CHAPTER 5

THEME AND TABLE AVENUE WRAPS

This chapter contains Avenue Wraps pertaining to the handling of themes and their tables including (a) layer (theme) and table creation, retrieval, visibility control and manipulation, (b) creation and extraction of fields and attributes thereof, (c) extraction of lists of records, editing of records and features, and storing of values and shapes, (d) redrawing of features. (e) querying and summarizing tables, (f) performing calculations on table cells, and (g) creation of shapefiles and personal geodatabases.

An example has been included at the end of this chapter. The example demonstrates how to create a (a) shapefile, and (b) a table, as well as, how to edit, query and summarize the table.

The Avenue Wraps of this chapter are listed below in alphabetical order with a short description and the chapter - page number where a full description may be found.

◗	**CreateShapeFile**	To create a new shapefile based upon the information explicitly defined in the calling arguments (no user interaction)	5-64
◗	**FindLayer**	To find a layer in a map returning an ILayer object	5-23
◗	**FindTheme**	To find a layer in a map	5-25
◗	**Sample Code**	Code examples on how to create a shapefile and a table, and how to perform various other operations	5-79

The source listing of each of the above Avenue Wraps may be found in Appendix D of this publication.

5.1 Layer (Theme) and Table Related Avenue Wraps

LAYERS (THEMES) and TABLES

5.1.1 Subroutine avCheckEdits

avCheckEdits

This subroutine enables the programmer to perform checks on the editing of data. This routine first determines whether the editor is in an edit state or not. If it is not in an edit state, this routine does nothing. If it is in an edit state, it checks to see if the given data set is currently being edited. If it is not, the routine saves the edits on the data set currently being edited, and starts the editor on the given data set. If the given data set is currently being edited, the routine does nothing.

The corresponding Avenue request is:

There is no corresponding Avenue request.

The call to this Avenue Wrap is:

Call **avCheckEdits**(pEditor, pDataSet)

GIVEN: pEditor = the ArcMap Editor extension
pDataSet = the data set to be processed. If the word NOTHING is specified, and if the editor is in an edit state, the editor is stopped, and any edits that may have been made are saved. If the editor is in not in an edit state, the routine does nothing.

RETURN: Nothing

The given and returned variables should be declared where first called as:

```
Dim pEditor As IEditor
Dim pDataSet As IDataset
```

5.1.2 Subroutine avCreateTable

This subroutine enables the programmer to create a new dBase file, from an existing ITable object, and a set of rows (ICursor object) and add the new dBase file to the document. In using this subroutine, note the following:

1. If the dBase file to be created exists on the disk, it will be deleted prior to creating the new file. The user will not be asked for confirmation whether the existing file is to be deleted or not.
2. If the given name of the table to be created does not contain a complete

LAYERS (THEMES) and TABLES

pathname, the current working directory will be used. Some examples of filName include: c:\project\test\atable.dbf
atable.dbf

3. If the table can not be exported for any reason what so ever, an error message to that effect will be displayed.
4. This subroutine is called by the avTableSort subroutine.
5. The argument filName can or can not contain the .dbf extension

The corresponding Avenue request is:
There is no corresponding Avenue request.

avCreateTable

The call to this Avenue Wrap is:
Call **avCreateTable**(pTable, pCursor, filName)

GIVEN: pTable = the ITable object to be processed
pCursor = the ICursor object containing the data that will be written to the new .dbf file.
filName = the name of the new dBase file to be created.

RETURN: Nothing

The given and returned variables should be declared where first called as:
Dim pTable As ITable, pCursor As ICursor, filName As String

5.1.3 Function avFTabExport

This function enables the programmer to create a new (a) shapefile, (b) dBase file or (c) comma delineated text file. In using this function, note the following:

1. In specifying the name of the new shapefile or table (aFileName), if the name does not contain a complete path name, the current working directory will be used. Some examples of the name include:
   c:\project\test\atable c:\project\test\atable.shp
   atable atable.shp
   The name may or may not contain the extensions .dbf, .txt or .shp
2. If the item to be created, aFileName, exists on disk, it will be deleted, before the exporting is performed, without informing the user/developer.
3. If selected features are to be exported and there are no selected features, the entire theme will be exported.
4. If the theme can not be exported for any reason what so ever, avFTabExport will be set to NOTHING.

5. If the new theme or table is not to be added to the map, avFTabExport will be set to NOTHING.
6. Use the subroutine avInvalidateTOC to refresh the Table of Contents.
7. aTheme and aFileName can not be identical, they must be different, if not an error is generated.

The corresponding Avenue request is:

```
newVTab = aFTab.Export (aFileName, aClass, SelRecrds)
```

The call to this Avenue Wrap is:


```
newVTab = avFTabExport(aTheme, aFileName, aClass, _
                       SelRecrds, addToDoc)
```

GIVEN:
- aTheme = the name of the theme to be exported.
- aFileName = name of the shapefile or table to be created.
- aClass = the type of table to be created. Specify: "dBase", "TEXT" or "SHAPE".
- SelRecrds = indicator whether selected, or all records of aTheme are to be exported. Specify:
  - true to export selected records only.
  - false to export all records.
- addToDoc = optional indicator which may or may not appear in the argument list that denotes whether the new shapefile or table, aFileName, is to be added to the map. Specify:
  - true to add the new shapefile or table.
  - false to not add.

RETURN:
- newVTab = the IFields object that is created.

The given and returned variables should be declared where first called as:

```
Dim aTheme, aFileName, aClass As String
Dim SelRecrds As Boolean, addToDoc As Boolean
Dim newVTab As IFields
```

5.1.4 Function avFTabMakeNew

This function enables the programmer to create a new shapefile, the name and type (class) of which are specified by the programmer as given arguments. In using this function, note the following:

1. Regarding the name of the shapefile:

LAYERS (THEMES) and TABLES

Some examples of a valid shapefile name include:

c:\project\test\l_0ln or c:\project\test\l_0ln.shp

l_0ln or l_0ln.shp

2. The shapefile name may or may not contain the extension .shp.
3. If the name does not contain a complete path name, the current working directory will be used.
4. Regarding the shapefile type (class), it may be one of the following:

POINT	MULTIPOINT	POLYLINE	POLYGON
POINTM	MULTIPOINTM	POLYLINEM	POLYGONM
POINTZ	MULTIPOINTZ	POLYLINEZ	POLYGONZ

5. This function creates three fields called FID, SHAPE and ID.
6. Subsequently to this function, the function avAddDoc may be used to add the shapefile to the map, if need be.
7. If the shapefile to be created exists on the disk, the routine will abort, and the existing shapefile will not be overwritten

The corresponding Avenue request is:

```
theNewFTab = FTab.MakeNew(aFileName, aClass)
```

avFTabMakeNew

The call to this Avenue Wrap is:

```
Set theNewFTab = avFTabMakeNew(aFileName, aClass)
```

GIVEN: aFileName = name of the shapefile to be created (refer to notes 1, 2 and 3 above)

aClass = type of shapefile to be created (refer to note 4 above)

RETURN: theNewFTab = feature layer object that is created

The given and returned variables should be declared where first called as:

```
Dim aFileName, aClass As String
Dim theNewFTab As IFeatureLayer
```

5.1.5 Function avGetShapeType

This function enables the programmer to get the default shape type for a theme. In using this function, note the following:

1. The shapefile type (class), may be one of the following:

POINT	MULTIPOINT	POLYLINE	POLYGON
POINTM	MULTIPOINTM	POLYLINEM	POLYGONM
POINTZ	MULTIPOINTZ	POLYLINEZ	POLYGONZ

2. The Avenue request corresponding to the subject function operates on the FTab of a theme, while this VB function operates on the layer (theme).

The corresponding Avenue request is:

```
anFTab = theTheme.GetFTab
theShapeType = anFTab.GetShapeClass
```

The call to this Avenue Wrap is:

```
theShapeType = avGetShapeType(pmxDoc, theTheme)
```

avGetShapeType

GIVEN: pmxDoc = the active view
theTheme = the theme to be processed

RETURN: theShapeType = the default shape type of the theme

The given and returned variables should be declared where first called as:

```
Dim pmxDoc As IMxDocument
Dim theTheme As Variant
Dim theShapeType As esriGeometryType
```

5.1.6 Subroutine avGetTables

This subroutine enables the programmer to get a list of the names of the tables in the document, as well as, a list of their corresponding ITable objects.

The corresponding Avenue request is:

There is no corresponding Avenue request.

The call to this Avenue Wrap is:

```
Call avGetTables(pmxDoc, nameList, tableList)
```

avGetTables

GIVEN: pmxDoc = the active view

RETURN: nameList = list containing the names of the tables in the document
tableList = list containing ITable objects which correspond to the table names in nameList

The given and returned variables should be declared where first called as:

```
Dim pmxDoc As IMxDocument
Dim nameList As New Collection, tableList As New Collection
```

LAYERS (THEMES) and TABLES

5.1.7 Subroutine avGetThemeExtent

This subroutine enables the programmer to get the smallest rectangle enclosing a layer (theme).

The corresponding Avenue request is:

There is no corresponding Avenue request.

avGetThemeExtent

The call to this Avenue Wrap is:

Call **avGetThemeExtent**(pmxDoc, theTheme, theRect)

GIVEN: pmxDoc = the active view
theTheme = the theme to be processed

RETURN: theRect = the smallest rectangle enclosing all of the features in theTheme

The given and returned variables should be declared where first called as:

```
Dim pmxDoc As IMxDocument
Dim theTheme As Variant
Dim theRect As IEnvelope
```

5.1.8 Subroutine avInvalidateTOC

This subroutine enables the programmer to refresh the display of the Table of Contents.

The corresponding Avenue request is:

```
aView.InvalidateTOC(theName)
```

avInvalidateTOC

The call to this Avenue Wrap is:

Call **avInvalidateTOC**(theName)

GIVEN: theName = name of the theme or table in the Table of Contents to be refreshed. If NULL is specified, the entire Table of Contents will be refreshed.

RETURN: nothing

The given and returned variables should be declared where first called as:

```
Dim theName As Variant
```

5.1.9 Function avIsCoverage

This function enables the programmer to determine whether a layer (theme) is stored within a coverage, or not

The corresponding Avenue request is:

There is no corresponding Avenue request.

The call to this Avenue Wrap is:

avIsCoverage

theAnsw = **avIsCoverage**(theName)

GIVEN: theName = name of input object for which its layer database type is to be determined

RETURN: theAnsw = flag denoting whether the input object is a coverage layer (theme) or not (true = it is, false = it is not a coverage layer)

The given and returned variables should be declared where first called as:

```
Dim theName As Variant
Dim theAnsw As Boolean
```

5.1.10 Function avIsEditable

This function enables the programmer to determine if a layer (theme) or table is editable or not. Note that in:

- Avenue, the editability status is asked of a Field, FTab or VTab,
- Avenue Wrap, the editability status is asked of a theme, or table name.

To check editability of a Field use the Editable property on a field object:

Avenue	Avenue Wraps
col = theVTab.FindField("aField")	col = theVTab.FindField("aField")
if (col.IsEditable) then	Set pField = theVTab.Field (col)
	if (pField.Editable) then

The corresponding Avenue request is:

```
theAnsw = theFTab.IsEditable
```

The call to this Avenue Wrap is:

avIsEditable

theAnsw = **avIsEditable**(theName)

GIVEN: theName = name of theme or table for which its editability status is to be checked

LAYERS (THEMES) and TABLES

RETURN: theAnsw = editability status of the layer or table (true = is editable, false = is not editable)

The given and returned variables should be declared where first called as:
Dim theName As Variant
Dim theAnsw As Boolean

5.1.11 Function avIsFTheme

This function enables the programmer to determine whether a layer (theme) is of the feature layer type, or not

The corresponding Avenue request is:
theAnsw = aTheme.Is(FTheme)

avIsFTheme

The call to this Avenue Wrap is:
theAnsw = **avIsFTheme**(theName)

GIVEN: theName = name of input object for which its feature layer type is to be determined

RETURN: theAnsw = flag denoting whether the input object is a feature layer (theme), or not (true = it is, false = it is not a feature layer)

The given and returned variables should be declared where first called as:
Dim theName As Variant
Dim theAnsw As Boolean

5.1.12 Function avIsSDE

This function enables the programmer to determine whether a layer (theme) is stored within a SDE geodatabase, or not

The corresponding Avenue request is:
There is no corresponding Avenue request.

avIsSDE

The call to this Avenue Wrap is:
theAnsw = **avIsSDE**(theName)

GIVEN: theName = name of input object for which its layer database type is to be determined

RETURN: theAnsw = flag denoting whether the input object is a SDE layer (theme) or not (true = it is, false = it is not a SDE layer)

The given and returned variables should be declared where first called as:

```
Dim theName As Variant
Dim theAnsw As Boolean
```

5.1.13 Function avIsVisible

This function enables the programmer to determine if an object is visible or not. Note that in:

- Avenue, the visibility status is asked of an object (aTheme),
- Avenue Wrap, the visibility status is asked of a theme name (theName).

The corresponding Avenue request is:

```
theAnsw = aTheme.IsVisible
```

The call to this Avenue Wrap is:

```
theAnsw = avIsVisible(theName)
```


GIVEN: theName = name of input object for which its visibility status is to be determined

RETURN: theAnsw = visibility status of the layer (true = is editable, false = is not editable)

The given and returned variables should be declared where first called as:

```
Dim theName As Variant
Dim theAnsw As Boolean
```

5.1.14 Subroutine avSetEditable

This subroutine enables the programmer to start or stop the editing of a layer (theme) or table. In using this subroutine, note the following:

- For layers, editing is not terminated (the editor is not stopped), but rather, any buffered writes are flushed. This allows the user to undo an edit.
- For tables the editing is terminated.
- To terminate the editing on layers use the subroutine avStopEditing.

The corresponding Avenue request is:

```
aVTab.SetEditable(eStatus)                    '---FTab or VTab
```

LAYERS (THEMES) and TABLES

avSetEditable

The call to this Avenue Wrap is:

Call **avSetEditable**(pmxDoc, theTheme, eStatus)

GIVEN: pmxDoc = the active view
theTheme = name of the theme or table to be processed
eStatus = editing status. Specify:
True to start editing, or
False to stop editing

RETURN: nothing

The given and returned variables should be declared where first called as:

```
Dim pmxDoc As IMxDocument
Dim theTheme As Variant, Status As Boolean
```

5.1.15 Subroutine avSetEditable2

This subroutine enables the programmer to start or stop the editing of a layer (theme) and is similar to avSetEditable with the following exceptions:

- The argument list expects two items rather than three.
- An IFeatureLayer object is passed into the subroutine rather than the name of a layer.
- If the editing is terminated all edits which have been made to the layer will be committed to disk.

The corresponding Avenue request is:

```
aFTab.SetEditable(eStatus)                    '---FTab only
```

avSetEditable2

The call to this Avenue Wrap is:

Call **avSetEditable2**(pLayer, eStatus)

GIVEN: pLayer = layer object to be processed
eStatus = editing status. Specify:
True to start editing, or
False to stop editing

RETURN: nothing

The given and returned variables should be declared where first called as:

```
Dim pLayer As IFeatureLayer
Dim eStatus As Boolean
```

5.1.16 Subroutine avSetEditableTheme

This subroutine enables the programmer to stop the editing of a layer (theme) or set the type of task to be performed. This subroutine operates only on layers (themes), and can be used to display the handles of a feature. In using this subroutine note that the global variable ugSketch is used to keep track of whether a sketch session is active or not. If the value of ugSketch = 0, a sketch session is not active, if ugSketch = 1, a sketch session is active.

The corresponding Avenue request is:

```
theAnsw = aView.SetEditableTheme(aTheme)
```

The call to this Avenue Wrap is:

```
Call avSetEditableTheme(pmxDoc, theTheme, theType)
```

GIVEN:
- pmxDoc = the active view
- theTheme = name of the theme to be processed, if NULL, the editor will be stopped saving any edits that may have been made
- theType = the type of task to be performed. Specify:
  - 0 : to stop the current sketch session,
  - 1 : to modify feature,
  - 2 : to create new feature,
  - 9 : same as 0 except assign the current sketch geometry to the feature that is stored globally (ugLastFeatureSV), or
  - NULL : to do nothing

RETURN: nothing

The given and returned variables should be declared where first called as:

```
Dim pmxDoc As IMxDocument
Dim theTheme As Variant, theType As Variant
```

5.1.17 Subroutine avSetVisible

This subroutine enables the programmer to set the visibility status of an object.

