

MODULE 3: DOCUMENTS

Module Overview

Once master tables and pages are in place, the next step is to implement the functionality that lets users perform transactions with the master data. There are several ways that users can enter transactional information into Microsoft Dynamics NAV 2013. Documents are an intuitive feature that enables users to enter and manage transactions in a simple way. Documents consist of a header and lines.

In almost every functional area of Microsoft Dynamics NAV 2013, there are various types of documents which let users create and manage various transactions. Documents are frequently complex and span multiple functional areas, such as sales orders. You can use sales orders not only to manage shipments and invoicing of goods to customers, but also to coordinate shipping activities with the warehouse department or manufacturing activities with the production department. Documents can also be very simple, and manage a very narrow area of functionality in a single functional area. For example, you can use reminders to remind customers of overdue payments.

The seminar management functionality lets users manage seminar registrations. Each seminar is delivered in a single room by a single trainer, with a single starting and ending date. Multiple participants attend each seminar. The concept of documents lets you develop a functionality that is easy to use and that users will intuitively understand.

Objectives

- Import and export objects as text files.
- Support multilanguage functionality.
- Use document pages.
- Use virtual tables.
- Use temporary tables.
- Review the various types of tables.
- Review different page and table C/AL functions.
- Create additional tables and pages to maintain registrations.


Prerequisite Knowledge

Before developing the solution for handling seminar registrations, you must become familiar with several more development concepts in Microsoft Dynamics NAV 2013.

Working with Objects as Text Files

Microsoft Dynamics NAV 2013 Development Environment lets you import and export objects to move them between different environments. For example, you can export objects from your development environment, and then import them into the test environment or the production environment.

You can export and import objects in the following formats.

Format	Remarks
<p>Dynamics NAV Object Format (.fob)</p>	<p>This is the binary format that lets you import and export the objects in their compiled state. You do not require a special license to import or export the .fob files. Users can use their own end-user license to export objects into a .fob file, or to import a .fob file that they receive from you.</p> <p>After you import objects from a .fob file, the objects are saved as compiled.</p> <hr/> <p> Note: You should still compile the imported objects to make sure that all references to other objects are valid, and to update the metadata.</p>
<p>Text Format (.txt)</p>	<p>This is the plain text format that lets you view, or even change the contents of a file after you export it, and before you import it into another environment. To import or export the objects in .txt files, you must have a developer license, and be able to design the objects that you want to export or import.</p> <p>After you import the objects from a .txt file, the objects are saved as un-compiled.</p>

Format	Remarks
XML format (.xml)	This format resembles the text format, except that the objects are saved in a more structured way that lets you automate the analysis or changes in the exported files. Everything that applies to the .txt files about licensing limitations and the compiled state also applies to the .xml files.

Developers frequently import and export objects as text files, because it provides the following benefits:

- You can easily analyze the contents of the objects before you import them.
- You can change the contents of the objects outside Microsoft Dynamics NAV 2013 Development Environment.
- Microsoft Dynamics NAV 2013 does not let you import the objects if you do not have an appropriate license. This reduces risks of accidental or unintentional replacement of objects in a production environment.



Note: Microsoft Dynamics NAV 2013 Development Environment does not ask you for confirmation when you import objects from .txt files. If your license lets you import objects, it always replaces the existing objects with the imported objects.

To export one or more objects as a text file, follow these steps:

1. In **Object Designer**, select the objects that you want to export.
2. Click **File > Export**.
3. In the **Export Objects** dialog box, in the **Save as type** field, select Text Format (*.txt), in the **File name** field.
4. Type a file name, and then click **Save**.

The resulting text file contains all details of the object. The first line for every object in the file begins with the word OBJECT, the object type, number, and name. The text for the rest of the object follows.

The following example shows how an object is represented in the text format:

```


OBJECT Table 123456701 Seminar Setup
{
  OBJECT-PROPERTIES
  {
    Date=15.06.12;
    Time=20:18:41;
    Modified=Yes;
    Version List=CSD1.00;
  }
  PROPERTIES
  {
    CaptionML=ENU=Seminar Setup;
  }
  FIELDS
  {
    { 1 ; ;Primary Key ;Code10 ;CaptionML=ENU=Primary Key }
    { 2 ; ;Seminar Nos. ;Code10 ;TableRelation="No. Series";
      CaptionML=ENU=Seminar Nos. }
    { 3 ; ;Seminar Registration Nos.;Code10 ;TableRelation="No. Series";
      CaptionML=ENU=Seminar Registration Nos. }
    { 4 ; ;Posted Seminar Reg. Nos.;Code10 ;TableRelation="No. Series";
      CaptionML=ENU=Posted Seminar Reg. Nos. }
  }
  KEYS
  {
    { ;Primary Key ;Clustered=Yes }
  }
  FIELDGROUPS
  {
  }
  CODE
  {
    BEGIN
    {
      CSD1.00 - 2012-06-15 - D. E. Veloper
      Chapter 2 - Lab 2
      - Table created
    }
    END.
  }
}
    
```

FIGURE 3.1: TABLE 123456701, SEMINAR SETUP IN TEXT FORMAT

This file consists of several sections:

Section	Remarks
OBJECT-PROPERTIES	This section contains the object date, time, modified flag, and version properties.

Section	Remarks
PROPERTIES	<p>This section includes any properties that you set on the object level by accessing the Properties window for the object. In the example that was mentioned earlier, it is the CaptionML property, however, it can be many other properties.</p> <p>If the object has any object-level triggers, such as the table object OnInsert, OnModify, OnDelete, and OnRename triggers, they are included in the PROPERTIES section.</p> <p><i>Triggers</i> are included only in the text file when they contain C/AL code. <i>Properties</i> are included only when the value is set to something other than the default value. Properties that have default values and triggers without any C/AL code are not included in an object text file.</p>
FIELDS	<p>This section includes definitions of table fields, their properties, and any C/AL triggers that are defined on fields.</p>
KEYS	<p>This section contains the list of the keys in the table.</p>
FIELDGROUPS	<p>This section contains the list of field groups that are defined on the table. Use field groups to define fields that are displayed in the quick lookup box in the RoleTailored client.</p>
CODE	<p>This section includes the global variables, text constants, and functions in the object. At the end of the CODE section, within the BEGIN..END. block, are the contents of the Documentation trigger.</p>

 **Note:** The number of sections in the object's .txt file varies depending on the object type.

When you change and save the contents of the .txt file, you can import the file back into Microsoft Dynamics NAV 2013 by using the following procedure:

1. Open the **Object Designer**.
2. Click **File > Import**.
3. Select the appropriate file in the **Import Objects** dialog box, and then click **Open**. The objects are imported.

You must compile the objects before you can use them. You cannot use the **Import Worksheet** when importing a text file. This means the following:

- No warning is received about overwriting existing tables.
- You cannot skip the import of some objects.
- You cannot merge objects.

Multilanguage Functionality in Text Messages

When you create messages for the user, you must make sure that the text and the object names in the messages are enabled for multilanguage functionality.

If your code has to display any errors, confirmations, or messages, do not enter those text messages directly in the C/AL code. This makes your code dependent on the language that you used initially. Any users who run Microsoft Dynamics NAV 2013 in another language may be unable to understand the messages.

In the following example, the confirmation and error texts are hardcoded. You should avoid writing code such as this.

Bad practice example

```
IF CONFIRM('Do you want to delete the customer?') THEN BEGIN

    Cust.DELETE;

    MESSAGE('Customer no. %1 is successfully deleted.',Cust."No.");

END;
```

Microsoft Dynamics NAV 2013 does not recommend hardcoding text messages directly in the C/AL code because it ignores multilanguage functionality. It degrades the user experience by enabling your code to correctly run only under a single user interface language. In the earlier example, the captions of tables and fields were hardcoded. If these captions and fields were changed, the resulting text message could be confusing for users.

When you display text in a dialog box, a page, or a report, you must define such text as a text constant. This enables the following:

- Users always see the message in their language.
- It makes your user interface more consistent, if you display the same message in multiple locations in the code.
- Other developers can localize your code into their language without changing the code.

To define and declare a text constant, do the following:

1. Click either **C/AL Globals** or **C/AL Locals** on the **View** menu.
2. On the **Text Constants** tab, in the Name column, type the name of the constant. As a convention, all names of text constants start with Text. This is followed by a number. For example, Text001.
3. In the ConstValue column, enter the text that Microsoft Dynamics NAV 2013 must display to users. This text is in the same language as Microsoft Dynamics NAV 2013 Development Environment.

To define the text constant value in other languages, do the following:

1. In the ConstValue column, click the **AssistEdit** button to open the **Multilanguage Editor**.
2. In the Language column, enter the language name abbreviation, or select from the list of languages.
3. In the Value column, enter the text value in the selected language.
4. Click **OK** to accept your changes and close the **Multilanguage Editor**.

The following example shows the correct way to display text messages to users in a way that enables the multilanguage functionality of Microsoft Dynamics NAV 2013:

Using text constants to display multilanguage enabled texts

```
// Text001: Do you want to delete the %1?  
  
// Text002: %1 %2 %3 is successfully deleted.  
  
IF CONFIRM(Text001,FALSE,Cust.TABLECAPTION) THEN BEGIN  
  
    Cust.DELETE;  
  
    MESSAGE(Text002,  
  
        Cust.TABLECAPTION,Cust.FIELDCAPTION("No."),Cust."No.");  
  
END;
```

When you refer to tables and fields in your errors, confirmations, or messages, you must use the TABLECAPTION and FIELDCAPTION functions. These functions return the caption of a table or a field translated into the language that the user selected in the client. Do not use the TABLENAME or FIELDNAME functions, because they return only the name of the object. This is always in a single language (typically English), and cannot be translated automatically to other languages.

Document Pages

A *Document page* (also known as a **Header/Line**, or a **Master/Detail** page) combines FastTabs, similar to those found in card pages, with a **ListPart** page. It displays records from two tables with a one-to-many relationship, in a single page.

The Document page itself acts as a master page for the header or main table. The subpage of the ListPart type shows the related records from the lines or detail table. For example, the **Sales Invoice** page (Page 43) is used to create, view, or change sales invoice documents. Similar to Card pages, it displays fields from the header table that are grouped in several FastTabs.

In addition to these FastTabs, the page also has a special type of FastTab. This FastTab displays the subpage that displays the records from the lines table that are related to the header table. For the **Sales Invoice** page, the main page is associated with the header table, **Sales Header** (Table 36). The subpage (Page 47) is associated with the lines table, **Sales Line** (Table 37). The two pages are linked by the **Document Type** and **Document Number** fields. They define the relationship between the **Sales Header** table and the **Sales Line** table.

In **Page Designer**, you define a subpage control with a Type of Part, and PartType of Page. Then you set the PagePartID property to the ID of the **ListPart** page that you want to show as a subpage. You also have to set the SubPageLink property to establish the link between the main page and the subpage.

In the **Sales Invoice** page, the properties of the subpage's Part element, the PagePartID property is set to the **Sales Invoice Subform** (Page 47). To link the subpage records to the current **Sales Header** record, the SubFormLink property is set to "Document No.=FIELD(No.)".

The **Sales Invoice Subform** page is associated to the line table, **Sales Line** (Table 37). The page itself is a ListPart page that has a Repeater element that contains the required fields from the **Sales Line** table. Document subpages always use the following standards:

- The key fields from the lines table are not displayed. Instead, they link to the main table through the **SubPageLink** property that automatically populates the linked key fields with the values from the main table. For example, the **Sales Invoice Subform** page does not include the key fields **Document Type**, **Document No.**, and **Line No.** The values of the **Document Type** and **Document No.** fields are populated automatically by the link between the main page and the subpage.
- The **AutoSplitKey** property is set to **Yes**. This property causes the **Line No.** field to be populated automatically by the system when users create new rows in the subpage.

