Chapter No. 10
"Integration"
In this package, you will find:
A Biography of the author of the book
A preview chapter from the book, Chapter NO.10 "Integration"
A synopsis of the book’s content
Information on where to buy this book

About the Author

Rakesh Raul is from a small town in India, with a vision of doing something big in programming. He completed his first diploma in programming at the age of 16, and continued higher studies in computer software development.

He started his programming career with a small software development company in Mumbai. After 2 years of development in Visual Basic, he was introduced to Microsoft Dynamics NAV Version 3. For the first 2-3 years he worked as a Microsoft Dynamics NAV developer and at the same time he learned all the areas of the product and earned his first Microsoft Certification-Business Solutions Professional. He continues to stay updated with new releases of the product and is certified in multiple areas for versions 4.0, 5.0, 2009, and 2013. Apart from Microsoft Dynamics NAV, he also has good knowledge of Microsoft SQL Server and Business Intelligence.

His seven-year journey with Microsoft Dynamics NAV includes more than 30 implementations; one horizontal and two vertical solution designs and development.

Currently, he works in Tectura, India, as a Senior Technical Consultant. Tectura is a worldwide provider of business consulting services delivering innovative solutions.

For More Information:
www.packtpub.com/microsoft-dynamics-nav-7-programming-cookbook/book
I would like to thank my wife, Ashwini, for supporting and always standing by my side in good and bad days.

I would like to take this opportunity to thank all the mentors I was blessed with, who unconditionally shared their knowledge and inspired me.

Mibuso and all Microsoft Dynamics NAV related blogs are a great boon for all NAV consultants. I would like to thank all the contributors of these great sites.

Love you Aabha, my cute little princess!

For More Information:  
www.packtpub.com/microsoft-dynamics-nav-7-programming-cookbook/book
Microsoft Dynamics NAV 7 Programming Cookbook

Microsoft Dynamics NAV 7 is a product of the Microsoft Dynamics family. It's a business management solution that helps simplify and streamline business processes, such as finance, manufacturing, customer relationship management, supply chains, analytics, and electronic commerce for small and medium-sized enterprises. Microsoft Dynamics partners can have full access to the source code, which is very easy to customize. Learning NAV programming in NAV 7 will give a full inside view of the ERP system and open doors to many other exciting areas.

The Microsoft Dynamics NAV 7 Programming Cookbook will take you through interesting topics that span a wide range of areas, for example, integrating the NAV system with other software applications, such as Microsoft Office and creating reports to present information from multiple areas of the system. You will not only learn the basics of NAV programming, but you will also be exposed to the technologies that surround the NAV system, such as .NET programming, SQL Server, and NAV system administration.

The first half of the cookbook will help programmers using NAV for the first time by walking them through the building blocks of writing code and creating objects, such as tables, pages, and reports.

The second half focuses on using the technologies surrounding NAV to build better solutions and administration of the NAV service tier. You will learn how to write .NET code that works with the NAV system and how to integrate the system with other software applications, such as Microsoft Office or even custom programs.

What This Book Covers

Chapter 1, String, Dates, and Other Data Types, describes the method of working with the most common data types. You will learn how to use the functions related to data types. Every recipe includes actual NAV code with a brief explanation about code that will make the data type learning process very interesting.

Chapter 2, General Development, covers the C/AL development structure that includes loops, conditional statements, functions, and so on. You will find some recipes describing C/AL specific commands and functions.

Chapter 3, Working with Tables, Records, and Queries, focuses on the database structure and data retrieval. You will learn how to design a table using filters to retrieve specific data. This chapter will also discuss new object type Query.

For More Information:
www.packtpub.com/microsoft-dynamics-nav-7-programming-cookbook/book
Chapter 4, Designing Pages, focuses on data presentation using pages. You will learn how to develop different types of pages including Role Center, Queue, wizard, and many more.

Chapter 5, Report Design, explains how to design an RDLC report. You will find recipes describing the process of adding a request page, setting filters, linking two reports and many more interesting topics related to reports.

Chapter 6, Diagnosing Code Problems, explains how to use built-in tools to debug code problems. You will also learn about debugging the NAV application server.

