Visual Basic.NET Introduction

ROAD MAP

- Introduction
- Programming Word
- Spellchecking Automation of Documents
- Programming Excel
- Using the Worksheets Collection and Worksheet Object
- Range Object
- Math Parser Automation
- Programming Outlook
- Retrieve Emails, Automatic Replying Emails

INTRODUCTION

- object model
  - a collection of classes that represents the objects, or entities, each application can handle, the properties that determine the characteristics of these entities, and the methods that act on them
  - E.g Word
- The object models of the Office applications aren’t part of Visual Studio

Programming Word

- Word
  - a top-notch word processor
  - tap into the power of Word through the object model it exposes
  - create documents, format them, and store them as DOC files
  - print these files, or e-mail them as attachments to a list of addresses
  - reuse Word's spell-checking capabilities to correct documents at runtime
- Microsoft Word provides numerous objects
- The top-level Word object is the Application object
  - represents the current instance of the application

Programming Word

- To use the object model of Microsoft Word in your VB project
  - add a reference to the Microsoft Word application to the project
- To program against Word's object
  - create a variable that represents the application

```
Dim WordApp As New Word.Application
```

- Under the Application object is the Documents collection
  - contains a Document object for each open document. Using an object variable of the Document type
  - open an existing document or create a new document

```
doc1 = WordApp.Documents.Open("My Word Samples.doc")
```

Programming Word

- To create a new document
  - Add method of the Documents collection

```
Dim newDoc As New Word.Document
newDoc = WordApp.Documents.Add
newDoc.SaveAs("My new Document")
```

- If the document has been saved already
  - The Save Method
  - Three arguments are optional

```
Documents.Close(saveChanges, originalFormat, routeDocument)
```

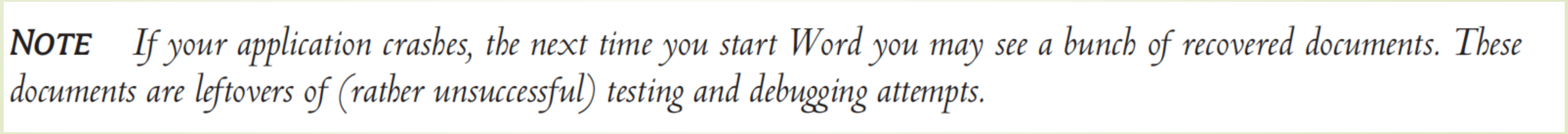
Programming Word

- If you have multiple open documents at once
  - select the active document with the ActiveDocument property

```
doc1 = WordApp.ActiveDocument
```

- After you're done processing the document
  - the SaveAs method

```
Documents.SaveAs(fileName)
```



NOTE *If your application crashes, the next time you start Word you may see a bunch of recovered documents. These documents are leftovers of (rather unsuccessful) testing and debugging attempts.*

Programming Word

- To terminate the application
  - Quit method

```
Private Sub Form1_Closing(ByVal sender As Object, _
                ByVal e As System.ComponentModel.CancelEventArgs) _
                Handles MyBase.Closing
   WordApp.Documents.Close(Word.WdSaveOptions.wdDoNotSaveChanges)
   WordApp.Quit()
   WordApp = Nothing
End Sub
```

Spell-Checking Documents

- ProofreadingErrors collection
  - a property of the Range object
  - contains the misspelled words in the Range
  - Call SpellingErrors method
  - *DRange* is Range object (a paragraph or an entire document)

```
Dim SpellCollection As ProofreadingErrors
Set SpellCollection = DRange.SpellingErrors
```

- SpellingSuggestions collection
  - To retrieve the list of alternate words
  - a method of the Application object, not of the Range object you’re spell-checking

```
Dim CorrectionsCollection As SpellingSuggestions
Set CorrectionsCollection = AppWord.GetSpellingSuggestions("antroid")
```

