Microsoft Visual Basic 2005: Reloaded
Second Edition

Chapter 7
Sub and Function Procedures
Objectives

After studying this chapter, you should be able to:

• Explain the difference between a Sub procedure and a Function procedure
• Create a Sub procedure
• Create a procedure that receives information passed to it
• Explain the difference between passing data by value and passing data by reference
Objectives (continued)

• Explain the purpose of the **sender** and **e** parameters
• Associate a procedure with more than one object and event
• Create a Function procedure
• Convert an object to a different type using the `TryCast` keyword
Procedures

- **Procedure**: a block of program code that performs a specific task
- Two types of procedures:
  - **Function procedure**: returns a value after performing its task
  - **Sub procedure**: does not return a value
Sub Procedures

• **Event procedure:**
  – Sub procedure that is associated with a specific object and event
  – Automatically processed when the associated event occurs

• **Independent Sub procedure:**
  – Collection of code that can be invoked from one or more places in an application
  – Not associated with an event
  – Processed only when called (invoked)
Sub Procedures (continued)

• Independent Sub procedures:
  – Eliminate the duplication of code in different parts of a program
  – Allow a large, complex application to be broken into small and manageable tasks
  – Allow multiple programmers to work on an application simultaneously
Sub Procedures (continued)

**HOW TO...**

**Create an Independent Sub Procedure**

**Syntax**

```vbnet
Private Sub procedurename([parameterlist])
[statements]
End Sub
```

**Example**

```vbnet
Private Sub ClearLabels()
    ' removes the contents of the labels that
    ' display the regular, overtime, and gross pay

(Figure is continued on next page)```
Sub Procedures (continued)

```vbnet
regularLabel.Text = String.Empty
overtimeLabel.Text = String.Empty
grossLabel.Text = String.Empty
End Sub
```

**Steps for entering an independent Sub procedure in the Code Editor window**

1. Open the Code Editor window.
2. Click a blank line in the Code Editor window. The blank line can be anywhere between the `Public Class` clause and the `End Class` clause. However, it must be outside of any other Sub or Function procedure.
3. Type the Sub procedure header and press the Enter key on your keyboard. When you press the Enter key, the Code Editor will automatically enter the Sub procedure footer (`End Sub`) for you.

![Figure 7.1: How to create an independent Sub procedure](image)
Sub Procedures (continued)

• Independent Sub procedures:
  – Have a procedure header and procedure footer
  – Use Pascal case for names
  – Optionally contain a parameter list

• **Parameters:**
  – Declared in the procedure header
  – Store the information passed into the procedure when it is invoked

• **Call statement:** invokes an independent Sub procedure
Sub Procedures (continued)

**HOW TO...**

**Call an Independent Sub Procedure**

**Syntax**

```vbnet
Call procedurename([argumentlist])
```

**Example**

```vbnet
Call ClearLabels()
```

**Figure 7.2** How to call an independent Sub procedure
The Gadis Antiques Application

![Image of the Gadis Antiques Application window showing calculated pay details.](Image)
The Gadis Antiques Application (continued)

Visual Basic code

Private Sub ClearLabels()
    ' clears the labels that display the
    ' regular, overtime, and gross pay
    regularLabel.Text = String.Empty
    overtimeLabel.Text = String.Empty
    grossLabel.Text = String.Empty
End Sub

Private Sub MainForm_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles Me.Load
    ' fills the combo boxes with values
    For hours As Decimal = 30D To 50D Step 0.5D
        hoursComboBox.Items.Add(hours.ToString("N1"))
    Next hours
    For rates As Decimal = 7.75D To 12.5D Step 0.25D
        ratesComboBox.Items.Add(rates.ToString("N2"))
    Next rates
End Sub

Private Sub exitButton_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles exitButton.Click
    Me.Close()
End Sub

Private Sub calcButton_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles calcButton.Click
    ' calculates the regular pay, overtime pay,
    ' and gross pay
    Const Message As String = "The hours and rate entries must be numeric."
    Dim hoursWorked As Decimal
    Dim payRate As Decimal
    Dim regularPay As Decimal

    (Figure is continued on next page)
The Gadis Antiques Application
(continued)

```vbnet
Dim overtimePay As Decimal
Dim grossPay As Decimal
Dim isConvertedHours As Boolean
Dim isConvertedRate As Boolean

