

MODULE 10: INTERFACES

Module Overview

Microsoft Dynamics NAV® 2013 frequently exchanges information with external systems. For example, you may have to send electronic payment information to a bank, exchange purchase order information with your vendor's business management application, or receive item information from a bar code or RFID reader. In those situations, Microsoft Dynamics NAV 2013 performs only part of the transaction or the process. It interfaces with external software or hardware to perform other parts of the transaction or the process.

Microsoft Dynamics NAV 2013 includes several features that streamline interfacing to external applications or devices. This module covers the following features:

- Component Object Model (COM) Technologies
- File handling
- Microsoft .NET Interoperability

Objectives


- Explain how to use Automation and OCX to perform tasks with other applications.
- Describe file handling functions to import or export data.
- Design and implement email capability.

Prerequisite Knowledge

To interface Microsoft Dynamics NAV 2013 with other applications, use Component Object Model (COM) technologies in the Microsoft Dynamics NAV Development Environment to extend the functionality of Microsoft Dynamics NAV 2013. Also use external files to exchange information between Microsoft Dynamics NAV 2013 and third-party applications or devices.

Component Object Model (COM) Technologies


Use Component Object Model (COM) technologies in the Microsoft Dynamics NAV Development Environment to extend the functionality of Microsoft Dynamics NAV 2013. In the development environment, you can use two types of COM technologies: automation and custom controls (OCX).

 **Note:** *Automation and OCX objects are not supported under the Microsoft Dynamics NAV Web client and SharePoint applications that run Microsoft Dynamics NAV Portal Framework.*

Automation

Automation is a client/server infrastructure that enables one application to access and communicate with another application. By using automation, an application, such as Microsoft Office Word, exposes its internal functions and routines as automation objects. Microsoft Dynamics NAV 2013 accesses these objects through an automation client that runs in the Microsoft Dynamics NAV 2013 client for Windows. The application that exposes the automation object, such as Word, acts as the automation server, and the automation client acts as the client.

Automation enables tasks that run manually to run automatically instead. For example, you can write a script to extract data from a database, put the data into a Microsoft Office Excel workbook, and then display the data graphically.

 **Note:** *Microsoft Dynamics NAV 2013 Server is a 64-bit application. However, not all COM components can run on 64-bit operating systems. Therefore, automation objects can run only on the client side. If you try to create or run any automation objects on the server side, a run-time error occurs. Instead of implementing server-side automation objects, use Microsoft .NET Framework Interoperability.*

Custom Controls (OCX)

Custom controls are OLE Control Extensions (OCX) or ActiveX controls. These are specific types of Automation objects. OCX and ActiveX controls are generally small programs or application objects that you start from the Microsoft Dynamics NAV 2013 application to perform a specific function or task. Use custom controls for various types of tasks.

By default, there are several available custom controls in Microsoft Dynamics NAV 2013 on the **Tools** menu in the development environment. To develop custom controls, you can use tools such as Microsoft Visual C++ or Microsoft Visual Basic. Both products use wizards that make it easy to develop COM objects. You can also develop functional controls without understanding the complex details of COM.

Only nonvisual controls are supported. You cannot use a control to add graphical elements to a Microsoft Dynamics NAV object. For example, you cannot add a third-party control to a page. However, the control can display information and interact with the user in its own window.



Additional Reading: *If you must add visual, graphical, or user-interface elements to a page in the Microsoft Dynamics NAV 2013 client for Windows, use control add-ins. Control add-ins are delivered as Microsoft .NET Framework-based assemblies that you install on computers that run the Microsoft Dynamics NAV 2013 client for Windows. You register the control add-in in Microsoft Dynamics NAV Server. Then use the control add-in with pages. To read more about the control add-ins, refer to the RoleTailored Client Control Add-in Overview topic in Microsoft Dynamics NAV Developer and IT Pro Help.*

Automation and OCX Variables

To use an automation object or ActiveX control, you must declare a variable of type Automation or OCX. Before you can use an automation object, you must create an instance of the object by calling the CREATE function.

The following is the syntax of the **CREATE** function..

```
[Ok :=] CREATE(Automation [,NewServer] [,OnClient])
```

The following are the parameters of the CREATE function.

