

University of Montana

ScholarWorks at University of Montana

Graduate Student Theses, Dissertations, &
Professional Papers

Graduate School

2003

Using Visual Basic & ArcObjects to create ActiveX controls for MAGIS-Express Import Wizard

Li Mei Piao

The University of Montana

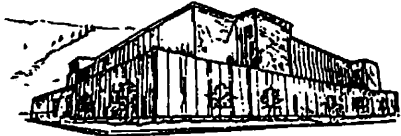
Follow this and additional works at: <https://scholarworks.umt.edu/etd>

Let us know how access to this document benefits you.

Recommended Citation

Piao, Li Mei, "Using Visual Basic & ArcObjects to create ActiveX controls for MAGIS-Express Import Wizard" (2003). *Graduate Student Theses, Dissertations, & Professional Papers*. 8357.
<https://scholarworks.umt.edu/etd/8357>

This Thesis is brought to you for free and open access by the Graduate School at ScholarWorks at University of Montana. It has been accepted for inclusion in Graduate Student Theses, Dissertations, & Professional Papers by an authorized administrator of ScholarWorks at University of Montana. For more information, please contact scholarworks@mso.umt.edu.



Maureen and Mike
MANSFIELD LIBRARY

The University of
Montana

Permission is granted by the author to reproduce this material in its entirety, provided that this material is used for scholarly purposes and is properly cited in published works and reports.

****Please check "Yes" or "No" and provide signature****

Yes, I grant permission X

No, I do not grant permission

Author's Signature: L. Mansfield

Date: 8/29/2003

Any copying for commercial purposes or financial gain may be undertaken only with the author's explicit consent.

**USING VISUAL BASIC & ARCOBJECTS TO CREATE
ACTIVEX CONTROLS FOR MAGIS-EXPRESS IMPORT WIZARD**

By

Piao Li Mei

B.S. Capital University of Medicine, P.R. China

Presented in partial fulfillment of the requirements

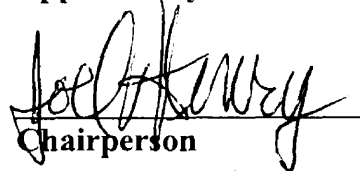
for the degree of

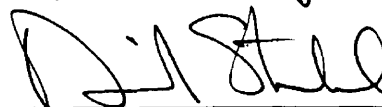
Master of Science

The University of Montana

2003

Approved by:


Chairperson


Dean, Graduate School

9-2-03

Date

UMI Number: EP39158

All rights reserved

INFORMATION TO ALL USERS

The quality of this reproduction is dependent upon the quality of the copy submitted.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if material had to be removed, a note will indicate the deletion.



UMI EP39158

Published by ProQuest LLC (2013). Copyright in the Dissertation held by the Author.

Microform Edition © ProQuest LLC.

All rights reserved. This work is protected against
unauthorized copying under Title 17, United States Code



ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 - 1346

Using Visual Basic & ArcObjects to Create ActiveX Controls for MAGIS-Express
Import Wizard

Director: Joel Henry



This project describes the creation of three ActiveX controls for the Import Wizard of MAGIS-Express, a large software package for making plans and schedules of land management and transportation-related activities on a geographic and temporal basis in the presence of multiple and sometimes conflicting objectives. All the controls are implemented in Microsoft Visual Basic 6 together with ESRI ArcObjects to handle some issues related to geographic data, like coverages and shapefiles. All three controls have their own user interfaces to gather the user's input and produce corresponding actions. The import wizard contains four forms written in Microsoft Visual FoxPro, in which the first three forms are the containers of the three ActiveX controls. The Import Wizard is a part of the MAGIS-Express software. The purpose of the Import wizard is to lead the user through a sequence of steps to import area geographic data into MAGIS-Express, such as selecting geospatial databases, selecting exit nodes, performing data checks (i.e. format, completeness, integrity), and populating MAGIS internal tables with attribute values from the associated databases.

TABLE OF CONTENTS

| | |
|---|------------|
| CHAPTER 1 INTRODUCTION | 1 |
| CHAPTER 2 MAGIS-EXPRESS & ITS IMPORT WIZARD | 4 |
| 2.1 What are MAGIS, MAGIS-Express and Import Wizard? | 4 |
| CHAPTER 3 SPECIFICATIONS AND REQUIREMENTS | 9 |
| 3.1 ActiveX Control for Import Wizard Form1 | 9 |
| 3.2 ActiveX Control for Import Wizard Form2 | 16 |
| 3.3 ActiveX Control for Import Wizard Form3 | 21 |
| CHAPTER 4 DESIGN DOCUMENTS | 26 |
| 4.1 UML & Detail Design | 26 |
| 4.1.1 <i>UML and Detail Design for ActiveX control in Import Wizard form1</i> | 26 |
| 4.1.2 <i>UML and Detail Design for ActiveX control In Import Wizard form2</i> | 49 |
| 4.1.3 <i>UML and Detail Design for ActiveX control In Import Wizard form2</i> | 65 |
| CHAPTER 5 IMPLEMENTATION & TESTING | 79 |
| 5.1 OLE (ActiveX) | 79 |
| 5.2 Architecture of the Component Object Model (COM) | 80 |
| 5.3 ActiveX Controls (OLE controls) | 85 |
| 5.3.1 <i>Types of ActiveX Components</i> | 86 |
| 5.4 Creating ActiveX Controls Using Visual Basic | 89 |
| 5.4.1 <i>Why Use Visual Basic 6</i> | 89 |
| 5.4.2 <i>Visual Basic ActiveX Control Creations Basics</i> | 90 |
| 5.5 Some of ArcObjects used in MAGIS-Express Import Wizard | 93 |
| 5.6 Testing and Debugging | 96 |
| 5.6.1 <i>Testing Strategies</i> | 96 |
| 5.6.2 <i>Debugging Method</i> | 98 |
| CHAPTER 6 DISCCUSION | 100 |
| 6.1 Why use ActiveX controls in the MAGIS-Express Import Wizard | 100 |
| 6.2 Visual Basic vs. VC++ in ActiveX Controls Implementation for the MAGIS- Express Import Wizard..... | 102 |
| 6.3 Software Reuse | 102 |
| APPENDIX A: A QUICK REFERENCE TO TABLES INVOLVED IN THIS PROJECT..... | 103 |
| REFERENCES..... | 108 |

Acknowledgements:

I would like to thank professor Joel Henry, my advisor, for his invaluable guidance, patience and encouragement.

Special thanks are also due to professor Hans Zurring for giving me such a great opportunity to take part in the development of MAGIS, also for his guidance, suggestion and comments on this paper.

Thanks also go to professor Alden Wright for accepting to be a member of my committee and his guidance.

I am particular grateful to my supervisor, Ms Judy Troutwine, for her technical guidance, suggestion, patience and support in every phase of the work.

Finally I would like to thank all the members in MAGIS software developing team for their cooperation and assistance.

CHAPTER 1

INTRODUCTION

This project illustrates the process of building the ActiveX controls for the MAGIS-Express Import Wizard. The process includes the following steps.

Multi-Resource Analysis and Geographic Information System (MAGIS) is a sophisticated planning and decision-making tool for natural resource managers that enables forest managers to schedule management activities and analyze the ecological and economic consequences of those activities. MAGIS is composed of a commercial mathematical linear programming software package, GIS software, and an optional tree growth simulator, all sharing a common graphical user interface. MAGIS provides a convenient tool for solving a wide variety of natural resources management problems. MAGIS Express, one of the two modes of MAGIS, is a simplified version of MAGIS with a timber and road emphasis and numerous pre-defined variables to streamline the model building process. MAGIS-Express contains a number of graphical user interface wizards to simplify the use of the software. The Import Wizard is one of these used for importing area data.

OLE technology was introduced by Microsoft. OLE, which stands for Object Linking and Embedding, is a collection of unrelated technologies, all based on a standard approach for working with objects. All the objects handled by OLE are component objects or window objects. These objects incorporate the same technology called Component Object Model (COM). COM makes it possible for different applications to manipulate objects they know nothing about. COM technology enables the data-centric

computing that forms the foundation for everything from future Microsoft operating systems to OLE controls [7]. ActiveX control is another name for OLE control.

The MAGIS-Express Import Wizard contains four Visual FoxPro (VFP) forms, each of the first three contain an ActiveX control created by the author using Microsoft Visual Basic in combination with ArcObjects.

ArcObjects comes with the ESRI ArcGIS software package. They are the development platform of ArcGIS DeskTop software that is comprised of three components namely ArcMap, ArcCatalog, and ArcToolBox. The reason that we can use ArcObjects with Visual Basic (Visual Basic) is that these ArcObjects are created using Microsoft Component Object Model (COM) technology mentioned above. So, any COM compliant programming language like Visual Basic can extend these ArcObjects.

Besides ArcObjects, the MAGIS-Express Import Wizard also uses an ActiveX control called MapControl. MapControl is provided by ArcGIS for creating standalone applications to display and manipulate geographic data.

The type of geographic data manipulated in this wizard project are coverages and shapefiles, both of them belong to the ArcGIS Vector data model that represents geographic phenomena with points, lines, and polygons.

The Import Wizard is designed for importing user selected area geospatial data into the MAGIS-Express working space. The purpose of the Import Wizard is to guide the user through a sequence of steps to select geospatial databases, select exit nodes, perform data checks (i.e. format, completeness, integrity), and populate MAGIS internal tables with attribute values from the associated databases [5]. The ActiveX control for *Form1* asks the user to select one “Roads” and one “Treatment Units” geospatial

database, and then copies the data to the desired ArcInfo workspace. It also performs the data conversions as needed (coverages to shapefiles, or shapefiles to coverages). The ActiveX control in *Form2* of the wizard is used to select “Exit node(s)” for each traffic type and check the connectivity. The ActiveX control for *Form3* allows the user to select the ID fields for “Treatment Units” and “Roads” data. All of the forms in this Import Wizard are Microsoft Visual FoxPro (VFP) forms supporting the ActiveX controls.

The following chapters will look into some details of developing these ActiveX controls. *Chapter2* contains a brief introduction to MAGIS, MAGIS-Express, and its Import Wizard. *Chapter3* lists the detail specifications and requirements of the Import Wizard. *Chapter4* provides the different kinds of design documents of the wizard. *Chapter5* illustrates the implementation technologies applied to this project as well as the testing strategies and debugging methods. Finally, in *Chapter6*, three related issues will be discussed, such as the reasons we use ActiveX controls in this Import Wizard, the reasons we use Visual Basic to create these ActiveX controls, and the software reuse issues.

CHAPTER 2

MAGIS-EXPRESS & ITS IMPORT WIZARD

2.1 What are MAGIS, MAGIS-Express and Import Wizard?

MAGIS is a large software package that helps natural resource managers to make plans and schedules for land management and transportation-related activities based on geographic and temporal information for different objectives, such as providing habitat for terrestrial and aquatic organisms, producing commodity outputs such as forage and sawlogs, and providing recreational access and use. MAGIS focuses on tactical planning, and provides both optimization and simulation modes [7].

MAGIS was built and introduced by the Montana Department of Natural Resources & Conservation, Forestry Division, the School of Forestry at the University of Montana, and the Intermountain Research Station, Forest Service, U.S. Department of Agriculture for the purpose of providing a sophisticated planning and decision-making tool to meet the more and more complex requirements for creating feasible forest management plans [7].

MAGIS consists of three major parts: a commercial mathematical programming package (MPSIII/pc) for solving the defined matrix (generated by the DATAFORM here based on the geospatial data imported into MAGIS), GIS software for data input and display of results (ArcGIS), and an optional tree growth simulator (SPS or an equivalent). The control programs of MAGIS are written in Visual FoxPro and DATAFORM (a mathematical database manager and a data manipulation language managing MPSIII/pc for matrix generation and report writing in this project.) that is associated with DBMSs.

The Visual FoxPro DBMS in this project is used for managing DBF format (*.dbf files, a type of database table files that can be recognized by Visual FoxPro.) spatial data linked to geographic locations, as well as providing the graphical user interfaces [7].

The following steps briefly describe how MAGIS works

1. User defined datum for making a management plan are imported into MAGIS from a GIS environment and can be modified as needed. Two kinds of data are involved. One, Forest- wide data form the basis for calculations, such as activity costs; management regimes; vegetative states and pathways; road types; construction and re-construction options; and fixed road costs. Two, Area data (GIS coverages), are used by the model to make the calculations.
2. A calculating matrix model is generated by MAGIS based on the data obtained in step 1.
3. Then the SETUP procedure in MAGIS is used to enter the specifications for building a management scenario that contains the selection of land management projects, road network link projects, and traffic routing to the traffic termination locations.
4. The MAGIS optimizer, C-WHIZ, solves the LP matrix for that management scenario according to the user selected objectives, constraints, decision variables, and special relations.
5. Finally MAGIS displays the result to the user in either tabular reports or GIS-based graphics format as preferred.

MAGIS runs on a minimum-computing platform consisting of Windows 95 (or later) environment and requires a 486 or Pentium-class processor having a minimum of 256 MB RAM.

MAGIS-EXPRESS is one of two versions of MAGIS that has a timber and road emphasis and numerous pre-defined variables to simplify the model building procedures. MAGIS-Express contains several wizards written in Visual FoxPro (VFP) to lead users step-by-step through the model solving procedures. Each of those wizards contains a friendly graphical user interface. The MAGIS-Express Import Wizard is one of them. The reason why Visual FoxPro forms are used here instead of Visual Basic forms directly is because Visual FoxPro can manage those DBF format geographic data, such as the attribute tables for Shapefiles.

The Import Wizard is designed for importing user selected area geospatial data into the MAGIS-Express working space. The purpose of the Import Wizard is to guide the user through a sequence of steps to select geospatial databases, select exit nodes, perform data checks (i.e. format, completeness, integrity), and populate MAGIS internal tables with attribute values from the associated databases [5]. The ActiveX control for *Form1* asks the user to select one “road” and one “Treatment Units” geospatial database, then copies the data to a desired ArcInfo workspace, and performs the data conversions as needed (coverages to shapefiles, or shapefiles to coverages). The ActiveX control in *Form2* of the wizard is used to select “Exit node(s)” for each traffic type and check the connectivity of the road network. The ActiveX control for *Form3* allows the user to select the ID fields for “treatment units” and “roads” data.

The menu system of MAGIS-Express and the location of the Import Wizard in this system is shown in Figure 2.2.1 – Figure 2.2.7.

Figure 2.2.1 MAGIS-Express menu system

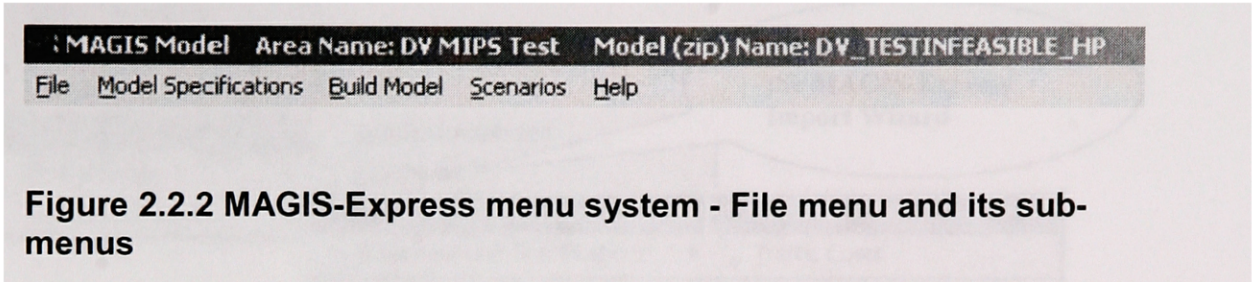


Figure 2.2.2 MAGIS-Express menu system - File menu and its sub-menus

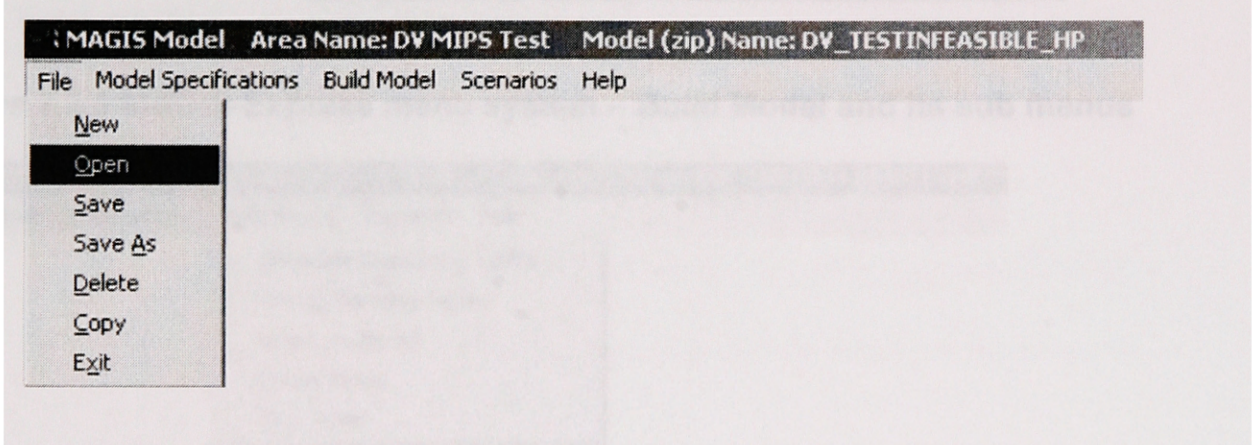


Figure 2.2.3 MAGIS-Express menu system - Model Specifications – Planning Framework – Resource Information and its sub menus

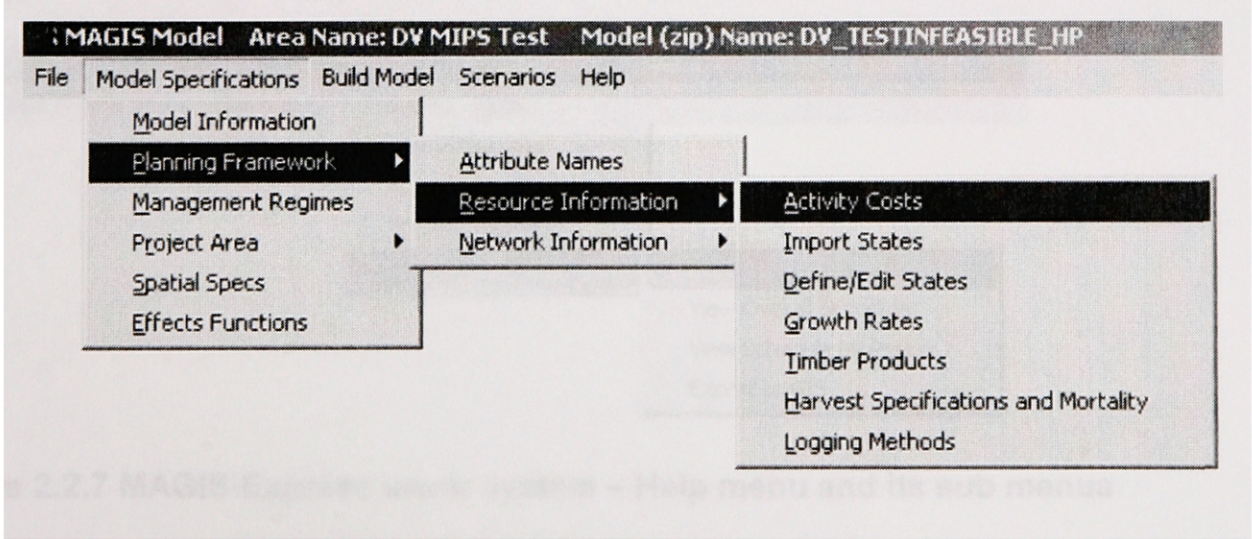


Figure 2.2.4 MAGIS-Express menu system - Model Specifications – Project Area – NetWork Specification and its sub menus

