

CHAPTER 1

INTRODUCTION TO AVENUE WRAPS

This book, **Avenue Wraps**, presents a set of guidelines for converting ArcView® 3.x Avenue scripts into Microsoft's Visual Basic® and ArcObjects™ code, jointly referred to in this book as "VB code", for incorporation into ArcGIS™ software. It is noted that the words "ArcObjects™ code" as used in this book represents a generic term for the use of ArcObjects™ properties and methods in conjunction with VB code. The reason for using Visual Basic® and not Visual C++®, or any other COM compliant software, is that the syntax of Visual Basic® is much closer to that of Avenue, than Visual C++.

Certain of the conversion guidelines presented in this book refer to Avenue requests that have a direct VB (subroutine or function) or ArcObjects™ counterpart. A direct counterpart implies that the operation is generally the same in both programming environments, even though the name may be somewhat different, and the location of the argument lists may follow rather than precede the request. For the most part, these guidelines are presented in Chapter 2 of this book. Speaking of terminology, the word "program" as used herein is a generic term referring to a main line program, subroutine or function, and is considered synonymous with the Avenue word "script", and "VB macro". Also included in Chapter 2 are the changes that need to be made to the Avenue code that do not utilize Avenue requests, that is, the syntax of the code. For example dealing with "for each" loops, "if" statements, intrinsic functions and so forth.

Unfortunately, there are numerous Avenue requests that do not meet the above criteria of a direct counterpart. In order to assist an Avenue programmer to expedite the conversion of Avenue code to VB code, various subroutines and functions have been written in Visual Basic® that simulate the operation of what are considered to be the most common Avenue requests. The scope and use of these subroutines and functions, jointly referred to as Avenue Wraps™, are presented in the subsequent chapters of this book, and a listing of each Avenue Wrap™ is presented in Appendix D.

The use of this book assumes a knowledge of Avenue programming, and a basic knowledge, although not extensive expertise in Visual Basic® and ArcObjects™ programming. Although primarily intended for those planning on converting Avenue code to ArcObjects™, certain of

I
N
T
R
O
D
U
C
T
I
O
N
S

A
V
E
N
U
E

W
R
A
P
S

T
O

**DECLARATION
OF VARIABLES**

the Avenue Wraps™ contained herein will prove helpful to those that do not convert from Avenue, but instead wish to program directly into VB code. In using this book, it is recommended that the reader first peruse the entire book, order of chapter reading is not essential, so as to familiarize him or her self as to the contents and the various Avenue Wraps™.

1.1 Comments on the Declaration of Variables

An attempt has been made to make the Avenue Wraps™ as generic as possible. Each Avenue Wrap™ description is preceded by its corresponding Avenue request, if one exists. The use of an Avenue Wrap™ is basically a substitution of an Avenue Wrap™ for an Avenue request. The name of an Avenue Wrap™ is essentially the same as that of the corresponding Avenue request. There are, however, two differences between an Avenue Wrap™ and an Avenue request. The first is that the argument list, for some Avenue Wraps™, will differ slightly with their Avenue counterpart. The second is that, whereas Avenue requests could be concatenated, Avenue Wraps™ cannot.

The description of each Avenue Wrap™ includes the Dim declaration statements of its associated variables, as well as a description of each of the variables. Note that this is a significant difference between Avenue and "VB code". Variables did not have to be declared in Avenue, however with "VB code", variables do. Regarding the Dim statements, the reader is alerted to the following:

- Dim statements should appear in the program that calls the subject subroutine or function, or in a preceding program that may be calling the program that calls a subroutine or function. For example consider a subroutine called Force being called by subroutine BBB, which in turn is called by AAA. In this case, the Dim statements for the variables y1, x1, y2, x2, dist, az should only appear in the subroutine AAA. So that the declarations would look like this:

```
Public Sub AAA
    Dim x3 As Double, y3 As Double, y1 As Double
    Dim x1 As Double, y2 As Double, x2 As Double
    Dim dist As Double, az As Double
    ... do something
    Call BBB(x3, y3, y1, x1, y2, x2, dist, az)
End Sub

Public Sub BBB(x3, y3, y1, x1, y2, x2, dist, az)
    Dim only whatever variables are local to BBB
```

It is noted that an Avenue "list" corresponds to a VB "collection", which is different than a VB array. In this book, the words "list" and "collection" are used as meaning the same. Where distinction is necessary with regards to arrays, it is so done.

```

' ---y1, x1, y2, x2 are the given variables
' ---dist and az are the returned variables
  Call Force(y1, x1, y2, x2, dist, az)
' ...do whatever with dist and az
End Sub
'

Public Sub Force(y1, x1, y2, x2, dist, az)
Dim only whatever variables are local to Force
' ... solve for dist and az
End Sub

```

DECLARATION OF VARIABLES

- Regarding the Dim statement of a function itself, consider the following sample statement, which may appear in a public sub called CCC, that uses the function avAddDoc (an Avenue Wrap™):

```
theLayer = avAddDoc(aDoc)
```

In the text of Appendix D, in which said Avenue Wrap™ is presented, one may get the impression when looking at the function's listing, that in addition to any other variables in the function's argument list, the function name, avAddDoc in this case, should be declared in the calling program as:

```
Dim avAddDoc As Integer
```

This is not the case. When a variable is set equal to a function, it is the variable and not the function name that should appear in a Dim statement within the calling program. In this case, in public sub CCC it is the variable *theLayer* that is declared as:

```
Dim theLayer As Integer      ' correct declaration
theLayer = avAddDoc(aDoc)
```

and not the function name:

```
Dim avAddDoc As Integer     ' incorrect declaration
theLayer = avAddDoc(aDoc)
```

- Only declare one variable per line, unless the type is specified for each variable, for example:

```
Dim opmode As Integer, xyzRecs As Integer
Dim x1 As Double, y1 As Double, z2 As Double
Dim aTitle1 As String, aTitle2 As String
```

If the type is not specified, the variable will default to be of Variant type, which could result in problems if the variable is to be purely numeric in nature.