Parameter	Type	Remarks
Automation	Automation	The Automation variable that you previously declared.
NewServer	Boolean	If NewServer is FALSE (this is the default value), then the CREATE function tries to reuse an already running instance of the automation server that is referenced by Automation before it creates a new instance. If NewServer is TRUE, then the CREATE function always creates a new instance of the automation server.
OnClient	Boolean	This parameter is used for backward compatibility. In Microsoft Dynamics NAV 2013 it must evaluate to TRUE. You can either set the parameter to TRUE, or define a Boolean type variable that evaluates to TRUE. If you omit this parameter or use the FALSE Boolean constant, a compile-time error occurs. If you use an expression or a variable that evaluates to FALSE at run time, then a run-time error occurs.

If you test for the return value, you receive TRUE if the automation object was successfully created, and FALSE if it could not be created. If you do not test for the return value, and the creation fails, then a run-time error occurs.

COM Considerations

If you want only to use existing automation objects or custom controls, then you do not require extensive COM knowledge. The functionality that is provided by a COM object is the same as any C/AL function.

However, following are several items to consider when you use automation:

- Exception handling is not supported. You cannot retrieve information about exceptions from a control or automation server.
- You must install the COM object or the whole application that exposes its functionality through automation on every computer that runs Microsoft Dynamics NAV 2012 client for Windows.
- C/AL data types do not map directly to COM data types. So, the mechanisms for calling methods in a control, passing parameters, and receiving return values are somewhat complex. You frequently must study the documentation for the automation server to learn about the actual data types that methods of COM objects accept or return. This is especially true if methods or properties use the IDispatch type or enumerations.



Additional Reading: To learn more about how to use COM in Microsoft Dynamics NAV 2013, refer to the *Extending Microsoft Dynamics NAV Using COM* topic in the Microsoft Dynamics NAV Developer and IT Pro Help.

Demonstration: Use Automation to Create a Chart in Microsoft Excel

In this demonstration, you transfer data from the **G/L Entry** table to Microsoft Office Excel and create a chart. This example shows how to handle enumerations by creating a chart in Excel that shows the distribution of personnel expenses by departments.

Demonstration Steps

1. Create a codeunit and declare variables.
 - a. In Object Designer, click **Codeunit**, and then click **New** to create a new codeunit.
 - b. On the **View** menu, click **C/AL Globals**.
 - c. On the **Variables** tab, define the following variables.

Name	Data Type	Subtype
GLEntry	Record	G/L Entry
xlApp	Automation	Application
xlBook	Automation	Workbook
xlSheet	Automation	Worksheet
xlChart	Automation	Chart
xlRange	Automation	Range



Note: All of the automation objects must be from the Microsoft Excel 14.0 Object Library automation server. When creating these variables, in the Subtype property, click the **AssistEdit** button, then in the Automation Server text box, look up the Microsoft Excel 14.0 Object Library automation server. Then, select the appropriate class, as indicated in the Subtype column.

- d. Close the **C/AL Globals** window.
2. Add code to filter the **G/L Entry** table to only the relevant records.
 - a. In the OnRun trigger, enter the following C/AL code.

```
GLEntry.SETCURRENTKEY(  
    "G/L Account No.",  
    "Business Unit Code",  
    "Global Dimension 1 Code",  
    "Global Dimension 2 Code",  
    "Posting Date");  
GLEntry.SETFILTER("G/L Account No.",'8700..8790');
```

3. Add code to create an instance of Excel.
 - a. Append the following C/AL code to the end of the OnRun trigger.

```
CREATE(xlApp,FALSE,TRUE);  
xlApp.Visible := TRUE;
```

4. Add code to add a new workbook to Excel.
 - a. Append the following C/AL code to the end of the OnRun trigger.

```
xlBook := xlApp.Workbooks.Add(-4167);  
xlSheet:= xlApp.ActiveSheet;  
xlSheet.Name := 'Personnel Expenses';
```



Note: In the code example that was mentioned earlier, value -4167 substitutes the *xlWBATWorkSheet* enumerator. To find enumerator values of COM classes, you can use Visual Studio. For Excel, you can also use Visual Basic Editor. This is built into Excel.

