MODULE 9: XMLPORTS

Module Overview

XMLports are objects that you can use to export data from or import data into Microsoft Dynamics NAV 2013. XMLports can manipulate the data in XML format or text format. Additionally, they can read from and write to files, or can access data directly in memory by using streams. XML format enables information exchange between different computer systems, and is a preferred format for importing and exporting data, as most computer systems can easily handle it. When data needs to be exchanged with legacy systems, XMLports also provide the support for files in text format.

Importing and exporting data is useful when data is collected outside of Microsoft Dynamics NAV and must be incorporated into the database, or when the data must be distributed from the Microsoft Dynamics NAV database to an external location. Streaming is useful when you are manipulating data that is stored in BLOB (Binary Large Object) fields in Microsoft Dynamics NAV 2013 tables, or when you exchange data between Microsoft Dynamics NAV 2013 and .NET objects.

Objectives

The objectives are:

- Describe the fundamentals of an XMLport and its components.
- Review how to design XMLports.
- Explain the Request Page functionality.
- Describe the process of using XMLports from C/AL code.
- Create XMLports for export and import with XML format.
- Create XMLports for export and import with fixed and a variable text format.

XMLPort Fundamentals

XMLports are used to export and import data to and from external XML document files or text files. XML documents that are created by XMLports encapsulate the data in XML format. This helps streamline the exchange of information between different computer systems. You can use text files when exchanging information with legacy systems or with Microsoft Office Excel.

The following figure shows components of an XMLport and how they are related:

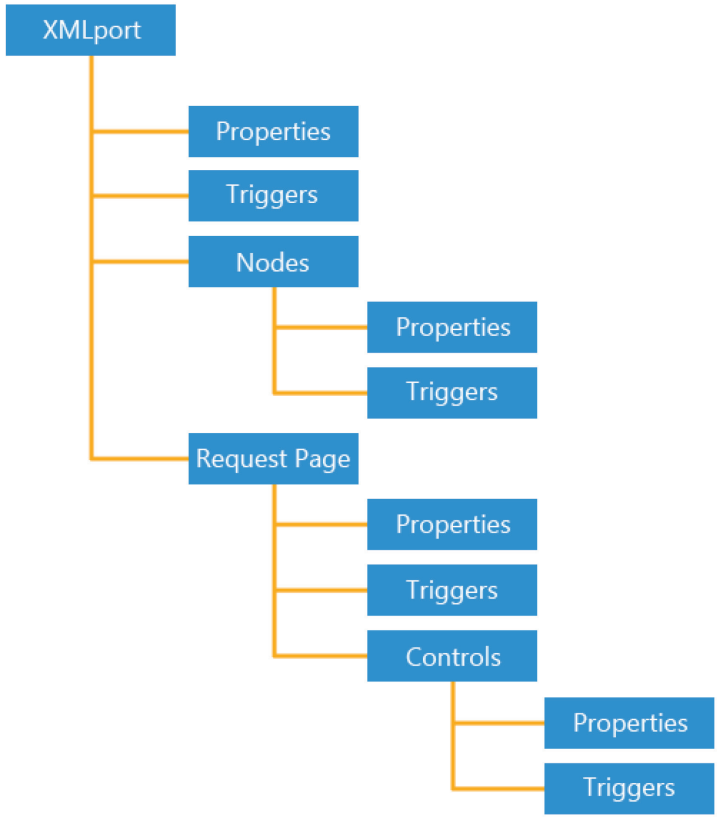

FIGURE 9.1: XMLPORT COMPONENTS

XMLports are created and designed in the **XMLport Designer**, which you can access from the **Object Designer**.

Properties

The properties of XMLport define the basic characteristics of the input or output files that XMLport shall access. The properties of the XMLport nodes describe the data source and define the behavior of the XMLport during import and export operations.

Triggers

Triggers are predefined functions that run when certain events occur. The bodies of these functions are first empty and can be defined by the developer. Writing C/AL code in triggers lets developers change the system's default behavior.

The following types of triggers are specific to XMLports:

- XMLport triggers
- Node triggers
- Request page triggers

Nodes

Nodes form the structure of XMLport and the structure of the XML document or text file that is to be imported from or exported to. Each node has several important properties that can be accessed directly from the **XMLport Designer**. The following table describes each node and its properties.

Node Property	Remarks
Node Name	Used to specify the XML node name of the XML element or attribute. Node names must be entered in the order in which they are presented in the XML document. Parent elements must come before their child elements. Indent the node names of child elements under their parent elements by using one indentation for each level. List attributes under the elements that they define and indent them to the child level.
Node Type	Used to specify whether the name in the Node Name represents data of the element or attribute types. The drop-down list in the **NodeType** field contains two options: Element and Attribute. The default setting is Element.
Source Type	Used to specify the data structure that the Node Name corresponds to. The **SourceType** field has a drop-down list that contains three options: Text, Table, and Field. The default setting is Text.
Data Source	Used together with the Source Type to specify the data source from the data structure.

The following table provides additional information about Source Type options.

Source Type Value	Remarks
Text	Select this option when the XML data cannot be mapped directly to the database or when the external file requires the information, which the database does not require or contain. The value of the **Text** field is put into a text variable that is specified in the **VariableName** property. If you do not specify the name, then the node name is used by default. You can change the variable type into a big text by setting the **TextType** property to BigText.
Table	Select this option to indicate that a node maps to a table. As with the Text option, specify a variable name for the table that also functions like a global record variable. By default, the variable name is the name of the table.
Field	Select this option to indicate that a node maps to a field in the table. However, for this selection to be valid, you must declare a table as the parent of the field. Failure to do this prevents the XMLport object from compiling.

