

# APPENDIX A: ADDITIONAL EXERCISES

## Objectives

The objectives are:

- Add seminar translations.
- Add multiple dimensions
- Manage seminar planning

## Introduction

This chapter includes four excersies not found in the previous chapters.

## Adding Seminar Translations

In this section a solution is developed that allows multilanguage support.

### Analysis

The client needs the ability to enter translations of the seminar name for Multilanguage functionality. This is accomplished by creating a **Seminar Translations** form and table, similar to those implemented for some Microsoft Dynamics NAV master files. To see an example in Microsoft Dynamics NAV, view the **Item Card** form (30), the **Item Translations** form (35) and the **Item Translation** table (30).

### GUI Design

This solution will require a **Seminar Translations** form (Figure A-1), that allows the entry of translations of the seminar name.

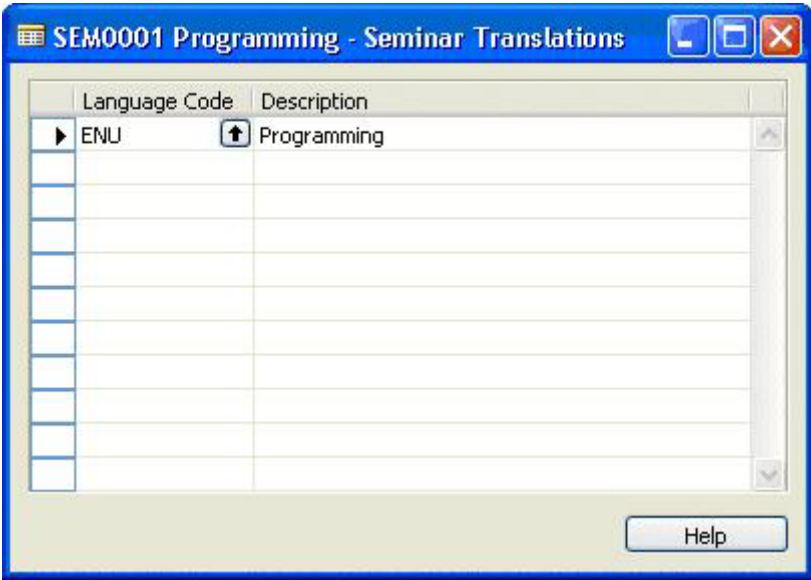
The image shows a Microsoft Dynamics NAV form titled "SEM0001 Programming - Seminar Translations". It features a table with two columns: "Language Code" and "Description". The first row contains the values "ENU" and "Programming". Below this, there are several empty rows for additional entries. A "Help" button is located at the bottom right of the form.

FIGURE A-1: THE SEMINAR TRANSLATIONS FORM (123456709)

The **Seminar Translations** form will be accessible from the **Seminar Card (Form 123456700)** through the addition of a new menu item to the **Seminar Menu Button** just above the Extended Texts menu item (after the separator).

Menu Button	Options	Comment
Seminar	Translations	Opens the <b>Seminar Translations</b> form (123456709) for the selected entry.

The **Seminar Translations** form will also be available from the **Seminar List** form (123456701), through the same **Translations** menu item as the **Seminar Card** form.

### Table Design

One new table will be required: the **Seminar Translations** (123456706) change request table, which will contain the following fields:

No.	Field Name	Type	Length	Comment
1	Seminar No.	Code	20	Must not be blank. Relation to Seminar table.
2	Language Code	Code	10	Must not be blank. Relation to Language table.
3	Description	Text	30	
4	Description 2	Text	30	

### Exercise 13-1 – Adding Seminar Translations

Perform the following tasks to add the seminar translations to the seminar master files.