Standard Microsoft Dynamics NAV naming conventions for these forms and tables are as follows:

Type	Naming Convention	Example
Document Table (Header)	Name of Transaction or Document + Header	Sales Header (Table 36)
Document Table (Line)	Name of Transaction or Document + Line	Sales Line (Table 37)
Document Page	Name of Document Represented	Sales Invoice (Page 43)
Document Subpage	Name of Document Represented + Subform	Sales Invoice Subform (Form 47)

Page Functions

Page functions are called through the *CurrPage* variable. This variable is a reference to the instance of the current page and is available only in the C/AL code in page objects. The following table presents the page functions.

Function	Remarks
CurrPage.SAVERECORD	Saves the current record to the database.
CurrPage.UPDATE	Saves the current record, and then updates the controls in the page. If the <i>SaveRecord</i> parameter is <i>TRUE</i> , this function saves the record before the system updates the page. If this parameter is <i>FALSE</i> , the system updates the page.
CurrPage.SETSELECTIONFILTER	Notes the records that the user has selected on the page, marks those records in the specified table, and sets the filter to marked-only.

Function	Remarks
CurrPage.ACTIVATE	Brings the current page into the focus of the user.
CurrPage.CLOSE	Closes the current page.

Virtual Tables

A *virtual table* contains information that is provided by the system and is presented in C/SIDE as a table object. Microsoft Dynamics NAV 2013 provides access to several virtual tables. They resemble regular database tables, and you access them in the same manner, except that virtual tables are read-only and the information that they contain cannot be changed. Virtual tables are not stored in the database as typical tables, but they are generated by the system at run time.

Because a virtual table behaves the same as a regular physical table, you can use the same methods to access their information. For example, you can use filters to obtain subsets or ranges of integers or dates. Here is a list of virtual tables that is available in Microsoft Dynamics NAV 2013:

- Object (Table 2000000001)
- Date (Table 2000000007)
- Session (Table 2000000009)
- Drive (Table 2000000020)
- File (Table 2000000022)
- Integer (Table 2000000026)
- Table Information (Table 2000000028)
- System Object (Table 2000000029)
- AllObj (Table 2000000038)
- Printer (Table 2000000039)
- License Information (Table 2000000040)
- Field (Table 2000000041)
- License Permission (Table 2000000043)
- Permission Range (Table 2000000044)
- Windows Language (Table 2000000045)
- Database (Table 2000000048)
- Code Coverage (Table 2000000049)
- SID - Account ID (Table 2000000055)
- AllObjWithCaption (Table 2000000058)
- Key (Table 2000000063)
- Debugger Call Stack (Table 2000000101)

- Debugger Variable (Table 2000000102)
- Debugger Watch Value (Table 2000000103)

Because the virtual tables are not stored in the database, you cannot run them directly from the **Object Designer**. You only can view the contents of a virtual table if you create a page that uses a virtual table as its source table, or you can access its contents from C/AL code.

You will rarely use many virtual tables. The following virtual tables are most frequently used.

Virtual Table	Remarks
Object	Contains the list of all objects in a Microsoft Dynamics NAV 2013 application.
Date	Contains the list of all periods (days, weeks, months, quarters, or years) between January 03, 0001, and December 31, 9999.
Integer	Contains the list of all integer numbers.
AllObjWithCaption	Contains the list of all objects in a Microsoft Dynamics NAV 2013 application, together with their captions.
Field	Contains the list of all fields in all tables, with captions and other metadata about the fields.

You use the **Date** virtual table for the Seminar Registration process. The **Date** virtual table provides easy access to days, weeks, months, quarters, and years. Each row represents a single period. The table defines the periods with five fields as follows.

Field	Remarks
Period Type	The type of the period (Day, Week, Month, Quarter, Year).
Period Start	The date of the first day in the period.
Period End	The date of the last day in the period.

Field	Remarks
Period No.	The number of the period within the parent. For example, any period of type Day, where the day is Monday, has the Period No. equal to 1; or any period of type Month, where the month is February, has the Period No. equal to 2.
Period Name	The display name of the period, localized to the current display language.

Temporary Tables

A *temporary table* is a memory-based table; a record type variable that exists only in the computer's memory. Temporary tables are not physical tables in the database, but are always based on physical tables. Unlike virtual tables, they are not read-only.

A temporary table can do almost anything that a regular database table does. However, the information in the table is lost when the table is closed.

The write transaction principle that applies to ordinary database tables does not apply to temporary tables: **COMMIT** does not affect temporary tables, and **ERROR** does not roll back any earlier changes to the data in the temporary tables.

The advantage of using a temporary table is that all interaction with a temporary table occurs in memory on the service tier. This reduces the load both on the network and on the SQL Server. To perform many operations on data in a specific table in the database, it can be helpful to load the information into a temporary table for processing.

Defining a temporary table is equal to defining a record variable. To define a temporary record variable, use the following steps:

1. In either the **C/AL Globals** or **C/AL Locals** variable window, define a variable with the data type Record. Select a table in the **Subtype** field.
2. Open the **Properties** window for the variable, and set the **Temporary** property to **Yes**.

By default, record type variables are linked to a physical table in the database. By setting a record variable's **Temporary** property to Yes, the record variable becomes a temporary table.

System Tables

System Tables are stored in the database just like regular tables. However, they are different because they are created automatically by Microsoft Dynamics NAV 2013 Development Environment when you create a new database. The system tables track different system-related information. Following are a few examples:

- Security permissions
- Object metadata
- Record links
- Charts

You can read, write, change, and delete the data in system tables exactly like database tables.

Registrations

CRONUS International Ltd. organizes seminars, and requires functionality that lets users schedule seminars and manage seminar registrations. Now that you have developed the tables and pages for managing master data for seminars, you must develop the functionality that lets users manage seminar registrations, their primary type of transactions.

Users must be able to schedule seminars to occur at a specific time, in a specific seminar room, and to be delivered by a specific instructor. For each scheduled seminar, users must be able to register participants. The most intuitive way to deliver such functionality is documents.

Solution Design

The CRONUS International Ltd. functional requirements define the seminar scheduling functionality as follows:

- Users must be able to schedule seminars. Each seminar has a starting date, an allocated seminar room, an assigned instructor, the minimum and the maximum participants, and the price. The minimum participants and the price information are always taken from the seminar master record. The maximum participants are taken as the lower number of maximum participants of the seminar and maximum participants of the room.
- A seminar cannot be scheduled in a room that cannot hold at least the minimum number of participants for the seminar.

- For each scheduled seminar, users must enter additional comments, such as necessary equipment, or other special requirements.
- It must be possible to assign additional expenses to a scheduled seminar, such as catering expenses or equipment rentals.

These functional requirements indicate that each scheduled seminar is a piece of information separate from the seminar. For scheduled seminars, information from multiple tables is referenced. There is also specific subsidiary information for seminars. This includes seminar expenses and comments.

Additionally, the functional requirements define the registration functionality as follows:

- Users must be able to register one or more participants for scheduled seminars. For each registered participant, it must be possible to specify if additional expenses must be invoiced for this registration. The default is Yes.
- If the room maximum capacity exceeds the maximum participants that are defined for the seminar, then the user who maintains registrations can decide to register more participants up to the room's maximum capacity. Users must be clearly warned if they are registering participants over the maximum number of participants for the seminar.

The combination of these requirements indicates the following separate information areas in the management of seminar registrations:

- Scheduled seminar that includes information about the seminar, the room, the instructor, and some subsidiary information. This includes expenses and comments.
- Seminar registration that includes information about participants in the seminar and how they should be invoiced.

The following diagram shows the logical design of the tables in the seminar registration process. On the left side are the prerequisite tables; in the middle are the main processing tables; and on the right side are additional subsidiary tables. The asterisk (*) indicates the tables that must be created. See the "Logical Entity Relationship Diagram."

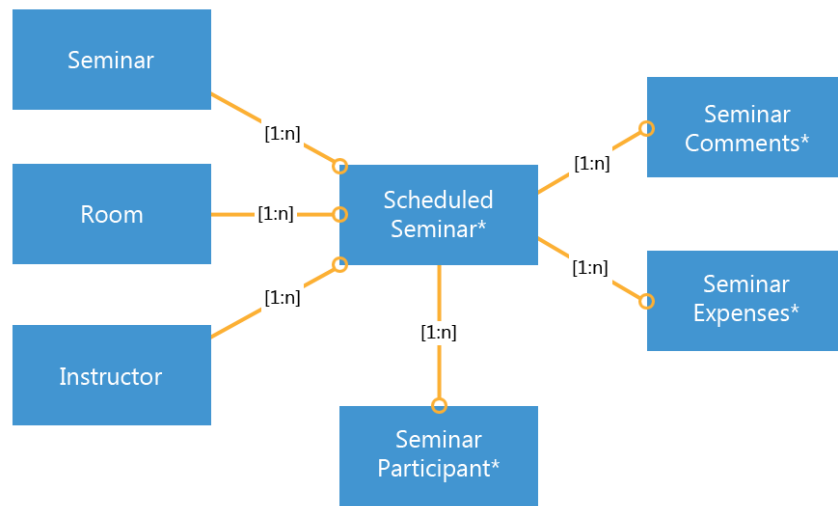


FIGURE 3.2: LOGICAL ENTITY RELATIONSHIP DIAGRAM

The diagram suggests that the best solution for managing registrations is to use the document functionality of Microsoft Dynamics NAV 2013 with the following tables:

- Seminar Registration Header: the information about the scheduled seminar. This includes information about the seminar, the room, and the instructor.
- Seminar Registration Line: the information about the participants in the seminar.

Seminar comments and expenses can be subsidiary tables for the **Seminar Registration Header** table.

The following “Seminar Registration Tables” figure shows the final tables for the seminar registration functionality, together with their relationships. New tables are indicated by an asterisk (*).

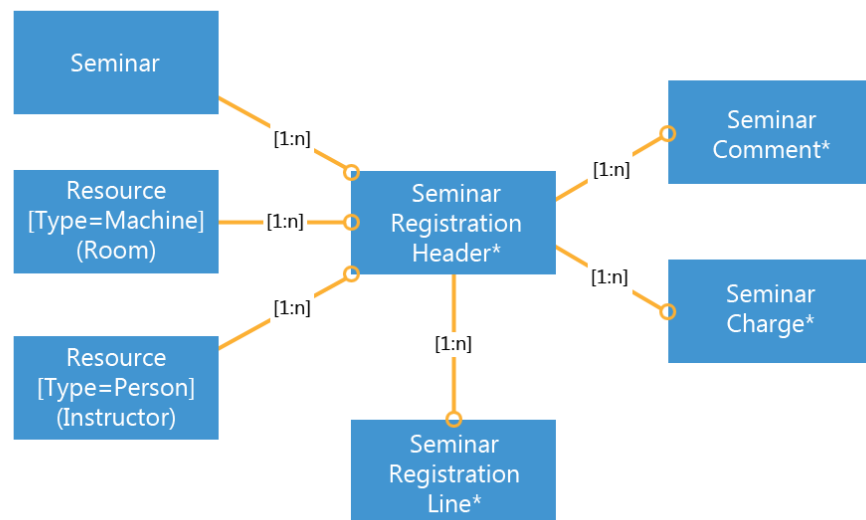


FIGURE 3.3: SEMINAR REGISTRATION TABLES

Development

You must develop the tables and the pages to manage the seminar registration information. Because you decided to use the documents functionality, these tables and pages must follow the standard Microsoft Dynamics NAV 2013 principles for document tables and pages. You must provide all of the functionality that users experience with other document pages elsewhere in the Microsoft Dynamics NAV 2013 application.