' convert input to Decimal
isConvertedHours = _
    Decimal.TryParse(hoursComboBox.Text, hoursWorked)
isConvertedRate = _
    Decimal.TryParse(rateComboBox.Text, payRate)

' if input can be converted to a number,
' perform calculations and display results
' otherwise, clear labels and display message
If isConvertedHours AndAlso isConvertedRate Then
    If hoursWorked <= 40 Then
        ' calculate regular pay only
        regularPay = hoursWorked * payRate
    Else
        ' calculate regular and overtime pay
        regularPay = 40 * payRate
        overtimePay = (hoursWorked - 40) * payRate * 1.5D
    End If

    ' calculate gross pay
    grossPay = regularPay + overtimePay

    ' display calculated results
    regularLabel.Text = regularPay.ToString("N2")
    overtimeLabel.Text = overtimePay.ToString("N2")
    grossLabel.Text = grossPay.ToString("N2")
Else
    MessageBox.Show(Message, "Gadis Antiques", _
        MessageBoxButtons.OK, MessageBoxIcon.Information)
End If

End Sub

Private Sub clearButton_Click(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles clearButton.Click
Call ClearLabels()
End Sub

Private Sub hoursComboBox_TextChanged(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles hoursComboBoxTextChanged
Call ClearLabels()
End Sub

(Figure is continued on next page)
The Gadis Antiques Application (continued)

Private Sub rateComboBox_TextChanged(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles rateComboBox.TextChanged
    Call ClearLabels()
End Sub
End Class

FIGURE 7.4 Code for the Gadis Antiques application
Including Parameters in an Independent Sub Procedure

- **Parameter**: stores data that is passed to the procedure when the procedure is invoked
- When invoking a procedure with parameters, you must pass:
  - The same number of parameters
  - The same type of parameters
  - The parameters in the same order as declared in the procedure
- Can pass a variable, named constant, literal constant, or keyword as parameter
Passing Variables

• Each variable has a value and a unique memory address
• Variable can be passed to a procedure in two ways:
  – *By value*: you pass the variable’s value
  – *By reference*: you pass the variable’s address
• *By value*: the procedure receives only the value and cannot change the actual variable’s value
• *By reference*: the procedure receives the address, and can make changes to the variable’s value
Passing Variables by Value

bullet Use the keyword **ByVal** before the parameter in the procedure declaration
bullet **ByVal** is the default method of passing variables
bullet Procedure cannot change the actual variable’s value
Passing Variables by Value (continued)
Passing Variables by Value (continued)

Visual Basic code

' Project name: Pet Information Project
' Project purpose: The project displays a message
' that contains a pet's name and age.
' Created/revised by: <your name> on <current date>

Option Explicit On
Option Strict On

Public Class MainForm

Private Sub DisplayMessage(ByVal pet As String, ByVal years As String)
' displays the pet information passed to it

messageLabel.Text = "Your pet " & pet & " is " & years & " years old."
End Sub

Private Sub getinfoButton_Click(ByVal sender As Object,
ByVal e As System.EventArgs) Handles getinfoButton.Click
' gets the pet information, then displays the
' information in a message

(Figure is continued on next page)
Passing Variables by Value (continued)

Dim petName As String
Dim petAge As String

petName = InputBox("Pet's name:", "Name Entry")
petAge = InputBox("Pet's age (years):", "Age Entry")

Call DisplayMessage(petName, petAge)
End Sub

Private Sub exitButton_Click(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles exitButton.Click
    Me.Close()
End Sub
End Class

Figure 7.6 Code for the Pet Information application
Passing Variables by Value (continued)

![Bonus Calculator](image)