 ***Note:*** *When you declare a node to be of the Text source type, and then specify the VariableName or Node Name property, the text variable is declared automatically. You do not have to explicitly declare such a variable in the* ***C/AL Globals*** *window. However, you can still use that variable in code or in references as if you did declare it explicitly through* ***C/AL Globals****.*

The Data Source has the following interactions with the Source Type:

- If you specify Table as the Source Type, clicking **AssistEdit** on the **DataSource** field opens the **Table List** window. Select a table from the **Table List** window or by setting the node's **SourceTable** property.
  - If a variable name is defined for the table, the format of the value that is shown in the **DataSource** field will be TableVariableName(tablename).
  - If a variable name is not defined, the format of the value is displayed as <tablename>(tablename).
- If Field is specified as the Source Type, clicking **AssistEdit** on the **DataSource** field opens the **Field Lookup** window. Select a field from the **Field Lookup** window or by setting the node's **Sourcefield** property. The format of the value is displayed as tablevariablename::fieldname.

Request Page

The request page is run before the actual XMLport begins running and is used to collect filters and options from the user.

Note: *The* ***OnInit*** *trigger runs before the client displays the request page. The* ***OnPreXMLport*** *trigger runs after the user clicks* ***OK****.*

Request pages in XMLports resemble request pages in reports. They have similar filter functionality grouped in FastTabs, and you can add more controls by creating other FastTabs.

The Request page in XMLport only has the **OK** and **Cancel** buttons (compared to **Print**, **Preview**, and **Cancel** buttons in the request page in reports), and a drop down box where you choose whether you want to use the XMLport for importing or exporting text or XML documents.

Design XMLports

Designing an XMLport is actually designing the structure of the external file, whether it is an XML document or a text file. This primarily involves setting various properties in the XMLport objects, such as XMLport properties and Node properties.

An XML document is a well-structured hierarchy of nodes which at the same time contain the data, and also describe the nature of the data in the document.

To create an XMLport to import data from an XML document, specify all the XML nodes (specify node names) and indicate the type of each, whether it represents an element or an attribute. Map these nodes to corresponding data structures (tables or fields) in the Microsoft Dynamics NAV database or to variables. When an XMLport object is called to handle an incoming XML document, it reads the incoming data stream (for example, a file) and performs the processing and database actions.

To create an XMLport to export data to an XML document, build the node structure of the XML document by using the XMLport Designer and map the data. When an XMLport object is called to export data, it reads the required data from the database or variables, adds the necessary XML nodes to form the XML document, and writes the document to a data stream (such as a file).

For incoming documents of these types, use C/AL code to perform the necessary database manipulation to achieve the result that you want.

Designing an XMLport is actually designing the structure of the external file, whether it is an XML document or a text file. This primarily involves setting various properties in the XMLport objects, such as XMLport properties and Node properties.

Data Model

The data model maps the data between the external document and the tables and fields in the database. The data model consists of nodes that typically represent tables and fields.

Nodes of the **Table** type specify the tables from which or into which data will be exported or imported. This includes any necessary relationship between tables. Nodes of the **Field** type specify which fields from the parent table will be included in the import or export.

When an XMLport is used to export data, each table node in the XMLport is iterated for all records in the underlying table. Sorting order, keys, and table views can be set to achieve specific ordering of exported data or to filter the resulting sets according to predefined criteria.

When an XMLport is used to import data, records that are read from the external file are inserted into the table that corresponds to the table node in the XMLport. You can examine the records before inserting them, specify whether records are inserted automatically, and decide whether records already in the database are overwritten or updated when a record with the same primary key is read from the external file. You can also determine to completely skip records from import.

External File

The layout of the external file is defined by setting properties, including setting the XMLport properties and XMLport node properties.

External files that are handled through XMLports can be in XML or plain text format. In both cases, the definition of the nodes in XMLport Designer must match the structure of the external file being exported or imported.

When an XMLport is used to export data, the XMLport definition describes how the fields and records will be mapped to the structure of the external file, and will result in the external file containing the exact structure as defined in the XMLport. XMLport cannot export more fields or information than specified in the XMLport definition.

When an XMLport is used to import data, the XMLport definition describes the minimum number of fields, and depending on the format of the XMLport and structure of the file, the external file might contain more fields than defined by the XMLport definition.

XMLport Properties

The XMLport properties generally describe the XMLport. Several properties, such as Direction and FileName, can be set and reset dynamically. For example, developers can create an XMLport where the user can do one of the following:

- Select whether to import or export.
- Select the name of the external file to read from or write to.
- Generate a file name automatically when the XMLport is run.

The following table shows the properties that are available on XMLports.

Property	Remarks
Direction	Defines whether the XMLport can handle import, export, or both. The following options are available: • Import • Export • Both Default is Both.
DefaultFieldsValidation	Determines whether fields will be validated during import or assigned. This setting can be overridden for a specific field by specifying the **FieldValidate** property on the field.
FileName	The default file name that is used by the XMLport. If nothing is specified, then the name of the XMLport will be suggested as the import or export file name.
UseRequestPage	Specifies if the XMLport will display a request page that has options and filters. If this is set to **No**, then the XMLport will continue directly onto import or export, depending on the value of the **Direction** property. If Direction is **Both**, then the request page will provide a choice of Import or Export. Default is import.

Note: *The **Direction** property strictly defines how an XMLport can be used. If direction is explicitly set to **Import** or **Export**, then the XMLport cannot be used to handle documents in the opposite direction. If you want an XMLport to handle import and export, then you must make sure that it specifies the direction of **Both**.*

Node Properties

The node properties describe the XMLport node that defines the XMLport's structure. Depending on the node type, different properties will be available. The following table shows examples of node properties.