Programming Excel

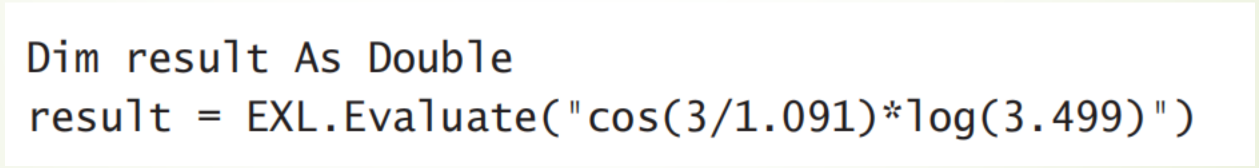
- Add a reference to the Microsoft Excel 9.0 Object Library item
- Select Add Reference, and double-click the name of the Excel library
- Add an instance of Excel's object model to your application
- Declare a variable of the Excel.Application type

```
Dim EXL As New Excel.Application
```

- *EXL*
  - a new instance of Excel

Programming Excel

- To access Excel’s functionality
  - use a hierarchy of objects
- Two important methods
  - the Calculate method: recalculates all open worksheets
  - the Evaluate method: evaluates math expressions and returns the result



```
Dim result As Double
result = EXL.Evaluate("cos(3/1.091)*log(3.499)")
```

The Worksheets Collection and the Worksheet Object

- Each workbook in Excel contains one or more worksheets
- Worksheets collection
  - contains a Worksheet object for each worksheet in the current workbook

```
Application.Worksheets.Add(before, after, count, type)
```

The Worksheets Collection and the Worksheet Object

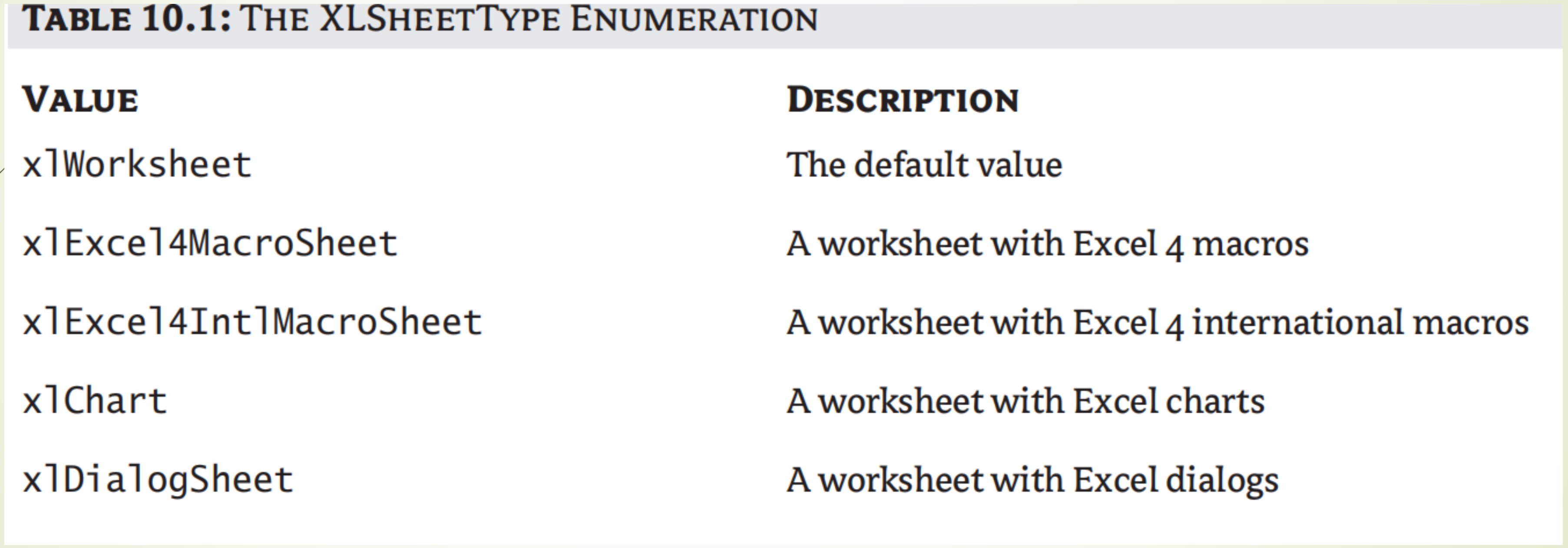


TABLE 10.1: THE XLSHEETTYPE ENUMERATION

VALUE	**DESCRIPTION**
`xlWorksheet`	The default value
`xlExcel4MacroSheet`	A worksheet with Excel 4 macros
`xlExcel4IntlMacroSheet`	A worksheet with Excel 4 international macros
`xlChart`	A worksheet with Excel charts
`xlDialogSheet`	A worksheet with Excel dialogs