5. Add code to transfer data from Microsoft Dynamics NAV 2013 to Excel.
 - a. Append the following code to the end of the OnRun trigger.

```
GLEEntry.SETRANGE("Global Dimension 1 Code",'ADM');  
  
GLEEntry.CALCSUMS(Amount);  
  
xlSheet.Range('A2').Value := 'Administration';  
  
xlSheet.Range('A3').Value := GLEEntry.Amount;  
  
GLEEntry.SETRANGE("Global Dimension 1 Code",'PROD');  
  
GLEEntry.CALCSUMS(Amount);  
  
xlSheet.Range('B2').Value := 'Production';  
  
xlSheet.Range('B3').Value := GLEEntry.Amount;  
  
GLEEntry.SETRANGE("Global Dimension 1 Code",'SALES');  
  
GLEEntry.CALCSUMS(Amount);  
  
xlSheet.Range('C2').Value := 'Sales';  
  
xlSheet.Range('C3').Value := GLEEntry.Amount;
```

6. Add code to create the chart.
 - a. Append the following code to the end of the OnRun trigger.

```
xlRange := xlSheet.Range('A2:C3');  
  
xlChart := xlBook.Charts.Add;  
  
xlChart.Name := 'Personnel Expenses - Chart';  
  
xlChart.ChartWizard(xlRange,-4102,7,1,1,0,0,'Personnel Expenses');
```

7. Test the code.
 - a. Save the codeunit as 90005, Create Excel Chart.
 - b. Close the **C/AL Editor** window.
 - c. In Object Designer, locate the codeunit 90005, Create Excel Chart.
 - d. Click **Run**.

The “Excel Chart Created through Automation” figure shows the output of the **Create Excel Chart** codeunit.

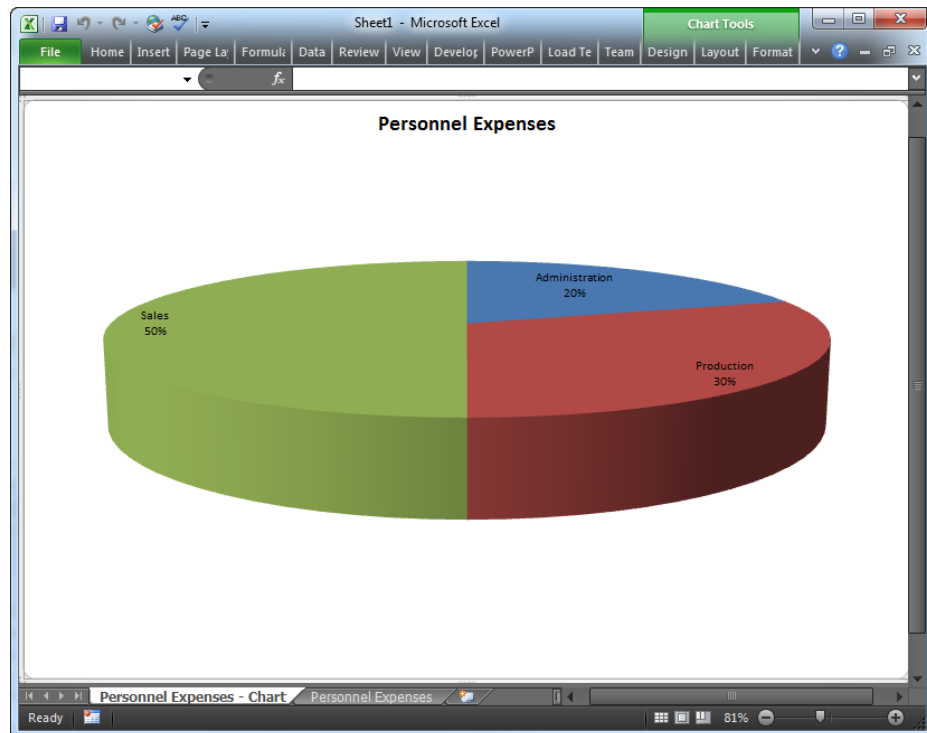


FIGURE 10.1: EXCEL CHART CREATED THROUGH AUTOMATION

File Handling

Use file-handling to import or export text-based data files that require parsing to interpret on input, or special formatting when output. For example, you can use file handling to import data that is not record-oriented and does not have either fixed-width or delimited structure. You can import data from or export data to any external file through file variables.

The File Data Type


To access an external file in C/SIDE, first declare a variable of type File. This is a complex data type. It has many functions that are used to open, read, write, and close external files.

Use each file variable to access one file at a time. If you want to access multiple files at the same time, you must declare one file variable for each file that you access.

The following table explains the most important file functions that are used to handle the files.

