CHAPTER 9: INTERFACES

Objectives

The objectives are:

- Explain how to use an automation server.
- Use OCX.
- Work with XMLPorts.
- Handle files.
- Send e-mail programmatically.

Introduction

This chapter deals with extending Microsoft Dynamics™ NAV using Component Object Model (COM) and file handling. First the essentials of COM are discussed, both from the Automation Server and OCX perspectives. Then automation is used to add e-mail confirmation functionality to the seminar solution. Finally, file handling is used to generate an XML participant list. These are examples of how Microsoft Dynamics NAV solutions can be extended to provide enhanced functionality.

Using an Automation Server

In C/SIDE, developers can use Component Object Model (COM) technologies in one of two ways: by using custom controls (OCXs) or by using Automation (C/SIDE in the role of an automation controller or client) to perform tasks with other applications. To create the e-mail confirmations, use C/SIDE as an automation server.

Using an automation server consists of five basic steps:

1. Declare the creatable (top-level) interface (class) of the automation server as a variable of type Automation.
2. Declare all the other interfaces (classes) as variables of type Automation.
3. Use the C/AL function CREATE on the variable declared in step 1. Do not use CREATE on any other variables.
4. Use the methods and properties of the automation server in C/AL code. The syntax and semantics for these methods and properties are documented in the documentation for each automation server.
5. The top-level object can be destoryed (with CLEAR) if desired. Otherwise, it is automatically destroyed when the variable goes out of scope.

It is a good idea to create a separate codeunit for code that uses automation because of performance issues, and because an object using automation can only be compiled on a machine on which the automation server is installed.

The data being transferred must be in text format.

Using Custom (or OCX) Controls

Microsoft Dynamics NAV supports custom controls (which are also known as OLE controls or ActiveX® controls), but it only supports non-visual controls.

To use a custom control, it must be physically installed on the target machine and registered with the operating system.

To use a custom control, declare it as a global or local variable with a data type of OCX. Then select the desired custom control from a list by clicking the lookup button in the **Subtype** field. If the desired control is not in the list, the control can be installed and registered manually. Information on how to do this is in the Application Designer's Guide.

Once the control is declared as a variable, its methods and properties can be accessed. To see the methods and properties available, use the C/AL Symbol menu. Methods can be run just as any other function is run in C/AL, and properties can be read and set the same as any other object properties.

XMLPort Triggers

There are three events that fire for an XMLPort object:

- **OnInitXMLPort**: This trigger fires when the XMLport is loaded and before any table views and filters are set.
- **OnPreXMLPort**: This trigger fires after the table views and filters are set and before the XMLport is run.
- **OnPostXMLPort**: This trigger fires after all the data items in an XMLPost have been processed.

There are also event triggers for every field in an XMLPort which are dependent on whether the XMLPort is importing or exporting. For more information on those triggers, consult the online C/SIDE Reference Guide.

File Handling

Use file-handling to create the XML file. Through File Variables, import data from or export data to any file accessible through the operating system.

The File Data Type

To gain access to an external file from within C/SIDE, first declare a variable of type File. This is a complex data type, which has numerous methods (functions) used to open, read, write, and close external files.

Each File variable can be used to access one file as a time. This makes is neceesary to declare one File variable for each file that must be accessed simultaneiously.

Opening Files for Import or Export

Before a file can be opened for use, it must be set up properly. First determine whether it will be used for reading (import) or for writing (export). Although theoretically a file can be open for writing and then read as well, this is normally not done in Microsoft Dynamics NAV.

