

**MICROSOFT® BUSINESS SOLUTIONS
NAVISION® 4.0**

**COURSE 8401A: DEVELOPMENT II – C/SIDE
SOLUTION DEVELOPMENT**

Microsoft Internal Use Only

Last Revision: January, 2005

The information contained in this document represents the current view of Microsoft Corporation on the issues discussed as of the date of publication. Because Microsoft must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented after the date of publication.

This document is for informational purposes only. MICROSOFT MAKES NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, AS TO THE INFORMATION IN THIS DOCUMENT.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

© 2004 Microsoft Corporation. All rights reserved. Microsoft Business Solutions–Navision, Microsoft SQL Server, Microsoft Windows, Microsoft Visual Basic, Microsoft Excel and Microsoft Word are either trademarks or registered trademarks of Microsoft Corporation or Great Plains Software, Inc. or their affiliates in the United States and/or other countries. Great Plains Software, Inc. is a subsidiary of Microsoft Corporation. The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

Introduction	1
Welcome	1
About the Navision Development II Course.....	2
Navision Development Concepts	4
Student Objectives	5
Chapter 1: Business Case Diagnosis & Analysis	7
Introduction.....	8
Diagnosis – Executive Summary.....	8
Analysis – Functional Requirements	9
Other Requirements	10
Data Model	11
Project Plan	12
Quick Interaction: Lessons Learned.....	13
Chapter 2: Managing Master Files	15
Introduction.....	16
Triggers	17
Multilanguage Functionality	24
Microsoft SQL Server	25
Test Your Skills – Managing Master Files – Diagnosis	26
Implementation of Use Case 1 – Managing Participants.....	27
Implementation of Use Case 2 – Managing Rooms	30
Implementation of Use Case 3 – Managing Instructors.....	38
Implementation of Use Case 4 – Managing Seminars	41
Testing Master Files	50
Test Your Knowledge	52
Conclusion.....	53
Quick Interaction: Lessons Learned.....	54
Chapter 3: Managing Registrations	55
Introduction.....	56
Exporting Objects as Text Files.....	57
Multilanguage Functionality in Text Messages.....	58
Main/Sub Forms	59
Matrix Forms	60
Types of Tables.....	61
Additional Functions	64
Test Your Skills – Managing Registrations – Diagnosis	66
Implementation of Use Case 1 – Managing Seminar Registration.....	67
Testing Seminar Registrations	89
Conclusion.....	91
Quick Interaction: Lessons Learned.....	92

Chapter 4: Managing Posting	93
Introduction.....	94
Posting	95
Performance Issues	105
Debugging Tools	107
Test Your Skills – Managing Posting – Diagnosis	110
Implementation of Use Case 1 – Managing Seminar Registration Posting.....	111
Testing Managing Posting.....	139
Test Your Knowledge	141
Conclusion.....	141
Quick Interaction: Lessons Learned.....	142
Chapter 5: Managing Integration	143
Introduction.....	144
Changing Tables that Contain Data	145
Test Your Skills – Managing Integration – Diagnosis	146
Implementation of Use Case 1 – Managing Seminar Feature Integration.....	147
Implementation of Use Case 2 – Managing Navigate Integration	152
Testing Managing Integration.....	155
Test Your Knowledge	156
Conclusion.....	156
Quick Interaction: Lessons Learned.....	157
Chapter 6: Managing Reporting	159
Introduction.....	160
Reporting.....	161
Test Your Skills – Managing Reporting – Diagnosis	164
Implementation of Use Case 1 – Managing Participant List Reporting	165
Implementation of Use Case 2 – Managing Certificate Confirmation.....	173
Implementation of Use Case 3 – Managing Invoice Posting.....	176
Test Your Knowledge	183
Conclusion.....	183
Quick Interaction: Lessons Learned.....	184
Chapter 7: Managing Statistics	185
Introduction.....	186
Using FlowFilters in Calculation Formulas	187
Test Your Skills – Managing Seminar Statistics – Diagnosis	188
Implementation of Use Case 1 – Managing Statistics	189
Test Your Knowledge	195
Conclusion.....	195
Quick Interaction: Lessons Learned.....	196

Chapter 8: Managing Dimensions	197
Introduction.....	198
Dimensions.....	199
Using Navision Developer’s Toolkit.....	201
Test Your Skills – Managing Dimensions – Diagnosis	203
Implementation of Use Case 1 – Managing Dimensions in Master Files	204
Implementation of Use Case 2 – Managing Dimensions in Registration.....	210
Implementation of Use Case 3 – Managing Dimensions in Seminar Posting ...	223
Implementation of Use Case 4 – Managing Dimensions in Invoicing.....	231
Testing Managing Dimensions	233
Test Your Knowledge	235
Conclusion.....	235
Quick Interaction: Lessons Learned.....	236
Chapter 9: Managing Interfaces	237
Introduction.....	238
Using an Automation Server	239
Using Custom (or OCX) Controls	240
XMLPort Triggers	240
File Handling	241
Test Your Skills – Managing Interfaces – Diagnosis	244
Implementation of Use Case 1 – Managing E-mail Confirmation.....	245
Implementation of Use Case 2 – Managing XML Participant List	250
Conclusion.....	253
Test Your Knowledge	254
Quick Interaction: Lessons Learned.....	255
Chapter 10: Deployment	257
Introduction.....	258
Deployment Tasks.....	258
Ongoing Support Phase	261
Conclusion.....	276
Quick Interaction: Lessons Learned.....	277
Chapter 11: Course Summary	279
Course Summary	279
Chapter 12: Review Questions	283
Review Questions	283
Chapter 13: Additional Exercises	289
Introduction.....	290
Adding Seminar Translations	291
Adding Multiple Dimensions	293
Managing Seminar Planning	294
Conclusion.....	303

Appendix A: Sample Reports	305
Sample Participant List.....	306
Sample Certificate Confirmation.....	307
Appendix B: Sample XML Participant List	309
Sample XML Sem. Reg.-Participant List	310
Appendix C: USING C/FRONT	311
Introduction.....	312
Using C/FRONT	312
Two Interfaces – DLL and OCX.....	312
Accessing Data From the Database Using C/FRONT in Visual Basic	314
Writing Data Back to the Database Using C/FRONT in Microsoft Excel	320
Limitations of C/FRONT	320
Appendix D: Answers to Review Questions	321
Answers to Review Questions.....	322
Index	325

INTRODUCTION

Welcome

We know training is a vital component of retaining the value of your Microsoft® Business Solutions investment. Our quality training from industry experts keeps you up-to-date on your solution and helps you develop the skills necessary for fully maximizing the value of your solution. Whether you choose Online Training, Classroom Training, or Training Materials, there's a type of training to meet everyone's needs. Choose the training type that best suits you so you can stay ahead of the competition.

Online Training

Online Training delivers convenient, in-depth training to you in the comfort of your own home or office. Online training provides immediate access to training 24 hours a day. It's perfect for the customer who doesn't have the time or budget to travel. Our newest online training option, eCourses, combine the efficiency of online training with the in-depth product coverage of classroom training, with at least two weeks to complete each course.

Classroom Training

Classroom Training provides serious, in-depth learning through hands-on interaction. From demonstrations to presentations to classroom activities, you'll receive hands-on experience with instruction from our certified staff of experts. Regularly scheduled throughout North America, you can be sure you'll find a class convenient for you.

Training Materials

Training Materials enable you to learn at your own pace, on your own time with information-packed training manuals. Our wide variety of training manuals feature an abundance of tips, tricks, and insights you can refer to again and again:

Microsoft Business Solutions Courseware: The Microsoft Business Solutions Training Courseware are very detailed training manuals, designed from a training perspective. These manuals include advanced topics as well as training objectives, exercises, interactions, and quizzes.

Look for a complete list of manuals available for purchase on the Microsoft Business Solutions website: www.microsoft.com/BusinessSolutions.

About the Navision Development II Course

The Microsoft Navision Development II course provides conceptual and practical information for all phases of developing customized solutions in Navision. The training material is designed for use in an instructor-led training course, but you can also use it for self-study purposes.