Property Name	Remarks
Indentation	Sets the indentation level of an element or attribute in the **XMLport Designer**.
NodeName	Specifies the name of a node in an XML document. This property can also be accessed directly in the **XMLport Designer**.
NodeType	Specifies whether an XML object is an element or an attribute. You can access this property directly in the **XMLport Designer**.
SourceType	Specifies what a particular node in the XMLport designer corresponds to in the Microsoft Dynamics NAV database. This property can also be accessed directly in the **XMLport Designer**.
SourceTable	Selects the table to be mapped to XML data. This property can also be accessed from the **Data Source** field in the **XMLport Designer**.
SourceField	Selects the field that is to be mapped to XML data. This property can also be accessed from the **Data Source** field in the **XMLport Designer**.
TextType	Specifies which type of text that an element or attribute contains.
FieldValidate	Specifies whether the values in the **SourceField** must be validated by the **OnValidate** trigger for the field during the import.
VariableName	Specifies a variable name for the table that is specified in the SourceTable property. Also used to specify a variable name for the text if it is specified in the SourceType property.
SourceTableView	Used to sort and filter the data in the table that is set as the source for the XML node.
CalcFields	Automatically calculates the information for FlowFields that are entered in the **CalcFields** property.

Property Name	Remarks
AutoCalcField	Specifies if the value in the FlowField must be automatically calculated before the field is exported.
LinkTable	Determines to which table the XML item must be linked, and can only be used for the **SourceType** of Table.
LinkTableForceInsert	Specifies whether data from the linked table must be forcibly inserted into a table to prevent the system from generating an error. This feature is useful if you have a header to line relationship in your XML document. The table and the header information must be inserted before you can insert the line information. As a result, you can use this property to ensure that the header information is inserted before the XMLport starts reading the line information.
LinkFields	Specifies the fields from the two tables that are linked by the **LinkTable** property. This is available only for XML items with Table as the data source.
Temporary	Creates a temporary table in the XMLport.
MinOccurs	Specifies the minimum number of times that an element can occur.
MaxOccurs	Specifies the maximum number of times that an element can occur.
Occurrence	Specifies whether an attribute must occur in the data that is being imported or exported.

Node Properties for Import

When Node Type is set to Table, there are three more properties that specify the behavior of the XMLport during import operations: AutoSave, AutoUpdate, and AutoReplace.

The following table shows how these properties are used.

Property Name	Remarks
AutoSave	Specifies whether imported records are automatically written to the table.
AutoUpdate	Specifies whether a record in the database that has the same primary key as a record in the import file is updated with values from the imported record. Values of database fields that are not present in the import file will not be modified by the XMLport.

Property Name	Remarks
AutoReplace	Sets whether imported records automatically replace existing records with the same primary key. Values of database fields that are not present in the import file will be reset to their initial value, according to their **InitValue** property.

These three properties determine how records that are read from the external file are handled. They are also used to resolve the conflict that arises when a record that is read from the external file during import has the same primary key as a record that already exists in the database table. The following table shows combinations of these property values and the results.

AutoSave	AutoUpdate	AutoReplace	Record exists in the database and in the external file.	Record exists only in the external file.
No	N/A	N/A	The record in the database is not automatically updated or replaced.	The record from the external file is not automatically inserted into the database.
Yes	No	No	A runtime error occurs and the import is stopped.	The record from the external file is automatically inserted into the database.
Yes	No	Yes	The record from the external file replaces the record in the database.	The record from the external file is automatically inserted into the database.
Yes	Yes	N/A	The record from the external file updates the record in the database.	The record from the external file is automatically inserted into the database.

See also XMLport Properties in the Developer and IT Pro Help for Microsoft Dynamics NAV 2013 at http://go.microsoft.com/fwlink/?LinkId=268740.

Design and Run XMLports

You create and modify XMLports in the **XMLport Designer**, which you can access from the **Object Designer**.

In Microsoft Dynamics NAV, you can run XMLports from the following:

- Department places, through the Menu Suite object
- **RunObject** property of controls and actions in page objects
- C/AL code

Note: *As with many other object types, during development you can also run XMLports from the* ***Object Designer****, or the* ***XMLport Designer****.*

Demonstration: Create an XMLport to Export to XML Documents

The following steps show how to create an XMLport to export the sales order data from the **Sales Header** and **Sales Line** tables to an external XML document.

Demonstration Steps

1. Create an XMLport object and set the properties to allow export in XML format.
   a. In the Object Designer's XMLport list, click **New**. The XMLport Designer opens.
   b. Open the **Properties** window for the XMLport and set the following properties:
      - **Direction**: Export (This specifies the XMLport to be used only for export.)
      - **Format**: Xml (This specifies the external file to be an XML document.)
      - **Format/Evaluate**: XML Format/Evaluate

Note: *Always use the XML Format/Evaluate, because this guarantees that you can exchange the documents with other systems under other regional settings. Documents created with C/SIDE Format/Evaluate can only be exchanged with Microsoft Dynamics NAV and properly interpreted during import only under the same regional settings under which they were exported.*

   c. Close the **Properties** window.

2. Define that the root element of the document is of type Text, and name it "Root". Under root element, configure an element with name "SalesHeader", set **Source Type** to table, and then set **Data Source** to Sales Header.
    a. Click the first empty line in the XMLport Designer and type the following:

Node Name	Node Type	Source Type
Root	Element	Text

    b. Go to the next line and type the following to add a source table. Make sure that it is indented under the Root element.

Node Name	Node Type	Source Type	Data Source
SalesHeader	Element	Table	36

Note: *The **Sales Header** table ID is 36. You can type 36 directly into the **Data Source** field, or you can use the **LookUp** button on the **Data Source** field and select the **Sales Header** table.*

3. Configure the SalesHeader element to only include the documents of type Order.
    a. Select the SalesHeader element row, then on the **View** menu, click **Properties**.
    b. In the **SourceTableView** property, click the **AssistEdit** button.
    c. In the **Table View** window, in the **Table Filter** field, click the **AssistEdit** button.
    d. In the **Table Filter** window, in the **Field** column, choose Document Type, in the **Type** column, choose CONST, and then in the **Value** column, enter Order.
    e. Click **OK** to accept changes and close the **Table Filter** window.
    f. Click **OK** to accept changes and close the **Table View** window.
    g. Close the **Properties** window.