DECLARATION OF VARIABLES

These types of variables are referred to as non-object types. A variable of object type is stored as a 32-bit (4-byte) address, which refers to an object. In addition, objects support methods and properties. ArcObjects™ offers the programmer a lot of different types of objects which enable the programmer to interact with the ArcGIS™ software. The approach taken in this book, when declaring objects, is to do so one per line, such as:

```
Dim pmxDoc As IMxDocument
Dim pPolygon As IPolygon
Dim graphList As New Collection
Dim theFields As New Collection
```

Note that, for the sake of conservation of space, in Appendix D, more than one object may be declared on a single line.

- Certain Avenue Wraps™ utilize global variables. Unlike Avenue, where a global variable was prefixed with the `_` character, global variables in "VB code" begin with the letters **ug**, and are initialized in the Avenue Wrap™ `avInit`, presented in Chapter 3 of this book. Global variables are used to pass information between the Avenue Wraps™ much like a common block would be used in a Fortran program.

The variables within each Avenue Wrap™ argument list are described as to their nature, and are classified as to being either "given" or "returned" arguments within the subroutine or function. There are some Avenue Wraps™ that may not have given and/or returned variables in their argument list. In such cases, the word "nothing" will appear.

1.2 Overview of the Chapter Contents

In the subsequent chapters, Avenue Wrap™ subroutine and function names that appear in bold type, such as **avGetDisplayFlush**, indicate that the complete listing of the subroutine or function is contained in Appendix D. Subroutines or functions that are not shown in bold type are either standard VB or ArcObjects™ code.

The available Avenue Wraps™ are presented in the chapters identified below, and they have been grouped according to function.

► Chapter 2 - General Conversion Guidelines

This chapter presents (a) a set of general syntax guidelines for converting Avenue to VB and ArcObjects™ code, (b) some Avenue Wraps™, and (c) addresses the syntax concerning the following:

<ul style="list-style-type: none">• Numbers and arithmetic operations,• String manipulation,• Transcendental and other functions,• Querying and testing variables,• Lists, arrays and collections, and• Program flow control such as Do and For loops.• Data type declaration including a summary table of how various types of variables, views, theme and table names, message box contents, and the like should be declared (dimmed). <p>▶ Chapter 3 - Project Organization Avenue Wraps™ This chapter contains Avenue Wraps™ that are associated with the establishment of an ArcView® project and its views, or are of a general application nature.</p> <p>▶ Chapter 4 - File I/O Avenue Wraps™ This chapter addresses Avenue Wraps™ that are concerned with the opening and closing of files, reading of files, writing to files, and other file handling operations including the extraction of data from delimited files.</p> <p>▶ Chapter 5 - Theme and Table Avenue Wraps™ This chapter contains Avenue Wraps™ pertaining to the handling of themes and their tables including:</p> <ul style="list-style-type: none">• Theme and table creation and retrieval,• Querying, editing and summarizing tables, and• Performing calculations on table cells. <p>▶ Chapter 6 - Feature Selection Avenue Wraps™ This chapter is comprised of various Avenue Wraps™ that help create and manipulate selection sets from various groups of features or rows.</p> <p>▶ Chapter 7 - Message and Menu Box Avenue Wraps™ This chapter is comprised of the Avenue Wraps™ that help create boxes with messages to the user, or present the user questions for action, provision for input of data, and/or selection from predefined lists.</p> <p>▶ Chapter 8 - Geometric Routines Avenue Wraps™ This chapter contains Avenue Wraps™ that enable the programmer to perform various geometric operations such as creating points, line, circles, and polygons, and extracting information thereof.</p>	OVERVIEW OF CHAPTER CONTENTS
---	---

<p>OVERVIEW OF CHAPTER CONTENTS</p>	<ul style="list-style-type: none"> <p>▶ Chapter 9- User Document Interaction This chapter contains various subroutines and functions that facilitate the graphic interaction ("making picks") between the user and ArcMap™.</p> <p>▶ Chapter 10- Graphics and Symbols Avenue Wraps™ This chapter contains Avenue Wraps™ that enable the programmer to manipulate graphics, assign attributes and symbology, and create and work with graphic text.</p> <p>▶ Chapter 11 - Classification and Legends This chapter contains various subroutines and functions with which the programmer may classify layers, create legends, and work with symbol palettes.</p> <p>▶ Chapter 12 - Utility Macros This chapter provides two subroutines that assist the programmer to (a) mass export VBA code to a specified directory, and (b) import or load VBA code from a specified directory into the current ArcMap™ project. This chapter also contains various subroutines and functions which are used by certain Avenue Wraps™ transparently to the programmer. In addition, these subroutines and functions may be used in the development of individual code to carry out certain geometric and other operations.</p> <p>▶ Appendices</p> <ul style="list-style-type: none"> <p>A Palette Index Values In this appendix, the programmer will find the various symbol index values for each of the 17 standard ArcMap™ palettes.</p> <p>B Color Calibration Diagrams In this appendix, the calibration charts for the gray scale, CMYK color model, HSV color model and RGB color model will be found.</p> <p>C Table of Avenue Request to Avenue Wrap™ Macro Mapping The table of this appendix summarizes in alphabetical order the most common Avenue requests and VB or ArcObjects statements, most of which are individually discussed in this book, and identifies the corresponding Avenue Wrap™ and location of its discussion by chapter and section.</p> <p>D Avenue Wrap™ Macro Listing In this appendix, the complete listing of the various Avenue Wrap™ scripts is presented in alphabetical script name order.</p>
--	---

1.3 Getting Started

GETTING STARTED

1.3.1 General Commentary

So we are now ready to start converting Avenue code. Before performing any conversion work, it is recommended that the **Visual Basic Environment (VB and VBA)** and **Visual Basic for Applications Development Environment** sections in the **ArcObjects Developer Help** be read. These sections can be found under the Contents tab, clicking on Getting Started, followed by clicking on Getting Started Start Page, see Figure 1-1. These are not very long sections but they provide valuable information that helps to explain what is presented in the subsequent chapters.