**Region 1 sales:** 1000

**Region 2 sales:** 2000

**Bonus amount:** $300.00

**Figure 7.7** Sample run of the Bonus Calculator application
Passing Variables by Value (continued)

```vbnet
' Project name: Bonus Calculator Project
' Project purpose: The project calculates a 10% bonus.
' Created/revise by: <your name> on <current date>

Option Explicit On
Option Strict On

Public Class MainForm

Private Sub CalcAndDisplayBonus(ByVal sales1 As Decimal, ByVal sales2 As Decimal, ByVal rate As Decimal)
    ' Calculates and displays a bonus amount based
    ' on the sales amounts and bonus rate passed to it

(Figure is continued on next page)
```
Passing Variables by Value (continued)

```vbnet
Dim bonus As Decimal

bonus = (sales1 + sales2) * rate
bonusLabel.Text = bonus.ToString("C2")
End Sub

Private Sub exitButton_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles exitButton.Click
Me.Close()
End Sub

Private Sub calcButton_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles calcButton.Click
' Calculates a 10% bonus amount

Const BonusRate As Decimal = 0.1D
Dim reg1Sales As Decimal
Dim reg2Sales As Decimal
Dim isConvertedReg1 As Boolean
Dim isConvertedReg2 As Boolean

isConvertedReg1 = _
Decimal.TryParse(reg1TextBox.Text, reg1Sales)
isConvertedReg2 = _
Decimal.TryParse(reg2TextBox.Text, reg2Sales)

If isConvertedReg1 AndAlso isConvertedReg2 Then
    Call CalcAndDisplayBonus(reg1Sales, reg2Sales, BonusRate)
Else
    MessageBox.Show("The sales amounts must be numeric.", _
                   "Bonus Calculator", MessageBoxButtons.OK, _
                   MessageBoxIcon.Information)
End If

reg1TextBox.Focus()
End Sub

Private Sub reg1TextBox_Enter(ByVal sender As Object, ByVal e As System.EventArgs) Handles reg1TextBox.Enter
reg1TextBox.SelectAll()
End Sub

Private Sub reg2TextBox_Enter(ByVal sender As Object, ByVal e As System.EventArgs) Handles reg2TextBox.Enter
reg2TextBox.SelectAll()
End Sub

Private Sub reg1TextBox_TextChanged(ByVal sender As Object, ByVal e As System.EventArgs) Handles reg1TextBox.TextChanged
End Sub

[Figure is continued on next page]
Passing Variables by Value (continued)

```vbnet
bonusLabel.Text = String.Empty
End Sub

Private Sub reg2TextBox_TextChanged(ByVal sender As Object, ByVal e As System.EventArgs) Handles reg2TextBox.TextChanged
    bonusLabel.Text = String.Empty
End Sub

End Class
```

**Figure 7.8** Code for the Bonus Calculator application
Passing Variables by Reference

• Use the keyword `ByRef` before the parameter in the procedure declaration
• Procedure receives the address of the variable, and is able to change the variable’s value
Passing Variables by Reference (continued)

Figure 7.9 Sample run of the Gross Pay application
Passing Variables by Reference (continued)

Visual Basic code

' Project name:               Gross Pay Project
' Project purpose:           The project calculates gross pay.
' Created/revised by:        <your name> on <current date>

Option Explicit On
Option Strict On

Public Class MainForm

    Private Sub CalcGrossPay(ByVal hours As Decimal, _
                               ByVal rate As Decimal, _
                               ByRef gross As Decimal)

(Figure is continued on next page)
Passing Variables by Reference (continued)

' calculates gross pay
If hours <= 40D Then
  gross = hours * rate
Else
  gross = hours * rate + (hours - 40D) * rate / 2D
End If
End Sub

Private Sub MainForm_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles Me.Load
' fills list boxes with values, then ' selects a default item
For hours As Decimal = 0.5D To 50D Step 0.5D
  hoursListBox.Items.Add(hours.ToString)
Next hours
For rates As Decimal = 7.25D To 10.5D Step 0.25D
  ratesListBox.Items.Add(rates.ToString)
Next rates
hoursListBox.SelectedItem = "40.0"
ratesListBox.SelectedIndex = 0
End Sub

Private Sub calcButton_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles calcButton.Click
' calculates and displays a gross pay amount
Dim hoursWkd As Decimal
Dim rateOfPay As Decimal
Dim grossPay As Decimal

hoursWkd = Convert.ToDecimal(hoursListBox.SelectedItem)
rateOfPay = Convert.ToDecimal(ratesListBox.SelectedItem)
Call Calc Gross Pay(hoursWkd, rateOfPay, grossPay)
grossLabel.Text = grossPay.ToString("C2")
End Sub

Private Sub exitButton_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles exitButton.Click
Me.Close()
End Sub

Private Sub hoursListBox_SelectedIndexChanged(ByVal sender As Object, ByVal e As System.EventArgs) _
(Figure is continued on next page)
Passing Variables by Reference (continued)

