CHAPTER 3: REGISTRATIONS

Objectives

The objectives are:

- Importing and exporting objects as text files.
- Supporting multi-language functionality.
- Using main and sub-forms.
- Creating matrix forms.
- Using virtual tables.
- Using temporary tables.

Introduction

Now that the master tables and forms are in place, the next step is to implement functionality to allow users to perform transactions with the master data. As with the tasks in the previous chapter, this will require frequent referral to planning documentation, and we'll employ analysis, design, and implementation phases.

Before starting the analysis phase, however, there are some concepts and features that will be necessary during the implementation phase. They are:

- Working with objects as text files
- Multi-language support
- Main/sub forms
- Maxtrix forms
- Table types
- Useful C/AL functions

By the end of this chapter, the tables and forms necessary for registering participants in seminars will be in place.

Prerequisite Information

Before getting into the subject of registration, there is some information that should be reviewed.

Working with Objects as Text Files

Analyzing and modifying objects can sometimes be much easier when the objects are in text file format. Any object in the **Object Designer** can be exported as text and imported back into Microsoft Dynamics™ NAV as an un-compiled object.

To export one or more objects as a text file:

1. Select the object or objects in the Object Designer.
2. Click FILE→EXPORT in the menu bar.
3. In the Export Objects dialog window, select Text Format as the Save as type. Enter a file name.
4. Click **OK**. One text file is created containing the text representations for the object or objects.

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

```
OBJECT Table 123456703 Instructor
```

OBJECT-PROPERTIES Section

The next section contains the object's date, time, and version properties.

PROPERTIES Section

The following section includes the object's triggers (those that contain code) and properties (those that do not contain default values).

SUBOBJECTS Section

The next section lists the sub-objects. For tables, these consist of FIELDS and KEYS. This section contains the sub-object properties that do not contain default values and the triggers that contain code.

CODE Section

The last section contains the global variables and functions for the object. Once the text files has been changed and saved, import the text file back into Microsoft Dynamics NAV.

1. Open the Object Designer.
2. Click FILE→IMPORT.
3. Select the appropriate file in the Import Objects dialog window and click **Open**. The object or objects are imported. The objects must be compiled before they can be used.

The Import Worksheet

Finally, it is important to note that the **Import Worksheet** cannot be used when importing a text file, which means:

- No warning will be received about overwriting existing tables
- There is no opportunity to skip the import of some objects
- There is no opportunity to merge objects

Multilanguage Functionality in Text Messages

When creating messages for the user, make sure that the text and the object names in the messages are enabled for multi-language functionality.

Error and text messages must be entered as text constants so that they can be easily translated. The **C/AL Globals** and **C/AL Locals** forms have a **Text Constants** tab with a hidden column, ConstValueML, which displays all the languages for a text constant. Text constants replace the use of hard-coded, language-dependent text strings.

Once it has been assigned as a text constant, the message can then be used in code as in the following example:

```
ERROR(Text001); //where Text001 is defined as Text Constant
in the global or local variables
```

The following example uses the FIELDCAPTION function within an error message. Text Constant 025 value is "Please enter "Yes" in %1 and/or %2 and/or %3".

```
ERROR(Text025,FIELDCAPTION(Receive),
  FIELDCAPTION(Invoice), FIELDCAPTION(Ship));
```

When the code is run, the error message translates into: Please enter "Yes" in Receive and/or Invoice and/or Ship.

When referring to fields in a message, the code should refer to the caption rather than the name of the field. By using the FIELDCAPTION function, the current caption of a field is returned as a string. The field property **CaptionML** must be populated to enable this functionality. The TABLECAPTION function can be used to return the table name. The following code returns the caption of the **Document Type** field.

```
SalesLine.FIELDCAPTION("Document Type");
```

Main/Sub Forms

A main/subform is a combination of a card form with a tab control and a subform that contains a table box. For example, re the view the **Sales Invoice** form (Form 43), which is used to create, view, or modify sales invoice documents. The Sales Invoice form has both a tab control box, and a subform. This tab control on the main form is associated with the header table, Sales Header (Table 36). The **HorzGlue** and **VertGlue** property for the tabcontrol should be set to **Both** and **Top** respectively.

In the properties of the subform, notice that the "SubFormID" property is set to the "Sales Invoice Subform" (Form 47). To link the subform records to the current **Sales Header** record, the **SubFormLink** property is set. The **HorzGlue** and **VertGlue** property for the subform is set to **Both**.

The **Sales Invoice** Subform indicates that the form is associated to the line table, **Sales Line** (Table 37). The form is a tabular form, essentially a table box that displays certain fields from the table. Notice that the key fields from the **Sales Line**, **Document Type**, **Document No**., and **Line No.**, are NOT displayed. As this is a worksheet type form, it follows these standards*:*

- The key fields from the **Sales Line, Document Type, Document No.,** and **Line No**., are NOT displayed.
- **HorzGlue** and **VertGlue** properties for the tablebox on this subform are set to **Both**.
- **AutoSplitKey** property set to **Yes**.

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

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 Form	Name of Document Represented	Sales Invoice (Form 43)
Document Subform	Name of Document Represented + "Subform"	Sales Invoice Subform (Form 47)

To follow Microsoft Dynamics NAV standards, the **Seminar Registration** window in this chapter is similar to the existing **Sales Invoice** window. For more information about design standards, refer to the Application Designer's Guide.

Matrix Forms

A matrix form is created by using the **MatrixBox** form control and should be used in cases where there is a many-to-many relationship between two tables. In Microsoft Dynamics NAV, it is used in the system to show totals by time period. Form 113, **Budget**, is an example of a Matrix form used for this purpose.

It is important to note that Maxtrix forms are not supported by the three-teir architecture in Microsoft Dynamics NAV 5.1. Functionality in existing applications that is implemented with Maxtrix forms will therefore have to be re-worked in order to transistion the application to NAV 5.1.

A matrix box is a composite control (comprised of more than one control) that can show information from several tables at the same time. The first two tables are the vertical and the horizontal table of the **MatrixBox** control. In the matrix part of the control, each cell can be used to display information that is calculated on the basis of fields in these two tables, or on information that is retrieved from other tables (with values from the first two tables being used to select records).

Each cell in the matrix is the intersection of a record from the vertical and the horizontal table. The part to the left of the vertical divider bar displays records from the vertical table, the table that is the source table of the form, in a way similar to an ordinary **TableBox** control. To the right of the divider bar is the matrix itself. Above the matrix, the records from the horizontal table are displayed (in the style that is normally used for the labels in a table box). The title area of this area is the matrix heading. The horizontal table is called the matrix source table.