Target Audience

This training material is intended for Microsoft Certified Business Solutions Partner employees who sell and implement Navision solutions. The curriculum is designed for participants who have completed the Navision Development I course and passed the Navision Development I certification exam.

Training Objectives

The goal of this course is to apply the basic skills taught in the Navision Development I course to developing solutions in Microsoft Navision. The best way to learn solution development is to try to develop a solution yourself. Therefore, the course is structured around a business case, which constitutes a small development project. You will be developing a solution to meet the requirements of a customer. During the course, you will use the concepts, practices and methodology first introduced in the Navision Development I course that ideally should be applied for developing any kind of Microsoft Navision solution.

Course Structure

The course begins with the presentation of the business case. The subsequent chapters each handle one aspect of the development project by introducing the necessary concepts, standards and technical components that will be used for the development. Some chapters also include a code walkthrough that will illustrate many of these concepts.

Each set of chapter exercises is introduced with a diagnosis of the client's needs and the associated use cases. One exercise covers one use case and consists of:

- An analysis of the individual use case.
- The design of the solutions for each use case.
- The development for each use case.
- The testing for each use case.

Some exercises have been designated as optional, even though they cover useful and examinable material, due to the time restraints on the instructor-led course. It is intended that participants complete them outside of the classroom.

At the end of the course, we will discuss deployment and on-going maintenance issues.

Course Content

This course covers the following topics:

- Creating solutions following the Microsoft Navision Implementation Methodology
- Internal documentation
- Debugging tools
- Performance issues
- Complex data variables and their member functions
- Multilanguage functionality
- Analysis of objects
- Posting routines
- Architecture of a basic Microsoft Navision document
- Microsoft Navision dimensions
- Deploying customized software solutions

Certification

You must be certified to develop customized solutions in Microsoft Navision. This course prepares you for the Navision Development II certification exam.

Courseware

A database containing the finished objects and demonstration data for this course is provided.

Suggested Course Duration

The suggested course duration for Navision Development II training is five days of instructor-led classroom time. Additional time will be required for most students to complete the additional exercises and review materials for exam preparation.

Further Information

You can find further information about the technical topics mentioned in this course in the C/SIDE[®] online Help (called C/SIDE Reference Guide). For more information regarding design issues, see the Application Designer's Guide.

Navision Development Concepts

Before you begin development, you must first determine what tools and concepts you will use to analyze the problems presented and to design and develop the solution to those problems. These tools and concepts are split into:

- The "rules" that you will use in development.
- The "methodology" which you will use to approach the problem.

Rules

To develop the solution, you use Microsoft Navision's development environment, C/SIDE, and its proprietary programming language, C/AL. You access C/SIDE by clicking TOOLS→OBJECT DESIGNER in the menu bar or SHIFT + F12.

It is assumed that participants in Development II possess a good understanding of basic C/AL and C/SIDE functionality as covered in the Development I course.

NOTE: During the development of our solution, you may find it useful to consult the C/SIDE Reference Guide to find information about data types, properties, triggers, controls and functions, among other things. In addition to accessing the C/SIDE Reference Guide from the Help menu, you can press F1 to get context-sensitive Help.

Methodology

We will take a phased approach to implementing our solution. This methodology includes the use of UML Use Cases and Activity Diagrams in analyzing and designing solutions. We will describe actions related to our development project as belonging to the following phases:

- Diagnostic Phase
- Analysis Phase
- Design Phase
- Development & Testing Phase
- Deployment Phase
- On-Going Support Phase

Student Objectives

What do you hope to learn by participating in this course?

List three main objectives below.

1.

2.

3.

Microsoft Internal Use Only

Microsoft Internal Use Only

CHAPTER 1: BUSINESS CASE DIAGNOSIS & ANALYSIS

This chapter contains the following sections:

- Introduction
- Diagnosis – Executive Summary
- Analysis – Functional Requirements
- Other Requirements
- Data Model
- Project Plan

Microsoft Internal Use Only

Introduction

Now that it is established how to approach our business problem, you are ready to look at the business case. This chapter covers the Diagnosis and Analysis phases of our project.

The Diagnosis section provides the "executive summary" of the business case, including our client's profile and a high-level description of their needs. The following Analysis sections describe in more detail the client's specific requirements for the system. They contain the data models based on those requirements and they explain the basic project plan that we will follow to make the solution a reality.

Following this chapter, the Design and Development phases are broken down in the chapter exercises. These exercises constitute separate pieces of the development project, but they are interrelated and therefore dependencies do exist.

After the development is completed, the final chapter discusses deployment issues.

Diagnosis – Executive Summary

You are a Certified Microsoft® Business Solutions–Navision® Developer working for a Microsoft Certified Business Solutions Partner. The project to which you have been assigned is for CRONUS International Training Academy, a software training center that is a branch of CRONUS International Ltd. Due to significant growth, they need a new computer-based system that will allow them to both store and integrate all of their seminar, instructor, customer, and financial information in one solution.

The client currently uses a full suite of Microsoft Navision granules under the parent company CRONUS International Ltd. To take advantage of their investment and of the existing functionality and flexibility of Microsoft Navision, they have decided to add a customized seminar management module to their current solution.

Generally speaking, the new module should allow them to track their master data, to register participants in their seminars, to create invoices for customers, and to have an overview of their statistics. The preliminary analysis and design of their processes and requirements have already been done and are included in this training manual.

If the client is satisfied with the seminar management module, this type of system could be sold to other training academies. You must therefore develop the solution in such a way that other companies could also use it.

Analysis – Functional Requirements

The client has defined their requirements for the new seminar management module by providing the following description of how they run their training academy.

Seminars

The CRONUS training department holds several different seminars. All seminars have a fixed duration and allow a maximum and a minimum number of participants. They can be overbooked in some cases, depending on the capacity of the assigned room. They can be cancelled if there are not enough participants. The price of a seminar is fixed. You would like to take advantage of the current Job functionality in Microsoft Navision and define each seminar as a job. When a seminar is completed, the seminar should be posted as a job, with additional seminar-specific information.

Each seminar is held in a seminar room. Some seminars are held in-house and some are outsourced. You want to be able to track costs and prices for our rooms (that is, the rental fee). If a seminar takes place in-house, you want to assign one of our rooms.

Gather and review certain information from the customer, such as details of his seminar participants, the number of seminars participated in and the names of these seminars.

Instructors

Each seminar is taught by an instructor, who is an employee from our company. To make use of our existing resource information, each instructor must be set up as a Resource in Microsoft Navision.

Participants

Seminar participants come from a company that is set up in Microsoft Navision as a customer. The participants must be handled separately from the customers. Furthermore, every customer can register several participants for a seminar. Participants cannot be registered for a seminar unless they are connected with a customer. This is necessary because you want to invoice customers for the participation at seminars.

Registration

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

A 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 rental. In the registration information, you must also be able to specify how the seminar should be invoiced (for example, whether to include expenses or catering).

You should be able to set up additional comments for each seminar that would allow you to specify necessary equipment or any other particular requirements for the specific course.

Invoicing

When a seminar is finished, invoice the customers for the participation of their registered participants. Invoicing will be by project and resources.

Reporting and Statistics

You should be able to print a list of the participants registered for a seminar.

Upon completion of some seminars, participants receive a seminar certificate. Create such certificates from the system for individual participants.

You would like to see statistical information regarding the total price from each seminar; breaking it down into what is chargeable and what is not chargeable. You want to see these statistics for different time periods, for example, for a month, for Last Year, for This Year, and To Date.

Interfaces

You should be able to send an e-mail notification to the customer's participants in several types of situations, such as registration confirmation. You want to be able to export the participant list for a seminar as an XML file.

Dimensions

Add analysis features by extending standard Microsoft Navision dimension functionality to the seminar module. Dimensions should be available for the master files, registrations, posting and invoicing.

Additional Requirements

You need a calendar system that will give us an overview of our 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.

Other Requirements

To make the solution as user-friendly for the client as possible, keep the following requirements in mind throughout development:

Easy to learn: The seminar management module must be easy to understand, and the terminology and symbols must be consistent with the rest of the program. This means that if the user knows how to use other applications in Microsoft Navision, they must also be able to intuitively learn our solution.