The Worksheets Collection and the Worksheet Object

- To create a new worksheet
  - declare a variable of the Worksheet type
  - add it to the Worksheets collection

```
Dim WSheet As New Excel.Worksheet()
WSheet = EXL.Workbooks.Add.Worksheets.Add
```

- Access the worksheets from within your code and populate them

```
WSheet = EXL.Workbooks.Item(1).Worksheets("Sheet2")
WSheet.Cells(1, 1) = "TOP LEFT CELL"
```

The Worksheets Collection and the Worksheet Object

- Add worksheets to the current Workbook with the Add method of the Worksheets collection

```
WSheet = EXL.ActiveWorkbook.Worksheets.Add()
WSheet.Cells(1, 1) = "First Cell"
```

- To access an individual worksheet
  - Worksheet collection's Item method

```
Application.Worksheets.Item(2)
Application.Worksheets.Item("SalesData")
```

Range Object

- The basic object for accessing the contents of a worksheet

- Basic Syntax

```
Worksheet.Range(cell1:cell2)
```

- The Range property is an object

```
range1 = Worksheet.Range(4, 8)
range2 = range1.Range(3, 5)
```

Range Object

- To create a Selection object

```
Range("A2:D2").Select
```

- To change the appearance of the selection

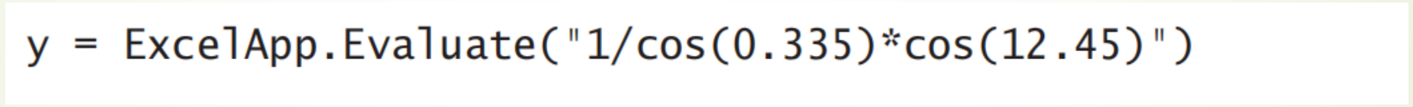
```
Selection.Font.Bold = True
Selection.Font.Size = 13
```

- Combine two different ranges with the Union method and assign them to a new Range object

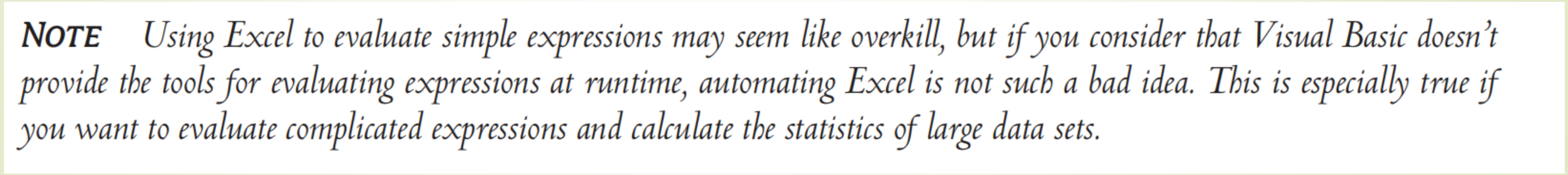
```
Set Titles = Worksheet.Range ("A1:A10")
Set Totals = Worksheet.Range ("A100:A110"
Set CommonRange = Union(Titles, Totals)
CommonRange.Font.Bold = True
```

Using Excel as a Math Parser

- To evaluate complicated math expressions
- Example:
  - the Evaluate method: Excel.Application object
  - initialized the *ExcelApp* object variable