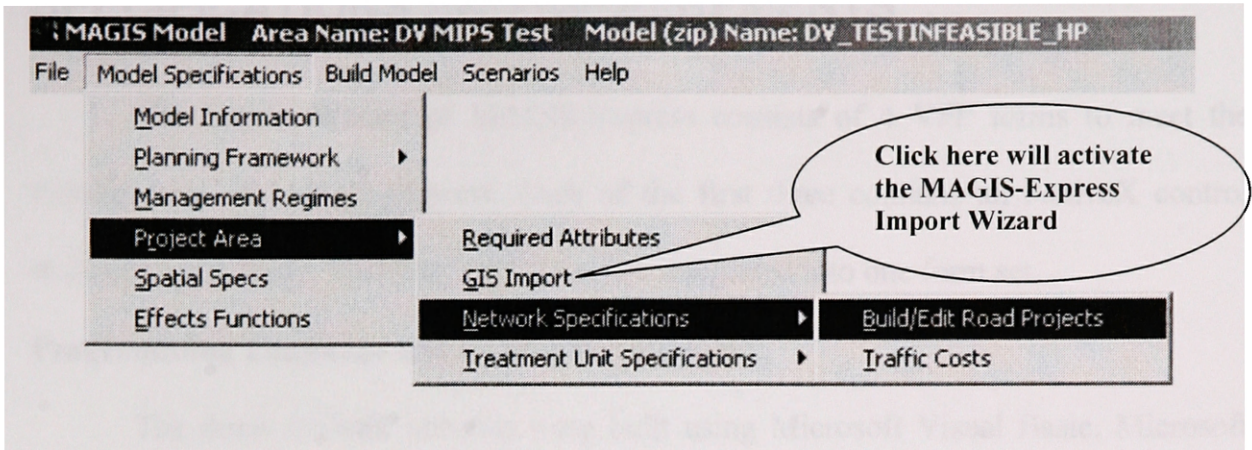


Figure 2.2.5 MAGIS-Express menu system – Build Model and its sub menus

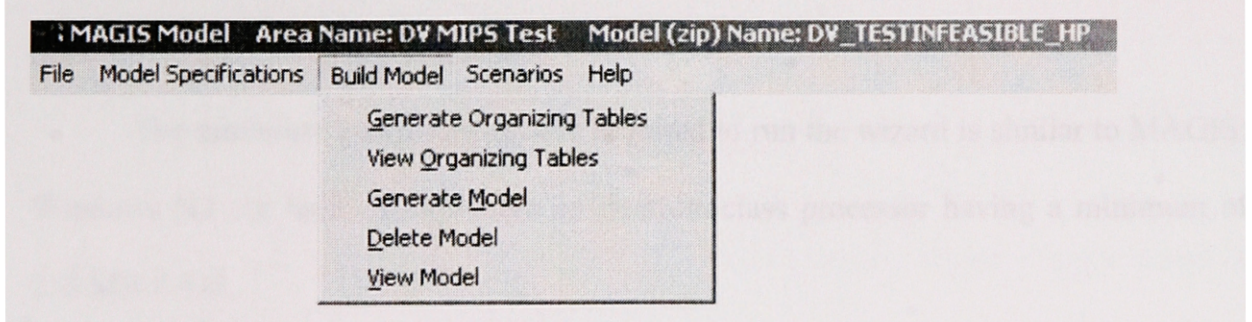


Figure 2.2.6 MAGIS-Express menu system – Scenario menu and its sub menus

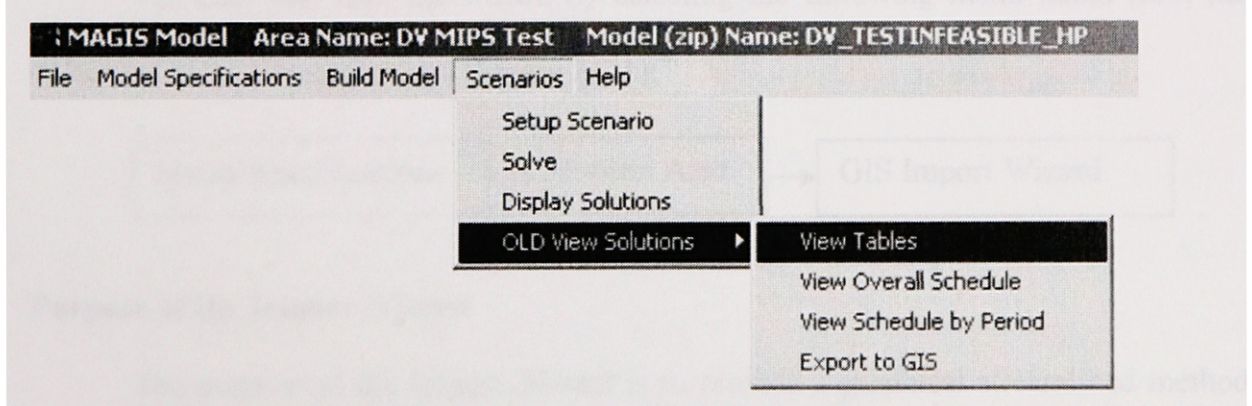
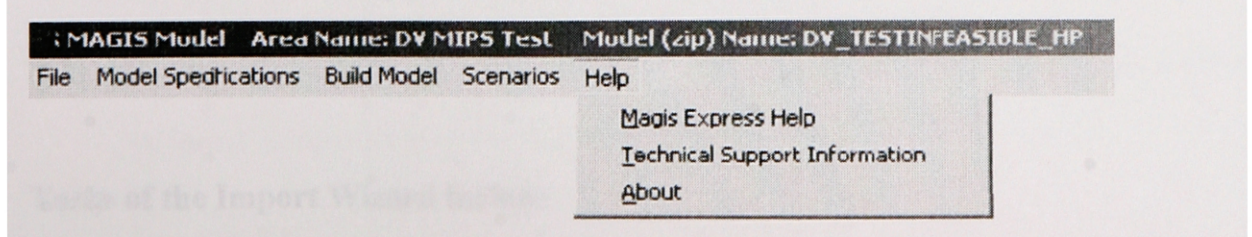


Figure 2.2.7 MAGIS-Express menu system – Help menu and its sub menus



CHAPTER 3

SPECIFICATIONS AND REQUIREMENTS

The Import Wizard of MAGIS-Express consists of 4 VFP forms to meet the requirements of MAGIS-Express. Each of the first three contains an ActiveX control built in Visual Basic. The four VFP forms are integrated into one form set.

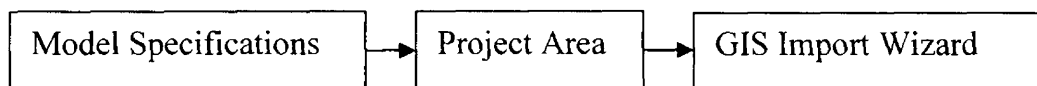
Programming Language and System Requirement:

The three ActiveX controls were built using Microsoft Visual Basic. Microsoft Visual FoxPro will be used to create the four forms of the Import Wizard with the first three forms as the ActiveX containers.

The minimum computing system required to run the wizard is similar to MAGIS: Windows NT (or later) environment on Pentium-class processor having a minimum of 256 MB RAM.

Location of the Import Wizard in MAGIS-Express Menu System

The user can load the wizard by selecting the following menu items from the MAGIS-Express Menu System (Figure 2.2.1):



Purpose of the Import Wizard

The purpose of the Import Wizard is to provide a graphical streamlined method that permits the user to select and import spatial data of a selected geographical project area for which a planning project is to be formulated through MAGIS-Express.

Tasks of the Import Wizard include

- Selecting geospatial databases, such as coverages or shapefiles, and converting their data formats into those used by MAGIS-Express.
- Selecting exit nodes for each traffic type.
- Selecting feature identification fields for the specified feature classes of the user selected import database.
- Performing data checks (i.e. format, completeness, integrity) and populating MAGIS-Express internal tables with attribute values from the user selected databases.

Error handling and screen messages

- A standard error control method is needed to handle run time errors and provide error messages to the user.
- Other messages should be prompted to the user to provide required information at run time.

3.1 ActiveX Control for Import Wizard Form1

This ActiveX control will reside in the first Visual FoxPro form of the Import Wizard. It will occupy the whole surface of VFP *Form1* of the wizard. No other graphical components will show in *Form1*. The main task of this control is to allow the user to select geospatial databases, such as coverages or shapefiles, and convert their data into those formats used by MAGIS-Express.

1. Graphical interfaces Design for Import Wizard Form1

Figure 3.1.1 shows the graphical user interface of the first ActiveX control in form1 of the MAGIS-Express Import Wizard [3].

Figure 3.1.2 shows the “Select directory interface” after the user selects the “coverage” radio button and clicks the “Browse” button.

Figure 3.1.3 shows the “Select Roads Shapefile” interface after the user selects the “shapefile” radio button and clicks the “Browse” button (the sample is for “roads” theme).

Figure 3.1.1 Graphical User interface for MAGIS-Express Import Wizard Form1

MAGIS-Express Import Wizard 1

1. Select geospatial databases

Select one Roads and one Treatment Units geospatial database.

Roads Coverage ShapeFile Browse...

Treatment Coverage ShapeFile Browse...

Help Next > Cancel

Figure 3.1.2 MAGIS-Express Import Wizard Form1 – Select Directory interface

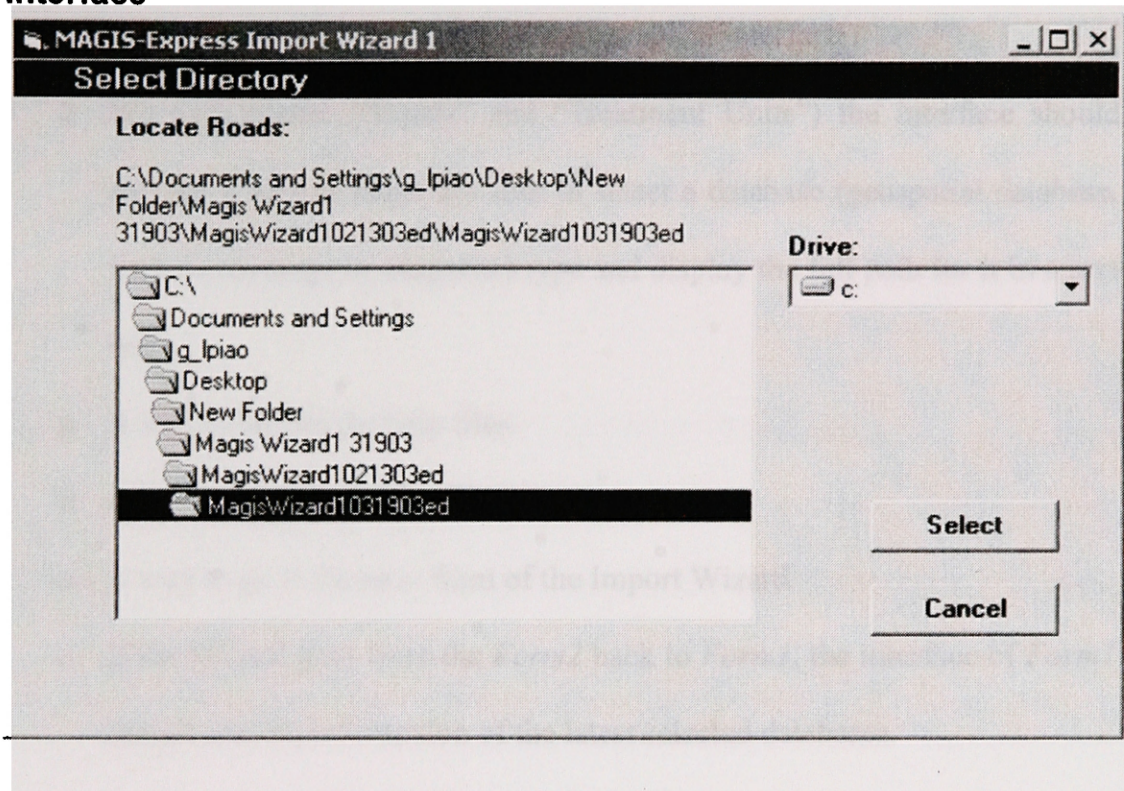
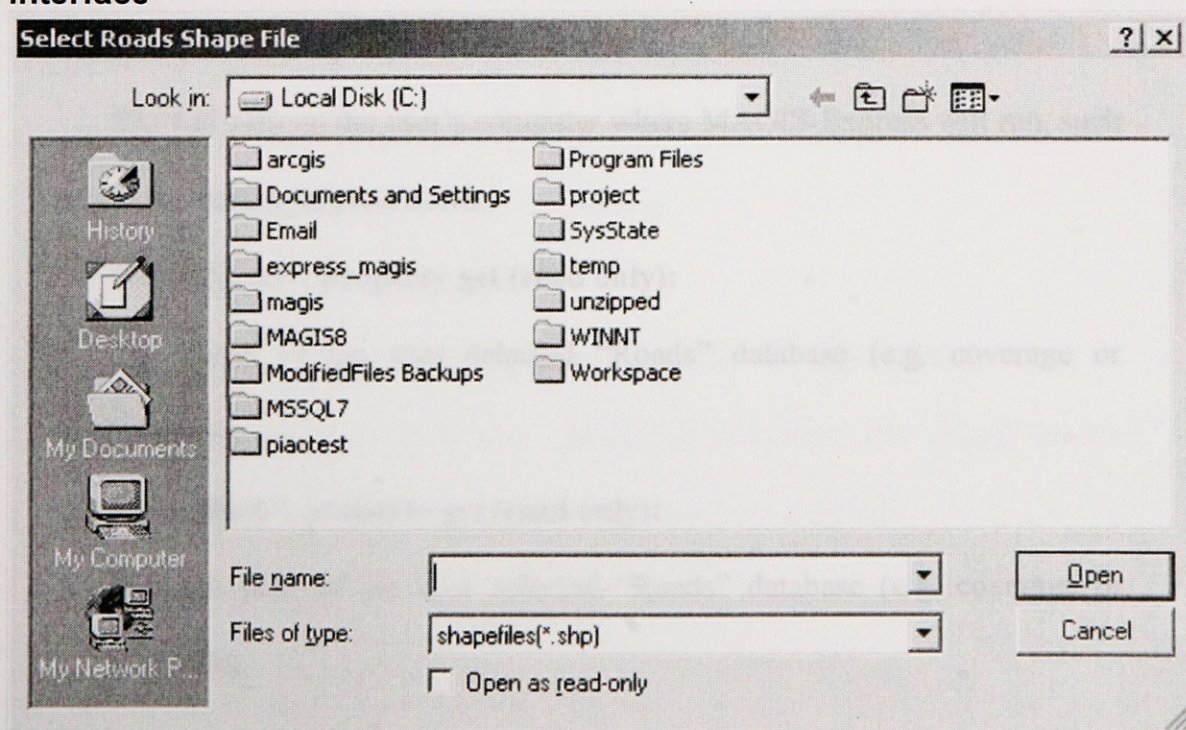


Figure 3.1.3 MAGIS-Express Import Wizard Form1 – Select Shape file interface



2. Graphical user interface will include:

- A brief user guide on the top of the graphical interface
- For each theme (“Roads” and “Treatment Units”) the interface should provide a way to allow the user to select a database (geospatial database, such as coverage or shapefile) type and display the full path for it in a text box.
- A way to launch the help files.
- A way to cancel the wizard.
- A way to go to the next form of the Import Wizard.
- If the Wizard goes from the *Form2* back to *Form1*, the interface of *Form1* should contain information of the latest selected databases.

3. Customized properties in this control should be the following:

- **“runpath”, property let (write only):**
The full path on the user’s computer where MAGIS-Express will run, such as “C:\magis_express\main.
- **“RoadName”, property get (read only):**
The name of the user selected “Roads” database (e.g. coverage or shapefile).
- **“RoadPath”, property get (read only):**
The full path of the user selected “Roads” database (e.g. coverage or shapefile).

❑ **“TreatName”, property get (read only):**

The name of the user selected “Treatment Unit” database (e.g. coverage or shapefile).

❑ **“TreatPath”, property get (read only):**

The full path of the user selected “Treatment Unit” database (coverage or shapefile).

4. Events will be raised from this ActiveX control to its container:

Some events should be raised by the ActiveX control to its container (VFP form) so that the container can handler these events, as they were their own event procedures.

- ❑ **HelpButton_Click event:** allows the container form to access help files.
- ❑ **NextButton_Click event:** allows the container to go to next form in the form set of the project.
- ❑ **CancelButton_Click event:** allows the container to exit the project.

5. Validating and checking tasks are as following:

For coverage data format:

- ❑ Ensure that any coverage selected is in an ArcInfo coverage workspace.
- ❑ Validate the presence of a map projection definition file (*.prj) in the coverage directory (or that a map projection has been defined).
- ❑ Validate for topology on “Treatment Units” coverage polygon and arc feature classes, and for “Roads” coverage arc and node feature classes.

For shapefile data format:

- ❑ Validate for presence of these files: <name>.shp, <name>.dbf, <name>.shx, and <name>.prj).

For any format:

- ❑ Validate both the user selected databases for each of the two themes, the “Treatment Units” database and the “Roads” database, have the same map projection.

Error handling:

- ❑ If any of the above validations fails inform the user and exit the Import wizard, else continue to execute task 5 as followed.

Preprocessing user selected Geospatial database:

- ❑ Creating the new ArcInfo workspace named “gis” under the relative directory: “[drive name]:” + “\magis_express\temp\magisTemp\”. If this directory doesn’t exist, create one.
- ❑ Copy the user selected geospatial databases to the new workspace.
- ❑ For each of the two themes, “Roads” and “Treatment Units”, in the new “gis” workspace, make a data set for both coverage and shapefile format.
- ❑ Rename the coverages and the root name of shapefiles in the new “gis” workspace using the MAGIS-Express specific names: “roads” for “Roads” database, “treat” for “Treatment Units” database.

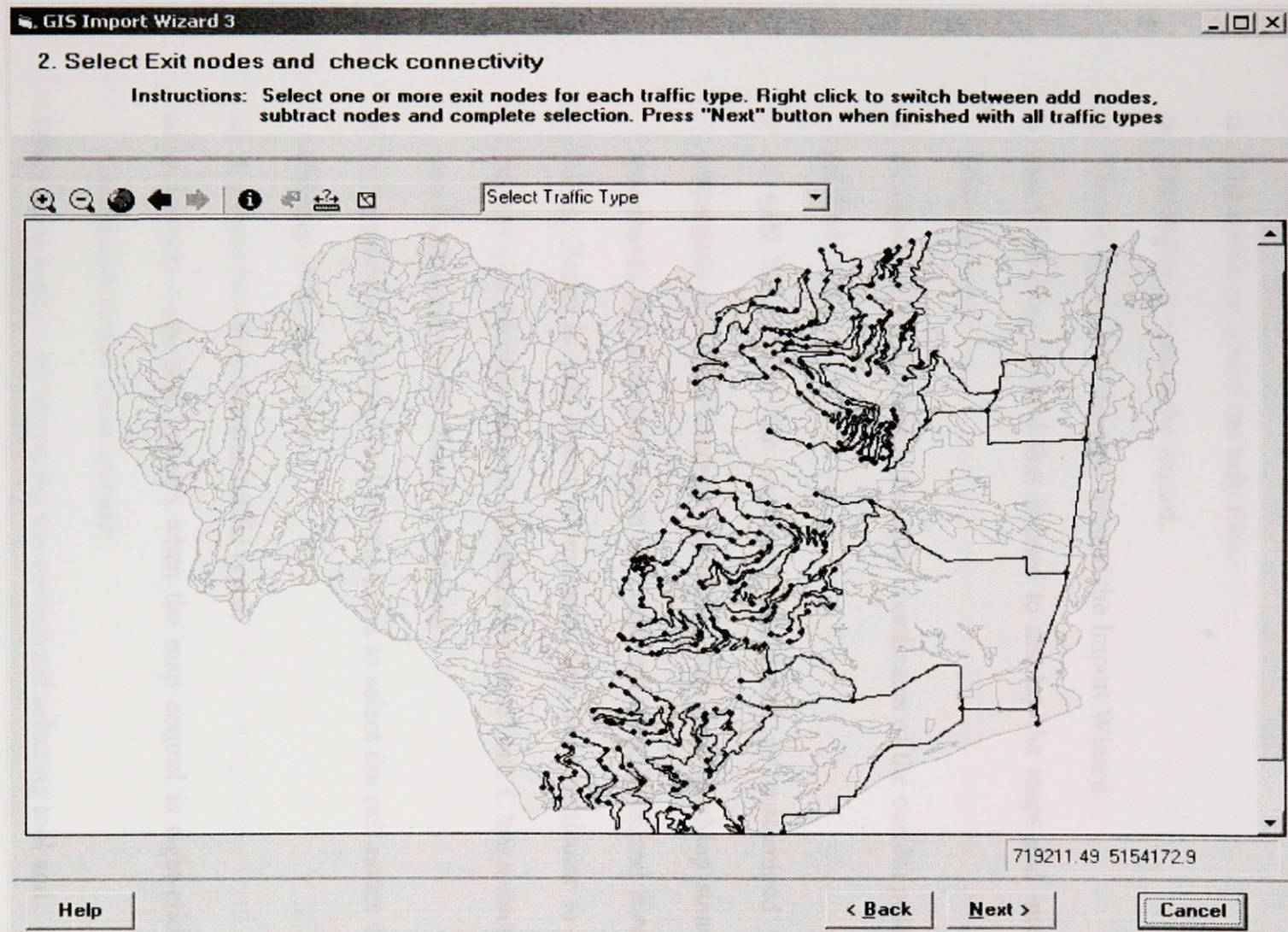
3.2 ActiveX Control for Import Wizard Form2

This ActiveX control will sit in the second Visual FoxPro form of the Import Wizard. It will occupy the whole surface of Visual FoxPro *Form2* of the wizard. No other graphical components will show in *Form2*. This control is used for selecting exit nodes for each traffic type.

1. Graphical interfaces Design for Import Wizard Form2

Figure 3.1.4 on the next page shows the graphical user interface of the second ActiveX control in *Form2* of the MAGIS-Express Import Wizard [3].

Figure 3.1.4 MAGIS-Express Import Wizard Form2 Graphical User interface



2. The graphical user interface will include:

- ❑ A brief user guide on the top of the graphical interface
- ❑ The ability to launch the help files.
- ❑ The ability to cancel the wizard.
- ❑ The ability to go to the next form of the Import Wizard.
- ❑ An ESRI Map Control that is used to display the maps and interact with the users.
- ❑ A status bar for displaying the map coordinates of the current position of the mouse.
- ❑ A tool bar control for containing the ESRI or customized map commands and tools used for managing map data. ArcMap standard commands and tools are: Zoom in, Zoom out, Pan, Back and forward zoom, Zoom to full extent. Commands and tools particular to this control are: Identify feature, Measure distance tool, Clear selection, and a customized tool - Zoom to Selected.
- ❑ A customized selecting tool for the users to select the exit nodes from the map in the map control
- ❑ A combo box that contains traffic types.
- ❑ A context menu that appears when the map control is right-clicked. Commands in that menu include:

Select more nodes – for triggering the customized selecting tool and

– adding more exit nodes to the map's selection set.

Unselect nodes – for triggering the customized selecting tool and

_ removing selected nodes from the selection set.

Done – for ending the selection process and starting further data

_ Validation and /or checking.

3. Customized properties in this control include the following:

□ **“runpath”, property let (write only):**

The full path on the user’s computer where MAGIS-Express will run, such as “C:\magis_express\main.

□ **“DoResize”, property let (write only):**

A boolean value telling the ActiveX control if it should be resizable.

□ **“mapProjUnits”, property get (read only):**

Provide the map projection unit of the user selected geospatial database, such as inches, feet, miles yard, etc.

4. Events will be raised from this ActiveX control to its container:

Some events should be raised by the ActiveX control to its container (VFP form) so that the container can handler these events, as they were their own events.

□ **HelpButton_Click event:** allows the container form to access help files.

□ **BackButton_Click event:** allows the container to go to previous form in the form set of the project.

□ **NextButton_Click event:** allows the container to go to next form in the form set of the project.

□ **CancelButton_Click event:** allows the container to exit the project.

5. Validating and checking tasks are as following:

- Check that one or more exit nodes for the current traffic type have been selected before deleting and replacing the records in table “_6a.dbf” (Appendix A). This insures that table “_6a.dbf” does not have records for that traffic type replaced unless the user has made a new selection for that traffic type.
- There must be at least one exit node for each traffic type in table “_6.dbf” that contains the selected exit nodes (Locations in the road network where the forest products are loaded out of the planning area through the road network.).
- The context menu should not be available unless the user has selected a traffic type.
- The background of the map displayed in map control will be polygon coverage in a light neutral color for geographic reference. Polygon boundaries are drawn as a separate feature in a neutral, but darker color.
- The “Roads” coverage will be drawn with symbols for existing and proposed roads in black. Road nodes are displayed with marker symbols in contrasting color. As a node is selected the identification value is displayed as a label for the node. The label font should be black and in a size easily visible at any display scale [3].
- The context menu of the map control will only be available after the user has selected a traffic type from the combo box.

- ❑ The customized selecting tool will only be active after the user has selected a traffic type from the combo box.
- ❑ Commands and tools on the toolbar control will be available at any time. If, during the node selection process, the user selects a command or tool on the toolbar, such as “Pan Zoom in”, then to continue the selection, the user must use the context menu to reactivate the selecting tool.
- ❑ The “Zoom to selected” tool will only be available when at least one “exit nodes” is selected.