Efficient: Experienced users should be able to work with the program efficiently. This means, for instance, that the most frequently used functions should be accessible from the keyboard. It should be possible to efficiently use both the mouse and the keyboard.

Clarity: The user interface must be so intuitive that the least-experienced user can easily understand how the program functions.

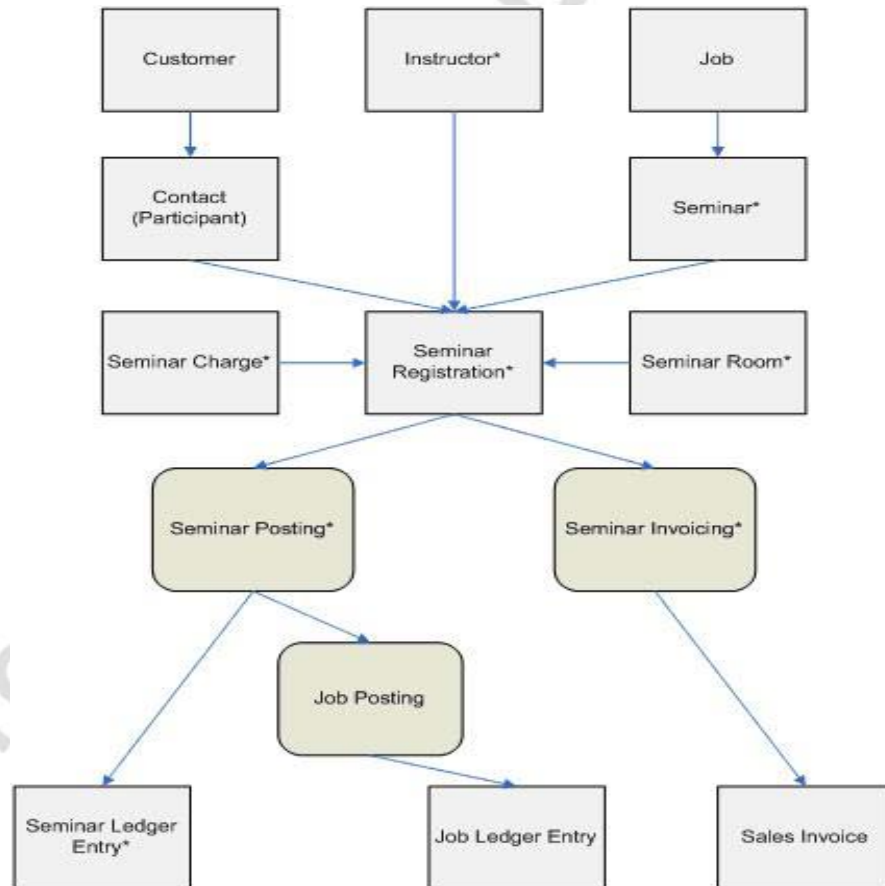
Easy error correction: The program must be built so that there are as few opportunities for error as possible. Error messages must explain the cause of the error and provide a suggestion as to how the user can correct the error.

And finally, because the solution for the seminar management module could be sold as an add-on to other Microsoft Navision customers, you must adhere to the guidelines contained in the following documentation:

- Application Designer's Guide

Data Model

The following data model has been created on the basis of the client's specification of the functional requirements:



*New table or process

Project Plan

As in any other project, our business case must be broken down into tasks, which make up the exercises in each chapter. There are dependencies among the exercises and the technical complexity of the project increases as development progresses.

The structure of the exercises reflects the principles of the implementation methodology. Diagnosis and analysis phases therefore precede the development and testing of the individual tasks. The end result of a chapter's exercises will be a deliverable for that particular stage of the project.

The tasks for this project are:

- Managing Master Files (Chapter 2)
- Managing Registrations (Chapter 3)
- Managing Posting (Chapter 4)
- Managing Integration (Chapter 5)
- Managing Reporting (Chapter 6)
- Managing Statistics (Chapter 7)
- Managing Dimensions (Chapter 8)
- Managing Interfaces (Chapter 9)

Quick Interaction: Lessons Learned

Take a moment to write down three Key Points you have learned from this chapter:

1.

2.

3.

Microsoft Internal Use Only

Microsoft Internal Use Only

CHAPTER 2: MANAGING MASTER FILES

This chapter contains the following sections:

- Introduction
- Triggers
- Complex Data Types
- Multilanguage Functionality
- Microsoft® SQL Server®
- Test Your Skills
 - Diagnosis
 - Managing Participants
 - Managing Rooms
 - Managing Instructors
 - Managing Seminars
 - Testing Master Files
- Test Your Knowledge
- Conclusion

Microsoft Internal Use Only

Introduction

Positioning – What is our Starting Point?

We are at the beginning of our project. The best way to start any project is to build a solid foundation. The foundation for our seminar management module will be our master data. Our first task is to learn the basics required to create our master files. We then define the types of master data on which we will base the seminar management module.

Preconditions

As we are now creating master files, our only technical precondition is a clean database. We need to know the functional requirements for the system. Our client has provided us with a description.

Business Goals

By the end of this chapter, we will have created both the master files for the seminar management module and the user interfaces for entering data into the master files.

Educational Goals

By completing this chapter, you should have learned or reacquainted yourself with the following:

- Microsoft® Business Solutions–Navision® standards for master files.
- Working with event triggers, specifically table event triggers.
- Working with the complex data types and their member functions.
- Introduction to multi-language functionality.

Triggers

Types of Triggers

There are three kinds of triggers in C/AL:

- **Documentation Trigger:** This is not a trigger as it contains no C/AL code. Instead, a programmer can use the Documentation trigger to write internal documentation for an object.
- **Event Trigger:** The name of these triggers always starts with "On." The C/AL code in an event trigger is executed when the named event occurs. For instance, the code in an OnInsert trigger for a table is executed when a record is inserted into the table.
- **Function Trigger:** These triggers are created whenever you create a function within an object. The C/AL code in this function trigger is executed whenever the function is called.

The code in an event trigger is executed by the fin.exe program when a specified event happens. When the user leaves a field, presses F3 (Microsoft Navision standard shortcut for inserting a new record) or closes a form, event triggers fire.

Code in an event trigger must stand completely alone. It should not depend on an event occurring first. The user normally controls these events.

Documentation Trigger

When referring to "Internal Documentation", this means the documentation that is written directly into the objects themselves. There are three types of internal documentation: the Documentation trigger of each object, code comments and Description fields in the field definitions. For new objects, the only documentation that might be necessary is in the Documentation Trigger.

If you look at the code within each object, notice that the Documentation trigger is the first trigger within the object. This trigger is never executed, so it should not contain code. Instead, it is designed such that you can enter comments about the object.

The use of the Documentation trigger for a new object (that you create in this project) is optional. Here is an example of documentation:

Microsoft Business Solutions			

Project: Solution Development Training			
jtd: John T. Doe			
No.	Date	Sign	Description

001	05.05.2004	jtd	Create new Object

Note that the initials of the developer responsible, "jtd," appear along with a reference number (001). Each new object you create should be given the 001 sequence number. Then, when you modify the object for the first time, you document it by writing sequence number 002. The second modification would be documented as 003, and so on.

Table Event Triggers

There are four events that fire for a table object. There are also two event triggers for every field in a table.

- **OnInsert:** This trigger fires when the user inserts a record or the INSERT function is called with a parameter of TRUE. On a typical form, the insert does not occur until the user successfully leaves the primary key fields that are displayed on the form. If no primary key fields are being displayed on the form, the insert will be delayed until at least the record is changed. You can set the DelayedInsert property on a form to ensure that a record is not inserted until the user leaves the entire record.
- **OnModify:** This trigger fires when the user modifies a record or the MODIFY function is called with a parameter of TRUE. On a form, the modification does not occur until the user leaves the record.
- **OnDelete:** The OnDelete trigger fires when the user deletes a record or the DELETE function is called with a parameter of TRUE. On a form, the delete does not occur until the user presses F4 and clicks the **Yes** button on the confirmation dialog (there is no confirmation dialog on records deleted in code).