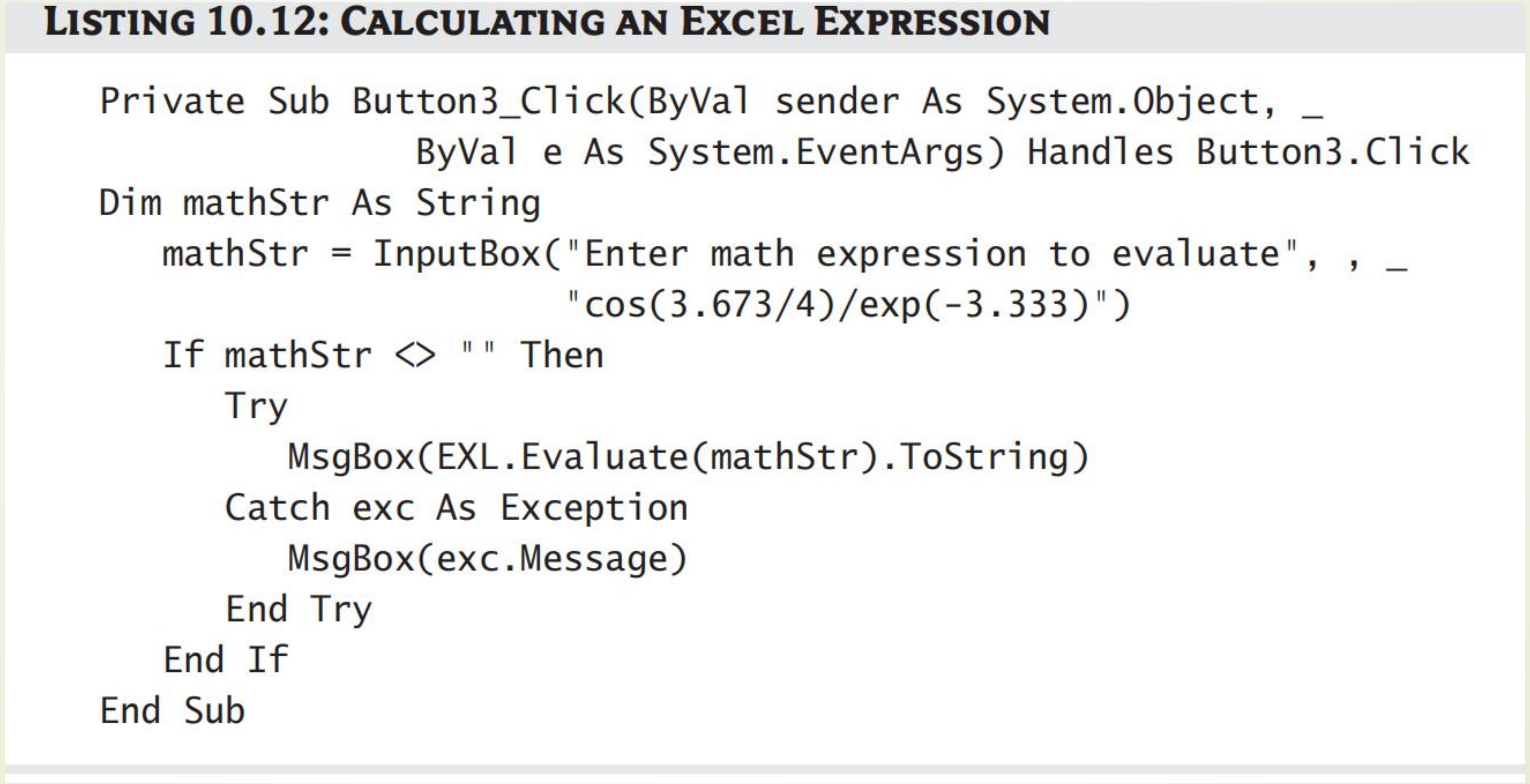


```
y = ExcelApp.Evaluate("1/cos(0.335)*cos(12.45)")
```



NOTE *Using Excel to evaluate simple expressions may seem like overkill, but if you consider that Visual Basic doesn't provide the tools for evaluating expressions at runtime, automating Excel is not such a bad idea. This is especially true if you want to evaluate complicated expressions and calculate the statistics of large data sets.*

Using Excel as a Math Parser



LISTING 10.12: CALCULATING AN EXCEL EXPRESSION

```
Private Sub Button3_Click(ByVal sender As System.Object, _
               ByVal e As System.EventArgs) Handles Button3.Click
Dim mathStr As String
   mathStr = InputBox("Enter math expression to evaluate", , _
                      "cos(3.673/4)/exp(-3.333)")
   If mathStr <> "" Then
      Try
         MsgBox(EXL.Evaluate(mathStr).ToString)
      Catch exc As Exception
         MsgBox(exc.Message)
      End Try
   End If
End Sub
```