The approach recommended in this book for converting Avenue code into "VB code", is to initially perform the conversion work in the **VBA**, Visual Basic for Applications, environment and then build an extension, if appropriate, in the **VB**, Visual Basic, environment. The reason for doing so is that the **VBA** environment is similar to the Avenue environment in that the programmer can easily test and debug the application directly within ArcMap™. Once the application has been tested and is ready for distribution, the programmer can either (a) package the application as a protected

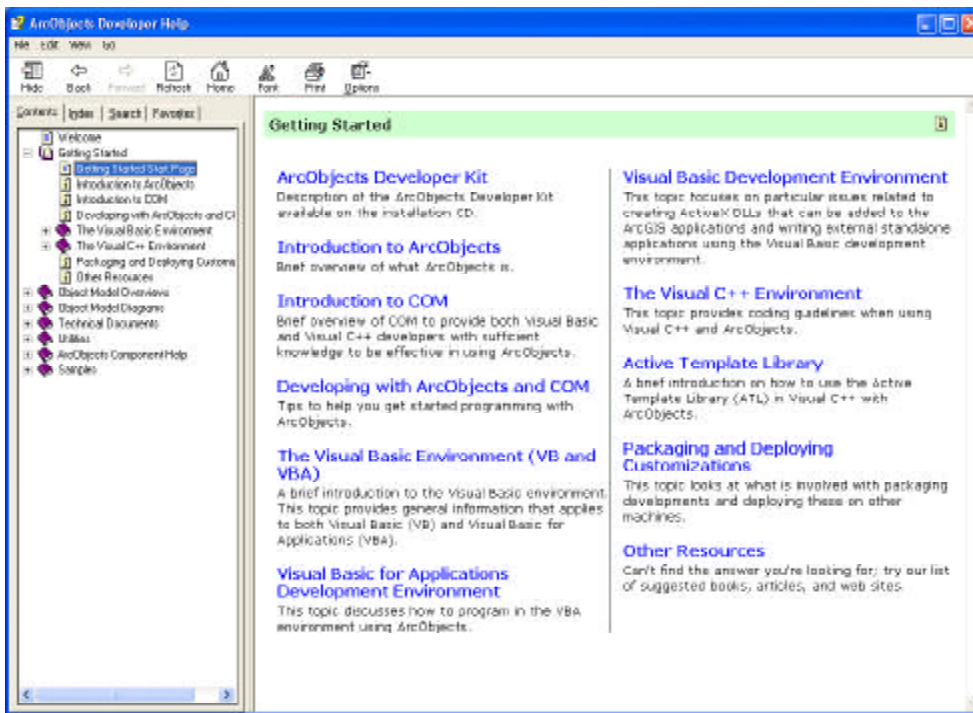


Figure 1-1 ArcObjects Developer Help

GETTING STARTED

project file, similar to an encrypted ArcView-Avenue project file, or (b) build an extension in the **VB**, Visual Basic, environment. Although there will be some duplicitous work (i.e. creating tools, menu items, etc.), any other approach would lead to a longer development/conversion cycle.

When an extension is to be built the programmer will be working in two different, yet somewhat similar, environments, first **VBA** and secondly (once the application has been tested) **VB**. It is suggested that the listings in Appendix D be reviewed, in terms of the variable declaration statements, to see how variables should be declared so that any re-coding can be eliminated. The point is that a programming style should be adopted such that the code can exist in both a **VBA** and **VB** environment.

When a protected project file is to be distributed, the end user working with the protected project file will still be able to customize the project file, but will be unable to modify or view the customizations provided by the developer. The developer in protecting the project file assigns a password that must be entered in order to modify or view the customizations. If the password is not properly specified, the end user is unable to modify or view the developer's customizations.

The following pages contain three examples of converting Avenue code into VB code. Note that the steps that are listed are merely a suggested procedure, and are presented here for the novice programmer who for the first time enters the development realm of ArcMap™ and VB.

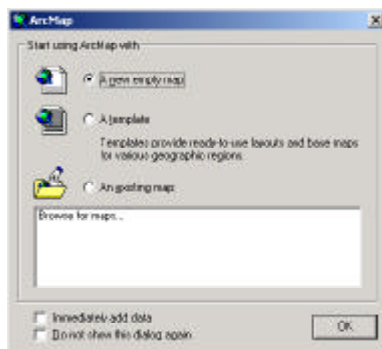



Figure 1-2
Existing Project Browser

- 1 **Invoke** ArcMap, at which time the window of Figure 1-2 is displayed within the window of Figure 1-3.
- 2 Accept the default selection to create a new empty map, and **click** at the **OK** button. The browser window disappears.
- 3 **Click** at the **T**ools menu and then at the **M**acros and **V**isual Basic Editor sub-menus (see Figure 1-4) to display the VBE work environment of Figure 1-5.

The VBE environment is divided into five areas, the menu and tool bars across the top, the Project sub-window in the upper left corner, the Properties sub-window below it, and the main work area (the large dark gray area) to the right.

1.3.2 Converting an Avenue System Script

For our first conversion example we will convert the Avenue system script View.ClearSelect.

- ✎ **4 Invoke** ArcView 3.x, and load either an existing project, or create a blank project.
- ✎ **5 Click** at the Scripts icon in the Project window, and then at the **New** button of said sub-window (see Figure 1-6A).
- ✎ **6 Click** at the **Load System Script** button, , to display a list of the system scripts, scroll down to **select** the **View.ClearSelect** script, and **click** at the **OK** button. See Figure 1-6(B). The system script is now displayed in the new script window. See Figure 1-6(C). The function of this script is to deselect (clear) any selected features in the active theme within the active document.