4. Under SalesHeader, define four new elements for fields **Sell-to Customer No.**, **No.**, **Order Date**, and **Currency Code**. Name the fields Customer, No, Date, and Currency, respectively.
    a. Go to the next line and type the following on the next few lines to add fields from the Sales Header table. Make sure that they are indented under the SalesHeader element.

Node Name	Node Type	Source Type	Data Source
Customer	Element	Field	Sales Header::Sell-to Customer No.
No	Element	Field	Sales Header::No.
Date	Element	Field	Sales Header::Order Date
Currency	Element	Field	Sales Header::Currency Code

 Note: *Instead of typing the value into the* ***Data Source*** *field, you can use the* ***LookUp*** *button to open the* ***Field Lookup*** *window and select the fields shown.*

5. Define another element with name "SalesLine", Source Type set to Table, and Data Source set to Sales Line. Indent this element under the SalesHeader and link the SalesLine to only include the lines that belong to the current header.
   a. Go to the next line and type the following to add a source table. Make sure that it is indented under the SalesHeader element.

Node Name	Node Type	Source Type	Data Source
SalesLine	Element	Table	37

   b. Select the SalesLine element row, and the on the **View** menu, click **Properties**.
   c. In the **LinkTable** property, enter "Sales Header".
   d. In the **LinkFields** property, press the **AssistEdit** button to access the **DataItem Link** window.
   e. In the **DataItem Link** window, enter the following information:

Field	Reference Field
Document Type	Document Type
Document No.	No.

   f. Press **OK** to accept the changes and close the **DataItem Link** window.
   g. Close the **Properties** window.

6. Under SalesLine, define five new elements for fields **Type**, **No.**, **Quantity**, **Unit Price**, and **Line Discount %**. Name the fields Type, No, Quantity, Price, and Discount, respectively.

a. Go to the first empty line and type the following on the next few lines to add fields from the Sales Line table. Make sure that they are indented under the SalesLine element.

Node Name	Node Type	Source Type	Data Source
Type	Element	Field	Sales Line::Type
No	Element	Field	Sales Line::No.
Quantity	Element	Field	Sales Line::Quantity
Price	Element	Field	Sales Line::Unit Price
Discount	Element	Field	Sales Line::Line Discount %

7. Save the XMLport as 123456700, **Sales Order Export XML**.
   a. Compile and save the XMLport by clicking **File > Save As**. The **Save As** dialog box appears.
   b. Enter "123456700" in the **ID** field and "Sales Order Export XML" in the **Name** field. Make sure that the **Compiled** check box is selected and then click **OK**. This compiles and saves the XMLport.

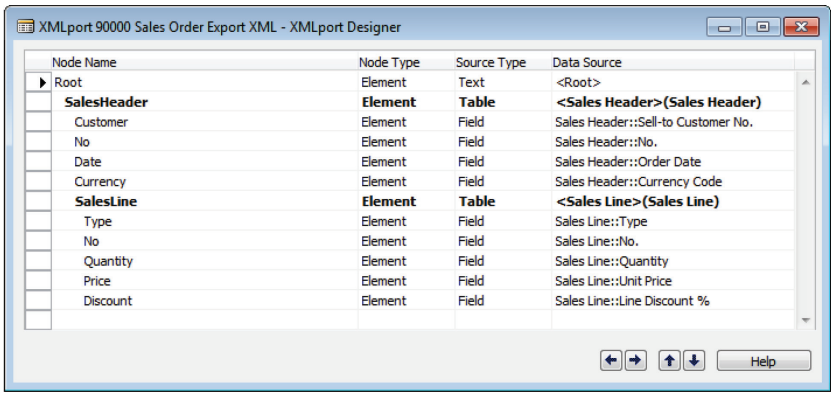

FIGURE 9.2: SALES ORDER EXPORT XML XMLPORT

c. Close the **XMLport Designer**.

Demonstration: Create an XMLport to Import from XML Documents

The following steps show how to create an XMLport to import data from an external XML document to the Vehicle table.

Demonstration Steps

1. Create a new XMLport, and set the properties to allow import from files in XML format.
   a. In the Object Designer's XMLport list, click **New**. The XMLport Designer opens.
   b. Open the **Properties** window for the XMLport and set the following properties:
      - **Direction**: Import (This specifies the XMLport to be used only for import.)
      - **Format**: Xml (This specifies the external file to be an XML document.)
      - **Format/Evaluate**: XML Format/Evaluate
   c. Close the **Properties** window.

2. Define that the root element of the document is of type Text, and name it "Root". Under root element, configure an element with name "Vehicle", set **Source Type** to table, and then set **Data Source** to table 90000.
   a. Click the first empty line in the XMLport Designer, and then type the following:

Node Name	Node Type	Source Type
Root	Element	Text

   b. Go to the next line and type the following to add a source table. Make sure that it is indented under the Root element.

Node Name	Node Type	Source Type	Data Source
Vehicle	Element	Table	90000

3. Under Vehicle, define three new elements for fields Vehicle::Model, Vehicle::Serial No., and Vehicle::List Price. Name the elements Model, Name, and ListPrice respectively.
   a. Go to the next line and type the following on the next few lines to add several fields from the Vehicle table. Make sure that they are indented under the Vehicle element.

Node Name	Node Type	Source Type	Data Source
Model	Element	Field	Vehicle::Model
SerialNo	Element	Field	Vehicle::Serial No.
ListPrice	Element	Field	Vehicle::List Price

4. Save the XMLport, as 123456701, Vehicle Import XML.
   a. Compile and save the XMLport by clicking **File > Save As**. The **Save As** dialog box appears.
   b. Type "123456701" in the **ID** field and "Vehicle Import XML" in the **Name** field. Make sure that the **Compiled** check box is selected and then click **OK**. This compiles and saves the XMLport.
   c. Close the **XMLport Designer**.