Function	Remarks
OPEN	Opens an ASCII or binary file. This function does not create the file if it does not exist. If you call OPEN on a file variable that refers to an open file, then a run-time error occurs. The OPEN function does not automatically close the existing file and open a new file.
CREATE	Creates and opens an ASCII or binary file. If the file already exists, it is truncated and then opened. Similar to OPEN , if you call CREATE on a file variable that refers to an open file, then a run-time error occurs.
CLOSE	Closes a file that was opened by OPEN or CREATE . If the file is not open, a run-time error occurs. You always must call explicitly CLOSE if you intend to reuse the same file variable for accessing more than one file.
WRITEMODE	Sets the read/write status of a file variable before you open the file. You cannot write to files that are not in write mode. You can also use this function to test whether the file is in write mode. You cannot switch between read and write modes. Therefore, make sure that you call WRITEMODE before you open a file.
TEXTMODE	Sets whether a file should be opened as an ASCII file or a binary file. Retrieves the current setting of this option for a file. You cannot switch between text and binary modes. Therefore, make sure that you call TEXTMODE before you open or create a file.
READ	Reads from an ASCII or binary file. If the file is in text mode, it reads a single line of text into a text variable. If the file is in binary mode, it reads a single variable of a simple C/AL data type. READ does this by reading as many bytes from the external file as the variable occupies in memory.
WRITE	Writes to an ASCII or binary file. If the file is in text mode, it writes the variable formatted as text followed by a new line character. If the file is in binary mode, it writes the variable exactly as it is stored in memory.

Function	Remarks
POS	Retrieves the current position of the file pointer in an ASCII or binary file.
LEN	Retrieves the length of an ASCII or a binary file. The length is always reported in bytes.

 **Additional Reading:** *There are many more functions on the File data type. To learn more about these functions, refer to the Microsoft Dynamics NAV Developer and IT Pro Help.*

ASCII and Binary Modes

There are two methods for reading or writing data in external files. You set the desired method by calling the **TEXTMODE** function. Following are two possible settings:

- ASCII (**TEXTMODE**=TRUE) – Each file access reads or writes a line of text. Use any type variable. It is converted to or from text during the reading or writing.
- BINARY (**TEXTMODE**=FALSE) – Each file access reads or writes a single variable. The variable is read or written in its binary format as it is stored in memory. The internal format can only be read if it is written in the same byte sequence that Microsoft Dynamics NAV 2013 uses to store variables. This is not a limitation when it reads binary files that are created by Microsoft Dynamics NAV 2013. However, you may experience issues when you handle files that were created by other applications or systems.

Demonstration: Write to and Read from Binary Files

In this demonstration, you write values of three variables into a binary file, clear the variables, and then read those variables back from the binary file.

Demonstration Steps

1. Create a codeunit and declare variables.
 - a. In Object Designer, click **Codeunit**, and then click **New** to create a new codeunit.
 - b. On the **View** menu, click **C/AL Globals**.
 - c. On the **Variables** tab, define the following variables.

Name	Data Type	Length
Int	Integer	
Bool	Boolean	

Name	DataType	Length
String	Text	250
Bin	File	

2. Create two functions to write and read contents of the variables to and from a file.
 - a. On the **Functions** tab, define the following two functions: **Save** and **Load**. The functions do not accept any parameters or return any values.
 - b. Close the **C/AL Globals** window.
 - c. In the function trigger of the **Save** function, enter the following C/AL code.

```

BinFile.TEXTMODE(FALSE);

BinFile.CREATE('C:\Temp\Variables.txt');

BinFile.WRITE(Int);

BinFile.WRITE(Bool);

BinFile.WRITE(String);

BinFile.CLOSE;
```

- d. In the function trigger of the **Load** function, enter the following C/AL code.

```

BinFile.TEXTMODE(FALSE);

BinFile.OPEN('C:\Temp\Variables.txt');

BinFile.READ(Int);

BinFile.READ(Bool);

BinFile.READ(String);

BinFile.CLOSE;
```

3. Add code that does the following:
 - Sets values to variables.
 - Saves the variables.
 - Clears the variables.
 - Loads the variables.
 - Displays variable values to the user.