NOTE: When the INSERT, MODIFY and DELETE commands are used in code, the default value is FALSE. Only when the TRUE parameter is set will these triggers fire.

- **OnRename:** This trigger fires when the user renames a record (this involves changing any of the primary key fields) or the RENAME function is called (this trigger is not dependant on a parameter value). On a typical form, the rename does not occur until the user changes the primary key fields and leaves them.

Each field in a table also has these two triggers available:

- **OnValidate:** The OnValidate trigger fires when the user changes a field and leaves that field. This trigger fires before any other trigger can occur, for example, OnModify, OnInsert or OnRename (this includes form triggers that will be discussed later). You can use this trigger to make sure that what the user entered into this field is valid, or to populate other fields with values based on what the user entered.
- **OnLookup:** The OnLookup trigger fires when the user clicks on the lookup button for that field. Any code whatsoever in this trigger (even comments) will cause C/SIDE to run only that trigger and ignore the default lookup that could have been set up by a table relation. For most fields this trigger is not used for that very reason. A table relation is usually used instead.

Form Event Triggers

The events for form triggers are those that deal with the form opening and closing, with being activated and deactivated, and with retrieving and modifying records for the form.

The initial triggers that run for a form are the OnInit and OnOpenForm triggers. The record and controls are not yet available in the OnInit trigger. When the form is not the active window, but becomes the active window, the OnActivateForm trigger is run, just as when it is deactivated, the OnDeactivateForm trigger is run.

The more complicated triggers are those that fire when records are handled in the form. A form trigger exists for every step of a record's handling in a form. For instance, the OnFindRecord trigger fires when a form is opened and a record is retrieved, and the OnAfterGetRecord fires when the record is retrieved but not yet displayed. If the form uses a table box, the OnAfterGetCurrRecord fires when only the current record is retrieved. The OnBeforePutRecord trigger fires when the record is about to be saved, and the OnCloseForm trigger fires when the form is about to close.

It is important to note that there are three form event triggers, OnInsertRecord, OnModifyRecord and OnDeleteRecord, which correspond to triggers at the table level. If you use code at both the form and the table level, the triggers at the form level will be executed first, followed by the triggers at the table level.

NOTE: These are just a few of the table, and form event triggers. See the C/SIDE Reference Guide for detailed explanations of all the event triggers.

Codeunit Triggers

Codeunits are simple since they only contain two triggers by default, a Documentation trigger and an OnRun trigger. As previously discussed, the Documentation trigger is for information only. In the OnRun trigger, you can write code that you want to be executed when the codeunit is run.

Complex Data Types

Complex data types are those that have more than just a value and operators. Simple data types like integers have a value. You can add or subtract integers to get new values, and so on. Complex data types, however, have properties or methods along with a value (or values). For instance, a form data type has a value (the form object), but it also has methods like RUN that displays the form to the user. It also has properties like LOOKUPMODE.

Record Data Type

The first complex data type you use in this project is the record data type. You can access tables through record data types. These variables are called record variables. Record variables are the guards that protect the data. They are used to access, change, and manipulate the data. There are a few concepts that you must keep in mind when working with record variables:

- The record variable is a place in memory for ONE record from the associated table.
- The record variable is also "conscious" of the entire set of records contained in the table.
- The record variable is NOT the record in the table, or the table itself.

Record variables have fields just like the table that they access. For example, a record variable with a subtype of Customer would have fields like **No.**, **Name**, **Address**, **City**, and so on. Changing the fields of a record variable does not change the record in the database.

For each event trigger, C/SIDE provides two record variables for you to use. One is called Rec and can be considered the current record for the event. The other is called xRec and can be considered the previous version of Rec. These two record variables allow you to view the current record and make any needed changes. They can have slightly different meanings depending on which table event trigger they are being used in.

Table Event Trigger	Rec	xRec
OnInsert	The record about to be inserted	The last record the user was on before pressing F3
OnModify OnRename	Holds the value of the fields about to be put into the database	Holds the previous values of the fields (the values when the record was retrieved)
On Delete	Holds the value of the fields that are about to be deleted	Holds the previous values of the fields before the user changed the record

Retrieving a Record

To retrieve a record or to navigate to the next record in a record set, you must use one of the functions described in the following. These functions allow you to search for specific records according to certain criteria, or to move forward or backward in a record set:

GET: This function uses the primary key values to find the matching record in the database. Once the record is found, it sets the record variable's fields to the values in the database.

Examples

```
Customer.GET('50000');//No. is the primary key field
ItemUnitOfMeasure.GET('70000','PCS');//Compound primary key
Customer.GET(CustNo);//Using a variable
IF Customer.GET(CustNo) THEN //Avoids the error if GET
fails to find the record
```

FIND: FIND works much like GET, except that FIND respects, and is limited by, the current filters setting. In addition, FIND can be instructed to look for records where the key value is equal to, larger than or smaller than the search string. The parameter for FIND is a string that can be any one of the following: +, -, =, <, or >

Examples

```
SemRegistrationLine.FIND('-');//Find the first record in
the record set
SemRegistrationLine.FIND('+');//Find the last record in the
record set
```

NEXT: The NEXT function retrieves the next record in the filter. The steps parameter allows you to specify the direction and the number of records to skip. A negative number retrieves previous records, and a positive number retrieves following records. The default value is 1. If you enter 3 for the step parameter, the function will skip the next two records in the filter and retrieve the third.

Examples

```
Customer.FIND('-');//Finds the first record in the
Customer table (10000)
Customer.NEXT;//retrieves the next record in the table
(20000)
Customer.NEXT(-1);//retrieves the previous record (10000)
Customer.NEXT(3);//retrieves the third next record (40000)
Customer.FIND('+');//finds the last record in the Customer
table
Customer.NEXT;//returns 0 because there is no "next"
record
```

Defining a Record Set

A set of records can be referred to by the term "record set." A record set is defined by the Key and Filters that the record variable has been assigned. To assign a new key or filter to a record variable, use the method functions SETCURRENTKEY, SETRANGE or SETFILTER.

SETCURRENTKEY: The SETCURRENTKEY function assigns a new key to a record variable. All record variables use the primary key by default. If you would like to sort the records in a different way, you must use this method function. The new key becomes the current key and is used by FIND, NEXT and other functions until another key is selected or until the key is reset to the primary key. What you pass into this function is a list of fields. The function then searches the table's keys to find a matching key.

Examples

```
Customer.SETCURRENTKEY("Search Name"); //Sorts by Search  
Name  
Customer.SETCURRENTKEY("No."); //resets to the primary key
```

SETRANGE: The SETRANGE function provides a quick way to set a simple filter on a field. If you call this function with a field that already has a filter, the system removes that filter before it sets the new one. A range in C/SIDE is of the form "FromValue ToValue." This is the only type of filter that SETRANGE can perform. Once a filter is applied to the record variable, the record set is changed so that it only includes those records that meet the filter criteria. FIND or NEXT functions on a filtered record set will only retrieve records in that record set. The GET function, however, ignores all filters and gets the record from the database if it can.

Examples

```
Customer.SETRANGE("No.", '10000', '50000'); //Includes  
customers with numbers from 10000 to 50000  
Customer.SETRANGE("No.", '30000'); //Includes just 30000  
Customer.SETRANGE("No."); //removes all filters on No.  
Customer.SETRANGE("No.", '111'); //Includes NO records  
(using Cronus sample data)
```

SETFILTER: The SETFILTER function provides a way to set a complex filter on a field. If you call this function with a field that already has a filter, the system removes that filter before it sets the new one or you can use the RESET function to remove filters. You can construct filters using the following operators: (), ..., &, |, <, <=, >, <>, *. You can also use replaceable parameters (%1, %2, and so on) just like the MESSAGE function.

Note that setting a filter with SETFILTER will remove a previous one set by SETRANGE and vice versa.