Following is typical functionality for the documents:

- There are two tables: **Header**, and **Lines**.
- There is a page of type Document for the **Header** table.
- There is a page of type ListPart for the **Lines** table.
- The Document page includes the ListPart page as a page part. Typically, this page part carries the caption Lines.
- There is a page of type List for the **Header** table that shows all records.
- On the List page there is a FactBox that shows details about the principal master record of the **Header** table. For example, for sales invoices, there is a FactBox that shows the details about the customer.
- On the Document page there is a FactBox that shows details about the principal master record of the **Line** table. For example, for a sales invoice, there is a FactBox that shows the details about the item on the selected sales invoice line.

Tables

To support the seminar registration functionality, you must develop the following new tables.

Table	Remarks
Table 123456710 Seminar Registration Header	Holds the information for one scheduled seminar. This is known as a <i>registration</i> .
Table 123456711 Seminar Registration Line	Holds the information for one participant in a seminar registration.
Table 123456704 Seminar Comment Line	Holds comments for the seminar registrations.
Table 123456712 Seminar Charge	Holds charges that are related to the seminar registration. These are in addition to the individual participant charges of the Seminar Registration Line table.

Pages

The pages for the seminar registration and the navigation between them reflect the relationships that are shown in the "Seminar Registration Tables" diagram. Start by defining the simplest pages first, so that they can be integrated with the more complex pages later in the development process.

The **Seminar Comment List** page displays the comments for a seminar as shown in the following illustration.

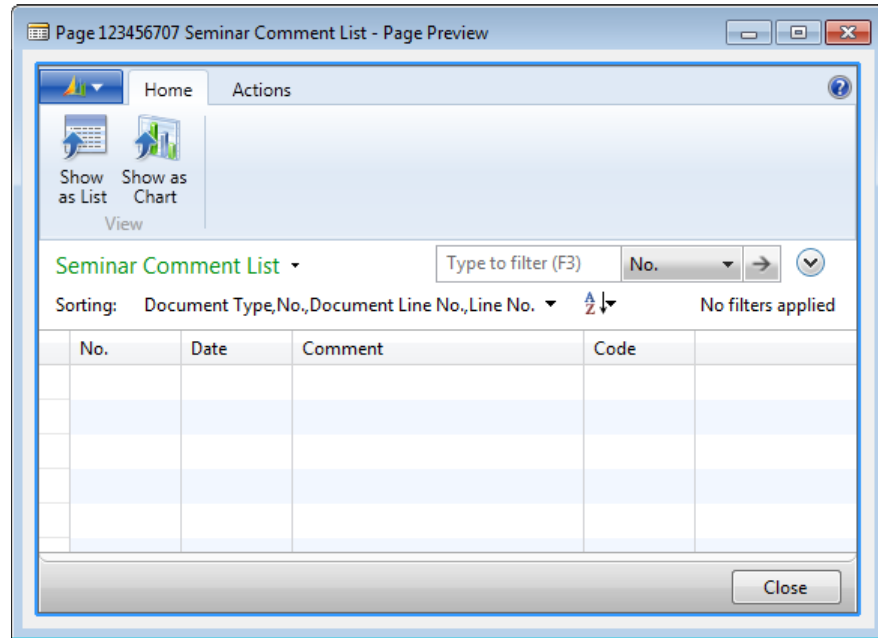


FIGURE 3.4: SEMINAR COMMENT LIST PAGE (123456707)

The **Seminar Comment Sheet** page, as shown in the following the “Seminar Comment Sheet Page,” permits the entry of comments for a seminar.

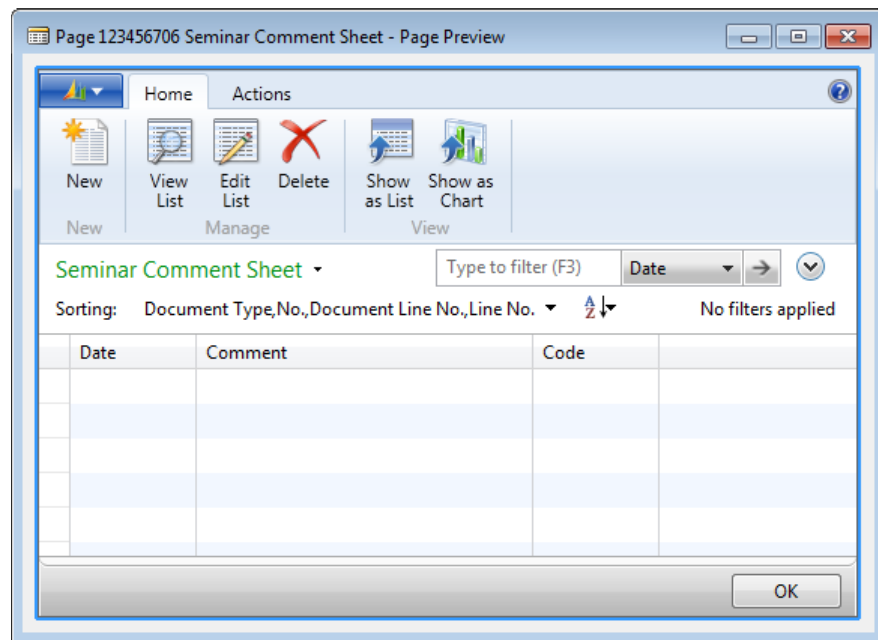


FIGURE 3.5: SEMINAR COMMENT SHEET PAGE (123456706)

The **Seminar Charges** page, as shown in the following “Seminar Charges Page” figure, permits the entry of charges for a seminar.

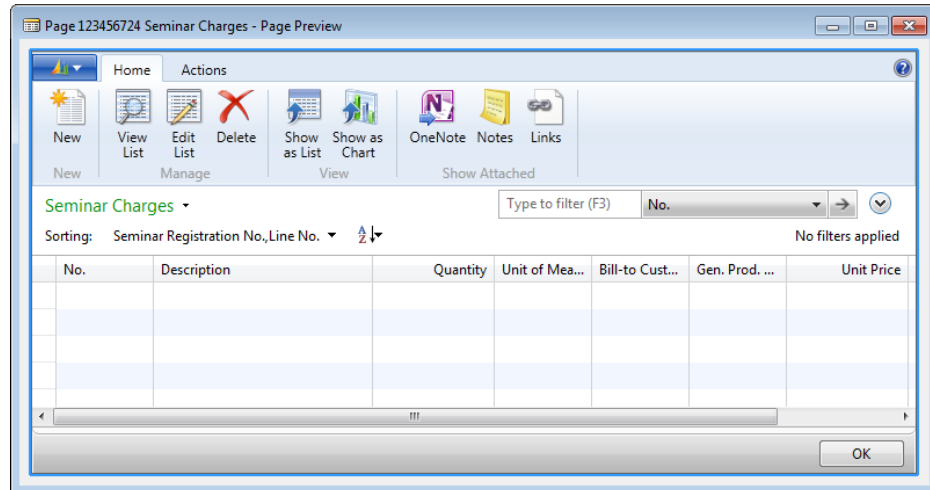


FIGURE 3.6: SEMINAR CHARGES PAGE (123456724)

After you create these supporting pages, define the document pages. In addition to the document and matching list page, you must define the following pages to support the document functionality:

- The ListPart page for the lines
- The CardPart pages that are used as FactBox pages in the **Seminar Registration** and **Seminar Registration List** pages

The “Seminar Registration Subform Page” figure shows the Seminar Registration Subform in the page preview mode. This page is never used directly. The only purpose of this page is to include it as a subpage on the **Seminar Registration** document page.

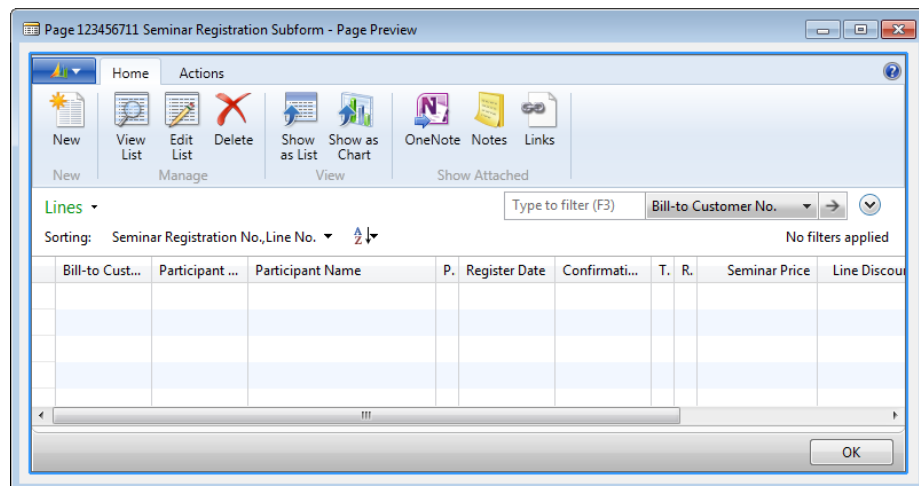


FIGURE 3.7: SEMINAR REGISTRATION SUBFORM PAGE (123456711)

For seminar registrations, there is a FactBox that shows the information about the seminar.

The following image shows the **Seminar Details FactBox** page in the page preview mode. Use this page as a page part on the **Seminar Registration List** page.

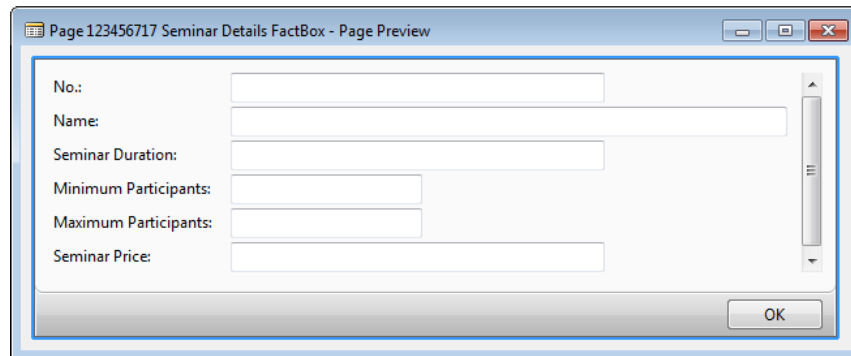


FIGURE 3.8: SEMINAR DETAILS FACTBOX PAGE (123456717)

The **Seminar Registration** page is an example of a Document page, because it includes several FastTabs to manage the **Seminar Registration Header** information, and a FastTab to manage the **Seminar Registration Line** information. There are also several FactBoxes available. They provide more insight into the information in the header or the lines.

The “Seminar Registration Page” figure shows the Seminar Registration page that has its FastTabs and FactBoxes.