The corresponding Avenue request is:

```
aTheme.SetVisible (aStatus)
```

The call to this Avenue Wrap is:

```
Call avSetVisible(theName, aStatus)
```

LAYERS (THEMES) and TABLES

GIVEN: theName = name of input object for which its visibility status is to be defined
aStatus = the visible state of the input object. Specify:
true for visible, or
false for not visible

RETURN: nothing

The given and returned variables should be declared where first called as:
Dim theName As Variant
Dim aStatus As Boolean

5.1.18 Subroutine avStartOperation

This subroutine enables the programmer to start an operation within an edit session.

The corresponding Avenue request is:
There is no corresponding Avenue request.

avStartOperation

The call to this Avenue Wrap is:
Call **avStartOperation**

GIVEN: nothing

RETURN: nothing

5.1.19 Subroutine avStopEditing

This subroutine enables the programmer to terminate the editing of a layer (theme) or table. This subroutine stops the editor committing any edits that may have been made to the layer (theme) or table, thus prohibiting the undo of said edits.

The corresponding Avenue request is:
aTable.StopEditing

avStopEditing

The call to this Avenue Wrap is:
Call **avStopEditing**

GIVEN: nothing

RETURN: nothing

The given and returned variables should be declared where first called as:

5.1.20 Subroutine avStopOperation

This subroutine enables the programmer to stop an operation within an edit session.

The corresponding Avenue request is:

There is no corresponding Avenue request.

The call to this Avenue Wrap is:

Call **avStopOperation**(oprMssg)

GIVEN: oprMssg = edit operation message that will appear to the right of the Undo menu item under the Edit menu item

RETURN: nothing

The given and returned variables should be declared where first called as:

```
Dim oprMssg As Variant
```

NOTE: When the editor is stopped it is not possible to use the Undo command under the Edit menu item, so that, if the Undo command is to be used, the Editor must be active (in use). The subroutine avStopEditing merely signals that an edit operation has been completed and that the operation should be added to the Undo list. It does not terminate the edit session and as such the editor is left in an active state and the user is able to employ the Undo command, if need be.

5.1.21 Subroutine avThemeInvalidate

This subroutine enables the programmer to redraw either the entire display or only that of a theme.

The corresponding Avenue request is:

```
aTheme.Invalidate(rdStatus)
```

The call to this Avenue Wrap is:

Call **avThemeInvalidate**(pmxDoc, theTheme, rdStatus)

LAYERS (THEMES) and TABLES

GIVEN: pmxDoc = the active view
theTheme = name of theme to be processed
rdStatus = redraw status. Specify:
True to redraw entire view, or
False to redraw the theme only

RETURN: nothing

The given and returned variables should be declared where first called as:
Dim pmxDoc As IMxDocument
Dim theTheme As Variant, rdStatus As Boolean

5.1.22 Subroutine avThemeSetName

This subroutine enables the programmer to set the name or alias for a layer (theme). The programmer is given the option whether to update, or not the Table of Contents (TOC) when the alias is assigned to the layer (theme). This is controlled by the given variable updateTOC. If many layers (themes) are to be modified, it is better to update the TOC at the end of the modifications, rather than after every single modification.

The corresponding Avenue request is:
theTheme.SetName(newName)

avThemeSetName

The call to this Avenue Wrap is:
Call **avThemeSetName**(pmxDoc, theTheme, newName, updateTOC)

GIVEN: pmxDoc = the active view
theTheme = name of theme to be processed
newName = new name or alias to be assigned to the layer (theme)
updateTOC = update status True = update the TOC
False = do not update the TOC

RETURN: nothing

The given and returned variables should be declared where first called as:
Dim pmxDoc As IMxDocument
Dim theTheme As Variant, newName As Variant
Dim updateTOC As Boolean

5.1.23 Function avVTabExport

This function enables the programmer to export an existing table in order to create a new dBase or text file type. In using this function, note the following:

1. In specifying the name of the new table (aFileName), if the name does not contain a complete path name, the current working directory will be used. Some examples of a table name include:

   c:\project\test\atable c:\project\test\atable.dbf
   atable atable.dbf

   The name may or may not contain the extension .dbf or .txt
2. If the table to be created, aFileName, exists on disk, it will be deleted, before the exporting is performed, without informing the user/developer.
3. If selected records are to be exported, and there are no selected records, the entire table will be exported. Selected records must be selected programmatically prior to invoking this function.
4. If the table can not be exported for any reason what so ever, avVTabExport will be set to NOTHING.

The corresponding Avenue request is:

```
newTable = aTable.Export (aFileName, aClass, SelRecrds)
```

The call to this Avenue Wrap is:

avVTabExport

```
newTable = avVTabExport(aTable, aFileName, aClass, _
                        SelRecrds)
```

GIVEN:
- aTable = the name of the table to be exported.
- aFileName = name of the table to be created.
- aClass = the type of table to be created. Specify: "dBase" or "TEXT".
- SelRecrds = indicator whether selected, or all records of aTable are to be exported. Specify:
  - true to export selected records only.
  - false to export all records.

RETURN:
- newTable = the table object that is created.

The given and returned variables should be declared where first called as:

```
Dim aTable, aFileName, aClass As String
Dim SelRecrds As Boolean
Dim newTable As ITable
```

LAYERS (THEMES) and TABLES

avVTabMake

5.1.24 Function avVTabMake

This function enables the programmer to open an existing dBase or text type file. In using this function, note the following:

1. If the name of the given file (aFileName) to be opened does not contain a complete pathname the current working directory will be used. Some examples of name include:

   c:\project\test\atable c:\project\test\atable.dbf
   atable atable.dbf

   The extension .dbf or .txt indicates the type of table to be opened.
2. If aFileName does not contain an extension, the procedure assumes that a dBase file is to be opened .
3. The forWrite and skipFirst given arguments are ignored, as of this version, and as such they have no impact upon this procedure .
4. If aFileName can not be opened, avVTabMake will be set to NOTHING.
5. After the file has been opened with the subject function, use the function avAddDoc to add the file's table into the Table of Contents.

The corresponding Avenue request is:

```
aVTab = VTab.Make (aFileName, forWrite, skipFirst)
```

The call to this Avenue Wrap is:

```
aVTab = avVTabMake(aFileName, forWrite, skipFirst, _
                   aClass)
```

GIVEN:
- aFileName = name of the table to be opened.
- forWrite = indicates if the table is to be made editable once it is opened (see Note 3).
- skipFirst = indicates if the first record in the table is to be ignored (see Note 3).
- aClass = type of table to be created. Specify: "dBase" or "TEXT". If this argument is specified it will override any extension that may appear in aFileName

RETURN:
- aVTab = table object that is created

The given and returned variables should be declared where first called as:

```
Dim aFileName As String
Dim forWrite As Boolean, skipFirst As Boolean, aClass As String
Dim aVTab As ITable
```

5.1.25 Function avVTabMakeNew

This function enables the programmer to create a new dBase or text file type table. In using this function, note the following:

1. In specifying the name of the new table, if the name does not contain a complete path name, the current working directory will be used. Some examples of a table name include:

   c:\project\test\atable c:\project\test\atable.dbf
   atable atable.dbf

   The name may or may not contain the extension .dbf or .txt
2. Two fields called OID and ID will be created by this routine.
3. The function avAddDoc can be used to add the table to the map, if need be.
4. If the table to be created exists on disk, the routine will abort, and the existing table will not be overwritten.

The corresponding Avenue request is:

```
theNewTable = VTab.MakeNew(aFileName, aClass)
```

The call to this Avenue Wrap is:

```
Set theNewTable = avVTabMakeNew(aFileName, aClass)
```


GIVEN: aFileName = name of the table to be created (see Note 1).
aClass = type of table to be created. Specify: dBase or TEXT

RETURN: theNewTable = table object that is created

The given and returned variables should be declared where first called as:

```
Dim aFileName, aClass As String
Dim theNewTable As ITable
```

5.1.26 Function FindLayer

This function enables the programmer to find a layer (theme) in a map and will return an ILayer object if the specified layer is found. If the layer can not be found, the returned value will be set to the object, Nothing. Note that the functions FindTheme and avFindDoc are similar to this function and may be of interest to the reader. In using this function, note the following:

LAYERS (THEMES) and TABLES

1. The global variable, ugLayer, or Avenue Wraps property, avwraps.Layer, will contain an ILayer reference to the layer that is found, if a layer is not found this variable will be set to Nothing.
2. The global variable, ugTable, or Avenue Wraps property, avwraps.Table, is initialize to the object, Nothing, when this function is called. If the avFindDoc function is used and a table is found, ugTable and avwraps.Table will contain an IStandaloneTable reference to the table.
3. The global variable, ugLayerStrg, or Avenue Wraps propery, avwraps.LayerString, will contain the name of the layer that is found, if a layer is not found this variable to be equal to a single blank character.
4. The global variable, ugLayerIndx, or Avenue Wraps propery, avwraps.LayerIndex, will contain the index value for the location of the layer in the IMap.Layer property. So that the statement:

   Set theLayer = aMap.Layer(ugLayerIndx), or

   Set theLayer = aMap.Layer(avwraps.LayerIndex)

   could be used to get an ILayer object. Index values begin at 0.
5. If a feature layer is found, the global variable, ugpFCls, or Avenue Wraps propery, avwraps.FeatureClass, will contain an IFeatureClass reference to the layer, if a layer is not found this variable will be set to Nothing.
6. If a feature layer is found, the global variable, ugWrkSpcType, or Avenue Wraps propery, avwraps.WrkSpcType, will contain a value representing the type of workspace that is associated with the layer that was found. This value and its representation is as follows:

   0 A File-based workspace. e.g. coverages, shapefiles

   1 A True Geodatabases that are local to your machine, e.g. Access

   2 A Geodatabase that requires a remote connection. e.g. SDE, OLE DB
7. If a feature layer is found, the global variable, ugWrkSpcDesc, or Avenue Wraps propery, avwraps.WrkSpcDesc, will contain a text string representing the description of the workspace that is associated with the layer that was found. This text string and its representation is as follows:

ArcInfo Workspace	Denotes an Arc/Info Coverage Layer
PC ArcInfo Workspace	Denotes a PC Arc/Info Coverage Layer
CAD Workspace	Denotes a DXF, DWG, etc. Layer
Personal Geodatabase	Denotes a Personal Geodatabase
Shapefiles	Denotes a Shapefile Layer
UNKNOWN	If the Layer was not found

   Using the global variable or Avenue Wraps property is a good way of ascertaining what type of layer is being processed.

LAYERS (THEMES) and TABLES

The corresponding Avenue request is:

aTheme = aView.FindTheme(theName)

The call to this Avenue Wrap is:

Set theLayer = **FindLayer**(aMap, theName)

FindLayer

GIVEN: aMap = map to be searched
theName = name of the layer to be found

RETURN: theLayer = the layer in the map

The given and returned variables should be declared where first called as:

```
Dim aMap As IMap, theName As Variant
Dim theLayer As ILayer
```

5.1.27 Function FindTheme

This function enables the programmer to find a layer (theme) in a map. This is similar to FindLayer with the exception, it does not return an ILayer object but rather, it returns a variable of Variant type. If the layer to be found can not be found, the returned value will be a single blank character. The reader is referred to the notes that appear in the FindLayer description since they hold true for this function, as well.

The corresponding Avenue request is:

aTheme = aView.FindTheme(theName)

The call to this Avenue Wrap is:

theTheme = **FindTheme**(aMap, theName)

FindTheme

GIVEN: aMap = map to be searched
theName = name of layer to be found

RETURN: theTheme = the layer in the map

The given and returned variables should be declared where first called as:

```
Dim aMap As IMap, theName As Variant
Dim theTheme As Variant
```

Example For example purposes, let us assume that:

- We are developing an application using the Avenue Wraps DLL,
- We have a layer called Theme1 and we wish to determine what type of layer it is:

LAYERS (THEMES) and TABLES

The code shown below could be used to accomplish the above task:

```
Dim pMxApp As IMxApplication
Dim pmxDoc As IMxDocument
Dim pActiveView As IActiveView
Dim pMap As IMap
Dim theTheme As Variant

'
'   ---Get the active view
Call avGetActiveDoc(pMxApp, pmxDoc, pActiveView, pMap)
'
'   ---Find the theme to be examined
theTheme = FindTheme(pMap, "Theme1")
'
'   ---Use the Avenue Wraps workspace description property
'   ---to determine the type of layer we have
If (UCase(avwraps.WrkSpcDesc) = "ARCINFO WORKSPACE") Then
   MsgBox "An ArcInfo Workspace was found."
'
Elseif(UCase(avwraps.WrkSpcDesc) = "PC ARCINFO WORKSPACE") Then
   MsgBox "A PC ArcInfo Workspace was found."
'
Elseif(UCase(avwraps.WrkSpcDesc) = "CAD WORKSPACE") Then
   MsgBox "A CAD drawing was found."
'
Elseif(UCase(avwraps.WrkSpcDesc) = "PERSONAL GEODATABASE") Then
   MsgBox "A Personal GeoDatabase was found."
'
Elseif(UCase(avwraps.WrkSpcDesc) = "SHAPEFILES") Then
   MsgBox "A Shapefile was found."
'
Elseif(UCase(avwraps.WrkSpcDesc) = "UNKNOWN") Then
   MsgBox "The theme does not exist, or" + _
          "it is not a feature layer."
'
End If
```

5.2 Theme or Table Attribute Field Related Avenue Wraps

ATTRIBUTE FIELDS

5.2.1 Function avAddFields

This function enables the programmer to add attribute fields in a layer (theme) or table. In using this function, note the following:

1 In order to add fields into a layer or table, the editor can not be in an edit state. Thus this function will stop the editor, saving any changes that may have been made, prior to adding the fields.

2. In both Avenue and Avenue Wraps, the items in the given collection (list) are objects, not strings. Thus, before calling this function, the given argument, theFields, must be populated with items declared as iField. The Avenue Wrap avFieldMake may be used to create the iField items.

The corresponding Avenue request is:

```
anFTab.AddFields(theFields)
```

The call to this Avenue Wrap is:

```
errFlag = avAddFields(pmxDoc, theTheme, theFields)
```

avAddFields

GIVEN:	pmxDoc	= the active view
	theTheme	= the theme or table to be processed
	theFields	= list of fields to be added (see Note 2 above)
RETURN:	errFlag	= error flag (0 = no error, 1 = error)

The given and returned variables should be declared where first called as:

```
Dim pmxDoc As IMxDocument
Dim theTheme As Variant
Dim theFields As New Collection
Dim errFlag As Integer
```

5.2.2 Function avFieldGetType

This function enables the programmer to get the type of field that a field object is. In using this function, one of the numbers shown below will be returned to indicate the type of field object that was processed:

0 : Small Integer	1 : Long Integer
2 : Single-precision float	3 : Double-precision float
4 : String	5 : Date
6 : Long Integer denoting the OID	7 : Geometry
8 : Blob	

ATTRIBUTE FIELDS

avFieldGetType

The corresponding Avenue request is:

```
theFieldType = aField.GetType
```

The call to this Avenue Wrap is:

```
theFieldType = avFieldGetType(pField)
```

GIVEN: pField = field object to be processed

RETURN: theFieldType = numeric value denoting type of field (see above)

The given and returned variables should be declared where first called as:

```
Dim pField As iField
Dim theFieldType As esriFieldType
```

5.2.3 Function avFieldMake

This function enables the programmer to create a field that can be added to a layer (theme) or table. In using this function, note the following:

1. Specify the key word below for the argument aFieldType to denote the indicated type of field object:

BYTE	Small Integer	CHAR	String
DATE	Date	DECIMAL	Single
DOUBLE	Double	FLOAT	Single
ISODATE	Date	ISODATETIME	Date
ISOTIME	Date	LOGICAL	String
LONG	Integer	MONEY	Double
SHORT	Small Integer	BLOB	Blob
VCHAR	String		

2. This routine can not be used to create a geometry field.
3. The first 10 characters in aName are used. Use avSetAlias after the table has been created to define the desired full field name.