Examples

```
Customer.SETFILTER("No.", '10000|20000|30000'); //Filters
down to the first three customers
Customer.SETFILTER("Credit Limit (LCY)", '>17500');
//Filters down customers with credit limits over 17,500
Customer.SETFILTER("No.", '>10000 & <> 20000'); //Filter
down to all customers after 10000 but not 20000
```

Using the Database Record

There are many useful functions that work with record variables, some in conjunction with filters and ranges. Common functions are listed below:

INSERT: Use this function to insert a record into a table. It takes the values of the fields in the record variable and inserts those into the table. If you want to run the OnInsert trigger of the table during the insert, you must pass TRUE into the function.

MODIFY: The MODIFY function allows you to update the database record with the values in the record variable that have changed. If you want to run the OnModify trigger of the table during the modification, you must pass TRUE into the function.

MODIFYALL: Like MODIFY, but changes the values of the entire record set. Take care that the filter or range is set or you will modify all records in the table.

DELETE: The DELETE function allows you to erase or remove the database record identified by the primary key values in the record variable. If you want to run the OnDelete trigger of the table during the deletion, you must pass TRUE into the function.

DELETEALL: The method DELETEALL can be used to delete the entire record set. Take care that the filter or range is set or you will modify all records in the table.

RENAME: The RENAME function can be used to change the primary key fields of a record in the database. Like the GET function, you must pass the new primary key values as parameters to this function. It uses the values in the primary key fields of the record variable to find the record in the database and then uses the new values to change the record. This function works the same as the user changing the primary key fields on a form – the change is propagated to all related tables. The code in the OnRename trigger for the record is always run after the RENAME method is used.

INIT: Use the INIT function to initialize a record variable. The function does not initialize primary key fields.

CALCFIELDS: Use this method to force the calculation of FlowFields or BLOBs in a record variable. By default, the record variable does not calculate these types of fields. You must specify the FlowFields or BLOBs that you want to calculate.

COPY: Use this function to copy a record variable to another record variable. All fields, filters, marks and keys are included in the copy.

COUNT: Use this function to count the number of records in the record set, represented by the record variable. This takes into account any Filters that are set and use the current key (which can increase or decrease the speed depending on the filters set).

VALIDATE: Use this function to call the OnValidate trigger for the field you specify. If NewValue is passed in as a parameter, the function first assigns the field with NewValue. The VALIDATE function first checks the TableRelation property and then executes the OnValidate trigger of the field.

Multilanguage Functionality

Microsoft Navision is multi-language enabled. This means that a localized version of Microsoft Navision can present itself in different languages. Users can change the language that is used to display texts, and the change is immediate. There is no need to stop and restart Microsoft Navision.

Before you start working in a multi-language-enabled database, you must set the working language to English – United States. Click TOOLS→LANGUAGE and select English – United States.

To enable multi-language functionality when developing solutions, there are a few guidelines to follow. When creating objects in Microsoft Navision, the Name property of an object must always be in English – United States (ENU), but it should also never be visible to the user.

Think of Name as the internal name of an element. For every field or object that a user will see, you must set the CaptionML property. The CaptionML contains the caption for an element for each language code. The Caption property displays the caption for the selected language and will be defaulted from CaptionML.

Accordingly, you must make sure that text that is displayed in the user interface must be ENU. You can run a test by selecting all objects in the Object Designer, clicking TOOLS→LANGUAGE→EXPORT, saving the file as a text file, and then opening that file in Notepad. There must be no other language code than A1033 for ENU in that text file.

Microsoft SQL Server

Table Relations on Microsoft SQL Server

You will create a number of table relations when creating our master files, including some with conditions. When running Microsoft SQL Server, the TableRelation properties and SQL Server relationships are automatically synchronized when you create a table and when you redesign a table.

Use the Maintain Relationships setting on the Microsoft Navision database to enable and disable the creation and maintenance of foreign key constraints for each TableRelation property of a Navision table.

If you select this option, external tools will have access to the table relationships (foreign key constraints) that exist between the Microsoft Navision tables. These relationships are disabled and are not used to enforce data integrity but are intended for modeling purposes only. For more information about table relationships in the SQL Server Option, see the Application Designer's Guide.

Microsoft Internal Use Only

Test Your Skills – Managing Master Files – Diagnosis

Description

Our seminar management module requires master data. To determine what master data will be needed, we must read the functional requirements again.

For our purposes, the key statements in the functional requirements are those that reveal what the main elements of the seminar module will be. Some of these statements are:

The CRONUS training department holds several different classes, or seminars.

- Each seminar is held in a seminar room.
- Each seminar is taught by an instructor.
- Participants in a seminar come from a company that is set up in Microsoft Navision as a customer.

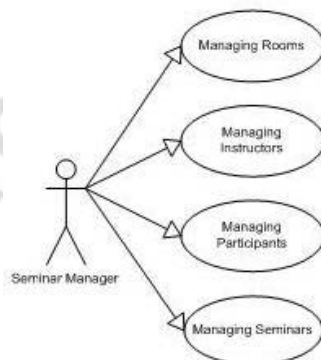
From these statements, we can see that our main objects in the seminar module are going to be seminars, rooms, instructors and participants. With this knowledge, we can then split up the task of creating the master files into use cases.

Use Cases

For this chapter, each use case will consist of managing one master file.

Begin by analyzing the Microsoft Navision contacts, which you will use for the participants, and then move on to developing the master files.

The following use case diagram illustrates the relationships between the master files:



Implementation of Use Case 1 – Managing Participants

Managing Participants – Analysis

Our client's functional requirements describe the management of participants in the following way:

Seminar participants come from a company that is set up in Microsoft Navision as a customer. The participants must be handled separately from the customers. Furthermore, every customer can register several participants for a seminar. Participants cannot be registered for a seminar unless they are connected with a customer. This is necessary because you want to invoice customers for the participation at seminars.

With this information, you can now define how participants will be managed in our module:

Purpose

The system must be able to manage participant information so that our client can register their customers' participation in seminars and invoice them accordingly.

Preconditions

A company with which a participant can be associated must exist as a customer in the Customer table in the program.

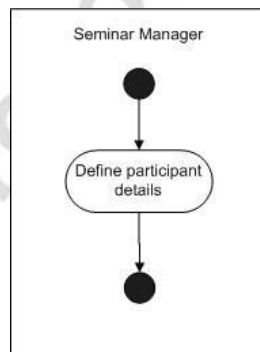
Postconditions

A participant is defined and associated with a customer.

Main Scenario

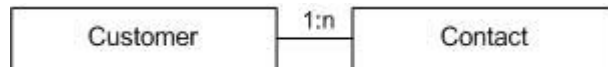
The seminar managers will define participant details. This includes the participant's name, contact information (including e-mail) and association with a defined customer.

Activity Diagram



Managing Participants – Design

It is our design goal to use existing Microsoft Navision functionality as much as possible. We have therefore decided that it would be best to manage participants as "contacts." These are defined in the Contact table, which has a relation to the Customer table. In this way, we can use existing Contact data and functionality.



GUI Design

Forms 5050 Contact Card and 5052 Contact List already exist in Microsoft Navision for the entry and display of contact details. No additional navigation will be necessary.

Functional Design

No additional functions have been defined for these objects.

Table Design

Both tables exist already. No additional fields are necessary.

Managing Participants – Code Walkthrough

Since we are using the existing contact functionality in Microsoft Navision, no additional modifications are necessary. Let's take a look at some Microsoft Navision standards and features. For further detail on these topics, \ refer to the Development I manual or the Application Designer's Guide.

A card form is a form where users can view one record at a time from a master table. For example, the Contact Card form (Form 5050) is used to create, view, and modify contacts in the Contact master table (Table 5050). It is standard for the first tab on a card form to be labeled "General" as you can see on the Contact Card. The **Primary Key** field of the related master table normally appears first in the **General** tab.

A list form is a form where users can view multiple records at a time in a table box. For example, the Contact List form (Form 5052) can be used to lookup many contacts in a table box, and then return the contact value to the card form.