1. Create the **Seminar Translation** table (123456706) according to the table design with a primary key of **Seminar No.**, **Language Code**.
  - Set the property to specify form 123456709 as the lookup form for this table.
2. In the **Seminar** table, enter code in the appropriate table triggers to perform the following tasks:
  - When a record is deleted, the program also deletes the corresponding records from the **Seminar Translation** table.
3. Create the **Seminar Translations** form (123456709). The only fields necessary on this form are **Language Code**, **Description** and **Description 2**.
  - Set the property so that the **Description 2** field is not visible.
  - Set the property to specify that this form is the lookup form for the **Seminar Translation** table.
4. Add the Translations menu item to the **Seminar Card** form (123456700) as shown in the GUI design section previously.
5. Add the Translations menu item to the **Seminar List** form (123456701) as shown in the GUI design section previously.

## Adding Multiple Dimensions

In Chapter 8, dimensions were added to the seminar modules, but the client wants to increase the usability of this feature by defining multiple dimensions.

To see a sample of how this works in Microsoft Dynamics NAV, open the **Customer Card** and select **Dimensions** from the **Customer** menu button to open the **Default Dimensions** form. This is the same kind of dimension functionality implemented in the seminar module. Now open the **Customer List** and select more than one customer. Under the **Customer** menu item there is a submenu under **Dimensions** with two choices, Dimensions-Single and Dimensions-Multiple. Select **Dimensions-Multiple** to open the **Default Dimensions-Multiple** form. From here the user can specify default dimensions for multiple customers at a time. This solution enables multiple dimensions functionality in the **Seminar List**, the **Seminar Room List** and the **Instructors** forms.

### GUI Design

Add menu items for dimensions to the forms from which master data is defined.

**Seminar List (Form 123456701):** Add a **Dimensions** menu item just after the Comments menu item on the **Seminar** menu button (just as on the **Seminar Card** form).

Menu Button	Options	Comments
Seminar	Dimensions	Opens submenu with the following options: Dimensions-Single (SHIFT+CTRL+D): Opens the <b>Default Dimensions</b> form (540) for the selected entry. The link should run whenever the form is updated. Dimensions-Multiple: Runs code in the <b>OnPush</b> trigger to open form 542 Default Dimensions-Multiple.

**Seminar Room List (Form 123456704):** Add a **Dimensions** menu item that opens a submenu with the Dimensions-Single and Dimensions-Multiple options as shown previously for the **Seminar List** form.

**Instructors (Form 123456705):** Modify the existing **Dimensions** menu item on the **Instructors** menu button. The menu item is changed to open a submenu with the Dimensions – Single and Dimensions – Multiple options as shown previously for the **Seminar List** form.

### Exercise 13-2 – Adding Multiple Dimensions

From the list form of a master file, as with the Seminar List form, the user can select several records and set the dimensions for all of them at one time by using the Dimensions – Multiple option from the menu button. Make some modifications to the **Default Dimensions-Multiple** form to enable this functionality for the seminar module.

1. In the **Default Dimensions - Multiple** form (542), create a function called **SetMultiSeminar** for the form with an ID of 123456700 and a parameter of a record variable of the **Seminar** table, which is passed by reference.
2. Enter code in the function trigger so that the function runs the **CopyDefaultDimToDefaultDim** function for every record in the **Seminar** record variable.
3. Create two functions called **SetMultiSemRoom** and **SetMultiInstructor**. Set the IDs of these functions to 123456701 and 123456702, respectively. Set the parameters and enter code into the function triggers so that they copy the default dimensions for the **Seminar Room** and **Instructor** records, respectively. They should do this in a way similar to the **SetMultiSeminar** function just created.
4. Add the dimensions menu items to the **Seminar List**, **Seminar Room List** and **Instructors** forms as shown in the GUI design.

## Seminar Planning

In this section, a solution is discussed that provides the client with a calendar system that gives an overview of seminar dates to help in seminar planning.

### Analysis

The client's functional requirements describe the seminar planning overview in the following way:

*You need a calendar system to give overview of your seminar dates to help in seminar planning. You want to be able to view seminars by date and to set filters to see the overview for seminars with a specific seminar status, seminar room or instructor.*

When seminar managers want to gain an overview of which seminars are scheduled for certain dates, they look at a seminar planning overview form that allows them to view different dates and look at different periods of time (such as a week or a month).