Programming Outlook

- To make your applications e-mail–aware
- to mail-enable your Visual Basic applications by manipulating the object model of Outlook

- Outlook
  - maintains a list of contacts organized in folders
  - Eg. appointment scheduling and routing e-mail

Programming Look

- To contact Outlook and program the objects it exposes

```
Dim OLApp As New Outlook.Application
```

- Outlook
  - doesn't expose a single object like a Document or Worksheet that gives you access to the information it can handle
  - contains several objects including mail messages, contacts, and tasks

Programming Look

- To access the folder objects
  - create a MAPI message store

```
Dim OLApp As New Outlook.Application()
Dim OLNameSpace As Outlook.Namespace
OLNameSpace = OLApp.GetNamespace("MAPI")
```

- The various folders maintained by Outlook can be accessed by:

```
olFolderCalendar            olFolderInbox              olFolderOutbox

olFolderContacts            olFolderJournal            olFolderSentMail

olFolderDeletedItems        olFolderNotes              olFolderTask

olFolderDrafts
```

Programming Outlook

- To retrieve all the items in the Contacts folder

```
Set allContacts = OLNameSpace.GetDefaultFolder(olFolderContacts).Items
```

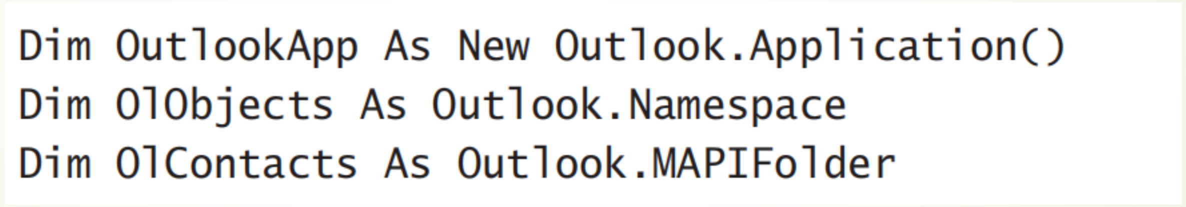
- Each folder contains different types of information
  - Contacts folder: ContactItem objects, the Inbox
  - Outbox folders: MailItem objects
  - Calendar folder: a collection of AppointmentItem objects.

Retrieving Information

- Outlook stores different types of information in different folders
- Outlook's folders do not correspond to physical folders on the disk
  - just the basic organizational units of Outlook
- Contact information
  - is stored in the Contacts folder
- incoming messages
  - are stored in the Inbox folder