- a. In the OnRun trigger, enter the following code.

```
Int := 15;

Bool := TRUE;

String := 'Hello, World!';

Save;

CLEARALL;

Load;

MESSAGE('Int: %1\Bool: %2:String: %3',
  Int,Bool,String);
```

4. Test the code.
 - a. Save the codeunit as 90004, Read and Write Variables.
 - b. Close the **C/AL Editor** window.
 - c. In Object Designer, locate the codeunit 90004, Read and Write Variables.
 - d. In Windows Explorer, create the C:\Temp folder.
 - e. Click **Run**.

Microsoft .NET Interoperability

You can extend Microsoft Dynamics NAV 2013 with functionality that is available in Microsoft .NET Framework assemblies. You can take advantage of the .NET Framework interoperability so that Microsoft Dynamics NAV objects can interact with .NET Framework objects. In the Microsoft Dynamics NAV objects, you can reference .NET Framework assemblies and call their members directly from C/AL code. Use assemblies from the .NET Framework class library. These are found in the global assembly cache, your own custom assemblies, or third-party assemblies.

.NET Framework interoperability offers an alternative to COM so that you can extend your solution. For example, you can use .NET Framework interoperability to do the following:

- Consume web services.
- Integrate with Microsoft Office products.
- Create .NET Framework applications that target the Microsoft Dynamics NAV 2013 client for Windows.



Note: .NET Framework objects can run both on the server side and client side. Client-side objects are not supported under the Microsoft Dynamics NAV Web client and SharePoint applications that run Microsoft Dynamics NAV Portal Framework.



Additional Reading: To learn more about Microsoft .NET Framework interoperability, refer to Microsoft .NET Framework Interoperability module of the course C/SIDE Introduction in Microsoft Dynamics NAV 2013, or to the Extending Microsoft Dynamics NAV Using Microsoft .NET Framework Interoperability topic in Microsoft Dynamics NAV 2013 Developer and IT Pro Help.

Email Confirmation

CRONUS International Ltd. wants to automate its communication with their customers as much as they can. To meet this requirement, you must design and develop a solution that sends email confirmations to all seminar participants.



Solution Design

The Functional Requirements Document for Microsoft Dynamics NAV 2013 implementation at CRONUS International Ltd. states the following:

- When a seminar is confirmed, seminar managers must be able to send an automatic email confirmation to all registered seminar participants.
- Seminar managers must be able to easily customize the template for the seminar confirmation notification email. The template must allow for the following:
 - Ability to enter free text
 - Placeholders for recipient name
 - Seminar name
 - Seminar starting date

Sending Email from Microsoft Dynamics NAV 2013

For the first requirement, you must provide a capability to send email messages from Microsoft Dynamics NAV 2013. Following are several capabilities that help you develop this functionality and meet this requirement:

- Microsoft Outlook automation library
- Microsoft Collaboration Data Objects (CDO) automation library
- Microsoft.Dynamics.Nav.SMTP .NET assembly that is provided with Microsoft Dynamics NAV 2013

- Codeunit 397, Mail
- Codeunit 400, SMTP Mail

Generally, when you select the optimal solution from a range of options, you should always opt for the simplest choice that meets the customer's requirements. It should provide as many of the following criteria as possible:

- Ease of development and maintenance
- Maximum forward compatibility
- Least amount of dependencies on outside applications
- No unnecessary dependencies on third-party products
- No redundancies with existing capabilities
- Fewest changes to customer infrastructure

When you compare these choices with the list of available technologies, you should discard automation, because it does not provide maximum forward compatibility. It runs on the client that requires the customer to install and maintain a compatible version of the automation server on each client computer.

Do not use the Microsoft.Dynamics.Nav.SMTP .NET assembly directly. It is not the simplest solution to develop and maintain, and it may not be sufficiently forward compatible. It also creates redundancy with the existing functionality of Mail and SMTP Mail codeunits.

The Mail and SMTP Mail codeunits already provide email capabilities. Therefore, you should select one of these two options instead of developing a completely custom email sending capability. The Mail codeunit depends on Outlook. Therefore, it may require configuration and maintenance on all client computers. The SMTP Mail codeunit depends on an external SMTP server. Because CRONUS International Ltd. uses a dedicated SMTP server, this requires less maintenance and no infrastructure changes to the CRONUS environment.

Therefore, your solution should use the SMTP Mail codeunit to manage sending email confirmations to seminar participants.

