoft Dynamics SL Object Model:

- Parent.exe
- Swimapi.dll

Now, these two references must be replaced. For unmanaged clients (Visual Basic 6.0), use the reference, Microsoft.Dynamics.SL.ObjectModel.tlb, which can be found in the Program Files folder under Common Files\Microsoft Shared\DynamicsSL:

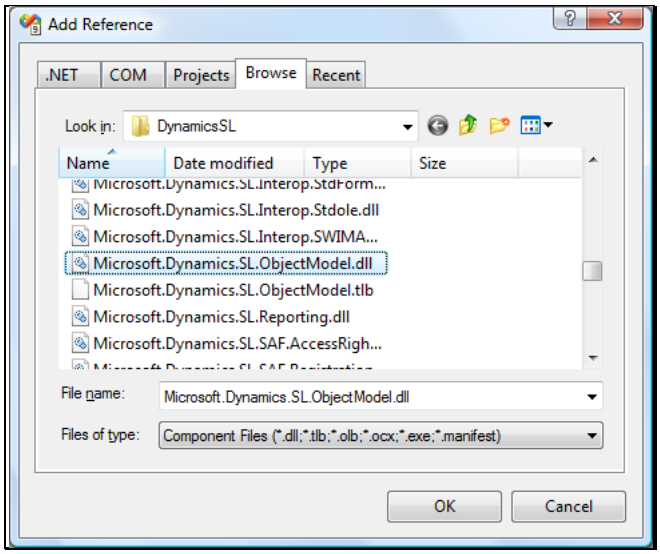


Figure 5: References – VBObjectModelClient project (Visual Basic 2008)

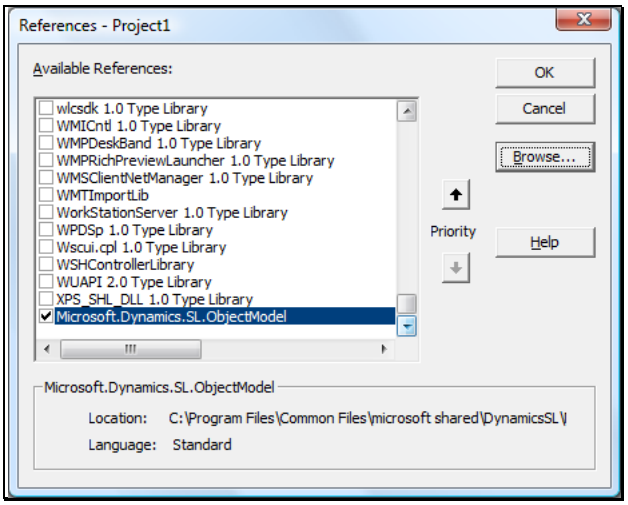


Figure 6: References – VBObjectModelClient project (Visual Basic 6)

Microsoft.Dynamics.SL.ObjectModel.tlb is a COM callable wrapper (CCW) that permits unmanaged clients to call code in the managed .dll, Microsoft.Dynamics.SL.ObjectModel.dll. This file is created by the .NET Assembly Registration Tool (Regasm.exe). This will permit the Microsoft Dynamics SL Object Model to support both Visual Basic 6.0 and .NET clients.

For .NET clients, include a reference to the assembly Microsoft.Dynamics.SL.ObjectModel, which can be found in the Global Assembly Cache. The physical file name of this assembly is Microsoft.Dynamics.SL.ObjectModel.dll, and is located in the Program Files folder under Common Files\Microsoft Shared\DynamicsSL.