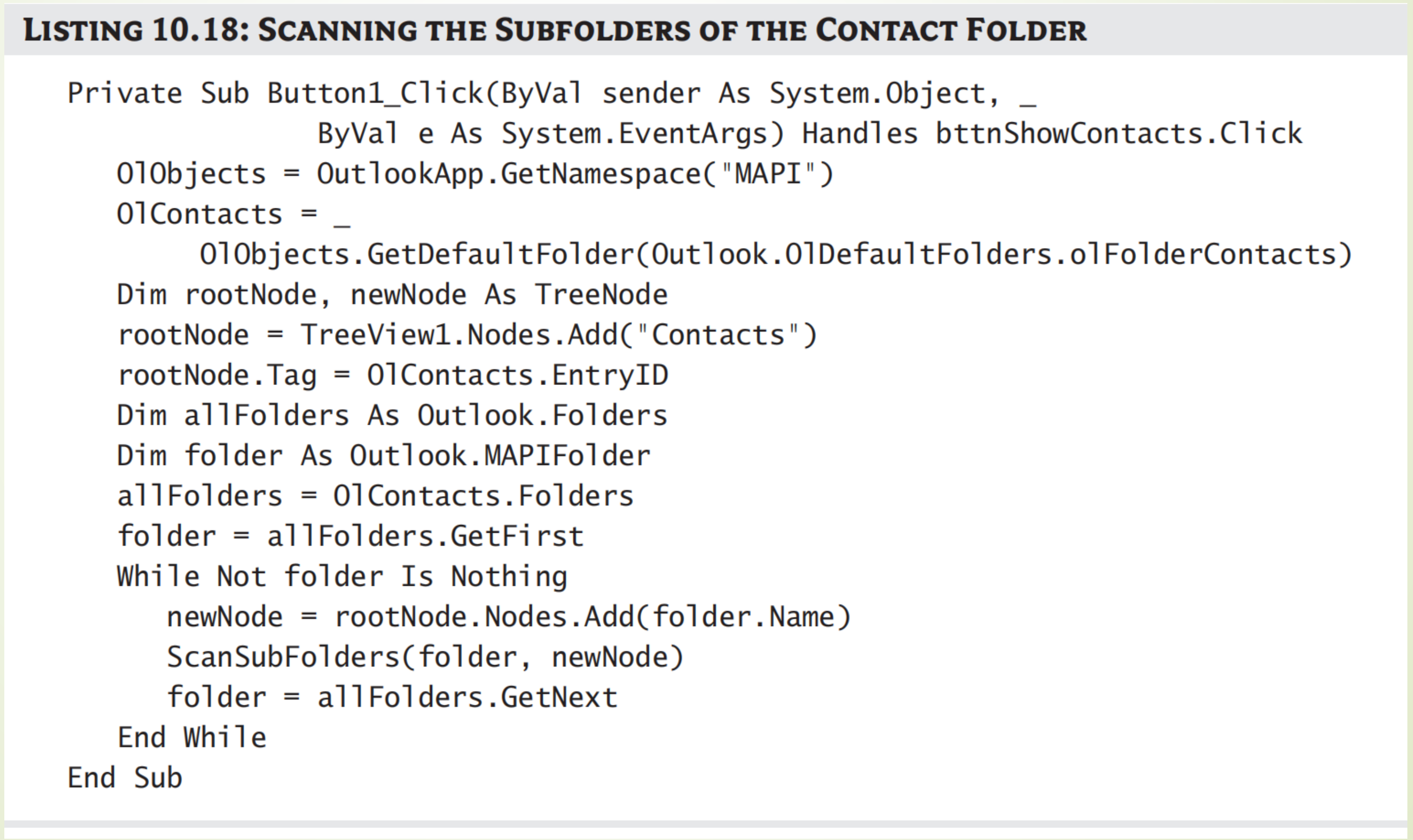
Recursive Scanning of the Contacts Folder

- Most users organize their contacts in subfolders to better classify them and simplify searching
- *recursive programming*



```
Dim OutlookApp As New Outlook.Application()
Dim OlObjects As Outlook.Namespace
Dim OlContacts As Outlook.MAPIFolder
```