Customizing the Template

To enable seminar managers to customize the template, you can select among several options, such as the following:

- Importing a template into a BLOB field in a setup table
- Providing a table with text fields where each row represents a line in the confirmation email
- Using an external template file
- Using a file in a SharePoint library

The solution in this module selects the external template file. Therefore, you must provide a text field in the **Seminar Setup** table where users can configure the path of the text file that contains the confirmation email template.



Note: *The suggested solution in this module does not imply that this is the best choice. There are very strong arguments for and against any of the given choices.*

To enable users to specify free text and use placeholders for the location of a recipient's name, seminar name, and seminar starting date, use the **STRSUBSTNO** function and replace the numbered specifiers (%1, %2, %3, and so on) with variables. Because CRONUS International Ltd. needs only three variables, you can define the following placeholders to be used in the confirmation template.

Placeholder	Meaning
%1	Name of the participant
%2	Name of the seminar for which the participant is registered
%3	Starting date of the seminar

Solution Development

To meet CRONUS International Ltd. requirements, you must create and customize several objects.

Tables

Customize the following table.

Table	Changes
123456701 Seminar Setup	Add a new field to enable users to configure the path of the confirmation email template file.

Pages

Customize the following pages.

Page	Changes
123456702 Seminar Setup	Add the new field from the Seminar Setup table.
123456710 Seminar Registration	Add an action to send seminar registration emails.

Page	Changes
123456713 Seminar Registration List	Add an action to send seminar registration emails.

Codeunits

Create the following codeunit.

Codeunit	Remarks
123456705 SeminarMailManagement	Contains functions to send seminar confirmation emails to registered participants of a seminar.

Lab 10.1: Create Email Confirmations

Scenario

You are a developer at a company that is implementing Microsoft Dynamics NAV 2013 at CRONUS International Ltd. Your goal is to develop the customizations to enable seminar managers to send automated email confirmations to seminar participants.

You first have to enable users to configure the email template that the solution uses to send confirmations to registered participants. Then you must make sure that any configuration settings in CRONUS International Ltd. database enable Microsoft Dynamics NAV 2013 to interface with the SMTP server that is installed at the localhost server, and send email confirmations to participants. Finally, you must develop the codeunit that automates sending confirmations, and add an action that calls the codeunit to the Seminar Registration pages.

Exercise 1: Import the Setup Table and Page

Exercise Scenario

Your colleague Isaac customized the **Seminar Setup** table and corresponding page to allow users to configure the path of the email template file.

Import the file that Isaac provides.

Task 1: Import the Object File

High Level Steps

1. Import the starter object.

Detailed Steps

1. Import the starter object.
 - a. Click **File > Import**.
 - b. Browse to the file Mod10\Labfiles\Lab 10.A - Starter.fob.
 - c. Click **Open**.
 - d. Click **OK** to open the Import Worksheet. The **Import Worksheet** window displays the following objects.

Type	No.	Name
Table	123456701	Seminar Setup
Page	123456702	Seminar Setup

- e. Click **Replace All**, and then click **OK** to complete the import.
- f. Click **OK** to close the **Import Objects** window.

Exercise 2: Verify the Configuration

Exercise Scenario

The next step is to make sure that Microsoft Dynamics NAV 2013 communicates with an SMTP server by configuring the **SMTP Mail Setup** page. Then you must make sure that there is an email address in the **Company Information** page. This email address is used in the **From** field for the outgoing email confirmations.

Task 1: Configure the SMTP Mail Setup Card

High Level Steps

1. Configure **SMTP Mail Setup** to use the local SMTP server.

Detailed Steps

1. Configure **SMTP Mail Setup** to use the local SMTP server.
 - a. In the **Search** field, enter "SMTP Mail Setup."
 - b. In the results, click the **SMTP Mail Setup** page.
 - c. In the **SMTP Server** field, enter "localhost."
 - d. Make sure that the **Authentication** field is set to Anonymous.



Note: Actual SMTP settings may be different for any real-life scenario.

- e. Close the **SMTP Mail Setup** page.

Task 2: Configure the Company Info Card

High Level Steps

1. Configure an email address in the Company Info card.

Detailed Steps

1. Configure an email address in the Company Info card.
 - a. In the **Search** field, enter "Company Information."
 - b. In the results, click the **Company Information** page.
 - c. On the **Communication** FastTab, in the **E-Mail** field, enter "cronus@cronus.com."
 - d. Close the **Company Information** page.