Like other controls, the **MatrixBox** control has its own triggers. These triggers are:

- OnFindRecord
- OnNextRecord, OnAfterGetRecord
- OnAfterGetCurrRecord
- OnBeforePutRecord

It can be confusing to create a matrix box for the first time. The best way to learn the process is to create a sample. These steps describe how to create a simple multiplication table:

1. Create a new blank form and select the table to be used as the vertical table as the source. In this example, use the **Integer** table.
2. From the toolbox, add a matrix box to the form.
3. In the **Property** window, set the **Name** property of the **MatrixBox** control to Mul**t**Table, and set the **HorzGlue** and **VertGlue** properties to **Both**. Set the **MatrixSourceTable** property to the **Integer** table.
4. Insert a text box with a source expression of the **No.** field.
5. Add a text box without a source expression to the empty part of the **MatrixBox** control. Check the **InMatrix** property of this text box – it should be **Yes**.
6. Add a text box without a source expression to the last empty part (the matrix heading) of the **MatrixBox** control. Check the **InMatrixHeading** property of this text box – it should be **Yes**.
7. Add a source expression to the text box in the matrix heading (the last one added). The source expression should point to a field from the **MatrixSourceTable**. If the matrix box was named **MultTable**, and the horizontal table is the Integer table, it could be: CurrForm.MultTable.MatrixRec.Number.
8. Add a source expression to the text box in the matrix. Here, it could be: CurrForm.MultTable.MatrixRec.Number * Number.
9. Test the form by running it.

Types of Tables

Thus far, regular database tables have been used to implement the seminar module. The upcoming exercises make use of other kinds of tables.

Virtual Tables

A virtual table contains information provided by the system. C/SIDE provides access to a number of virtual tables. They work in much the same way as regular database tables, but the information in them cannot be changed – they contain read-only information. Virtual tables are not stored in the database as normal tables are, but are computed by the system at run time.

Because a virtual table can be treated the same as an ordinary table, the same methods can access their information. For example, use filters to get subsets or ranges of integers or dates. Here is a list of virtual tables that are available in Microsoft Dynamics NAV 5.0:

- Date
- Integer
- File
- Drive
- Monitor
- Session
- Database File
- Table Information
- Field
- Server
- Windows Object
- Windows Group Member
- SID – Account ID
- User SID
- Along with many others

The **Object** and **Field** tables are particularly important when working with C/AL.

Because the virtual tables are not stored in the database, they cannot be viewed directly. To view a virtual table, create a tabular form. The virtual tables have the highest numbers in the table list (2000000000+). The Application Designer's Guide has further information on a number of the virtual tables.

The **Date** virtual table will be used. The **Date** virtual table provides easy access to days, weeks, months, quarters, and years. This table has three fields:

- **Period Type**: Days, weeks, months, quarters, or years
- **Period Start**: The date of the first day in the period
- **Period End**: The date of the last day in the period

Temporary Tables

A temporary table is a memory-based table. Temporary tables do not exist in the database, and – unlike virtual tables – are not read-only. A temporary table can do almost anything that a normal database table can, but the information in the table is lost when the table is closed.

The write transaction principle that applies to normal database tables does not apply to temporary tables. Refer to the Application Designer's Guide more information on the transaction principle,.

The advantage of using a temporary table is that all interaction with a temporary table takes place on the client. This reduces the load both on the network and on the server. To perform many operations on data in a specific table in the database, load the information into a temporary table for processing. Because all operations are local, this speeds up the process.

Creating a temporary table is similar to creating a record variable:

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

The temporary table variable is now ready to be used just like any other record variable.

System Tables

System Tables are stored in the database as in regular tables, but they differ in that they are created automatically by the system. System table data can be read, writen, modified, and deleted. The eight system tables in C/SIDE are primarily used to manage security and permissions. The Application Designer's Guide has detailed information on each one of the System Tables.

Additional Functions

Useful Functions Review

In Development I, the following instructions were introduced. These functions are used later in this chapter. Refer to the C/SIDE Reference Guide for more information.

- CALCDATE
- DATE2DMY
- DATE2DWY
- ROUND
- RUNMODAL
- CONFIRM
- MESSAGE
- ERROR
- TESTFIELD
- WORKDATE
- VALIDATE
- FORMAT
- COUNT

Form Functions

Form functions are called through the **CurrForm** variable. This variable is a reference to the instance of the current form. The following lists a few of the most useful form functions available. See the C/SIDE Reference Guide for a complete list.

CurrForm.UPDATE: Use this function to save the current record and then update the controls in the form. If the **SaveRecord** parameter is set to FALSE, this function does not save the record before the system updates the form.

CurrForm.SETSELECTIONFILTER: Use this function to have the system note the records the user has selected on the form, mark those records in the table specified, and set the filter to "marked only".

CurrForm.CLOSE: Use this function to close the current form.

CurrForm.EDITABLE: Use this function to return the current setting of the Editable property and to change the setting of the property. Most, but not all properties can also be changed from within code. This property is especially useful for using the same form for viewing and editing, but only allow some users to edit the records.

Control Functions

Microsoft Dynamics NAV also has a number of functions available for form controls. To access the functions of a control, name the control first, and use the following syntax:

```
CurrForm.<ControlName>.<Function>
```

The following list contains information about the most useful control functions, but it is not a complete list. Refer to the C/SIDE Reference Guide for more information on control functions.

EDITABLE: Use this function to return the current setting of the **Editable** property and to change the setting of the property. Other properties can also be changed from within code, but not all of them can.

VISIBLE: Use this function to return the current setting of the **Visible** property and to change the setting of the property. This property is especially useful if only some users are allowed to view a particular field.

Be aware that if a text box is made not visible in a table box control, the user can change that property by going to **View**, **Show Columns**.

UPDATEEDITABLE: Use this function to make a text box not editable or editable dynamically. It does not change the Editable property. The next time the user enters the text box, this function must be called again to make it editable or not editable. This function can only be called from the **OnBeforeInput** trigger of the control.

UPDATEFONTBOLD: Use this function to dynamically make a field **Bold** or not. This function can only be called from the **OnFormat** trigger of the control.

Solution Analysis

Now that the master data is complete, make it possible for users to apply the master data to daily transactions related to registrations. The functional requirements define the role of registrations this way:

If a customer wants to register one or more participants for a seminar, enter the relevant information into a registration form.

Therefore, the main process related to registrations is managing seminar registrations. There is an additional optional requirement to manage seminar planning. This use case and related exercise is covered in the Additional Exercises section.

Based on the results of the diagnosis, the analysis and implementation phases of this chapter are the following use case:

- Managing Seminar Registration

The client's functional requirements describe the process of seminar registration in the following way:

Customer can register one or more participants for a seminar. Each registration is assigned a job number. It must be possible to assign additional expenses to an instance of a seminar, such as catering expenses or equipment rentals. The registration information much include how the seminar should be invoiced (for example, whether to include expenses or catering).