Chapter 7, Roles and Security, focuses on NAV user security, which includes creating roles and assigning permissions to a role. It will also explain about security filters and filter groups.

Chapter 8, Leveraging Microsoft Office, describes different methods to integrate with the Microsoft Office suite, which includes Word, Excel, InfoPath, and Visio.

Chapter 9, OS Interaction, focuses on different ways to integrate with the Windows operating systems. You will learn how to search the filesystem as well as how to query the system registry.

Chapter 10, Integration, describes different ways of integrating NAV with other applications. You will learn how to exchange data using flat file and XMLport. You will find a recipe describing how to use ADO to access data stored in other databases.

Chapter 11, Working with the SQL Server, provides an introduction to the SQL Server environment. You will learn about writing queries, configuring automated backups, and maintaining SQL logfiles. There is a recipe that will help you to understand the Sum Index Field Technology.

Chapter 12, NAV Server Administration, will help you to learn and understand the NAV service tier. It will also explain about creating a user and maintaining a NAV license.

For More Information:
www.packtpub.com/microsoft-dynamics-nav-7-programming-cookbook/book
In this chapter, we will cover the following recipes:

- Sharing information through XMLports
- Writing to and reading from a file using the C/AL code
- Creating web services
- Consuming web services
- Sending data through FTP
- Printing a report in a PDF, Excel, and Word format
- Writing your own automation using C#
- Using ADO to access outside data

Introduction

Business depends on multiple applications. Until now, all these applications were hosted in-house, on the company's own server, which integrated applications by exchanging a flat file or directly accessing the database. In the last few years, technologies have taken a big leap and introduced numerous ways of managing business applications; cloud computing is one of them.

Microsoft has made sure that Dynamics NAV will continue to meet its customer's integration needs for this new type of infrastructure. In this chapter, we will be taking a look at the different ways of integration with Microsoft Dynamics NAV.

This chapter will show you how to share information using XML or flat text files, creating and consuming web services, and loading files on an FTP server. These recipes will serve as a foundation for all your future integration efforts.

For More Information:
www.packtpub.com/microsoft-dynamics-nav-7-programming-cookbook/book
Sharing information through XMLports

Exporting or importing data is a common requirement for financial or ERP applications. There are a number of formats in which data can be asked, but when there is no manual intervention with data, or that data has to be used by two different applications, then it is mostly asked in a delimited, fixed length or XML format. **Extensible Markup Language (XML)** is a format for creating structured computer documents. XMLports are object types in Microsoft Dynamics NAV that help to create these types of documents.

**How to do it...**

1. Let's get started by creating a new XMLport from **Object Designer**.
2. Add the following variables to XMLport designer:

<table>
<thead>
<tr>
<th>Node Name</th>
<th>Node Type</th>
<th>Source Type</th>
<th>Data Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Root</td>
<td>Element</td>
<td>Text</td>
<td>&lt;Root&gt;</td>
</tr>
<tr>
<td>Customer</td>
<td>Element</td>
<td>Table</td>
<td>&lt;Customer&gt;(Customer)</td>
</tr>
<tr>
<td>No</td>
<td>Element</td>
<td>Field</td>
<td>&lt;Customer&gt;::No.</td>
</tr>
<tr>
<td>Name</td>
<td>Element</td>
<td>Field</td>
<td>&lt;Customer&gt;::Name</td>
</tr>
</tbody>
</table>

3. XMLport designer should look similar to the following screenshot:

4. Now save and close the XMLport.

For More Information:  
www.packtpub.com/microsoft-dynamics-nav-7-programming-cookbook/book
How it works...

XMLports are similar to discontinued NAV integration object type dataports. Developing XMLports is a bit different from developing dataports. The following screenshot displays a portion of an output file, which will help in understanding the XMLport structure:

```
<?xml version="1.0" encoding="UTF-16"?>
  <Root>
  - <Customer>
    <No>01121212</No>
    <Name>Spotsmeyer's Furnishings</Name>
  </Customer>
  - <Customer>
    <No>01445544</No>
    <Name>Progressive Home Furnishings</Name>
  </Customer>
  - <Customer>
    <No>01454545</No>
    <Name>New Concepts Furniture</Name>
  </Customer>
  - <Customer>
    <No>01905893</No>
    <Name>Candoxy Canada Inc.</Name>
  </Customer>
  - <Customer>
    <No>01905899</No>
    <Name>Elkhorn Airport</Name>
  </Customer>
  - <Customer>
    <No>01905902</No>
    <Name>London Candoxy Storage Campus</Name>
  </Customer>
```

XML is a tree-like structure made up of nodes. Every file has to start with a parent/root node. Under the root node, we define a table from which we want to read the data, and finally we define fields we want to use from that table. In this recipe, we have used a customer table.