6. Two DBF tables are used in this ActiveX control (Appendix A)

- ❑ *_6.dbf* – Used to provide traffic type values.
- ❑ *_6a.dbf* - Used to store selected “exit node” information.

3.3 ActiveX Control for Import Wizard Form3

This ActiveX control will sit in the third Visual FoxPro form of the Import Wizard. It will occupy the whole surface of *Form3* of the wizard. No other graphical components will show in *Form3*. The main task of this control is to allow the user to select the identification fields for “Treatment Units” and “Roads”.

1. Graphical interfaces Design for Import Wizard Form3

The following Figure 3.1.5 shows the graphical user interface of the third ActiveX control in *Form3* of the MAGIS-Express Import Wizard [3].

Figure 4.1.5 Graphical User Interface for MAGIS-Express Import Wizard

Form3

GIS Import Wizard 3

3. Select ID fields
Select the identification fields for treatment units and roads

Treatment Units ID field

From Node Field

To Node Field

Help < Back Next > Cancel

2. The graphical user interface will provide the following features:

- A brief user guide on the top of the graphical interface
- The ability to launch the help files.
- The ability to cancel the wizard.
- The ability to go to the next form of the Import Wizard.

- A combo box populated with all the non-internal field names of the “treatment units” database selected by the user as the candidates for the identification field for the “treatment units” data set.
- Two combo boxes populated with all the non-internal field names of the “Roads” database selected by the user as the candidates for the identification field for the “Roads” data “From node field” and “To node field”.

3. Customized properties in this control should be the following:

- **“runpath”, property let (write only):**

The full path on the user’s computer where MAGIS-Express will run, such as “C:\magis_express\main”.
- **“SelTreatIDFldName”, property let/get (read/write):**

The selected identification field name for “Treatment Units”.
- **“SelRdToIDFldName”, property let/get (read/write):**

The selected identification field name for “Roads” “To node field”.
- **“SelRdFromIDFldName”, property let/get (read/write):**

The selected identification field name for “Roads” “From node field”.
- **“delAddedFlds”, property let (write):**

A Boolean value that indicates if the old selected ID fields need to be removed from the database.

4. Events will be raised from this ActiveX control to its container:

Some events should be raised by the ActiveX control to its container (VFP form) so that the container can handler these events, as they were their own events.

- ❑ **HelpButton_Click event:** allows the container form to access help files.
- ❑ **BackButton_Click event:** allows the container to go to previous form in the form set of the project.
- ❑ **NextButton_Click event:** allows the container to go to next form in the form set of the project.
- ❑ **CancelButton_Click event:** allows the container to exit the project.

5. Validating and checking tasks are as following:

- ❑ The coverage or shapefile internal field names can not be put into the combo boxes drop down lists, such as “*covname_ID*”.
- ❑ Ensure that the user has selected one identification field for “Treatment Units” and two for “Roads”. If not, print a message to the screen to try again or cancel.
- ❑ The user selected ID fields for both the shapefile DBF tables and the coverage feature classes must be renamed as the MAGIS_Express VFP procedures standard names namely: “*CUT_UN_ID*” for “treatment units” IDs, “*FROM_NODE*” for “Roads” “*FROM_NODE*” ID and “*TO_NODE*” for “Roads” “*TO_NODE*” ID.
- ❑ If the attribute tables contain any fields named the same as the above standard field names, only those standard names can be populated into any of the three suitable combo boxes.

- If the window control comes back from Wizard *Form4* (A VFP form without any ActiveX control in it), the previous selected and renamed fields will be removed from the database DBF and attribute tables, according to the value of the property “delAddedFlds”.

After the user has selected all three ID fields, the control must add a new field to the DBF and attribute tables for each data type (coverage and shapefile) of the two selected databases (“Roads” and “Treatment units”) using the three standard names (“CUT_UN_ID”, “FROM_NODE”, “TO_NODE”) as needed for the new field name.

CHAPTER 4

DESIGN DOCUMENTS

4.1 UML & Detail Design

4.1.1 UML and Detail Design for ActiveX control in Import Wizard Form1

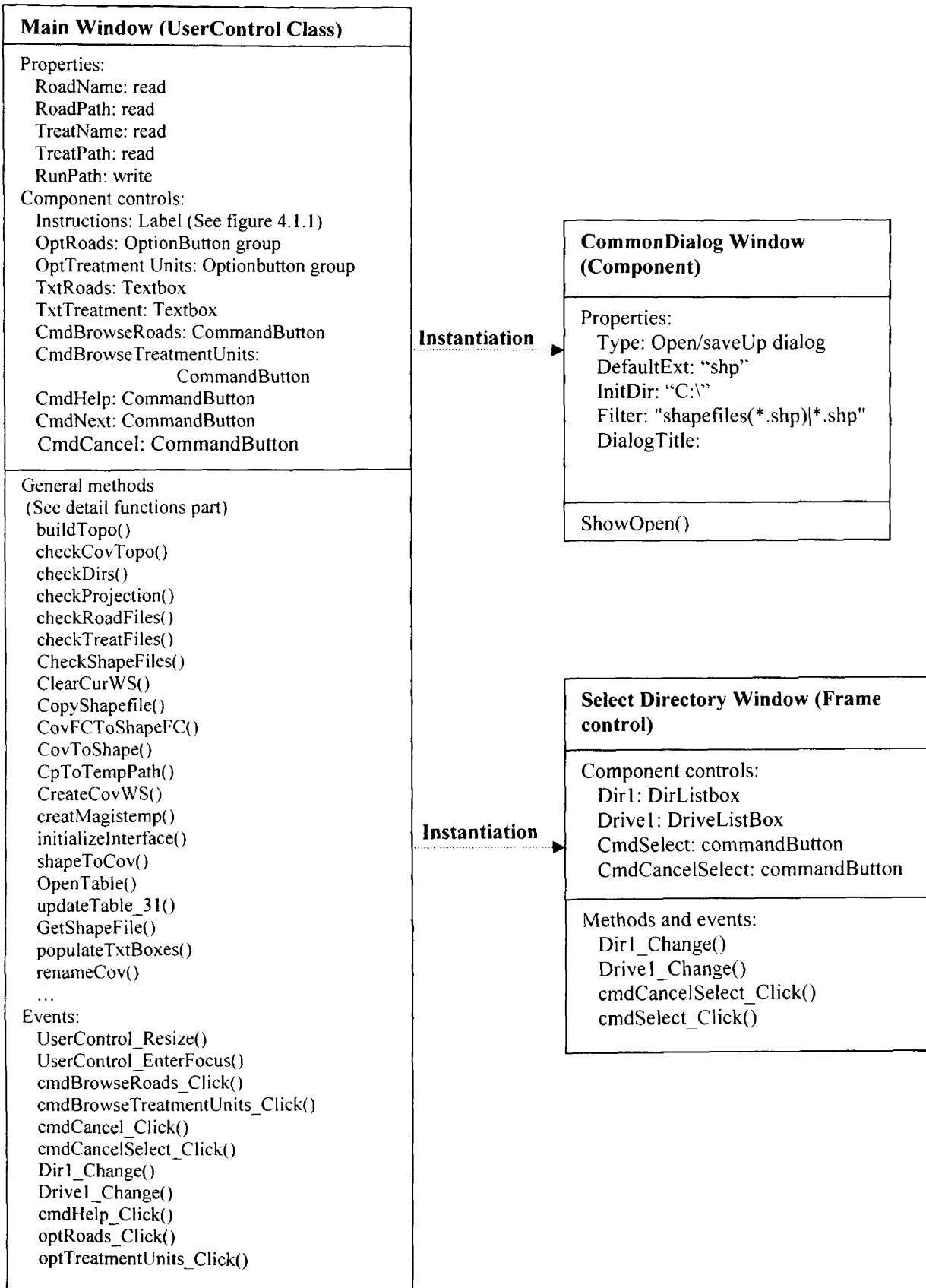
The only module in this ActiveX control is the UserControl class.

1. UML

The UML for this control contains three window classes. The main window represents the UserControl Object. The other two are MS Common Dialog Box window and the Select Directory (Figure 4.1.1.1).

The main window, the UserControl Object, will open the Common Dialog window or the Select Directory window after the corresponding command button in the main window was clicked.

Figure 4.1.1.1 UML for ActiveX control 1 in Import Wizard

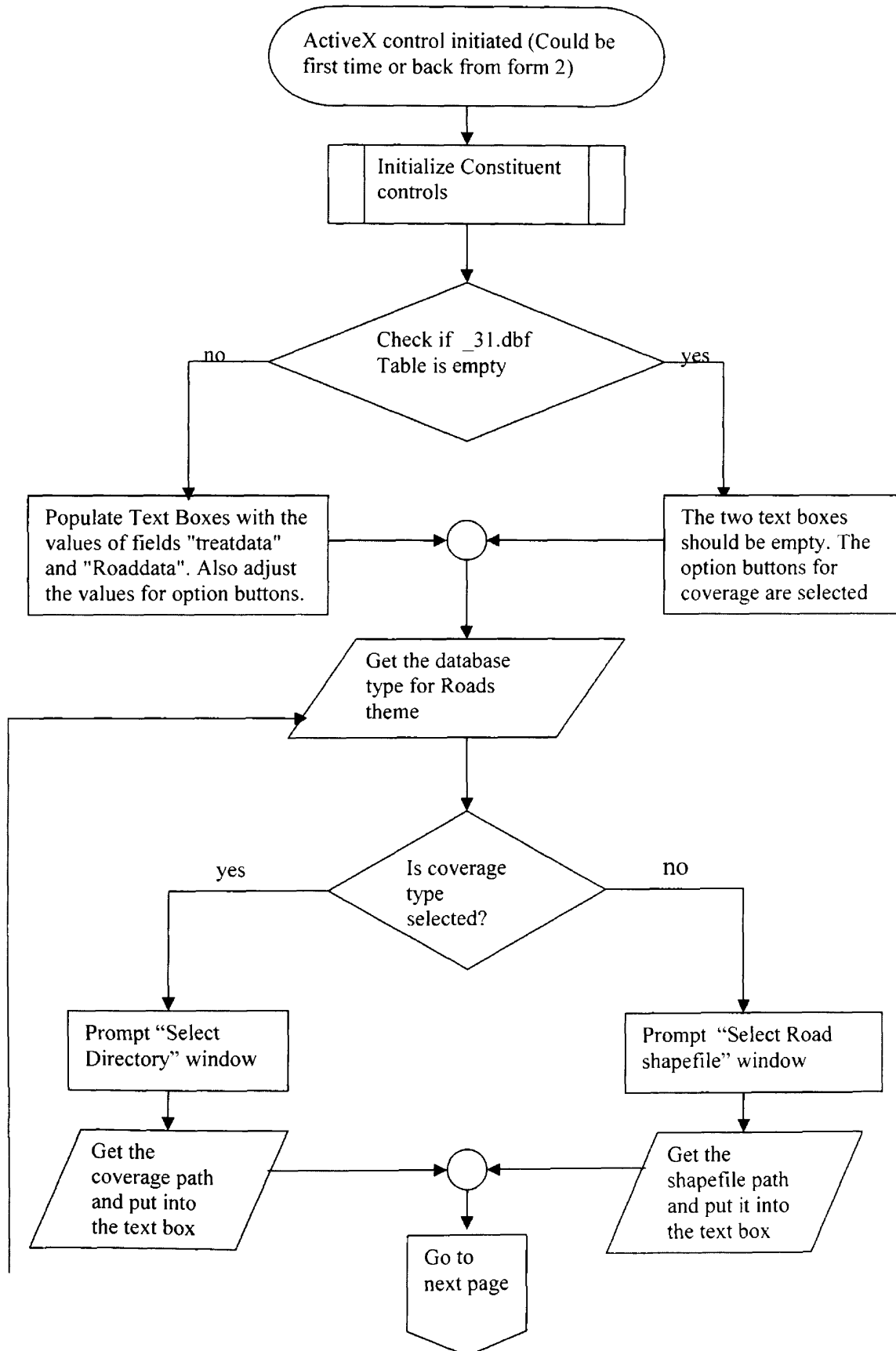


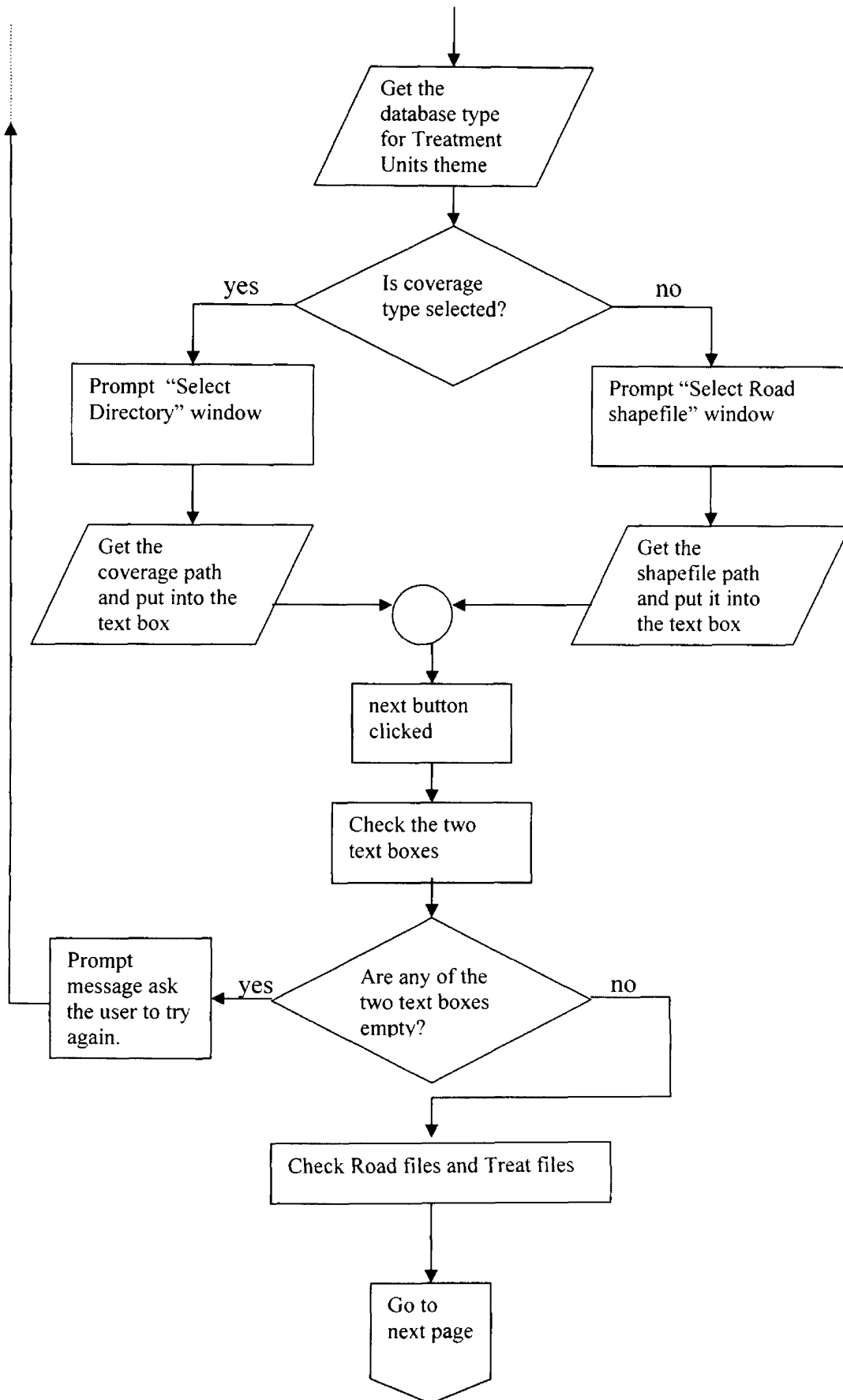
2. Flowchart

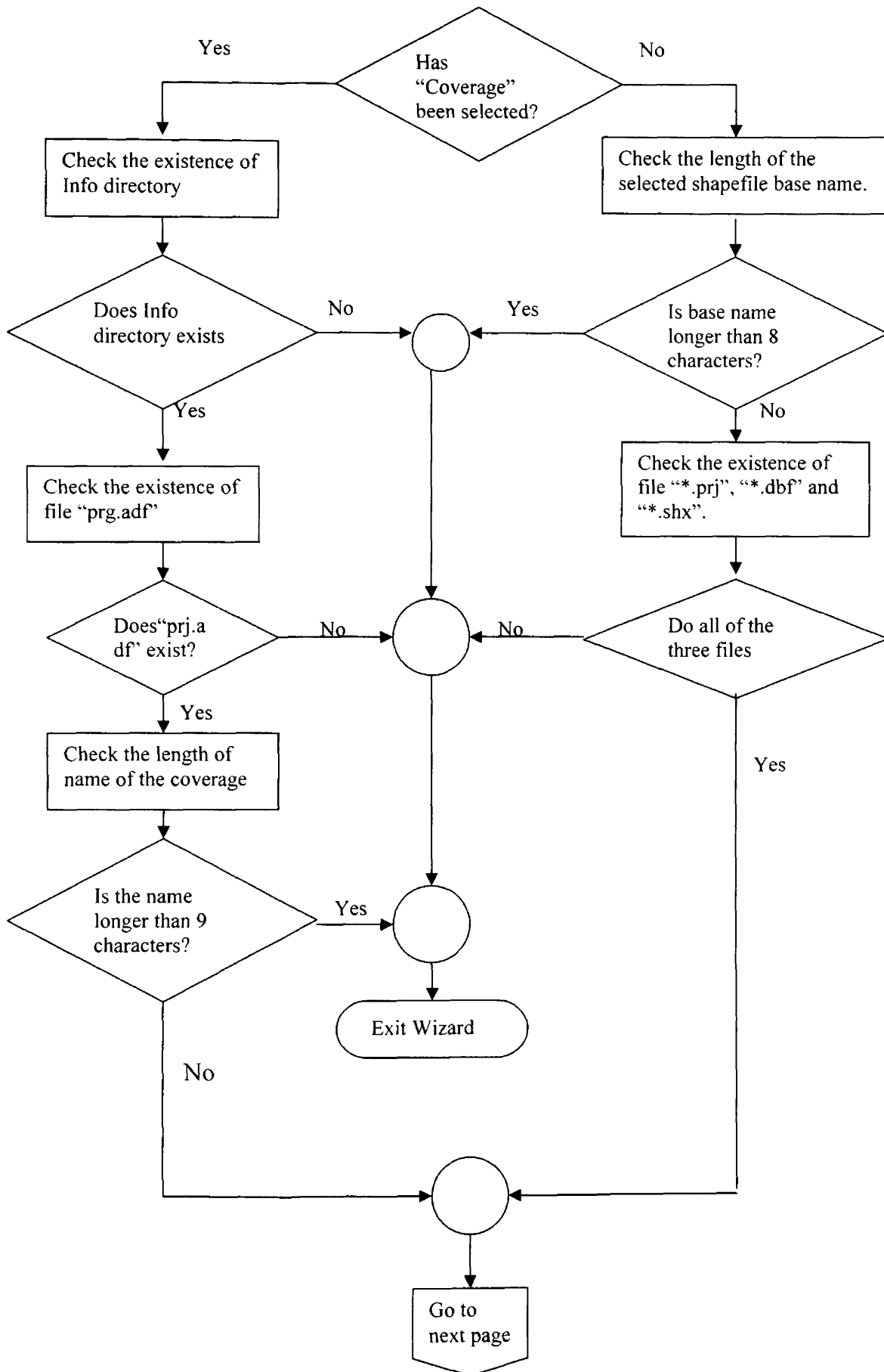
The Flowchart displays the actions that are taken after this ActiveX control is initiated (Figure 4.1.1.2).

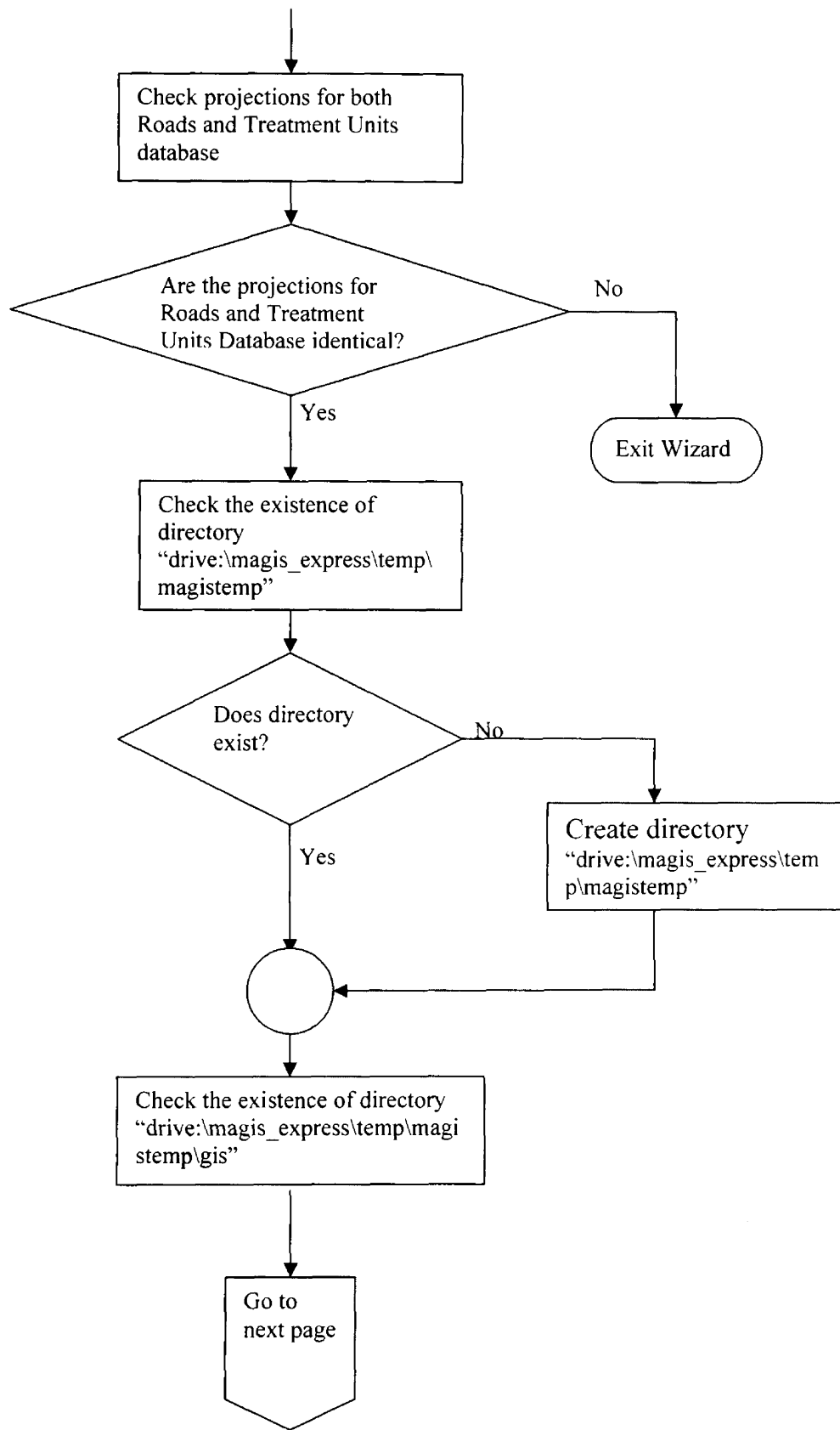
After ActiveX control is initiated (Could be first time or back from form 2). The control will initialize the component controls in the interface according to a serial checking (such as path checking...). After the user selected spatial databases for both “Treatment Units” and “Roads”, the control will execute required validations. If both types of databases pass the validations, the control will copy these databases to designated MAGIS-Express directory and make the data conversion there as needed, such as coverage file into shapefile or vice versa. The original paths for the two databases will be written into table “_31gis.dbf” (For description of this table, see Appendix A.) for later use. Then the control will raise “NextButton_Click” event to its container form and from there the program flow will go to the next form in the same wizard.