The program uses the company's Base Calendar (go to GENERAL LEDGER→SETUP→BASE CALENDAR) to schedule the seminar dates so they are not held on weekends or on scheduled holidays.

## Design

The seminar planning overview is a representation of the relationship between the existing data of seminar registrations and calendar dates. This relationship is a many-to-many relationship, which means that any number of seminars can be held on any number of dates. Because of this many-to-many relationship, a "matrix" type representation of the data is required.

For a review of matrix forms, see Chapter 3, Registration.

## GUI Design

Only one new form is necessary for the **Seminar Planning** overview, which appears as Figure A-2.

[illegible]

FIGURE A-2: THE SEMINAR PLANNING FORM (FORM 123456733)

The left side of the matrix shows the seminar registration header, and the right side shows dates. The filters in the top allow filtering on **Status**, **Instructor** or **Room**.

The buttons along the bottom are option buttons that change the view of the dates. The 1 button shows the view by day, the 7 button shows the view by week, the 31 button shows the view by month, the 3 button shows the view by quarter and the 12 button shows the view by year. Finally, the last button shows the view by accounting period.

### Functional Design

The following functions will be needed to complete the implementation of the seminar planning overview:

**CheckDate:** A function that helps ensure that no seminars are scheduled for weekends or holidays. This function takes a date as a parameter, checks it against the company's base calendar to see whether the date is a "working" date and returns true or false depending on the answer.

**GetSelectionFilter:** A function is needed for the **Instructor Filter** and **Room Filter** fields at the top of the form. When the user clicks the lookup button on these fields, the program shows the Instructors form and **Seminar Rooms List** form for the **Instructor Filter** and **Room Filter** fields, respectively. When, from these forms, the user selects the values to use in the filter, the program should read the selection and create a "filter string" that can be used by the **SETFILTER** function to filter the Instructor and/or Room table.

For instance, suppose the user selected the fields shown in Figure A-3.



Code	Name	Maximum...	Resource...
ROOM01	Room 1	25	SEMINAR...
ROOM02	Room 2	50	SEMINAR...
ROOM03	Room 3	20	SEMINAR...
ROOM04	Room 4	15	SEMINAR...

Seminar ... Help

FIGURE A-3: MULTIPLE SELECTIONS ON A FORM

A **GetSelectionFilter** function is required to produce the string 'ROOM01..ROOM03.' On the other hand, if the user had selected only the first and third records, the string should be 'ROOM01|ROOM03.'

**Table Design**

For this form to work, create a **Seminar Registration Buffer** table. This is a simple table used to temporarily store information for this form.

The **Seminar Registration Buffer** table (123456730) contains the following fields:

No.	Field Name	Type	Length	Comment
1	Entry No.	Integer		Must not be blank.
2	Sem. Reg. No.	Code	20	Relation to Seminar Registration Header table.
3	Date	Date		
4	Allocation	Decimal		Decimal Places 0:1

**Exercise 13-3 – Creating the Seminar Planning Form**

1. Begin by creating the **Seminar Planning** form (123456733) as a basic form with a matrix box.
  - The source table for the form is a "vertical" table (as described above), **Seminar Registration Header**.
  - Give the matrix box the simple name of **PlanningMatrixBox**.
  - Set the property for the matrix box to specify that the matrix box is not editable.
  - Set the matrix box source table as the virtual **Date** table.

Now add the buttons to the bottom of the form so that the user can control which time periods are shown. The buttons at the bottom of the form are called **Trendscape option** buttons. The **Trendscape option** button is a special control type. Notice that they operate much like normal option buttons, except that they use system bitmaps.

2. To simplify things, copy these buttons from a standard form. Use the **Item Availability** form (157) by Periods for this purpose. After copying them and pasting them to the form as shown in the GUI design, look at the properties. Notice that their source expressions are options of the global Option variable called **PeriodType**. Copy this variable from the standard form as well, or it manually. The options for this variable correspond to the option buttons: **Day**, **Week**, **Month**, **Quarter**, **Year** and **Period**. Delete the code in the OnPush trigger of these buttons.