Exercise 3: Create the Codeunit

Exercise Scenario

Create a codeunit that uses the SMTP Mail codeunit to send out email messages to seminar registrants. Let the user specify the contents of the email and the recipient. Add the new email function to an action menu for easy access.

Task 1: Create the Codeunit

High Level Steps

1. Create a new codeunit to handle seminar email confirmations.
2. Declare global variables.
3. Declare global text constants.
4. Add a local function that initializes the confirmation sending process by reading relevant setup records. This guarantees that required fields contain information initializing global variables, and that the field reads the confirmation email template from the configured external file.
5. Add a local function that creates and sends an email to the recipient with the subject and body passed as parameters. The function must track the number of failures during the send process, and must return the success state of the send operation.
6. Add a function that initializes the send process. For each registration line in the registration header passed as a parameter the function sends the email confirmation to the email address of the contact that is specified in the line. If the send process succeeds, the function updates the **Confirmation Date** field in the registration line. At the conclusion, it must display a message that states that the operation succeeded, or show the number of failures with the last error message.

Detailed Steps

1. Create a new codeunit to handle seminar email confirmations.
 - a. In Object Designer, create a new codeunit.
 - b. Save the codeunit as 123456705, SeminarMailManagement.

2. Declare global variables.
 - a. In the **C/AL Globals** window, create the following global variables.

Name	Data Type	Subtype
SeminarSetup	Record	Seminar Setup
CompanyInfo	Record	Company Information
SMTP	Codeunit	SMTP Mail
Environment	DotNet	System.Environment
NoOfErrors	Integer	
ConfirmationBody	Text	



Note: You can find the *System.Environment* class in the *mscorlib* assembly.

3. Declare global text constants.
 - a. In the **C/AL Globals** window, create the following text constants.

Name	ConstValue
Text001	Registration Confirmation for Seminar "%1"
Text002	Sending email confirmations was not successful. The last error message was:\\%1
Text003	All email confirmations were sent successfully.

4. Add a local function that initializes the confirmation sending process by reading relevant setup records. This guarantees that required fields contain information initializing global variables, and that the field reads the confirmation email template from the configured external file.
 - a. In the **C/AL Globals** window, create a new function and name it **Initialize**.
 - b. Define the following local variables.

Name	Data Type
Template	File
Line	Text

- c. In the **Initialize** function trigger, enter the following code.

```

SeminarSetup.GET;

SeminarSetup.TESTFIELD("Confirmation Template File");

CompanyInfo.GET;

CompanyInfo.TESTFIELD(Name);

CompanyInfo.TESTFIELD("E-Mail");

CLEAR(SMTP);

NoOfErrors := 0;

ConfirmationBody := "";

Template.TEXTMODE(TRUE);

Template.WRITEMODE(FALSE);

Template.OPEN(SeminarSetup."Confirmation Template File");

WHILE Template.POS < Template.LEN DO BEGIN

    Template.READ(Line);

    ConfirmationBody += Line + Environment.NewLine;

END;

Template.CLOSE;
    
```

5. Add a local function that creates and sends an email to the recipient with the subject and body passed as parameters. The function must track the number of failures during the send process, and must return the success state of the send operation.
 - a. In the **C/AL Globals** window, create a new function and name it **SendMail**.
 - b. Configure the function to receive the following parameters.

Var	Name	Data Type
No	ToEmail	Text
No	Subject	Text
No	Body	Text

- c. Configure the function to return a Boolean value.
- d. Configure a local variable of name CompanyInfo, type Record, and subtype Company Information.

e. In the **SendMail** function trigger, enter the following code.

```
SMTP.CreateMessage(
    CompanyInfo.Name,
    CompanyInfo."E-Mail",
    ToEmail,
    Subject,
    Body,
    FALSE);

IF NOT SMTP.TrySend THEN BEGIN
    NoOfErrors := NoOfErrors + 1;
    EXIT(FALSE);
END ELSE
    EXIT(TRUE);
```

6. Add a function that initializes the send process. For each registration line in the registration header passed as a parameter the function sends the email confirmation to the email address of the contact that is specified in the line. If the send process succeeds, the function updates the **Confirmation Date** field in the registration line. At the conclusion, it must display a message that states that the operation succeeded, or show the number of failures with the last error message.