The corresponding Avenue request is:

```
theNewField = Field.Make(aName, aFieldType, nchr, ndr)
```

avFieldMake

The call to this Avenue Wrap is:

```
Set theNewField = avFieldMake(aName, aFieldType, nchr, ndr)
```

GIVEN: aName = name of field to be created
aFieldType = type of field to be created (see Note 1 above)
nchr = total character width of field including decimal point and negative sign, if they are to appear in the field

ATTRIBUTE FIELDS

ndr = number of digits to the right of the decimal point. Specify 0 for non-numeric fields

RETURN: theNewField = field object that was created

The given and returned variables should be declared where first called as:

```
Dim aName, aFieldType As String, nchr As Long, ndr As Long
Dim theNewField As IFieldEdit
```

5.2.4 Function avGetAlias

This subroutine enables the programmer to retrieve the alias that has been assigned to a field for a layer or a table. The current layer or table is processed. The global variables ugLayer and ugTable represent the current layer and table, while the Avenue Wraps properties, avwraps.Layer and avwraps.Table, represent the current layer and table when using the Avenue Wraps DLL. The subroutines avGetFTab or avGetVTab can also be used to establish the current layer or table.

The corresponding Avenue request is:

```
aField.GetAlias(anAlias)
```

The call to this Avenue Wrap is:

```
anAlias = avGetAlias(col)
```

avGetAlias

GIVEN: col = index value representing the field that an alias is to be assigned to

RETURN: anAlias = the string representing the alias to be assigned to the field

The given and returned variables should be declared where first called as:

```
Dim col As Long
Dim anAlias As String
```

5.2.5 Subroutine avGetFields

This subroutine enables the programmer to get a list of attribute field names for a layer (theme) or table. These are not the alias names for the fields.

The corresponding Avenue request is:

```
theList = aVTab.GetFields
```

ATTRIBUTE FIELDS

avGetFields

The call to this Avenue Wrap is:

Call **avGetFields**(theVTab, theList)

GIVEN: theVTab = field list for the theme or table

RETURN: theList = list of field names for an attribute table

The given and returned variables should be declared where first called as:

```
Dim theVTab As IFields
Dim theList As New Collection
```

5.2.6 Function avGetPrecision

This function enables the programmer to get the decimal precision for a field. This is the number of digits to the right of the decimal point. This function always returns zero for fields contained in a personal geodatabase.

The corresponding Avenue request is:

aNumb = aField.GetPrecision

avGetPrecision

The call to this Avenue Wrap is:

aNumb = **avGetPrecision**(theVTab, fieldIndex)

GIVEN: theVTab = field list for the theme or table
fieldIndex = index of the field to be processed

RETURN: aNumb = decimal precision for the field

The given and returned variables should be declared where first called as:

```
Dim theVTab As IFields
Dim fieldIndex As Long
Dim aNumb As Long
```

5.2.7 Subroutine avGetUniqueValues

This subroutine enables the programmer to get a list of the unique values for a specific field in a theme (layer) or table.

The corresponding Avenue request is:

There is no corresponding Avenue request.

The call to this Avenue Wrap is:

Call **avGetUniqueValue**(pmxDoc, theTheme, aField, aList)

ATTRIBUTE FIELDS

avGetUniqueValues

GIVEN: pmxDoc = the active view
theTheme = the theme or table to be processed
aField = field for which unique values are desired

RETURN: aList = list of unique values (this list is not sorted)

The given and returned variables should be declared where first called as:

```
Dim pmxDoc As IMxDocument
Dim theTheme As Variant, aField As String
Dim aList As New Collection
```

5.2.8 Function avRemoveFields

This function enables the programmer to remove attribute fields from a layer (theme) or table. In using this function, note the following:

1. In order to remove fields from a layer or table, the editor can not be in an edit state. Thus, this routine will stop the editor, saving any changes that may have been made, prior to removing the fields
2. If an invalid index value appears in the list, -1, it will be ignored (an error is not generated)
3. Do not use this routine to delete the SHAPE field
4. The items in the given argument list, theFields, are numeric index values (not objects) for the fields to be deleted.

The corresponding Avenue request is:

```
aVTab.RemoveFields(theFields)
```

The call to this Avenue Wrap is:

```
errFlag = avRemoveFields(pmxDoc, theTheme, theFields)
```

avRemoveFields

GIVEN: pmxDoc = the active view
theTheme = the theme or table to be processed
theFields = list of fields to be removed (see Note 4 above)

RETURN: errFlag = error flag (0 = no error, 1 = error detected)

The given and returned variables should be declared where first called as:

```
Dim pmxDoc As IMxDocument
Dim theTheme As Variant, theFields As New Collection
Dim errFlag As Integer
```

ATTRIBUTE FIELDS

avSetAlias

5.2.9 Subroutine avSetAlias

This subroutine enables the programmer to assign an alias to a field for a layer or a table. The current layer or table is processed. The global variables ugLayer and ugTable represent the current layer and table, while the Avenue Wraps properties, avwraps.Layer and avwraps.Table, represent the current layer and table when using the Avenue Wraps DLL. The subroutines avGetFTab or avGetVTab can also be used to establish the current layer or table.

The corresponding Avenue request is:

aField.SetAlias(anAlias)

The call to this Avenue Wrap is:

Call **avSetAlias**(col, anAlias)

GIVEN: col = index value representing the field that an alias is to be assigned to

anAlias = the string representing the alias to be assigned to the field

RETURN: nothing

The given and returned variables should be declared where first called as:

```
Dim col As Long
Dim anAlias As String
```

5.3 Theme or Table Record Related Avenue Wraps

LAYER (THEME) and TABLE RECORDS

5.3.1 Function avAddRecord

This function enables the programmer to add a record into a layer (theme) or table.

The corresponding Avenue request is:

```
theRecordID = aVTab.AddRecord
```

The call to this Avenue Wrap is:

avAddRecord

```
theRecordID = avAddRecord(pmxDoc, theTheme)
```

GIVEN: pmxDoc = the active view
theTheme = the theme or table to be processed

RETURN: theRecordID = the ID of the record that was added. If a record can not be added it will be -1.

The given and returned variables should be declared where first called as:

```
Dim pmxDoc As IMxDocument
Dim theTheme As Variant
Dim theRecordID As Long
```

5.3.2 Subroutine avGetFTab

This subroutine enables the programmer to get the attribute table, feature class and associated layer (theme) for a specified theme. Note that if a table, rather than a theme, is specified, the values for the theFeatureClass and theLayer arguments will be set to Nothing, while theFTab object will reflect the attributes for the table.

The corresponding Avenue request is:

```
theFTab = aTheme.GetFTab
```

The call to this Avenue Wrap is:

avGetFTab

```
Call avGetFTab(pmxDoc, theTheme, _
               theFTab, theFeatureClass, theLayer)
```

GIVEN: pmxDoc = the active view
theTheme = the theme or table to be processed

LAYER (THEME) and TABLE RECORDS

RETURN: theFTab = the attribute table for the theme
theFeatureClass = the feature class for the theme
theLayer = the associated layer for the theme

The given and returned variables should be declared where first called as:

```
Dim pmxDoc As IMxDocument, theTheme As Variant
Dim theFTab As IFields, theFeatureClass As IFeatureClass
Dim theLayer As IFeatureLayer
```

5.3.3 Subroutine avGetFTabIDs

This subroutine enables the programmer to get a list of the object identification numbers (OIDs) for a layer (theme).

The corresponding Avenue request is:

There is no corresponding Avenue request.

avGetFTabIDs

The call to this Avenue Wrap is:

Call **avGetFTabIDs**(pmxDoc, theTheme, theRecsList)

GIVEN: pmxDoc = the active view
theTheme = the theme or table to be processed

RETURN: theRecsList = the list of OIDs for the layer (theme)

The given and returned variables should be declared where first called as:

```
Dim pmxDoc As IMxDocument, theTheme As Variant
Dim theRecsList As New Collection
```

5.3.4 Subroutine avGetFTabIDs2

This subroutine enables the programmer to build an array which contains the object identification numbers (OIDs) for all of the features in a layer (theme). This subroutine is identical to avGetFTabIDs with the exception that an array is passed back rather than a collection. In using this subroutine, note the following:

1. The first OID appears in the first element of the array and can be accessed as shown below:

   ```
   firstOID = theRecsArray(1)
   ```

2. To determine the number of elements in the array use the function, UBound, as shown below:

   ```
   totalIDs = UBound(theRecsArray)
   ```

LAYER (THEME) and TABLE RECORDS

3. If the array can not be built, the number of elements in the array will be one and the value of the first element in the array will be set to -1.
4. Arrays process faster than lists, as such use this subroutine rather than avGetFTabIDs when the layer contains a large number of features.

The corresponding Avenue request is:

There is no corresponding Avenue request.

The call to this Avenue Wrap is:

avGetFTabIDs2

Call **avGetFTabIDs2**(pmxDoc, theTheme, theRecsArray)

GIVEN: pmxDoc = the active view
theTheme = the theme or table to be processed

RETURN: theRecsArray = the array of OIDs for the layer (theme)

The given and returned variables should be declared where first called as:

```
Dim pmxDoc As IMxDocument, theTheme As Variant
Dim theRecsArray() As Long
```

5.3.5 Function avGetNumRecords

This function enables the programmer to get the number of records in a layer (theme), or table.

The corresponding Avenue request is:

```
numRecs = aVTab.GetNumRecords
```

The call to this Avenue Wrap is:

avGetNumRecords

numRecs = **avGetNumRecords**(pmxDoc, theTheme)

GIVEN: pmxDoc = the active view
theTheme = the theme or table to be processed

RETURN: numRecs = number of records in the theme or table

The given and returned variables should be declared where first called as:

```
Dim pmxDoc As IMxDocument
Dim theTheme As Variant
Dim numRecs As Long
```

LAYER (THEME) and TABLE RECORDS

5.3.6 Subroutine avGetVTab

This subroutine enables the programmer to get the attribute table for a layer (theme) or table.

The corresponding Avenue request is:

```
theVTab = aTable.GetVTab
```

avGetVTab

The call to this Avenue Wrap is:

```
Call avGetVTab(pmxDoc, theTheme, theVTab)
```

GIVEN: pmxDoc = the active view
theTheme = the theme or table to be processed

RETURN: theVTab = the attribute table for the theme or table

The given and returned variables should be declared where first called as:

```
Dim pmxDoc As IMxDocument, theTheme As Variant
Dim theVTab As IFields
```

5.3.7 Subroutine avGetVTabIDs

This subroutine enables the programmer to get a list of the object identification numbers (OIDs) for a table.

The corresponding Avenue request is:

There is no corresponding Avenue request.

avGetVTabIDs

The call to this Avenue Wrap is:

```
Call avGetVTabIDs(pmxDoc, theTable, theRecsList)
```

GIVEN: pmxDoc = the active view
theTable = the table to be processed

RETURN: theRecsList = the list of object identification numbers (OIDs) for the table

The given and returned variables should be declared where first called as:

```
Dim pmxDoc As IMxDocument, theTable As Variant
Dim theRecsList As New Collection
```

5.3.8 Subroutine avGetVTabIDs2

LAYER (THEME) and TABLE RECORDS

This subroutine enables the programmer to build an array which contains the object identification numbers (OIDs) for all of the records in a table. This subroutine is identical to avGetVTabIDs with the exception that an array is passed back rather than a collection. In using this subroutine, note the following:

1. The first OID appears in the first element of the array and can be accessed as shown below:

   firstOID = theRecsArray(1)

2. To determine the number of elements in the array use the function, UBound, as shown below:

   totalIDs = UBound(theRecsArray)

3. If the array can not be built, the number of elements in the array will be one and the value of the first element in the array will be set to -1.
4. Arrays process faster than lists, as such use this subroutine rather than avGetVTabIDs when the table contains a large number of records.

The corresponding Avenue request is:

There is no corresponding Avenue request.

The call to this Avenue Wrap is:

Call **avGetVTabIDs2**(pmxDoc, theTable, theRecsArray)

avGetVTabIDs2

GIVEN: pmxDoc = the active view
theTable = the table to be processed

RETURN: theRecsArray = the array of OIDs for the table

The given and returned variables should be declared where first called as:

```
Dim pmxDoc As IMxDocument, theTable As Variant
Dim theRecsArray() As Long
```

5.3.9 Function avReturnValue

This subroutine enables the programmer to retrieve a value from a specific field in a specific row of a layer (theme) or table. In using this subroutine, note the following:

1. Do not use this function to retrieve geometry from the SHAPE field. Use this subroutine to retrieve attribute information only (see avGetFeature for how to extract the geometry of a feature).

LAYER (THEME) and TABLE RECORDS

avReturnValue

2. If an error is detected avReturnValue will be set to NULL.

The corresponding Avenue request is:

anObj = aVTab.ReturnValue (aField, aRecord)

The call to this Avenue Wrap is:

anOb = **avReturnValue**(pmxDoc, theTheme, aField, aRecordj)

GIVEN:	pmxDoc	= the active view
	theTheme	= the theme or table to be processed
	aField	= field to be written to
	aRecord	= record of theme or table to be processed
RETURN:	anObj	= object to be stored (attribute information only, no geometry)

The given and returned variables should be declared where first called as:

```
Dim pmxDoc As IMxDocument, theTheme As Variant
Dim aField As Long
Dim aRecord As Long
Dim anObj As Variant
```

5.3.10 Subroutine avSetValue

This subroutine enables the programmer to store a value in a specific field of a specific row of a layer (theme) or table. In using this subroutine, note the following:

1. Do not use this subroutine to store geometry in the SHAPE field. Use this subroutine to store attribute information only.
2. To store geometry in the SHAPE field, use the subroutine avSetValueG.
3. While in Avenue the same request may be used to write attribute information and geometry, there are two distinct Avenue Wrap requests because of there are two distinct interfaces in ArcObjects.
4. While the Avenue request operates on an FTab or VTab, the Avenue Wrap operates on a layer (theme) or table name.
5. This procedure does not write the record, aRecord, to disk until the procedure is called with the argument, anObj, set to "StoreRec". This is done to eliminate multiple disk writes thereby yielding increased performance (see the avSetValueG description for an alternative to calling this subroutine with the argument, anObj, set to "StoreRec").

LAYER (THEME) and TABLE RECORDS

6. When the argument, anObj, is set to "StoreRec", the argument, aField, is ignored.

The corresponding Avenue request is:

```
aVTab.SetValue(aField, aRecord, anObj)
```

The call to this Avenue Wrap is:

avSetValue

```
Call avSetValue(pmxDoc, theTheme, aField, aRecord, anObj)
```

GIVEN:

pmxDoc	=	the active view
theTheme	=	the theme or table to be processed
aField	=	field to be written to
aRecord	=	record of theme or table to be processed
anObj	=	object to be stored (attribute information only, no geometry)

RETURN: nothing

The given and returned variables should be declared where first called as:

```
Dim pmxDoc As IMxDocument
Dim theTheme As Variant
Dim aField, aRecord As Long
Dim anObj As Variant
```

5.3.11 Subroutine avSetValueG

This subroutine enables the programmer to store a shape in the SHAPE field of a specific row of a layer (theme). In using this subroutine, note the following:

1. Do not use this subroutine to store attribute information. Use this subroutine to store geometry in the SHAPE field only.
2. To store attribute information, use the subroutine avSetValue.
3. While in Avenue the same request may be used to write attribute information and geometry, there are two distinct Avenue requests because of there are two distinct interfaces in ArcObjects.
4. While the Avenue request operates on an FTab or VTab, the Avenue Wrap operates on a layer (theme) name.
5. This procedure writes the record to disk after the shape has been stored.
6. Calling this procedure after calling avSetValue eliminates the need to call avSetValue with the anObj argument set to "StoreRec" because this procedure writes the record to disk.

LAYER (THEME) and TABLE RECORDS

avSetValueG

The corresponding Avenue request is:

There is no corresponding Avenue request.

The call to this Avenue Wrap is:

Call **avSetValueG**(pmxDoc, theTheme, aField, aRecord, aShape)

GIVEN:	pmxDoc	= the active view
	theTheme	= the theme or table to be processed
	aField	= field to be written to
	aRecord	= record of theme or table to be processed
	aShape	= shape to be stored (geometry only, no attribute information only)

RETURN: nothing

The given and returned variables should be declared where first called as:

```
Dim pmxDoc As IMxDocument
Dim theTheme As Variant
Dim aField As Long, aRecord As Long
Dim aShape As IGeometry
```

5.4 Feature Handling Related Avenue Wraps

FEATURE HANDLING

5.4.1 Subroutine avFeatureInvalidate

This subroutine enables the programmer to redraw a feature.

The corresponding Avenue request is:

There is no corresponding Avenue request.