Along with changing references, any code that specifies a fully-qualified type name, like

```
Parent.SIVToolbar
SWIMAPI.SIVApplication
```

must be modified to reflect the new library name. Include the following line in all of your source files and simply remove the library qualification:

```
Imports Microsoft.Dynamics.SL.ObjectModel
```

or, use fully-qualified type names:

```
Microsoft.Dynamics.SL.ObjectModel.SIVToolbar
Microsoft.Dynamics.SL.ObjectModel.SIVApplication
```

Enumerations for Unmanaged (Visual Basic 6.0) Clients Only

When Microsoft.Dynamics.SL.ObjectModel.tlb is created by Regasm, all enumerations are converted using the following pattern:

```
<enum name>_<enum_value>
```

Listed below are enumerations for unmanaged Visual Basic 6.0 clients and their Visual Basic 2005 equivalents:

Enumerations	Visual Basic 2005 equivalents
sivRecFndNotFound	sivRecordFound_sivRecFndNotFound
sivRecFndFound	sivRecordFound_sivRecFndFound
sivMsgOk	sivMessageType_sivMsgOk
sivMsgOkCancel	sivMessageType_sivMsgOkCancel
sivMsgAbortRetryIgnore	sivMessageType_sivMsgAbortRetryIgnore
sivMsgYesNoCancel	sivMessageType_sivMsgYesNoCancel

Enumerations	Visual Basic 2005 equivalents
sivMsgYesNo	sivMessageType_sivMsgYesNo
sivMsgRetryCancel	sivMessageType_sivMsgRetryCancel
sivMsgRspOk	sivMessageResponse_sivMsgRspOk
sivMsgRspCancel	sivMessageResponse_sivMsgRspCancel
sivMsgRspAbort	sivMessageResponse_sivMsgRspAbort
sivMsgRspRetry	sivMessageResponse_sivMsgRspRetry
sivMsgRspIgnore	sivMessageResponse_sivMsgRspIgnore
sivMsgRspYes	sivMessageResponse_sivMsgRspYes
sivMsgRspNo	sivMessageResponse_sivMsgRspNo
sivMsgRspClose	sivMessageResponse_sivMsgRspClose
sivCurrencyViewBase	sivCurrencyView_sivCurrencyViewBase
sivCurrencyViewTransaction	sivCurrencyView_sivCurrencyViewTransaction
sivEntStatusInserted	sivEntityStatus_sivEntStatusInserted
sivEntStatusUpdated	sivEntityStatus_sivEntStatusUpdated
sivEntStatusNotChanged	sivEntityStatus_sivEntStatusNotChanged

Default Properties

The concept of a default property on a class is not supported in .NET, unless the property is indexed. At first glance, you might be tempted to think code like the following will continue to work:

```
Dim solomonToolBar As Microsoft.Dynamics.SL.ObjectModel.SIVToolbar
solomonToolBar = New Microsoft.Dynamics.SL.ObjectModel.SIVToolbar
solomonApp = solomonToolBar.StartApplication("SomApp.exe")
solomonApp.Controls("someCtrl") = ControlValue.Text
```

However, *solomonApp.Controls("someCtrl")* is actually shorthand for *solomonApp.Controls.Item("someCtrl")*. *Item* is the default property of the type SIVControls (which is a collection, and therefore, an indexed property) and is declared to be of type SIVControl. In our example, *Item(index)* represents an SIVControl instance, but since SIVControl cannot have a default property, the code *solomonApp.Controls("someCtrl")* or *solomonApp.Controls.Item("someCtrl")* is not equivalent to *solomonApp.Controls("someCtrl").Value*. In the example above, to set the value of the SIVControl, you should use:

```
solomonApp.Controls("someCtrl").Value = ControlValue.Text
```

For managed clients of the Microsoft Dynamics SL Object Model, the old shortcuts of assuming default properties will no longer work. However, for unmanaged clients the shortcut syntax will continue to work since the classes containing collections (SIVControls and SIVProperties) are attributed specifying IDispatch (late-binding only). At runtime, this will permit the shortcut syntax to be supported but only for unmanaged client code.