Lab 9.1: Create an XMLport to Export XML Data

Scenario

Simon is a developer working for CRONUS International Ltd. CRONUS International has decided to start selling Microsoft Dynamics NAV training courses as its business.

Simon has already created a Course table to record course information and Card and List pages to interface with the Course table. Now, Simon must create an XMLport to export all the course details from the **Course List** page in the XML format.

Objectives

The objectives are:

- Create an XMLport.
- Extend a page to run an XMLport.

Exercise 1: Create an XMLport for export to the XML document.

Exercise Scenario

Simon first creates the XMLport object that imports the document.

Task 1: Create a new XMLport

High Level Steps

1. Create a new XMLport objects, and set the properties to export in XML format.
2. Create an element of type Text, and name it “Root”.
3. Under Root, create a child element of type Table, name it “Course”, and set its source to 90010.
4. Add child elements to Course, for fields Code, Name, Description, Type, Duration, Price, Active, Difficulty, and Passing Rate.
5. Save the XMLport as 123456703, Course Export XML.

Detailed Steps

1. Create a new XMLport objects, and set the properties to export in XML format.
    a. In the **Object Designer**'s **XMLport list**, click **New**. The XMLport Designer opens.

b. Open the **Properties** window for the XMLport and set the following properties:
   - **Direction**: Export (This specifies the XMLport to be used only for export.)
   - **Format**: Xml (This specifies the external file to be an XML document.)
   - **Format/Evaluate:** XML Format/Evaluate

c. Close the **Properties** window.

2. Create an element of type Text, and name it "Root".
   a. Click the first empty line in the XMLport Designer, and then type the following:

Node Name	Node Type	Source Type
Root	Element	Text

3. Under Root, create a child element of type Table, name it "Course", and set its source to 90010.
   a. Go to the next line and type the following to add a source table. Make sure that it is indented under the Root element.

Node Name	Node Type	Source Type	Data Source
Course	Element	Table	90010

4. Add child elements to Course, for fields Code, Name, Description, Type, Duration, Price, Active, Difficulty, and Passing Rate.
   a. Go to the next line and type the following on the next few lines to add several fields from the Course table. Make sure that they are indented under the Course element.

Node Name	Node Type	Source Type	Data Source
Code	Element	Field	Course::Code
Name	Element	Field	Course::Name
Description	Element	Field	Course::Description
Type	Element	Field	Course::Type
Duration	Element	Field	Course::Duration
Price	Element	Field	Course::Price
Active	Element	Field	Course::Active
Difficulty	Element	Field	Course::Difficulty

Node Name	Node Type	Source Type	Data Source
PassingRate	Element	Field	Course::Passing Rate

5. Save the XMLport as 123456703, Course Export XML.
    a. Compile and save the XMLport by clicking **File** > **Save As**. The **Save As** dialog box appears.
    b. Type "123456703" in the **ID** field and "Course Export XML" in the **Name** field. Make sure that the **Compiled** check box is selected and then click **OK**. This compiles and saves the XMLport.
    c. Close the **XMLport Designer**.

Task 2: Add action to page

High Level Steps

1. Design page 90011, Course List, from the **Object Designer** and access the **Action Designer**.
2. Add an action container and an action.
3. Set the properties to run XMLport Course Export XML.
4. Compile, save, and close the page.
5. Run the **Export Course Detail (XML)** action from the page and export the course details into a file.

Detailed Steps

1. Design page 90011, Course List, from the **Object Designer** and access the **Action Designer**.
    a. In Object Designer, select page **90011**, **Course List** and click **Design**.
    b. On the **View** menu, click **Page Actions** to open the **Action Designer** for the page.

2. Add an action container and an action.
    a. Type the following on the empty line:

Type	SubType
ActionContainer	ActionItems

    b. Type the following on the next line and indent it under the ActionContainer.

Caption	Type
Export Course Details (XML)	Action

3. Set the properties to run XMLport Course Export XML.
   a. Open the **Properties** window for the RunXMLport XML action and set the RunObject property to XMLport Course Export XML.
   b. Close the **Properties** window.

4. Compile, save, and close the page.
   a. On the **File** menu, click **Save**.
   b. In the **Save** window, make sure that **Compiled** check box is selected, and then click **OK**.
   c. Close the Page Designer.

5. Run the **Export Course Detail (XML)** action from the page and export the course details into a file.
   a. In the **Object Designer**, click **Page**.
   b. Select page 90011, **Course List**, and then click **Run**.
   c. On the **Actions** tab in the ribbon, click **Export Course Details (XML)**.
   d. In the request page for the **Export Course Details** XMLport, click **OK**.
   e. In the **Export File** dialog window, click **Save**.
   f. In the **Export** dialog window, accept all the defaults, and then click **Save**.
   g. Locate the file you have created, double click it to open it in Internet Exporer, and review its contents.

Importing and Exporting Plain Text

Concerning data exchange between applications, XML provides many advantages over the text format. XML-structured data means integrity, increased flexibility, and better human readability. XML format also provides extensibility, such as attaching schemas to achieve data validation, or applying style sheets to transform the data into a completely different XML structure.

Despite these powerful features, situations still occur where data has to be exchanged in a plain text format.

Several legacy systems do not support XML at all or cannot easily handle data in XML format. Additionally, many external applications, such as Microsoft Office Excel, can easily produce or read data in text format. To simplify this exchange, XMLports can also export and import data in text format, in addition to XML format.

Variable Text and Fixed Width Formats

Although the text formatting rules are not as strict as those imposed by XML format, the data in text files must still be structured in a way that a computer can easily manipulate.

The two ways to structure plain text data are:

- Variable Text
- Fixed Width

With Variable Text format, columns are separated by using a delimiter—such as comma, colon or tab character—whereas in Fixed Width format, each column of data has a fixed length for characters.