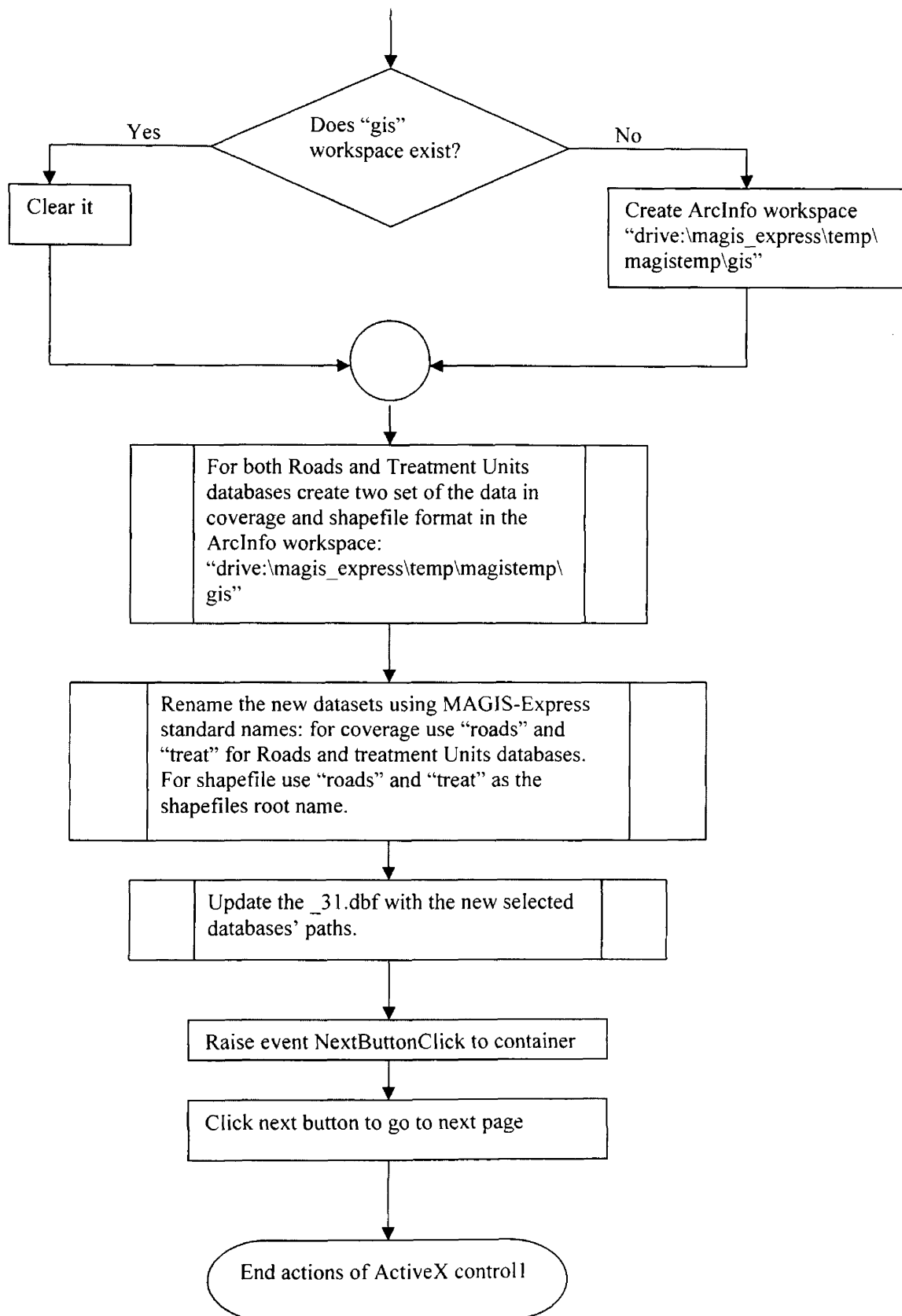
Figure 4.1.1.2 Flowchart for ActiveX control 1 in Import Wizard











3. Detailed functions of this ActiveX control:

Properties:

❑ **Property Get RoadName()**

Returns the root name (without file extensions) of the “Road” coverage or shapefiles.

❑ **Property Get RoadPath()**

Return the original path of the “Roads” coverage or shapefiles on the user's computer.

❑ **Property Let runPath()**

Returns the path where MAGIS-Express runs.

❑ **Property Get TreatName()**

Returns the root name (without file extensions) of the “Treatment Unit” coverage or shapefiles.

❑ **Property Get TreatPath()**

Returns the original path of the “Treatment Units” coverage or shapefiles on the user's machine.

General Functions:

❑ **Private Function buildTopo(covPath As String, CovName As String, topoNum As Integer, covType As Integer) As Boolean**

Parameter list:

String CovPath: Path of the coverage that needs the new topology.

String CovName: Name of the coverage

Integer: topoNum: Type of the needed topology

Integer: covType: Type of the coverage (“Roads” or “Treatment
Units”)

Output: Return a Boolean value indicating if the function is successful.

Description: Build the topology for a coverage feature class

□ **Private Function checkCovTopo(covPath As String, CovName As
String, covType As Integer) As Integer**

Parameter list:

String CovPath: Path of the coverage that needs the new topology.

String CovName: Name of the coverage

Integer: covType: Type of the coverage (“Roads” or “Treatment
Units”)

Output: Return an Integer value indicating the needed topology type.

Description: Checks the topology for a coverage and returns the topology
type wanted.

□ **Private Function checkDirs() As Boolean**

Parameter list: Null

Output: Return a Boolean value indicating if the function is successful.

Description: Check if any of the two path text boxes is empty. If so return
false, else return true.

❑ **Private Function checkProjection(roadCov As Boolean, treatCov As Boolean) As Boolean**

Parameter list:

Boolean roadCov: True if it's a "Roads" coverage, false if it's a "roads" shapefile.

Boolean treatCov: True if it's a "Treatment Units" coverage, false if it's a shapefile.

Output: Return a Boolean value indicating if the function is successful.

Description: Call getProjectionName() function and check the projections for "Roads" and "Treatment Units" databases to see if they are identical.

❑ **Private Function checkRoadFiles() As Boolean**

Parameter list: Null.

Output: Return a Boolean value indicating if the function is successful.

Description: Call getPathOrName(), CheckShapeFiles() function and check if the files selected for the "Roads" database are in correct format. If not return false to let caller cancel the event.

❑ **Private Function CheckShapeFiles(tempPath As String, tType As Integer) As Boolean**

Parameter list:

String tempPath: Path of the shapefile to be checked.

Integer tType: Indicating if it's the "Roads" or "Treatment Units" database.

Output: Return a Boolean value indicating if the function is successful.

Description: Check if the file *basename.shx*, *basename.prj*, *basename.dbf* exist.

□ **Private Function checkTreatFiles() As Boolean**

Parameter list: Null.

Output: Return a Boolean value indicating if the function is successful.

Description: Call `getPathOrName()`, `CheckShapeFiles()` and check if the files selected for “Treatment Units” database is in correct format. If not return false to let caller cancel the event.

□ **Private Sub clearCurWS(myPath As String, keyFile As String)**

Parameter list:

String myPath: Directory that needs to be cleared.

String keyFile: File prefix that needs to be deleted.

Output: Null.

Description: Delete all sub directories under "myPath".

□ **Private Function CopyShapefile(SourcePath As String, sourceShapeName As String, destPath As String, newName As String) As Boolean**

Parameter list:

String SourcePath: Directory path of the copied shapefile.

String sourceShapefile Name: Name of the copied shapefile.

String destPath: Directory path of the copy shapefile.

String newName: New shapefile name.

Output: Return a Boolean value indicating if the function is successful.

Description: Copy shapefiles to “\magistemp\gis” directory as needed.

- **Private Function CovFCToShapeFC(destPath As String, destFCName As String, sourcePath As String, sourceFCName As String) As Boolean**

Parameter list:

String destPath: Destination directory.

String destFCName: Destination feature class name.

String sourcePath: Source directory.

String sourceFCName: Source feature class name.

Output: Return a Boolean value indicating if the function is successful.

Description: Convert coverage feature class to shapefile feature class.

- **Private Function CovToShape(sourcePath As String, CovName As String, destWSPath As String, covType As Integer) As Boolean**

Parameter list:

String sourcePath: Source path of the converted coverage.

String Covname: Source coverage name.

String destWSPath: Destination workspace path.

Integer covType: Database type (“Roads” or “Treatment Units”).

Output: Return a Boolean value indicating if the function is successful.

Description: Call CovFCToShapeFC() and convert coverage to shapefile.

- **Private Function CreateCovWS(wsName As String) As Boolean**

Parameter list:

String wsName: Create new coverage workspace (ArcInfo workspace).

Output: Return a Boolean value, indicating if the function is successful.

Description: Create a coverage workspace given a workspace name.

□ **Private Function creatMagistemp() As Boolean**

Parameter list: Null.

Output: Return a Boolean value, indicating if the function is successful.

Description: Create Magistemp subdirectory under the directory: "X:\magis\temp\".

□ **Private Sub delShpFiles(filePath As String)**

Parameter list:

String filePath: Directory path of the shapefiles to be deleted.

Output: null

Description: Delete old shapefiles in "..magis\temp".

□ **Private Function delShpTableFd(pTablename As String, cboType As Integer) As Boolean**

Parameter list:

String pTablename: Name of the table from which the internal fields need to be deleted.

Integer cboType: Databases type ("Roads" or "Treatment units").

Output: Return a Boolean value- indicating if the function is successful.

Description: Delete the internal ID fields for shapefiles to prevent the duplicated ID fields.

□ **Private Function findDir(myPath As String, key As String) As Boolean**

Parameter List:

String myPath: The directory where the “key” directory might be.

String key: The key directory looked for.

Output: Return a Boolean value- indicating if the function is successful

Description: Check to see if the directory “key” exists in “myPath”.

□ **Private Function getPathOrName(tempPath As String, pathType As Integer) As String**

Parameter list:

String TempPath: The directory used to find the file or directory name.

Integer pathType: Type of the Return value.

Output: Return a String, directory path name.

Description: Return a directory path or a file or directory name according to the second parameter "pathType".

□ **Private Function getProjectionName(fileType As Integer, isRoad As Boolean) As String**

Parameter list:

Integer fileType: Coverage or shapefile.

Boolean isRoads: Flag to indicate if it is a “roads” database

Output: return a String, projection name.

Description: Return the spatial reference name.

□ **Private Sub GetShapeFile(themeType As Integer)**

Parameter list: Integer themeType: database type, such as coverage or shapefile

Output: null.

Description: Open the open/save dialog box, that allows the user to choose the file path.

□ **Private Sub initializeInterface()**

Parameter list: Null.

Output: Null

Description: Initialize the values of the controls on the interface.

□ **Private Function OpenTable(strWorkspace As String, strTableName As String) As ITable**

Parameter list:

String strWorkspace: The path of the table to be opened.

String strTableName: The name of the table.

Output: Return an object of type “Itable”, an object of table.

Description: Open a DBF table, return the opened table to calling procedure.

□ **Private Sub populateTxtBoxes()**

Parameter list: Null.

Output: Null

Description: Initialize the values of the textbox controls with the values in “_31gis.dbf” (See Appendix A for table description).

- **Private Function renameCov(SourcePath As String, sourceCvrgName As String, covType As Integer) As Boolean**

Parameter list:

String SourcePath: Directory path of the coverage to be renamed.

String sourceCvrgName: Name of the renamed coverage.

Integer covType: Type of the renamed coverage (“Roads” or “Treatment Unit”)

Output: Return a Boolean value indicating if the function is successful.

Description: Rename “Treatment Units” coverage to "treat", and “Roads” coverages to “Roads”.

- **Private Function renameFiles(folderPath As String) As Boolean**

Parameter list:

String folderPath: Directory path of the shapefiles to be renamed.

Output: Return a Boolean value indicating if the function is successful.

Description: Rename shape files to treat.shp and roads.shp.

- **Private Function shapeToCov(inShpName As String, outCovName As String, isRoad As Boolean) As Boolean**

Parameter list:

String inShpName: Directory path of the shpfiles to be converted.

String outCovName: The out coverage name.

Boolean isRoad: Indicates if it is for “Roads” database.

Output: Return a Boolean value indicating if the function is successful.

Description: Convert shapefile to coverage.

- **Private Sub updateTable_31(roadChanged As Boolean, treatChanged As Boolean)**

Parameter list:

Boolean roadChanged: Indicates if the “Roads” database path has changed.

Boolean treatChanged: Indicates if the “Treatment Units” database path has changed.

Output: Null.

Description: Convert shapefile to coverage.

General events of the constituent controls in the ActiveX control:

- **cmdBrowseRoads_Click()**

Triggered when the upper Browse button is clicked. This will prompt the “select directory” or “select shapefile” interface for “Roads” database.

- **cmdBrowseTreatmentUnits_Click()**

Triggered when the lower Browse button is clicked. This will prompt the “select directory” or “select shapefile” interface for “Treatment Units” database.

- **cmdNext_Click()**

Triggered when the “Next” button is clicked. When this event occurs the following tasks will be performed:

- a. Call checkDirs() to validate if both text boxes are empty, if so prompt the user and ask the user to try again, else go to b.

- b. Call checkRoadFiles() to validate the selected “Roads” database:

If the user selected coverage for “Roads” database:

- a) Validate the coverages are in an ArcInfo coverage workspace. If not exit wizard, else go to b).
- b) Validate the “prj.adf” exists in coverage. If not exit wizard, else go to c)
- c) Validate the name of the coverage is longer than 9 characters. If so exit wizard, else go to c.

If the user selected shapefile for “Roads” database:

- a) Validate the name of the base name of the selected shapefile is longer than 8. If so exit wizard, else go to b).
- b) Call CheckShapeFiles() to validate “.prj”, “.dbf” and “.shx” files exist. If any of these doesn’t exist, that means the selected shapefile is not valid, so exit wizard, else go to c.

- c. Call checkTreatFiles() to Validate the selected “Treatment Units” database:

If the user selected coverage for “Treatment Units” database:

- a) Validate the coverages are in an ArcInfo coverage workspace. If not exit wizard, else go to b).
- b) Validate the “prj.adf” exists in coverage. If not exit wizard, else go to c).
- c) Validate the name of the coverage is longer than 9 characters. If so exit wizard, else go to d.

If the user selected shapefile for “Treatment Units” database:

- a) Validate the name of the base name of the selected shapefile is longer than 8 characters. If so exit wizard, else go to b).
- b) Call CheckShapeFiles() to validate “.prj”, “.dbf” and “.shx” files exist. If any of these doesn’t exist, that means the selected shapefile is not valid, so exit wizard, else go to step d.
- d. Call checkProjection() to validate if the projections for both “Roads” and “Treatment Units” databases are identical. If not, exit wizard, else go to step e.
- e. Call findDir() to validate if the subdirectory “\MAGISTemp” exists under the directory “drive:\magis\temp”, if not create one.
- f. Call findDir() to validate if the subdirectory “\gis” exists under the directory “drive:\magis\temp\MAGISTemp”, if it is, clear it. If not, call createCovWS() to create one.
- g. For both “Roads” and “Treatment Units” databases there must be two the data sets in the “drive:\magis\temp\MAGISTemp\gis” workspace, one in shapefiles format, the other in coverage format.

Methods to be used include:

- a) CopyShapefile() to copy the selected shapefiles to the specified workspace.

- b) ShapeToCov() to convert the selected shapefile into coverage format.
- c) BuildTopo() to build the topology for the new converted coverages.
- d) CovToShape(): convert coverages into shapefile format.
- e) CpToTempPath(): copy user selected coverages to the Specified workspace: “*drive:\magis\temp\MAGISTemp\gis*”
- h. Rename the coverages and shapefiles in the “*drive:\magis\temp\MAGISTemp\gis*” using the MAGIS-Express standard names: for coverage use “roads” and “treat” for “Roads” and “Treatment Units” databases; for shapefiles use “roads” and “treat” as the root names.
- i. Call “updateTable_31()” function to update the databases path records in table “_31gis.dbf” (See Appendix A.) with current user selected the two database paths.
- j. Raise “NextButtonClick” event.

□ **cmdCancel_Click()**

Triggered when the cancel button is clicked. The “CancelButtonClick” event will be raised.

□ **cmdHelp_Click()**

Triggered when the help button is clicked. The “HelpButtonClick” event will be raised.

□ **cmdCancelSelect_Click()**

Triggered when the cancel button in the “select directory” interface is clicked. This will close the “select directory” interface and return to wizard *Form1*.

□ **cmdSelect_Click()**

Triggered when the select button in the “select directory” interface is clicked. This will close the “select directory” interface and populate the text box in the same row with the directory the user selected, then return to wizard *Form1*.

□ **Dir1_Change()**

Triggered when the directory selection is changed in the directory list of “Select Directory” interface. This will change the logo on top of the DirectoryListbox to display the current highlighted directory.

□ **Drive1_Change()**

Triggered when the drive selection changed in the drive box of “Select Directory” interface. This will change path the “DirectoryListbox” to display the Drive in the Drive box is changed.

□ **optRoads_Click & optTreatmentUnits_Click**

Triggered when the option buttons are clicked. If the option buttons for coverage are clicked, the cmdBrowse_clicked() event will prompt the “Select Directory” window (Figure 3.1.2). If the option buttons for shapefile are clicked, the cmdBrowse_clicked() event will prompt the “Select Shapefile” window (Figure 3.1.3).

□ **UserControl_EnterFocus()**

Triggered after ActiveX control in the container receives focus. The “initializeInterface()” will be called to initialize the graphical interface including:

- a. Initialize all the constituent controls in the ActiveX control: The “Select Directory” interface is invisible.
- b. Call “populateTxtBoxes()” to initialize the text boxes using the values in “_31gis.dbf” table (Appendex A.). The values for the options buttons will correspond to the values in the two text boxes. If the two text boxes are empty, the first two option buttons (for coverage) should be selected.

□ **UserControl_Resize()**

Triggered when the ActiveX control is resized.

Events to be raised by this ActiveX control to its container:

□ **NextButtonClick()**

Raised when the “Next” button is clicked.

□ **CancelButtonClick()**

Raised when the “Cancel” button is clicked. This will cause the wizard to exit.

□ **HelpButtonClick()**

Raised when the “Help” button is clicked. This will prompt the help files.

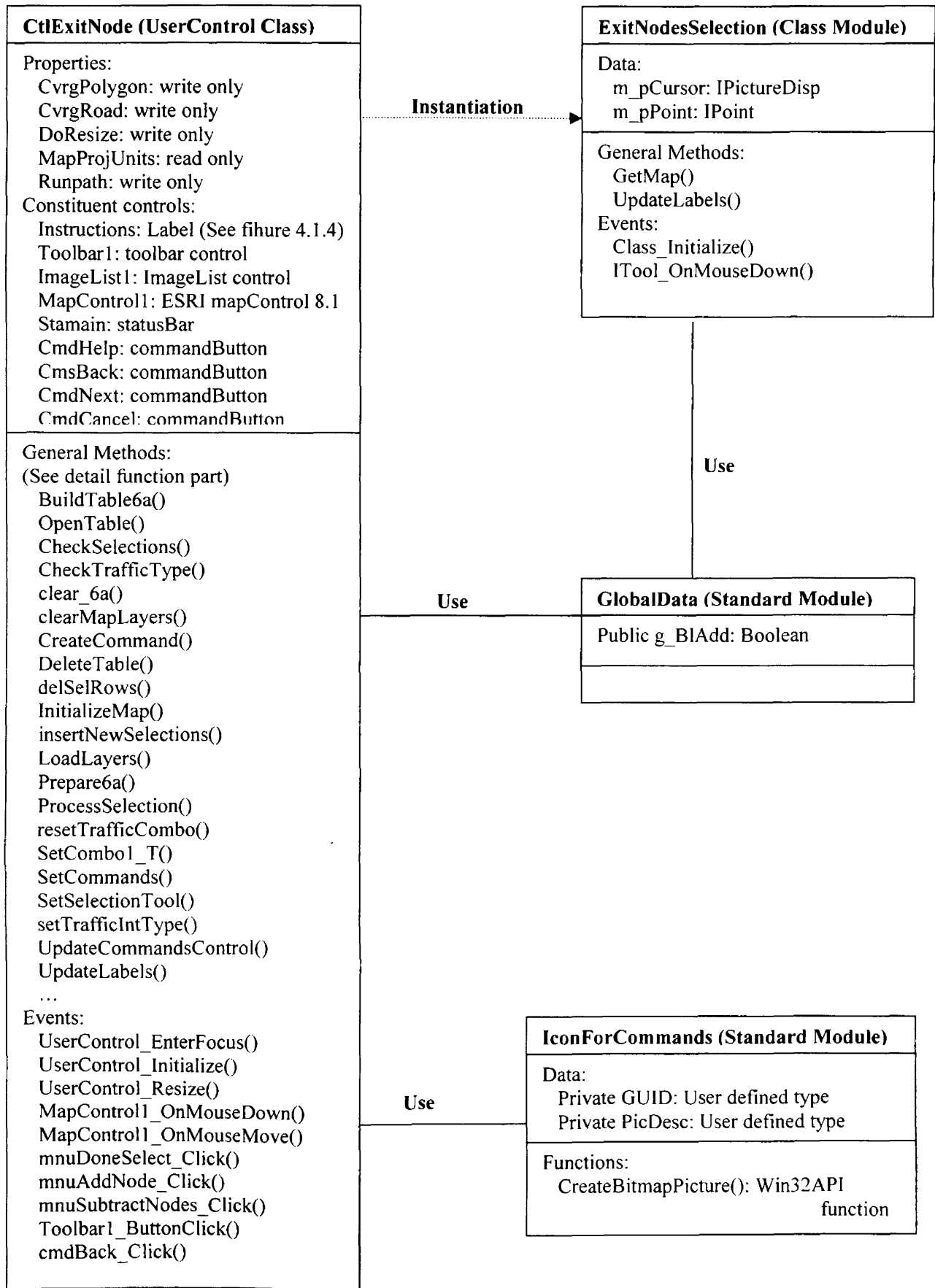
4.1.2 UML and Detail Design for ActiveX control In Import Wizard Form2

This ActiveX control consists of UserControl class, ExitNodeSelection class, IconForCommands module and globalData module.