SIVControl Default Property

Under Visual Basic 6.0, each control had a default property. For example, Caption was the default property of a Label control. A label's Caption could be changed by simply setting a label variable to some string value, like the following:

```
Label = "Some New Caption"
```

The Microsoft Dynamics SL Object Model followed the same convention, allowing the setting of a SIVControl object directly:

```
Dim sivLabel as SIVControl
sivLabel = solomonApp.Controls("lmodule")
```

```
sivLabel = "Some New Caption"
```

Since a label had a default property of Caption, the Object Model understood the developer's intent and permitted the shortcut.

To allow backward compatibility, the Microsoft Dynamics SL Object Model will permit the developer to set an SIVControl default property (using the explicit .Value syntax for managed clients as described in "Default Properties") on the former set of Visual Basic 6.0 controls, such as Label, Form, Button, etc., which are known to Object Model, and on COM controls that specify a default property. So, this code will continue to work:

```
Dim sivLabel as SIVControl
sivLabel = solomonApp.Controls("lmodule")
sivLabel.Value = "Some New Caption"
```

For new WinForm controls, there are no default properties. Therefore, the Object Model cannot gauge the developer's intent and cannot permit this shortcut. If you attempt the shortcut, system message 7560 will be returned to the Object Model client. To work around this issue, be specific about the property to set, rather than attempting the default property shortcut.

Deterministic Freeing of SIVToolbar and SIVApplication Objects

Since the Microsoft Dynamics SL Object Model classes are now written in Visual Basic .NET, setting a variable of one of these types to Nothing will **not** cause an immediate cleanup of the Object Model's COM resources to occur as it did previously. The resource will stay in memory until the .NET garbage collector is run. This means, for example, that an application will continue to run even if an SIVApplication variable is set to Nothing.

To get immediate cleanup of the Object Model resource, call the .Dispose method. This is the preferred method for utilizing the disposable behavior implemented on all Microsoft Dynamics SL Object Model classes. All classes now implement the IDisposable interface, and therefore, support calling a publicly-exposed Dispose method. Here is a code sample that illustrates calling the Dispose method:

```
solomonToolBar.Dispose()
solomonToolBar = Nothing
```

Dispose frees any COM resources and internal memory still held by the Object Model and performs the type of cleanup that was formerly done by setting the variable to Nothing. Note that the object will stay in memory until garbage collection completes, but since Dispose has already freed all critical resources, the small amount of memory held by the object reference is trivial and can stay in memory until the garbage collection process runs.

Error Handling

Unstructured error handling (that is, the "On Error" method of error handling) is still supported in Visual Basic .NET. However, structured error handling (exception handling) is now available and can be used in managed Microsoft Dynamics SL Object Model clients. An exception raised by the Object Model will be of type System.Runtime.InteropServices.COMException. This type of exception includes an ErrorCode, which is analogous to the Err object's Err.Number field. Here is the mapping between the Err Object and the System.Runtime.InteropServices.COMException:

Visual Basic 6.0	Visual Basic 2005
Err.Number	ComException.ErrorCode
Err.Description	COMException.Message
Err.Source	COMException.Source

The following example converts legacy unstructured error handling to structured exception handling:

Visual Basic 6.0 Client

```
On Error Resume Next
sivProperties("Name").Value = "TryToChangeNameProperty"
If Err.Number <> 0 Then
   Dim lErrNumber As Integer
   If InStr(Err.Source, "Solomon") <> 0 Then
       lErrNumber = Err.Number - vbObjectError
   ElseIf Err.Number - vbObjectError > 0 Then
       lErrNumber = Err.Number - vbObjectError
   Else
       lErrNumber = Err.Number
   End If

   Call MsgBox("Error number #" + Str(lErrNumber) + ": " + Err.Description,
MsgBoxStyle.Exclamation, Err.Source)

End If
```