```vbnet
Handles hoursListBox.SelectedValueChanged
grossLabel.Text = String.Empty
End Sub
Private Sub rateListBox_SelectedValueChanged__
    (ByVal sender As Object, ByVal e As System.EventArgs) __
Handles rateListBox.SelectedValueChanged
    grossLabel.Text = String.Empty
End Sub
End Class
```

**Figure 7.10** Code for the Gross Pay application
Passing Variables by Reference (continued)

<table>
<thead>
<tr>
<th>Memory locations belonging only to the calcButton’s Click event procedure:</th>
</tr>
</thead>
<tbody>
<tr>
<td>hoursWkd</td>
</tr>
<tr>
<td>41.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Memory locations belonging only to the CalcGrossPay procedure:</th>
</tr>
</thead>
<tbody>
<tr>
<td>hours</td>
</tr>
<tr>
<td>41.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Memory location belonging to both procedures:</th>
</tr>
</thead>
<tbody>
<tr>
<td>grossPay (calcButton’s Click event procedure)</td>
</tr>
<tr>
<td>gross (CalcGrossPay procedure)</td>
</tr>
<tr>
<td>332.00</td>
</tr>
</tbody>
</table>

**Figure 7.11** Contents of memory after the CalcGrossPay procedure header and selection structure are processed.
Passing Variables by Reference (continued)

<table>
<thead>
<tr>
<th>hoursWkd</th>
<th>rateOfpay</th>
<th>grossPay</th>
</tr>
</thead>
<tbody>
<tr>
<td>41.0</td>
<td>8.00</td>
<td>332.00</td>
</tr>
</tbody>
</table>

**Figure 7.12** Contents of memory after the appropriate variables and variable name are removed.
Associating a Procedure with Different Objects and Events

• **Handles** keyword:
  – Appears in event procedure header
  – Indicates the object and event associated with the procedure
  – Controls when the procedure is invoked

• By default, the event procedure name matches the name of the associated object and event
Associating a Procedure with Different Objects and Events (continued)

Visual Basic code

' Project name: Gadis Antiques Project
' Project purpose: The project calculates an employee's
'                 regular, overtime, and gross pay
' Created/revised by: <your name> on <current date>

Option Strict On
Option Explicit On

Public Class MainForm

    Private Sub ClearLabels()
        ' clears the labels that display the
        ' regular, overtime, and gross pay

        regularLabel.Text = String.Empty
        overtimeLabel.Text = String.Empty

    (Figure is continued on next page)
Associating a Procedure with Different Objects and Events (continued)

```
grossLabel.Text = String.Empty
End Sub

Private Sub MainForm_Load ...

Private Sub exitButton_Click ...

Private Sub calcButton_Click ...

Private Sub clearButton_Click(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles clearButton.Click
    Call ClearLabels()
End Sub

Private Sub hoursComboBox_TextChanged(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles hoursComboBox.TextChanged
    Call ClearLabels()
End Sub

Private Sub rateComboBox_TextChanged(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles rateComboBox.TextChanged
    Call ClearLabels()
End Sub
```

Figure 7.13 Partial code for the Gadis Antiques application
Associating a Procedure with Different Objects and Events (continued)

• **Event procedure:**
  – Name of event procedure can be changed
  – Can be associated with more than one object and event as long as each event has the same parameters
• Add the additional object/events to the `Handles` clause
• **Sender parameter:** contains the memory address of the object that raised the event
• **e parameter:** contains additional information about the object that raised the event
Associating a Procedure with Different Objects and Events (continued)

```vbnet
Option Strict On
Option Explicit On

Public Class MainForm

Private Sub ClearLabels(ByVal sender As Object, ByVal e As System.EventArgs)
Handles cancelButton.Click, hoursComboBox.TextChanged, rateComboBox.TextChanged
    ' clears the labels that display the regular, overtime, and gross pay
    regularLabel.Text = String.Empty
    overtimeLabel.Text = String.Empty
    grossLabel.Text = String.Empty
End Sub