The following bulleted list contains the File methods used to prepare a file for opening and for actually opening (and closing) it. As with other methods, these are called through the File variable. For example, if the File variable is named **ImportFile**, then using the **Open** method would look like this:

```
ImportFile.OPEN(Name);
```

Skip the "ImportFile." in front of each of these methods:

- **WRITEMODE(NewWriteMode)**: Sets the Read/Write status of a File variable before opening the file. If **NewWriteMode** is TRUE, the file can be writeen. If **NewWriteMode** is FALSE, then the file can only be used for reading. Note that another way to use this method is: **IsWriteMode := WRITEMODE;.** By using this method after the file is open, you can tell whether a file is available for writing or not.
- **TEXTMODE(NewTextMode)**: Sets the type of data to be read or written. If TRUE, the file is opened in Text mode, and if FALSE, it is opened in Binary mode. This method can only be used before opening the file. Note that another way to use this method is: **IsTextMode := TEXTMODE;**. With this method (used only after the file is open), it can be determined whether a file is being read or written in text mode or binary mode.
- **QUERYREPLACE(NewQueryReplace)**: Use this method for opening a file using the **CREATE** method. If **NewQueryReplace** is TRUE, and the file already exists, C/SIDE asks the user before a replace is allowed. If NewQueryReplace is FALSE, and the file already exists, C/SIDE erases the old file and replaces it with a new (empty) one, without asking the user. If the file does not already exist, this method has no effect.
- **OPEN(FileName)**: Use this method to open an already existing file. Be sure to set the WRITEMODE and the TEXTMODE before opening a file. FileName is the full path name of the file. When used as a statement (without looking at the return value), and the file indicated by **FileName** does not exist, **OPEN** causes a run-time error occurs. If OPEN is used as an expression (looking at the return value), and the file indicated by **FileName** does not exist, **OPEN** returns **FALSE**, but if the file does exist, it is opened, and **OPEN** returns **TRUE**.
- **CREATE(FileName)**: Use this method to create and open a file. Be sure to set the WRITEMODE, the **TEXTMODE**, and **QUERYREPLACE** before creating a file. FileName is the full path name of the file. If **CREATE** is used as a statement (without looking at the return value), and the file indicated by FileName cannot be created (say the path does not exist), a run-time error occurs. If **CREATE** is used as an expression (looking at the return value) and the file indicated by **FileName** cannot be created, **CREATE** returns **FALSE**. But if the file is created, it is opened, and **CREATE** returns **TRUE**. If the file already exists, it is cleared and opened. Whether the user is warned about this or not depends on the parameter to the **QUERYREPLACE** method called before calling **CREATE**.
- **CLOSE**: Use this method to close access to the file through this file variable. Once this is called, the file variable cannot be used again to access a file unless it is re-opened or re-created. If **CLOSE** is called and the file is not open, a run-time error occurs.

Methods of Reading Files

There are two methods for reading or writing data in external files. The method is set using the previously described TEXTMODE method. There are two possible settings:

- **TEXT (TEXTMODE=TRUE)**: Each file access reads or writes a line of text. The variable used can be of any type and it is converted to (if writing) or from (if reading) text during the processing.
- **BINARY (TEXTMODE=FALSE)**: Each file access reads or writes a single variable. The variable is read or written in its internal format. For example, text is written as a null terminated string whose length is the defined length of the Text variable. Since TEXTMODE is limited to one variable per line, note that the file cannot contain more than 250 characters per line. Binary mode does not have this limitation, since it does not have lines. However, the internal format can only be read if it was written in the same manner from Microsoft Dynamics NAV. The one variable type that is useful when using Binary mode is the Char type, which enables reading or writing one byte at a time. In the SkeletonFileImport object, use Binary Mode and the Char type to import text files that can be of any length.

Reading or Writing Data in External Files

The following methods are used to read or write data in external files (once the file variable is open):

- **[Read :=] READ(variable)**: Read a variable from the file. The optional return value indicates the number of bytes read. In Text mode, a line of text is read and the text is evaluated into the variable passed in as a parameter. In Binary mode, the variable is read in its internal format, and thus the number of bytes read depends on the size of the variable.
- **WRITE(variable)**: Write a variable to the file. In Text mode, the variable is formatted to text and written out as a line of text. In Binary mode, the variable is written using its internal format.

E-mail Confirmation

E-mail confirmation provides the ability to send e-mail from Microsoft Dynamics NAV programatically. In this topic, a solution for providing e-mail confirmation to seminar participants is designed and implemented.