Additional comments for each seminar can be entered to allow for things like necessary equipment or other particular requirements.

Therefore, registering a participant in a seminar must accomplish or allow these things:

- A place is reserved for the participant in the seminar
- Invoicing information is gathered and stored
- The appropriate facilities are reserved
- The seminar and participant information can be tracked with a job number
- Additional expenses can be invoiced as well

The next step is to define the process of creating a new seminar. Once the details of a seminar are established, a seminar manager can create an instance of the seminar with this information:

- An instructor
- An available and properly equipped room
- Any charges or additional comments

Once the seminar has been created, a customer training manager can add participants to the seminar. This scenario is illustrated in Figure 3-1.

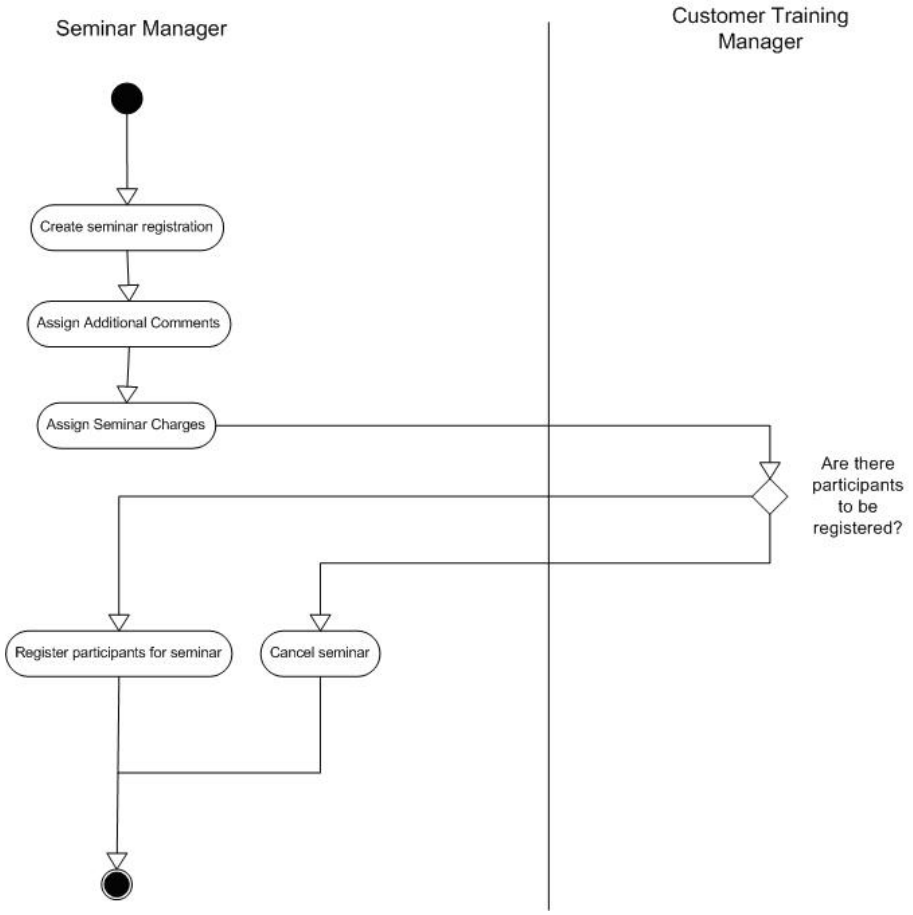


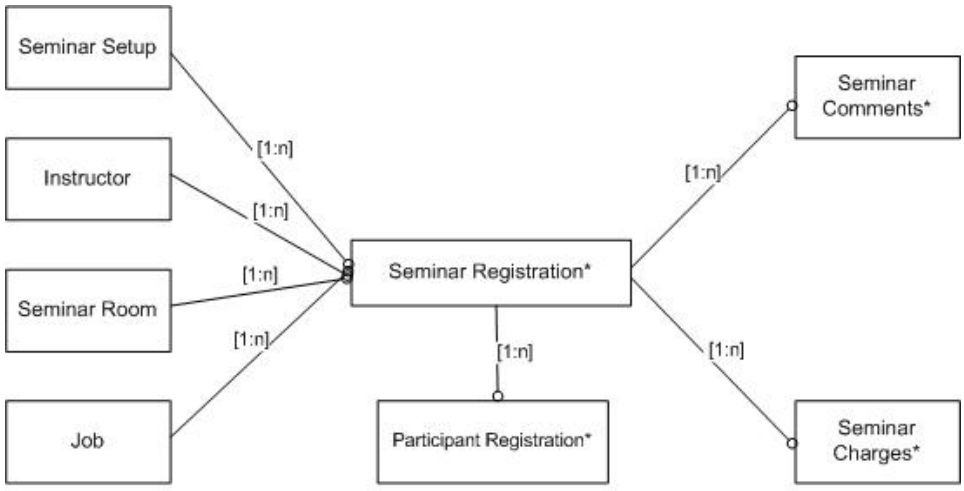
FIGURE 3–1: THE SEMINAR MANAGEMENT FLOWCHART

Solution Design

As can be seen from the analysis of the seminar registration management process, there are two main processes:

- The creation of seminar instances
  - Define seminar registration details
  - Assign additional comments
  - Assign seminar charges
- The registration of participants

Figure 3-2 shows the relationship between the system tables and the forms used to control these. Here, the prerequisite tables are shown to the left, the main processing tables are in the middle and the sub-process tables are to the right.



* new table

FIGURE 3-2: THE SEMINAR REGISTRATION RELATIONSHIPS

GUI Design

The forms for the seminar registration and the navigation between them reflect the relationships shown in the previous diagram. Start by defining the simplest forms first so they can be integrated with the more complex forms at the end of the GUI design.

The **Seminar Comment List** form displays the comments for a seminar (Figure 3-3).

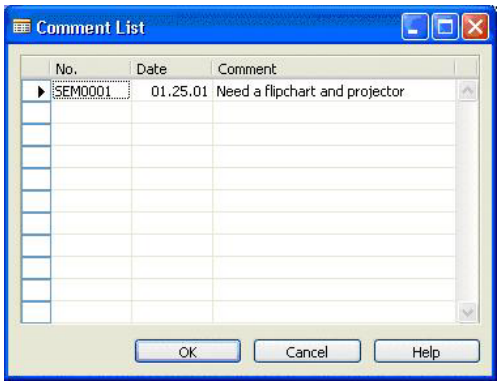


FIGURE 3–3: SEMINAR COMMENT LIST FORM (123456707)

The **Seminar Comment Sheet** form, shown in Figure 3-4, nables the entry of comments for a seminar.