Visual Basic 2005 Client

```
Try
    sivProperties("Name").Value = "TryToChangeNameProperty"
Catch ex As System.Runtime.InteropServices.COMException
    Dim lErrNumber As Integer
    If ex.Source.Contains("Solomon") = True Or ex.ErrorCode - vbObjectError > 0
Then
        lErrNumber = ex.ErrorCode - vbObjectError
    Else
        lErrNumber = ex.ErrorCode
    End If

    Select Case lErrNumber
       ' Check for specific error messages here

       ' Default message display
       Case Else
           MsgBox(ex.Message, MsgBoxStyle.Exclamation, String.Format("{0} Error
{1}", ex.Source, lErrNumber.ToString()))
    End Select

Catch ex As Exception
    MsgBox(ex.Message, MsgBoxStyle.Exclamation, String.Format("{0} Exception",
ex.Source))
End Try
```

Support for WinForm Controls

The Microsoft Dynamics SL Object Model supports WinForm Label and Button controls, which take the place of the Visual Basic Label and CommandButton controls that were available in earlier releases.

Label Properties

The following table lists Label properties that were available in earlier releases of the Microsoft Dynamics SL Object Model. It indicates whether or not the property is supported in the current release and lists the .NET equivalent to it, if applicable.

For more information, see the MSDN article, "Label Control Changes in Visual Basic .NET."

Label property name	Supported/not supported in Visual Basic 2005
Alignment	Supported; maps to TextAlign, ContentAlignment enum
Appearance	Not supported
AutoSize	Supported
BackColor	Supported
BackStyle	Not supported
BorderStyle	Supported; maps to BorderStyle enum
Caption	Supported; maps to Text
ControlType	Supported
DataField	Not supported
DataSource	Not supported
DragIcon	Not supported
DragMode	Not supported
Enabled	Supported
Font	Supported; use SIVFont instead of IFontDisp
ForeColor	Supported
Height	Supported
Index	Not supported
Left	Supported
LinkItem	Not supported
LinkMode	Not supported
LinkTimeout	Not supported
LinkTopic	Not supported
MouseIcon	Not supported
MousePointer	Supported; maps to Cursors class
Name	Supported
OleDragMode	Not supported
OleDropMode	Not supported
RightToLeft	Supported; maps to RightToLeft enum
TabIndex	Supported
Tag	Supported

Label property name	Supported/not supported in Visual Basic 2005
ToolTipText	Not supported
Top	Supported
UseMnemonic	Supported
Visible	Supported
WhatsThisHelpID	Not supported
Width	Supported
WordWrap	Not supported

CommandButton Properties

The following table lists CommandButton properties that were available in earlier releases of the Microsoft Dynamics SL Object Model. It indicates whether or not the property is supported in the current release and lists the .NET equivalent to it, if applicable.

Note: CommandButton maps to Button in the current of Microsoft Dynamics SL release.

For more information, see the MSDN article, "CommandButton Control Changes in Visual Basic.NET."

CommandButton property name	Supported/not supported in Visual Basic 2005
Appearance	Not supported
BackColor	Supported
Cancel	Supported; maps to Form.CancelButton
Caption	Supported; maps to Text
ControlType	Supported
Default	Supported; maps to Form.AcceptButton
DisabledPicture	Not supported
DownPicture	Not supported
DragIcon	Not supported
DragMode	Not supported
Enabled	Supported
Font	Supported; use SIVFont instead of IFontDisp
Height	Supported
HelpContextID	Not supported
Index	Not supported
Left	Supported
MaskColor	Not supported
MouseIcon	Not supported
MousePointer	Supported; maps to Cursors class
Name	Supported
OleDropMode	Not supported
Picture	Not supported
RightToLeft	Supported; maps to RightToLeft enum

CommandButton property name	Supported/not supported in Visual Basic 2005
Style	Not supported
TabIndex	Supported
TabStop	Supported
Tag	Supported
ToolTipText	Not supported
Top	Supported
UseMaskColor	Not supported
Value	Supported; setting True maps to action raising Click handler
Visible	Supported
WhatsThisHelpID	Not supported
Width	Supported

Support for Remaining WinForm Controls

The following table lists "standard controls" that were supported in Visual Basic 6.x and their Visual Basic .NET (WinForm) equivalents. For more information, see the MSDN article, "Differences Between Visual Basic 6.0 and .NET Controls."