The call to this Avenue Wrap is:

Call **avFeatureInvalidate**(pmxDoc, theFeature)

avFeatureInvalidate

GIVEN: pmxDoc = the active view
theFeature = the feature to be redrawn

RETURN: Nothing

The given and returned variables should be declared where first called as:

```
Dim pmxDoc As IMxDocument
Dim theFeature As IFeature
```

5.4.2 Subroutine avGetFeature

This subroutine enables the programmer to get a feature given a layer (theme) and an object ID. Use the ***Shape*** property of the IFeature object to get the feature's geometry and the ***Value*** property to simulate the ReturnValue Avenue request (see page D-2 for an example). Use the subroutine avGetTableRow when processing a table (VTab).

The corresponding Avenue request is:

```
theFeature = aFTab.ReturnValue ("shape", theObjId)
```

The call to this Avenue Wrap is:

Call **avGetFeature**(pmxDoc, theTheme, theObjId, theFeature)

avGetFeature

GIVEN: pmxDoc = the active view
theTheme = name of the theme to be processed
theObjId = the object id of the desired feature

RETURN: theFeature = the feature

The given and returned variables should be declared where first called as:

```
Dim pmxDoc As IMxDocument
Dim theTheme As Variant, theObjId As Long
Dim theFeature As IFeature
```

FEATURE HANDLING

With Avenue

```
aVTab = aTable.GetVTab
colA = aVTab.FindField("area")
theArea = aVTab.ReturnValue(colA, rec)
```

With Avenue Wraps

```
Dim pmxDoc As esricore.IMxDocument
Dim aTable As Variant
Dim aVTab As esricore.IFields
Dim rec As Long, colA As Long
Dim pRow As esricore.IRow
Dim theArea As Double
Call avGetVTab(pmxDoc, aTable, aVTab)
Call avGetTableRow(pmxDoc, aTable, rec, pRow)
colA = aVTab.FindField("area")
theArea = pRow.Value(colA)
```

Sample Code illustrating ReturnValue Simulation on a VTab

5.4.3 Subroutine avGetFeatData

This subroutine enables the programmer to get the feature data of a given layer (theme) and object ID.

The corresponding Avenue request is:

There is no corresponding Avenue request.

The call to this Avenue Wrap is:

avGetFeatData

Call **avGetFeatData**(pmxDoc, theTheme, theObjId, _
theFeature, theShape, shapeType)

GIVEN:	pmxDoc	= the active view
	theTheme	= name of the theme to be processed
	theObjId	= the object id of the desired feature
RETURN:	theFeature	= the feature
	theShape	= the geometry of a feature
	shapeType	= the shape type of a feature

The given and returned variables should be declared where first called as:

```
Dim pmxDoc As IMxDocument
Dim theTheme As Variant
Dim theObjId As Long
Dim theFeature As IFeature
Dim theShape As IGeometry
Dim shapeType As esriGeometryType
```

FEATURE HANDLING

5.4.4 Subroutine avGetGeometry

This subroutine enables the programmer to get the geometry of a feature given its layer (theme) and object ID.

The corresponding Avenue request is:

There is no corresponding Avenue request.

The call to this Avenue Wrap is:

avGetGeometry

Call **avGetGeometry**(pmxDoc, theTheme, theObjId, theShape)

GIVEN: pmxDoc = the active view
theTheme = name of the theme to be processed
theObjId = the object id of the desired feature

RETURN: theShape = the geometry of a feature

The given and returned variables should be declared where first called as:

```
Dim pmxDoc As IMxDocument
Dim theTheme As Variant
Dim theObjId As Long
Dim theShape As IGeometry
```

5.4.5 Subroutine avGetTableRow

This subroutine enables the programmer to get the IRow object given the name of a table and an object ID. Use avGetFeature when processing a theme.

The corresponding Avenue request is:

There is no corresponding Avenue request.

The call to this Avenue Wrap is:

avGetTableRow

Call **avGetTableRow**(pmxDoc, theTheme, theObjId, theRow)

GIVEN: pmxDoc = the active view
theTable = name of the table to be processed
theObjId = the object id of the desired record

RETURN: theRow = the IRow object

The given and returned variables should be declared where first called as:

```
Dim pmxDoc As IMxDocument
Dim theTheme As Variant
Dim theObjId As Long
Dim theRow As IRow
```

FEATURE HANDLING

avUpdateAnno

5.4.6 Subroutine avUpdateAnno

This subroutine enables the programmer to apply a transformation to an existing annotation feature (a feature in a feature annotation layer). In using this subroutine, note the following:

1. The rotation angle is added to the existing angle of the annotation (positive value denotes a counter-clockwise rotation, while a negative value denotes a clockwise rotation).
2. A scale factor greater than 1.0 increases the size of the annotation, while a value less than 1.0 decreases the size.
3. The X scale factor is always used in the scaling process, the Scale method does not seem to work as it should on Annotation features when the X and Y scale factors are different.
4. The layer that the feature resides in must be in an editable state.

The corresponding Avenue request is:

There is no corresponding Avenue request.

The call to this Avenue Wrap is:

```
Call avUpdateAnno(pFeature, oldX, oldY, newX, newY, _
                  rotang, scaleX, scaleY, newFeature)
```

GIVEN:	pFeature	=	the annotation feature to be modified
	oldX, oldY	=	the coordinates of the feature's control point
	newX, newY	=	the new coordinates of the feature's control point
	rotang	=	the rotation angle in degrees to be added to the existing angle of the feature
	scaleX	=	the X axis scale factor (greater than 0.0)
	scaleY	=	the Y axis scale factor (greater than 0.0)
RETURN:	newFeature	=	the new feature reflecting the transformation

The given and returned variables should be declared where first called as:

```
Dim pFeature As IFeature
Dim oldX, oldY, newX, newY, rotang, scaleX, scaleY As Double
Dim newFeature As IFeature
```

CALCULATE QUERY and SUMMARIZE

5.5 Calculating, Querying and Summarizing Layers (Themes) and Tables Related Avenue Wraps

5.5.1 Function avCalculate

avCalculate

This function enables the programmer to apply a calculation to a set of selected records for a specified field in a layer (theme) or table. If no records are selected, the entire layer (theme) or table is processed. Sample calculation strings are shown below. Note how the field names are handled depending upon the type of field being processed. The source code listing, presented in Appendix D, contains more detailed information pertaining to this function.

- Shapefile and Personal Geodatabase String field calculation:
  ```
  aCalcString = """abcd"""
  ```
- Shapefile and Personal Geodatabase Numeric field calculation:
  ```
  aCalcString = "([ID] - " + CStr(i) + ")"
  ```

The corresponding Avenue request is:

```
errFlag = aVTab.Calculate(aCalcString, aField)
```

The call to this Avenue Wrap is:

```
errFlag = avCalculate(pmxDoc, theTheme, aCalcString, aField)
```

GIVEN:
- pmxDoc = the active view
- theTheme = name of theme or table to be processed
- aCalcString = calculation string to be applied (see above)
- aField = index value of the field to be populated. Index value is between 0 and n-1, where n is the total number of fields.

RETURN: errFlag = error flag as noted below
- 0 : no error
- 1 : theme or table not found
- 2 : error in performing calculation
- 3 : no records selected
- 4 : an edit session has not been started

The given and returned variables should be declared where first called as:

```
Dim pmxDoc As IMxDocument
Dim theTheme As Variant
Dim aCalcString As String, aField As Long
Dim avCalculate As Integer
```

CALCULATE QUERY and SUMMARIZE

5.5.2 Subroutine avQuery

This subroutine enables the programmer to apply a query string to a layer (theme) or table. Sample query strings are shown below. Note how the field names are handled depending upon the type of field and the data source.

- Shapefile *String* field queries are case sensitive:
  ```
  aQueryString = """PTCODE""" + " = 'BBBB'"
  ```
- Personal Geodatabase *String* field queries are case insensitive:
  ```
  aQueryString = "PTCODE = 'bbbb'"
  ```
- Shapefile and Personal Geodatabase *Numeric* field query:
  ```
  aQueryString = "SLN >= 10"
  ```

In using this function, note the following:

1. Use avGetSelection to get the selection set representing the query result.
2. The query is applied even if the theme or layer is set to be not selectable.
3 An automation error message will be generated if the supplied query is invalid, for example (a) if the query string is built for a numeric field but the field is actually a string field, or (b) an attribute in the query does not exist in the theme or table being queried.
4. If the theme or table has a join the names of the fields in the query string must be prefixed with the name of the theme or table, for example, if layer ABCD has a join and an attribute called 123, the attribute should appear as "ABCD.123" in the query string.

The corresponding Avenue request is:

```
errFlag = aVTab.Query(aQueryString, selSet, setType)
```

avQuery

The call to this Avenue Wrap is:

```
Call avQuery(pmxDoc, theTheme, aQueryString, selSet, setType)
```

GIVEN:
- pmxDoc = the active view
- theTheme = name of theme or table to be processed
- aQueryString = query string to be applied (see above)
- selSet = theme selection set
- setType = type of selection desired
  - "NEW" : new selection set
  - "ADD" : add to current selection set
  - "AND" : select from current selection set

RETURN: nothing — Performs the query. Use the avGetSelection (see Chapter 6) to get the selection set containing the results of the query.

CALCULATE QUERY and SUMMARIZE

The given and returned variables should be declared where first called as:

```
Dim pmxDoc As IMxDocument
Dim elmntTheme As Variant, aQueryString As String
Dim selSet As ISelectionSet, setType As String
```

5.5.3 Function avSummarize

This function enables the programmer to summarize a layer (theme) or table on a specified field. In using this function, note the following:

1. If a theme is to be created and a path is specified in the input argument, aFileName, the theme will be stored in the specified path, if a table is to be created, it will be stored in the workspace of the theme or table that is being summarized, so that, in this case do not specify a full pathname and do not include an extension such as .dbf. If an extension appears in the name it will be removed with no error generated.
2. The type of summary (operation codes) to be performed on the items in the fieldList should be one of the following key words enclosed in double quotes:
   "Count" "Minimum" "Maximum" "Sum" "Average" "Variance" "StdDev" "Dissolve" (for use on the Shape field)
3. Since this routine passes NOTHING to theSumTable if an error is detected, make certain to check for this in the code that calls this function.
4. The number of items in the fieldList should be the same with that of the sumryList. If one of them is empty, so must be the other one.
5. If fieldList and sumryList are empty lists, or if they are passed in as NOTHING, the following default values will be used:
   - fieldList will contain two items each one being the value of aField.
   - The sumryList will contain two items, the first being the number of unique values within all rows of aField, and the second being the maximum unique value within all rows of aField.
6. If the theme or table to be created exists on the disk, the routine will overwrite the existing theme or table without asking or informing the user.

The corresponding Avenue request is:

```
theSumTable = aVTab.Summarize(aFileName, aType, aField,
                              fieldList, sumryList)
```

The call to this Avenue Wrap is:

avSummarize

```
Set theSumTable = avSummarize(pmxDoc, theTheme,
                              aFileName, aType, aField, fieldList,
                              sumryList)
```

CALCULATE QUERY and SUMMARIZE

GIVEN:
- pmxDoc = the active view
- theTheme = name of theme or table to be processed (see Note 1 above)
- aFileName = string name of the output table theSumTable to be created (see Notes 1 and 6 above)
- aType = type of output table, "dBase" or "Shape".
- aField = field that the theme or table is summarized on
- fieldList = additional fields to be summarized (see Note 4)
- sumryList = operation codes to be performed on the items in the fieldList (see Notes 2 and 4)

RETURN:
- theSumTable = the object summary table, whose name is that of aFileName. If an error is detected during the processing, the keyword NOTHING will be returned and a message to that effect will be displayed.

The given and returned variables should be declared where first called as:

```
Dim pmxDoc As IMxDocument, theTheme As Variant
Dim aFileName As String, aType As String, aField As String
Dim fieldList As New Collection, sumryList As New Collection
Dim theSumTable As ITable
```

Example 1 For example purposes, let us assume that:

- The table to be summarized is called "SchoolZones", and contains the data shown in Table 5-1(A),
- We wish to summarize on the field ZONE to obtain:
- (a) a count of the unique zone identification values, (b) maximum area per unique zone, and (c) minimum perimeter per unique zone.
- The summary table that is to be created is to be called "Zones" and should be of dBase format.

The call to the Avenue Wrap would be:

```
Call CreateList(fieldList)
fieldList.Add("ZONE")
fieldList.Add("ZONE")
fieldList.Add("AREA")
fieldList.Add("PERIM")
Call CreateList(sumryList)
sumryList.Add("Count")
sumryList.Add("Maximum")
sumryList.Add("Maximum")
sumryList.Add("Minimum")
```

```
Set aSTable = avSummarize(pmxDoc, "SchoolZones", _
                          "Zones", "dBase", "ZONE", _
                          fieldList, sumryList)
```

Shown in Table 5-1(B) are the results of the above summation.

The sample code on the following pages illustrates how the table could be created, records added, populated and summarized programmatically.

Example 2 Now let us consider the same table of Example 1, but with both the fieldList and sumryList arguments passed in as empty lists. The call to the Avenue Wrap would then be:

```
Call CreateList(fieldList)
Call CreateList(sumryList)
Set aSTable = avSummarize(pmxDoc, "SchoolZones", _
                          "Zones", "dBase", "ZONE", _
                          fieldList, sumryList)
```

Shown in Table 5-1(C) are the results of the above summation.

Attributes of SchoolZones

OID	ID	ZONE	AREA	PERIM
0	0	C-2	15.349	3270.72
1	0	R-4	21.537	3874.33
2	0	A-2	18.968	3635.92
3	0	C-2	14.663	1023.03
4	0	C-2	17.318	3474.18
5	0	R-4	16.259	3366.28

Record: 1 Show: All Selected Records (0 out of 6 Selected.) Options

Table 5-1(A) Sample Table to Be Summarized

Attributes of Zones

OID	ZONE	Last_ZONE	Maximum_AREA	Minimum_PERIM
0	1	A-2	18.968	3635.92
1	3	C-2	17.318	1023.03
2	2	R-4	21.537	3366.28