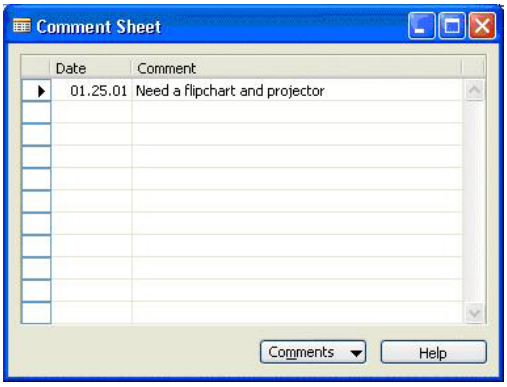


FIGURE 3–4: SEMINAR COMMENT SHEET FORM (123456706)

The **Seminar Charges** form (Figure 3-5) enables the entry of charges for a seminar.

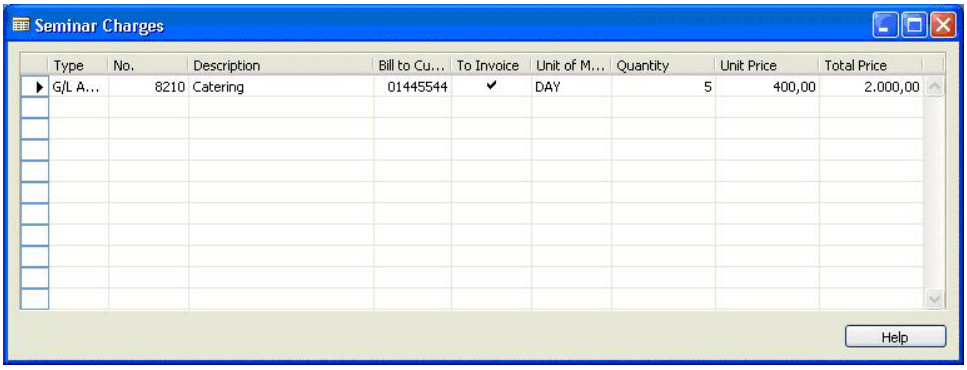


FIGURE 3-5: THE SEMINAR CHARGES FORM (123456724)

Since the seminar registration and participant registration are so closely linked, it is best to handle them in one window with header and lines as shown in Figure 3-6. This form actually consists of two forms, a header form, and a subform that contains the lines.

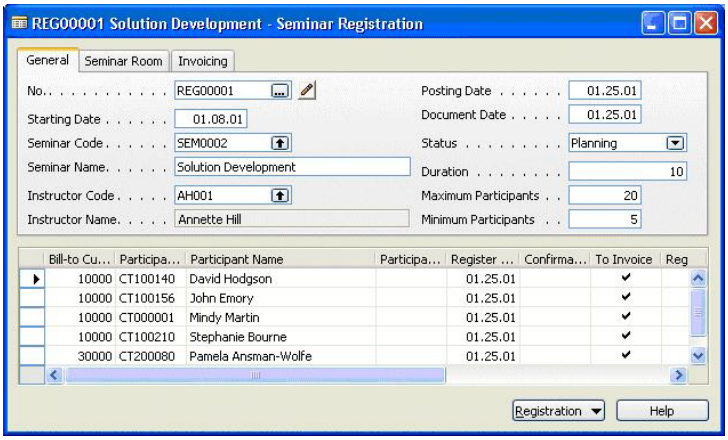


FIGURE 3-6: THE SEMINAR REGISTRATION FORM (MAIN 23456710, SUBFORM 123456711)

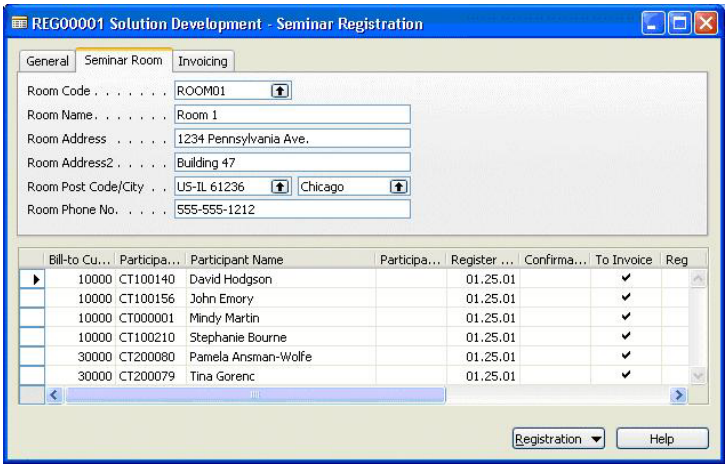


FIGURE 3-7: THE SEMINAR REGISTRATION FORM, SEMINAR ROOM TAB

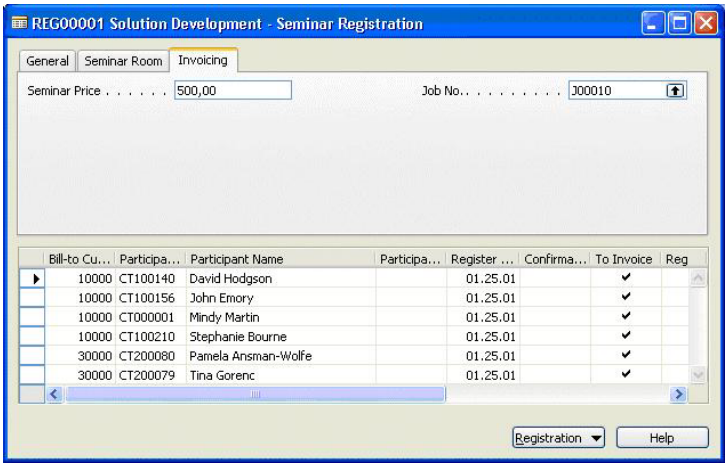


FIGURE 3-8: THE SEMINAR REGISTRATION FORM, INVOICING TAB

The **Seminar Registration List** form (Figure 3-9) displays the seminar registrations.