Visual Basic 6.0 Standard Controls	Equivalent .NET WinForm Controls
CheckBox	CheckBox
ComboBox	ComboBox
CommandButton	Button
Data	N/A Data binding is completely different in Visual Basic .NET.
DirListBox	N/A Replaced by OpenFileDialog control. During conversion, it will be replaced by VB6.DirListBox for compatibility.
DriveListBox	N/A Replaced by OpenFileDialog control. Replaced by VB6.DriveListBox during conversion for compatibility.
FileListBox	N/A Replaced by OpenFileDialog control. During conversion,it will be replaced by VB6.FileListBox for compatibility.
Form	Form
Frame	GroupBox
HScrollBar	HScrollBar
Image	N/A Use the PictureBox control to display a single image.
ImageList	ImageList
Label	Label

Visual Basic 6.0 Standard Controls	Equivalent .NET WinForm Controls
Line	N/A You will now use a class in the .NET CLR to draw a line.
ListBox	ListBox
OLE	N/A
OptionButton	RadioButton
Picture	PictureBox
Shape	N/A You will now use a class in the .NET Common Language Runtime (CLR) to draw shapes.
TextBox	TextBox
VscrollBar	VScrollBar

The new list of standard controls available to .NET-connected applications is much larger than the finite list available to Visual Basic 6.0 applications. Although not explicitly documented as supported, the remaining WinForm controls will be contained in the SIVControls collection. As with Label and Button, .NET reflection is used to obtain the list of properties found in the SIVProperties collection, owned by each SIVControl instance. In general, simple properties such as Top, Left, Enabled, Visible, and Tag should all work in the same fashion as they did in Visual Basic 6.0. New properties added for Visual Studio 2005 may or may not work, depending on the complexity of the property, and should be used at the risk of the developer.

Listed below are the property types that are supported by the Microsoft Dynamics SL Object Model. Enum properties for .NET will be supported if the underlying type of the enum is one of the simple types.

Property types supported by Microsoft Dynamics SL

System.Boolean

System.Byte

System.Char

System.DateTime

System.Decimal

System.Double

System.Drawing.Color — Maps to OLE color (see description below)

System.Drawing.Font — Maps to SIVFont class (see description below)

System.Int16

System.Int32

System.Int64

System.SByte

System.Single

System.String

System.UInt16

System.UInt32

System.UInt64

Caption Support

If a control supports the Text property, Caption will be added to the list of properties and mapped to Text by the property get/set logic in order to maintain backward compatibility.

Tag Support

The Tag property will be supported, even though it is an Object type. Most legacy code will set Tag to simple types, so in most cases, getting and setting the Tag property should continue to work.

Color Support

Client code will not pass System.Drawing.Color values to the Microsoft Dynamics SL Object Model. The Object Model will translate System.Drawing.Color to and from the numeric equivalent used in COM programming. Current Object Model clients should already be using numeric values to set and get color values.

Font support

Client code will not pass System.Drawing.Font values to the Microsoft Dynamics SL Object Model. Font properties will be represented as an SIVFont class. This contains an equivalent interface to the StdFont class, or the IFontDisp interface, from stdole.tlb. Currently, the Microsoft Dynamics SL Object Model client code could retrieve this interface via the Font property and then set the following individual properties:

- Name
- Size
- Bold
- Italic
- Underline
- Strikethrough
- Weight (not supported in .NET)
- Charset

Example

```
' set the current control's FontBold to True
Dim fontProperty As SWIMAPI.SIVProperty
Dim fontObj As stdole.IFontDisp
Set fontProperty = currentCtrl.Properties.Item("Font")
Set fontObj = fontProperty.Value
fontObj.Bold = True
```

In the same fashion, Microsoft Dynamics SL Object Model 7.0 code can retrieve the Font property directly, but in this case, the instance is of type Microsoft.Dynamics.SL.SIVFont. This instance can used to set any of the properties listed above.