3. Add three text boxes to the left side of the matrix box. The source expressions for these text boxes are the **No.** field, the **Seminar Code** field, and the **Seminar Name** field. Set the property to specify that the **Seminar Name** field expands and contracts with the form size.
4. Add the matrix body and matrix heading text boxes to the right side of the matrix box as described above.
5. The matrix heading text box is where the starting date of each period is to show. Because the period starting date changes depending on the **Period Type** chosen by the user (using the **Trendscape option** buttons), the expression used to calculate the value for this field is a variable. Set the source expression for the heading to a global variable with the data type Text, called **MatrixHeader**.
6. The matrix body text box contains the number of days allocated for the seminar. This is a calculated value that depends on the **Period Type** chosen by the user (using the **Trendscape option** buttons), so the source expression must be a variable calculated in the code for the form. Therefore, set the source expression for the body to a global variable with the data type **Decimal**, called **AllocationDay**. Set the properties for the matrix body text box so that the decimal places are 0:1 and so that the field is blank when the value is 0.
7. Add a tab control to the top of the form as shown in the GUI design. Set the property to specify that the tab expands and contracts with the form size.
8. Add the text boxes (with labels) for the three filter fields at the top of the form on the tab control. The source expressions for these fields will also be global variables.
  - The source expression for the **Status Filter** field is an option-type variable named **StatusFilter**, with the options corresponding to the status options for seminars (except for the Closed option, because we do not want to see old seminar data). There is an additional option of None so that the user can choose to see all seminars regardless of status.
  - The source expression for the **Instructor Filter** field is a text-type variable named **InstructorFilter**. Set the property for the text box so that the field is not cleared when the user performs a lookup.
  - The source expression for the **Room Filter** field is a text-type variable named **RoomFilter**. Set the property for the text box so that the field is not cleared when the user performs a lookup.

### **Exercise 13-4 – Adding Code to the Seminar Planning Form**

Some changes to existing forms will be required:

1. In the **Seminar Room List** form, create a new function called **GetSelectionFilter** with a return type of **Code** and a length of 80. The function must ultimately create a string that can be used as a filter from the user's selection from the form. See the Functional Design for more information on this function.
  - Create a local variable called **SelectionFilter** with data type **Code** and a length of 250. This variable stores the filter.
  - Begin by storing the user's selection from the form in a local variable for the **Seminar Room** record, called **SemRoom**.

---

**HINT:** Do this by using the **SETSELECTIONFILTER** function of the **CurrForm** variable.

---

- Go through the selection of records in the **SemRoom** variable and add the **Code** field for each selected record to the **SelectionFilter**. Do not forget to add the **|** or **..** between the entries.
2. Create a similar function for the Instructors form. Create a new function called **GetSelectionFilter** with a return type of **Code** and a Length of 80. The code for this function is the same as that created for the **GetSelectionFilter** function in the **Seminar Room List** form, except that different variables are used to get instructors instead of seminar rooms.

Next, calculate the values to fill the matrix body. To do this, use a new **Seminar Registration Buffer** table (as described in the table design section previously) as a temporary table. A temporary table is needed to avoid filling a table with permanent data every time the seminar planning overview is used.

3. Create the **Seminar Registration Buffer** table (123456730) as shown in the table design.
  - The primary key is the **Entry No.** field.
  - Set the **SumIndexFields** property for the key to **Allocation**.
  - Declare the **Seminar Registration Buffer** as a temporary table global variable in the **Seminar Planning** form.

Now that a temporary table has been declared as a variable, create the code in the **Seminar Planning** form that brings the form to life.

4. To start, tell the program which record in the Date table to find first so that it shows the work date rather than starting with the first date on the Date table, January 1,0000. Enter code in the **OnFindRecord** trigger of the matrix box to get the first date to show. To do this, use the **FindDate** function in the **PeriodFormManagement** codeunit. The **FindDate** function expects three parameters:
  - A search string
  - A calendar
  - A period type

As the search string, use the Which parameter of the **OnFindRecord** trigger. For the calendar parameter, use the matrix rec. Note that the **OnFindRecord** trigger returns a Boolean value, so use the EXIT function to return the value of the **FindDate** function.