Record: 1 Show: All Selected Records (0 out of 3 Selected.) Options

Table 5-1(B) Sample Table Summarized as per Example 1

Attributes of Zones

OID	ZONE	Last_ZONE
0	1	A-2
1	3	C-2
2	2	R-4

Record: 1 Show: All Selected Records (0 out of 3 Selected.) Options

Table 5-1(C) Sample Table Summarized as per Example 2

CALCULATE QUERY and SUMMARIZE

```
'
'   ---
'   ---VBA code that is associated with Example 1 illustrating how to
'   ---create, add records, populate and summarize a table
'   ---
'
    Dim pMxApp As IMxApplication, pmxDoc As IMxDocument
    Dim pActiveView As IActiveView, pMap As IMap
    Dim sTblName, sTblPthName As String
    Dim iok As Integer
    Dim pTable As ITable
    Dim irec As Long
    Dim pFld1 As IFieldEdit, pFld2 As IFieldEdit, pFld3 As IFieldEdit
    Dim fldList As New Collection
    Dim theVTab As IFields
    Dim col1, col2, col3 As Long
    Dim sumTblName As String
    Dim fieldList1 As New Collection, sumryList2 As New Collection
    Dim pSTable As ITable
'
'   ---Get the active view
    Call avGetActiveDoc(pMxApp, pmxDoc, pActiveView, pMap)
'
'   ---Define the name of the table to be created
    sTblName = "SchoolZones.dbf"
'
'   ---Define the full pathname of the table
    sTblPthName = "c:\temp\" + sTblName
'
'   ---Delete the table if it exists
    If (avFileExists(sTblPthName)) Then
       iok = avFileDelete(sTblPthName)
    End If
'
'   ---Create a dBase table
    Set pTable = avVTabMakeNew(sTblPthName, "dbase")
'
'   ---Make sure the table was created
    If (Not pTable Is Nothing) Then
'
'      ---Add the table to the map, the .dbf extension will not
'      ---appear in the table of contents (TOC)
       iok = avAddDoc(pTable)
'
'      ---Add six records to the table
       irec = avAddRecord(pmxDoc, sTblName)
       irec = avAddRecord(pmxDoc, sTblName)
       irec = avAddRecord(pmxDoc, sTblName)
       irec = avAddRecord(pmxDoc, sTblName)
       irec = avAddRecord(pmxDoc, sTblName)
       irec = avAddRecord(pmxDoc, sTblName)
'
'      ---Create three fields to be added to the table
       Set pFld1 = avFieldMake("ZONE", "VCHAR", 3, 0)
       Set pFld2 = avFieldMake("AREA", "DOUBLE", 12, 4)
       Set pFld3 = avFieldMake("PERIM", "DOUBLE", 12, 4)
'
'      ---Add the fields to a list
       Call CreateList(fldList)
       fldList.Add pFld1
       fldList.Add pFld2
       fldList.Add pFld3
'
'      ---Add the field list to the table
       iok = avAddFields(pmxDoc, sTblName, fldList)
```

```
'
'     ---Get the attribute table
      Call avGetVTab(pmxDoc, sTblName, theVTab)
'
'     ---Make the table editable since when it is added to
'     ---the map, it will not be editable
      Call avSetEditable(pmxDoc, sTblName, True)
'
'     ---Store the values for all six records that were added
      col1 = theVTab.FindField("ZONE")
      col2 = theVTab.FindField("AREA")
      col3 = theVTab.FindField("PERIM")
      Call avSetValue(pmxDoc, sTblName, col1, 0, "C-2")
      Call avSetValue(pmxDoc, sTblName, col2, 0, 15.349)
      Call avSetValue(pmxDoc, sTblName, col3, 0, 3270.72)
      Call avSetValue(pmxDoc, sTblName, col3, 0, "StoreRec")
      Call avSetValue(pmxDoc, sTblName, col1, 1, "R-4")
      Call avSetValue(pmxDoc, sTblName, col2, 1, 21.537)
      Call avSetValue(pmxDoc, sTblName, col3, 1, 3874.33)
      Call avSetValue(pmxDoc, sTblName, col3, 1, "StoreRec")
      Call avSetValue(pmxDoc, sTblName, col1, 2, "A-2")
      Call avSetValue(pmxDoc, sTblName, col2, 2, 18.968)
      Call avSetValue(pmxDoc, sTblName, col3, 2, 3635.92)
      Call avSetValue(pmxDoc, sTblName, col3, 2, "StoreRec")
      Call avSetValue(pmxDoc, sTblName, col1, 3, "C-2")
      Call avSetValue(pmxDoc, sTblName, col2, 3, 14.663)
      Call avSetValue(pmxDoc, sTblName, col3, 3, 1023.03)
      Call avSetValue(pmxDoc, sTblName, col3, 3, "StoreRec")
      Call avSetValue(pmxDoc, sTblName, col1, 4, "C-2")
      Call avSetValue(pmxDoc, sTblName, col2, 4, 17.318)
      Call avSetValue(pmxDoc, sTblName, col3, 4, 3474.18)
      Call avSetValue(pmxDoc, sTblName, col3, 4, "StoreRec")
      Call avSetValue(pmxDoc, sTblName, col1, 5, "R-4")
      Call avSetValue(pmxDoc, sTblName, col2, 5, 16.259)
      Call avSetValue(pmxDoc, sTblName, col3, 5, 3366.28)
      Call avSetValue(pmxDoc, sTblName, col3, 5, "StoreRec")
'
'     ---Commit the modifications to disk
      Call avSetEditable(pmxDoc, sTblName, False)
'
'     ---Define the name of the summary table to be created
      sumTblName = "Zones"
'
'     ---Define fields and operations to be used in summarization
      Call CreateList(fieldList1)
      fieldList1.Add ("ZONE")
      fieldList1.Add ("ZONE")
      fieldList1.Add ("AREA")
      fieldList1.Add ("PERIM")
      Call CreateList(sumryList2)
      sumryList2.Add ("Count")
      sumryList2.Add ("Maximum")
      sumryList2.Add ("Maximum")
      sumryList2.Add ("Minimum")
'
'     ---Summarize all records based upon the ZONE field
      Set pSTable = avSummarize(pmxDoc, sTblName, _
                                sumTblName, "dBase", "ZONE", _
                                fieldList1, sumryList2)
'
'     ---Check if the table summarized, if so add to map
      If (Not pSTable Is Nothing) Then
         iok = avAddDoc(pSTable)
      End If
   End If
```

CALCULATE QUERY and SUMMARIZE

CALCULATE QUERY and SUMMARIZE

avTableSort

5.5.4 Subroutine avTableSort

This subroutine enables the programmer to sort an existing table, based upon a field, in an ascending or descending order. The sorting process creates a new table, and does not alter the existing table. In using this subroutine, note the following:

1. If the table contains selected records, then only the selected records will be sorted, otherwise, the entire table will be sorted.
2. A new dBase file is created containing the results of the sort, and it is added to the document. The default name of the new dBase file will be of the form tblName_sort.dbf.
3. The optional argument, aFileName, can be used to explicitly define the name of the new dBase file. If aFileName does not contain a complete pathname, the current working directory will be used. Some examples of aFileName include: c:\project\test\atable.dbf
   atable.dbf
4. If the new dBase file that is to be created exists on disk, it will be deleted prior to creating the new file. Likewise, if the new table that will be added to the document as a result of the sorting currently exists in the document, it will be deleted prior to being added back in.
5. If the argument anOrder is true, the table is sorted in an ascending order, and if it is false, the table is sorted in a descending order. Note that this is the reverse of the corresponding Avenue request.

The corresponding Avenue request is:

```
tblName.Sort(aField, anOrder)
```

The call to this Avenue Wrap is:

```
Call avTableSort(tblName, aField, anOrder, Optional aFileName)
```

GIVEN:
- tblName = name of table to be sorted.
- aField = name of field that the sort is based upon.
- anOrder = the sort order as a Boolean (see Note 5).
- aFileName = optional argument denoting the name of the new dBase file that will be created.

RETURN: nothing

The given and returned variables should be declared where first called as:

```
Dim tblName, aField As String
Dim anOrder As Boolean
Dim aFileName As String
```

5.6 Shapefile and GeoDatabase Related Avenue Wraps

SHAPEFILE and GEODATABASE

5.6.1 Function avOpenFeatClass

This function enables the programmer to open a dataset for editing purposes. A dataset may be a shapefile, raster image, tin, coverage, access database or CAD drawing. The type of object returned is of IUknown type, however, depending upon the type of dataset (opmode) to be processed, the actual type of object returned will be:

1. for shapefiles, coverages, access database featureclass and CAD drawing with a specified featureclass; IFeatureClass
2. for an access database dataset; IFeatureDataSet
3. for rasters; IRasterDataset
4. for tins; ITin
5. for CAD drawing with no specified featureclass; ICadDrawingDataset

The corresponding Avenue request is:

There is no corresponding Avenue request.

The call to this Avenue Wrap is:

avOpenFeatClass

```
Set theObject = avOpenFeatClass(opmode, sDir, sName, _
                                aFCtype)
```

GIVEN:

- opmode = type of dataset to be opened. Specify
  1 : shapefile 2 : raster 3 : tin 4 : coverage
  5 : access database feature class
  6 : access database dataset
  9: cad drawing
- sDir = directory location of the dataset
- sName = name of the dataset (do not include any filename extension in the name)
- aFCtype = feature class type (only used for coverages, access databases and CAD) if not to be used specify as NULL, for opmode = 5 this is the name of the feature class to be opened for opmode = 6 this is the name of a dataset to be opened and sName is the name of the access database, for opmode = 9 this is the name of the feature class to be opened, valid values for this mode include POINT, POLYLINE, POLYGON and ANNOTATION.

SHAPEFILE and GEODATABASE

RETURN: theObject = dataset that has been opened. If the specified dataset cannot be found, or if found and it cannot be opened, due to permission rights or other reasons, then the keyword NOTHING is returned.

The given and returned variables should be declared where first called as:

```
Dim opmode As Integer
Dim sDir As String, sName As String, aFCtype As String
Dim theObject As IUnknown
```

```
'
'   ---
'   ---Sample illustrating how to delete a dataset in a
'   ---personal geodatabase.
'   ---
'
    Dim sDir As String, dbName As String, sDSName As String
    Dim pFDataset As esriCore.IFeatureDataset
    Dim dsName As String
'
'   ---Define the directory where the geodatabase resides
    sDir = "c:\temp"
'
'   ---Define the name of the personal geodatabase
    dbName = "Profile27"
'
'   ---Define the name of the dataset to be deleted
    sDSName = "Sheet_1"
'
'   ---Check if the personal geodatabase exists
    If (avFileExists(sDir + "\"+ dbName + ".mdb")) Then
'
'      ---Try opening the dataset, if possible
       Set pFDataset = avOpenFeatClass(6, sDir, _
                                       dbName, sDSName)
'
'      ---Make sure the dataset exists
       If (Not pFDataset Is Nothing) Then
'
'         ---Combine the dataset and database names using
'         ---a single space to separate the two items, note
'         ---that the full pathname must be used to define
'         ---the database
          dsName = sDSName+" "+sDir+"\"+dbName+".mdb"
'
'         ---Delete the existing dataset
          Call avDeleteDS(dsName)
       End If
    End If
```

SHAPEFILE and GEODATABASE

5.6.2 Function avOpenWorkspace

This function enables the programmer to open a workspace for processing. A workspace may be a shapefile, raster image, tin, coverage or access database.

The corresponding Avenue request is:

There is no corresponding Avenue request.

The call to this Avenue Wrap is:

avOpenWorkspace

Set theObject = **avOpenWorkspace**(opmode, sDir, sName)

GIVEN: opmode = type of workspace to be opened. Specify
1 : shapefile 2 : raster 3 : tin
4 : coverage 5 : access database
sDir = directory location of the workspace
sName = name of the workspace (do not include any filename extension in the name)

RETURN: theObject = workspace that has been opened. If the specified workspace cannot be found, or if found and it cannot be opened, due to permission rights or other reasons, then the keyword NOTHING is returned.

The given and returned variables should be declared where first called as:

```
Dim opmode As Integer
Dim sDir As String
Dim sName As String
Dim theObject As IWorkspace
```

5.6.3 Function CreateAccessDB

This function enables the programmer to create a personal geodatabase by specifying a directory location and the name of the .mdb file to be created.

The corresponding Avenue request is:

There is no corresponding Avenue request.

The call to this Avenue Wrap is:

CreateAccessDB

Set theObject = **CreateAccessDB**(sDir, sName, bOverWrite)

SHAPEFILE and GEODATABASE

GIVEN: sDir = directory location of the workspace
sName = name of the workspace
bOverWrite = flag denoting whether the database should be overwritten if it exists (true = overwrite, false = do not overwrite)

RETURN: theObject = the workspace object representing the new personal geodatabase

The given and returned variables should be declared where first called as:

```
Dim sDir As String, sName As String, bOverWrite As Boolean
Dim theObject As IWorkspace
```

5.6.4 Function CreateAnnoClass

This function enables the programmer to create an annotation feature class within a personal geodatabase (PGD). Note that the name of the feature class to be created (sName) should not contain the dash or hyphen (-) character and that the first character in the name should be an alphacharacter, not a number. The annotation feature class is stored in a dataset within the PGD.

The corresponding Avenue request is:

There is no corresponding Avenue request.

CreateAnnoClass

The call to this Avenue Wrap is:

```
Set theObject = CreateAnnoClass(pWorkspace, sName, _
                pFields, dRefScale, dUnits)
```

GIVEN: pWorkspace = workspace of the existing geodatabase
sName = annotation feature class name if the feature class is to appear in a dataset of the same name, otherwise, the annotation feature class name and the name of the dataset are separated by at least one space
pFields = attributes associated with the feature class
dRefScale = reference scale
dUnits = units of measure setting

RETURN: theObject = object representing the new annotation feature class in the existing geodatabase

SHAPEFILE and GEODATABASE

The given and returned variables should be declared where first called as:

```
Dim pWorkspace As IWorkspace
Dim sName As String, pFields As IFields
Dim dRefScale As Double, dUnits As esriUnits
Dim theObject As IFeatureClass
```

5.6.5 Function CreateFeatClass

This function enables the programmer to create a feature class within a dataset within a geodatabase. In creating a feature class note the following:

1. The function CreateNewShapefile can be used to create the IFeatureDataset object, if appropriate.
2. If pFields contains any geometry fields they will be ignored, only valid attribute fields will be processed.
3. If pFields is not specified only the OID and SHAPE fields will be added to the featureclass.
4. The name of the feature class to be created (sName) should not contain the dash or hyphen (-) character and that the first character in the name should be an alphacharacter, not a number.

The corresponding Avenue request is:

There is no corresponding Avenue request.

CreateFeatClass

The call to this Avenue Wrap is:

```
Set theObject = CreateFeatClass(pFeatDataset, sName, _
                                geomType, pFields)
```

GIVEN:	pFeatDataset	=	dataset within geodatabase to be processed
	sName	=	name of the feature class to be created (do not include any filename extension in the name)
	geomType	=	feature class geometry type
	pFields	=	feature class attributes
RETURN:	theObject	=	object representing the new feature class in the existing geodatabase

The given and returned variables should be declared where first called as:

```
Dim pFeatDataset As IFeatureDataset, sName As String
Dim geomType As esriGeometryType, pFields As IFields
Dim theObject As IFeatureClass
```

SHAPEFILE and GEODATABASE

5.6.6 Function CreateNewGeoDB

This function enables the programmer to create a personal geodatabase (PGD) with an annotation feature class in one of two modes. The first mode employs a file dialog box, see Figure 5-1(A), where the user is able to enter the name of the personal geodatabase. The second mode creates the personal geodatabase programmatically. In using this function note the following:

Mode 1: File Dialog Box is displayed

1. A stand-alone annotation feature class within a feature dataset is created by this function, the names of the feature dataset and the annotation feature class are the same (see note 2).
2. Optionally, the user can enter up to 3 names in the Name data entry field, with each name separated from each other by at least one space (blank character). When 1 name is given see note 1. When 2 names are specified, the first name defines the name of the dataset and feature class, while the second name defines the name of the PGD to be created. When 3 names are specified, the first defines the name of the feature class, the second defines the name of the dataset and the third defines the name of the PGD to be created.
3. Use CreateNewShapefile, specifying the .mdb file name extension in the default filename, to create a geodatabase that contains a feature class for Point, Polyline and Polygon features.
4. The new annotation feature class is automatically added to the map once it has been created.

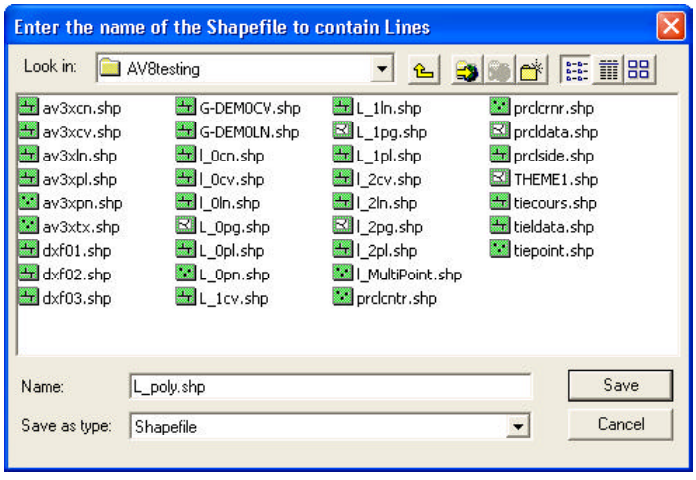

Figure 5-1(A)
File File Dialog Box
when default filename contains the .shp extension

5. If an existing .mdb file is selected, the user can either abort the command (CANCEL), add a new dataset to the .mdb file (NO) or overwrite the existing file (YES), see Figure 5-1(B).
6. When an existing .mdb file is appended the root name of the default filename is used as the name of the new annotation feature class.

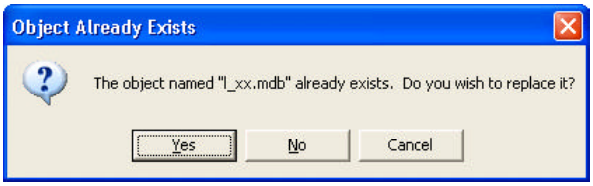

Figure 5-1(B)
Okay to Overwrite (Yes) or Append (No) an Existing Personal GeoDatabase Query

7. When an existing .mdb file is to be overwritten, if the file exists in the map the function will not delete the file but will inform the user and abort the function.
8. The Map Units for the data frame must be set to something other than Unknown Units, otherwise the MapScale property will result in an automation error message.
9. The default name of the personal geodatabase that is specified (defName) will appear in the file dialog box.