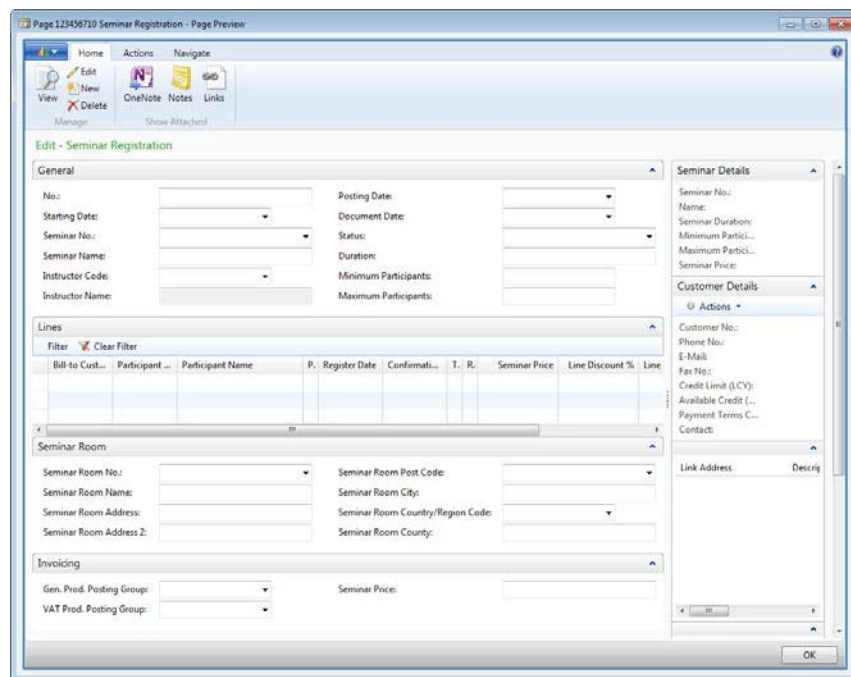


FIGURE 3.9: SEMINAR REGISTRATION PAGE (123456710)

The **Seminar Registration List** page, as shown in the following illustration, displays the seminar registrations.

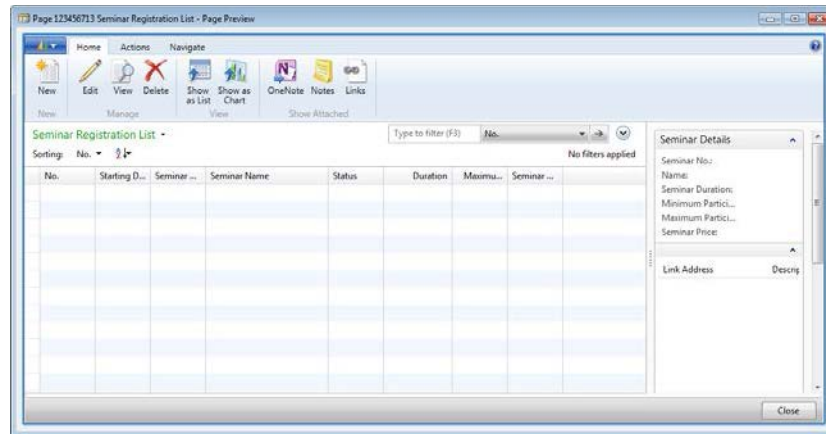


FIGURE 3.10: SEMINAR REGISTRATION LIST PAGE (123456713)

Lab 3.1: Importing, Reviewing and Completing Seminar Registration Tables

Scenario

Isaac, another developer at your company, has created tables to manage seminar registrations. Isaac has given you the text file that contains the tables that he has exported from his Microsoft Dynamics NAV 2013 Development Environment.

As a senior developer and Isaac's supervisor, you review the tables to make sure that they follow all Microsoft Dynamics NAV 2013 standards. You make any necessary corrections to the tables, their properties and fields, and their code.

Exercise 1: Import the Starter Objects

Exercise Scenario

Import the objects from the text file that Isaac provides. You know that the text file import overwrites any objects in the database without asking for confirmation. Therefore, you first review the contents of the file to make sure that it only creates new objects.

Task 1: Review the Text File

High Level Steps

1. Open the Mod03\Labfiles\Lab 3.A - Starter.txt file.
2. Note down all objects and their types that are contained in the file.
3. Make sure that only the following objects are present in the file:
 - o **Seminar Comment Line** table (123456704)
 - o **Seminar Registration Header** table (123456710)
 - o **Seminar Registration Line** table (123456711)
 - o **Seminar Charge** table (123456712)
 - o **Seminar Comment Sheet** page (123456706)
 - o **Seminar Comment List** page (123456707)The file should not contain any other objects.
4. Make sure that no imported objects exist in the database.

Detailed Steps

1. Open the Mod03\Labfiles\Lab 3.A - Starter.txt file.
 - a. Locate the Mod03\Labfiles\Lab 3.A - Starter.txt.
 - b. Double-click the file to open it in Notepad.

2. Note down all objects and their types that are contained in the file.
 - a. In Notepad, click **Edit > Find**.
 - b. In the **Find** dialog box, in **Find what**, type "OBJECT " (with space after the word "OBJECT").
 - c. Click **Find Next**.
 - d. Note down each object type, ID, and name.
 - e. Repeat steps c and d as long as you find objects.

3. Make sure that only the following objects are present in the file:
 - o **Seminar Comment Line** table (123456704)
 - o **Seminar Registration Header** table (123456710)
 - o **Seminar Registration Line** table (123456711)
 - o **Seminar Charge** table (123456712)
 - o **Seminar Comment Sheet** page (123456706)
 - o **Seminar Comment List** page (123456707)

The file should not contain any other objects.

- a. Compare the noted list of objects with the following table.

Type	ID	Name
Table	123456704	Seminar Comment Line
Table	123456710	Seminar Registration Header
Table	123456711	Seminar Registration Line
Table	123456712	Seminar Charge
Page	123456706	Seminar Comment Sheet
Page	123456707	Seminar Comment List

4. Make sure that no imported objects exist in the database.
 - a. In Microsoft Dynamics NAV 2013 Development Environment, open **Object Designer**.
 - b. Click **Table**.
 - c. Click the **ID** column in any row.
 - d. Click **Edit > Find** (or press CTRL+F) to open the **Find** dialog box.

- e. In the **Find What** field, type "123456704", and then click **Find First**.
- f. Make sure that the table is not found.
- g. Repeat steps e and f for each object in the imported file.

Task 2: Importing and Compiling Objects

High Level Steps

1. In **Object Designer**, import the Mod03\Labfiles\Lab 3.A - Starter.txt
2. Select the imported objects.
3. Compile the imported objects.

Detailed Steps

1. In **Object Designer**, import the Mod03\Labfiles\Lab 3.A - Starter.txt
 - a. Open the **Object Designer**.
 - b. Click **File > Import**.
 - c. In the **Import Objects** dialog window, browse to Mod03\Labfiles\Lab 3.A - Starter.txt
 - d. Click **Open**.
2. Select the imported objects.
 - a. In **Object Designer**, click **All**.
 - b. Click **View > Table Filter**.
 - c. In the **Table Filter** window, in the **Field** column, select the **Compiled** field, then in the **Filter** column, type "No".
 - d. Click **OK** to close the **Table Filter** window and apply the filter.
3. Compile the imported objects.
 - a. Press CTRL+A to select all displayed objects.
 - b. Click **Tools > Compile**.
 - c. In the confirmation dialog box, click **Yes** to confirm the compilation.

Exercise 2: Review the Seminar Registration Header Table

Exercise Scenario

After you import the objects into the database, you must review them and make sure that they comply with the standard for the document tables.

Task 1: Reviewing Table and Field Properties

High Level Steps

1. Design the table 123456710, **Seminar Registration Header**.
2. Make sure that you define the table Caption property.
3. Make sure that the Caption property is defined for each field.
4. Correct the Field Name and Length properties for the **Instructor Code** and **Room Code** fields.
5. Compile, save, and then close the table. Reopen it in Table Designer.

Detailed Steps

1. Design the table 123456710, **Seminar Registration Header**.
 - a. In **Object Designer**, click **Table**.
 - b. Find table 123456710, **Seminar Registration Header**.
 - c. Click **Design**.
2. Make sure that you define the table Caption property.
 - a. In **Table Designer**, click the first empty row.
 - b. Click **View > Properties**, or press SHIFT+F4.
 - c. Verify that the Caption property is undefined.
 - d. Click the Value column for the Caption property, then press F8 (this copies the value from the Name property), and then press ENTER.
 - e. Close the **Properties** window.



Note: Make sure that every object in Microsoft Dynamics NAV 2013 has the Caption property defined, because this makes sure that the application takes advantage of the multilanguage functionality. As soon as you define the Caption property, the CaptionML property is defined automatically.

3. Make sure that the Caption property is defined for each field.
 - a. Select the **No.** field, and then press SHIFT+F4 to open the **Properties** window.
 - b. Make sure that the Caption property is defined.
 - c. Repeat this process for each field in the table. If you find a field that does not have a caption, make sure that you define the property by copying the value from the Name property.



Note: The **Minimum Participants** and **Room Name** fields do not require you to define the Caption property.

4. Correct the Field Name and Length properties for the **Instructor Code** and **Room Code** fields.
 - a. Select the **Instructor Code** field.
 - b. In the Field Name column, type "Instructor Resource No."
 - c. In the Length column, type "20".
 - d. Define the Caption property by following the same procedure as in the previous task.
 - e. Select the **Room Code** field.
 - f. In the Field Name column, type "Room Resource No."
 - g. In the Length column, type "20".
 - h. Define the Caption property by following the same procedure as in the previous task.
-



Note: Both fields are related to the **Resource** table. The field name must always indicate the table to which it relates. Additionally, the primary key in the **Resource** table is **No.**, therefore, the field name must end with **No.**, instead of **Code**. Finally, the length of all **No.** fields in master tables is 20, and not 10.

- i. Select the **Instructor Name** field, and then press SHIFT+F4.
 - j. Verify that the CalcFormula property has the following value:
"Lookup(Resource.Name WHERE (No.=FIELD(Instructor Code),Type=CONST(Person)))"
 - k. In the CalcFormula property, replace the words "Instructor Code" with "Instructor Resource No."
 - l. Verify that the CalcFormula property now has the following value:
"Lookup(Resource.Name WHERE (No.=FIELD(Instructor Resource No.),Type=CONST(Person)))"
 - m. Close the **Properties** window.
 5. Compile, save, and then close the table. Reopen it in Table Designer.
 - a. Click **File > Save**.
 - b. In the **Save** dialog window, make sure that the **Compiled** check box is selected, and then click **OK**.
 - c. Close the **Table Designer**.
 - d. Make sure that table 123456710 is still selected in the **Object Designer**, and then click **Design**.
-



Note: You must close the **Table Designer**, and then reopen it in the next task. This makes sure that all field names in the code are updated.

Task 2: Review Code

High Level Steps

1. Review the **Seminar Registration Header** table Documentation trigger.
2. Review the OnInsert trigger.
3. Create the **InitRecord** function, and put the initialization code (except for the number series initialization) from the OnInsert trigger into this function. Then call the **InitRecord** function from the OnInsert trigger.
4. In the OnDelete triggers, make sure that only the seminars in status Canceled can be deleted.
5. Review the rest of the OnDelete trigger to understand how the trigger cleans up any related records during deletion of a **Seminar Registration Header** record.
6. Review the OnValidate trigger for the **No.** field, to understand how it makes sure that changes to the **No.** field are permitted.
7. Review the OnValidate trigger for the **Starting Date** field, and understand how it makes sure that the **Starting Date** field can be changed only for seminar registrations in status Planning.
8. Review the OnValidate trigger for the **Seminar No.** field to understand how it makes sure that you cannot change the **Seminar No.** if there are participants in the seminar. It also populates the default values from the selected **Seminar** record, into the **Seminar Registration Header** record.
9. Review the OnValidate trigger for the **Room Resource No.** field to understand how it populates the seminar room fields. It also checks whether the seminar can register more participants if the room has more capacity than the maximum that is defined by the seminar master record.
10. Review the OnValidate triggers for the **Room Post Code** and **Room City** fields to understand how standard Microsoft Dynamics NAV 2013 functionality populates the post code, city, county, and country from post code or city values.
11. Review the OnValidate trigger for the **Seminar Price** field to understand how it checks for registered participants, and then updates the seminar price for each participant when the user confirms it.