Private Sub MainForm_Load ...
    '...
End Class
```

**Figure 7.14** A different version of the code for the Gadis Antiques application
Function Procedures

• **Function procedure** (or **Function**):
  – Block of code that performs a specific task
  – Returns a value after completing its task
• Visual Basic contains many built-in functions
• You can create your own functions with or without parameters
Function Procedures (continued)

HOW TO...

Create a Function Procedure

Syntax

Private Function procedurename([parameterrlist]) As datatype

                 [statements]

                    Return expression

End Function

Example

Private Function CalcNew(ByVal price As Decimal) As Decimal

                   ' calculates and returns a new price using the current
                   ' price passed to it and a 5% price increase rate

                    Return price * 1.05D

End Function

Steps for entering a Function procedure in the Code Editor window

1. Open the Code Editor window.
2. Click a blank line in the Code Editor window. The blank line can be anywhere between the Public Class clause and the End Class clause. However, it must be outside of any other Sub or Function procedure.
3. Type the Function procedure header and press the Enter key on your keyboard. When you press the Enter key, the Code Editor will automatically enter the Function procedure footer (End Function) for you.
Function Procedures (continued)

• Function procedure header:
  – As *datatype* clause indicates the type of the return value

• Function procedure footer statement:
  – End Function

• **Return** keyword:
  – Sets the value to be returned by the function
  – Ends the function
The Pine Lodge Application

Figure 7.16: Sample run of the Pine Lodge application
The Pine Lodge Application (continued)

```vbnet
Public Class MainForm

Private Function GetNewPay(ByVal current As Decimal, ByVal rate As Decimal) As Decimal
    ' calculates and returns the new hourly pay
    ' based on the current hourly pay and raise rate
    ' passed to it
    Dim raise As Decimal
    Dim newPay As Decimal
    raise = current * rate
    newPay = current + raise
    Return newPay
End Function

Private Sub ClearLabel(ByVal sender As Object, ByVal e As System.EventArgs) Handles listBox.SelectedIndexChanged, rateListBox.SelectedIndexChanged, newPayLabel.Click
    newPayLabel.Text = String.Empty
End Sub

Private Sub calcButton_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles calcButton.Click
    ' calls a function to calculate an employee's new
    ' hourly pay, then displays the new hourly pay
    Dim currentPay As Decimal
    Dim rateChoice As String
    Dim raiseRate As Decimal
    Dim newHourlyPay As Decimal
    currentPay = Convert.ToDecimal(currentListBox.SelectedItem)
    rateChoice = rateListBox.SelectedItem.ToString.TrimEnd("%")
    raiseRate = Convert.ToDecimal(rateChoice) / 100
    newHourlyPay = GetNewPay(currentPay, raiseRate)
End Sub
```

(Figure is continued on next page)
newPayLabel.Text = newHourlyPay.ToString("C2")
End Sub

Private Sub MainForm_Load(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles Me.Load
    ' fills the list boxes with values, then selects
    ' the first value
    For pay As Decimal = 7D To 12D Step 0.5D
        currentListBox.Items.Add(pay.ToString("N2"))
    Next pay

    For rate As Decimal = 0.02D To 0.11D Step 0.01D
        rateListBox.Items.Add(rate.ToString("P0"))
    Next rate

    currentListBox.SelectedIndex = 0
    rateListBox.SelectedIndex = 0
End Sub

Private Sub exitButton_Click(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles exitButton.Click
    Me.Close()
End Sub
End Class

FIGURE 7.17 Code for the Pine Lodge application
Programming Tutorial
Programming Example

![Rainfall Calculator](image)
Summary

• Function procedures return a value; Sub procedures do not return a value
• Event procedure: a Sub procedure associated with one or more objects and events
• Independent Sub and Function procedures: not associated with any specific object or event
• Call statement: used to invoke a procedure
• When calling a procedure, you must pass the same number, type, and order of parameter values as those declared in the procedure
Summary (continued)

- Values can be passed to a procedure *by value* or *by reference*
- **By Value:**
  - Provides only the value of the variable to the procedure
  - Use the `ByVal` keyword
- **By Reference:**
  - Provides the address of the variable to the procedure, allowing the procedure to change the variable’s value
  - Use the `ByRef` keyword
Summary (continued)

• Variables in the parameter list in a procedure header have procedure-level scope
• TryCast keyword: allows you to convert an object from one data type to another