Properties for Handling Text Files

Several XMLport object and XMLport node properties are specific to handling text files. These properties are described in the following table.

Name	Type	Property Of	Description
Format	Option	XMLport	Determines the import/export format of the XMLport. The options are as follows: • XML • Variable Text • Fixed Text
Width	Integer	Nodes (elements and/or attributes)	Determines the width of the fixed width field. If Fixed Length is selected in the Format property, all elements and attributes of the Text and Field types should have their Width property value set to greater than 0, or it gives compilation errors.
FieldDelimiter	String	XMLport	Determines what the text delimiter is for a field. The default value is a double quotation mark (").
FieldSeparator	String	XMLport	Determines what the field separator is for fields. The default value is a comma (,).
RecordSeparator	String	XMLport	Determines what the record separator is for records. The default value is <NewLine>.
TableSeparator	String	XMLport	Determines what the table separator is for tables. XMLports can import or export data from or into several tables at once. The tables are separated in the file by the table separator. The default value is <NewLine><NewLine>.

Note: *When Variable Text or Fixed Width formats are used, it does not matter*

if you use Element or Attribute as node types. However, you must still order all attributes above all elements, and you must not use more than one level of hierarchy below an element of the Table source type.

Demonstration: Create XMLports to Export Variable Text

The following demonstration shows how to create XMLports for export of plain text by copying and modifying an existing XMLport.

Demonstration Steps

1. Design XMLport **123456700**, Sales Order Export XML and save it as XMLport 123456702, Sales Order Export Variable.
   a. In the **Object Designer**, select XMLport 123456700, Sales Order Export XML, and then click the **Design**.
   b. On the **File** menu, click **Save As**.
   c. In the **Save As** window, in **ID** field, enter "123456702", then in the **Name** field, enter "Sales Order Export Variable", make sure that the **Compiled** check box is selected, and then **OK**.

2. Set the properties to define the variable text format, and set the field separator to the comma character (,), and make sure that no field delimiter is used.
   a. On the **View** menu, click **Properties**.
   b. In the **Properties** window for the XMLport set the following properties:
      - **Format**: Variable Text
      - **FieldSeparator**: ,
      - **FieldDelimiter**: <None>

   c. Close the **Properties** window.
   d. Select all the lines starting with the Table line for the Sales Line table, until the last line.
   e. Press F4 to delete the lines. A dialog box opens asking you if you are sure to delete the lines.
   f. Click **OK** to confirm the dialog box.
   g. On the **File** menu, click **Save**.
   h. In the **Save** window, make sure that the **Compiled** check box is selected, and then **OK**.

3. Run the XMLport.
   a. On the **File** menu, click **Run**.
   b. In the request page for the XMLport, click **OK**.
   c. In the **Export File** dialog window, click **Save**.
   d. In the **Export** dialog window, accept all the defaults, and then click **Save**.
   e. Locate the file you have created, double click it to open it in Notepad, and review its contents.

Demonstration: Run an XMLport from a Page

An XMLport can be run from an action in a page. The following steps show how to run an XMLport from a page in the RoleTailored client.

Demonstration Steps

1. Design page 90001, **Custom Page** and access the **Action Designer**.
   a. In the **Object Designer**, select page **90001**, **Custom Page**, and then click Design.
   b. On the **View** menu, click **Page Actions** to open the **Action Designer** for the page.

2. Add a new action to run XMLport Sales Order Export XML.
   a. Type the following on the empty line:

Caption	Type
Run XMLport	Action

   b. Set the **RunObject** property of the action to XMLport Sales Order Export XML.

3. Save the page, and run it to test the new action.
   a. On the **File** menu, click **Save**.
   b. In the **Save** window, make sure that **Compiled** check box is selected, and then click **OK**.
   c. Run the page and run the action.

Lab 9.2: Create an XMLport to Export Variable Text

Scenario

Simon is a developer working for CRONUS International Ltd. CRONUS International has decided to start selling Microsoft Dynamics NAV training courses as its business.

Simon has already created a Course table to record course information, Card and List pages to interface with the Course table, and an XMLport to export the course details in XML format. Some customers of CRONUS International are using legacy systems that cannot handle XML data. Simon must create an XMLport to export all the course details from the **Course List** page in the comma-separated text format.

Objectives

The objectives are:

- Create an XMLport to export data in variable text format.
- Extend a page to run an XMLport.

Exercise 1: Create an XMLport for Export to the Variable Text Document

Task 1: Create the XMLport

High Level Steps

1. Save XMLport Course Export XML as XMLport 123456704, Course Export Variable.
2. Modify the properties to export data in variable text format.
3. Compile, save, and close the XMLport.

Detailed Steps

1. Save XMLport **Course Export XML** as XMLport **123456704**, **Course Export Variable**.
   a. Design XMLport Course Export Xml from the Object Designer.
   b. Compile and save the XMLport with the ID of 123456704 and the name of Course Export Variable.

2. Modify the properties to export data in variable text format.
   a. Open the **Properties** window for the XMLport and set the following properties:
      - **Direction**: Export (This specifies the XMLport to be used only for export.)

- **Format**: Variable Text (This specifies the external file to be a variable text document.)

b. Close the Properties window.

3. Compile, save, and close the XMLport.
   a. On the **File** menu, click **Save**.
   b. In the **Save** window, make sure that **Compiled** check box is selected, and then click **OK**.

Task 2: Add action to run the XMLport

High Level Steps

1. Design page 90011, Course List, from the **Object Designer** and access the **Action Designer**.
2. Add an action to the action container.
3. Set the properties to run XMLport Course Export Variable.
4. Compile, save, and close the page.

Detailed Steps

1. Design page 90011, Course List, from the **Object Designer** and access the **Action Designer**.
   a. In Object Designer, select page **90011**, **Course List** and click **Design**.
   b. On the **View** menu, click **Page Actions** to open the **Action Designer** for the page.