Mode 2: File Dialog Box not displayed (programmatically create the PGD)

10. When aTitle = "CREATEandLOAD" this denotes that the default filename (defName) is to be created and loaded without displaying the file dialog box. In this mode of operation, defName can contain up to three items with each item separated from each other by a space:

    >>>Single Item condition<<<

    Under this condition, the programmer specifies the name of the personal geodatabase to be created and loaded. A full pathname for the personal geodatabase must be given. If the personal geodatabase exists, it will not be deleted but rather, it will be used as is. The programmer has to make sure that the personal geodatabase does not already exist in the map, otherwise, multiple copies of the personal geodatabase will appear in the TOC because the existing personal geodatabase will be loaded into the map. An example of defName to create a geodatabase that will be named L_0.mdb and will contain a feature dataset and an annotation feature class named L_0 is:

    ```
    defName = "c:\temp\L_0.mdb"
    ```

    >>>Two Item condition<<<

    Under this condition, the programmer specifies the name of a feature

SHAPEFILE and GEODATABASE

dataset to be created and a personal geodatabase in which the feature dataset is to be stored in. The personal geodatabase can either exist or not, if it does not it will be created. If the personal geodatabase exists, the feature dataset will be added to the personal geodatabase. If the feature dataset exists in the personal geodatabase, it will not be deleted but rather, it will be used as is. The programmer has to make sure the feature dataset does not already exist in the map, otherwise, multiple copies of the feature dataset will appear in the TOC because the existing feature dataset will be loaded into the map. An example of defName to create a geodatabase that will be named L_0.mdb and will contain a feature dataset and an annotation feature class named G_Grid is:

```
defName = "G_Grid c:\temp\L_0.mdb"
```

>>>Three Item condition<<<

Similar to the two item condition, described above, with the exception that the user can control the name of the dataset that is created. An example of defName to create a geodatabase that will be named L_0.mdb and will contain a feature dataset called Profile and an annotation feature class named G_Grid is:

```
defName = "G_Grid Profile c:\temp\L_0.mdb"
```

The corresponding Avenue request is:

There is no corresponding Avenue request.

CreateNewGeoDB

The call to this Avenue Wrap is:

Set theObject = **CreateNewGeoDB**(pFieldsI, geomType, _
defName, aTitle)

GIVEN:
- pFieldsI = attributes to be stored in the new geodatabase
- geomType = feature class geometry type (as of this implementation not used, so that, the keyword NOTHING can be entered)
- defName = default filename (see notes above)
- aTitle = file dialog message box title, if aTitle is equal to CREATEandLOAD no file dialog box will be shown, the shapefile or PGD will be created without user intervention, programmatically.

RETURN:
- theRect = object representing the new annotation feature class in the existing geodatabase

The given and returned variables should be declared where first called as:

```
Dim pFieldsI As esriCore.IFields
Dim geomType As esriCore.esriGeometryType
Dim defName As String, aTitle As String
Dim theObject As esriCore.IFeatureClass
```

5.6.7 Function CreateNewShapeFile

This function enables the programmer to create (a) a shapefile or (b) a personal geodatabase (PGD) in one of two modes. The first mode employs a file dialog box, see Figure 5-1(A), where the user is able to enter the name of the shapefile or personal geodatabase. The second mode creates theshapefile or personal geodatabase programmatically. In using this function note the following:

Mode 1: File Dialog Box is displayed

1. If the pFieldsI argument is set to NOTHING, a default shape field with a default spatial reference will be assigned, and one attribute called ID will be added to the shapefile or personal geodatabase.
2. If the defName argument contains the .shp filename extension, the dataset type that will be created will be a shapefile. If the .mdb filename extension is found, the type of dataset created will be a personal geodatabase. If no filename extension is given both types will appear in the list of available types and the user can pick the desired type, see Figure 5-1(C).

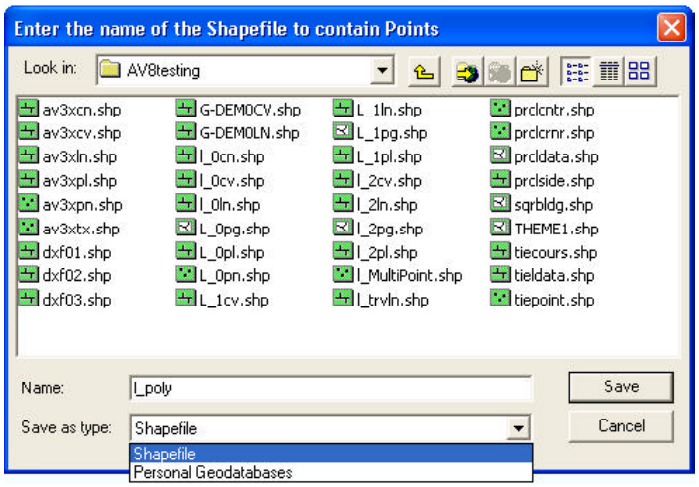

Figure 5-1(C)
CreateNewShapeFile File Dialog Box
when default filename contains no extension

SHAPEFILE and GEODATABASE

3. The new shapefile or geodatabase is automatically added to the map once it has been created.
4. When a personal geodatabase is created a feature dataset and a feature class are created using the same name, the feature class is added to the feature dataset (see note 5).
5. Optionally, the user can enter up to 3 names in the Name data entry field, with each name separated from each other by at least one space (blank character). When 1 name is given see note 4. When 2 names are specified, the first name defines the name of the dataset and feature class, while the second name defines the name of the PGD to be created. When 3 names are specified, the first defines the name of the feature class, the second defines the name of the dataset and the third defines the name of the PGD to be created.
6. If an existing .mdb file is selected, the user can either abort the command (CANCEL), add a new dataset to the .mdb file (NO) or overwrite the existing file (YES), see Figure 5-1(B).
7. When an existing .mdb file is appended the root name of the default filename is used as the name of the new feature class that is created.
8. When an existing .mdb file is to be overwritten, if the file exists in the map the function will not delete the file but will inform the user and abort the function.
9. The default name of the shapefile that is specified (defName) will appear in the file dialog box.

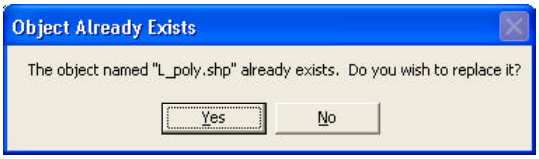

Figure 5-1(D)
Overwrite Existing Shapefile Query

10. If an existing .shp file is selected, the user can either abort the command (NO), or overwrite the existing file (YES), see Figure 5-1(D).
11. The geometry type (geomType) should be specified as:
    - esriGeometryPoint,
    - esriGeometryPolyline, or
    - esriGeometryPolygon.

    Use CreateNewGeoDB when dealing with annotation features. This function should be used when Point, Polyline or Polygon features are to be stored in the shapefile or personal geodatabase.

Mode 2: File Dialog Box not displayed (programmatically create the PGD)

12. When aTitle = "CREATEandLOAD" this denotes that the default filename (defName) is to be created and loaded without displaying the

file dialog box. In this mode of operation, defName can contain up to three items with each item separated from each other by a space:

>>>Single Item condition<<<

Under this condition, the programmer specifies the name of the shapefile or personal geodatabase to be created and loaded. A full pathname for the shapefile or personal geodatabase must be given. If the shapefile or personal geodatabase exists, it will not be deleted but rather, it will be used as is. The programmer has to make sure that the shapefile or personal geodatabase does not already exist in the map, otherwise, multiple copies of the shapefile or personal geodatabase will appear in the TOC because the existing shapefile or personal geodatabase will be loaded into the map. An example of defName to create a shapefile that will be named L_0.shp is:

```
defName = "c:\temp\L_0.shp"
```

An example of defName to create a geodatabase that will be named L_0.mdb and will contain a feature dataset and feature class both named L_0 is:

```
defName = "c:\temp\L_0.mdb"
```

>>>Two Item condition<<<

Under this condition, the programmer specifies the name of a feature dataset to be created and a personal geodatabase in which the feature dataset is to be stored in. The personal geodatabase can either exist or not, if it does not it will be created. If the personal geodatabase exists, the feature dataset will be added to the personal geodatabase. If the feature dataset exists in the personal geodatabase, it will not be deleted but rather, it will be used as is. The programmer has to make sure the feature dataset does not already exist in the map, otherwise, multiple copies of the feature dataset will appear in the TOC because the existing feature dataset will be loaded into the map. An example of defName to create a geodatabase that will be named L_0.mdb and will contain a feature dataset and feature class named G_Grid is:

```
defName = "G_Grid c:\temp\L_0.mdb"
```

>>>Three Item condition<<<

Similar to the two item condition described above with the exception that the user can control the name of the dataset that is created. An example of defName to create a geodatabase that will be named L_0.mdb and will contain a feature dataset called Profile and a feature class named G_Grid is:

```
defName = "G_Grid Profile c:\temp\L_0.mdb"
```

SHAPEFILE and GEODATABASE

CreateNewShapeFile

The corresponding Avenue request is:

There is no corresponding Avenue request.

The call to this Avenue Wrap is:

```
Set NewShapeFile = CreateNewShapeFile(pFieldsI, _
                   geomType, defName, aTitle)
```

GIVEN: pFieldsI = attributes to be stored in the new shapefile
geomType = shapefile geometry type
defName = default filename
aTitle = file dialog message box title

RETURN: NewShapeFile = feature class that is created

The given and returned variables should be declared where first called as:

```
Dim pFieldsI As esriCore.IFields
Dim geomType As esriCore.esriGeometryType
Dim defName As String
Dim aTitle As String
Dim NewShapeFile As esriCore.IFeatureClass
```

5.6.8 Function CreateShapeFile

This function enables the programmer to create a new shapefile using information explicitly defined in the calling arguments (no user interaction). In using this function, note the following:

1. The name of the shapefile to be created (strName) can or can not contain the .shp extension. If it does, it will be stripped off.
2. The geometry type (geomType) should be specified as:
   - esriGeometryPoint,
   - esriGeometryPolyline, or
   - esriGeometryPolygon.
3. The pFields argument is optional (can be omitted from the argument list). If it is not specified, a default shape field with a default spatial reference will be assigned, and one attribute called ID will be added to the shapefile.
4. The pCLSID argument is optional (can be omitted from the argument list). If it is specified, the pFields argument must also be specified.

The corresponding Avenue request is:

There is no corresponding Avenue request.

SHAPEFILE and GEODATABASE

The call to this Avenue Wrap is:

Set NewShapeFile = **CreateShapeFile**(featWorkspace, _
strName, geomType, pFields, pCLSID)

GIVEN: featWorkspace = directory location
strName = shapefile name
geomType = shapefile geometry type
pFields = shapefile attributes
pCLSID = geometry type subclass

RETURN: NewShapeFile = feature class that is created

CreateShapeFile

The given and returned variables should be declared where first called as:

Dim featWorkspace As esriCore.IFeatureWorkspace
Dim strName As String
Dim geomType As esriCore.esriGeometryType
Dim pfields As esriCore.IFields
Dim pCLSID As esriCore.UID
Dim NewShapefile As esriCore.IFeatureClass

```
'
'   ---
'   ---Sample illustrating how to create a new shapefile that
'   ---has a default spatial reference and three attributes
'   ---using a name that the user enters in a file dialog box.
'   ---The shapefile is to contain Polyline features and will
'   ---be added to the map once it has been created.
'   ---
'
    Dim pMxApp As esriCore.IMxApplication
    Dim pmxDoc As esriCore.IMxDocument
    Dim pActiveView As esriCore.IActiveView
    Dim pMap As esriCore.IMap
    Dim aDefName As String
    Dim pFieldsEdit As esriCore.IFieldsEdit
    Dim pFieldEdit As esriCore.IFieldEdit
    Dim pSR As esriCore.ISpatialReference
    Dim pGeomDef As esriCore.IGeometryDef
    Dim pGeomDefEdit As esriCore.IGeometryDefEdit
    Dim aMessage As String
    Dim pNShapeFile As esriCore.IFeatureClass
    Dim aMsg, aTitle2 As String
    Dim theTheme As Variant
    Dim theFTab As esriCore.IFields
    Dim pFeatureClass As esriCore.IFeatureClass
    Dim aLayer As esriCore.IFeatureLayer
    Dim shpFldName As String
    Dim shpType As esricore.esriGeometryType
'
```

SHAPEFILE and GEODATABASE

```
'   ---Get the active view
    Call avGetActiveDoc(pMxApp, pmxDoc, pActiveView, pMap)
'
'   ---Define the default shapefile name (since there is no
'   ---extension specified in the name, the Save as type: drop
'   ---down list will contain both Shapefile and Personal
'   ---Geodatabases)
    aDefName = "L_poly"
'
'   ---Check if the shapefile is in the map, we can not
'   ---create a shapefile if it exists in the map
    If (avFindDoc(aDefName) <> -1) Then
'       ---Remove the shapefile from the map, does not
'       ---delete it from the hard drive (disk)
       Call avRemoveDoc(aDefName)
    End If
'
'   ---Create the required shapefile attributes
'
'   ---Define the object ID field
    Set pFieldsEdit = New esriCore.Fields
    Set pFieldEdit = New esriCore.Field
    With pFieldEdit
        .name = "OID"
        .Type = esriCore.esriFieldTypeOID
        .aliasName = "Object ID"
       .IsNullable = False
    End With
    pFieldsEdit.AddField pFieldEdit
'
'   ---Assign the default spatial reference
    Set pSR = New esriCore.UnknownCoordinateSystem
    pSR.SetDomain -9999999999#, 9999999999#, _
                  -9999999999#, 9999999999#
    pSR.SetFalseOriginAndUnits 0, 0, 100000#
'
'   ---Define geometry type for shape field to be Polyline
    Set pGeomDef = New esriCore.GeometryDef
    Set pGeomDefEdit = pGeomDef
    With pGeomDefEdit
        .GeometryType = esriCore.esriGeometryPolyline
        .GridCount = 1
        .GridSize(0) = 10
        .AvgNumPoints = 2
        .HasM = False
        .HasZ = False
        Set .SpatialReference = pSR
    End With
'
'   ---Define the Shape Field
    Set pFieldEdit = New esriCore.Field
    With pFieldEdit
        .name = "Shape"
```

SHAPEFILE and GEODATABASE

```
        .Type = esriCore.esriFieldTypeGeometry
        .IsNullable = True
        .Editable = True
        .aliasName = "Shape"
        Set .GeometryDef = pGeomDef
    End With
    pFieldsEdit.AddField pFieldEdit
'
'   ---Add the desired attributes into the attribute list
'   ---In this example we will add an integer attribute, a
'   ---double attribute and a string attribute using arbitrary
'   ---field names and sizes
'
'   ---Map Number
    Set pFieldEdit = New esriCore.Field
    pFieldEdit.name = "MAP"
    pFieldEdit.Type = esriCore.esriFieldTypeInteger
    pFieldEdit.DomainFixed = False
    pFieldEdit.Editable = True
    pFieldEdit.IsNullable = False
    pFieldEdit.Precision = 8
    pFieldsEdit.AddField pFieldEdit
'
'   ---Line Length
    Set pFieldEdit = New esriCore.Field
    With pFieldEdit
        .name = "LEN"
        .Editable = True
        .IsNullable = False
        .Precision = 14
        .Scale = 4
        .Type = esriCore.esriFieldTypeDouble
    End With
    pFieldsEdit.AddField pFieldEdit
'
'   ---Description associated with the polyline
    Set pFieldEdit = New esriCore.Field
    pFieldEdit.name = "LINE_DESC"
    pFieldEdit.Type = esriCore.esriFieldTypeString
    pFieldEdit.Editable = True
    pFieldEdit.IsNullable = False
    pFieldEdit.Precision = 40
    pFieldsEdit.AddField pFieldEdit
'
'   ---Define the file dialog message box title
    aMessage = "Enter the name of the Shapefile " + _
               "to contain Lines"
'
'   ---Prompt the user to specify the shapefile name
    Set pNShapeFile = CreateNewShapefile(pFieldsEdit, _
                              esriCore.esriGeometryPolyline, _
                              aDefName, aMessage)
'
```

SHAPEFILE and GEODATABASE

```
'   ---Check if the command has been canceled (aborted)
    If (ugerror = 1) Then
       Exit Sub
    End If
'
'   ---Check if any problems were detected
    If pNShapeFile Is Nothing Then
'
'      ---Inform user of the problem
       aMsg = "Error creating Shapefile, check permissions."
       aTitle2 = "Create Shapefile"
       Call avMsgBoxWarning(aMsg, aTitle2)
       Exit Sub
'
'   ---Shapefile created properly
    Else
'
'      ---Get the name of the shapefile
       theTheme = pNShapeFile.aliasName
    End If
'
'   ---Get the attribute table for the theme
    Call avGetFTab(pmxDoc, theTheme, _
                   theFTab, pFeatureClass, aLayer)
'
'   ---Determine the name of the shape field for the theme
    shpFldName = pFeatureClass.ShapeFieldName
'
'   ---Determine the type of features stored in the theme
    shpType = pFeatureClass.ShapeType
```

5.7 Linking and Joining Tables

5.7.1 Function avIsJoined

This function enables the programmer to determine whether a field has been added to a VTab as a result of a Join.