In our output file, we can see that each value is surrounded by a node with the name of the field, and a set of fields is surrounded by a parent node, which is just our table named Customer.

There's more...

XMLports are not only made to export/import XML files, but we can even work with text files of a fixed length and delimited formats. To achieve this, we have to change the Format property of XMLport to Fixed Text or Variable Text. Along with this, we have the FieldStartDelimiter, FieldEndDelimiter, and FieldSeparator properties that help to read multiple file formats.

For More Information:
www.packtpub.com/microsoft-dynamics-nav-7-programming-cookbook/book
Exporting Sales Invoices in the CSV format

In the previous recipe, we took the first step by creating a very basic XMLport to export data from a single table. Now let's take another step. Here we will export the sales data, but this time it will be in the CSV (Comma Separated Values) format.

1. Let's get started by creating a new XMLport from Object Designer.
2. Go to the XMLport's properties from View | Properties (Shift + F4).
3. Set the properties mentioned in the following table:

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direction</td>
<td>Export</td>
</tr>
<tr>
<td>Format</td>
<td>Variable Text</td>
</tr>
<tr>
<td>FieldDelimiter</td>
<td>&lt;None&gt;</td>
</tr>
<tr>
<td>Table Separator</td>
<td>&lt;NewLine&gt;</td>
</tr>
</tbody>
</table>

4. Add the following variables to the XMLport designer:

<table>
<thead>
<tr>
<th>Node Name</th>
<th>Node Type</th>
<th>Source Type</th>
<th>Data Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Root</td>
<td>Element</td>
<td>Text</td>
<td>&lt;Root&gt;</td>
</tr>
<tr>
<td>PurchaseHeader</td>
<td>Element</td>
<td>Table</td>
<td>&lt;Purchase Header&gt;(Purchase Header)</td>
</tr>
<tr>
<td>PH_DocType</td>
<td>Element</td>
<td>Field</td>
<td>Purchase Header::Document Type</td>
</tr>
<tr>
<td>PH_No</td>
<td>Element</td>
<td>Field</td>
<td>Purchase Header::No.</td>
</tr>
<tr>
<td>PH_Vendor</td>
<td>Element</td>
<td>Field</td>
<td>Purchase Header::Buy-from</td>
</tr>
<tr>
<td>PH_OrderDate</td>
<td>Element</td>
<td>Field</td>
<td>Purchase Header::Order Date</td>
</tr>
<tr>
<td>PH PostingDate</td>
<td>Element</td>
<td>Field</td>
<td>Purchase Header::Posting Date</td>
</tr>
<tr>
<td>PurchaseLine</td>
<td>Element</td>
<td>Table</td>
<td>&lt;Purchase Line&gt;(Purchase Line)</td>
</tr>
<tr>
<td>PL_DocType</td>
<td>Element</td>
<td>Field</td>
<td>Purchase Line::Document Type</td>
</tr>
<tr>
<td>PL_DocNo</td>
<td>Element</td>
<td>Field</td>
<td>Purchase Line::Document No.</td>
</tr>
<tr>
<td>PL_LineNo</td>
<td>Element</td>
<td>Field</td>
<td>Purchase Line::Line No.</td>
</tr>
<tr>
<td>PL_Type</td>
<td>Element</td>
<td>Field</td>
<td>Purchase Line::Type</td>
</tr>
<tr>
<td>PL_No</td>
<td>Element</td>
<td>Field</td>
<td>Purchase Line::No.</td>
</tr>
<tr>
<td>PL UOM</td>
<td>Element</td>
<td>Field</td>
<td>Purchase Line::Unit of Measure</td>
</tr>
<tr>
<td>PL Quantity</td>
<td>Element</td>
<td>Field</td>
<td>Purchase Line::Quantity</td>
</tr>
<tr>
<td>PL Line Amt</td>
<td>Element</td>
<td>Field</td>
<td>Purchase Line::Line Amount</td>
</tr>
</tbody>
</table>