We will now translate the above Avenue system script into VB code. Since this script is so small, there is no need to copy it and paste it into a VBA module.

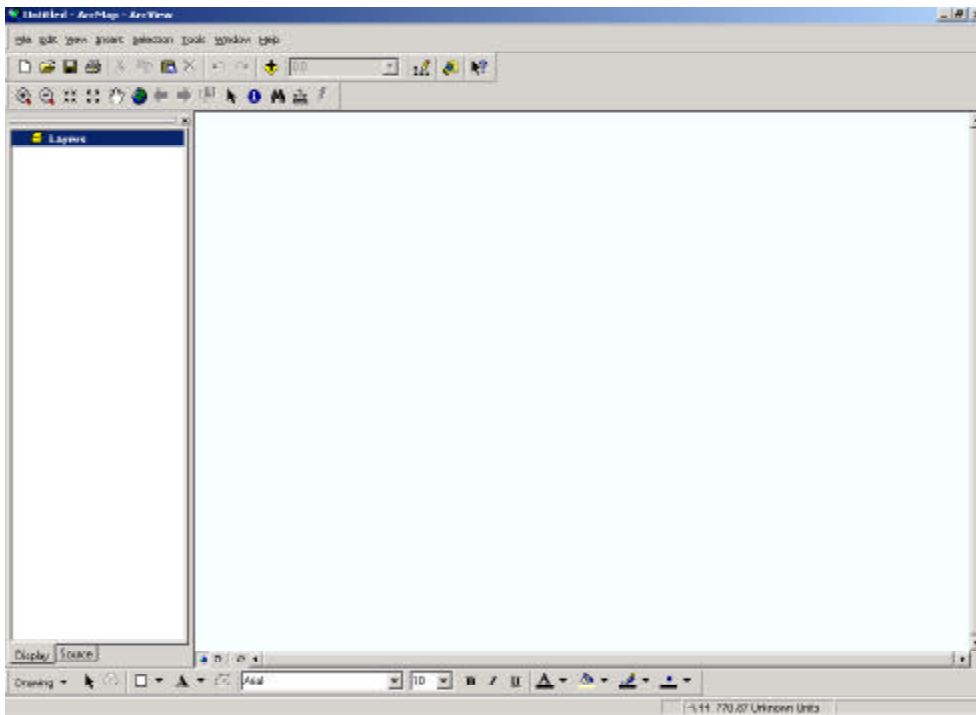


Figure 1-3 ArcMap Work Window

Converting
an Avenue
System Script

Converting an Avenue System Script

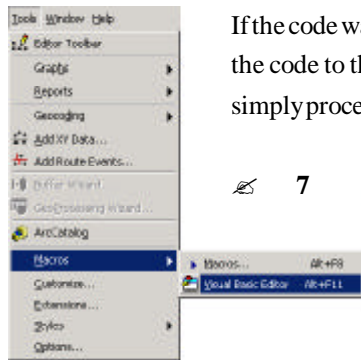


Figure 1-4 Tools Menu and Macros/Visual Basic Editor Sub-Menus

If the code was larger we could use normal Windows functionality to copy the code to the clipboard and paste it into a VBA module. Thus, we will simply proceed to code it into VBA using the Avenue Wraps DLL. So that,

7

Go back to the VBE work environment of Figure 1-5, and **click** at the **I**nsert menu and then at the **M**odule sub-menu. *Module1* is now displayed in the Properties sub-window of Figure 1-5, and *Module1 (Code)* is displayed in the work area of the same figure. This process is essentially the same as creating a new script window in *ArcView 3.x*.

8

In the Properties sub-window of Figure 1-5, **double-click** on top of the name of the module, *Module1*, and replace the name by key **entering** the new name to be assigned to the module, **avViewClearSelectMOD**, followed by **depressing** the **Enter** key. All references to *Module1* should have been changed to reflect the new module name. The extension MOD, which appears in the module name, denotes that the file is a module. It is not possible to have a module name that is the same as the name of a subroutine or function.

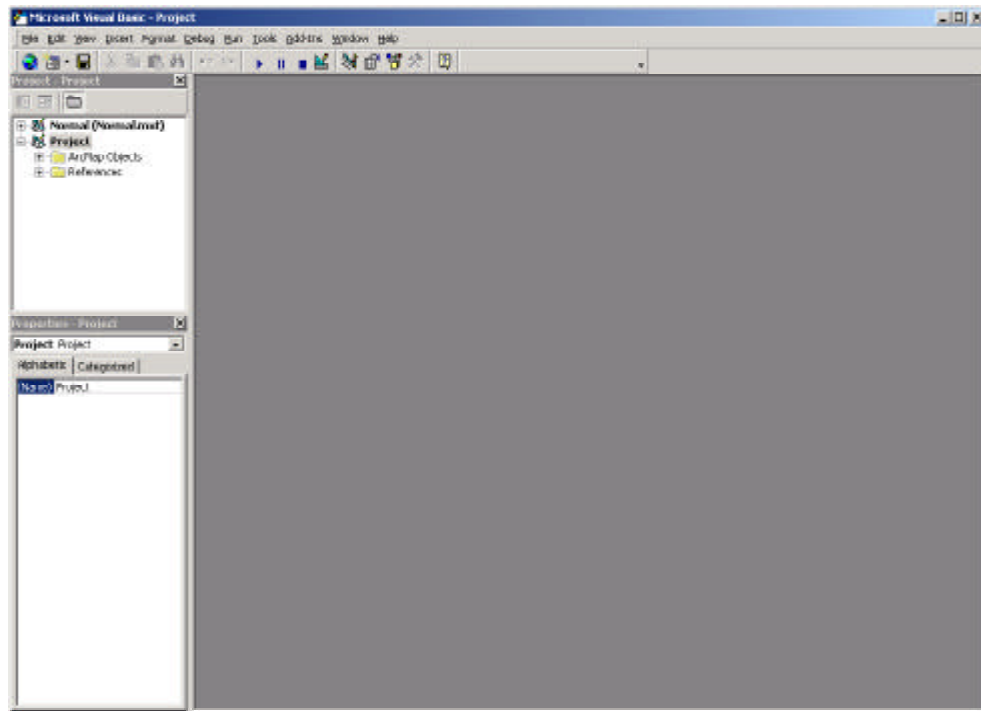


Figure 1-5 VBE Work Environment

Converting an Avenue System Script

(A) Blank ArcView 3.x Work Space

(B) Script Selection

```

theView = av.GetActiveDoc
for each t in theView.GetActiveThemes
  t.ClearSelection
end