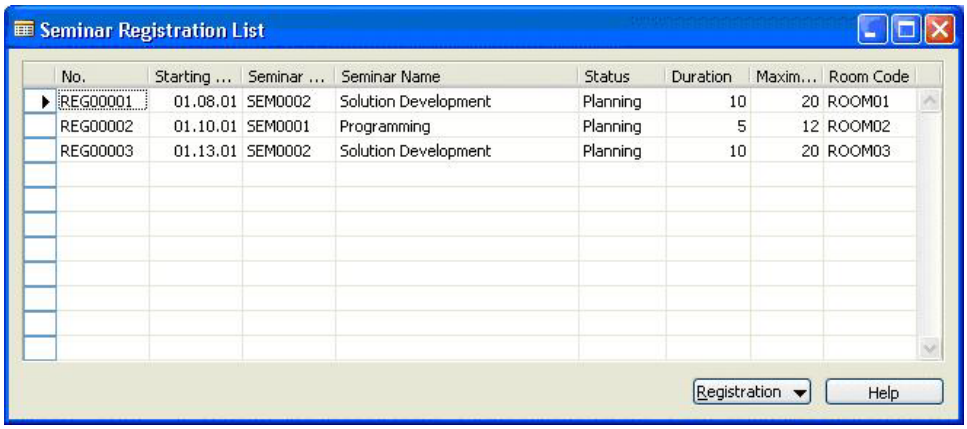


FIGURE 3-9: THE SEMINAR REGISTRATION LIST FORM (123456713)

Table Design

The following tables are required for registrations:

Table 123456710 Seminar Registration Header holds the information for one instance of a Seminar, which is referred to as a registration.

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 related to the seminar registration. These are in addition to the individual participant charges of the Seminar Registration Line table.

Lab 3.1 - Creating the Tables and Forms for Seminar Registration

The first task is to create the **Seminar Comment Line** table, which contains comments about the seminar registration, if any.

The fastest way to create this table is by exporting an existing table as a text file, making the appropriate changes to the text file and importing it back into Microsoft Dynamics NAV.

1. Create table 123456704 **Seminar Comment Line** by exporting the standard table **Sales Comment Line**, modifying it, and importing it back into Microsoft Dynamics NAV.
   - To do this, first export the **Sales Comment Line** table (44) as a text file using the steps given earlier in the chapter.
   - Open the file in Notepad. Change the object number to 123456704 and the object name to "Seminar Comment Line."
   - Replace all instances of the word "Sales", with, "Seminar."
   - Replace the **OptionString** values for the **Document Type** field to "Seminar Registration, Posted Seminar Registration" as shown in the table design.
   - Set the **LookupFormID** and **DrillDownFormID** properties to form 123456707.
   - Save the text file, import it into Microsoft Dynamics NAV, and compile it.

The completed table will have the following fields:

No.	Field Name	Type	Length	Comment
1	Document Type	Option		Options: Seminar Registration, Posted Seminar Registration.
2	No.	Code	20	
3	Line No.	Integer		
4	Date	Date		
5	Code	Code	10	
6	Comment	Text	80	

Note that the key for the table is **Document Type**, **No.**, **Line No.** This is also imported with the table.

The next step is to create the **Seminar Coment List** and **Seminar Comment Sheet** forms. These forms are used to enter and display comments about the registration, if any.

2. Create the **Seminar Comment List** form (123456707) with the fields **No.**, **Date,** and **Comment** as shown in the GUI design. Set the property to specify that this form is not editable.
3. Create the **Seminar Comment Sheet** form (123456706) with the fields **Date**, **Comment,** and **Code (not visible)**.
   - Set the properties to automatically split the key, allow multiple new lines, and to specify that the program should not insert the record until the user has left the record.
   - Add a menu button and menu item as follows:

Menu Button	Option	Comment
Comments	List (f5)	Opens the lookup form.

   - In the appropriate form trigger, run the function **SetUpNewLine** whenever the user enters a new record. This function was imported with the **Seminar Comment Line** table.

The next step is to create the **Seminar Registration Header** table, which contains information about specific offerings of the seminar.

4. Create table 123456710 **Seminar Registration Header** with the following fields:

No.	Field Name	Type	Length	Comment
1	No.	Code	20	
2	Starting Date	Date		
3	Seminar Code	Code	20	Relation to the Seminar table.
4	Seminar Name	Text	50	
5	Instructor Code	Code	10	Relation to Instructor table.
6	Instructor Name	Text	50	FlowField; The CalcFormula should look up the **Name** field on the Instructor table and must not be editable.
7	Status	Option		Options: Planning, Registration, Closed, Canceled.
8	Duration	Decimal		Decimal Places 0:1

No.	Field Name	Type	Length	Comment
9	Maximum Participants	Integer		
10	Minimum Participants	Integer		
11	Room Code	Code	20	Relation to Seminar Room table.
12	Room Name	Text	30	
13	Room Address	Text	30	
14	Room Address2	Text	30	
15	Room Post Code	Code	20	Relation to Post Code table. There should be no validation or testing of the table relation.
16	Room City	Text	30	
17	Room Phone No.	Text	30	
18	Seminar Price	Decimal		AutoFormatType=1
19	Gen. Prod. Posting Group	Code	10	Relation to Gen. Product Posting Group table.
20	VAT Prod. Posting Group	Code	10	Relation to VAT Product Posting Group table.
21	Comment	Boolean		FlowField; The CalcFormula should check whether lines exist on the Seminar Comment Line table for the current Seminar Registration Header. Must not be editable.
22	Posting Date	Date		
23	Document Date	Date		
24	Job No.	Code	20	Relation to Job table.
25	Reason Code	Code	10	Relation to Reason Code table.
26	No. Series	Code	10	Relation to No. Series table. Must not be editable.
27	Posting No. Series	Code	10	Relation to No. Series table.
28	Posting No.	Code	20	

The primary key for this table is the **No.** field, with a secondary key of **Room Code**. The sum index field for the secondary key is **Duration**. Set the property to specify form 123456713 as the lookup form.

The next step is to create the **Seminar Charge** table, which contains information for invoicing.

5. Create table 123456712 **Seminar Charge** with the following fields:

No.	Field Name	Type	Length	Comment
1	Seminar Registration No.	Code	20	Relation to table 123456710 Seminar Registration Header. Must not be blank.
2	Line No.	Integer		
3	Job No.	Code	20	Relation to table 167 Job
4	Type	Option		Options: Resource, G/L Account
5	No.	Code	20	If Type=Resource, relation to table 156 Resource If Type=G/L Account, relation to table 15 G/L Account.
6	Description	Text	50	
7	Quantity	Decimal		Decimal Places 0:5
8	Unit Price	Decimal		AutoFormatType = 2 Minimum value of 0.
9	Total Price	Decimal		Must not be editable. AutoFormatType=1
10	To Invoice	Boolean		Initial value is Yes.
11	Bill-to Customer No.	Code	20	Relation to Customer table.
12	Unit of Measure Code	Code	10	If Type=Resource, relation to the **Code** field of table 205 Resource Unit of Measure table, where the Resource No. = No. Otherwise, relation to table 204 Unit of Measure.
13	Gen. Prod. Posting Group	Code	10	Relation to table 251 Gen. Product Posting Group.
14	VAT Prod. Posting Group	Code	10	Relation to table 324 VAT Product Posting Group.
15	Qty. per Unit of Measure	Decimal		
16	Registered	Boolean		Must not be editable.