The corresponding Avenue request is:

theAnsw = aVTab.IsJoinedField(aField)

The call to this Avenue Wrap is:

theAnsw = **avIsJoined**(aVTab)

GIVEN: aVTab = name of VTab to be processed.

RETURN: theAnsw = flag denoting whether the input object has links or not.
true = has links, false = not linked

The given and returned variables should be declared where first called as:

```
Dim aVTab As String
Dim theAnsw As Boolean
```

5.7.2 Function avIsLinked

This function enables the programmer to determine whether a VTab has links (relates to other tables) or not.

The corresponding Avenue request is:

theAnsw = aVTab.IsLinked

The call to this Avenue Wrap is:

theAnsw = **avIsLinked**(aVTab)

GIVEN: aVTab = name of VTab to be processed.

RETURN: theAnsw = flag denoting whether the input object has links or not.
true = has links, false = not linked

The given and returned variables should be declared where first called as:

```
Dim aVTab As String
Dim theAnsw As Boolean
```

LINKING and COMBINING TABLES

avJoin

5.7.3 Function avJoin

This function enables the programmer to join aVTab2 to aVTab1 using user specified field names. In using this function, note that whereas the Avenue request returns a boolean (theAnsw), the Avenue Wrap returns an integer. See below the associated values under the returned argument theAnsw.

The corresponding Avenue request is:

theAnsw = aVTab1.Join(aField1, aVTab2, aField2)

The call to this Avenue Wrap is:

theAnsw = **avJoin**(aVTab1, aField1, aVTab2, aField2)

GIVEN:
- aVTab1 = the name of the VTab to which aVTab2 is to be joined.
- aField1 = the field in aVTab1 upon which the join is based.
- aVTab2 = the name of VTab to be joined to aVTab1.
- aField2 = the field in aVTab2 upon which the join is based.

RETURN:
- theAnsw = error flag, where the values below denote the indicated results of the function.
  - 0 : no error
  - 1 : error detected
  - 2 : aVTab1 does not exist
  - 3 : aVTab2 does not exist

The given and returned variables should be declared where first called as:

```
Dim aVTab1 As String, aField1 As String
Dim aVTab2 As String, aField2 As String
Dim avJoin As Integer
```

5.7.4 Function avLink

This function enables the programmer to link (relate)aVTab2 to aVTab1 using user specified field names. In using this function, note that whereas the Avenue request returns a boolean (theAnsw), the Avenue Wrap returns an integer. See below the associated values under the returned argument theAnsw.

The corresponding Avenue request is:

theAnsw = aVTab1.Link(aField1, aVTab2, aField2)

The call to this Avenue Wrap is:

theAnsw = **avLink**(aVTab1, aField1, aVTab2, aField2)

GIVEN: aVTab1 = the name of the VTab to which aVTab2 is to be linked.
aField1 = the field in aVTab1 upon which the join is based.
aVTab2 = the name of VTab to be linked to aVTab1.
aField2 = the field in aVTab2 upon which the link is based.

RETURN: theAnsw = error flag, where the values below denote the indicated results of the function.
- 0 : no error
- 1 : error detected
- 2 : aVTab1 does not exist
- 3 : aVTab2 does not exist

The given and returned variables should be declared where first called as:

```
Dim aVTab1 As String, aField1 As String
Dim aVTab2 As String, aField2 As String
Dim theAnsw As Integer
```

5.7.5 Function avUnJoinAll

This function enables the programmer to remove all joins from a VTab. In using this function, note that the Avenue Wrap returns an integer. See below the associated values under the returned argument theAnsw.

The corresponding Avenue request is:

aVTab.UnjoinAll

The call to this Avenue Wrap is:

theAnsw = **avUnJoinAll**(aVTab)

GIVEN: aVTab = the name of the VTab from which all joins are to be removed.

LINKING and COMBINING TABLES

RETURN: theAnsw = error flag, where the values below denote the indicated results of the function.
- 0 : no error
- 1 : error detected
- 2 : aVTab does not exist

The given and returned variables should be declared where first called as:
Dim aVTab As String
Dim theAnsw As Integer

5.7.6 Function avUnLinkAll

This function enables the programmer to remove all links (relates) from a VTab. In using this function, note that the Avenue Wrap returns an integer. See below the associated values under the returned argument theAnsw.

The corresponding Avenue request is:
aVTab.UnlinkAll

avUnLinkAll

The call to this Avenue Wrap is:
theAnsw = **avUnLinkAll**(aVTab)

GIVEN: aVTab = the name of the VTab from which all links are to be removed.

RETURN: theAnsw = error flag, where the values below denote the indicated results of the function.
- 0 : no error
- 1 : error detected
- 2 : aVTab does not exist

The given and returned variables should be declared where first called as:
Dim aVTab As String
Dim theAnsw As Integer

5.7.7 Function avUpdateJoin

This function enables the programmer to update the selection set of aVTab2 to reflect the selection set of aVTab1 based upon a join (relate). Furthermore, this procedure will refresh the selection set for the VTab being processed, aVTab1 in addition to the selection set of aVTab2.

The corresponding Avenue request is:

There is no corresponding Avenue request.

The call to this Avenue Wrap is:

avUpdateJoin

theAnsw = **avUpdateJoin**(aVTab1, aVTab2)

GIVEN: aVTab1 = the name of the VTab to which aVTab2 is joined.

VTab2 = the name of the VTab joined to aVTab1

RETURN: theAnsw = error flag, where the values below denote the indicated results of the function.

- 0 : no error
- 1 : error detected
- 2 : aVTab1 does not exist
- 2 : aVTab2 does not exist
- 4 : join was not found

The given and returned variables should be declared where first called as:

```
Dim aVTab1, aVTab2 As String
Dim aLink As Long
Dim theAnsw As Integer
```

5.7.8 Function avUpdateLink

This function enables the programmer to update the selection set in aVTab2 to reflect the selection set of aVTab1 based upon a specified link (relate). If aVTab2 is a layer and if aLinkI is negative the selection set of aVTab2 is updated but the display of the selected features is not. This is useful when performing loops where it is not necessary to have the screen redrawn after each iteration within the loop. Since refreshing the screen is slow the use of this function with a negative link ID value can be very useful. In using this function, note that the Avenue Wrap returns an integer. See below the associated values under the returned argument theAnsw.

LINKING and COMBINING TABLES

avUpdateLink

The corresponding Avenue request is:

There is no corresponding Avenue request.

The call to this Avenue Wrap is:

theAnsw = **avUpdateLink**(aVTab1, aVTab2, aLink)

GIVEN: aVTab1 = the name of the VTab to which aVTab2 is linked.

VTab2 = the name of the VTab linked to aVTab1

aLinkI = link number in aVTab1 to be updated (if aVTab2 is a layer and if aLinkI is negative the selection set of aVTab2 is updated but the display of the selected features is not)

RETURN: theAnsw = error flag, where the values below denote the indicated results of the function.

- 0 : no error
- 1 : error detected
- 2 : aVTab1 does not exist
- 2 : aVTab2 does not exist
- 4 : link number was not found

The given and returned variables should be declared where first called as:

```
Dim aVTab1, aVTab2 As String
Dim aLinkI As Long
Dim theAnsw As Integer
```

5.7.9 Function avUpdateLinks

This function enables the programmer to update the selection sets in all VTabs that are linked (related) to aVTab1. Furthermore, this function will refresh the selection set for the VTab being processed, aVTab1 in addition to all of the selection sets that aVTab1 has links (relates) with.

The corresponding Avenue request is:

There is no corresponding Avenue request.

avUpdateLinks

The call to this Avenue Wrap is:

theAnsw = **avUpdateLinks**(aVTab1)

GIVEN: aVTab1 = the name of the VTab to be processed

RETURN: theAnsw = error flag, where the values below denote the indicated results of the function.

- 0 : no error
- 1 : error detected
- 2 : aVTab1 does not exist
- 3 : no links were found

The given and returned variables should be declared where first called as:

Dim aVTab1 As String

Dim theAnsw As Integer

```
'
'   ---
'   ---Sample illustrating how to join a table to a layer and
'   ---transfer a value from the table, as a result of the
'   ---join, to a specific feature in the layer
'   ---
'
    Dim pMxApp As esriCore.IMxApplication
    Dim pmxdoc As esriCore.IMxDocument
    Dim pActiveView As esriCore.IActiveView
    Dim pMap As esriCore.IMap
    Dim aVTab1 As String, aField1 As String
    Dim aVTab2 As String, aField2 As String
    Dim iok As Integer
    Dim theFTab As esriCore.IFields
    Dim pFCls As esriCore.IFeatureClass
    Dim pFLyr As esriCore.IFeatureLayer
    Dim pTable As esriCore.iTable
    Dim colL As Long, colT As Long
    Dim oidList As New Collection
    Dim iRec As Long
    Dim pFeat As esricore.iFeature
    Dim pFeatRow As esricore.iRow
    Dim aVal As Variant
'
'   ---Get the active view
    Call avGetActiveDoc(pMxApp, pmxdoc, pActiveView, pMap)
'
'   ---Define the layer that will have a join and the field
'   ---that the join will be based upon
    aVTab1 = "sewnodes"
    aField1 = "NODID"
'
'   ---Define the table to be joined to the layer and the field
'   ---that the join will be based upon
    aVTab2 = "sewhydro"
    aField2 = "NODID"
```

LINKING and COMBINING TABLES

```
'
'  ---Join the table to the layer
   iok = avJoin(aVTab1, aField1, aVTab2, aField2)
'
'  ---Get the attribute table for the layer, note that the
'  ---attributes in theFTab contain only the attributes in
'  ---the layer not the layer and the table
   Call avGetFTab(pmxdoc, aVTab1, theFTab, pFCls, pFLyr)
'
'  ---In order to access the fields in the table which were
'  ---joined to the layer the ITable interface must be
'  ---used, otherwise only the attributes in the layer will
'  ---be found (theFTab now contains both sets of fields)
   Set pTable = pFLyr
   Set theFTab = pTable.Fields
'
'  ---Define a field in the layer attribute table after the
'  ---join was applied (note that the name of the layer must
'  ---precede the name of the field)
   colL = theFTab.FindField("sewnodes.GRELVZ")
'
'  ---Define the field in the table which should now appear
'  ---in the attribute table as a result of the join
   colT = theFTab.FindField("sewhydro.ELEV")
'
'  ---Make the theme editable
   Call avSetEditable(pmxdoc, aVTab1, True)
'
'  ---Start an operation
   Call avStartOperation
'
'  ---Get a list of the OIDs in the layer
   Call avGetFTabIDs(pmxdoc,aVTab1, oidList)
'
'  ---Define the record to be processed
   iRec = oidList.Item(1)
'
'  ---Get the feature in the layer to be modified
   Set pFeat = pFCls.GetFeature(iRec)
'
'  ---Get the IRow for the feature (record) since it contains the
'  ---results of the join
   Set pFeatRow = pTable.GetRow(iRec)
'
'  ---Get the value from the table that has been joined to the layer
   aVal = pFeatRow.Value(colT)
'
'  ---Transfer the table value to the feature (note that the pFeat
'  ---object, not the pFeatRow object, is used)
   pFeat.Value(colL) = aVal
'
'  ---Store the feature
   pFeat.Store
```

```
'
'  ---Stop the operation
   Call avStopOperation("Modify Feature")
'
'  ---Remove the join from the layer
   iok = avUnJoinAll(aVTab1)
'
'   ---
'   ---Sample illustrating how to link a table to a layer.
'   ---
'
   Dim pMxApp As esriCore.IMxApplication
   Dim pmxdoc As esriCore.IMxDocument
   Dim pActiveView As esriCore.IActiveView
   Dim pMap As esriCore.IMap
   Dim aVTab1 As String, aField1 As String
   Dim aVTab2 As String, aField2 As String
   Dim iok As Integer
   Dim sel As esriCore.ISelectionSet
   Dim aQuery As String
   Dim selTable As esriCore.ISelectionSet
'
'  ---Get the active view
   Call avGetActiveDoc(pMxApp, pmxdoc, pActiveView, pMap)
'
'  ---Define the layer that will have a link assigned to it and
'  ---the field that the link will be based upon
   aVTab1 = "sewnodes"
   aField1 = "NODID"
'
'  ---Define the table to be linked to the layer and the field
'  ---that the link will be based upon
   aVTab2 = "sewhydro"
   aField2 = "NODID"
'
'  ---Link the table to the layer
   iok = avLink(aVTab1, aField1, aVTab2, aField2)
'
'  ---Check if the link has been applied to the layer
   If (avIsLinked(aVTab1)) Then
      MsgBox "Link has been applied to: " + aVTab1
   End If
'
'  ---Get the current selection set for the layer
   Call avGetSelection(pmxdoc, aVTab1, sel)
'
'  ---Appy a query to the layer
   aQuery = "NODID = 82309"
   Call avQuery(pmxdoc, aVTab1, aQuery, sel, "NEW")
```

LINKING and COMBINING TABLES

```
'
'  ---Get the selection set for the layer which contains
'  ---the results of the query
   Call avGetSelection(pmxdoc, aVTab1, sel)
'
'  ---Update the selection set for the layer
   Call avUpdateSelection(pmxdoc, aVTab1)
'
'  ---Make sure the display is current
   pActiveView.Refresh
'
'  ---In order to have the linked table reflect the selection
'  ---in the layer we must update the link, if this is not
'  ---done the table selection will not reflect the link. Since
'  ---the layer has only one link assigned to it the link
'  ---number is one (1)
   Call avUpdateLink(aVTab1, aVTab2, 1)
'
'  ---Get the selection set for the table
   Call avGetSelection(pmxdoc, aVTab2, selTable)
'
'  ---Display the number of selected features in the layer
'  ---and the table
   MsgBox "Selected features = " + CStr(sel.Count) + Chr(13) + _
          "Selected records  = " + CStr(selTable.Count)
'
'  ---Remove the link from the layer
   iok = avUnLinkAll(aVTab1)
```

5.8 Sample Code

SAMPLE CODE

The sample code below contains two examples, (a) one that illustrates how to create a shapefile, in this example a polyline, and add a feature to it, and (b) another that illustrates how to create a table and perform various editing operations. In the course of these samples, certain other operations are demonstrated, some of which are used strictly for illustration purposes. Note that the various Avenue Wraps that are called below have been highlighted in bold font. Some of these Avenue Wraps are discussed in detail in other chapters.

```
'
'   ---
'   ---Example #1
'   ---Sample code illustrating how to create a Shapefile, and
'   ---add a feature to it.
'   ---
'
    Dim pMxApp As IMxApplication
    Dim pmxDoc As IMxDocument
    Dim pActiveView As IActiveView
    Dim pMap As IMap
    Dim sThmName, sPthName As String
    Dim PTheme As IFeatureLayer
    Dim aIndex As Long
    Dim iok As Integer
    Dim iRec As Long
    Dim theFTab As IFields
    Dim pFeatCls As IFeatureClass
    Dim pLayer As IFeatureLayer
    Dim pLineX As IPolyline
    Dim aField As Long
    Dim pFeature As esriCore.IFeature
    Dim shapeList As New Collection
    Dim nParts As Long
    Dim partList As New Collection
    Dim nPts As Long
    Dim pt1 As esriCore.IPoint
    Dim pt2 As esriCore.IPoint
    Dim X1 As Double, Y1 As Double
    Dim X2 As Double, Y2 As Double
    Dim aMsg As Variant
    Dim sTblName, sTblPthName As String
    Dim pTable As ITable
    Dim pFld1 As IFieldEdit
    Dim pFld2 As IFieldEdit
    Dim pFld3 As IFieldEdit
    Dim fldList As New Collection
    Dim theVTab As IFields
    Dim col As Long
    Dim nDigits As Long
```