2. Add an action to the action container.
   a. Type the following on the empty line and indent that line under the **ActionContainer**:

Caption	Type
Export Course Details (Variable Text)	Action

3. Set the properties to run XMLport Course Export Variable.
   a. Open the **Properties** window for the RunXMLport Variable action and set the **RunObject** property to XMLport Course Export Variable.
   b. Close the **Properties** window.

4. Compile, save, and close the page.
   a. On the **File** menu, click **Save**.
   b. In the **Save** window, make sure that the **Compiled** check box is selected, and then **OK**.

Using XMLports in C/AL Code

Even though you can integrate XMLports into the user interface, actual situations in which users would be exporting and importing data directly are infrequent. C/AL programming language provides several functions to control the behavior of XMLports programmatically.

In addition to importing data from and exporting data to external files, XMLports provide streaming capabilities to access data directly, regardless of the source or destination type. Instead of having to know the technical capabilities of the specific source or destination, programmers can access them consistently and effortlessly.

The following are typical types of source and destinations that are accessed from XMLports:

- Memory
- BLOB fields in the database
- Files
- DotNet variables

Importing and Exporting Data through C/AL Code

Occasionally, the direction of the XMLport and source or destination file name cannot be predetermined by you, cannot be left to the user's choice, and in addition must be controlled programmatically during run time. In these situations you can use C/AL functions to dynamically determine whether the XMLport will import or export data, and which file will be used as the source or destination.

The following table shows the C/AL functions that you can use to control the behavior of XMLports dynamically during run time.

Function or Property	Remark
IMPORTFILE	Specifies whether the XMLport will import the data or export the data.
FILENAME	Specifies the source or the destination file for the XMLport.
RUN	Runs the XMLport object.

The following examples show how to use the IMPORTFILE, FILENAME, and RUN functions.

Instructing XMLport to import data

```
Xml.IMPORTFILE := TRUE;
```

Note: *IMPORTFILE can only be used on XMLports that have the* ***Direction*** *property set to* ***Both****. If you try to set the direction to a different one than specified in the* ***Direction*** *property, a run-time error will occur.*

To specify the default file name to be used by XMLport, use the FILENAME function. The following example uses *importfile.xml* as the import file.

Specifying the default file name for an XMLport

```
Xml.FILENAME := 'importfile.xml';
```

Running XMLport programmatically

```
Xml.RUN;
```

Streaming Data by Using XMLports

Data streaming allows you to read from or written to different media in a consistent way, regardless of the actual technical implementation of the medium being accessed. For example, if you want to stream data to a BLOB field in the database, a file, or a .NET Framework application, you can access the data in the same way. Although these three media are fundamentally different in their internal implementation, you can use the same simple set of C/AL functions to access them similarly, without the need to understand or depend on internal technical specifics of BLOB fields, files, or .NET.

In addition to external files, XMLports can use streams as their source or destination. This lets you import data from or export data to any medium that supports streaming.

C/AL natively supports streaming for the following:

- Database BLOB fields
- Files
- .NET

The following table shows the C/AL functions that support streaming through XMLports.

Function	Remarks
SETSOURCE	Specifies the **InStream** variable to be used as the source for the data import.
SETDESTINATION	Specifies the **OutStream** variable to be used as the destination for the data export.
IMPORT	Imports the data from the designated variable by using the SETSOURCE function.
EXPORT	Exports the data into the designated variable by using the SETDESTINATION function.

Demonstration: Create a Codeunit to Run the XMLport for Export

The following demonstration shows how to create a codeunit to run the XMLport.

Demonstration Steps

1. Create a new codeunit.
   a. In the **Object Designer**, click **Codeunit**.
   b. Cick **New** to open the **C/AL Editor**.

2. Create a new variable of type File, and name it OutputFile, and another variable of type Stream, and name it OutStream.
   a. On the **View** menu, click **C/AL Globals**.
   b. In the **Variables** tab, type the following:

Name	DataType
OutputFile	File
Stream	OutStream

   c. Close the **C/AL Globals** window.

3. In **OnRun** trigger enter the code to create an output file at *C:\Sales Order Export.xml*, and stream the data from XMLport **Sales Order Export Variable** into the file.
   a. Select the first line under the **OnRun** trigger and type the following:

Code Example

```
OutputFile.CREATE('C:\Sales Order Export.xml');
OutputFile.CREATEOUTSTREAM(Stream);
XMLPORT.EXPORT(XMLPORT::"Sales Order Export Variable",Stream);
OutputFile.CLOSE;
MESSAGE('Sales Order Export Xml completed!');
```

Note: *This codeunit creates a new text file C:\Sales Order Export.xml.*

4. Save the codeunit as 90004, **XML Export – Sales Order**.
   a. Compile and save the codeunit by clicking **File, Save As**. The **Save As** dialog box appears.
   b. Type "90004" in the **ID** field and "XML Export - Sales Order" in the **Name** field. Make sure that the **Compiled** check box is selected and then click **OK**. This compiles and saves the codeunit.

5. Run the codeunit and view the result.
   a. Run the codeunit and view the result.
   b. Locate the new file *C:\Sales Order Export.xml* in the file system.
   c. Double-click this file to open it in the browser.

Demonstration: Create an XML Input File

The database table that is used in the following demonstration is the Vehicle table (which was created previously in module 2: Tables in this course). Before you create the XMLport for import, create the XML file to be used as the import file. The following steps show how to create an XML file that contains records to be imported to the Vehicle table.