The primary key is **Seminar Registration No.**, **Line No**. and a secondary key of **Job No.**

The next step is to create the **Seminar Charges** form, which will allow the input and display of invoice information.

6. Create the **Seminar Charges** form (123456724) with the following fields: **Type**, **No.**, **Description**, **Bill-to Customer No.**, **To Invoice**, **Unit of Measure Code**, **Quantity**, **Unit Price,** and **Total Price**. Set the property to automatically split the key.
7. Create the **Seminar Registration Line** table (123456711) with the following fields:

No.	Field Name	Type	Length	Comment
1	Document No.	Code	20	Relation to Seminar Registration Header table.
2	Line No.	Integer		
3	Bill-to Customer No.	Code	20	Relation to Customer table.
4	Participant Contact No.	Code	20	Relation to Contact table.
5	Participant Name	Text	50	Flowfield; Lookup the value based on the Participant Contact No. Must not be editable.
6	Register Date	Date		Must not be editable.
7	To Invoice	Boolean		Initial value is Yes.
8	Participated	Boolean		
9	Confirmation Date	Date		Must not be editable.
10	Seminar Price	Decimal		AutoFormatType = 2
11	Line Discount %	Decimal		Decimal places 0:5; The minimum value is 0 and the maximum is 100.
12	Line Discount Amount	Decimal		AutoFormatType = 1
13	Amount	Decimal		AutoFormatType = 1
14	Registered	Boolean		Must not be editable.

The primary key is **Document No.**, **Line No.**

The next steps are to create the **Seminar Registration** for and subforms.

8. Create subform 123456711 Seminar Registration Subform according to the GUI design. The fields to include on the subform are: **Bill-to Customer No.**, **Participant Contact No.**, **Participant Name**, **Participated**, **Register Date**, **Confirmation Date**, **To Invoice**, **Registered**, **Seminar Price**, **Line Discount %**, **Line Discount Amount** and **Amount**.
   - Set the width, height, and positioning properties for the form and the table box so that there is no empty space around the table box.
   - Set the properties for the **Line Discount %** and **Line Discount Amount** fields so that they are blank if the value is 0.
   - Set the form property to specify that the program automatically creates a key for a newly inserted record.

9. Create the **Seminar Registration** main form (123456710) as shown in the GUI design. Include the three tabs: **General**, **Seminar Room** and **Invoicing**, and the subform box.
   - Set the subform box properties so that the width and height are the same as that of the subform form. This means that the subform expands and contracts both horizontally and vertically when the user resizes the form. There must be no border. Give the subform the name **SeminarRegistrationLines**.
   - Set the subform box property to set the Seminar Registration Subform as the **SubFormID**. Set the property to link the subform to its table.
   - Set the Drilldown property of the **Instructor Name** to No.
   - Add menu button and menu items as follows:

Menu Button	Option	Comment
Registration	List (f5)	Opens the lookup form.
	Comments	Opens the **Seminar Comment Sheet** form (123456706). The link should run whenever there is an update.
	<Separator>	
	Charges	Opens the **Seminar Charges** form (123456724) for the corresponding **Seminar Registration No.** The link should run whenever there is an update.

10. Create the **Seminar Registration List** form (123456713) form with the following fields: **No.**, **Starting Date**, **Seminar Code**, **Seminar Name**, **Status**, **Duration**, **Maximum Participants** and **Room Code**.
    - Set the property to specify this form as not editable.
    - Add the menu button and menu item as follows:

Menu Button	Option	Comment
Registration	Card (SHIFT + F5)	Opens form 123456710 Seminar Registration for the selected entry.

Lab 3.2 - Adding Code for Seminar Charges

In the **Seminar Charge** table, enter code to perform the following tasks:

1. When a record is inserted into the table, the program gets the corresponding **Seminar Registration Header** record and sets the **Job No.** field to that of the **Seminar Registration Header**.
2. When a user deletes a record, the program checks that the **Registered** field is false. Users cannot delete registered seminars.

HINT: *Use the TESTFIELD function to test the Registered value.*

---

3. When a user enters or changes a value in the **Job No.** field, the program checks that the corresponding record in the Job table is not blocked and that the status of the job is Order.
4. When a user enters or changes a value in the **Type** field, the program sets the **Description** to blank.
5. When a user enters or changes a value in the **No.** field, the program checks the following:
   - If the **Type** is Resource:
     - Tests that the corresponding Resource is not blocked.
     - Tests that the **Gen. Prod. Posting Group** field is filled on the Resource record.
     - Sets the **Description** field of the Seminar Charge table to the Name from the Resource.
     - Sets the **Gen. Prod. Posting Group**, the **VAT Prod. Posting Group**, the **Unit of Measure Code**, and the **Unit Price** to the corresponding values in the Resource record.
   - If the **Type** is G/L Account:
     - Gets the corresponding G/L Account record and runs the CheckGLAcc function.
     - Tests that Direct Posting is TRUE for the G/L Account.
     - Sets the **Description** field of the Charge table to the Name of the G/L Account.
     - Sets the **Gen. Prod. Posting Group** and **VAT Prod. Posting Group** to the corresponding values in the G/L Account record.

Solution: The code in the No. – **OnValidate** trigger should look like this:

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

6. When a user enters or changes the value in the **Quantity** field, the program calculates the **Total Price** field by multiplying the Unit Price by the Quantity. The same thing is done when the user enters or changes the Unit Price.

HINT: *Use the ROUND function to ensure the correct number of decimal places.*

7. When a user enters or changes a value in the **Unit of Measure Code** field, the program does the following:
   - If the **Type** is Resource:
     - Gets the corresponding Resource record.
     - If the Unit of Measure Code is blank, the program sets it to the Base Unit of Measure Code from the Resource record.

- Finds the corresponding record in the Resource Unit of Measure table and sets the **Qty. per Unit of Measure** field to the corresponding value in the Resource Unit of Measure table.
- Calculates the Unit Price according to the Unit Price from the Resource record.

– If the **Type** is G/L Account:
  - Sets the Qty. per Unit of Measure to 1.
  - If the current field is the **Unit of Measure Code** field, the program validates the Quantity. Use CurrFieldNo to check the current field. Do this in case OnValidate is triggered by some other field or code besides Unit of Measure Code.

Solution: The code in the Unit of Measure Code – **OnValidate** trigger should look like this:

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

Lab 3.3 - Adding Code to the Seminar Registration Header Table and Form

In the **Seminar Registration Header** table, enter code to perform the following tasks. Create the error messages using text constants to take advantage of Multilanguage functionality.