For More Information:
www.packtpub.com/microsoft-dynamics-nav-7-programming-cookbook/book
5. After indenting all nodes, XMLport designer should look similar to the following screenshot:

![XMLport Design Screen](image)

6. Now set the following property for the **PurchaseHeader** node:

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SourceTableView</td>
<td>SORTING(Document Type, No.)</td>
</tr>
</tbody>
</table>

7. Set the following property for the **PurchaseLine** node:

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SourceTableView</td>
<td>SORTING(Document Type, Document No., Line No.)</td>
</tr>
</tbody>
</table>

8. Now save and close XMLport.

On execution of the preceding recipe, the system will show a dialog box to save the output file.

While setting a value of the XMLport's **TableSeparator** property, we keep one space before \(<\text{NewLine}>\). This change will not help in exporting data, but if we use the same dataport to import the file, the system needs to differentiate between a record and a table separator, and this time our setting will help.

---

For More Information:
www.packtpub.com/microsoft-dynamics-nav-7-programming-cookbook/book
See also
- The Browsing for a file recipe in Chapter 9, OS Interaction
- The Checking file and folder access permissions recipe in Chapter 9, OS Interaction
- Sending data through FTP

Writing to and reading from a file using the C/AL code

Even though the XMLport takes care of the file integration requirements, sometimes we may want to perform this activity by using the C/AL code. This recipe will demonstrate how to read or write from a file using the C/AL code.

How to do it...

1. Let's start by creating a new codeunit from **Object Designer**.
2. Add the following global variables:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>StremOut</td>
<td>OutStrem</td>
<td></td>
</tr>
<tr>
<td>FileOut</td>
<td>File</td>
<td></td>
</tr>
<tr>
<td>StremIn</td>
<td>InStrem</td>
<td></td>
</tr>
<tr>
<td>FileIn</td>
<td>File</td>
<td></td>
</tr>
<tr>
<td>TextLine</td>
<td>Text</td>
<td>250</td>
</tr>
</tbody>
</table>