1. UML

The “UserControl” object provides the main graphical user interface and contains the constituent controls. It also initiates an object of “ExitNodeSelection” class. The standard module “globalData” contains a global variable “g_BIAdd” that will be used by the “UserControl” object and “ExitNodeSelection” object to determine the status of the customized selection tool. The “ExitNodeSelection” class module is a template of the customized selection tool that can be instantiated by the “UserControl” object. The “IconForCommands” module contains a method to load icons for commands and tools (Figure 4.1.2.1).

Figure 4.1.2.1 UML for ActiveX control 2 in Import Wizard

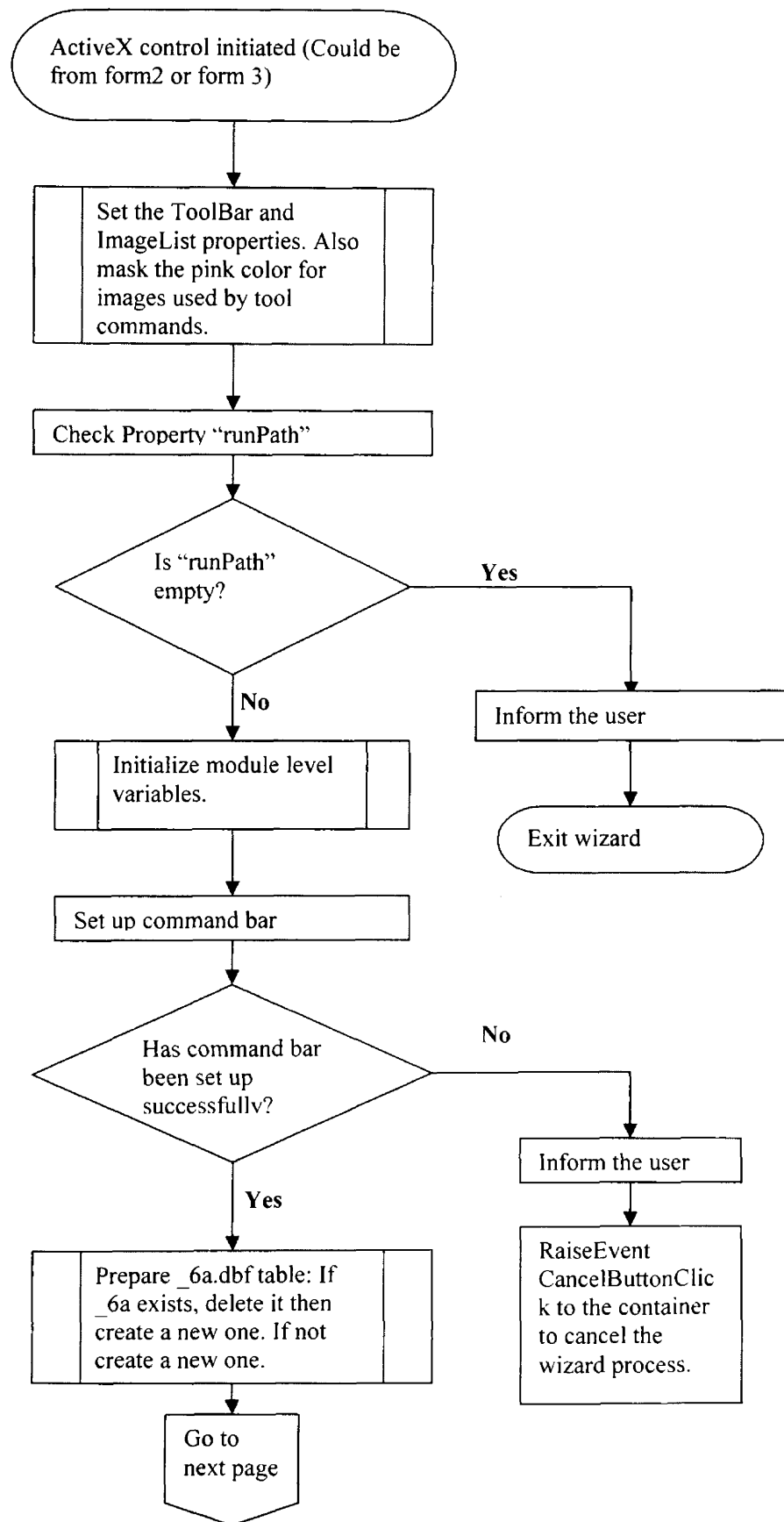


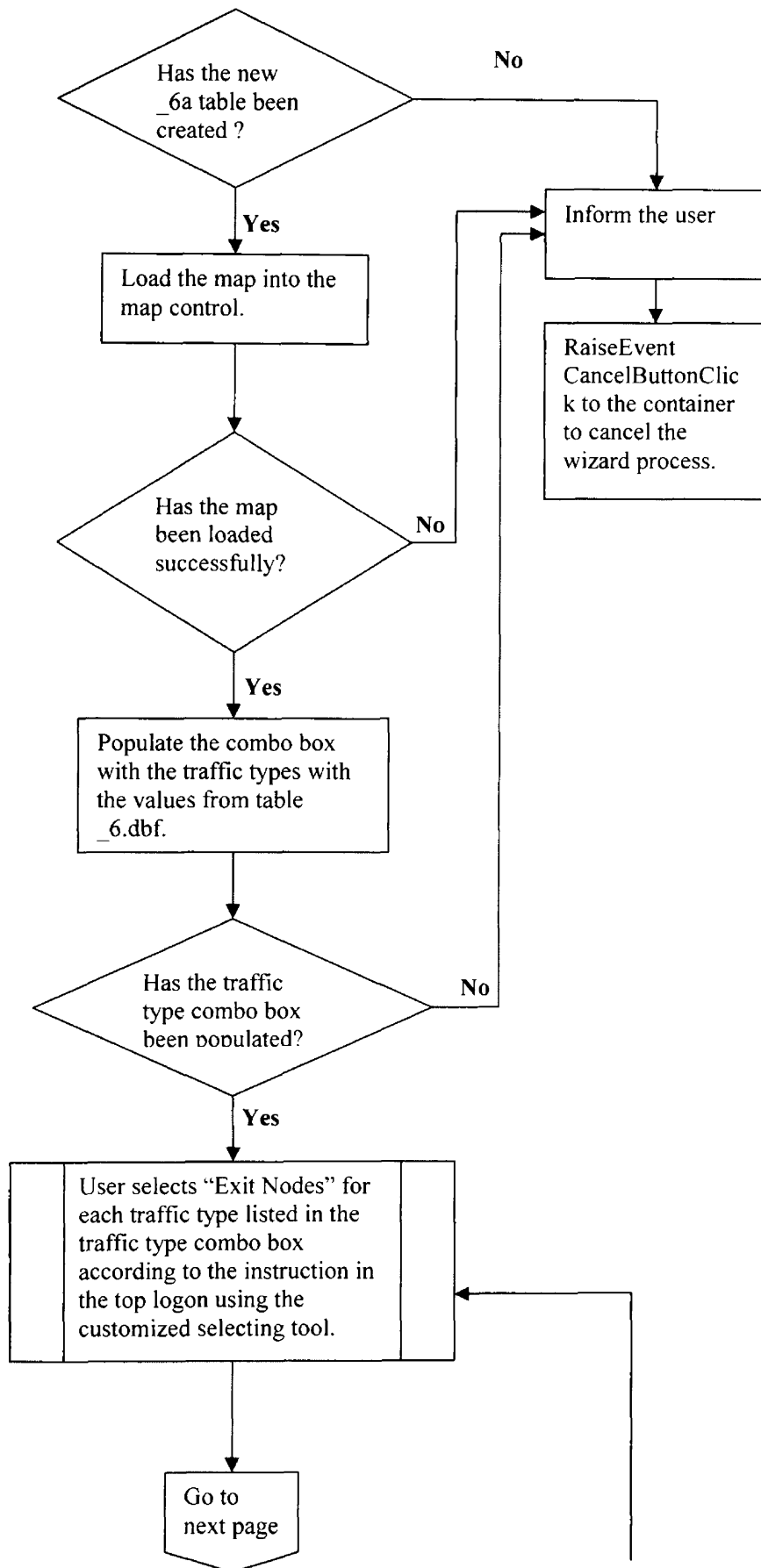
2. Flowchart

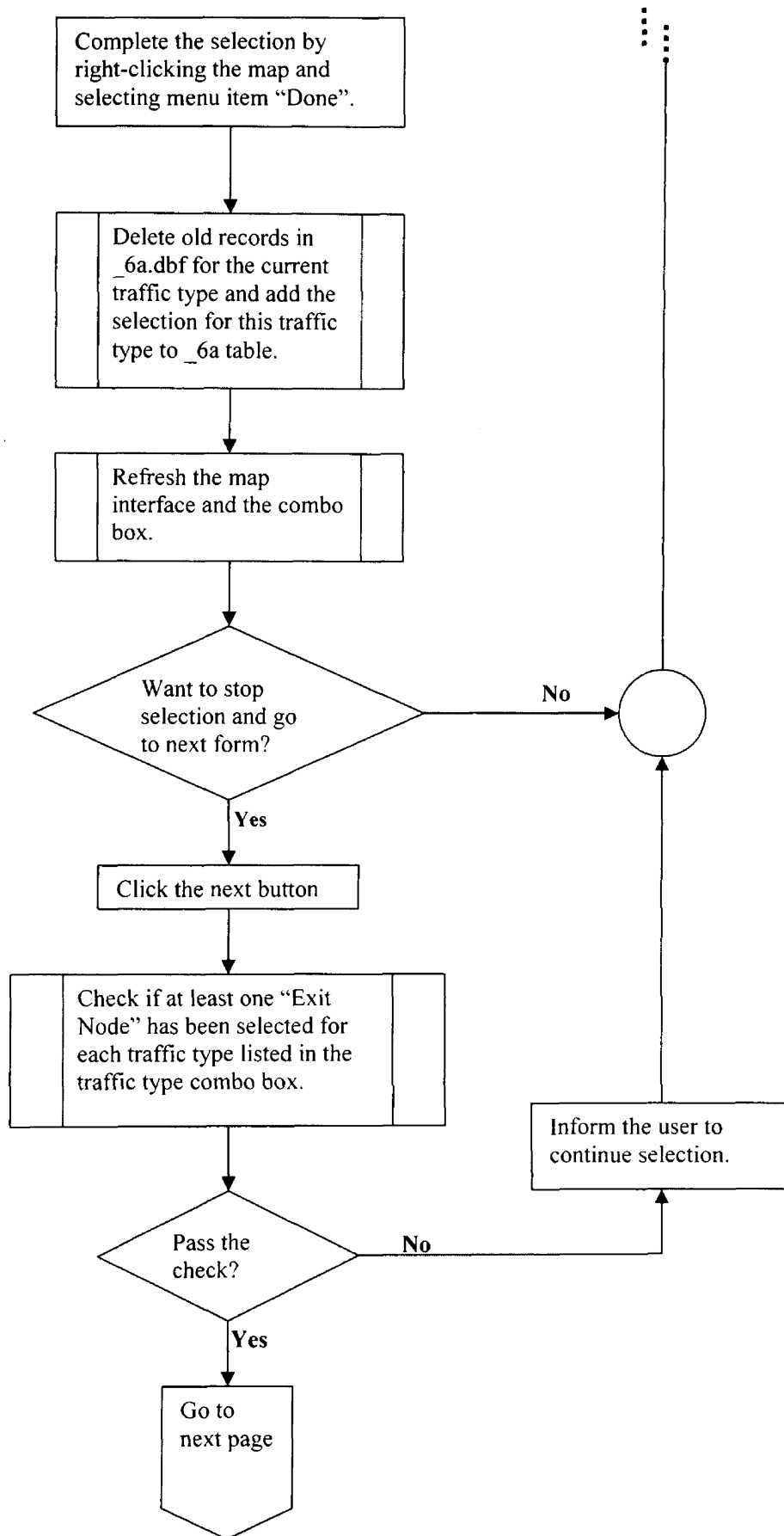
Flowchart displays the normal action steps executed after this ActiveX control is initiated (Figure 4.1.2.2).

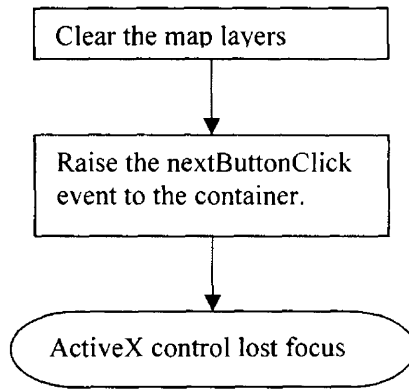
After ActiveX control is initiated (Could be first time or back from *Form2*). The control will initialize the component controls in the interface according to a serial validation rules (such as required properties have valid values available). Then the control will set up the commands and tools on the tool bar and prepare the tables that will used in this control. After that map layers will be loaded into the map control. The user then can select exit nodes (Geological locations on road network for loading products out of forest) from the map for every traffic type. The user finishes selection by clicking the next button. Then the control will validate the user's selection. If the selection passes the validation the control will raise "NextButton_Click" event to its container form and from there the program flow will go to the next form in the same wizard.

Figure 4.1.2.2 Flowchart for ActiveX control 2 in Import Wizard









3. Detailed attributes and functions of the UserControl Object in this ActiveX control:

Properties of the UserControl object:

□ **Property Let DoResize**

Indicates if the actions in event UserControl_Resize() will be executed.

□ **Property Get mapProjUnits()**

Returns "mapUnits" property of the "UserControl".

□ **Property Let runpath()**

The relative path where MAGIS-Express exists on the user's computer.

General Functions of UserControl object:

□ **Private Function BuildTable6a(path As String, name As String) As ITable**

Parameter list:

String path: Directory path of the DBF table.

String name: Name of the DBF table.

Output: Return an " Itable" object.

Description: Build a new dbf table named "_6a.dbf" (See Appendix A.).

□ **Private Function checkSelections() As String**

Parameter list: Null.

Output: Return a String that defines a chosen traffic type.

Description: Check the “_6a.dbf” table to see if there are no nodes selected for every traffic type. Return the traffic type from which no node was selected.

□ **Private Sub clear_6a()**

Parameter list: Null.

Output: Null.

Description: Delete all records in table “_6a.dbf” (Appendix A for table description).

□ **Private Sub clearMapLayers()**

Parameter list: Null.

Output: Null.

Description: Clear the map layers in the MapControl.

□ **Private Sub CreateCommand(pCommand As ICommand, bSeparator As Boolean)**

Parameter list:

ICommand pCommand: A command object.

Boolean bSeparator: Indicates if a separator is needed.

Output: Null.

Description: Create commands that will be used in this project and put them to a toolbar control.

□ **Private Function DeleteTable(path As String, name As String) As Boolean**

Parameter list:

String path: Directory path of the table to be deleted.

String name: Name of the table to be deleted.

Output: Return a Boolean value indicating if the function is successful.

Description: Delete an old dbf table “_6a.dbf”.

❑ **Private Sub delSelRows()**

Parameter list: Null.

Output: Null.

Description: Delete the rows in table “_6a.dbf” whose Int_t_type is IntType.

❑ **Private Function InitializeMap() As Boolean**

Parameter list: Null.

Output: Null.

Description: Call LoadLayers() to initialize the map in map control.

❑ **Private Sub insertNewSelections()**

Parameter list: Null.

Output: Null.

Description: Insert the selected features to dbf table "_6a.dbf".

❑ **Private Function OpenTable(strWorkspace As String, strTableName As String) As Itable**

Parameter list:

String strWorkspace: Directory path of the coverage workspace.

String strTableName: Name of the table to be opened.

Output: Return an Itable object.

Description: Open a DBF table named strTableName in “strWorkspace”.

□ **Private Function Prepare6a() As Boolean**

Parameter list: Null.

Output: Null.

Description: Call OpenTable(), BuildTable6a(), DeleteTable() to open dbf table “_6a.dbf”. If table doesn't exist, build one or delete it and create a new one.

□ **Private Sub ProcessSelection()**

Parameter list: Null.

Output: Null.

Description: Call function resetTrafficCombo(), insertNewSelections(), and UpdateLabels() to store the selected features to dbf table “_6a.dbf”.

□ **Private Sub resetTrafficCombo()**

Parameter list: Null.

Output: Null.

Description: call UpdateLabels() to clear the labels on the map and reset the “listIndex” of the “traffic type” combo box.

□ **Private Function selectionIsEmpty() As Boolean**

Parameter list: Null.

Output: Return a Boolean value indicating if the current map selection is empty.

Description: Check if the current map selection is empty and return a Boolean result.

❑ **Private Function SetCommands() As Boolean**

Parameter list: Null.

Output: Return a Boolean value indicating if the function is successful.

Description: Call CreateCommand() and UpdateCommandsControl() to set the command tool bar.

❑ **Private Sub setSelectionTool()**

Parameter list: Null.

Output: Null.

Description: Activate the customized selection tool.

❑ **Private Sub setTrafficIntType()**

Parameter list: Null.

Output: Null.

Description: Get the “int_t_type” value for the current traffic type from array “TrArray”.

❑ **Private Sub UpdateLabels()**

Parameter list: Null.

Output: Null.

Description: Update labels on the map displayed by the map control when a selection has been made.

General events of the constituent controls in the ActiveX control:

❑ **UserControl_EnterFocus()**

Triggered after the ActiveX control was activated. In this event the map in the Map Control will be initialized and the traffic type combo box will be populated.

□ **UserControl_Initialize()**

Triggered when the ActiveX control is initiated. This event initializes the constituent control in the user interface.

□ **UserControl_Resize()**

Triggered when the size of the ActiveX control is changed. This event repositions all the constituent controls in the user interface of the ActiveX control.

□ **cboTrafficType_Change()**

Triggered when the content of the traffic combo box changes. This event keeps the content in the combo box from being edited.

□ **cmdBack_Click()**

Triggered when the back button is clicked. It leads to wizard *Form1*. It also raises “BackButtonClick” event to its container.

□ **cmdCancel_Click()**

Triggered when the Cancel button is clicked. It exits the Import Wizard. It also raises the “CancelButtonClick” event to its container.

□ **cmdHelp_Click()**

Triggered when the Help button is clicked. It also raises the “HelpButtonClick” event to its container.

□ **cmdNext_Click()**

Triggered when the next button is clicked. The following tasks will be executed in order:

- a. Call `checkSelections()` to validate at least one exitnode was selected for each traffic type. If not, remind the user of the traffic type for which no node has been selected and ask the user to try again. Else, call `clearMapLayers()` to clear the selection of the focus map in the map control.
- b. Raise “NextButtonClick” event to the container.

□ **MapControl1_OnMouseDown()**

Triggered when the user clicks the object with either mouse button. This event decides when the context menu will be prompted up.

□ **MapControl1_OnMouseMove()**

Invoked when the user moves the mouse over the control. This event will make the status bar changes its texts to the map coordinates of current position that the mouse points to.

□ **mnuAddNode_Click()**

Occurs when the user presses and then releases a mouse button over menu item “mnuAddNode” in the context menu. This event activates the “Add Node” status of the customized selection tool.

□ **mnuDoneSelect_Click()**

Occurs when the user presses and then releases a mouse button over menu item “mnuDoneSelect” in the context menu. This event activates the following actions:

- a. Call function “setTrafficIntType()” to add the current traffic type (“IntType”) into array “trArray()”.
- b. Call function “delSelRows()” to delete the existing records in table _6a.dbf whose “Int_t_type” is “IntType”.
- c. Call function “ProcessSelection()” to process the new selections of the focus map in the map control. The process steps includes:
 - a) Call “insertNewSelections()” to insert the new selections.
 - b) Call “UpdateLabels()” to update the labels the selection tool made.
 - c) Call “resetTrafficCombo()” to reset the traffic combo box again to its initial state.

□ **mnuSubtractNodes_Click()**

Occurs when the user presses and then releases a mouse button over menu item “mnuSubtractNodes” in the context menu. This event activates the “Subtract Node” status of the customized selection tool.

□ **Toolbar1_ButtonClick**

Occurs when the user presses and then releases a mouse button over a command or tool item on the tool bar. It activates the clicked command or tool and sets it as the map control’s current tool.

Events to be raised by this ActiveX control to its container:

❑ **NextButtonClick()**

Raised when the “Next button” is clicked. It will lead the wizard to *Form3*

❑ **BackButtonClick()**

Raised when the back button is clicked. It will lead the wizard back to *Form1*.

❑ **CancelButtonClick()**

Raised when the “Cancel” button is clicked. This will cause the wizard to exit.

❑ **HelpButtonClick()**

Raised when the “Help” button is clicked. This will allow the container form to prompt the help files.

4. The ExitNodesSelection class in this ActiveX control:

This class is used as the template for instantiating the customized selection tool. It should have the ability to add or delete exit nodes to the current selections of the focus map in the Map Control. It will implement ArcObject “Itool” and “Icommand” interfaces. This tool user a global Boolean flag “g_BIAdd” in the standard module “globalData” to decide whether to add new nodes to the map selection or delete a node from the selection.

Some important functions and events in this class are:

- ❑ **GetMap():** Return the focus map.

- ❑ UpdateLabels(): Show labels for the selected exit nodes.
- ❑ Class_Initialize(): Set the cursor for this tool.
- ❑ ITool_OnMouseDown(): When the tool is activated, the user can select features by clicking the mouse.

2. The standard module “globalData” in this ActiveX control:

This module contains only a global Boolean variable “g_BlAdd” used by the “UserControl” object and “ExitNodesSelection” object to decide whether the status of the selection tool is "add to selection" or “deleted from selection”.