1. Create a new function called **AssistEdit** with a return type of **Boolean** that:
   - Takes a parameter called **OldSemRegHeader** which is a record variable of the **Seminar Registration Header** table.
   - Checks for a **Seminar Nos.** series in the **Seminar Setup** table.
   - If found, uses the **SelectSeries** function in the **NoSeriesManagement** codeunit to check the series number.
   - If **SelectSeries** returns TRUE, the program uses the **SetSeries** function in the **NoSeriesManagement** codeunit to set the No. field, and the program then exits TRUE.

   HINT: *The AssistEdit function created for the Seminar table is similar.*

2. Create a new function called **InitRecord** which:
   - Sets the **Posting Date** to the work date if the **Posting Date** is blank (= 0D)
   - Sets the **Document Date** to the work date.
   - Gets the **Seminar Setup** record and runs the **SetDefaultSeries** function of the **NoSeriesManagement** codeunit to set the **Posting No. Series** value.

Solution:

```
IF "Posting Date" = 0D THEN
  "Posting Date" := WORKDATE;
"Document Date" := WORKDATE;
SemSetup.GET;
NoSeriesMgt.SetDefaultSeries("Posting No.
Series",SemSetup."Posted Sem. Registration Nos.");
```

3. When a new record is inserted, if the program finds that the **No.** field is blank, it gets the **Seminar Registration Nos**. series from the **Seminar Setup table** and tests it. It then fills the **No.** field by using the **InitSeries** function of the **NoSeriesManagement** codeunit. The program then runs the new **InitRecord** function.

Solution: Enter the following code in the **OnInsert** trigger:

```
IF "No." = '' THEN BEGIN
  SemSetup.GET;
  SemSetup.TESTFIELD(SemSetup."Seminar Registration Nos.");
  NoSeriesMgt.InitSeries(SemSetup."Seminar Registration
Nos.",xRec."No. Series",0D,"No.","No. Series");
END;
InitRecord;
```

4. When the user deletes a record, the program tests that **Status** is **Canceled** and shows an error if **Status** is not **Canceled**. The program also shows an error if the header has registered **Seminar Registration Lines** or if there are associated **Seminar Charge** lines. The program then deletes corresponding records in the **Seminar Comment Line** table.

5. When a user attempts to rename a record, the program shows an error stating that a **Seminar Registration Header** cannot be renamed.

HINT: *Use a text constant and the TABLECAPTION function.*

6. If the user changes **No.** to a new value from what it was previously, the program tests whether the number series (from the **Seminar Registration Nos**. field in the **Seminar Setup**) is allowed to be changed manually by using the **TestManual** function from the **NoSeriesManagement** codeunit. It then sets the **No. Series** field to blank.

```
IF "No." <> xRec."No." THEN BEGIN
  SemSetup.GET;
  NoSeriesMgt.TestManual(SemSetup."Seminar Registration
Nos.");
  "No. Series" := '';
END
```

7. When the user changes a value in the **Starting Date** from what it was previously, the program tests that **Status** is **Planning**.

8. When the user changes a value in the **Seminar Code** from what it was previously, the program performs the following tasks:
   - Shows an error if there are any corresponding registered **Seminar Registration Line** records.
   - Gets the **Seminar** record and tests the following: that the **Blocked** field is false, that the **Gen. Prod. Posting Group** field is not blank and that the **VAT Prod. Posting Group** field is not blank.
   - Fills in the following fields with values from the Seminar record: **Seminar Name**, **Duration**, **Seminar Price**, **Gen. Prod. Posting Group**, **VAT Prod. Posting Group**, **Minimum Participants**, **Maximum Participants**. The program validates and fills in the **Job No.** field.

9. When the user enters or changes a value in the **Instructor Code**, the program calculates the value of the **Instructor Name** field.

10. When the user enters or changes a value in the **Room Code**, if the **Room Code** is blank, the program sets to blank the **Room Name**, **Room Address**, **Room Address2**, **Room Post Code**, **Room City** and **Room Phone No**. fields. Otherwise, the program gets the Seminar Room record and fills in those fields with the corresponding Seminar Room values.

11. When the user enters or changes a value in the **Room Code**, if the **Maximum Participants** in the corresponding **Seminar Room** record is less than the **Maximum Participants** in the **Seminar Registration Header**, the program asks the user whether the **Maximum Participants** in the **Seminar Registration Header** should be changed to the number of **Maximum Participants** from the **Seminar Room** record. If the user answers yes, the program changes the **Maximum Participants** value in the **Seminar Registration Header**.

HINT: *Use a CONFIRM message.*

---

12. When the user enters or changes a value in the **Room Post Code**, the program runs the ValidatePostCode function from the Post Code table.

13. When the user performs a lookup on the **Room Post Code** field, the program runs the LookUpPostCode function from the Post Code table.

14. When the user enters or changes a value in the **Room City** field, the program runs the ValidateCity function from the Post Code table.

15. When the user performs a lookup on the **Room City** field, the program runs the LookUpCity function from the Post Code table.

16. When the user changes the **Seminar Price** to a new value from what it was before and the Status is not Canceled, the program searches for corresponding, unregistered **Seminar Registration Lines**. If any are found, the program asks the user whether the **Seminar Price** should be changed in the unregistered **Seminar Registration Lines**. If the user answers yes, the program validates and modifies the lines with the new **Seminar Price**.

HINT: *Use the MODIFY function to change the* ***Seminar Registration Lines****.*

17. When the user changes the **Job No.** to a new value from what it was before, if the program finds any Seminar Charge records with the old **Job No.**, it asks the user whether the **Job No.** should be changed on the **Seminar Charge** lines. If the user answers yes, then the program modifies **Seminar Charge** lines with the new **Job No.**, otherwise if the user answers no, the program changes the **Job No**. back to the old Job No.

HINT: *Use the CONFIRM message function and use the MODIFYALL function to change the Seminar Charge lines.*

18. When validating the **Posting No. Series** field, if the **Posting No. Series** is not blank, the program tests that there are values in the **Seminar Registration Nos.** and **Posted Sem. Registration Nos**. fields on the Seminar Setup table. It then runs the TestSeries function from the NoSeriesManagement codeunit. Regardless of whether the **Posting No. Series** field is blank, the program tests that the **Posting No**. field is blank.

Solution: Enter the following code in the **Posting No. Series – OnValidate** trigger:

```
IF "Posting No. Series" <> '' THEN BEGIN
  SemSetup.GET;
  SemSetup.TESTFIELD("Seminar Registration Nos.");
  SemSetup.TESTFIELD("Posted Sem. Registration Nos.");
  NoSeriesMgt.TestSeries(SemSetup."Posted Sem. Registration
Nos.","Posting No. Series");
END;
TESTFIELD("Posting No.",'');
```

19. When the user performs a lookup on the **Posting No. Series** field, the program tests that there are values in the **Seminar Registration Nos.** and **Posted Sem. Registration Nos**. fields on the **Seminar Setup** table. If the LookupSeries function of the **NoSeriesManagement** codeunit is true, the program validates the **Posting No. Series** field.