```

(C) The View.ClearSelect Script

Figure 1-6 Opening an ArcView 3.x System Script

As a point of clarification, scripts in Avenue are referred to as procedures, in VB/VBA, and are stored in modules. Modules, unlike scripts, can contain more than one procedure. The most straight forward approach is to create a module, with a single procedure within it, for every script to be converted.

The script to be converted, in this example, is to be a public subroutine so that it may be called by various other procedures (subroutines and functions) to be written later on. Thus, in the VBE work environment of Figure 1-5, we will now create a public subroutine.

✎ **9 Click** in the title bar of the avViewClearSelectMOD window to make the window active. **Click** at the **I**nsert menu and then at the **P**rocedure... sub-menu. In the data field to the right of the Name: label, **type avViewClearSelect** and **click** at the **O**K button. Note that under the Type and Scope frames the default values denote a public subroutine is to be established.

The following lines of code will appear.

```

Public Sub avViewClearSelect()

End Sub

```

Converting an Avenue System Script

Referencing the Avenue Wraps avwraps.dll file.

Note that there are no arguments between the two parentheses in the Public Sub avClearSelect() line, thus indicating that this subroutine is not to have any given, nor any returned variable arguments.

In converting the Avenue script we will utilize the Dynamically Linked Library (DLL) implementation of the Avenue Wraps. In so doing, we eliminate the need to include all of the Avenue Wraps source in the converted application. To incorporate the Avenue Wraps DLL we must make a reference in the VBE work environment to the Avenue Wraps DLL.

➤ **10** Click at the **T**ools menu and then at the **R**eferences... sub-menu to display the **R**eferences-Project window (Figure 1-7).

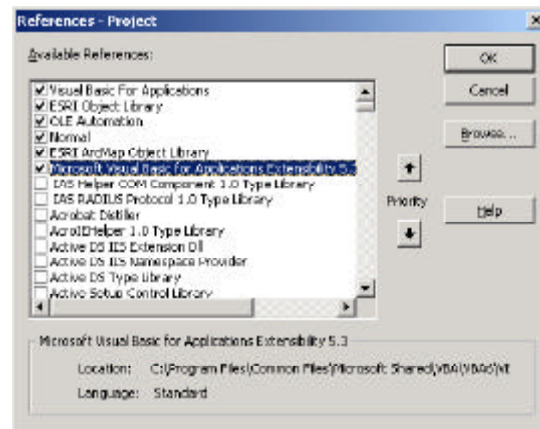


Figure 1-7 References - Project Window

➤ **11** Click at the **B**rowse button to display the **A**dd Reference file dialog box. Then,

- **N**avigate to the directory in which the avwraps.dll file is located (see Figure 1-8),
- **C**lick at the name of the avwraps.dll file,
- **C**lick at the **O**pen button (Figure 1-7 is displayed again, and includes the avwraps.dll file), and then
- **C**lick at the **O**K button to confirm.

The avwraps.dll has now been referenced in the VBA application, and all of the Avenue Wraps are now available to the developer.

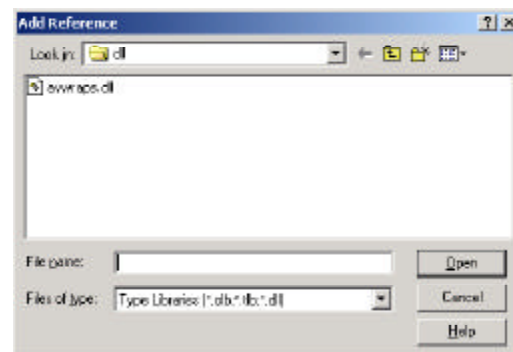


Figure 1-8 Add Reference Window

➤ **12** Click in front of the word Public to position the cursor and **t**ype **O**ption **E**xplicit, followed by **d**epressing the **E**nter key, **t**wice. The code should look like this:

```
Option Explicit
```

```
Public Sub avViewClearSelect()
```

```
End Sub
```

The statement `Option Explicit` informs the VBE compiler that all variables are to be explicitly declared. Although an optional statement, it helps tremendously in the development/conversion process.

Now, **click** at the blank line between the second and third lines (between `Public Sub avViewClearSelect` and `End Sub`), and **key enter** the statements below. The comment statements explain the code.

```
'
' ---The Avenue Wrap below gets the active document.
' ---Refer to section 3.1.3 of this book.
Call avGetActiveDoc(pMxApp, pmxDoc, pActiveView, pMap)
'
' ---The Avenue Wrap below creates a collection of the
' ---various layers (themes) in the active document.
' ---Refer to section 3.1.4 of this book.
Call avGetActiveThemes(pmxDoc, ThemesList)
'
' ---We will now determine the number of layers (themes) in
' ---the active document
NumThemes = ThemesList.Count
'
' ---The Avenue Wrap below within the For Each loop will
' ---deselect the selected features in each of the themes
' ---Refer to section 6.2.2 of this book.
For i = 1 To NumThemes
    theTheme = ThemesList.Item(i)
    Call avClearSelection(pmxDoc, theTheme)
Next
```

- ✍ **13** It is now time to declare the various variables that were used in the above code. In declaring these variables, note the following:

**Converting
an Avenue
System Script**

Converting an Avenue System Script

- Keep in mind the comments in the previous section 1.1 that the calling arguments (given and returned) of a subroutine or function should be declared in the program where they were first called, not in the subroutine or function itself. Since this subject subroutine has no given and no returned variables, all variables used in this subroutine are considered local and should be declared within it.
- Variables used locally in a program may be declared anywhere in the procedure (subroutine or function), but prior to where first used. However, it may be considered more appropriate, and easier to locate if need be, if they are all located at the beginning of the program, below the name of the subroutine or function, and more or less, in the order of their use.