12. Review the code in the OnValidate trigger for the **Posting No. Series** field to understand how it uses the standard functionality in the NoSeriesManagement codeunit to test whether the user has entered a valid number series.
13. Review the OnLookup trigger for the **Posting No. Series** field to understand how it uses the standard lookup functionality as defined in the NoSeriesManagement codeunit.
14. Review the **AssistEdit** function to make sure that it contains the code that resembles the one that you wrote in "Master Tables and Pages."

Detailed Steps

1. Review the **Seminar Registration Header** table Documentation trigger.
 - a. Click **View > C/AL Code**, or press F9.
 - b. In the **C/AL Editor** window, scroll to the beginning of the code.
 - c. Make sure that the Documentation trigger is defined.
2. Review the OnInsert trigger.
 - a. Make sure that the OnInsert trigger contains the code that initializes the number series that is based on the **Seminar Setup** table.

The code should look exactly the same as the following example:

```
IF "No." = " THEN BEGIN

    SeminarSetup.GET;

    SeminarSetup.TESTFIELD("Seminar Registration Nos.");

    NoSeriesMgt.InitSeries(SeminarSetup."Seminar Registration Nos.",xRec."No.
Series",0D,"No.,""No. Series");

END;
```

- b. Check whether there is any code that initializes the fields to certain default values.

The code should look exactly like the following example:

```
IF "Posting Date" = 0D THEN

    "Posting Date" := WORKDATE;

    "Document Date" := WORKDATE;

    SeminarSetup.GET;

    NoSeriesMgt.SetDefaultSeries("Posting No. Series",SeminarSetup."Posted Seminar
    Reg. Nos.");
```



Note: After the number series is initialized in document header tables, many other fields frequently are initialized to default values according to the business process requirements for the document. By convention, all such code is put into the **InitRecord** function that is called immediately after the number series is initialized.

3. Create the **InitRecord** function, and put the initialization code (except for the number series initialization) from the OnInsert trigger into this function. Then call the **InitRecord** function from the OnInsert trigger.
 - a. Click **View > C/AL Globals**.
 - b. On the Functions tab, in the first empty line, type "InitRecord".
 - c. Close the **C/AL Globals** window.
 - d. In the OnInsert trigger, select the code starting with IF "Posting Date" = 0D THEN until the end of the trigger, and then press CTRL+X to cut the code.
 - e. Confirm the cut operation.
 - f. In the OnInsert code, where the deleted code was present, type "InitRecord".

The **OnInsert** trigger should look like the following code example.

```
IF "No." = " THEN BEGIN

    SeminarSetup.GET;

    SeminarSetup.TESTFIELD("Seminar Registration Nos.");

    NoSeriesMgt.InitSeries(SeminarSetup."Seminar Registration Nos.",xRec."No.
    Series",0D,"No.,"No. Series");

END;

InitRecord;
```

- g. Scroll down to the **InitRecord** function.
- h. In the first line of the **InitRecord** function trigger, press CTRL+V to paste the code.

Following is the content of the **InitRecord** function trigger:

```
IF "Posting Date" = 0D THEN

"Posting Date" := WORKDATE;

"Document Date" := WORKDATE;

SeminarSetup.GET;

NoSeriesMgt.SetDefaultSeries("Posting No. Series",SeminarSetup."Posted Seminar
Reg. Nos.");
```

4. In the OnDelete triggers, make sure that only the seminars in status Canceled can be deleted.
 - a. At the beginning of the **OnDelete** trigger, verify that the value of the **Status** field is Canceled. Use the **TESTFIELD** function.

Insert the following line of code before all other code in the OnDelete trigger:

```
TESTFIELD(Status,Status::Canceled);
```

5. Review the rest of the OnDelete trigger to understand how the trigger cleans up any related records during deletion of a **Seminar Registration Header** record.
 - a. Review the code that checks the lines that belong to current **Seminar Registration Header** to make sure that no lines in status Registered exist. If such lines exist, the code throws an error. Otherwise, the code continues with deleting all the lines.

The following block of code first makes sure that only seminars with unregistered lines can be deleted, and then deletes the lines.

```
SeminarRegLine.RESET;

SeminarRegLine.SETRANGE("Document No.,"No.");

SeminarRegLine.SETRANGE(Registered,TRUE);

IF SeminarRegLine.FIND('-') THEN

ERROR(

Text001,
```

```
SeminarRegLine.TABLECAPTION,  
SeminarRegLine.FIELDCAPTION(Registered),  
TRUE);  
SeminarRegLine.SETRANGE(Registered);  
SeminarRegLine.DELETEALL(TRUE);
```

- b. Review the code that deletes all **Seminar Charge** records for the current **Seminar Registration Header**.

The following block of code deletes all seminar charges for a seminar registration.

```
SeminarCharge.RESET;  
SeminarCharge.SETRANGE("Document No.,"No.");  
IF NOT SeminarCharge.ISEMPTY THEN  
ERROR(Text006,SeminarCharge.TABLECAPTION);
```

- c. Review the code that deletes all Seminar Comment Line records for the current Seminar Registration Header record.

The following block of code deletes all comments for a seminar registration.

```
SeminarCommentLine.RESET;  
SeminarCommentLine.SETRANGE("Document  
Type",SeminarCommentLine."Document Type"::"Seminar Registration");  
SeminarCommentLine.SETRANGE("No.,"No.");  
SeminarCommentLine.DELETEALL;
```

6. Review the OnValidate trigger for the **No.** field, to understand how it makes sure that changes to the **No.** field are permitted.
 - a. Locate the No. – OnValidate trigger.
 - b. Make sure that the following block of code exists.

```
IF "No." <> xRec."No." THEN BEGIN

    SeminarSetup.GET;

    NoSeriesMgt.TestManual(SeminarSetup."Seminar Registration Nos.");

    "No. Series" := '';

END;
```



Note: This code resembles the same code in the master tables that you wrote in "Master Tables and Pages." It uses the NoSeriesManagement codeunit and its **TestManual** function.

7. Review the OnValidate trigger for the **Starting Date** field, and understand how it makes sure that the **Starting Date** field can be changed only for seminar registrations in status Planning.
 - a. Locate the Starting Date – OnValidate trigger.
 - b. Make sure that the following code exists.

```
IF "Starting Date" <> xRec."Starting Date" THEN

    TESTFIELD(Status,Status::Planning);
```



Note: Use the **TESTFIELD** function when you have to throw an error if the value of a field does not match a specific value. Use this function to provide a consistent experience across the application.

8. Review the OnValidate trigger for the **Seminar No.** field to understand how it makes sure that you cannot change the **Seminar No.** if there are participants in the seminar. It also populates the default values from the selected **Seminar** record, into the **Seminar Registration Header** record.
 - a. Locate the Seminar No. – OnValidate trigger.
 - b. Make sure that the following block of code exists.

The following block of code makes sure that you cannot change the **Seminar No.** if there are participants registered for the seminar.

```
SeminarRegLine.RESET;

SeminarRegLine.SETRANGE("Document No.,"No.");

SeminarRegLine.SETRANGE(Registered,TRUE);

IF NOT SeminarRegLine.ISEMPTY THEN
```



```

ERROR(
    Text002,
    FIELDCAPTION("Seminar No."),
    SeminarRegLine.TABLECAPTION,
    SeminarRegLine.FIELDCAPTION(Registered),
    TRUE);

```

- c. Make sure that the following block of code exists.

The following block of code retrieves the **Seminar** record, and populates the default values from that location into the current **Seminar Registration Header** record.

```

Seminar.GET("Seminar No.");
Seminar.TESTFIELD(Blocked,FALSE);
Seminar.TESTFIELD("Gen. Prod. Posting Group");
Seminar.TESTFIELD("VAT Prod. Posting Group"); "Seminar Name" :=
Seminar.Name;
Duration := Seminar."Seminar Duration";
"Seminar Price" := Seminar."Seminar Price";
"Gen. Prod. Posting Group" := Seminar."Gen. Prod. Posting Group";
"VAT Prod. Posting Group" := Seminar."VAT Prod. Posting Group";
"Minimum Participants" := Seminar."Minimum Participants";
"Maximum Participants" := Seminar."Maximum Participants";

```



Note: *OnValidate* triggers frequently check for certain conditions when users change the value in the field. In all such situations, you compare Rec.**Field** with xRec.**Field** to check whether the value has changed.

9. Review the OnValidate trigger for the **Room Resource No.** field to understand how it populates the seminar room fields. It also checks whether the seminar can register more participants if the room has more capacity than the maximum that is defined by the seminar master record.
 - a. Locate the Room Resource No. – OnValidate trigger.
 - b. Review the code that cleans up the seminar room fields if the user has specified the empty value, or populates those fields from the selected resource if the user has specified a non-empty value.

The following block of code empties the seminar room fields if the user specifies an empty value for the **Room Resource No.** field, or populates the fields from the **Resource** table if the user has specified a value for the **Room Resource No.**

```
IF "Room Resource No." = '' THEN BEGIN

    "Room Name" := '';

    "Room Address" := '';

    "Room Address 2" := '';

    "Room Post Code" := '';

    "Room City" := '';

    "Room County" := '';

    "Room Country/Reg. Code" := '';

END ELSE BEGIN

    SeminarRoom.GET("Room Resource No.");

    "Room Name" := SeminarRoom.Name;

    "Room Address" := SeminarRoom.Address;

    "Room Address 2" := SeminarRoom."Address 2";

    "Room Post Code" := SeminarRoom."Post Code";

    "Room City" := SeminarRoom.City;

    "Room County" := SeminarRoom.County;

    "Room Country/Reg. Code" := SeminarRoom."Country/Region Code";

END;
```

- c. Make sure that there is code that does the following:
 - Compares the maximum number of participants for the room with the maximum number of participants for the seminar.
 - Asks the user to confirm whether the seminar can accept more participants.

```
IF (CurrFieldNo <> 0) THEN BEGIN

    IF (SeminarRoom."Maximum Participants" <> 0) AND

        (SeminarRoom."Maximum Participants" < "Maximum Participants")

    THEN BEGIN

        IF CONFIRM(Text004,TRUE,

            "Maximum Participants",

            SeminarRoom."Maximum Participants",

            FIELDCAPTION("Maximum Participants"),

            "Maximum Participants",

            SeminarRoom."Maximum Participants")

        THEN

            "Maximum Participants" := SeminarRoom."Maximum Participants";

    END;

END;
```

10. Review the OnValidate triggers for the **Room Post Code** and **Room City** fields to understand how standard Microsoft Dynamics NAV 2013 functionality populates the post code, city, county, and country from post code or city values.