Solution Analysis

The client's functional requirements in Chapter 1 describe e-mail confirmation like this:

The solution should allow e-mail notification to be sent to the customer's participants in several situations, such as registration confirmation.

This feature enables seminar managers to send out e-mail confirmations to all of the participants registered for a seminar to inform them that they are registered and to remind them of the starting date.

Solution Design

Before attempting to implement a solution, it is vital to create a design, or plan. In this case, GUI changes will be necessary to make the feature accessible to seminar managers, and functions will be needed to execute the e-mail confirmation logic.

GUI Design

E-mail confirmation will be available from the **Seminar Registration** form. This will require the **Seminar Registration** form to be modified by adding a new **Functions** menu button as shown in Figure 9-1.

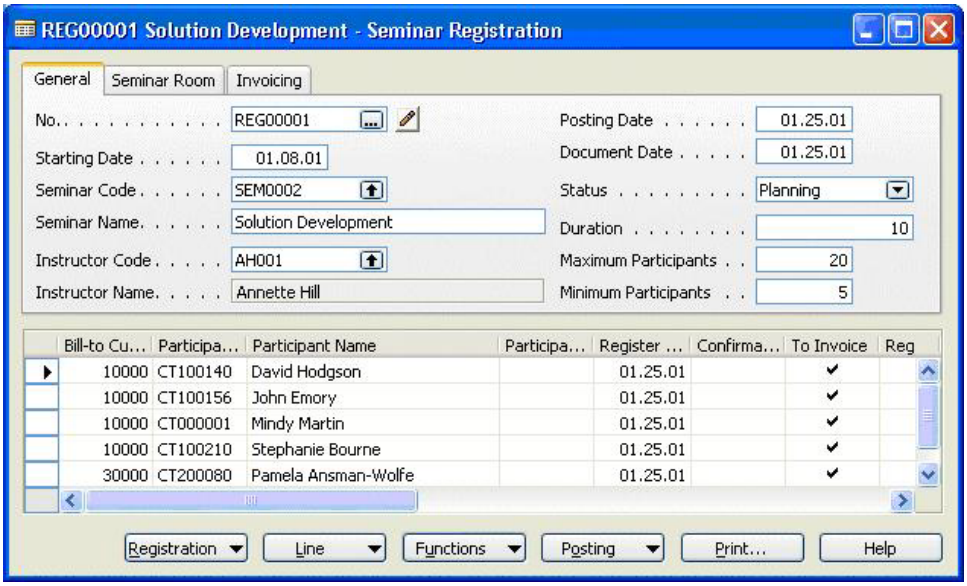

FIGURE 9-1: THE SEMINAR REGISTRATION FORM (123456710)

The e-mail confirmation functionality will also be available from the **Seminar Registration List** form, so a **Functions** button will be added to this form as well, as shown in Figure 9-2.

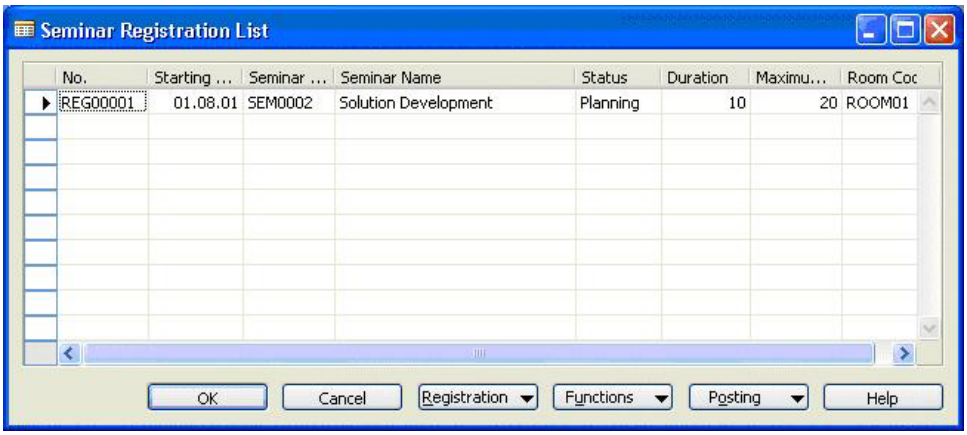

FIGURE 9-2: THE SEMINAR REGISTRATION LIST FORM (123456713)