✍ **14** Click below the second line (Public Sub ...), and **key enter** the declaration statements below.

```
Dim pMxApp As IMxApplication
Dim pmxDoc As IMxDocument
Dim pActiveView As IActiveView
Dim pMap As IMap
Dim ThemesList As New Collection
Dim NumThemes As Long, i As Long
Dim theTheme As Variant
```

Writing computer code is in essence no different than writing any text, be it fiction or a technical report. The style of the text is dependent upon the style of the writer. There are, however, certain writing rules that are essential to any text, such as clarity, and in the case of software code, in-line documentation describing what is being done. There is a thinking among many, particularly experienced programmers, that the code itself is adequate documentation; thus, comment statements are many times sparse, if at all. No matter how good a programmer is, the need will come some months, if not years later, when (a) the programmer's memory may not be as good as first thought, and (b) when program logic may not be as easily discernible. It is at this point of time, when properly located comment statements will be more than just welcomed and appreciated. This is not to justify the number of comment statements in the preceding sample code. One may justifiably claim that they are verbose, and some of them redundant. The purpose of their introduction was purely to lucidly describe to a novice as to what is being done.

On the issue of code clarity, it is quite important to describe at the very start of a procedure the purpose or objective of the procedure, and identify its given and returned variables, as well as any special conditions that may pertain to, or required, by the program for its proper execution. If the program is a main line program, a skeleton description of its flow may be apropos. In the "good old days" of computer programming, a detailed flow chart was a requirement, if not a necessity. Nowadays, a concise description in good English, or what may be the language of the programmer, is considered appropriate. Thus it is now time to do just that.

- ✍ **15 Click** below the first line (Option Explicit), and **key enter** the comment statements below. Note that comment statements can be created by using the ' character, which may appear anywhere on a data line. Again note that these comments are just a suggestion which the authors utilize in their programming. Others may customize them to their specific needs and desires.

```

\
\ * * * * *
\ *
\ * Name: avViewClearSelect File Name: avclears.bas *
\ *
\ * * * * *
\ *
\ * PURPOSE: Deselect all selected features in all layers *
\ * (themes) of an active application (document) *
\ *
\ * GIVEN: nothing *
\ *
\ * RETURN: nothing *
\ *
\ * Dim (in this area the declaration statements of the given *
\ * and returned variables of the program would be shown *
\ * as comments. In the subject example, since there are *
\ * no given and returned variables, omit these comments) *
\ *
\ * * * * *
\

```

The complete code of the above example is contained in Table 1-1.

Converting an Avenue System Script

Note that the procedure name and the disk file name may be the same, or different. An 8.3 file name convention is used here.

Converting an Avenue System Script

Table 1-1 SAMPLE VB CODE

```
Option Explicit
\
\ * * * * *
\ *
\ *   Name: avViewClearSelect           File Name: avclears.bas *
\ *
\ * * * * *
\ *
\ *   PURPOSE:  Deselect all selected features in all layers *
\ *              (themes) of an active application (document) *
\ *
\ *   GIVEN:    nothing *
\ *
\ *   RETURN:   nothing *
\ *
\ *   Dim (in this area the declaration statements of the given *
\ *         and returned variables of the program would be shown *
\ *         as comments. In the subject example, since there are *
\ *         no given and returned variables, omit these comments) *
\ *
\ * * * * *
\
```

```
Public Sub avViewClearSelect()
'
'   Dim pMxApp As IMxApplication
'   Dim pmxDoc As IMxDocument
'   Dim pActiveView As IActiveView
'   Dim pMap As IMap
'   Dim ThemesList As New Collection
'   Dim NumThemes As Long, i As Long
'   Dim theTheme As Variant
'
'   ---The Avenue Wrap below gets the active document.
'   ---Refer to section 3.1.3 of this book.
'   Call avGetActiveDoc(pMxApp, pmxDoc, pActiveView, pMap)
'
'   ---The Avenue Wrap below creates a collection of the
'   ---various layers (themes) in the active document.
'   ---Refer to section 3.1.4 of this book.
'   Call avGetActiveThemes(pmxDoc, ThemesList)
'
'   ---We will now determine the number of layers (themes) in
'   ---the active document
'   NumThemes = ThemesList.Count
'
'   ---The Avenue Wrap below within the For Each loop will
'   ---deselect the selected features in each of the themes
'   ---Refer to section 6.2.2 of this book.
'   For i = 1 To NumThemes
'       theTheme = ThemesList.Item(i)
'       Call avClearSelection(pmxDoc, theTheme)
'   Next
'
End Sub
```


1.3.3 Converting a Geometric Avenue Script

In this example, we will convert a user developed Avenue script, iccomdis, which is supposed to be given two pairs of Cartesian coordinates and compute, and thus return, the distance between them. The Avenue listing of this script may be found in Table 1-2. When converted, this script is to be a function and operate in the same manner as the Avenue script. The data lines, which appear in the color red, have a VB syntax error.