- a. Make sure that the following code exists.

```
PostCode.ValidatePostCode("Room City","Room Post Code","Room  
County","Room Country/Reg. Code",(CurrFieldNo <> 0) AND GUIALLOWED);
```



Note: When you use address fields, make sure that you use the **Post Code**, **City**, **County**, and **Country/Region Code** fields. Then you can use the standard functions in the **Post Code** table. This populates all other fields based on the choice in either **Post Code** or the **City** field.

11. Review the OnValidate trigger for the **Seminar Price** field to understand how it checks for registered participants, and then updates the seminar price for each participant when the user confirms it.
 - a. Make sure that the following block of code exists.

```
IF ("Seminar Price" <> xRec."Seminar Price") AND
  (Status <> Status::Canceled)
THEN BEGIN
  SeminarRegLine.RESET;
  SeminarRegLine.SETRANGE("Document No.,"No.");
  SeminarRegLine.SETRANGE(Registered,FALSE);
  IF SeminarRegLine.FINDSET(FALSE,FALSE) THEN
    IF CONFIRM(Text005,FALSE,
      FIELDCAPTION("Seminar Price"),
      SeminarRegLine.TABLECAPTION)
    THEN BEGIN
      REPEAT
        SeminarRegLine.VALIDATE("Seminar Price","Seminar Price");
        SeminarRegLine.MODIFY;
      UNTIL SeminarRegLine.NEXT = 0;
      MODIFY;
    END;
  END;
```

12. Review the code in the OnValidate trigger for the **Posting No. Series** field to understand how it uses the standard functionality in the NoSeriesManagement codeunit to test whether the user has entered a valid number series.
 - a. Locate the Posting No. Series – OnValidate trigger.
 - b. Make sure that the following code exists:

```
IF "Posting No. Series" <> " " THEN BEGIN

    SeminarSetup.GET;


    SeminarSetup.TESTFIELD("Seminar Registration Nos.");


    SeminarSetup.TESTFIELD("Posted Seminar Reg. Nos.");

    NoSeriesMgt.TestSeries(SeminarSetup."Posted Seminar Reg. Nos.", "Posting No. Series");

END;

TESTFIELD("Posting No.", "");
```

 **Note:** The **TestSeries** method of the NoSeriesManagement codeunit makes sure that the number series that is specified by the user (typically passed as the second parameter) is a valid number series. This number series is defined by the setup (typically passed as the first parameter). The valid number series can be only the number series that is specified in the setup, or any related number series that is defined in the **No. Series Relationship** table.

 **Note:** The last line, TESTFIELD("Posting No.", ""); makes sure that the user can change the number series, only if the **Posting No. Series** field has not yet been assigned. Documents sometimes let users reserve a posting number. If this is the case, the user may not change the **Posting No. Series** field again.

13. Review the OnLookup trigger for the **Posting No. Series** field to understand how it uses the standard lookup functionality as defined in the NoSeriesManagement codeunit.
 - a. Locate the Posting No. Series – OnLookup trigger.
 - b. Make sure that the following block of code exists.

```
WITH SeminarRegHeader DO BEGIN

    SeminarRegHeader := Rec;

    SeminarSetup.GET;
```

```
SeminarSetup.TESTFIELD("Seminar Registration Nos.");  
  
SeminarSetup.TESTFIELD("Posted Seminar Reg. Nos.");  
  
IF NoSeriesMgt.LookupSeries(SeminarSetup."Posted Seminar Reg. Nos.", "Posting  
No. Series")  
  
THEN BEGIN  
  
    VALIDATE("Posting No. Series");  
  
END;  
  
Rec := SeminarRegHeader;  
  
END;
```

14. Review the **AssistEdit** function to make sure that it contains the code that resembles the one that you wrote in "Master Tables and Pages."
 - a. Locate the **AssistEdit** function.
 - b. Make sure that the following code exists.

```
WITH SeminarRegHeader DO BEGIN  
  
    SeminarRegHeader := Rec;  
  
    SeminarSetup.GET;  
  
    SeminarSetup.TESTFIELD("Seminar Registration Nos.");  
  
    IF NoSeriesMgt.SelectSeries(SeminarSetup."Seminar Registration  
Nos.", OldSeminarRegHeader."No. Series", "No. Series") THEN BEGIN  
  
        SeminarSetup.GET;  
  
        SeminarSetup.TESTFIELD("Seminar Registration Nos.");  
  
        NoSeriesMgt.SetSeries("No.");  
  
        Rec := SeminarRegHeader;  
  
        EXIT(TRUE);  
  
    END;  
  
END;
```

Reviewing the Table Code

In Microsoft Dynamics NAV 2013, transactional tables, including document tables, always include lots of code. You have already reviewed the **Seminar Registration Header** table. Now review other tables that you imported.

Most of the code in transactional tables is specific to the transaction that the table supports. But there are frequently many patterns that you can recognize in many other transactional tables. Price calculations for discounts or conversions between different units of measures are several concepts that behave in the same manner. There are many more patterns and concepts that you will recognize if you review the transactional tables in Microsoft Dynamics NAV 2013. When you develop a solution that involves those concepts, you can apply the solution patterns that are present in many standard tables.

Demonstration: Reviewing the Seminar Registration Line Table Code

Document tables are frequently full of code that runs business logic to safeguard the integrity of the document transactions. Many fields in document tables populate the default values from different master records into other fields, run various types of validations, or call operations such as amount, discount, unit of measure, VAT, and other types of calculations.

Reviewing the business logic of the **Seminar Registration Line** table helps you understand the kind of business logic that you must add to any document line tables that you develop.

Demonstration Steps

1. Design table **123456711, Seminar Registration Line**.
 - a. In **Object Designer**, click **Table**.
 - b. Locate the table **123456711, Seminar Registration Line**.
 - c. Click **Design** to open the table in **Table Designer**.
2. Access the C/AL code for the table and review the **GetSeminarRegHeader** and **UpdateAmount** functions.
 - a. Click **View > C/AL Globals**, and then click the **Functions** tab.
 - b. Review the **GetSeminarRegHeader** and **UpdateAmount** functions. Each function performs one of the common tasks of the table code. Because the **Seminar Registration Line** table frequently refers to the **Seminar Registration Header** table, it must keep the reference to the header. It does this in the *SeminarRegHeader* variable.

The **GetSeminarRegHeader** function retrieves the current header into the *SeminarRegHeader* variable if it has changed since it was last read. This is the code in the body of the function.

GetSeminarRegHeader Function


```
IF SeminarRegHeader."No." <> "Document No." THEN BEGIN  
  
    SeminarRegHeader.GET("Document No.");  
  
END;
```

- c. The **UpdateAmount** function is called when the user changes one of the fields that consist of the amount calculation formula for the line. The fields in this calculation are **Seminar Price**, **Line Discount Amount**, and **Line Discount %**. Validation of either of these fields eventually calls the UpdateAmount function.

The following code shows the body of the **UpdateAmount** function. It rounds the number to the precision that is defined in the **Amount Rounding Precision** field in the **General Ledger Setup** table.

UpdateAmount Function

```
GLSetup.GET;  
  
Amount := ROUND("Seminar Price" - "Line Discount Amount",GLSetup."Amount  
Rounding Precision");
```

 **Note:** When you round amounts anywhere in the code, you should retrieve the **General Ledger Setup** table, and round to the precision that is defined in the **Amount Rounding Precision** field. You can omit the rounding precision. If you do this, the system automatically rounds the number to the same precision through the call to the **ReadRounding** function of codeunit 1. For more information about rounding, read the description of the **ROUND** function in the Developer and IT Pro Help.

3. Review the OnInsert trigger to understand how it sets the default **Registration Date**, **Seminar Price**, and **Amount** field value.
 - a. The OnInsert trigger first retrieves the **Seminar Registration Header** record by the call to the **GetSeminarRegHeader** function. It then sets the default values for the **Seminar Price** and **Amount** fields by reading them from the **Seminar Price** field from the Seminar Registration Header record.

The following code is the body of the **OnInsert** trigger:

OnInsert trigger

```
GetSeminarRegHeader;  
  
"Registration Date" := WORKDATE;  
  
"Seminar Price" := SeminarRegHeader."Seminar Price";  
  
Amount := SeminarRegHeader."Seminar Price";
```

4. The OnDelete trigger makes sure that only the lines that are not registered can be deleted.
 - a. When you have to check whether a field contains a specific value, and to throw an error if it does not contain the value, call the **TESTFIELD** function.

The OnDelete trigger uses the **TESTFIELD** function to make sure that the **Registered** field value is *FALSE*.

OnDelete Trigger

```
TESTFIELD(Registered,FALSE);
```

5. Review the OnValidate trigger for the **Bill-to Customer No.** field. It makes sure that you cannot change the customer for the registered line.
 - a. Most of validations only have to occur if the value in the field has changed. You check that by comparing the value to the xRec value.

The following code is the body of the Bill-to Customer No. – OnValidate trigger:

Bill-to Customer No. – OnValidate Trigger

```
IF "Bill-to Customer No." <> xRec."Bill-to Customer No." THEN BEGIN  
  
    IF Registered THEN BEGIN  
  
        ERROR(Text001,  
  
            FIELDCAPTION("Bill-to Customer No."),  
  
            FIELDCAPTION(Registered),  
  
            Registered);
```

```
END;
```

```
END;
```

6. Review the Participant Contact No. – OnValidate trigger to understand how it makes sure that the contact the user chose is related to the customer that is specified in the **Bill-to Customer No.** field.
 - a. The Participant Contact No. – OnValidate trigger filters the information in the **Contact Business Relation** table to determine whether the contact that the user has specified is related to the customer that is referenced in the **Bill-to Customer No.** field. If the contact is not related to the customer, an error that describes the problem is thrown.
 - b. The trigger also calls the CALCFIELD function to retrieve the **Participant Name** field from the **Contact** table. The **Participant Name** field is a FlowField that uses the Lookup method to dynamically calculate the value of the field.

The following code is the body of the **Participant Contact No. – OnValidate** trigger.

Participant Contact No. – OnValidate Trigger

```
IF ("Bill-to Customer No." <> "") AND
    ("Participant Contact No." <> "")
THEN BEGIN
    Contact.GET("Participant Contact No.");
    ContactBusinessRelation.RESET;
    ContactBusinessRelation.SETCURRENTKEY("Link to Table","No.");
    ContactBusinessRelation.SETRANGE("Link to Table",ContactBusinessRelation."Link
to Table"::Customer);
    ContactBusinessRelation.SETRANGE("No.,"Bill-to Customer No.");
    IF ContactBusinessRelation.FINDFIRST THEN BEGIN
        IF ContactBusinessRelation."Contact No." <> Contact."Company No." THEN
        BEGIN
            ERROR(Text002,Contact."No.",Contact.Name,"Bill-to Customer No.");
```

```
END;

END;

END;
```

7. Review the code in the Participant Contact No. – OnLookup triggers to understand how the code first filters the contacts that are related to the customer that is referenced in the **Bill-to Customer No.** field. The code then modally shows the **Contact** page so the user can look up a value.
 - a. The Participant Contact No. – OnLookup trigger also uses the **Contact Business Relation** table to filter the contacts that belong to the referenced customer.
 - b. Microsoft Dynamics NAV 2013 can display a read-only page that lets the user select one of the records, and then click **OK** to confirm the selection, or **Cancel** to give up. To call this functionality, run the page modally by using the **RUNMODAL** function, and test the result of this function. The result of *ACTION::LookupOK* means that the user selected a record, and confirmed the selection by clicking **OK**.



Note: The **RUN** function calls the page, and then immediately continues executing C/AL code. The **RUNMODAL** function calls the page so that only that page can receive focus. It then waits for the user to close the page before it continues executing C/AL code.