3. The standard module “IconForCommands” in this ActiveX control:

This standard module contains a method for creating icons for commands and tools. A “ Win32API” function “OleCreatePictureIndirect()” is called from this module to finish the task.

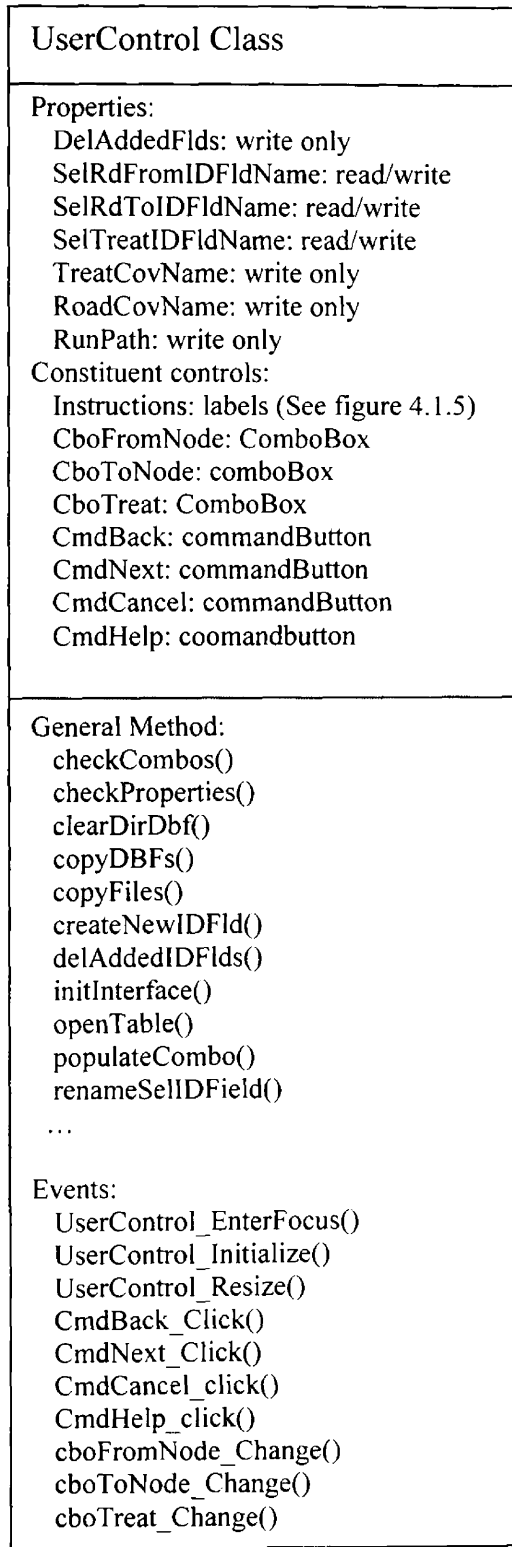
4.1.3 UML and Detail Design for ActiveX control in Import Wizard Form3

This ActiveX control project contains only the “UserControl” class “ctlSelectIDFields”.

1. UML

UML for this ActiveX control contains only the “UserControl” class “ctlSelectIDFields” (Figure 4.1.3.1).

Figure 4.1.3.1 UML for ActiveX control 3 in Import Wizard

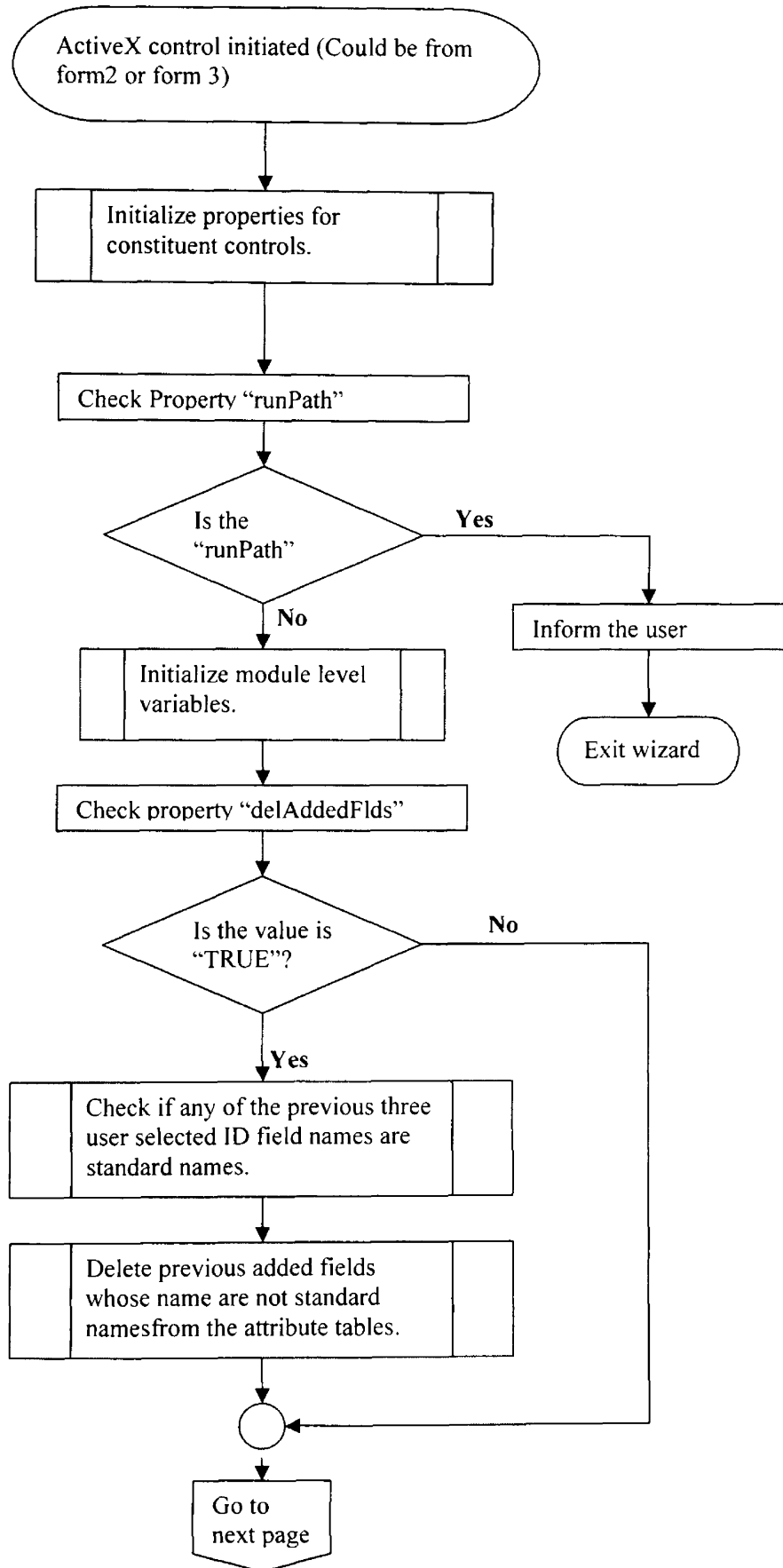


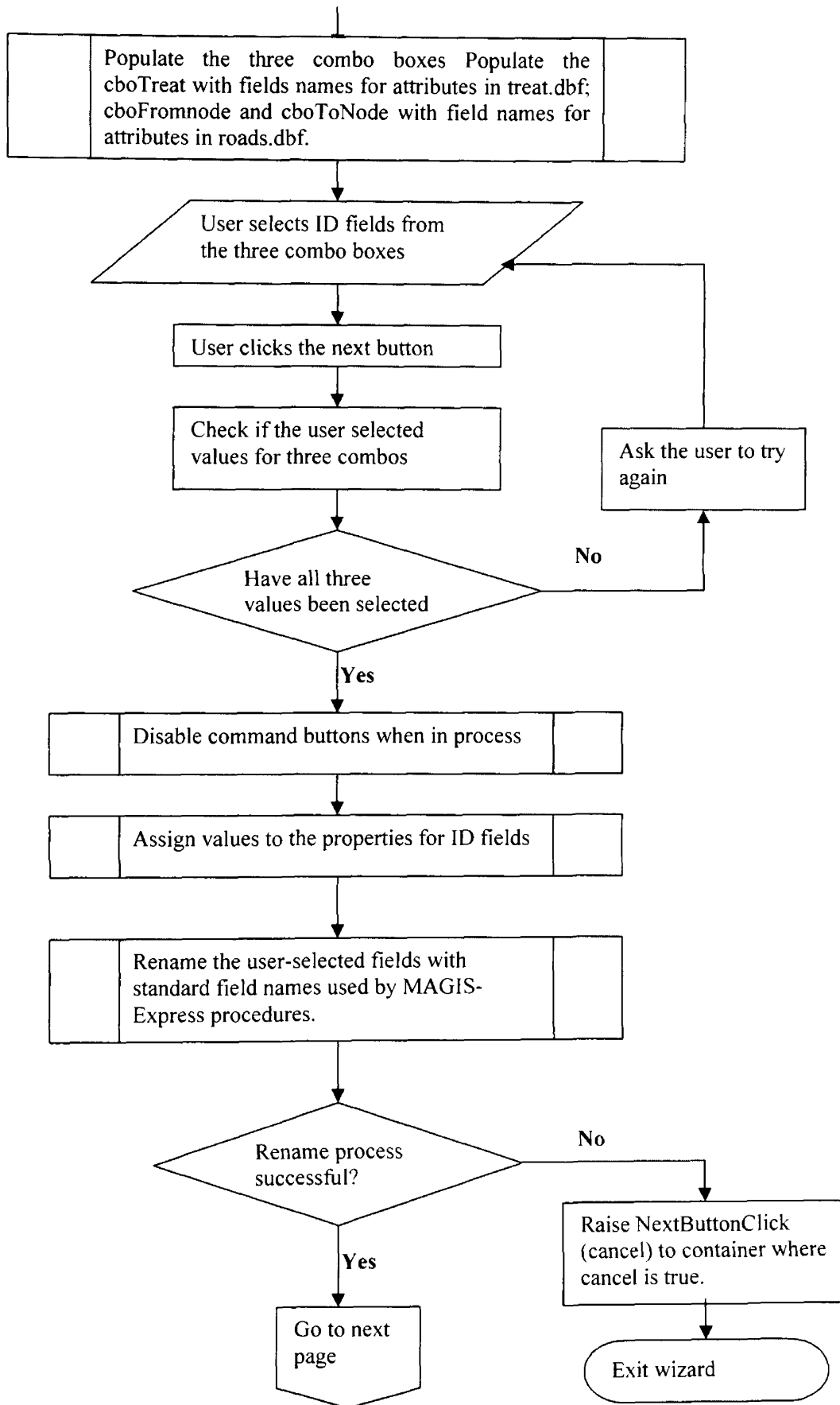
2. Flowchart

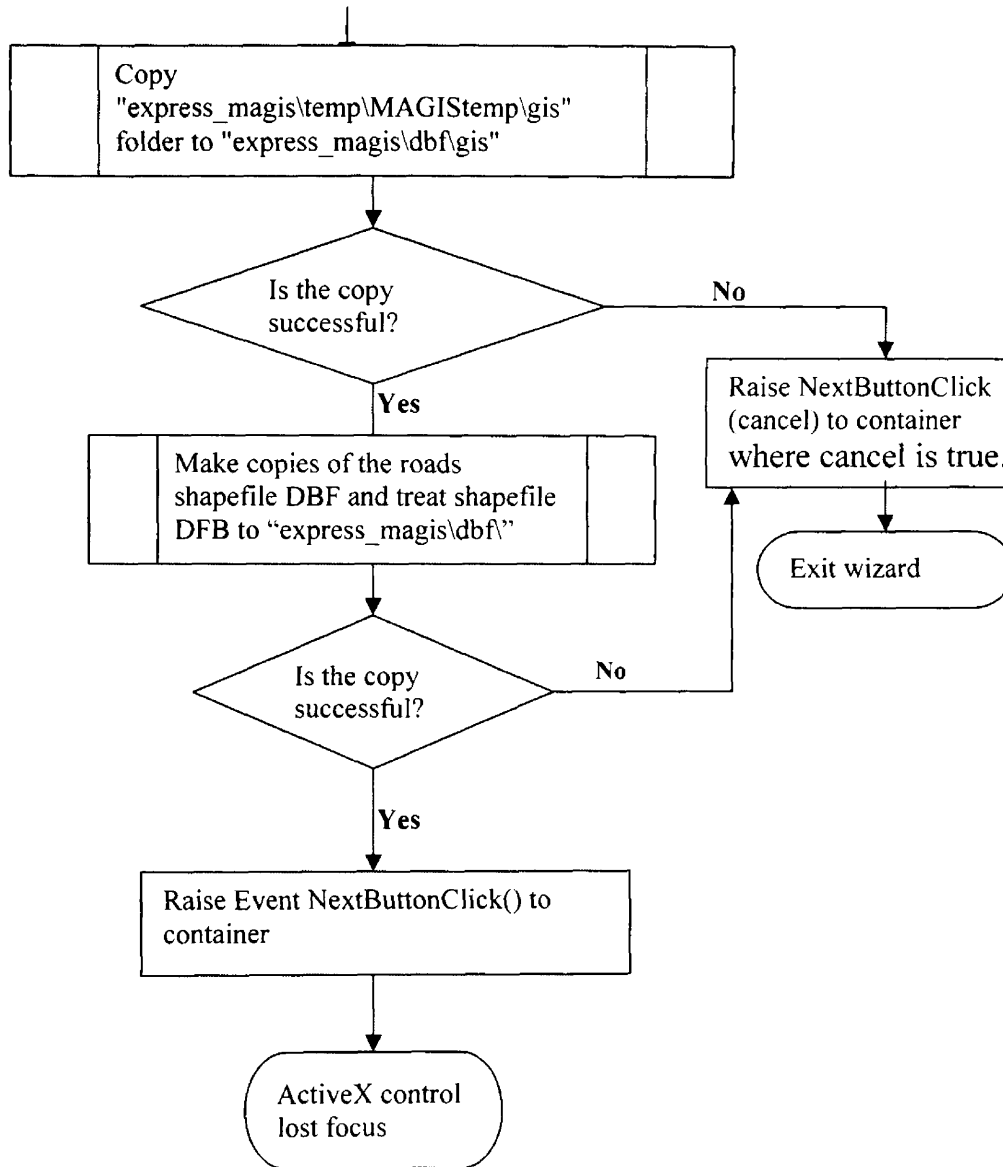
The flowchart displays the steps executed after this ActiveX control is instantiated (Figure 4.1.3.2).

After ActiveX control was initiated (Could be first time or back from *Form4*). The control will initialize the component controls in the interface according to a serial validation (such as required properties have valid value available). Then the control will populate the three combo boxes with corresponding table field names. After that the user can select ID values from the combo boxes. The user finishes selection by clicking the next button. Then the control will validate the user's selection and create new field in those tables with the selected fields' values. The new created fields will be assigned MAGIS Express standard names. The control then will raise "NextButton_Click" event to its container and from there the program flow will go to the next form in the same wizard.

Figure 4.1.3.2 Flowchart for ActiveX control 3 in Import Wizard







4. Detailed functions in this ActiveX control:

Properties of the UserControl object in this ActiveX control:

- ❑ **Property Let delAddedFlds()**
Indicates if it is necessary to delete the previous added fields from the attribute tables of the data sets.
- ❑ **Property Let roadCovName()**
Set the coverage name for “Roads” database.
- ❑ **Property Let treatCovName()**
Set the coverage name for “Treatment Unit” database.
- ❑ **Property Let runPath()**
Set the relative path where MAGIS-Express exists on the user’s machine.
- ❑ **Property Get / Let SelRdFromIDFldName()**
Returns / set the user selected fields ID for “Roads” "From_node".
- ❑ **Property Get / Let SelRdToIDFldName()**
Returns / set the user selected fields ID for “Roads” "To_node".
- ❑ **Property Get / Let SelTreatIDFldName()**
Returns / set the user selected fields ID for “Treatment Units” database.

General Functions:

- ❑ **Private Function checkCombos() As Boolean**
Parameter list: Null.
Output: Return a Boolean value indicating if the function if successful.

Description: Check if the three combo boxes are empty.

❑ **Private Function checkProperties() As Integer**

Parameter list: Null.

Output: Return an integer value indicating which attribute has been assigned.

Description: Check variables "m_strSelTreatID", "m_strSelRdFromID", and "m_strSelRdToID" to see if any of them have been assigned specific values of the fields in coverage and shapefile tables, return the number of ones whose values are not "CUT_UN_ID", "FROM_NODE" or "TO_NODE".

❑ **Private Sub clearDirDbf(dbfGisPath As String)**

Parameter list:

String dbfGisPath: The directory path of the dbf tables.

Output: Null.

Description: Clear the "..express_magis\dbf\gis" folder if it exists.

❑ **Private Function copyDBFs(sourcePath, desPath) As Boolean**

Parameter list:

String sourcePath: Directory path of the copied file.

String desPath: Directory path of the copy file.

Output: return a Boolean value indicating if the function is successful.

Description: Copy the "Roads" shapefile DBF and "Treatment Units" shapefile DBF tables to "..\express_magis\dbf".

- **Private Sub copyFCFld(pFeatClass As IFeatureClass, oldName As String, newName As String)**

Parameter list:

IFeatureClass pFeatClass: A Feature class object.

String oldName: Old name of the feature class object.

String newName: New name of the feature class object.

Output: Null.

Description: Copy all features of the user selected ID field to the new field of the feature class.

- **Private Function copyFiles(sourcePath As String, desPath As String) As Boolean**

Parameter list:

String sourcePath: Directory path of the copied file.

String desPath: Directory path of the copy file.

Output: Return a Boolean value indicating if the function is successful.

Description: Copy the "..\MAGIStemp\gis" folder to "..\express_magis\dbf\gis"

- **Private Function createNewIDFld(newFldName As String, oldFldLength As Long, oldFldType As esriFieldType) As IField**

Parameter list:

String newFldName: New field name.

Long oldFldLength: Old field character length.

esriFieldTYpe oldFldType: Data type of the old field.

- Output: Return an IField object.
Description: Returns a newly created field.
- **Private Sub delAddedIDFlds(idType As Integer)**
Parameter list:
 Integer idType: Type of the ID field.
Output: Null.
Description: Delete the added ID fields "CUT_UN_ID", "FROM_NODE", "TO_NODE" according to the value of idType.
- **Private Sub initInterface()**
Parameter list: Null.
Output: Null.
Description: Initialize the values of the controls on the interface. Delete the added ID fields from the attribute tables.
- **Private Sub populateCombo(pFC As IFeatureClass, cboType As Integer)**
Parameter list:
 IFeatureClass pFC: A feature class object.
 Integer cboType: Type of the combo box.
Output: Null.
Description: Populate the combo box named "cboTreat" with field names for attributes in table "treat.dbf". Populate combo boxes "cboFromnode" and "cboToNode" with field names for attributes in "roads.dbf" (Appendix A).

- ❑ **Private Function renameSelIDField(idType As Integer) As Boolean**

Parameter list:

Integer idType: Type of the ID field.

Output: Return a Boolean value indicating if the function is successful.

Description: Make decision to select suitable actions to rename different fields by calling functions renameTreatID() and renameRoadID().

General events of the constituent controls in this ActiveX control:

- ❑ **UserControl_EnterFocus()**

Triggered after ActiveX control in the container was activated. This event calls function initInterface() to initialize the values of the controls on the interface. Delete the added three fields from the coverage or shapefiles' attribute tables.

- ❑ **UserControl_Initialize()**

Initialize the attributes of the constituent controls on the interface.

- ❑ **UserControl_Resize()**

Triggered when the size of the ActiveX control is changed. This event repositions all the constituent controls in the user interface of the ActiveX control.

- ❑ **cboFromNode_Change()**

Triggered when the content of the “cboFromNode” combo box is changed by the user. This event helps prevent the user from editing the contents of the cboFromNode.

❑ **cboToNode_Change()**

Triggered when content of the “cboToNode” combo box is changed by the user. This event helps prevent the user from editing the contents of the combo box named “cboToNode”.

❑ **cboTreat_Change()**

Triggered when the user changes the content of the “cboTreat” combo box. This event helps prevent the user from editing the contents of the combo box named “cboTreat”.

❑ **cmdBack_Click()**

Triggered when the “Back” button is clicked. It leads to wizard *Form2*. It also raises the “BackButtonClick” event to its container.

❑ **cmdCancel_Click()**

Triggered when the “Cancel” button is clicked. It exits the Import Wizard. It also raises “CancelButtonClick” event to its container.

❑ **cmdHelp_Click()**

Triggered when the “Help” button is clicked. It also raises “HelpButtonClick” event to its container.

❑ **cmdNext_Click()**

Triggered when the “Next” button is clicked. The following tasks will be executed in order:

- a. Check if the user has selected values for all three Combo Boxes. If not prompt a message to ask the user to try again. Else, go to b.
- b. Disable command buttons and combo boxes when in process.
- c. Assign values to the properties for the new user selected id fields.
- d. Call checkProperties() method to check if any of the new user selected ID fields has a MAGIS-Express standard name.
- e. Call renameSelIDField () to rename the user-selected fields whose names are not the standard names with standard field names used by MAGIS-Express procedures. Such as “CUT_UN_ID” for “Treatment Units” Ids, “FROM_NODE” for “Roads” from nodes IDs, “TO_NODE “ for “Roads to nodes IDs. If the rename process succeeds, go to f. Else raise nextButtonClick() event with the “cancel” parameter set to “true” to the container form to exit the wizard.
- f. Call function “clearDirDbf()” to delete the “\gis\” sub directory from the dbf path.
- g. Call “copyFiles()” to Copy the“..\MAGIStemp\gis” folder to “..\express_magis\dbf\gis”. If successful, go to h. Else raise event “nextButtonClick()” with the “cancel” parameter set to “true” to exit the wizard.

- h. Call “copyDBFs()” function to make copies of the “Roads” shapefile DBF table and “Treatment Units” shapefile DBF table to “express_magis\dbf”. If successful, go to i. Else raise event nextButtonClick() to the container form with the “cancel” parameter set to “true” to exit the wizard.
- i. Raise event “NextButtonClick”.

Events to be raised by this ActiveX control to its container:

□ **NextButtonClick()**

Raised when the “Next” button is clicked. It will lead the wizard to *Form4*.

□ **BackButtonClick()**

Raised when the “Back” button is clicked. It will lead the wizard back to *Form2*.

□ **CancelButtonClick()**

Raised when the “Cancel” button is clicked. This will cause the wizard exits.