Solution: Enter the following code in the **Posting No. Series – OnLookup trigger**:

```
WITH SemRegHeader DO BEGIN
  SemRegHeader := Rec;
  SemSetup.GET;
  SemSetup.TESTFIELD("Seminar Registration Nos.");
  SemSetup.TESTFIELD("Posted Sem. Registration Nos.");
  IF NoSeriesMgt.LookupSeries(SemSetup."Posted Sem.
Registration Nos.","Posting No. Series") THEN
    VALIDATE("Posting No. Series");
  Rec := SemRegHeader;
END;
```

20. In the **Seminar Registration** form, when the user clicks the **AssistEdit** on the **No.** field, the program runs the **AssistEdit** function for **xRec**, and if it returns **true**, updates the current form.

HINT: *Use the UPDATE function of the CurrForm object to update the current form.*

21. Enter code in the appropriate trigger so that after the form gets the record, the program removes the filter on the **No.** field.

Lab 3.4 - Adding Code for Seminar Registration Lines

In the **Seminar Registration Line** table, enter code to perform the following tasks:

1. Create a new function called **GetSemRegHeader** that retrieves the corresponding **Seminar Registration Header** record and stores it in a global variable.
2. Create a new function, **CalcAmount**, to calculate the **Amount** field as the **Seminar Price** reduced by the Line Discount %. Use the ROUND function.
3. When the user inserts a new line, the program retrieves the corresponding Seminar Registration Header record, sets the **Register Date** to the current work date, and sets both the **Seminar Price** and the **Amount to the Seminar Price** from the **Seminar Registration Header**.
4. When the user deletes a record, the program tests that the line is not registered.
5. When the user changes the value in the **Bill-to Customer No**. field from what it was previously, the program shows an error if the line is registered.
6. When the user performs a lookup on the **Participant Contact No.**, we want to show the **Contact List** filtered for the **Bill-to Customer Number**.
   - To do so, the program filters the **Contact Business Relation** table to the appropriate customer using the **Bill-to Customer No**. It then filters the **Contact** table to only those records where the **Company No**. is the same as that of the **Contact Business Relation** record. The program runs the **Contact List** form (using the RUNMODAL function) using the filtered **Contact** record, and if the user selects a contact and clicks **OK**, the program assigns the **Contact No.** field of the **Contact** record to the **Participant Contact No.**

Solution: Insert the following code in the Participant Contact No. – OnLookup trigger:

```
ContBusinessRelation.RESET;
ContBusinessRelation.SETRANGE("Link to
Table",ContBusinessRelation."Link to Table"::Customer);
ContBusinessRelation.SETRANGE("No.","Bill-to Customer
No.");
IF ContBusinessRelation.FIND('-') THEN BEGIN
  Cont.SETRANGE("Company No.",ContBusinessRelation."Contact
No.");
  IF FORM.RUNMODAL(FORM::"Contact List",Cont) =
ACTION::LookupOK THEN
    "Participant Contact No." := Cont."No.";
END;

CALCFIELDS("Participant Name");
```

7. When the user enters or changes a value in the **Seminar Price**, the program validates the Line Discount %.
8. Create a new function called **UpdateAmount** to calculate the **Amount** field to equal the Seminar Price minus the Line Discount Amount. Use the ROUND function with the **Amount Rounding Precision** field on the G/L Setup table as the precision parameter.
9. When the user enters or changes a value in the **Line Discount %,** the program calculates the **Line Discount Amount** (rounded by using the Amount Rounding Precision from the G/L Setup table) and then updates the **Amount** field using the new UpdateAmount function.
10. When the user enters or changes a value in the **Line Discount Amount**, if the Seminar Price is not 0, the program calculates the **Line Discount %** using the **Line Discount Amount** and the **Seminar Price**. If the Seminar Price is 0, the program sets the **Line Discount %** to 0. The program runs the UpdateAmount function.
11. When the user enters or changes a value in the Amount, the program checks that the **Bill-to Customer and Seminar Price** fields are not empty. It rounds the Amount value with the **Amount Rounding Precision** field from the G/L Setup table as the precision parameter. It calculates the **Line Discount Amount** and the **Line Discount %**.

Solution: Insert the following code in the 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
  "Line Discount %" := ROUND("Line Discount Amount" /
"Seminar Price" * 100,0.00001)
ELSE
  "Line Discount %" := 0;
```

Testing Seminar Registrations

Use the following test script to check the **Seminar Registrations** functionality. It is assumed that at least one **Seminar Room**, **Instructor** and **Seminar** is set up from the test script in the last chapter.

1. Select the **Seminar Registration** form in the **Object Designer** and click **Run**.
2. Tab off the **No.** field. The next number in the number series set up in the last chapter is automatically filled into the **No.** field. Note that when a new record is started, the **Posting Date** and **Document Date** values are set to the work date.
3. Select a **Seminar Code** and see that the **Seminar Name** is populated.
4. Select an **Instructor Code** and see that the **Instructor Name** is populated.
5. On the **Seminar Room** tab, select a **Room Code** and see that the other fields are populated.
6. On the **Invoicing** tab, enter a **Price** and a **Job No**. The values do not matter for now, but the **Job No**. becomes important in subsequent chapters when registrations are posted.
7. In the subform, use the lookup to select a **Bill-to Customer**. Then use the **Participant Contact No**. lookup. The values in the **Contact List** should be filtered to only show **Contact** related to the **Bill-to Customer**. Select a **Contact** and click **OK**. The **Participant Name** field should be populated automatically.
8. Tab through the rest of the fields on the line. Note that the **Seminar Price** is defaulted from the **Invoicing** tab. Enter a value in the **Line Discount %** and notice the **Line Discount Amounts** and **Amount** fields calculate accordingly.
9. Select the **Charges** menu item from the **Registration** command button and enter a new charge. See that the line item values are populated appropriately based on the **Resource** or **G/L Account** selected.
10. That concludes the testing for this use case. If there are areas that did not function as expected, make the necessary changes.

Conclusion

In this chapter, the tables and forms necessary to register participants in seminars was created, along with code to improve usability and data validation.

The next step is to take the transaction information and create a posting routine that can certify participants and create ledger entries for completed courses. It will also be possible to post invoices to customers.

Test Your Knowledge

Review Questions

1. What function is used to retrieve a field's caption? A table's caption?

2. Why is it necessary to be cautious when importing objects as text files?

3. When is a matrix box used on a form?

4. Can you write to a virtual table?

5. When are virtual tables computed by the system?

6. What does the AutoSplitKey property do?

7. What function can you use to force calculation of a FlowField?

Quick Interaction: Lessons Learned

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

1.

2.

3.