The following code is the body of the **Participant Contact No. – OnValidate** trigger:

Participant Contact No. – OnValidate

```
ContactBusinessRelation.RESET;

ContactBusinessRelation.SETRANGE("Link to Table",ContactBusinessRelation."Link
to Table"::Customer);

ContactBusinessRelation.SETRANGE("No.,"Bill-to Customer No.");

IF ContactBusinessRelation.FINDFIRST THEN BEGIN

    Contact.SETRANGE("Company No.",ContactBusinessRelation."Contact No.");

    IF PAGE.RUNMODAL(PAGE::"Contact List",Contact) = ACTION::LookupOK THEN
    BEGIN

        "Participant Contact No." := Contact."No.";
    END;
END;
```

```
END;  
  
END;  
  
CALCFIELDS("Participant Name");
```

8. Review the Seminar Price – OnValidate trigger.
 - a. Many triggers frequently call validations of other fields to run business logic in OnValidate triggers of those other fields. Do this when a field is part of a calculation with several fields. Write the calculation code in only one of the fields' OnValidate trigger, and then call the validation of that field from other locations as necessary.
 - b. The Seminar Price – OnValidate trigger calls the Line Discount % - OnValidate trigger to run the calculation code there.

You run the validation of another field by calling the VALIDATE function.

Seminar Price – OnValidate trigger

```
VALIDATE("Line Discount %");
```

9. Review the Line Discount % - OnValidate and Line Discount Amount – OnValidate triggers to understand how the code calculates one of the discount values that are based on the other value.
 - a. The code in the Line Discount % - OnValidate trigger calculates the **Line Discount Amount** field from the **Line Discount %** field.

Line Discount % - OnValidate Trigger

```
IF "Seminar Price" = 0 THEN BEGIN  
  
    "Line Discount Amount" := 0;  
  
END ELSE BEGIN  
  
    GLSetup.GET;  
  
    "Line Discount Amount" := ROUND("Line Discount %" * "Seminar Price" *  
0.01, GLSetup."Amount Rounding Precision");  
  
END;  
  
UpdateAmount;
```

- b. The code in the Line Discount Amount – OnValidate trigger calculates the **Line Discount %** field from the **Line Discount Amount** field.

Line Discount Amount – OnValidate Trigger

```

IF "Seminar Price" = 0 THEN BEGIN

    "Line Discount %" := 0;

END ELSE BEGIN

    GLSetup.GET;

    "Line Discount %" := ROUND("Line Discount Amount" / "Seminar Price" *
100,GLSetup."Amount Rounding Precision");

END;

UpdateAmount;

```

10. Review the Amount – OnValidate trigger to understand how the **Line Discount Amount** field is calculated based on the price and the amount specified.
 - a. When users enter the **Amount** directly, then the difference between the **Seminar Price** and the **Amount** is the discount that is assigned to the **Line Discount Amount** field.
 - b. Based on the value of the **Line Discount Amount** field, the application calculates the **Line Discount %** field.

Amount – OnValidate Trigger

```

TESTFIELD("Bill-to Customer No.");

TESTFIELD("Seminar Price");

GLSetup.GET;

Amount := ROUND(Amount,GLSetup."Amount Rounding Precision");

"Line Discount Amount" := "Seminar Price" - Amount;

IF "Seminar Price" = 0 THEN BEGIN

    "Line Discount %" := 0;

END ELSE BEGIN

    "Line Discount %" := ROUND("Line Discount Amount" / "Seminar Price" *
100,GLSetup."Amount Rounding Precision");

END;

```

Demonstration: Reviewing the Seminar Charge Table

Many tables in Microsoft Dynamics NAV 2013 frequently relate to several other tables from the same field. This concept is known as *conditional relationships*. These are relationships where one field relates to several tables that are based on the value of another field. For example, the **No.** field in the **Sales Line** table relates to several tables based on the value of the **Type** field. When these relationships are used, there are several patterns that you can see in the existing tables. Apply these patterns to your custom tables to maintain a consistent user experience across the application.

The **Seminar Charge** table also uses the conditional relationship concept by relating to the **G/L Account** and **Resource** tables from the **No.** field, based on the value that is specified in the **Type** field.

Demonstration Steps

1. Design table **123456712, Seminar Charge**.
 - a. In **Object Designer**, click **Table**.
 - b. Locate the table 123456712, **Seminar Charge**.
 - c. Click **Design** to open the table in the **Table Designer**.
2. Review the Type – OnValidate trigger.
 - a. Whenever a table has a conditional relationship, you need to reset any fields defaulted from one relationship, whenever the value in the conditional field changes.
 - b. In the **Seminar Charge** table, when the **Type** field changes, the record is initialized by calling the **INIT** function. Because the **INIT** function initializes all the non-primary key fields that include the **Type** field, the value of the **Type** field must be stored just before it calls **INIT**, and then retrieved just after it calls **INIT**.

The following code applies a pattern that you can recognize in several other standard tables in Microsoft Dynamics NAV 2013.

Type – OnValidate Trigger

```
IF Type <> xRec.Type THEN BEGIN  
  
    Description := "";  
  
END;
```

3. Review the No. – OnValidate trigger to understand how the values for several fields are defaulted from other master records.
 - a. When master tables are referenced, and users select a field from those tables, values for many of the fields in the referencing table are copied from the master table.
 - b. When you use master records, you first must make sure that the record is not blocked.



Note: All master tables should contain the field **Blocked**. This lets users prevent the use of specific records.

- c. In conditional relationships, you typically use the CASE statement to test for various conditions and read defaults from different master tables.

The No. – OnValidate trigger reads defaults from either the **G/L Account** or **Resource** tables, depending on the value in the **Type** field.

No. – OnValidate Trigger

```
CASE Type OF
Type::Resource:
    BEGIN
        Resource.GET("No.");
        Resource.TESTFIELD(Blocked,FALSE);
        Resource.TESTFIELD("Gen. Prod. Posting Group");
        Description := Resource.Name;
        "Gen. Prod. Posting Group" := Resource."Gen. Prod. Posting Group";
        "VAT Prod. Posting Group" := Resource."VAT Prod. Posting Group";
        "Unit of Measure Code" := Resource."Base Unit of Measure";
        "Unit Price" := Resource."Unit Price";
    END;
Type::"G/L Account":
    BEGIN
```

```
GLAccount.GET("No.");  
  
GLAccount.CheckGLAcc();  
  
GLAccount.TESTFIELD("Direct Posting",TRUE);  
  
Description := GLAccount.Name;  
  
"Gen. Prod. Posting Group" := GLAccount."Gen. Bus. Posting Group";  
  
"VAT Prod. Posting Group" := GLAccount."VAT Bus. Posting Group";  
  
END;  
  
END;
```

4. Review the OnValidate triggers for the **Quantity**, **Unit Price**, and **Total Price** fields to understand the relationship between those fields.
 - a. When the user changes either the **Quantity** or the **Unit Price**, the system calculates the **Total Price**.
 - b. When the user changes the **Total Price**, the system calculates the **Unit Price**.
 - c. If you do not want to manually check the **Amount Rounding Precision** field in the **General Ledger Setup** table, you can omit the rounding precision parameter when it calls the **ROUND** function. This automatically reads the **Amount Rounding Precision field** from the **General Ledger Setup** table through function ReadRounding in Codeunit 1, ApplicationManagement.

Quantity – OnValidate and **Unit Price – OnValidate** contain the same code.

Quantity – OnValidate and Unit Price – OnValidate Triggers

```
"Total Price" := ROUND("Unit Price" * Quantity,0.01);
```

5. Review the Unit of Measure Code – OnValidate trigger to understand how units of measure influence the price calculations.
 - a. When a unit of measure is employed, there may be a conversion between different units of measure involved, too. Prices of items, resources, and so on, are always defined in the base unit of measure of the entity.

- b. In a transaction, when the user selects a unit of measure other than the base unit of measure, the price is recalculated to reflect the change. For example, suppose that the price in PCS is 10, and there are 4 PCS in a BOX. If the user selects BOX as the unit of measure for the transaction, then the price is recalculated to 40. This recalculation typically happens at the validation of the **Unit of Measure Code** field.
- c. If the **Type** is **Resource**, the Unit of Measure Code – OnValidate trigger retrieves the referenced resource, and then retrieves the specified unit of measure from the **Resource Unit of Measure** table to assign the **Qty. per Unit of Measure** field. Finally, it recalculates the **Unit Price** from the default resource price and **Qty. per Unit of Measure**. The VALIDATE function makes sure that the **Total Price** is recalculated, too.

Unit of Measure Code – OnValidate Trigger

```
CASE Type OF
Type::Resource:
    BEGIN
        Resource.GET("No.");
        IF "Unit of Measure Code" = " THEN BEGIN
            "Unit of Measure Code" := Resource."Base Unit of Measure";
        END;
        ResourceUOM.GET("No.", "Unit of Measure Code");
        "Qty. per Unit of Measure" := ResourceUOM."Qty. per Unit of Measure";
        "Total Price" := ROUND(Resource."Unit Price" * "Qty. per Unit of Measure");
    END;
Type::"G/L Account":
    BEGIN
        "Qty. per Unit of Measure" := 1;
    END;
END;
```

Demonstration: Reviewing the Seminar Comment Line Table and Pages

In Microsoft Dynamics NAV 2013, all master records and documents enable users to define free-text comments. Microsoft Dynamics NAV 2013 provides the consistent functionality for all comment features across the application. It tracks the date that the comment was entered, the user name, and the text of the comment. Comments do not perform any task but to let users manage indistinct parts of business processes by sharing additional unstructured information among them.

CRONUS International Ltd. plans to use comments to record special equipment and other requirements for their seminar registrations.

Demonstration Steps

1. Review the **Seminar Comment Line** table.
 - a. Comment tables always have a composite key that ends with the **Line No.** field to take advantage of the AutoSplitKey functionality.
 - b. Comment tables typically relate to several tables, and they manage those relationships through a combination of **Type** and **No.** fields.
 - c. The **Seminar Comment Line** table relates to the **Seminar Registration Header** table, and the **Posted Seminar Registration Header** table that you develop in the next module.
 - d. All comment tables contain a function that is named SetupNewLine that sets the **Date** field to **WORKDATE** if there are no other comment lines for the related entity. This guarantees that every first comment line on a specific date contains the value in the **Date** field. Any further comments on the same date do not contain any value in the **Date** field. This makes the comment sheets appear more organized and readable, and eliminates unnecessary information.

The **SetupNewLine** function sets the **Date** value to **WORKDATE** for any first comment line on a given date.

SetupNewLine Function

```
SeminarCommentLine.SETRANGE("Document Type","Document Type");  
  
SeminarCommentLine.SETRANGE("No.,"No.");  
  
SeminarCommentLine.SETRANGE("Document Line No.,"Document Line No.");  
  
SeminarCommentLine.SETRANGE(Date,WORKDATE);
```

```
IF NOT SeminarCommentLine.FIND('-') THEN
```

```
Date := WORKDATE;
```

2. Review the **Seminar Comment Sheet** page.
 - a. The comment sheet pages let users enter new comments for documents and master records.
 - b. The comment sheet page always calls the SetupNewLine function on its source table, from the OnNewRecord trigger. This makes sure that the new line is always configured according to the business rules that are coded in the SetupNewLine function.