Converting
a Geometric
Avenue Script

Table 1-2 SAMPLE AVENUE CODE

```

|
| * * * * *
| *
| *   Name: iccomdis                               File Name: iccomdis.ave *
| *
| * * * * *
| *
| *   PURPOSE:  TO COMPUTE THE DISTANCE BETWEEN TWO POINTS *
| *
| *           Given: X1,Y1 = coordinates of point 1 *
| *                  X2,Y2 = coordinates of point 2 *
| *
| *           Return: DIST = distance between the two points *
| *
| * * * * *
| *
| *   SCRIPTS CALLED BY SCRIPT "iccomdis" :          none *
| *
| * * * * *
|
|   ---Extract the incoming values
|   X1 = SELF.Get(0)
|   Y1 = SELF.Get(1)
|   X2 = SELF.Get(2)
|   Y2 = SELF.Get(3)
|
|   ---COMPUTE DISPLACEMENTS BETWEEN POINTS
|   DX = X2 - X1
|   DY = Y2 - Y1
|
|   ---CHECK IF POINTS ARE IDENTICAL
|   IF((DX.Abs < _gTolV3) AND (DY.Abs < _gTolV3)) then
|       DIST = 0.0
|
|   ---HANDLE CASE OF TWO UNIQUE POINTS
|   else
|
|       ---COMPUTE THE DISTANCE
|       DIST = ((DX*DX) + (DY*DY)).Sqrt
|   end
|
|   ---Return the script "iccomdis" results
|   Return DIST

```

Converting a Geometric Avenue Script

The first three steps of this conversion process for this script are the same as those of the preceding conversion process, so that once we have entered the VBE work environment, we will commence with Step 4.

- ✎ **4** Click at the **F**ile menu and then at the **I**mport sub-menu of Figure 1-4. The conventional Window file browser window is now displayed.
- ✎ **5** **N**avigate to the appropriate directory and **s**elect the desired file, in this example the file is called iccomdis.ave. The contents of Table 1-2 are displayed.

Table 1-3 SAMPLE VB CODE

```
Option Explicit
'
' * * * * *
' *
' *   Name: iccomdis                               File Name: iccomdis.bas *
' *
' * * * * *
' *
' *   PURPOSE:  TO COMPUTE THE DISTANCE BETWEEN TWO POINTS *
' *
' *   GIVEN:    X1,Y1   = coordinates of the first point *
' *             X2,Y2   = coordinates of the second point *
' *
' *   RETURN:   iccomdis = distance between the two points *
' *
' *   Dim X1, Y1, X2, Y2, iccomdis As Double *
' *
' * * * * *
'
Public Function iccomdis(X1, Y1, X2, Y2) As Double
'
'   Dim DX As Double, DY As Double
'
'   ---COMPUTE DISPLACEMENTS BETWEEN POINTS
'   DX = X2 - X1
'   DY = Y2 - Y1
'
'   ---CHECK IF POINTS ARE IDENTICAL
'   If ((Abs(DX) < ugTolV3) And (Abs(DY) < ugTolV3)) Then
'       iccomdis = 0#
'
'   ---HANDLE CASE OF TWO UNIQUE POINTS
'   Else
'
'       ---COMPUTE THE DISTANCE
'       iccomdis = Sqr((DX * DX) + (DY * DY))
'   End If
'
End Function
```

Note that this is a different approach, in importing an Avenue script into a VBA module, from what was shown in the previous example. In addition, the user will need to change the Files of type: display, to be all files, in order to see the appropriate file(s) in the Import File dialog box.

When importing Avenue scripts in this manner, the default name of the module will be moduleX, where X is the next available number beginning at one. The programmer will then have to change the name following the method described in Step 8 of the previous example.

In general, comment lines will appear in a VBA module in the color, green. Data lines that have a syntax error will appear in the color, red.

The comments below make reference to the converted code shown in Table 1-3.

- ✍ **6** **Modify** the program banner as shown at the top of Table 1.3. Note the:
 - Description of the objective (purpose) of the program;
 - Identification and description of the given and returned variable arguments; and
 - Comment declaration statements regarding the given and returned variables.

- ✍ **7** Since this program is to return only one variable, the distance between the two points, it will be defined as a public function.

- ✍ **8** The variables DX and DY are the only variables local to this program.

- ✍ **9** The program logic is to remain the same. Note the following:
 - The change in the ending of the "If" statement loop.
 - The changes in the name and location of the calling arguments of the absolute value and square root intrinsic functions.
 - The comparison of the DX and DY variables against the global variable ugToIV3. The global variable is set by the Avenue Wrap™ avInit (see section 3.1.8).

**Converting
a Geometric
Avenue Script**

Converting Another Geometric Avenue Script

1.3.4 Converting Another Geometric Avenue Script

This example is to convert a user developed Avenue script, iccomppt, which is supposed to compare the Cartesian coordinates of a point with those of another given point, and return an indicator flag concerning the comparison, as well as a new pair of coordinates for the first point depending upon whether a match is made or not. The Avenue listing of this script may be found in Table 1-4. When converted, this script is to be a subroutine and operate in the same manner as the Avenue script.

The first three steps of this conversion process for this script are the same as those of the preceding conversion process, thus we will commence with Step 4.

- ✍ **4** Click at the **F**ile menu and then at the **I**mport sub-menu of Figure 1-4. The conventional Window file browser window is now displayed.
- ✍ **5** **N**avigate to the appropriate directory and **s**elect the file, iccomppt.ave. The contents of Table 1-4 are displayed.

The comments below make reference to the converted code shown in Table 1-5.

- ✍ **6** **M**odify the program banner as shown at the top of Table 1.5. Note the:
 - Description of the objective (purpose) of the program;
 - Identification and description of the given and returned variable arguments; and
 - Comment declaration statements regarding the given and returned variables.
- ✍ **7** Since this program is to return more than one variable, it will be defined as a public subroutine.
- ✍ **8** Following the definition of the subroutine are the declarations of the various variables local to this subroutine. When a program is small, say a page or two or thereabouts, it is relatively easy to identify all variables and declare them. However, let us be realistic. If a program is large, some if not many of the variables will not be recognized for declaration until the testing period, when the compiler will bring those variables to the programmer's attention. This will occur because the Option Explicit statement appears in the module.