- a. In **C/AL Globals** window, create a new function and name it **SendConfirmations**.
- b. Configure the function to receive a parameter of name SemRegHeader, type Record, and subtype Seminar Registration Header.
- c. Define the following local variables for the function.

Name	Data Type	Subtype
SemRegLine	Record	Seminar Registration Line
Contact	Record	Contact

d. In the function trigger for the **SendConfirmations** function, enter the following code.

```

Initialize;

SemRegLine.SETRANGE("Document No.",SemRegHeader."No.");

IF SemRegLine.FINDSET(TRUE) THEN

    REPEAT

        Contact.GET(SemRegLine."Participant Contact No.");

        Contact.TESTFIELD("E-Mail");

        IF SendMail(

            Contact."E-Mail",

            STRSUBSTNO(

                Text001,

                SemRegHeader."Seminar Name"),

            STRSUBSTNO(

                ConfirmationBody,

                Contact.Name,

                SemRegHeader."Seminar Name",

                SemRegHeader."Starting Date"))

        THEN BEGIN

            SemRegLine."Confirmation Date" := TODAY;

            SemRegLine.MODIFY;

        END;

    UNTIL SemRegLine.NEXT = 0;

IF NoOfErrors > 0 THEN

    ERROR(Text002,SMTP.GetLastSendMailErrorText);

MESSAGE(Text003);

```

- e. Compile, save, and then close the codeunit.

Task 2: Customize the Pages

High Level Steps

1. Add a function to the **Seminar Registration** page to send confirmations to all seminar participants.
2. Add a function to the **Seminar Registration List** page to send confirmations to all participants of the seminar.

Detailed Steps

1. Add a function to the **Seminar Registration** page to send confirmations to all seminar participants.
 - a. Design page 123456710, **Seminar Registration**.
 - b. Add a global variable named SemMailMgt for the codeunit 123456705, SeminarMailManagement.
 - c. Add an action to the ActionItems container, and configure the following properties.

Property	Value
Caption	Send Registration Co&nfirmlations
Image	SendConfirmation
Promoted	Yes
PromotedCategory	Process

- d. In the OnAction trigger for the Send Registration Confirmations action, enter the following code.

```
SemMailMgt.SendConfirmations(Rec);
```

2. Add a function to the **Seminar Registration List** page to send confirmations to all participants of the seminar.
 - a. Design page 123456713, **Seminar Registration List**.
 - b. Repeat the same customizations that you made for the page 123456710, **Seminar Registration**.

Module Review

Module Review and Takeaways

This module demonstrates how to use the built-in capabilities of Microsoft Dynamics NAV 2013 to interface with external applications. It listed the most common interfacing capabilities. This includes COM technologies, Microsoft .NET Interoperability, and data exchange through files.

Microsoft Dynamics NAV 2013 supports automation and OCX controls. However the support is limited to client-side objects and those objects that do not provide any user interface. Server-side COM is not supported, because not all COM components support 64-bit architecture.

A better alternative to COM is Microsoft .NET Interoperability which lets you run .NET assemblies on the server side. You can also use Microsoft .NET Framework to extend the capabilities of Microsoft Dynamics NAV 2013 client for Windows.

Microsoft Dynamics NAV 2013 does not support COM or .NET Interoperability under the web client or Microsoft Dynamics NAV Portal Framework.

Test Your Knowledge

Test your knowledge with the following questions.

- 1. Which to data types allow you to access COM objects from C/AL?

- 2. COM objects can only be instantiated and accessed on the server side.

() True

() False

- 3. What C/AL function is used to create an instance of an automation server class?

C/SIDE Solution Development in Microsoft Dynamics® NAV 2013

4. Microsoft Dynamics NAV 2013 can read and write data in text or binary files.

() True

() False

5. Can you use the same File variable to access data in multiple files?

Test Your Knowledge Solutions

Module Review and Takeaways

1. Which two data types allow you to access COM objects from C/AL?

MODEL ANSWER:

OCX and Automation.

2. COM objects can only be instantiated and accessed on the server side.

True

False

3. What C/AL function is used to create an instance of an automation server class?

MODEL ANSWER:

CREATE

4. Microsoft Dynamics NAV 2013 can read and write data in text or binary files.

True

False

5. Can you use the same File variable to access data in multiple files?

MODEL ANSWER:

Yes, but you can only access one file at a time. You must call CLOSE on the first file, before you OPEN or CREATE the second file, and so on.