3. Add the following code in the `OnRun` trigger:

```c
IF NOT FileOut.CREATE('D:\NAVFile.txt') THEN
  IF NOT FileOut.OPEN('D:\NAVFile.txt') THEN
    ERROR('Unable to write to file!');
  FileOut.CREATEOUTSTREAM(StreamOut);
  StreamOut.WRITETEXT('Line 1');
  StreamOut.WRITETEXT();
  StreamOut.WRITETEXT('Line 2');
  StreamOut.WRITETEXT();
  FileOut.CLOSE;
ELSE
  IF NOT FileIn.OPEN('D:\NAVFile.txt') THEN
    ERROR('Unable to read file!');

For More Information:
www.packtpub.com/microsoft-dynamics-nav-7-programming-cookbook/book
FileIn.CREATEINSTREAM(StreamIn);
WHILE NOT StreamIn.EOS DO BEGIN
    StreamIn.READTEXT(TextLine);
    MESSAGE('%1', TextLine);
END;
FileIn.CLOSE;

4. Save and close the codeunit.

How it works...

In this recipe, we are first creating a new file using the CREATE function. If the system fails to create that file, we consider that there may be a file present with the same name in that location. Then we try to open that file; if we fail in this attempt as well, we generate an error message as we do not have any file to work with.

As we are writing data to a file, we have to use OutStream. Actually, the activity of sending data to a file is done by the stream object's WRITETEXT function. This function does not send a carriage return; that's why we are using the WRITETEXT function with a blank parameter. After we finish writing to the file, we close the file.

The process of reading from a file is very similar to writing. Instead of using an OutStream variable, we use an InStream variable. It has the EOS (End of Stream) function. The EOS function returns the True value when we reach the end of the file. Until we reach the end of the file, we can retrieve data using the READTEXT function. The parameter of the READTEXT function is of the text datatype, which stores the line of text. In our code, we use the MESSAGE function to display the line read by our code.

See also

- Sharing information through XMLports

Creating web services

The web services allow sharing of an application's functionalities to an external system and its users. It also takes proper authorization before sharing any information. In Microsoft Dynamics NAV, creating a web service is an easy task; we can expose pages, codeunits, and queries as web services.

For More Information:
www.packtpub.com/microsoft-dynamics-nav-7-programming-cookbook/book
How to do it...

1. Start Microsoft Dynamics NAV RoleTailored client and follow the steps.
2. Either search for Web Service in NAV's search, or from the Department menu, visit the following path:
3. CRONUS International Ltd. | Departments | Administration | IT Administration | General | Web Services.
4. Create a new web service using the following record:

<table>
<thead>
<tr>
<th>Object Type</th>
<th>Object ID</th>
<th>Service Name</th>
<th>Published</th>
</tr>
</thead>
<tbody>
<tr>
<td>Page</td>
<td>22</td>
<td>Customer List</td>
<td>Yes</td>
</tr>
</tbody>
</table>
5. Close the page.

How it works...

We can publish two types of web services, SOAP and OData. The SOAP web service provides flexibility for building an operation-centric service. We can publish a page or a codeunit as SOAP services. The OData web service is designed for querying tabular data. We can publish a page or a query object as OData services. The SOAP services allow us to create, read, update, and delete operations using the Page object, whereas OData services only support the read-only operations.

Creating a web service requires us to expose a page, codeunit, or query object type, provide a service name, and check the published field. At this time, the system does not have any idea about the service type. The system chooses the service type when we execute or use it. NAV provides the configuration of web services from the Microsoft Dynamics NAV administrator console. Here we specify the ports for both the service types, and we also have an option to enable or disable the service.

For More Information:
www.packtpub.com/microsoft-dynamics-nav-7-programming-cookbook/book
To verify the web service, start Windows Internet Explorer and provide an address in the following format:

http://<Server>:<WebServicePort>/<ServerInstance>/WS/<CompanyName>/services

Our service type depends on the value mentioned for WebservicePort in the previous address format.

See also

- The Creating an InfoPath form for the NAV data recipe in Chapter 8, Leveraging Microsoft Office
- Consuming web services

**Consuming web services**

Microsoft Dynamics NAV provides an easy interface to create web services, which allow us to expose the NAV data with business logic and proper authentication. Now, let's see how to use these web services.

**How to do it...**

1. Let's get started by creating a new codeunit from **Object Designer**.
2. Add the function name as **GetCustomer**.
3. The function should take the following parameter:

<table>
<thead>
<tr>
<th>Name</th>
<th>DataType</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>CustNo</td>
<td>Code</td>
<td>20</td>
</tr>
</tbody>
</table>

4. Add the following local variable to the function:

<table>
<thead>
<tr>
<th>Name</th>
<th>DataType</th>
<th>Subtype</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customer</td>
<td>Record</td>
<td>Customer</td>
</tr>
</tbody>
</table>

5. The function should return a text variable of length 50.
6. Add the following code to the function:

   ```plaintext
   IF Customer.GET(CustNo) THEN
      EXIT(Customer.Name)
   ELSE
      EXIT('Not Found!');
   ```
7. Save and close the codeunit.

8. Search for the Web Services page in RoleTailored client.

9. In the Web Services page, choose New.

10. Create a new web service with our codeunit ID, and in the service name, enter ConsumeWS.

11. Finally, mark the checkbox in the Published column.

12. Create a new Console Application project in Visual Studio.

13. In Solution Explorer, right-click on the Reference node and choose Add Service Reference.

14. In the Add Service Reference window, choose the Advance button.

15. In the Service Reference Settings window, choose Add Web Reference. Then enter http://localhost:7047/DynamicsNAV70/WS/Services (this may be different depending on the web server, service name, and NAV company name) and click on the green button with an arrow.

16. When the ConsumeWS service is displayed, choose View Service. Then enter WebService in Web reference name: and choose Add Reference.

For More Information:
www.packtpub.com/microsoft-dynamics-nav-7-programming-cookbook/book
17. Add the following code to the program:

```csharp
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace ConsumeWebService
{
    using WebService;
    public class ConsumeWebService
    {
        public static void Main(string[] args)
        {
            ConsumeWS ws = new ConsumeWS();
            ws.UseDefaultCredentials = true;
            Console.WriteLine(ws.GetCustomer("10000"));
            Console.ReadLine();
        }
    }
}
```

18. Compile, save, and close the program.

**How it works...**

In the previous recipe, we had created a codeunit that returns the name of a customer if executed successfully, else it returns a text saying Not Found. Then we published this codeunit as a web service to make it available for external applications.

In the .NET program, we need to provide the right reference of our web service, otherwise we will not be able to build our application with the code provided in the previous recipe.

To use our web service in the .NET program, we have provided its reference. The `using WebService` line tells the program to use the functions from our web service. Then we created an instance of our service `ConsumeWS` and used the default credentials. Now we can call the functions of our page or of the codeunit. As we have created the `GetCustomer` function in our codeunit, we are using that function for finding Customer No. 10000.

**See also**

- The Creating an InfoPath form for the NAV data recipe in Chapter 8, Leveraging Microsoft Office
- Creating web services

**For More Information:**

Sending data through FTP

Sometimes, our client may ask us to upload a data file on the FTP server. We can use the Windows built-in client to develop our FTP upload program.

Getting ready

Make sure we have an active FTP server and logon credentials.

How to do it...

1. Let’s start by creating a new codeunit from Object Designer.
2. Add a function name FTP that takes in the following parameters:

<table>
<thead>
<tr>
<th>Name</th>
<th>DataType</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>UserName</td>
<td>Text</td>
<td>50</td>
</tr>
<tr>
<td>Password</td>
<td>Text</td>
<td>50</td>
</tr>
<tr>
<td>ServerName</td>
<td>Text</td>
<td>50</td>
</tr>
<tr>
<td>FileToMove</td>
<td>Text</td>
<td>250</td>
</tr>
</tbody>
</table>

3. Then add the following local variables to the function:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>BatchFileName</td>
<td>Text</td>
<td>250</td>
</tr>
<tr>
<td>BatchFile</td>
<td>File</td>
<td></td>
</tr>
<tr>
<td>BatchFileStream</td>
<td>OutStream</td>
<td></td>
</tr>
<tr>
<td>BatchFileData</td>
<td>Text</td>
<td>250</td>
</tr>
</tbody>
</table>

4. Now add the following code to the function:

```pascal
BatchFileData := 'D:\Temp\navFTP.dat';
BatchFileName := 'D:\Temp\navFTP.bat';
BatchFile.CREATE(BatchFileName);
BatchFile.CREATEOUTSTREAM(BatchFileStream);
BatchFileStream.WRITETEXT('@echo off');
BatchFileStream.WRITETEXT('echo user ' + UserName + ' >> ' + BatchFileData);
BatchFileStream.WRITETEXT('echo ' + Password + ' >> ' + BatchFileData);
BatchFileStream.WRITETEXT;
```

For More Information:

www.packtpub.com/microsoft-dynamics-nav-7-programming-cookbook/book
BatchfileStream.WRITETEXT('echo bin >> ' +BatchFileData);
BatchfileStream.WRITETEXT;
BatchfileStream.WRITETEXT('echo put ' +FileToMove + ' >> ' + BatchFileData);
BatchfileStream.WRITETEXT;
BatchfileStream.WRITETEXT('echo quit >> ' +BatchFileData);
BatchfileStream.WRITETEXT;
BatchfileStream.WRITETEXT('FTP -n -s:' +BatchFileData + ' ' + ServerName);
BatchfileStream.WRITETEXT;
BatchfileStream.WRITETEXT('del ' + BatchFileData);
BatchFile.CLOSE;
CREATE(WshShell,FALSE,TRUE);
WshShell.Run(BatchFileName);

5. Write the following code in the OnRun trigger of the codeunit:
FTP('YourUserName', 'YourPassword', 'YourServer', 'YourFile');

6. Save and close the codeunit.

**How it works...**

File Transfer Protocol (FTP) is a way of sending data from one filesystem to another. Windows provides a command-line utility to upload a file on FTP. Even though it is very basic, it works well for our integration requirement.

We have created two files: a batch file and a data file. A batch file instructs the FTP program and transfers the data file. Let’s go through every line.

To enhance the security, we have added the first line, @echo off. It will not display any command of our program on the screen. On the next two lines, we have applied the same principle to secure our username and password. We are instructing the batch file for sending text to an actual file, hence we are adding >>BatchFileData at the end of all the lines. Next, we are setting the transfer type as binary and sending the file.

**There's more...**

For a list of available options or parameters that can be used with the FTP program, type ftp ? in the command prompt.

For More Information:
www.packtpub.com/microsoft-dynamics-nav-7-programming-cookbook/book
Integration

With the default settings of NAV, you may get the following error on execution of this recipe:

You are receiving this error because of Navision's service configuration. Open CustomSettings.Config from your server instance; the default path for this is C:\Program Files\Microsoft Dynamics NAV\70\Service\Your Instance name. You need to remove the file extension "bat" value from the default value of the key ClientServicesProhibitedFileTypes.

See also

- Sharing information through XMLports

**Printing a report in a PDF, Excel, and Word format**

Sharing recorded information is very important in our day-to-day activities; having the same format for the shared information is very important. If clients want to send a purchase order to a vendor, they will prefer to do so in a PDF format, or if the management wants to do some further analysis on a report's data, it will prefer Excel. Let's see how to do this.

**How to do it...**

1. Let's create a new codeunit from **Object Designer**.
2. Add the following global variables:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Subtype</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>FileName</td>
<td>Text</td>
<td></td>
<td>250</td>
</tr>
<tr>
<td>Customer</td>
<td>Rec</td>
<td>Customer</td>
<td></td>
</tr>
</tbody>
</table>

3. Add the following code in the **OnRun** trigger:

   ```
   Customer.setrange(City,' London');
   //Export to PDF
   FileName := 'C:\NAVReports\CustomerList.pdf';
   ```

For More Information:

www.packtpub.com/microsoft-dynamics-nav-7-programming-cookbook/book
REPORT.SAVEASPDF(101, FileName,Customer);
//Export to Excel
FileName := 'C:\NAVReports\CustomerList.xls';
REPORT.SAVEASPDF(101, FileName,Customer);
//Export to Word
FileName := 'C:\NAVReports\CustomerList.doc';
REPORT.SAVEASPDF(101, FileName,Customer);

4. Save and close the codeunit.

**How it works...**

Saving the report in a PDF, Excel, or Word format is a very simple activity. NAV provides a built-in function for each file type, which takes three parameters. The first parameter is a report object. The second parameter is the name and location of the file. The last parameter is an optional parameter; it is used for filtering a record set on which the report is getting generated. In our example, we have taken the report 101 (Customer-List) and we have applied a filter on the city London.

**See also**

- The Browsing for a file recipe in Chapter 9, OS Interaction

**Writing your own automation using C#**

C/AL provides almost everything to meet our client's requirements. Sometimes though, we may need to extend the scope of C/AL to take care of some complex requirements. In this recipe, we will see an example of how to develop a basic .NET application, and more importantly, how to use it within NAV.

**How to do it...**

1. Let's get started by creating a new Class Library project in Visual Studio and follow the steps.
2. Create a new file with the following code:

```csharp
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Runtime.InteropServices;
namespace NAVAdd
{
```

**For More Information:**

Integration

```csharp
[ClassInterface(ClassInterfaceType.AutoDual)]
[ProgId("NAVAdd")]
[ComVisible(true)]
public class NAVAdd
{
    public int Add(int a, int b)
    {
        return a + b;
    }
}
```

3. View the properties for the project.
4. Then on the **Application** tab, set the **Assembly** name to **NAVAdd**.
5. After that, on the **Build** tab, set the **Register for COM interop** property to true (checked):

![Register for COM interop](image)

6. Save and compile your objects.
7. Create a new codeunit from **Object Designer**.

For More Information:
www.packtpub.com/microsoft-dynamics-nav-7-programming-cookbook/book
8. Add the following global variable:

<table>
<thead>
<tr>
<th>Name</th>
<th>DataType</th>
<th>Subtype</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAVAdd</td>
<td>Automation</td>
<td>'NAVAdd'.NAVAdd</td>
</tr>
</tbody>
</table>

9. Add the following code in the OnRun trigger:

```plaintext
CREATE (NavAdd, FALSE, TRUE);
MESSAGE('%1', NavAdd.Add(2, 3));
```

10. Save and close the codeunit.

**How it works...**

In our Visual Studio program, we are setting the `ClassInterfaceType.AutoDual` value to call the `ClassInterface` attribute, which will register a program automatically. The second attribute `ProgID` is the name of our program. Finally, to instruct the system about class registration, we are using the last attribute, `COMVisible`.

Now, we set some properties of our program. To register our class as `Automation`, we need to select the `Register for COM interop` property. As soon as we compile this program, we can see that our `NAVAdd` is available in the `Automation` list.

**See also**

- The Querying the registry recipe in Chapter 9, OS Interaction

**Using ADO to access outside data**

**ActiveX Data Object (ADO)** is a set of COM objects used for accessing data sources. ADO allows developers to write programs to access data without knowing what a database structure is, or how the database is implemented. Let’s see how to use ADO in C/AL programing.

**How to do it...**

1. Let’s get started by creating a new codeunit from **Object Designer**.
2. Create a function named `CreateConnectionString`.

**For More Information:**

3. Add the following parameters to the function:

<table>
<thead>
<tr>
<th>Name</th>
<th>DataType</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>ServerName</td>
<td>Text</td>
<td>50</td>
</tr>
<tr>
<td>DatabaseName</td>
<td>Text</td>
<td>50</td>
</tr>
<tr>
<td>UserName</td>
<td>Text</td>
<td>50</td>
</tr>
<tr>
<td>Password</td>
<td>Text</td>
<td>50</td>
</tr>
</tbody>
</table>

4. Set the function's return value of type `text` with length `1024`.

5. Add the following code to the function:

```c
EXIT(
    'Driver={SQL Server};' + 'Server=' + ServerName + ';
    'Database=' + DatabaseName + ';
    'Uid=' + UserName + ';
    'Pwd=' + Password + ';');
```

6. Add the following global variables:

<table>
<thead>
<tr>
<th>Name</th>
<th>DataType</th>
<th>Subtype</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADOConnection</td>
<td>Automation</td>
<td>'Microsoft ActiveX Data Objects 6.0 Library'.Connection</td>
<td></td>
</tr>
<tr>
<td>ADORecordSet</td>
<td>Automation</td>
<td>'Microsoft ActiveX Data Objects 6.0 Library'.Recordset</td>
<td></td>
</tr>
<tr>
<td>SQLString</td>
<td>Text</td>
<td></td>
<td>250</td>
</tr>
</tbody>
</table>

7. Write the following code in the `OnRun` trigger of the codeunit:

```c
CREATE(ADOConnection,FALSE,TRUE);
ADOConnection.ConnectionString := CreateConnectionString('localhost', 'Book', 'Super', 'rrsaw0201');
ADOConnection.Open;
SQLString:= 'SELECT * FROM [CRONUS International Ltd_$Customer] WHERE [No_] = ''10000'';
CREATE(ADOCRecordSet,FALSE,TRUE);
ADOCRecordSet:=ADOConnection.Execute(SQLString);
ADOCRecordSet.MoveFirst;
REPEAT
  MESSAGE(FORMAT(ADOCRecordSet.Fields.Item('Name').Value));
  ADORecordSet.MoveNext;
UNTIL ADORecordSet.EOF;
ADOCRecordSet.Close;
ADOConnection.Close;
```

8. Save and close the codeunit.
How it works...

First of all, we are setting up the connection string, which carries the server, database, and logon information. Once we open the connection, we can send our query to the database. In this recipe, we are selecting the customer information from the Customer table with a filter for Customer No. 10000.

To view the query result, we open the record set. Even though we know there will be only one record, we loop through the record set, just to understand how to play with multiple records. For looping, we use the simple **REPEAT UNTIL** loop till the end of the record set. Finally, to read data from each field, we use the `Fields.Item(FieldName)` syntax. To send the cursor to the next record, we use the `MoveNext` function.
Where to buy this book


Free shipping to the US, UK, Europe and selected Asian countries. For more information, please read our shipping policy.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.