Demonstration Steps

1. In Notepad, create the XML document as shown in the code example.
   a. Open the Notepad.
   b. Type the following:

Code Example

```
<Root>
 <Vehicle>
  <Model>FIAT</Model>
  <SerialNo>1000</SerialNo>
  <ListPrice>15000</ListPrice>
 </Vehicle>
 <Vehicle>
  <Model>FIAT</Model>
  <SerialNo>2000</SerialNo>
  <ListPrice>40000</ListPrice>
 </Vehicle>
 <Vehicle>
  <Model>MAZDA</Model>
  <SerialNo>5500</SerialNo>
  <ListPrice>30000</ListPrice>
 </Vehicle>
 <Vehicle>
  <Model>MAZDA</Model>
  <SerialNo>7500</SerialNo>
  <ListPrice>25000</ListPrice>
 </Vehicle>
</Root>
```

c. Save the file as VehicleXML.xml and close the Notepad.

2. Save the file as VehicleXML.xml

Demonstration: Create a Codeunit to Run an XMLport for Import

The following steps show how to create a codeunit to run an XMLport.

Demonstration Steps

1. Create a new codeunit.
   a. In the Object Designer's Codeunit list, click **New**. The C/AL Editor opens.

2. Define a new variable of type XmlPort, name it VehicleImport, then create a new variable of type File, and name it InputFile, and then create a new variable of type InStream, and name it Stream.
   a. Click **View**, **C/AL Globals**.
   b. In the **Variables** tab, type the following:

Name	DataType	Subtype
VehicleImport	XmlPort	Vehicle Import Xml
InputFile	File	
Stream	InStream	

   c. Close the **C/AL Globals** window.

3. In **OnRun** trigger write the code to open file *C:VehicleXML.xml*, and stream it into the database through VehicleImport XMLport.
   a. Click the first line under the OnRun trigger and type the following:

Code Example

```
InputFile.OPEN('C:\VehicleXML.xml');
InputFile.CREATEINSTREAM(Stream);

VehicleImport.SETSOURCE(Stream);

VehicleImport.IMPORT;
InputFile.CLOSE;
MESSAGE('Vehicle Import Xml completed!');
```

This codeunit imports an XML file called *C:\VehicleXML.xml*.

4. Save the codeunit as 123456705, XML Import – Vehicle.
   a. Compile and save the codeunit by clicking **File > Save As**. The **Save As** dialog box appears.
   b. Type "123456705" in the **ID** field and "XML Import – Vehicle" in the **Name** field. Make sure that the **Compiled** check box is selected and then click **OK**. This compiles and saves the codeunit.

5. Manually delete all records from table **90000**, **Vehicle**.
   a. Run table **90000**, **Vehicle** from the Object Designer.
   b. Delete all the records in the table, and then close the table.

6. Run codeunit **90005**, **XML Import – Vehicle**, and then table **90000**, **Vehicle**, and verify that the import has resulted in records in the table.
    a. Run the codeunit and then run table **90000**, **Vehicle** from the Object Designer and view the result. The table now contains the imported record.

Module Review

Module Review and Takeaways

XMLports are objects that are used for importing data from and exporting data to external files. XMLports can import data and export data in XML format or plain text format.

Use XMLport to export data from Microsoft Dynamics NAV or import data into Microsoft Dynamics NAV in plain text format instead of XML format in the following scenarios:

- Supporting businesses that use earlier versions of Microsoft Dynamics NAV.
- Changing Microsoft Dynamics NAV data in a tool that does not support XML.

Microsoft Dynamics NAV 2013 uses XMLports as a means to export and import data from and to Microsoft Dynamics NAV database. Even though they are called XMLports, they can also handle text files, in the fixed and variable formats.

XMLports can access external files directly, or can use streaming to access data in any external object that supports streaming through C/AL InStream or OutStream objects.

Test Your Knowledge

Test your knowledge with the following questions.

1. In an import XMLport, if the AutoSave property is set to No and the AutoUpdate property is set to Yes, what does the XMLport do with the records from the external file?

2. When must the Width property of an XMLport node be used?

3. Every XMLport node must be bound to an actual field in a table.

   ( ) True

   ( ) False

4. What does the CalcFields property do on a node of the source type **Table**?

5. When you export, which XMLport format lines up the information in columns?

6. When you export, which XMLport format creates the smallest possible file?

7. When you import, which XMLport property can be set so that the user cannot view the **Options** FastTab?

8. Which property is used to specify whether the name in the **NodeName** field represents data of the element or attribute types?

9. Which property specifies that the node name corresponds to a table in the Microsoft Dynamics NAV database?

10. An XMLport cannot modify existing data in the database.

    ( ) True

    ( ) False

11. What is the purpose of the **Direction** property in the XMLport?

Test Your Knowledge Solutions

Module Review and Takeaways

1. In an import XMLport, if the AutoSave property is set to No and the AutoUpdate property is set to Yes, what does the XMLport do with the records from the external file?

   MODEL ANSWER:

   The XMLport only reads the records from the file, but does not automatically insert them, and does not automatically update or replace data in the database.

2. When must the Width property of an XMLport node be used?

   MODEL ANSWER:

   You must use this property when the Format property of the XMLport is set to Fixed Text.

3. Every XMLport node must be bound to an actual field in a table.

   ( ) True

   (√) False

4. What does the CalcFields property do on a node of the source type **Table**?

   MODEL ANSWER:

   It sets the lists of FlowFields that the XMLport automatically calculates during export.

5. When you export, which XMLport format lines up the information in columns?

   MODEL ANSWER:

   Fixed Text.

6. When you export, which XMLport format creates the smallest possible file?

   MODEL ANSWER:

   Variable Text

7. When you import, which XMLport property can be set so that the user cannot view the **Options** FastTab?

   MODEL ANSWER:

   UseRequestPage

8. Which property is used to specify whether the name in the **NodeName** field represents data of the element or attribute types?

   MODEL ANSWER:

   NodeType

9. Which property specifies that the node name corresponds to a table in the Microsoft Dynamics NAV database?

   MODEL ANSWER:

   SourceType

10. An XMLport cannot modify existing data in the database.

    ( ) True

    (√) False

11. What is the purpose of the **Direction** property in the XMLport?

    MODEL ANSWER:

    It specifies whether an XMLport can be used for import, export, or both.