Functional Design

To send e-mails, use an Automation variable with the subtype of an automation server class that can create and send e-mails. This automation server is the **Navision Attain ApplicationHandler**, and the class used is **MAPIHandler**. The implementation will require a new codeunit that can create and send the e-mails.

Lab 9.1 - Creating E-mail Confirmations

Begin by creating a codeunit to create the e-mail:

1. Create a codeunit called Seminar Mail (123456705).
2. Declare global record variables for the Seminar Registration Header, Seminar Registration Line, Customer, and Contact tables.
3. Declare an integer for storing the number of errors.
4. Declare a variable called MAPIHandler, of type Automation and of subtype Navision Attain ApplicationHandler and of class MAPIHandler.
5. Define text constants for the different sections of the e-mail confirmation, including the subject line, the greeting, the confirmation sentence, and the signature.
6. Define a function called NewConfirmationMessage that has a parameter of a record variable for the Seminar Registration Line table, which is passed by reference.
7. Enter code in the function trigger so that the function performs the following tasks:
   - Create an instance of the **MAPIHandler** (using the **CREATE** function).
   - Create the e-mail message by assigning the appropriate elements of the message to the properties of the **MAPIHandler** variable. Assign the **ToName** field using the **Contact** record, the **CCName** using the **Customer** record, and the Subject
   - Use the **AddBodyText** method of the **MAPIHandler** to create each line of the e-mail message.

***NOTE**: To see all the properties and methods available in the SMTP Mail codeunit, open the C/AL Symbol Menu by pressing F5 or clicking View, C/AL Symbol Menu, and locate the SMTP variable in the left most column.*

   - Sends the message using the **Send** method of the **MAPIHandler**.
   - Use the **ErrorStatus** property of the **MAPIHandler** to check the number of errors. If there are none, set the **Confirmation Date** of the **Seminar Registration Line** to today's date.

8. Define a function called SendAllConfirmations with a parameter of a record variable for the Seminar Registration Header.

9. Enter code in the function trigger so that the function runs the NewConfirmationMessage function for each Seminar Registration Line associated with the Seminar Registration Header that was passed to it.

10. With the code in place, the remaining step is to make the functionality available to the user. Add a new menu button and menu item to the Seminar Registration form (123456710) as follows:

Menu Button	Options	Comments
Functions	Send E-mail Confirmations	Runs code to Send E-mail Confirmations.

11. Enter code in the appropriate trigger so that when the user selects the Send E-mail Confirmations menu item, the program runs the **SendAllConfirmations** function.

12. Add a new menu button and menu item to the **Seminar Registration List** form (123456713) as follows:

Menu Button	Options	Comments
Functions	Send E-mail Confirmations	Runs code to Send E-mail Confirmations.

13. Enter code in the appropriate trigger so that when the user selects the Send E-mail Confirmations menu item, the program runs the **SendAllConfirmations** function.

XML Participant List

Solution Analysis

The client's functional requirements in Chapter 1 describe the need for a participant list in XML format:

The participant list for a seminar should be exportable as an XML file.

Solution Design

Generating a participant list in XML can be accomplished using an XMLPort.

GUI Design

The following menu item will need to be added to the **Partner Menu** MenuSuite (80):

Menu Type	Menu Name	Group	Comments
Item	Create XML List	Periodic Activities	Runs report 123456705 XML Sem. Reg.-Participant List.