Table 1-4 SAMPLE AVENUE CODE

```

|
| * * * * *
| *
| * Name: iccomppt                               File Name: iccomppt.ave *
| *
| * * * * *
| *
| * PURPOSE: CHECK IF A GIVEN POINT MATCHES ANOTHER POINT USING *
| *           A TOLERANCE BASED UPON THE DISPLAY OF THE VIEW *
| *           Given: XCORD,YCORD = coord's point to be matched *
| *                   X2,Y2      = coord's to be matched against*
| *           Return: XCORD,YCORD = input values if NOFND = 0 *
| *                   = X2,Y2 values if NOFND = 1 *
| *           NOFND = 0 : no match was found *
| *                   = 1 : match found within tolerance. *
| *
| * * * * *
|
| ---Get the active view                                <<<-----
| theView = av.GetActiveDoc
|
| ---Extract the incoming values
| XCORD = SELF.Get(0)
| YCORD = SELF.Get(1)
| X2     = SELF.Get(2)
| Y2     = SELF.Get(3)
|
| ---Obtain the snap tolerance value
| returnList = av.Run("SetViewSnapTol",{theView,X2,Y2})
|
| ---Set the tolerance using the projected coordinate system since
| ---the coordinates being used are in a projected coordinate system
| difxxx = returnList.Get(4)
|
| ---DEVELOP ENCLOSING BOX ABOUT POINT
| XTOLUP = X2 + difxxx
| XTOLDN = X2 - difxxx
| YTOLUP = Y2 + difxxx
| YTOLDN = Y2 - difxxx
|
| ---CHECK IF PICK WITHIN ENCLOSING BOX
| IF((XCORD > XTOLUP) or (XCORD < XTOLDN)) then
|     NOFND = 0
|
| elseif((YCORD > YTOLUP) or (YCORD < YTOLDN)) then
|     NOFND = 0
|
| ---POINT MATCHES SPECIFIED POINT
| else
|     NOFND = 1
|
|     ---ADJUST GIVEN COORDINATES
|     XCORD = X2
|     YCORD = Y2
| end
|
| ---Return the script "iccomppt" results
| Return {XCORD,YCORD,NOFND}
|

```

Converting
Another
Geometric
Avenue Script

Converting
Another
Geometric
Avenue Script

Table 1-5 SAMPLE VB CODE

```

Option Explicit
'
' * * * * *
' *
' *   Name: iccomppt                               File Name: iccomppt.bas *
' *
' * * * * *
' *
' * PURPOSE:  CHECK IF POINT IS WITHIN A TOLERANCE, THAT VARIES *
' *           BASED UPON THE VIEW DISPLAY, OF ANOTHER POINT *
' *
' * GIVEN:    XCORD,YCORD = coordinates of point to be checked *
' *           X2,Y2       = coordinates of the base point *
' *
' * RETURN:   XCRD2,YCRD2 = input values if NOFND = 0 *
' *           = X2,Y2     values if NOFND = 1 *
' *           NOFND      = 0 : no match was found *
' *           = 1 : match was found within the point *
' *                   snapping tolerance. *
' *
' * Dim XCORD, YCORD, X2, Y2, XCRD2, YCRD2 As Double *
' * Dim NOFND As Integer *
' *
' * * * * *
'
Public Sub iccomppt(XCORD, YCORD, X2, Y2, XCRD2, YCRD2, noFnd)
'
    Dim pMxApp As IMxApplication
    Dim pmxDoc As IMxDocument
    Dim pActiveView As IActiveView
    Dim viewRect As Double
    Dim thePoint As IPoint
    Dim difxxx As Double, difzzz As Double, difPrj As Double
    Dim XTOLUP As Double, XTOLDN As Double
    Dim YTOLUP As Double, YTOLDN As Double
'
' ---Get the active view
    Set pMxApp = Application
    Set pmxDoc = Application.Document
    Set pActiveView = pmxDoc.ActiveView
'
' ---Initialize the coordinates to be passed back
    XCRD2 = XCORD
    YCRD2 = YCORD
'
' ---Obtain the snap tolerance value based upon the current
' ---view extent
    Call SetViewSnapTol(pmxDoc, X2, Y2, _
        viewRect, thePoint, difxxx, difzzz, difPrj)
'
' ---DEVELOP ENCLOSING BOX ABOUT POINT
    XTOLUP = X2 + difPrj
    XTOLDN = X2 - difPrj
    YTOLUP = Y2 + difPrj
    YTOLDN = Y2 - difPrj

```

Table 1-5 SAMPLE VB CODE (continued)

```
'  
' ---CHECK IF PICK WITHIN ENCLOSING BOX  
If ((XCORD > XTOLUP) Or (XCORD < XTOLDN)) Then  
    noFnd = 0  
'  
    ElseIf ((YCORD > YTOLUP) Or (YCORD < YTOLDN)) Then  
        noFnd = 0  
'  
' ---POINT MATCHES SPECIFIED POINT  
Else  
    noFnd = 1  
'  
' ---ADJUST GIVEN COORDINATES  
    XCRD2 = X2  
    YCRD2 = Y2  
End If  
'  
End Sub
```

- ✎ 9 The program logic is to remain the same. Note the following
- The changes in how the subroutine SetViewSnapTol is called. In Avenue, the script, SetViewSnapTol, returned a list, returnList, which contained five items. In the VB code, the subroutine, SetViewSnapTol, passes back the 5 items, not in a list, but as individual items. It is the programmer's discretion as to how to handle the returned arguments in a subroutine call.
 - The change in the ending of the "If" statement loop.

Converting
Another
Geometric
Avenue Script

Converting
Another
Geometric
Avenue Script