Example

```
' set the current control's FontBold to True (longhand code)
Dim currentCtrl As SIVControl
currentCtrl = sivMyApp.Controls("cAutoRefNbr")
Dim fontProperty As Microsoft.Dynamics.SL.ObjectModel.SIVProperty
```

```
Dim fontObj As Microsoft.Dynamics.SL.ObjectModel.SIVFont
fontProperty = currentCtrl.Properties.Item("Font")
fontObj = fontProperty.Value
fontObj.Bold = True

' set the current control's FontBold to True (shorthand code)
sivMyApp.Controls("cAutoRefNbr").Properties.Item("Font").Value.Bold = True
```

Support for COM Controls

The logic for determining the list of SIVProperties for COM (Active-X) controls remains unchanged in the current Microsoft Dynamics SL Object Model. The property list is constructed by iterating standard COM interfaces. See “Label Properties” and “CommandButton Properties” for more information.

Custom Object Support

Two primary scenarios will occur when you are using Custom Objects:

- Scenario 1 — Developer-defined COM object embedded in application

  Under Visual Basic 6.0, the application must be built as an ActiveX EXE project. The class that the application wishes to expose as a custom object must be marked *PublicNotCreateable*.

  When this application is converted to Visual Basic 2005, the class is converted to a .NET class. ExposeCustomObject cannot be called on an instance of this class unless the CLR thinks it is a COM class. Placing the ComVisible(True) attribute on the class declaration will suffice to keep everything working as before.

Sample

clsVBTObjectModelApp.cls.vb

```
Option Strict Off
Option Explicit On

Imports System.Runtime.InteropServices

<ComVisible(True)> _
Public Class clsVBTObjectModelApp

Private m_Status As Short

Public Event Alert()

Public Sub SignalDone()
RaiseEvent Alert()
End Sub

Public Property ObjStatus() As Short
Get
ObjStatus = m_Status
End Get
Set(ByVal Value As Short)
```

```
m_Status = Value
End Set
End Property

Public Sub New()
MyBase.New()
End Sub

End Class
```

Declaration, and instantiation, of class in developer code

```
Private WithEvents m_CustomObject As New clsVBTObjectModelApp
```

Call to ExposeCustomObject

```
Call ExposeCustomObject(m_CustomObject)
```

- Scenario 2 — Call ExposeCustomObject on an ActiveX control that is contained in the project

  The application calls ExposeCustomObject passing an ActiveX control that is contained within application program. This scenario will continue to work as is, and will require no change by the developer of the application.

Glossary of Terms

automation client

A Windows application that uses objects provided by an Automation server. This documentation tells you how to write Automation clients that use the Microsoft Dynamics SL Object Model as an Automation Server.

automation server

A Windows application that provides objects for other applications (known as Automation clients) to manipulate. This documentation tells you how to write Automation clients that use the Microsoft Dynamics SL Object Model as an Automation server.

base currency

The currency into which all transactions are translated for purposes of financial statements, reporting, and closings. Every Microsoft Dynamics SL installation specifies an unchangeable base currency.

Basic Script Language

Microsoft Dynamics SL's Visual Basic-like internal scripting language; also referred to as *BSL*.

company ID

One of several possible entities for which a particular Microsoft Dynamics SL installation maintains accounting information. The company ID is required as part of the Microsoft Dynamics SL login context.

component

A program that provides objects designed according to the COM specification. The Microsoft Dynamics SL Object Model includes a toolbar and an application *component*.

component object model

A general standard for implementing objects in a computing environment; also referred to as *COM*.

customization level

The type of customization that will run for Microsoft Dynamics SL applications under a particular session of the Microsoft Dynamics SL toolbar. Customization level types include, Standard (no customizations), All Users, One User, and Self.