5. Enter code in the **OnNextRecord** trigger of the matrix box to get the next date. To do this, use the **NextDate** function, which returns an integer, in the **PeriodFormManagement** codeunit.
6. In the code for the matrix box, set the value for the **MatrixHeader** variable. Do this in the **OnAfterGetRecord** trigger of the matrix box. For this, use the function **CreatePeriodFormat** from the standard codeunit **PeriodFormManagement**, which returns the value for the **MatrixHeader**. As parameters, this function expects a period type and a date. In this case, the period type is what was selected by the user (with the **Trendscape option** buttons), and the date parameter is the starting date from the virtual Date table. To reference the source table of the matrix box, use the following formula:

CurrForm.<Matrix Box Name>.MatrixRec.<Field Name> .

Test the code by running the form and testing the period option buttons.

Now a function to check whether dates are weekends or holidays is needed.

7. Create a **CheckDate** function for the **Seminar Planning** form as described in the functional design. This function checks the date suggested against the changes to the company's base calendar, stored in the **Base Calendar Changes** table (7601) , to see whether the date is a "Nonworking" date. The function gets the **Base Calendar Code** from the **Company Information** table and finds the corresponding records in the **Base Calendar Changes** table. If the date to check is found in any of the "nonworking" dates on the table, the function returns false.

---

**HINT:** Use the *DATE2DMY* and *DATE2DWY* functions to work with the dates.

---

Next, fill the temporary table with the dates on which the seminars run. This buffer is to be filled every time the form is opened and every time the user clicks a period type option button.

8. Create a **FillBuffer** function that first resets the **Seminar Registration Header** table and empties the buffer table. Then, for every line in the **Seminar Registration Header** table, for every day in the Duration of the seminar, the function inserts a line into the buffer table with a date for the seminar to run on (using the **CheckDate** function to make sure the date is a working day).

---

**HINT:** Use the *CALCDATE* function when working with the dates.

---

- Call the **FillBuffer** function from the appropriate places in the form and option button triggers.

The next step is to set the **AllocationDay** variable with information from the buffer table.

9. Enter code in the appropriate trigger so that when the matrix box gets a record, the program filters the buffer table on the **Seminar Registration No.** and on the dates that fall between the **Period Start** and **Period End** for the date table, and then performs a **CALCSUMS** on the **Allocation** field in the **Seminar Registration Buffer**. The value for the **AllocationDay** variable will be the result of the **CALCSUMS**.

Now add code to enable the filter fields at the top of the form.

10. Write a function called **SetRecFilters** to filter the **Seminar Registration Header** table using the values entered into each of the filter fields at the top of the form. At the end of the function, the program updates the current form using the **UPDATE** function.
  - Call the **SetRecFilters** function from the appropriate triggers so that it runs when the form is opened and after the user enters or changes a value in the filter fields.
11. Enter code in the appropriate trigger so that when the user performs a lookup on the **InstructorFilter** field, the program runs the Instructors form modally. If the user selects a record and clicks **OK**, the program stores the result of the **GetSelectionFilter** function from the Instructors form in a text variable, and exits TRUE. If the user clicks **Cancel** from the form, the program exits FALSE.

12. Enter code in the appropriate trigger so that when the user performs a lookup on the **RoomFilter** field, the program runs the **Room List** form modally. If the user selects a record and clicks **OK**, the program stores the result of the **GetSelectionFilter** function from the **Room List** form in a text variable, and exits TRUE. If the user clicks **Cancel** from the form, the program exits FALSE.
13. Finally, set up the drilldown on the matrix body text box so that it opens the **Seminar Registration List** form for the appropriate seminar.
14. Test the **Seminar Planning** form.

## Conclusion

In this chapter, additional functionality was added to the seminar module to conform with Microsoft Dynamics NAV standards for dimensions and multilanguage support. A seminar planning overview was also created, that brings together information from seminar registrations and the system's virtual date table.