Standard navigation for these forms calls for the list form to be accessed from the card form, and vice versa. For instance, you'll notice that from the Contact Card form, under the **Contact** command button, there is a menu item to the Contact List form with an F5 shortcut. If you follow the menu item, it will open up the Contact List form that also has a **Contact** command button. Under this button, you will find a menu item to the Customer Card form with a SHIFT + F5 shortcut. These links and shortcuts are standard navigation features in Microsoft Navision and should be included in all Card and List forms.

On the Contact List form, the "RunFormLinkType" property of the Card menu item is set to "OnUpdate." The RunFormLinkType property is associated to the form you are launching from this menu item. OnUpdate means that the launched form will be updated whenever the source form is updated.

For example, open the Contact Card form from the Contact list form and notice what happens to the Card form when you scroll through the List form. Compare this to what happens if the RunFormLinkType property is set to "OnOpen."

Standard Microsoft Navision naming conventions for these forms and tables are:

Type	Naming Convention	Example
Master Table	Singular Record	Customer (Table 18)
Card Form	Name of Table + 'Card'	Customer Card (Form 21)
List Form	Name of Table + 'List'	Customer List (Form 22)

Some other items of note that will be used in the upcoming exercises:

- The DataCaptionFields property for Table 5050 Contacts is set to **No., Name**. These are the fields shown in the Contact Card caption area. For usability purposes, you should specify the DataCaptionFields or DataCaptionExpr property for all tables and forms.
- The command buttons at the bottom of forms normally have the HorzGlue and VertGlue properties set to Right and Bottom to ensure that they stay in place relative to the right and bottom when a form is resized. As the defaults are set to Left and Top, these are commonly forgotten properties.
- Microsoft Navision has code to validate City and Post Codes that you will be using in the exercises. You can see this code in Contacts by looking at the OnValidate and OnLookup triggers for the **City** and **Post Code** fields in the Contacts table.

Implementation of Use Case 2 – Managing Rooms

Managing Rooms – Analysis

The next step is to create the tables to manage the seminar rooms in which the seminars are held.

Our client's functional requirements describe the management of seminar rooms in this way:

Each seminar is held in a seminar room. Some seminars are held in-house and some are outsourced. We want to be able to track costs and prices for our rooms (that is, the rental fee). If a seminar takes place in-house, we want to assign one of our rooms.

With this information, we can define how seminar rooms will be managed in our module:

Purpose

Managing the seminar rooms gives our client a way to track the availability of rooms, their costs and the prices that can be charged for renting the rooms.

Preconditions

None.

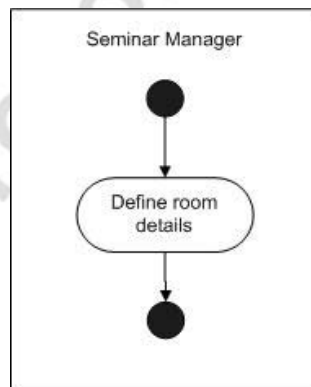
Postconditions

We will have created a Seminar Room table and a simple form from which this table can be accessed.

Main Scenario

The seminar managers describe room details. These details include the room name, address, the maximum capacity, and assigned cost and price.

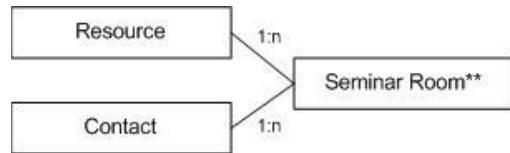
Activity Diagram



Managing Rooms – Design

From our analysis of the room management process, we can see that we only need one basic table and form.

The tracking of prices and costs is the only requirement that is not so simple. We know, however, that the Resource table in the standard Microsoft Navision functionality provides a way of tracking this kind of information for company resources such as employees or machinery.



**new table

GUI Design

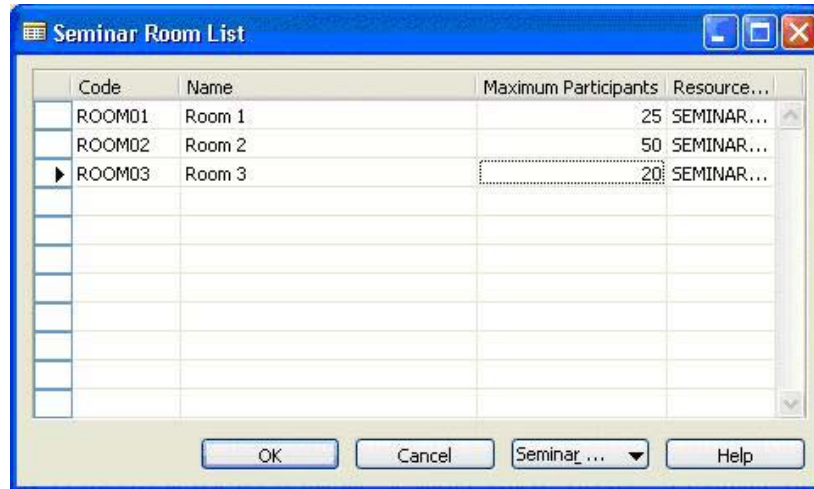
The table should be accessible through two types of forms: a card form for easy data entry and a tabular form to provide an overview of the data in the table.

Seminar Room Card (Form 123456703): This form enables the entry of room details.

GENERAL TAB

COMMUNICATION TAB

Seminar Room List (Form 123456704): This form displays room details.



Functional Design

There are no functions specified for these objects.

Table Design

The following tables will be required:

123456702 Seminar Room will hold the Seminar Room Name, Address, Resource No. and other room-specific information.

Managing Rooms – Development

The task now is to create the objects for managing rooms in the new module. As you can see in the GUI design, we need one table and two forms from which to access the table.

Exercise 1 – Creating the Table and Forms for Seminar Rooms

Microsoft Navision provides a Comment table that we can use with our tables. We would also like to use Microsoft Navision's extended text functionality with our Seminars. (You can learn more about this functionality in the Navision Online Help.) To follow these Microsoft Navision standards, we will need to include our new tables into these Navision tables.

1. In table 97 Comment Line, add the option Seminar Room to the options for the field **Table Name**.
2. In tables 279 Extended Text Header and 280 Extended Text Line, add the option Seminar Room to the options for the field **Table Name**.

3. Create table 123456702 Seminar Room with the following fields:

No.	Field Name	Type	Length	Comment
1	Code	Code	10	Must not be blank.
2	Name	Text	30	
3	Address	Text	30	
4	Address 2	Text	30	
5	City	Text	30	
6	Post Code	Code	20	Relation to table 225 Post Code.
7	Country Code	Code	10	Relation to table 9 Country.
8	Phone No.	Text	30	
9	Fax No.	Text	30	
10	Name 2	Text	50	
11	Contact	Text	50	
12	E-Mail	Text	80	
13	Home Page	Text	90	
14	Maximum Participants	Integer		
15	Allocation	Decimal		Must not be editable.
16	Resource No.	Code	20	Relation to table 156 Resource, where Type=Machine.
17	Comment	Boolean		FlowField; Checks whether any lines exist on the Comment Line table for the Seminar Room table and the corresponding seminar room code. Must not be editable.
18	Internal/External	Option		Options: Internal,External
19	Contact No.	Code	20	Relation to table 5050 Contact

The primary key is **Code**. Set the property to specify form 123456704 as the lookup form for this table.

HINT: For all fields, you should set the **Caption value** to the **Name**. To save time, use F8 (Copy Previous shortcut).

4. Use the wizard to create form 123456703 Seminar Room Card as shown in the GUI design. Set the property to specify that this form will be updated when activated.

- Add a menu button and menu items as follows:

Menu Button	Options	Comment
Seminar Room	List (F5)	Opens the lookup form.
	Comments	Opens form 124 Comment Sheet for the selected entry.
	Extended Texts	Opens form 386 Extended Text for the selected entry.

- Add a command button next to the **Code** field to provide access to the Comment Sheet for the corresponding record.

HINT: Copy the command button with the pencil picture from the existing form 21 Customer Card and paste it into your new form. (The local variables for these objects will be copied as well.) Set the **RunFormLink** property for this button so that only the comments corresponding to the selected Seminar Room are displayed.

- Add a command button next to the **E-mail** field from which an e-mail can be sent.