Recursive Scanning of the Contacts Folder

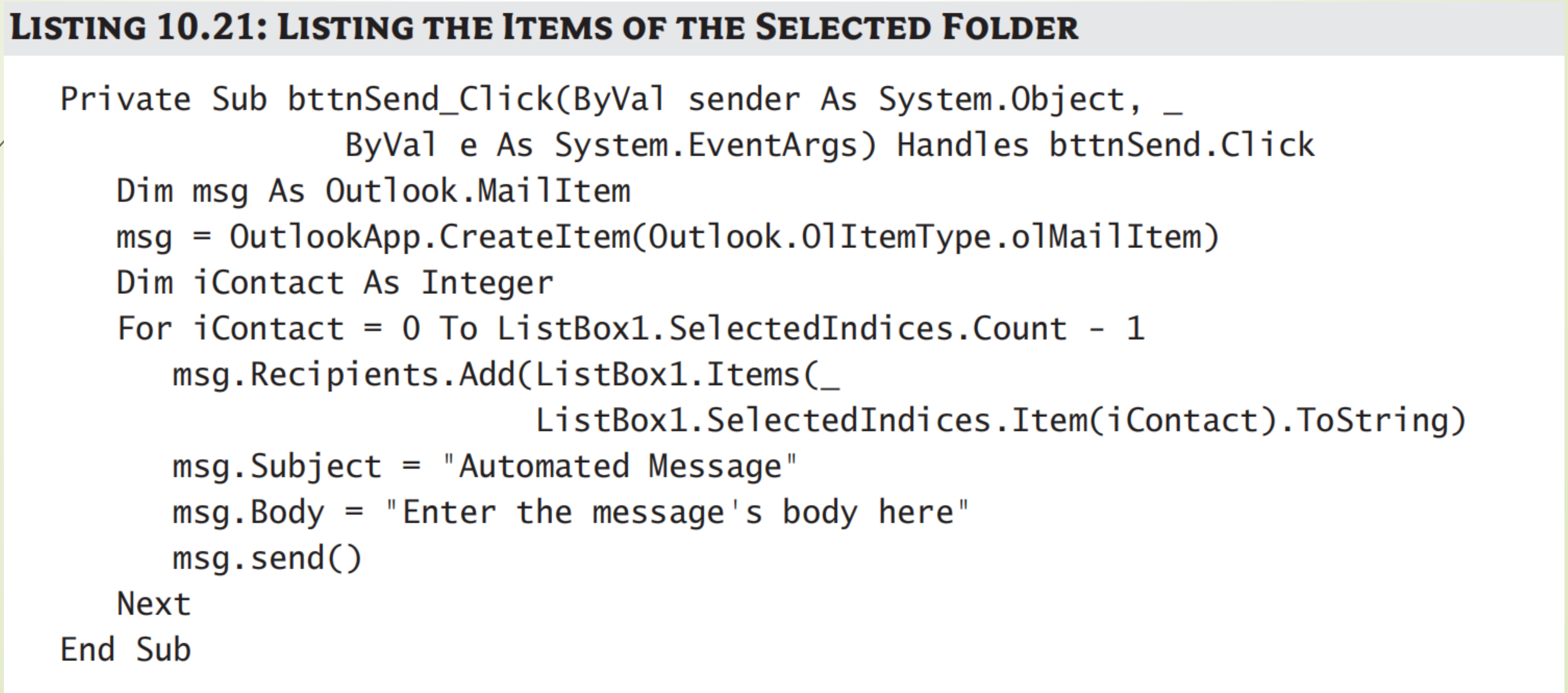


LISTING 10.18: SCANNING THE SUBFOLDERS OF THE CONTACT FOLDER

```
Private Sub Button1_Click(ByVal sender As System.Object, _
               ByVal e As System.EventArgs) Handles bttnShowContacts.Click
   OlObjects = OutlookApp.GetNamespace("MAPI")
   OlContacts = _
        OlObjects.GetDefaultFolder(Outlook.OlDefaultFolders.olFolderContacts)
   Dim rootNode, newNode As TreeNode
   rootNode = TreeView1.Nodes.Add("Contacts")
   rootNode.Tag = OlContacts.EntryID
   Dim allFolders As Outlook.Folders
   Dim folder As Outlook.MAPIFolder
   allFolders = OlContacts.Folders
   folder = allFolders.GetFirst
   While Not folder Is Nothing
      newNode = rootNode.Nodes.Add(folder.Name)
      ScanSubFolders(folder, newNode)
      folder = allFolders.GetNext
   End While
End Sub
```

Automated Messages

- The message’s subject and body are hard-coded in this project



LISTING 10.21: LISTING THE ITEMS OF THE SELECTED FOLDER

```
Private Sub bttnSend_Click(ByVal sender As System.Object, _
               ByVal e As System.EventArgs) Handles bttnSend.Click
   Dim msg As Outlook.MailItem
   msg = OutlookApp.CreateItem(Outlook.OlItemType.olMailItem)
   Dim iContact As Integer
   For iContact = 0 To ListBox1.SelectedIndices.Count - 1
      msg.Recipients.Add(ListBox1.Items(_
                         ListBox1.SelectedIndices.Item(iContact).ToString)
      msg.Subject = "Automated Message"
      msg.Body = "Enter the message's body here"
      msg.send()
   Next
End Sub
```

Exercise

- (1) What does “xlWorksheet” mean in XLSheetType enumeration?
- (2) If you have multiple open documents at once， what property should you use?
- (3) What method should we use to create new document?