□ **HelpButtonClick()**

Raised when the “Help” button is clicked. This will allow the container form to display the help files.

CHAPTER 5

IMPLEMENTATION and TESTING

ActiveX control is one type of COM component based on the COM standard. The three ActiveX controls for MAGIS-Express Import Wizard were created in Microsoft Visual Basic 6 programming language. Some ESRI ArcObjects had also been used.

5.1 OLE (*ActiveX*)

OLE stands for “object linking and embedding”. OLE was suggested by Microsoft to implement its data-centric computing of its Windows operating system [1]. OLE is a collection of technologies for transferring and sharing information among applications; allowing them work with objects in a standard way.

The “Object Linking” means that a compound document can have a reference to the linked object, so any changes in the object can be shown in the document. For example, you may create a spreadsheet document using EXCEL and link it to a WORD document. “Object Embedding” means a compound document has only a copy of the object, and any changes in the object will not appear in the compound document until the object copy in the document is updated.

All the objects manipulated by OLE are special type of objects called Component Objects or Window objects that follow the same standard named Component Object Model (COM) which will be introduced next

Besides the COM technology, OLE also contains some other important technologies. It uses Universally Unique Identifier (UUID), Globally Unique Identifier (GUID), Class Identifier (CLSID), and Interface Identifier (IID) to recognize different kinds of items. It

also uses a standard method to direct an object container, like a MS Word application, on how to display different objects in it like an image, a table or a spreadsheet. OLE also helps to manipulate the marshaling of objects, a process to transfer objects among applications under 32-bits Windows operating systems. By using OLE Structured Storage, OLE helps the application to create its own file system to save the different kind of objects it contains together with their GUIDs. OLE Automation, a COM-based technology, enables one application to operate on the objects contained in other applications. It also supports late-binding, that is at run-time, to the objects.

ActiveX is another name for OLE. OLE (ActiveX) and COM architecture enable the component software to be reused. They also enable third parties' portable component software development that can be used by everyone who understands OLE and COM. This is why this project can create our own ActiveX components using Visual Basic 6 and use them in Visual FoxPro.

5.2 Architecture of the Component Object Model (COM)

The Component Object Model (COM) is a component software architecture that allows components developed by different software vendors to be built together into different kinds of applications. COM provides a standard for component interoperability. It is programming language-independent. It is portable to any platform that supports COM. It can be extended by anyone.

COM is the underlying architecture that forms the foundation for higher-level software technologies, like OLE. (Figure 5.2.1.)

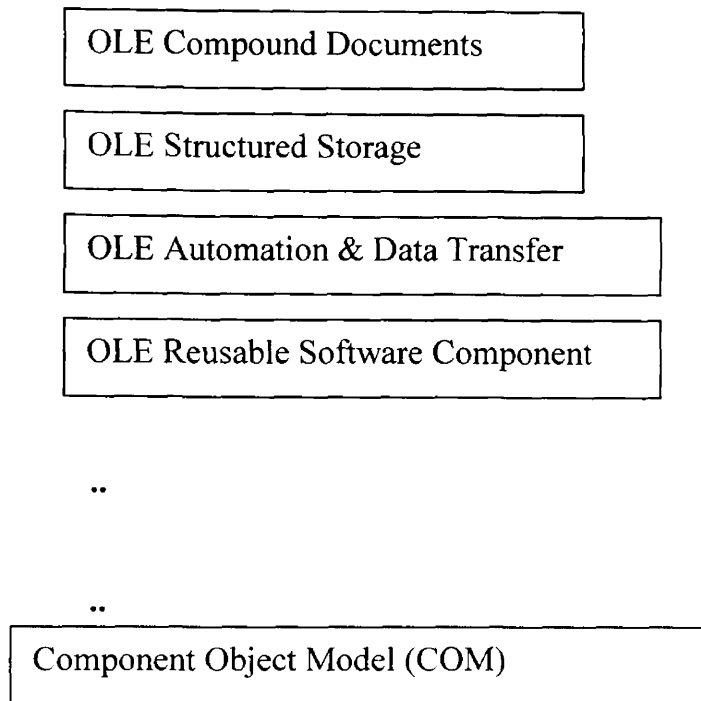


Figure 5.2.1 High-level OLE application services are built on the common Component Object Model foundation.

The fundamental standards provided by COM includes: a standard for function calls among the components (objects); a standard for interfaces that groups functions and properties; a standard for a basic interface that exists in every COM object acting in a standard way to help the object use other interfaces in other components. It also helps the COM object to track and control its own running existing references; a way to uniquely identify objects and interfaces and a mechanism to help manage the interactions of COM objects across processes or a network.

5.2.1 Managing Function Calls among the Components

The objects we mentioned here in the context of COM are different kinds of objects created using Object-oriented Programming (OOP) Objects languages like C++ or JAVA. These COM objects are pieces of codes that provide services to the rest of the systems. They are sometime called *components objects*.

For any given platform (hardware and operating system combination), COM defines a standard way to lay out virtual function tables (vtables) in memory, and a standard way to call functions through the vtables. Thus, any language that can call functions via pointers (C, C++, Small Talk®, Ada, and even Basic) all can be used to write components that can interoperate with other components written to the same binary standard [1]. The function calls are finished in the following way: The client holds a pointer to a vtable in a component object, that vtable contains some pointers pointing to the actual functions.

5.2.2 COM Interfaces

Attributes

Another characteristic of a COM component is that it can contain one or more sets of functions, each of those sets is called an interface. Every component object can ONLY access other components by using the pointers to their interfaces. So we could say that all OLE services are simply interfaces. This is an implementation of object encapsulation that is one of the fundamentals of the component software standard. The interfaces we mention here in the context of COM are different from those OOP related interfaces. A COM interface is actually a *contract* between software components to provide a small but useful set of syntax-related operations (methods). An interface is the definition of an

expected behavior and expected responsibilities, different objects can implement it in different ways but must all conform to the interface definition. This is another fundamental of the component software standard - polymorphism. An interface defines a standard through which the clients and the component objects communicate. Its interface identifier, a globally unique ID (GUID), identifies an interface. The GUID used by COM is 128 bits long, definitely unique in the world across space and time. When an interface is created, it is assigned a GUID. Anyone who wants to use the interface must use the identifier to get a pointer to the interface.

(Figure 5.2.2, 5.2.3, 5.2.4.)

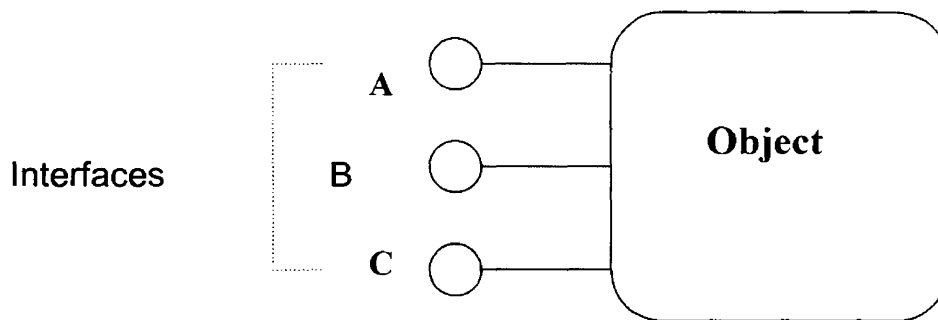


Figure 5.2.2 A typical diagram of a component object that supports three interfaces A, B, and C.

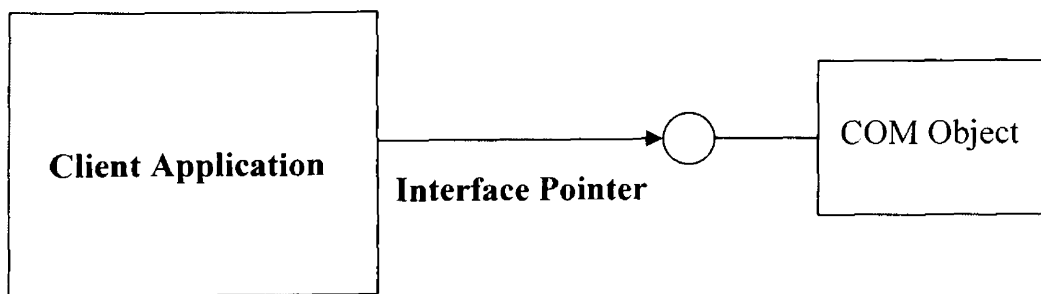


Figure 5.2.3 A client application uses an interface pointer to access an Interface of the COM object[4].

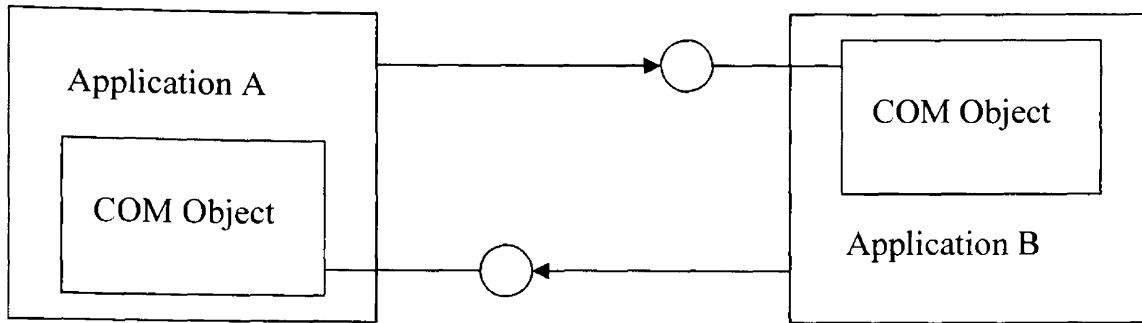


Figure 5.2.4 Two applications (A and B) may connect to each other's objects, in which case they extend their interfaces toward each other [4].

COM and the Client / Server Model

In COM, the interactions between a component object and its users are established based on a Client / Server paradigm. The component providing its services to its users is the server and the users of those services are the clients of the component. This client/server mechanism is secure because COM permits the server component and its clients to run in different process spaces. So if one component in a component-based application fails, it does not destroy other parts of the application. A COM component could be both a server and a client. For example Component A has interface pointers pointing to interfaces in another component B, so A is a client of B and B is the server. Component B at the same time has interface pointers pointing to interfaces in A, so A is also the server for component B and B is a client of A, as shown in Figure 5.2.4.

Component Object Library

The Component Object Library is component implemented by the Windows operating system that provides the mechanics of COM. The Component Object Library manipulates the COM mechanism behind all the component objects. It makes it possible for calls to the “IUnknow” interfaces. It helps launch components and establish connections between components.

The Component Object Library also enables the COM client/server model mentioned above. For instance when an application creates a component object, it passes the CLSID (class identifier) of that component object class to the Component Object Library. The Component Object Library uses that CLSID to look up the associated server code in the registration database. If the server is an executable, COM launches the .EXE file and waits for it to register its class factory through a call to **CoRegisterClassFactory** (a class factory is the mechanism in COM used to instantiate new component objects). If that code happens to be a dynamic-link library (DLL), COM loads the DLL and calls **DllGetClassFactory**. COM uses the object's **IClassFactory** to ask the class factory to create an instance of the component object and returns a pointer to the requested interface back to the calling application. The calling application doesn't need to know where the server application is run; it simply uses the returned interface pointer to communicate with the newly created component object [1].

To summarize, a COM component object should have a GUID as its identifier; it should support at least one interface (could be more than one); it can have data associated with it, but the only way to access these data is through the use of its interfaces.

5.3 ActiveX CONTROLS (OLE CONTROLS)

ActiveX components are COM Objects built based on the COM mechanism that enables building robust and reusable components. COM also allows components created by different parties to work together to form a new application. This is a key difference from the objects in the context of OOP. ActiveX Components can run either in-process or out-of-process with respect to the clients that use their objects. ActiveX controls are one of the several types of COM component objects. These different types of components are explained in section 5.3.1.

5.3.1 Types of ActiveX Components

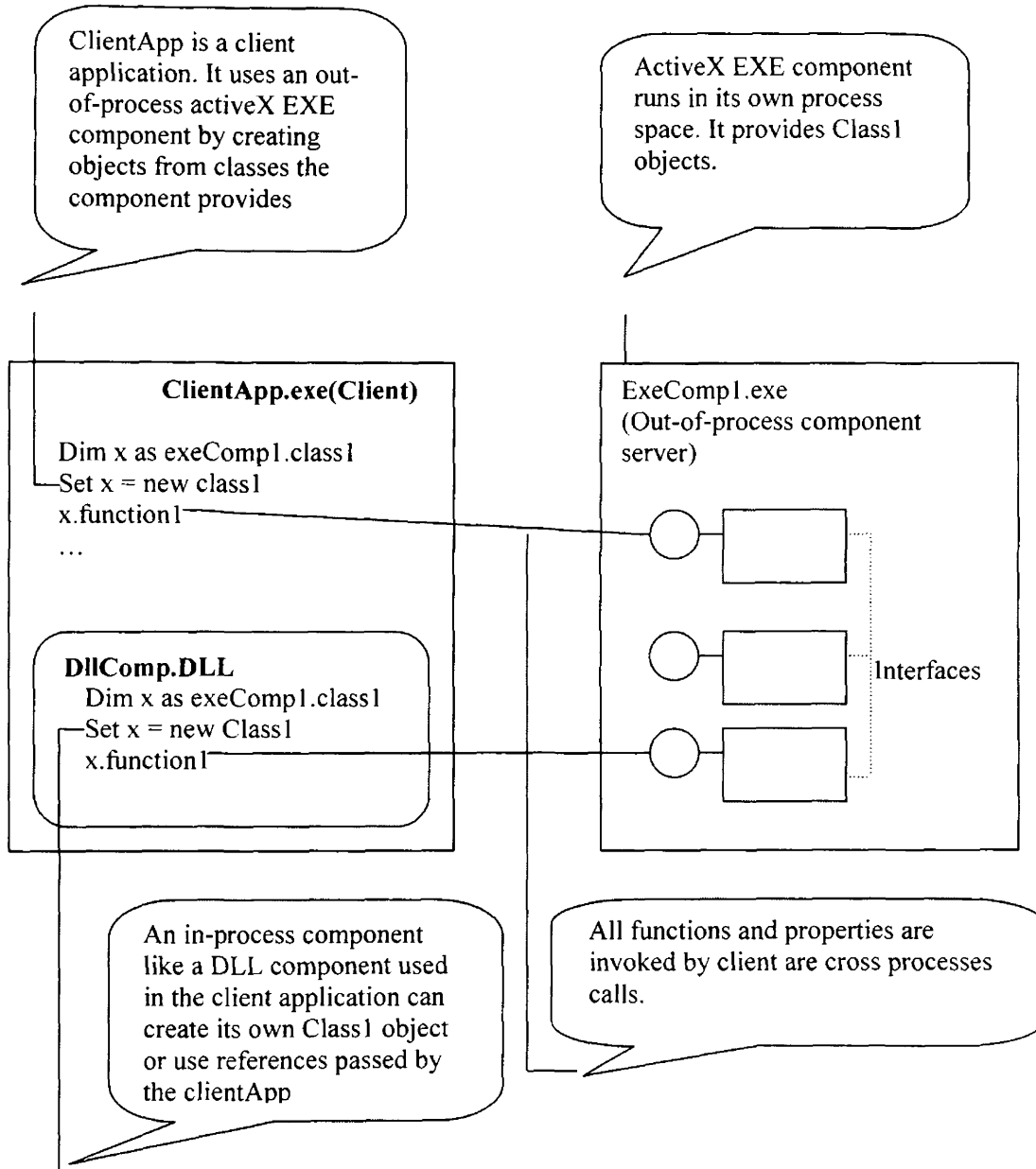
Several types of ActiveX control are files having the following extensions: .exe, .dll, .ocx and others. These types of files are just some of compiled files following the OLE (ActiveX) specifications. They are actually the same type of components.

- ActiveX EXE Component (*.exe file)

An Active EXE Component is an out-of-process component because it runs in its own process space. The client application of the EXE component runs in another process space (Figure 5.3.1).

An ActiveX EXE component is usually needed under a circumstance when services should be provided to multiple applications by an ActiveX component running on a remote computer. ActiveX EXE controls typically have no visible user interfaces.

Figure 5.3.1 Client and out-of-process component [4]

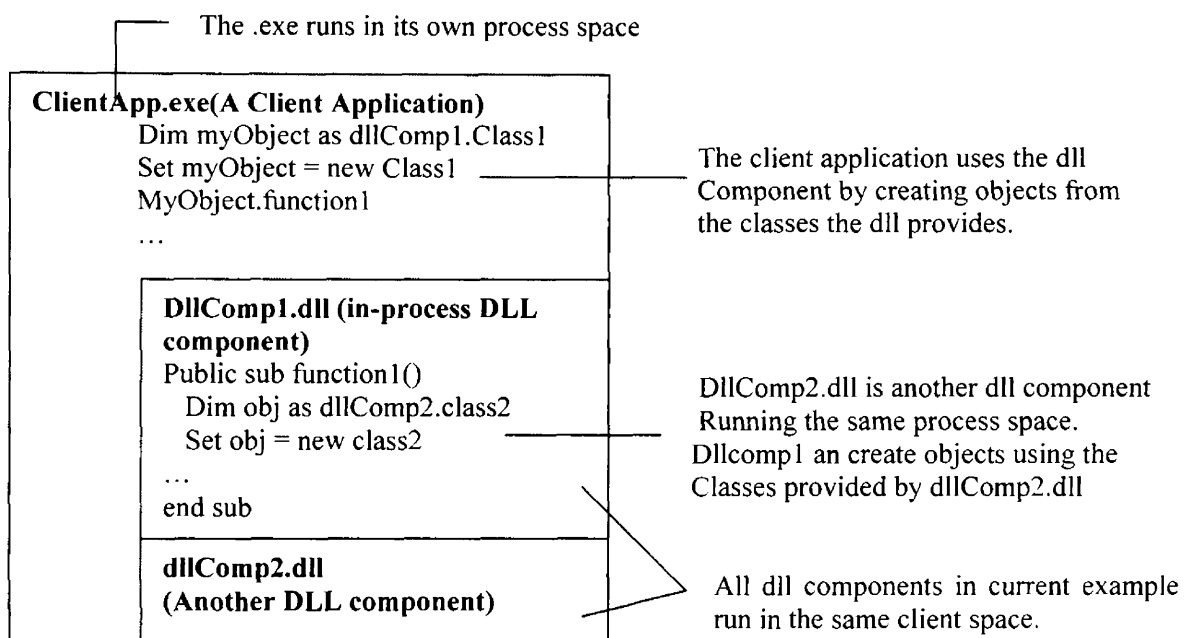


- ActiveX DLL component (*.dll file)

DLL stands for Dynamic Linked Library. An ActiveX DLL component is like libraries of objects. As with an ActiveX EXE component, a client application uses a code component by creating an object from one of the classes the

component provides, and invoking the object's properties, methods, and events. Unlike the EXE component, a DLL component runs in the same process space as the client application, so it is called an in-process component (Figure 5.3.2).

Figure 5.3.2 In-process components are used by applications or other in-process components [4]



- ActiveX Control (*.ocx file)

ActiveX controls, formerly called OLE controls, are standard user interface elements that allow you to assemble forms and dialog boxes rapidly. ActiveX controls are also in-process controls. They can be visible at the design time of a window application. For the MAGIS-Express Import Wizards, where we need a visible component that can be put into Visual FoxPro container forms at the design time of these forms, we need to build ActiveX controls.

5.4 Creating ActiveX Controls In Visual Basic

5.4.1 Why Use Visual Basic 6

Visual Basic and Object-Oriented Programming

Microsoft Visual Basic 6 provides a way to handle objects. You can create your own class in Visual Basic. It supports *encapsulation*, the first characteristic of the Object-Oriented Programming (OOP). It allows a single command name to be shared by multiple objects, the second characteristic of OOP called *polymorphism*. Although Visual Basic doesn't implement the third OOP characteristic named *inheritance*, and therefore is not a real OOP language, but it is an object-based programming language. Visual Basic does allow the programmers to accomplish many tasks that inheritance supports using technologies like Aggregation and Polymorphism.

Visual Basic and ActiveX controls creation

Although programming in Visual Basic is not as easy as it once was – many features have been added- Visual Basic is still one of the easiest ways to create Windows applications. For creating ActiveX controls, Visual Basic is also one of the safest ways. It provides an easy and reliable programming environment for creating ActiveX controls. Visual Basic handles many ActiveX (OLE) technologies for the programmer. It encapsulates many of the ActiveX objects and control-related interfaces into the Visual Basic development environment to make ActiveX control creation easy and quick so the programmer need not worry about many of the complexities of the OLE or COM. Visual

Basic also allows the programmers to extend the power of it by using Win32 API calls and third-party ActiveX controls and DLLs.

5.4.2 Visual Basic ActiveX Control Creations Basics

Visual Basic provides a graphical designer similar to a form designer for the author to create ActiveX controls.

1. Objects involved in the creation of the ActiveX controls in this project

Objects involved in creating an ActiveX control include:

- The UserControl Object

This is a sub object of the created ActiveX control object. Like other internal controls the UserControl object method and properties can be accessed by the created control. It also raises events to the created control. It provides graphical interface for the created ActiveX control where other constituent controls could be put into this interface to compose the visible interface of the whole ActiveX control. The “UserControl” object is the default object in an ActiveX control in a similar way that a form is the default object in a Visual Basic standard EXE project.

The “UserControl” object is a COM object. It contains interfaces that are sets of properties, methods and events to make an ActiveX control work and allow the container to communicate with it.

The events and properties of the “UserControl”s for the MAGIS-Express Import Wizard were implemented according to the detailed design outlined in the previous chapters.

□ Constituent Control Objects

Different kinds of Visual Basic internal controls can be put into the ActiveX control designer to compose the ActiveX control. Each of the three ActiveX controls of the Import Wizard contains such constituent controls, such as command buttons, radio buttons, text boxes, labels, frames and line. Other ActiveX controls can also be put into the created ActiveX control. For example: In ActiveX control for wizard *Form1*, we used MS Common Dialog; in ActiveX control for wizard *Form2*, we used ESRI Map control, Image List control, tool bar control...

The constituent controls used in the ActiveX controls for the MAGIS-Express Import Wizard follow the graphical interface design described in chapter 4 (figure 4.1.1 – 4.1.5.).

2. Methods create ActiveX controls for the MAGIS-Express Import Wizard

Generally speaking, there are three ways to create an ActiveX control:

□ Enhancing an exiting control:

Creating an ActiveX control by adding new properties or methods to an existing control, such as a Command button.

□ User-Draw Control:

Creating a control from scratch.