Functional Design

To implement this solution, it is necessary to understand the basics of how XMLPorts work. XMLPorts have three basic sections:

- A title, in this case, "Seminar Registration – Participant List".
- Heading information, in this case from the **Seminar Registration Header** including the **No.**, **Seminar Code**, **Seminar Name**, **Starting Date**, **Duration**, **Instructor Name**, and **Room Name** fields.
- Line information, in this case one for each registered participant from the **Seminar Registration Line**, including **Customer No**., **Contact No**., and **Name** fields.

The following table shows the basic sections that are necessary for this solution:

Section	Sample Section
Title	<Seminar_Registration_–_Participant_List>
Heading for Heading information	<Seminar>
Heading information	<No>REG0001</No>
Heading for Line information	<Participant>
Line information	<Customer_No>10000</Customer_No>

Lab 9.2 - Creating the XML Sem. Reg.-Participant List

Begin by creating the XMLport:

1. Create a new XMLport called **Sem. Reg.-XML Participant List** (123456700).
2. Define the data items for the XMLport, using the previous Functional Design section as a reference for which tables to get information from. Add code to calculate the FlowFields so that the **Instructor Name** and **Participant Name** appear in the XML file.

Now implement a way that the user can run this XMLport. To enable the user to select filters, create a new report.

3. Create a new Processing Only report called **Sem. Reg.-XML Participant List** (123456705).
4. This report has one data item, the **Seminar Registration Header** table. Set the name of this datatype to **SEMREGHEADER**.
5. Create global variables for **TestOutStream** (datatype Outstream) and **TestFile** (datatype File).
6. In the **OnPreReport** trigger, insert this code, substituting the file path:

```
TestFile.CREATE('C:\XML_Part_List.XML');
TestFile.CREATEOUTSTREAM(TestOutStream);
XMLPORT.EXPORT(123456700,TestOutStream,SEMREGHEADE
R);
TestFile.CLOSE;
MESSAGE('XML Completed');
```

7. All that remains is to add the menu item to the **Seminar** main menu. Add the new menu item to the **Periodic Activities** folder in the **Partner** MenuSuite.

Sample XML Participant List

The resulting XML participant list might look like this:

```
- <Seminar_Registration_-_Participant_List>
  - <Seminar>
          <No>REG00001</No>
          <Seminar_Code>SEM0002</Seminar_Code>
          <Seminar_Name>Solution Development</Seminar_Name>
          <Starting_Date>01.08.01</Starting_Date>
          <Duration>10</Duration>
          <Instructor_Name>Annette Hill<Instructor_Name />
          <Room_Name>Room 1</Room_Name>
- <Participant>
          <Customer_No>10000</Customer_No>
          <Contact_No>CT100140</Contact_No>
          <Name>David Hodgson</Name>
  </Participant>
- <Participant>
          <Customer_No>10000</Customer_No>
          <Contact_No>CT100156</Contact_No>
          <Name>John Emory</Name>
  </Participant>
- <Participant>
          <Customer_No>10000</Customer_No>
          <Contact_No>CT000001</Contact_No>
          <Name>Mindy Martin</Name>
  </Participant>
- <Participant>
          <Customer_No>10000</Customer_No>
          <Contact_No>CT100210</Contact_No>
          <Name>Stephanie Bourne</Name>
  </Participant>
  </Seminar>
  </Seminar_Registration_-_Participant_List>
```

Conclusion

In this chapter, COM automation was used to send e-mail messages to seminar participants and an XMLport to create an XML file of participants in a seminar.

At this point in the course, the seminar solution is complete. The next chapter addresses the deployment of custom solutions.

Test Your Knowledge - Interfaces

Review Questions

1. In what two ways can you use COM objects in C/SIDE?

2. When you want to use a variable for an automation server, what data type do you give the variable?

3. What C/AL function do you use to create an instance of an automation server class?

4. Can you transfer picture files to an automation server?

5. Where should you ideally place code that uses automation?

6. What event triggers exist for XMLports?

Quick Interaction: Lessons Learned

Take a moment to note three key points you have learned from this chapter:

1.

2.

3.