Lab 3.2: Create Seminar Registration Pages

Scenario

You are Isaac, the developer for the partner company that is implementing Microsoft Dynamics NAV 2013 for CRONUS International Ltd. After Simon has reviewed your work on the tables and pages and made the necessary changes, you are now ready to complete the work and develop the pages for managing seminar registrations.

You must create the core document pages that consist of a Document page to create, view, and edit Seminar Registration documents, a subpage for seminar registration lines, and a list page for viewing all Seminar Registration documents immediately. You also have to develop a FactBox that shows details about a seminar. Use this FactBox to decorate the document and list page.

Finally, you have to add any necessary code, and link the pages that enable users to move from the list to the document, and to access the related information, such as comments and charges.

Exercise 1: Import and Review the Pages

Exercise Scenario

Isaac has created several pages. He was not sure about specific changes. Therefore he has not finished certain pages. Import the objects he provides. This includes:

- **Seminar Registration** page. Isaac did not complete it. You must add controls to this page.
- **Seminar Registration List** page.
- **Seminar Registration Subform** page.
- **Seminar Details Factbox** page. Use it to decorate the **Seminar Registration** and **Seminar Registration List** pages.
- **Seminar Charges** page.

Then, review the pages. Make sure that you correct any issues that you find.

Task 1: Importing the Starter Objects

High Level Steps

1. Import the Mod03\Labfiles\Lab 3.B - Starter.fob file.

Detailed Steps

1. Import the Mod03\Labfiles\Lab 3.B - Starter.fob file.
 - a. In **Object Designer**, click **File > Import**.
 - b. Locate the Mod03\Labfiles\Lab 3.B - Starter.fob file and click **Open**.
 - c. In the confirmation dialog box click **Yes** to import all objects.
 - d. In the **Import Objects** window, click **OK**.

Task 2: Reviewing the Objects

High Level Steps

1. Review the **Seminar Details FactBox** page.
2. Review the **Seminar Registration Subform** page.
3. Review the **Seminar Registration** page.

Detailed Steps

1. Review the **Seminar Details FactBox** page.
 - a. In **Object Designer**, click **Page**.
 - b. Locate page **123456717**, **Seminar Details FactBox**.
 - c. Click **Design** to open the page in the **Page Designer**.
 - d. Select the first empty row, and press SHIFT+F4 to open the **Properties** window for the page.
 - e. Verify that the PageType property is set to CardPart. FactBoxes must be of CardPart or ListPart type.
 - f. Verify that the SourceTable property is set to Seminar. This FactBox shows information about one seminar.
 - g. Close the **Properties** window.
 - h. Verify that all field controls are added under the ContentArea container control. There must not be a group control. CardPart pages do not use groups.
 - i. Close the **Page Designer**.
2. Review the **Seminar Registration Subform** page.
 - a. In **Object Designer**, click **Page**.
 - b. Locate page 123456711, Seminar Registration Subform.
 - c. Click **Design** to open the page in the **Page Designer**.
 - d. Select the first empty row, and press SHIFT+F4 to open the **Properties** window for the page.

- e. Set the PageType property to ListPart. Document subpages must be of the ListPart type.



Note: Developers frequently make the mistake and set the PageType property for subpages to List. Selecting an incorrect page type causes the RoleTailored client to show incomplete user interface.

- f. Verify that the SourceTable property is set to Seminar Registration Line.
 - g. Verify that the Caption property is set to "Lines". The Document page shows the Caption property of the subpage as the caption for the FastTab that shows the document lines. This FastTab should always be named **Lines**.
 - h. Set the AutoSplitKey property to **Yes**. All document subpages must set this property to enable Microsoft Dynamics NAV 2013 to automatically assign the values in the **Line No.** field.
 - i. Close the **Properties** window.
 - j. Verify that there is a group control of type Repeater under the ContentArea container control. The repeater contains the field controls that the subpage displays.
 - k. Close the **Page Designer**.
 - l. In the **Save Changes** dialog box, click **OK**.
3. Review the Seminar Registration page.
 - a. In **Object Designer**, click **Page**.
 - b. Locate page **123456710, Seminar Registration**.
 - c. Click **Design** to open the page in the **Page Designer**.
 - d. Select the first empty row, and press SHIFT+F4 to open the **Properties** window for the page.
 - e. Set the PageType property to Document. Document pages must not be of the Card type.



Note: You must always select the correct page type. Pages with incorrect types may not display correctly. They can also cause more serious issues.

- f. Verify that the SourceTable property is set to Seminar Registration Header.
- g. Close the **Properties** window.
- h. Verify the content of the page. There are three group controls that represent FastTabs, and a FactBoxArea control. The page resembles a card page. It does not include the subpage.



Note: Document pages must include the subpage. You add the subpage in the next exercise.

- i. Select the **Customer Details FactBox** page part control.
 - j. Press SHIFT+F4 to show the Properties window for the control.
 - k. Check the SubPageLink property. It does not contain a value.
-



Note: The SubPageLink property links a page part to the parent page. It enables the page part to show the information related to the record shown in the parent page.

- l. Close the **Page Designer**.
- m. In the **Save Changes** dialog box, click **OK**.

Exercise 2: Completing the Document Pages

Exercise Scenario

Now that you have completed the review, you can complete the development of the seminar registration management pages.

Task 1: Completing the Seminar Registration Page

High Level Steps

1. Design the **Seminar Registration** page.
2. Add the subpage for the **Seminar Registration Subform** page, and name the control SeminarRegistrationLines.
3. Link the **SeminarRegistrationLines** page part to the main page.
4. Add the **Seminar Details FactBox** to the page.
5. Link the **Seminar Details FactBox** page part to the main page.
6. Link the **Customer Details FactBox** page part to the **SeminarRegistrationLines** page part.
7. Save, compile, and then close the page.

Detailed Steps

1. Design the **Seminar Registration** page.
 - a. In **Object Designer**, click **Page**.
 - b. Locate page **123456710, Seminar Registration**.
 - c. Click **Design** to open the page in the **Page Designer**.

2. Add the subpage for the **Seminar Registration Subform** page, and name the control SeminarRegistrationLines.
 - a. Select the Seminar Room group.
 - b. Press F3 to insert a new row.
 - c. In the Type column, select Part.
 - d. In the SubType column, select **Page**.
 - e. Click the **Left** button one time or press SHIFT+ALT+LEFT to reduce line indentation by one level.
 - f. Press SHIFT+F4 to open the **Properties** window for the part.
 - g. In the PagePartID property, type "Seminar Registration Subform".
 - h. In the **Name** property, type "SeminarRegistrationLines".

3. Link the **SeminarRegistrationLines** page part to the main page.
 - a. In the SubPageLink property click the **AssistEdit** button to open the **Table Filter** window.
 - b. In the Field column, type "Document No."
 - c. In the Type column select FIELD.
 - d. In the Value column, type "No."
 - e. Click **OK** to accept the changes and close the **Properties** window.

4. Add the **Seminar Details FactBox** to the page.
 - a. Select the row containing the **Customer Details FactBox** page part control.
 - b. Press F3 to insert a new row.
 - c. In the Type column, enter "Part". Verify that the SubType value is set to Page automatically.
 - d. Press SHIFT+F4 to show the **Properties** window for the new page part control.
 - e. Set the PagePartID property to "Seminar Details FactBox".
 - f. Do not close the **Properties** window.

5. Link the **Seminar Details FactBox** page part to the main page.
 - a. In the SubPageLink property, click the **AssistEdit** button to open the **Table Filter** window.
 - b. In the Field column, type "No."
 - c. In the Type column, select FIELD.
 - d. In the Value column, type "Seminar No."
 - e. Press **OK** to close the **Table Filter** window and apply the filter.
 - f. Close the **Properties** window.

6. Link the **Customer Details FactBox** page part to the **SeminarRegistrationLines** page part.
 - a. Select the **SeminarRegistrationLines** page part.
 - b. Press SHIFT+F4.
 - c. Copy the value of the ID property.
 - d. Close the **Properties** window.
 - e. Select the **Customer Details Factbox** page part.
 - f. Press SHIFT+F4.
 - g. In the ProviderID property value, paste the value that you copied from the ID property of the **SeminarRegistrationLines** page part.



Note: The **ProviderID** property specifies the subpage that provides the source table for the subpage link. If you leave it empty, then the source table for the link is the source table of the main page. If you specify the ID of an existing page part control, then the source table for the link is the source table of the specified page part.

- h. In the SubPageLink property, click the **AssistEdit** button to open the **Table Filter** window.
 - i. In the Field column, type "No."
 - j. In the **Type** column, select FIELD.
 - k. In the Value column, type "Bill-to Customer No."
 - l. Press **OK** to close the **Table Filter** window and apply the filter.
 - m. Close the **Properties** window.
7. Save, compile, and then close the page.
 - a. Click **File > Save**.
 - b. In the **Save** dialog box, make sure that the **Compiled** check box is selected, and then click **OK**.
 - c. Close the **Page Designer** window.

Module Review

Module Review and Takeaways

The “Documents” module described how to create the tables and pages that you must have to register participants in seminars, together with how to create code to improve usability and data validation.

You learned about documents, one of the standard features in Microsoft Dynamics NAV 2013 that lets users enter transaction information in an easy-to-use and intuitive manner. You also reviewed several objects and analyzed their structure and code to understand the most common logic and code patterns that are consistently applied to documents across Microsoft Dynamics NAV 2013.

The next step is to take this transaction information and create a posting routine that can certify participants and create ledger entries for completed courses. You also can post invoices to customers.

Test Your Knowledge

Test your knowledge with the following questions.

1. When importing objects from a text file, Object Designer does not ask you if there are any conflicts and always replaces objects if the objects of the same type and ID already exist in the application?

() True

() False

2. Text constants and variables of type Text or Code can be Multilanguage.

() True

() False

3. Which function can you use on a list page to note the records that the user has selected on the page, mark those records in the table, and set the filter to marked-only?

4. What kind of a table is table **2000000001, Object**?

- () Special table
- () Virtual table
- () Temporary table
- () System table

5. What is the difference between virtual tables and system tables?

6. COMMIT does not affect temporary tables and ERROR does not roll back any changes to temporary tables.

- () True
- () False

7. Which page type do you use for document lines subpage?

8. Which property do you need to set on a FactBox part to establish a link between a subpage and the FactBox, so that when the record changes in the subpage, the FactBox is updated?

Test Your Knowledge Solutions

Module Review and Takeaways

1. When importing objects from a text file, Object Designer does not ask you if there are any conflicts and always replaces objects if the objects of the same type and ID already exist in the application?

True

False

2. Text constants and variables of type Text or Code can be Multilanguage.

True

False

3. Which function can you use on a list page to note the records that the user has selected on the page, mark those records in the table, and set the filter to marked-only?

MODEL ANSWER:

CurrPage.SETSELECTIONFILTER

4. What kind of a table is table **2000000001, Object**?

Special table

Virtual table

Temporary table

System table

5. What is the difference between virtual tables and system tables?

MODEL ANSWER:

Virtual tables do not store physical information in the database and they are created at the run time by the system. You cannot insert, modify, or delete information in virtual tables. System tables are created by the system, but they store physical information in the database. You can customize them, or insert, modify, or delete information contained in them.

6. COMMIT does not affect temporary tables and ERROR does not roll back any changes to temporary tables.

True

False

7. Which page type do you use for document lines subpage?

MODEL ANSWER:

ListPart

8. Which property do you need to set on a FactBox part to establish a link between a subpage and the FactBox, so that when the record changes in the subpage, the FactBox is updated?

MODEL ANSWER:

ProviderID