□ Assembling a control from constituent controls:

Creating an ActiveX control using one or more existing controls. In Visual Basic the constituent controls that can be used by an ActiveX control include Visual Basic internal controls and other components [4].

All of the three ActiveX controls for the Import Wizard were built this way. The appearances of the graphical user interfaces of these controls are consistent of visible constituent controls.

4. General steps to code the ActiveX controls for the MAGIS-Express Import Wizard:

- a. Create a project group in Visual Basic consisting of the control project and a test project.
- b. Implement the appearance of the control by adding controls and/or code to the UserControl object according to the graphical interface design described in previous chapter.
- c. Implement the interface and features of your control (properties and methods).
- d. Declare and raise events as required so that the developer who's going to use those controls can have a way to response to those events as needed.
- e. Test new added features in the testing project one by one as they are added.
- f. Compile the new control component (.ocx file) and test it with all potential target applications.

5.5 ArcObjects Used in MAGIS-Express Import Wizard

ArcObjects is the development platform for ArcGIS Desktop that containing a suite of applications, such as ArcMap, ArcCatalog, ArcToolbox etc. ArcGIS is an integrated geographic information system. ArcGIS Desktop is one of its three key part, providing functions including mapping, data management, geographic analysis, data editing, and geoprocessing to meet the needs of a wide range of GIS users.

ArcObjects is built using Microsoft Component Object Model (COM) standard. Therefore, it is possible to extend ArcObjects by writing COM components using any COM-compliant development language, such as Visual C++ or Visual Basic.

The ArcObjects library that contains over 1000 classes and 2000 interfaces, is a comprehensive set of COM components designed to provide developers with the ability to extend and customize existing ArcGIS applications such as ArcMap and ArcCatalog or build their own applications [6].

ArcObjects helped to complete many tasks when building the Import Wizard ActiveX controls:

1. In the ActiveX control for *Form1*:

- Objects such as WorkspaceFactory, FeatureWorspace, Dataset, etc. are used to help build the topologies for coverages, create new ArcInfo workspaces, get the database projections, etc.
- Objects: WorkspaceName, QueryFilter, Row, etc. helps to manage tabular data in database attribute tables.

2. In the ActiveX control for *Form2*:

Besides the ArcObjects mentioned in item 1, this ActiveX control also uses map-related ArcObjects, such as GeoFeatureLayer, SimpleRenderer and RgbColor to implement tasks for manipulating map content including “loading map layers”, “clearing map layers”, “getting map units”, “managing map selections”, and so on.

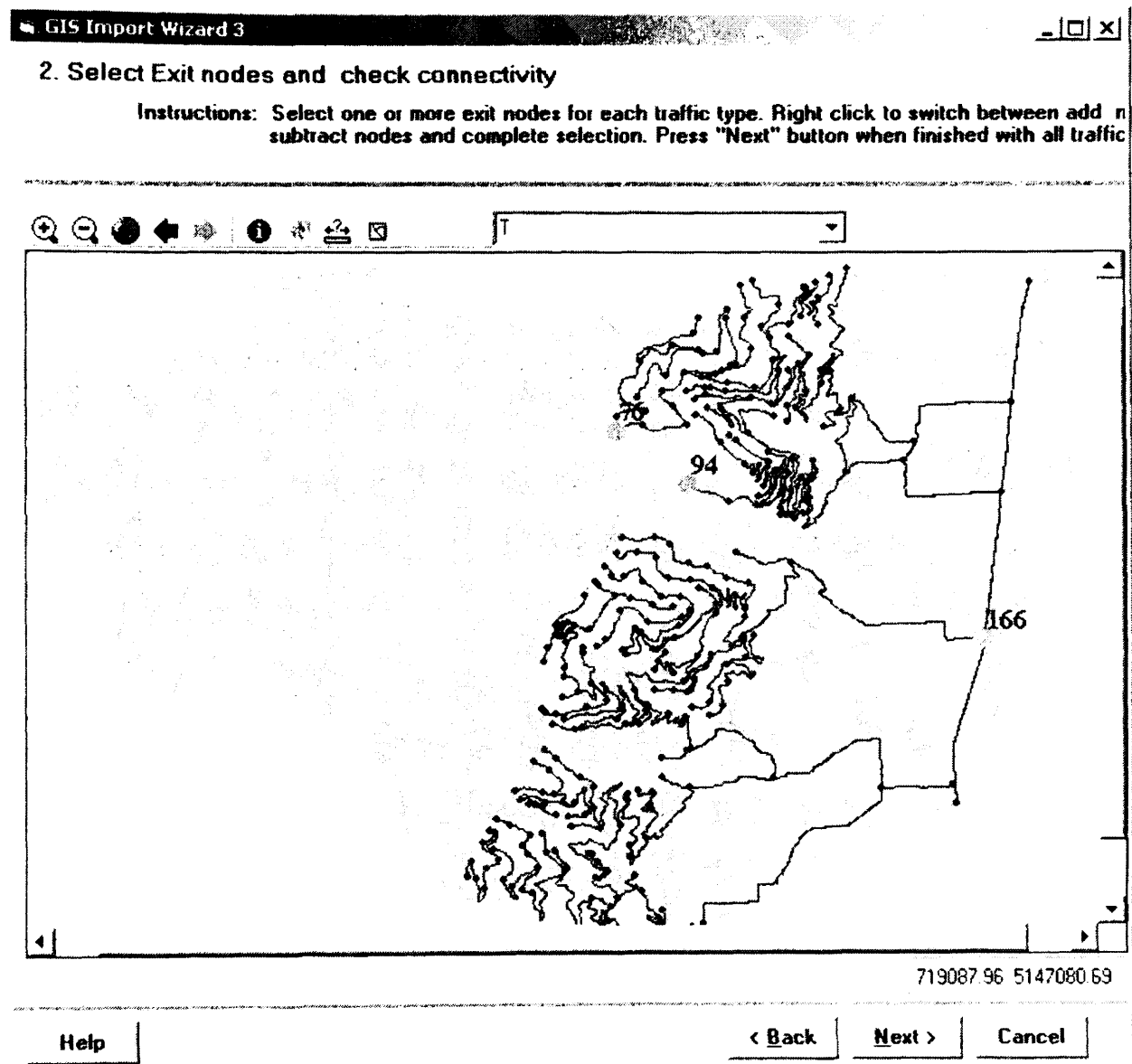
The “customized selection tool” for this control implements both ICommand and ITool interfaces to build the tool.

Figure 5.5.1 on next page shows the situation after the user selected several traffic exit nodes on the map in running *Form2*.

3. In the ActiveX control for **Form3**:

In the implementation of this ActiveX control, ArcObjects related to handle coverages, shapefiles, and tabular data have also been used, such as WorkspaceFactory, Workspace, Dataset, Row, QueryFilter and so on.

Figure 5.5.1 Using customized selection tool to select "exitnodes" on mapControl in Import Wizard Form2



5.6 Testing and Debugging

5.6.1 Testing Strategies

1. Defect Tracking:

The purpose of defect tracking is to record and track defects from the time they are detected until the time they are fixed [2].

A defect report table (VFP table) was used for this purpose. The structure of the defect table is as following:

| Field name | Date Type | Width | Description |
|-------------|------------------|-------|--|
| ID | Character | 4 | Defect ID |
| Date_found | Date | 8 | Date the defect is found |
| Finder | Character | 15 | Person who found the defect |
| Defect_loc | Memo | 4 | Defect location (in which class or procedure) |
| Priority | Character | 1 | H(igh), M(edium) or L(ow) |
| How found | Memo | 4 | Actions used to find the defect |
| Description | Memo | 4 | Description of the defect |
| Date_fixed | Date | 8 | Date the defect is fixed |
| Fixer | Character | 10 | Person who fixed the defect |
| Fix_method | Memo | 4 | Actions used to fix the defection |
| Doc_Type | Memo | 4 | The Document type the bug occurs |
| Doc_version | Character | 10 | Current version of the document contains the bug (e.g. 08-23-02) |
| Data_file | Character | 30 | Testing data set used to produce this bug |
| Others | Memo (binary) | 4 | Other description as needed |

2. Unit Testing

Unit testing strategies employed for this Import Wizard includes:

- **Block Testing:** Set up different conditions for each tested code block (such as loop or “if” decision making block) run the block to see if the result is correct.

- **Subroutine Testing:** Provide the parameter list for the tested subroutine, run the tested subroutine to see if the subroutine can produce correct results.
- **Error Trapping:** The Visual Basic Standard Error-Trapping device was used in every method (not including events) to provide more information on an error. Steps include:

- a. Make the first line of the procedure:

On error goto errorhandler

The statement “*On error goto errorhandler*” activates the error - trapping. If an error occurs the program will jump to the Error-handling routine.

- b. Type in regular statements for the procedure.

- c. End the procedure with:

Exit sub

Errorhandler:

Error-handling routine

Resume

The statement “Resume” causes the program to jump back to the line causing the error. The “Exit sub” causes the program to exit as soon as an error occurs. The “Errorhandler:” is a line label that can be replaced by any word less than 40 characters in length. It must be in the same procedure with the “On error goto Errorhandler” statement.

3. Integrating Testing & System Testing

Integrating Testing and System Testing were executed by other QA members in MAGIS working team. Tasks include integrating the final ActiveX controls into their respective VFP forms one by one and testing the whole wizard in the VFP environment.

5.6.2 Debugging methods:

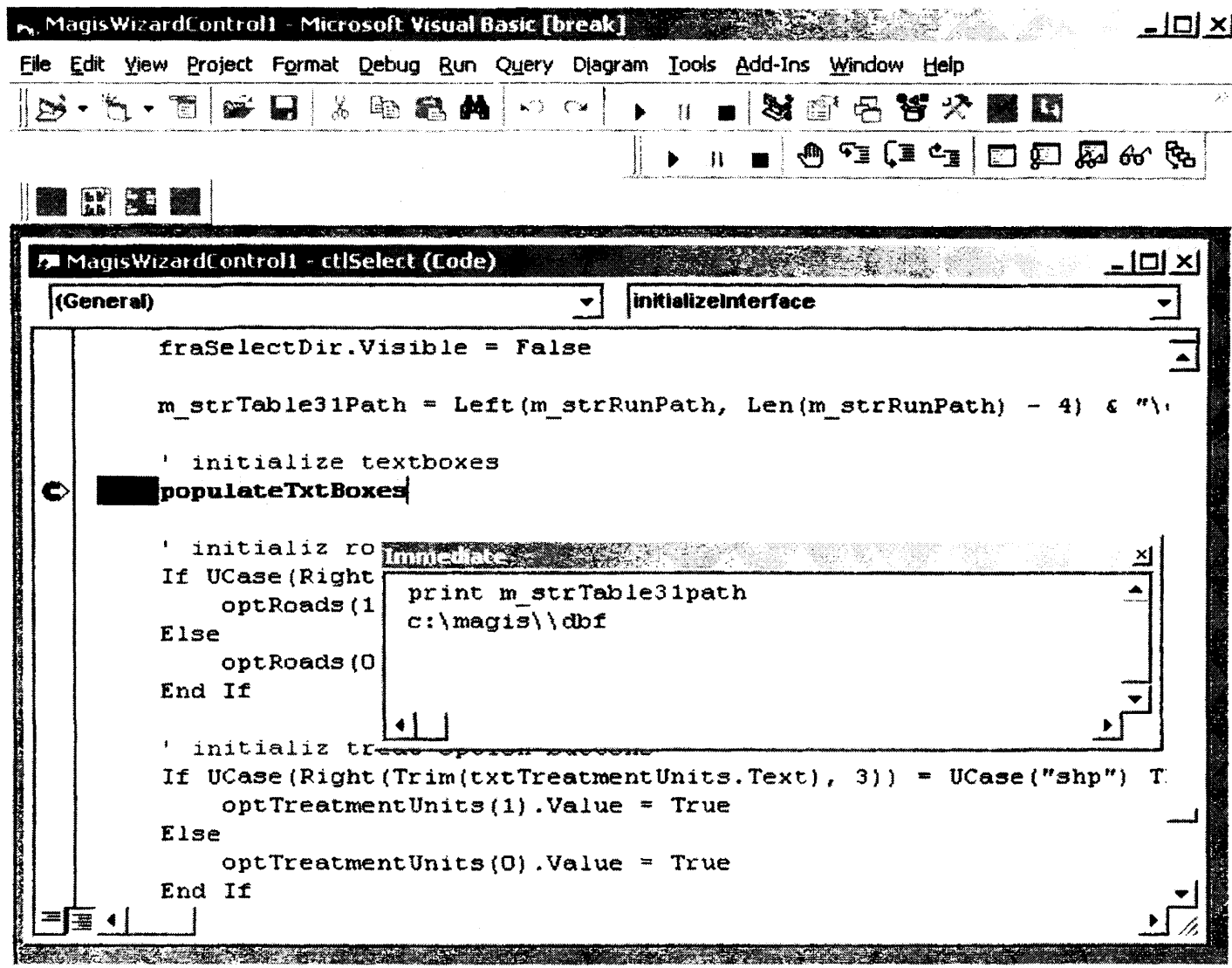
1. Print methods:

Placing print statements at strategic points in the program and displaying the values of the selected variables or expressions until the error is detected.

2. Using the Visual Basic debugger:

The Visual Basic debugger provides a way for the programmer to pause the program during the execution in order to view and alter the values of the variables that could be accessed through the three “Debug windows” on the tool bar. Figure 5.6.1 shows the situation when the program’ execution paused at a break point and the immediate window is used to print the value of a variable.

Figure 5.6.1 Using Visual Basic debugger



CHAPTER 6

DISCUSSION

6.1 Why use ActiveX controls in the MAGIS-Express Import Wizard

MAGIS-Express was created by reconstructing many existing graphical user interfaces written in Visual FoxPro (VFP) programming language. VFP is needed to manage DBF formatted spatial data associated with geographic locations. The Import Wizard is a new set of graphical user interfaces that help to facilitate the read data import procedure. It is a part of a suite of MAGIS-Express GUIs that are in VFP format. Microsoft VFP is a database management system that can also be used as a programming language to create some simple GUIs to handle data. It supports COM technology, which means that it allows ActiveX components as part of its project to improve its functionality.

ActiveX controls are ideal for creating reusable objects that have graphical user interfaces. Unlike other types of ActiveX components, such as DLL or EXE server, ActiveX controls are intended to support user interactions by providing graphical user interface. They run in-process, which means in the same process space with the calling application, the OLE need not to create a proxy object for it to establish server/client the communication path (This is an overhead for the system), therefore they have good performance. They are activated as soon as the user selects them. In addition to properties and methods, ActiveX controls can raise events to their container (such as a VFP form). The application programmer who uses the ActiveX control can add his own statements in the event procedures raised by the ActiveX controls to handle the events in their application. ActiveX controls are compatible with many containers, including Microsoft Office applications, VFP applications and Internet browsers.

6.2 Visual Basic vs. Visual C++ in ActiveX Controls Creation for MAGIS-Express Import Wizard.

Both Visual Basic and Visual C++ can be used to create ActiveX controls. Visual Basic was chosen for this project because:

- Compared to VC++, Visual Basic makes the process of creating an ActiveX control much easier and faster.

The ActiveX controls used in the MAGIS-Express Import Wizard need to contain different kinds of constituent controls, such as combo boxes, map controls, command buttons, etc. It is very difficult to create an ActiveX control using constituent controls using VC++. Using Visual Basic instead, this process can be completed very easily. Since Visual Basic encapsulated many of the ActiveX control-related interfaces and objects into the language itself and the programming environment.

- Visual Basic provides one of the safest ways to create the ActiveX controls.

Programmers who use VC++ need to deal with frequent memory exceptions during the creation of the applications. Visual Basic itself handles these situations for the programmer.

6.3 Software Reuse

ActiveX control is an example of software reuse. Once the controls are compiled into OCX files, they can be used in different kind of containers in different applications that want a user interface.

ActiveX controls in the Import Wizard have many unusual features specific to MAGIS-Express. The complexity of these controls makes them unique to the Import Wizard. But each control has a lot of functions that can be used by other applications. So these functions are put into a utility package as samples according to their purposes. For example those functions related to managing GIS datum (coverages or shapfiles) are put

in a Visual Basic standard model named “CoveragesAndShapefiles”, with a graphic user interface as a sample use for it. Functions related to handle DBF tables are put into a model called “ItableUtilities”, also with the sample that shows the way to use them.

These samples help other programmers to create similar ArcObjects related applications in Visual Basic. Therefore this is another form of software reuse.

Appendix A

A Quick Reference To Tables Involved In This Project

Table _31gis.dbf

Description:

Store model information, such as “Road” and “Treatment Units” database paths and last modify time.

Structure:

(Only those fields with a * will be used in the development of MAGIS-Express Import Wizard.)

| Field Name | Data Type | Description |
|-------------|-----------|------------------------------------|
| *roadsdata | Character | “Roads” database path |
| *treatdata | Character | “Treatment Units” database path. |
| standdata | Character | Not used in MAGIS-Express version. |
| *treat_time | Character | The last modify time |

Table _6.dbf

Description:

Store traffic types for road network in planning area.

Structure:

(Only those fields with a * will be used in the development of MAGIS-Express Import Wizard.)

| Field Name | Data Type | Description |
|-------------|-----------|---|
| *Int_t_type | Numeric | Integer format of the traffic type |
| *Traf_type | Character | Traffic types, such as |
| Out_type | Character | Output type, such timber or non-timber. |
| Conv_fact | Numeric | Capacity measure unit conversion factor. |
| cost | Numeric | Cost per unit for this traffic type |
| Ini_id1 | Numeric | Internal code for Nontimber output or species group#. |
| Ini_id2 | Numeric | Internal code for Nontimber output or species group#. |
| Ini_id3 | Numeric | Internal code for Nontimber output or species group#. |
| Ini_id4 | Numeric | Internal code for Nontimber output or species group#. |
| Ini_id5 | Numeric | Internal code for Nontimber output or species group#. |
| Int_act_id | Numeric | Internal activity id. |
| Int_mnt_cd | Numeric | Internal code for attribute class |
| Int_flo_cd | Numeric | Internal code for output |
| descrip | Memo | Description |

Table_6a.dbf**Description:**

Store the user selected "Exit nodes" (demand nodes in the road network where forest products such as timber are loaded out of the planning area through the road network)

Structure:

(Only those fields with a * will be used in the development of MAGIS-Express Import Wizard.)

| Field Name | Data Type | Description |
|-------------|-----------|--------------------------------------|
| *Int_t_type | Numeric | Integer format of the traffic type. |
| *demand_nod | Character | The user selected "Exitnode" number. |

Table roads.dbf**Description:**

This is the attribute table for "roads" line type shapefile. It is associated with the "roads" geographic features in the planning area. Each row in the table represents a geographic feature (a road link). Each column represents one attribute of a feature, such as horizon length, surface type, number of lanes, and so on.

Structure (Sample):

(Only those fields with a * will be used in the development of MAGIS-Express Import Wizard.)

| Field Name | Data Type | Description |
|-------------|-----------|---------------------------------|
| fnode | Numeric | Shapefile internal id field |
| tnode | Numeric | Shapefile internal id field |
| lpoly | Numeric | Shapefile internal id field |
| rpoly | Numeric | Shapefile internal id field |
| *Length | Float | Length of this segment |
| Roads | Numeric | Shapefile internal id field |
| Roads_id | Numeric | Shapefile internal id field |
| Fnode1 | Float | Shapefile internal id field |
| Tnode1 | Float | Shapefile internal id field |
| *Cur_status | Character | Current status of the road link |

| | | |
|------------|-----------|--|
| *Alignmt | Character | Road alignment |
| *Surf_ty | Character | Surface type of the road link |
| *Num_lanes | Numeric | Number of lanes on this road |
| *Horz_len | Numeric | Horizon length of the road link |
| *Terr_n | Character | |
| *speed | Float | Speed allowed on this road segment |
| *Rd_option | Character | Road option for this road link |
| *Average | Character | |
| *segment | Character | Road segment id |
| *suitable | Character | Indicating of this road segment is suitable. |
| *Fr_to | Character | ID field |
| *From_node | Numeric | The from node number of this road segment |
| *To_node | Numeric | The to node number of this road segment |

Table treat.dbf

Description:

This is the attribute table for “Treatment Units” polygon type shapefile. It is associated with the “Treatment Unit” geographic features in the planning area. Each row in the table represents a geographic feature (a treatment unit area). Each column represents one attribute of a feature, such as area, acres, dominant tree species, and so on.

Structure (Sample):

(Only those fields with a * will be used in the development of MAGIS-Express Import Wizard.)

| Field Name | Data Type | Description |
|------------|-----------|------------------------------------|
| *area | Float | Area of this unit area |
| *perimeter | Float | Perimeter of this unit area |
| Treat | Numeric | Shapefile internal id field |
| Treat_id | Numeric | Shapefile internal id field |
| *Acres | Float | Number of acres in this area unit |
| *mgtarea | Character | Management area |
| *Dom_sp | Character | Dominant tree species in this unit |

| | | |
|-------------|-----------|--|
| *Density | Character | Density of the trees in this unit |
| *Average | Character | |
| *hbty_grp | Character | Habit type group in this unit |
| *vol | Float | Volumes of log in this unit |
| *Sz_class | Character | Size class of the trees in this unit |
| *Time_inc | Numeric | Increasing time unit used in management of this unit |
| *Pas_act_id | Character | Past activity ID |
| *slope | Float | How slope it is in this area unit |
| *Yr_sin_act | Float | Number of years past since last treatment activity |
| *Log_meth | Character | Logging method used |
| *risk | Character | Risk class |
| polyid | Numeric | Shapefile internal id field |
| *Cut un id | Numeric | ID field |

References:

1. Appleman, Dan, 1999. *Developing COM/ActiveX Components with Visual Basic 6*. Indianapolis, Indiana: SAMS.
2. McCarthy, Jim, 1995, *Dynamics of Software Development*, Pg.10-11. Redmond, Washington: Microsoft Press.
3. Troutwine, Judy, 2002. "*Specifications and requirements document for MAGIS-Express Import Wizard*". Missoula, Montana: School of Forestry, University of Montana. Typewritten.
4. MSDN Library 2001, *Control Creation Basics*. Redmond, Washington: Microsoft Press.
5. Williams, Sara and Kindel, Charlie, 1994. *The Component Object Model: A Technical Overview*. MSDN Library. Seattle, WA: Microsoft Corp.
6. Zeiler, Michael, editor, 2001. *Exploring ArcObjects*. Redlands, California: ESRI.
7. Zurring, Hans, 1998, "*Overview of MAGIS: A Multi- Resource Analysis and Geographic Information System*". Missoula, Montana: School of Forestry, University of Montana.