HINT: Copy the command button from the existing form 21 Customer Card and paste it into your new form.

- Add a command button next to the **Home Page** field that will open a hyperlink entered into the field.

HINT: Copy the command button from the existing form 21 Customer Card and paste it into your new form.

5. Use the wizard to create form 123456704 Seminar Room List as shown in the GUI design. The only fields necessary on this form are **Code**, **Name**, **Maximum Participants** and **Resource No**.

- Set the properties to specify that the lines on this form are not editable and that the **Name** field will be "glued" to both sides of the form so that it expands with the form.

- Add a menu button and menu items as follows:

Menu Button	Options	Comments
Seminar Room	Card (SHIFT + F5)	Opens form 123456703 Seminar Room Card for the selected entry. The content of the card should change when the user selects a different entry in the list form.
	Comments	Opens form 124 Comment Sheet for the selected entry.
	Extended Texts	Opens form 386 Extended Texts for the selected entry.

NOTE: When creating tabular-type forms with the wizard in the Object Designer, fields with a data type of code, decimal and integer are created with a width of 1700 rather than a width of 1650, as is standard. Correct the field widths when creating a tabular form with the wizard.

Exercise 2 – Adding Code for Seminar Rooms

Our first task is to improve the functionality of the **City** and **Post Code** fields. We know that in Microsoft Navision, the standard table Post Code links cities and post codes. Therefore, we want to write code so that when a user enters a City, the program fills in the corresponding Post Code value from the Post Code table, and vice versa. The existing table Post Code has functions that will help us in doing this. (You have already seen this code used in the Contacts walkthrough.)

1. In the Seminar Room table, create a global C/AL variable called PostCode for the record Post Code.
2. We want to validate the **City** field using the ValidateCity function from the Post Code table. Use the C/AL Symbol Menu to lookup the parameters for this function and insert the following code in the City – OnValidate trigger:

```
PostCode.ValidateCity(City,"Post Code");
```

3. Enter code so that when the user performs a lookup on the **City** field, the program runs the Post Code table's LookUpCity function. Use the C/AL Symbol Menu to see what parameters must be sent to the function when calling it. When calling the function, set the ReturnValues parameter to TRUE.

Solution: The City – OnLookup trigger code will be:

```
PostCode.LookUpCity(City, "Post Code", TRUE);
```

4. Use the Post Code table's ValidatePostCode function to validate the value entered by the user in the **Post Code** field.

Solution: The Post Code – OnValidate trigger code will be:

```
PostCode.ValidatePostCode(City, "Post Code");
```

5. Enter code so that when the user performs a lookup on the **Post Code** field, the program runs the Post Code table's LookUpPostCode function. When calling the function, set the ReturnValues parameter to TRUE.

Solution: The Post Code – OnLookup trigger code will be:

```
PostCode.LookUpPostCode(City, "Post Code", TRUE);
```

The next task is to improve the functionality of the **Resource No.** and **Contact No.** fields by retrieving information. In the following steps, we want to create code so that when a user enters a **Resource No.** or a **Contact No.**, if the **Name** field is empty, it will be filled with the **Name** from the corresponding **Resource** or **Contact** record.

6. Create two global record variables, one for the Resource table and one for the Contact table.
7. Enter code so that when validating the **Resource No.** entered by the user, if the **Name** field in the Seminar Room table is empty, the program looks up the **Name** field in the corresponding Resource record and assigns it to the **Name** field in the Seminar Room table.

Solution: Enter this code in the Resource No. – OnValidate trigger:

```
IF Resource.GET("Resource No.") AND (Name = '') THEN  
    Name := Resource.Name;
```

8. Enter code so that when validating the **Contact No.** entered by the user, if the **Name** field in the Seminar Room table is empty, the program looks up the **Name** field in the corresponding Contact record and assigns it to the **Name** field in the Seminar Room table.

Solution: Enter this code in the Contact No. – OnValidate trigger:

```
IF Cont.GET("Contact No.") AND (Name = '') THEN  
    Name := Cont.Name;
```

9. Create two global record variables, one for the Comment Line and one for the Extended Text Header tables.
10. Enter code in the appropriate table trigger so that when a record is deleted, any corresponding records in the Comment Line table are deleted as well.

***HINT:** Use the DELETEALL function to delete the records.*

Solution: Enter this code in the OnDelete trigger:

```
CommentLine.SETRANGE("Table Name", CommentLine."Table  
Name"::"Seminar Room");  
CommentLine.SETRANGE("No.", Code);  
CommentLine.DELETEALL;
```

11. Enter code in the appropriate table trigger so that when a record is deleted, any corresponding records in the Extended Text Header table are deleted as well.

***HINT:** Use the DELETEALL function once again, but this time, make sure that the parameter is set to TRUE so that the code in the delete trigger of the Extended Text Header will fire. The reason we need to set the parameter to TRUE for Extended Text Header, where we didn't for Comment Line, is that the OnDelete trigger code for Extended Text Header includes code to delete the associated Extended Text Line records.*

Solution: Add this code to the OnDelete trigger:

```
ExtTextHeader.SETRANGE("Table Name", ExtTextHeader."Table  
Name"::"Seminar Room");  
ExtTextHeader.SETRANGE("No.", Code);  
ExtTextHeader.DELETEALL(TRUE);
```

You can now add code to the triggers in the forms to ensure that the form works properly.

12. In form 123456703 Seminar Room Card, enter code in the appropriate form trigger so that after the program has retrieved the record for the form, the program removes the filter on the **Code** field for the table.

Solution: Add this code to the Form – OnAfterGetRecord trigger:

```
SETRANGE ( Code ) ;
```

13. In the Seminar Room Card form, in the C/AL code for the **E-mail** command button, delete the existing code and enter code in the appropriate trigger so that when the user "pushes" the button, the program creates a new mail message.

***HINT:** A local C/AL variable for the Mail codeunit already exists in this trigger. Use the OpenNewMessage function from this codeunit.*

Solution: Add this code to the E-mail command button – OnPush trigger:

```
Mail.OpenNewMessage( "E-Mail" );
```

Implementation of Use Case 3 – Managing Instructors

Managing Instructors – Analysis

Our client's functional requirements describe the management of instructors in the following way:

Each seminar is taught by an instructor, who can be either an employee from our company or from an external company. To make use of our existing resource information, each instructor must be set up as a Resource.

With this information, we can now define how instructors will be managed in our module.

Purpose

The instructor information is managed to allow our client to assign instructors to seminars.

Preconditions

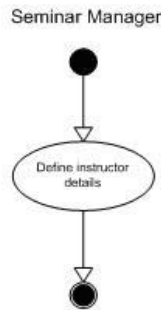
We will use the existing Microsoft Navision Resource table.

Postconditions

An instructor will be defined.

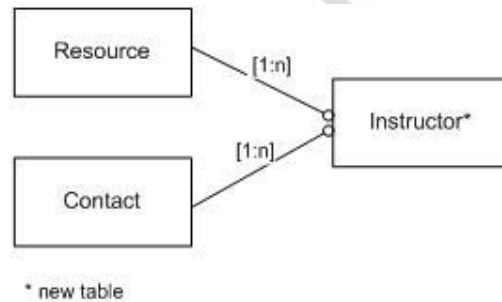
Main Scenario

The seminar managers will define instructor details. This includes the instructor's name, personal information and resource number.