data entity

A string that a Microsoft Dynamics SL application uses to refer to one of the record sets that it provides to the user. You need to use this string to perform certain operations on the data, such as navigation. See *data level*.

data level

An integer value that refers to what is known in the Microsoft Dynamics SL Object Model as data entity. This integer value is not available through the Microsoft Dynamics SL Object Model. Instead, you must use the string referring to the data entity.

display currency

The currency in which the current transaction on a Microsoft Dynamics SL screen is displayed, regardless of whether or not that currency is the current transaction currency or not. The interactive user may choose to toggle the display of all amounts on a screen between the current transaction currency and the system's base currency. For example, if a sales order is in Polish Zlotys (which would mean that the transaction currency for this sales order is Zlotys), but the system's base currency is in

U.S. dollars, the user could choose to view the screen either in transaction currency (Zlotys) or base currency (dollars).

entity

See *data entity*.

interactive development environment

In Visual Basic development, this term refers to the desktop environment that a programmer uses to work with programs; also referred to as *IDE*.

Initialize mode

A state of the toolbar that allows users with the proper rights to open applications and make changes to fields that are normally for display only.

instance

An object that has been created from a class type and that is pointed to by an object variable of that type. In Microsoft Dynamics SL Object Model programming, you use instances of **SIVApplication** and **SIVToolbar** objects. See *instantiate*.

instantiate

The act of initializing an instance of an object variable, such as an **SIVToolbar** or **SIVApplication** object.

interactive user

The end user of a Microsoft Dynamics SL application, as opposed to the automation client.

level

See *data entity*, *data level*, and *customization level*.

login context

Necessary Microsoft Dynamics SL-specific information about Microsoft Dynamics SL workstation session that is being run by the currently logged-on interactive user or automation client. Login context includes information about the system database, the system database server, the company ID, the user ID, and the password. This information must be furnished by the interactive user on the login screen or by the automation client when calling the **Login** method of the **SIVToolbar** object.

Microsoft Dynamics SL Software Development Kit

A set of software development tools that supplement Visual Basic and allow programmers to develop Microsoft Dynamics SL executables; also referred to as *Microsoft SL SDK*.

Microsoft SL SDK application

An application written with the Microsoft SL SDK. All Microsoft Dynamics SL executables are Microsoft SL SDK applications. The Microsoft Dynamics SL Object Model's main purpose is to allow you to manipulate Microsoft SL SDK applications. A Microsoft SL SDK application has automatic awareness of the toolbar and of Microsoft Dynamics SL API routines.

Microsoft SL SDK developer

A programmer who uses the Microsoft SL SDK library to create Microsoft Dynamics SL applications. The Microsoft SL SDK developer is not the main audience for this documentation, but rather the programmer who wants to use non-Microsoft Dynamics SL tools to manipulate Microsoft Dynamics SL applications.

object model

The set of all members (properties, events, and methods) of a COM component, including information about their behavior and relationship to each other. In the Microsoft Dynamics SL Object Model, this includes information about the **SIVApplication** and **SIVToolbar** objects and their members.

re-entrant

Object that allows the client programmer to call one of its methods or properties from within one of its event procedures. The Microsoft Dynamics SL Object Model is not re-entrant. For instance, it is not possible to manipulate the **SIVApplication** object in the event procedure of its **Message** event (you receive Microsoft Dynamics SL error 7554 if you attempt to do so).

system database

One of the databases of a Microsoft Dynamics SL installation. It contains administrative and system information about the installation. The system database name is required as part of the Microsoft Dynamics SL login context.

third-party control

A control in a Microsoft Dynamics SL application that is not a standard Microsoft Dynamics SL control; a control whose name begins with the prefix *SAF*.

transaction

A set of operations on data that stand or fall as a single unit. Transactions are defined for purposes of referential integrity in a multi-user environment.

transaction currency

The currency ID of the monetary amounts of a given transaction on a Microsoft Dynamics SL application screen. The transaction currency does not have to be the same as the system's base currency, but may be in some other currency permitted by the Microsoft Dynamics SL system's Currency Manager module.