SAMPLE CODE

```
    Dim pField As IField
    Dim aType As esriFieldType
    Dim idList As New Collection
    Dim aTotal As Double
    Dim jRec As Long
    Dim rec As Long
    Dim pRow As IRow
    Dim aVal As Double
    Dim nRec As Long
    Dim sel As ISelectionSet
    Dim aCalcString, aQueryString As String
    Dim sumTblName As String
    Dim fieldList1 As New Collection
    Dim sumryList2 As New Collection
    Dim pSTable As ITable
'
'   ---Get the active view                                  <<<------
    Call avGetActiveDoc(pMxApp, pmxDoc, pActiveView, pMap)
'
'   ---Define the name of the shapefile to be created
    sThmName = "L_poly.shp"
'
'   ---Define the full pathname of the shapefile
    sPthName = "c:\temp\" + sThmName
'
'   ---Create a polyline shapefile
    Set PTheme = avFTabMakeNew(sPthName, "POLYLINE")
'
'   ---Make sure the shapefile was actually created.
'   ---It is possible that, due to certain restrictions that may have
'   ---been imposed on the operating system by its administrator,
'   ---the shapefile may not have been created. In addition, if
'   ---the shapefile exists, it will not be created.
    If (PTheme Is Nothing) Then
       MsgBox "Error in creating shapefile: " + sThmName
'
'      ---Check if the shapefile exists
       If (avFileExists(sPthName)) Then
          MsgBox "Shapefile: " + sPthName + " exists"
'         ---Check if the shapefile exists in the map
          aIndex = avFindDoc(sThmName)
          If (aIndex <> -1) Then
'            ---Remove the shapefile from the map
             Call avRemoveDoc(sThmName)
             MsgBox "Shapefile: " + sThmName + " removed from TOC"
          End If
'         ---Delete the shapefile from disk
          iok = avDeleteDS(sPthName)
          If (iok = 0) Then
             MsgBox "Shapefile: " + sPthName + " deleted " + CStr(iok)
          Else
             MsgBox "Error deleting shapefile"
          End If
```

SAMPLE CODE

```
      Else
         MsgBox "Shapefile: " + sThmName + " does not exist" + _
                Chr(13) + "and could not create the shapefile"
      End If
'
'  ---Handle the case when the shapefile was created
   Else
      MsgBox "Shapefile: " + sThmName + " created"
'
'     ---Add the shapefile to the map
      iok = avAddDoc(PTheme)
      MsgBox "Shapefile: " + sThmName + " added to TOC"
'
'     ---Make the shapefile editable
      Call avSetEditable(pmxDoc, sThmName, True)
'
'     ---Start an operation that will be added to the Undo list
      Call avStartOperation
'
'     ---Add a record to the shapefile, this is a new feature that
'     ---has been added
      iRec = avAddRecord(pmxDoc, sThmName)
'
'     ---Get the attribute table
      Call avGetFTab(pmxDoc, sThmName, theFTab, pFeatCls, pLayer)
'
'     ---Create a line that will represent the geometry of a new
'     ---feature in the shapefile
      Set pLineX = avPolyline2Pt(20000#, 20000#, 30000#, 25000#)
'
'     ---Store the geometry for the new feature in the shape field
'     ---of the layer
      aField = theFTab.FindField("SHAPE")
      Call avSetValueG(pmxDoc, sThmName, aField, iRec, pLineX)
'
'     ---Redraw the theme to refresh the display
      Call avThemeInvalidate(pmxDoc, sThmName, True)
'
'     ---Stop the editing operation so that the operation consists
'     ---only of adding a single feature.
'     ---Note that the editor will remain in an edit state so that
'     ---the Undo capabilities can be utilized, if so desired
      Call avStopOperation("Add Feature")
      MsgBox "Feature added to map"
'
'     ---Display the coordinates of the endpoints of the line
'     ---First get the feature, since there is only one feature in
'     ---the shapefile we know it is at record zero
      Call avGetFeature(pmxDoc, sThmName, 0, pFeature)
```

SAMPLE CODE

```
'
'     ---Get a list of list of points which comprise the feature
      Call avAsList(pFeature, shapeList)
'
'     ---Determine the number of parts comprising the feature
      nParts = shapeList.Count
'
'     ---Get the first part comprising the feature
      Set partList = shapeList.Item(1)
'
'     ---Determine the number of points in the part
      nPts = partList.Count
'
'     ---Get the first and last points in the part
      Set pt1 = partList.Item(1)
      Set pt2 = partList.Item(2)
'
'     ---Get the X and Y coordinates for each point
      X1 = pt1.x
      Y1 = pt1.y
      X2 = pt2.x
      Y2 = pt2.y
'
'     ---Display the coordinates to three digits to the right of
'     ---the decimal point
      aMsg = "X1 = " + Dformat(X1, 1, 3) + "  " + _
             "Y1 = " + Dformat(Y1, 1, 3) + Chr(13) + _
             "X2 = " + Dformat(X2, 1, 3) + "  " + _
             "Y2 = " + Dformat(Y2, 1, 3)
      Call avMsgBoxInfo(aMsg, "Sample Exercise")
   End If
'
'  ---
'  ---Example #2
'  ---Sample illustrating how to create a dBase Table, and perform
'  ---various table editing operations.
'  ---
'
'  ---Define the name of the table to be created
   sTblName = "table1.dbf"
'
'  ---Define the full pathname of the table
   sTblPthName = "c:\temp\" + sTblName
'
'  ---Create a dBase table
   Set pTable = avVTabMakeNew(sTblPthName, "dbase")
'
```

SAMPLE CODE

```
'  ---Make sure the table was actually created.  It is possible that
'  ---the table was not created:
'  ---(a) due to certain restrictions that may have been imposed on
'  ---    the operating system by its administrator, or
'  ---(b) because the table may exist on the disk.
   If (pTable Is Nothing) Then
'     ---The table was not created.  Display an error message
      MsgBox "Error in creating table: " + sTblName
'
'     ---Check if the table exists in the disk.  If so remove it.
      If (avFileExists(sTblPthName)) Then
         MsgBox "Table: " + sTblPthName + " exists"
'        ---Check if the table exists in the map.  If so, remove it.
         aIndex = avFindDoc(sTblName)
         If (aIndex <> -1) Then
'           ---Remove the table from the map
            Call avRemoveDoc(sTblName)
            MsgBox "Table: " + sTblPthName + " removed from TOC"
         End If
'        ---Delete the table from disk
         iok = avDeleteDS(sTblPthName)
         If (iok = 0) Then
            MsgBox "Table: " + sTblPthName + " deleted " + CStr(iok)
         Else
            MsgBox "Error deleting table"
         End If
      Else
         MsgBox "Table: " + sTblName + " does not exist" + _
                Chr(13) + "and could not create the table"
      End If
'
'  ---Handle the case when the table is created
   Else
      MsgBox "Table: " + sTblName + " created"
'
'     ---Add the table to the map
      iok = avAddDoc(pTable)
      MsgBox "Table: " + sTblName + " added to TOC"
'
'     ---Add three records to the table
      iRec = avAddRecord(pmxDoc, sTblName)
      iRec = avAddRecord(pmxDoc, sTblName)
      iRec = avAddRecord(pmxDoc, sTblName)
      MsgBox "3 records added to " + sTblName
'
'     ---Create three fields that will be added to the table
      Set pFld1 = avFieldMake("StringF", "vchar", 20, 0)
      Set pFld2 = avFieldMake("DoubleF", "double", 12, 4)
      Set pFld3 = avFieldMake("LongF", "long", 10, 0)
```

SAMPLE CODE

```
'
'     ---Add the fields to a collection
      Call createList(fldList)
      fldList.Add pFld1
      fldList.Add pFld2
      fldList.Add pFld3
'
'     ---Add the fields collection to the table
      iok = avAddFields(pmxDoc, sTblName, fldList)
'
'     ---Get the attribute table for the VTab
      Call avGetVTab(pmxDoc, sTblName, theVTab)
'
'     ---Check to see whether the table is editable or not.
'     ---If not, make it so.
      If (Not avIsEditable(sTblName)) Then
         MsgBox "Table: " + sTblName + " is not editable"
'
'        ---Make the table editable
         Call avSetEditable(pmxDoc, sTblName, True)
         If (avIsEditable(sTblName)) Then
            MsgBox "Table: " + sTblName + " is now editable"
         End If
'
'        ---Store a value in the table, under a specific field,
'        ---for all three records that were added
         col = theVTab.FindField("StringF")
'
'        ---Make sure the field was found (it exists), a value of -1
'        ---for a field index denotes the field does not exist
         if (col <> -1)then
'           ---Store a value in the table for all three records
'           ---that were added. The "StoreRec" argument in the call
'           ---to avSetValue indicates that the record is to be
'           ---written to disk, if this call is not made the user
'           ---will not see the "test string" or any of the other
'           ---values in thedatabase (Note, the call to avSetValue
'           ---with the "StoreRec"argument should be made once,
'           ---after all other avSetValue calls have been made for
'           ---a record)
            Call avSetValue(pmxDoc, sTblName, col, 0, "test string")
            Call avSetValue(pmxDoc, sTblName, col, 0, "StoreRec")
            Call avSetValue(pmxDoc, sTblName, col, 1, "string 2")
            Call avSetValue(pmxDoc, sTblName, col, 1, "StoreRec")
            Call avSetValue(pmxDoc, sTblName, col, 2, "third string")
            Call avSetValue(pmxDoc, sTblName, col, 2, "StoreRec")
         End If
'
```

SAMPLE CODE

```
'       ---Get the field index value for the DoubleF field
        col = theVTab.FindField("DoubleF")
'
'       ---Store values for specific records
        Call avSetValue(pmxDoc, sTblName, col, 0, 14.3456)
        Call avSetValue(pmxDoc, sTblName, col, 0, "StoreRec")
        Call avSetValue(pmxDoc, sTblName, col, 1, 24.3456)
        Call avSetValue(pmxDoc, sTblName, col, 1, "StoreRec")
        Call avSetValue(pmxDoc, sTblName, col, 2, 34.3456)
        Call avSetValue(pmxDoc, sTblName, col, 2, "StoreRec")
'
'       ---Display the precision of the field just populated
        nDigits = avGetPrecision(theVTab, col)
        MsgBox "Digits right of Decimal for DoubleF = " + _
                                                    CStr(nDigits)
'
'       ---Display the field type of the field just populated
        Set pField = theVTab.Field(col)
        aType = avFieldGetType(pField)
        MsgBox "DoubleF field type = " + CStr(aType)
'
'       ---Get a list of the record IDs for the VTab
        Call avGetVTabIDs(pmxDoc, sTblName, idList)
'
'       ---Sum the values in the DoubleF field for all records
        aTotal = 0#
        For jRec = 1 To idList.Count
'           ---Extract a record ID from the list
            rec = idList.Item(jRec)
'           ---Get the IRow interface for the record
            Set pRow = pTable.GetRow(rec)
'           ---Extract the value for the DoubleF field
            aVal = pRow.Value(col)
'           ---Add the value to the total
            aTotal = aTotal + aVal
        Next
        MsgBox "Total for DoubleF = " + Dformat(aTotal, 1, 4)
'
'       ---Commit the modifications to the disk
        Call avSetEditable(pmxDoc, sTblName, False)
'
'       ---Determine the number of records in the table
        nRec = avGetNumRecords(pmxDoc, sTblName)
        MsgBox "Number of records in " + sTblName + " = " + _
                                                    CStr(nRec)
'
'       ---Select all of the records in the table
        Call avSetAll(pmxDoc, sTblName, sel)
        MsgBox CStr(sel.Count) + " records selected (all)"
```

SAMPLE CODE

```
'
'       ---Clear the selection
        Call avClearSelection(pmxDoc, sTblName)
        Call avGetSelection(pmxDoc, sTblName, sel)
        MsgBox CStr(sel.Count) + " records selected (none)"
'
'       ---Select the second and third records in the table
        Call avBitmapSet(pmxDoc, sTblName, 1)
        Call avBitmapSet(pmxDoc, sTblName, 2)
        Call avGetSelection(pmxDoc, sTblName, sel)
        MsgBox CStr(sel.Count) + " records selected "
'
'       ---Clear the second record from the selection
        Call avGetSelectionClear(pmxDoc, sTblName, 1)
        MsgBox "1 selected record deselected"
'
'       ---Start editing on the table
        Call avSetEditable(pmxDoc, sTblName, True)
'
'       ---Add 16 records to the table
        iRec = avAddRecord(pmxDoc, sTblName)
        iRec = avAddRecord(pmxDoc, sTblName)
        iRec = avAddRecord(pmxDoc, sTblName)
        iRec = avAddRecord(pmxDoc, sTblName)
        iRec = avAddRecord(pmxDoc, sTblName)
        iRec = avAddRecord(pmxDoc, sTblName)
        iRec = avAddRecord(pmxDoc, sTblName)
        iRec = avAddRecord(pmxDoc, sTblName)
        iRec = avAddRecord(pmxDoc, sTblName)
        iRec = avAddRecord(pmxDoc, sTblName)
        iRec = avAddRecord(pmxDoc, sTblName)
        iRec = avAddRecord(pmxDoc, sTblName)
        iRec = avAddRecord(pmxDoc, sTblName)
        iRec = avAddRecord(pmxDoc, sTblName)
        iRec = avAddRecord(pmxDoc, sTblName)
        iRec = avAddRecord(pmxDoc, sTblName)
        MsgBox "multiple records added"
'
'       ---Clear the selection set for the table
        Call avClearSelection(pmxDoc, sTblName)
'
'       ---Select two records
        Call avBitmapSet(pmxDoc, sTblName, 0)
        Call avBitmapSet(pmxDoc, sTblName, 1)
        MsgBox "two records selected"
'
'       ---Delete the selected records in the table
        Call avRemoveRecord(pmxDoc, sTblName, -1)
        MsgBox "two records deleted"
```

SAMPLE CODE

```
'
'        ---Stop editing on the table
         Call avSetEditable(pmxDoc, sTblName, False)
      End If
   End If
'
'  ---Get the attribute table
   Call avGetVTab(pmxDoc, sTblName, theVTab)
'
'  ---Make sure the table exists
   If (Not theVTab Is Nothing) Then
'
'     ---Make the table editable
      Call avSetEditable(pmxDoc, sTblName, True)
'
'     ---Build an arbitrary calculation string
      col = theVTab.FindField("LongF")
      nRec = avGetNumRecords(pmxDoc, sTblName)
      aCalcString = "([DoubleF] - " + CStr(nRec) + ")"
'
'     ---Apply a Calculation to two selected records
      Call avClearSelection(pmxDoc, sTblName)
      Call avBitmapSet(pmxDoc, sTblName, 0)
      Call avBitmapSet(pmxDoc, sTblName, 1)
      iok = avCalculate(pmxDoc, sTblName, aCalcString, col)
      MsgBox "2 records applied a calculation"
'
'     ---Apply a new Calculation to one selected record
      Call avClearSelection(pmxDoc, sTblName)
      Call avBitmapSet(pmxDoc, sTblName, 2)
      aCalcString = "([DoubleF] - 10)"
      iok = avCalculate(pmxDoc, sTblName, aCalcString, col)
      MsgBox "1 record applied a calculation"
'
'     ---Stop the editor
      Call avSetEditableTheme(pmxDoc, Null, Null)
'
'     ---Apply a Query to the table
      aQueryString = "LongF = 0.0"
      Call avQuery(pmxDoc, sTblName, aQueryString, sel, "NEW")
      Call avGetSelection(pmxDoc, sTblName, sel)
      MsgBox CStr(sel.Count) + " records selected"
'
'     ---Check if the Summary table exists in the TOC
      aIndex = avFindDoc("sumTable")
      If (aIndex <> -1) Then
         Call avRemoveDoc("sumTable")
         MsgBox "sumTable removed from TOC"
      End If
```

SAMPLE CODE

```
        If (avFileExists("sumTable.dbf")) Then
           Call avFileDelete("sumTable.dbf")
           MsgBox "sumTable deleted from disk"
        End If
'
'     ---Define the name of the summary table to be created
        sumTblName = "sumTable"
'
'     ---Summarize the selected records in the table based upon the
'     ---LongF field.
'     ---The default operation codes will be used.
'     ---That is why fieldList1 and sumryList2 are empty colections.
        Call CreateList(fieldList1)
        Call CreateList(sumryList2)
        Set pSTable = avSummarize(pmxDoc, sTblName, _
                                  sumTblName, "dBase", UCase("LongF"), _
                                  fieldList1, sumryList2)
'
'     ---Check if the table could not be summarized
        If (pSTable Is Nothing) Then
           MsgBox "Error in summarizing " + sTblName
'     ---Handle case when the table was summarized without error
        Else
           iok = avAddDoc(pSTable)
           MsgBox "Summary table: " + sumTblName + " added to TOC"
        End If
'
'  ---Handle case when table does not exist
     Else
        MsgBox "Table: " + sTblName + " does not exist"
     End If
'
```