Visual Basic for Applications

A standard dialect of Visual Basic that is used within many Windows applications for scripting tasks; also referred to as *VBA*.

Index

A

Application Browser 147

B

Button control 3

C

Cancel method 5
Collections
- controls 6
- Properties 64
- SIVControls 101
- SIVProperties 103

Constants 7
Controls
- Button 3
- DSLDate 16
- Label 38
- SAFCheck 68
- SAFCombo 70
- SAFFloat 73
- SAFGrid 75
- SAFInteger 76
- SAFMaskedText 78
- SAFOption 80
- StatusBar 118

Controls collection 6
ControlType property 9
Count property 11

D

Delete method 12, 15
DisableObjectModel statements 14, 20
DSLDate control 16

E

EventLog property 18
Events
- Message 50
- SubFormDisplay 119

Exercises
- Browsing the sample application 149
- integrating Microsoft Dynamics SL with Microsoft Office 127

ExposeCustomObject statement 21

F

First method 22

G

GetBusinessDate method 25
GetCurrencyIDs method 27
GetCurrencyView method 28
GetCustomizationLevel method 29
GetCustomObject method 31
GetEntityStatus method 32
GetStatusBarText method 34

I

Ill-behaved controls 133
InitializeMode property 35
Item property 36

K

KeyControls Collection 37

L

Label control 38
Last method 40
Level property 42
LevelNumber Property 44
Login method 45
Logout method 48

M

Message Event 50
Methods
- Cancel 5
- Delete 12, 15
- First 22
- GetBusinessDate 25
- GetCurrencyIDs 27
- GetCurrencyView 28
- GetCustomizationLevel 29
- GetCustomObject 31
- GetEntityStatus 32
- GetStatusBarText 34
- Last 40
- Login 45
- Logout 48
- New 57
- Next 59

Previous 62
Quit 65
Save 83
SetBusinessDate 86
SetCurrencyIDs 88
SetCurrencyView 90
SetCustomizationLevel 92
SetStatusBarText 95
StartApplication 113, 116
Microsoft Dynamics SL Object Model
integrating with Microsoft Office 127
limitations 131
Microsoft Office
integrating with Microsoft Dynamics SL 127

N

Name property (SIVControl Object) 55
Name property (SIVProperty Object) 56
New method 57
Next method 59
Notes/Attachments Icon (not supported) 61

O

Objects
SIVApplication 98
SIVControl 99
SIVProperty 105
SIVToolbar 109

P

Previous method 62
Properties
ControlType 9
Count 11
EventLog 18
InitializeMode 35
Item 36
Level 42
Name (SIVControl Object) 55
Name (SIVProperty Object) 56
Value (SIVControl Object) 120
Value (SIVProperty Object) 123
Visible 125
Properties collection 64

Q

Quit method 65

R

Relative Date Dialog (not supported) 67

S

SAFCheck control 68
SAFCombo control 70
SAFContainer control (not available) 72
SAFFloat control 73
SAFGrid control 75
SAFInteger control 76
SAFMaskedText control 78
SAFOption control 80
SAFUpdate control (not available) 82
Save method 83
SetBusinessDate method 86
SetCurrencyIDs method 88
SetCurrencyView method 90
SetCustomizationLevel method 92
SetStatusBarText method 95
SIVApplication Object 98
SIVControl Object 99
SIVControls collection 101
SIVProperties collection 103
SIVProperty Object 105
SIVToolbar Object 109
StartAppAndAutomate function 111
StartApplication method 113, 116
Statements
DisableObjectModel 14, 20
ExposeCustomObject 21
StatusBar control 118
SubFormDisplay Event 119

V

Value property (SIVControl Object) 120
Value property (SIVProperty Object) 123
Visible property 125
Visual Basic
prerequisite knowledge 1