Activity Diagram**Managing Instructors – Design**

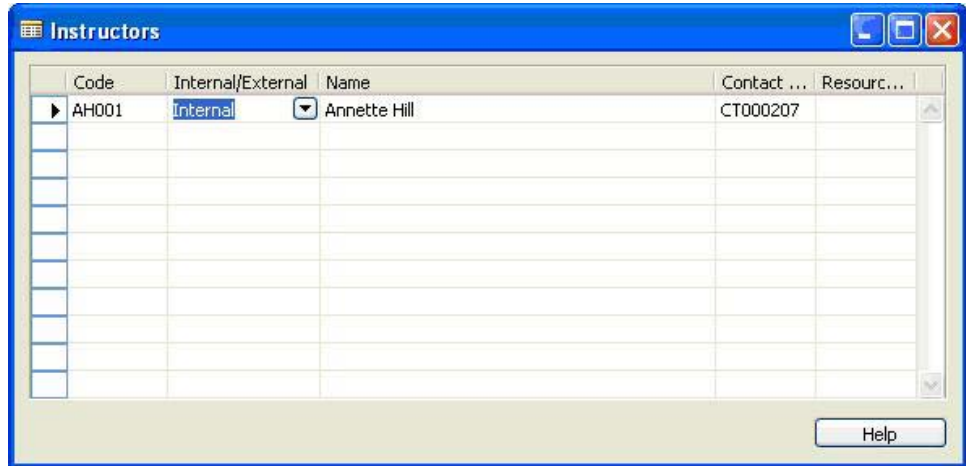
According to the analysis of the management of instructor information, we need to create one table for managing the instructors.

We manage instructors in the same way as we manage the seminar rooms. This means that we can define them as resources to track their assignments, costs, and prices.

**GUI Design**

Start by designing the simplest forms and then move on to the more complex ones.

Instructors (Form 123456705): This form enables the entry of instructor information.



Functional Design

No functions have been defined for these objects.

Table Design

The following tables will be required:

123456703 Instructor will hold the Instructor's name, Resource No., Contact No. and whether they are an employee or external.

Managing Instructors – Development

Our task now is to create the objects to enable the management of instructors in the new module.

Exercise 3 – Creating the Tables and Forms for Instructors

1. Create Table 123456703 Instructor with the following fields:

No.	Field Name	Type	Length	Comment
1	Code	Code	10	Must not be blank.
2	Name	Text	30	
3	Internal/External	Option		Options: Internal, External
4	Resource No.	Code	20	Relation to table 156 Resource, where Type=Person.
5	Contact No.	Code	20	Relation to table 5050 Contact

The primary key is the **Code** field. Set the property to specify form 123456705 Instructors as the lookup form for this table.

***NOTE:** Don't forget to specify the `DataCaptionFields` for the table and a `Caption` for each field in the table.*

2. In the Instructor table, enter code so that when the program validates the value in the **Resource No.** field, if the **Name** field in the Instructor record is empty, the program fills it with the **Name** field from the corresponding Resource record. Do the same for the **Contact No.** field.

***HINT:** You wrote very similar code in the previous exercise.*

3. Create form 123456705 Instructors as shown in the GUI design.

Implementation of Use Case 4 – Managing Seminars

Managing Seminars – Analysis

Our client's functional requirements describe the handling of seminars in the following way:

The CRONUS training department holds several different seminars. All seminars have a fixed duration and allow a maximum and a minimum number of participants. They can be overbooked in some cases, depending on the capacity of the assigned room. They can be cancelled if there are not enough participants. The price of a seminar is fixed. We would like to take advantage of the current Job functionality in Microsoft Navision and define each seminar as a job. When a seminar is completed, the seminar should be posted as a job, with additional seminar-specific information.

With this information, we can now define how seminars will be managed in our module.

Purpose

The Seminar table contains the main information about the seminars offered by the training academy.

Preconditions

Before setting up a seminar, setup information such as seminar numbering must exist. Instructor and participant information must also exist. The Job table must exist so that seminars can be defined as jobs.

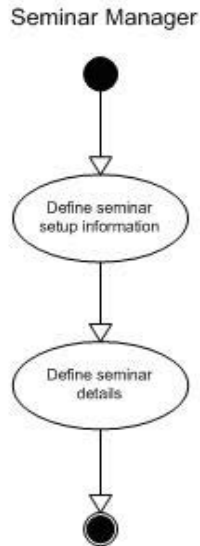
Postconditions

A new seminar is created with all the relevant relations to connected files.

Main Scenario

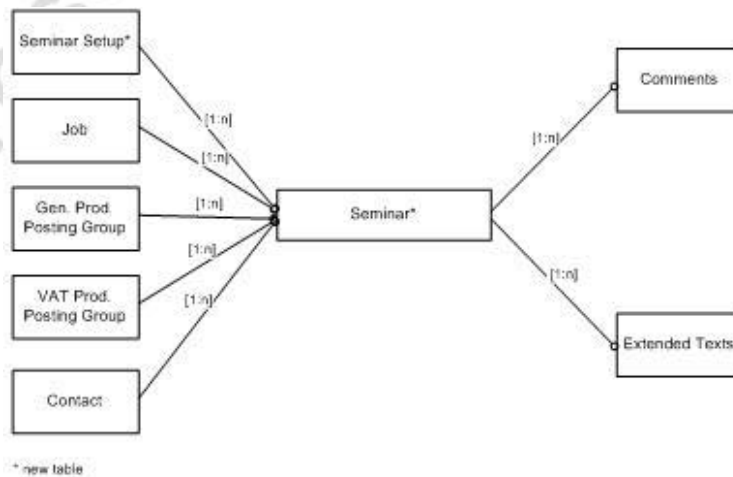
When defining a new seminar, seminar managers will describe the course details, including information such as the seminar name, duration, price, maximum and minimum number of participants, and a job code.

Activity Diagram



Managing Seminars – Design

As we have seen in the analysis of the seminar management process, we need two tables to manage seminar information. The first table will simply be a setup table where the user can define seminar numbering. There will then be a table to track the seminar profile information.



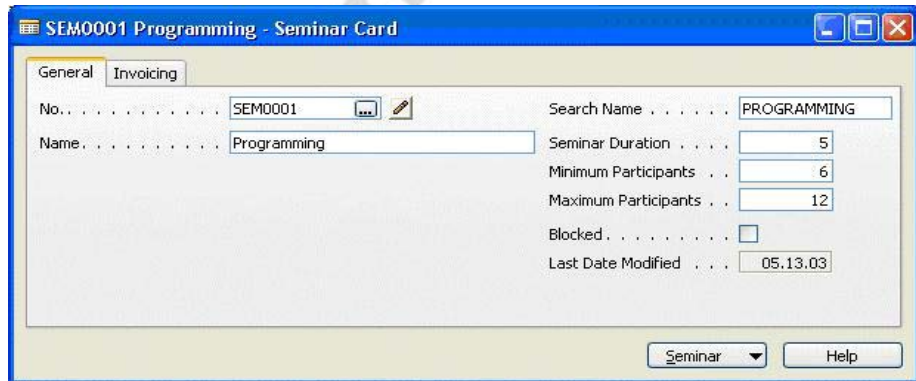
GUI Design

We start by designing the simplest forms and then move on to the more complex ones.

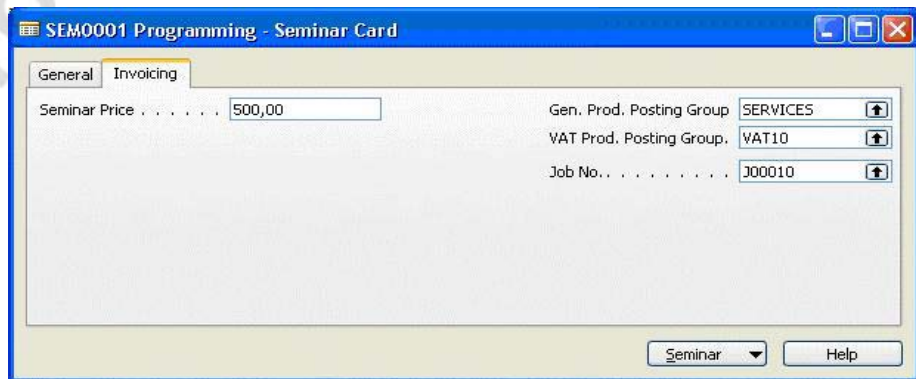
Seminar Setup (Form 123456702): This form enables the entry of setup information for the seminar module.



Seminar Card (Form 123456700): This form enables the entry of seminar details.



GENERAL TAB



INVOICING TAB

Seminar List (Form 123456701): This form displays seminar information.

No.	Name	Seminar Duration	Seminar Price	Gen. Pro...	VAT Pro...	Job No.
SEM0001	Programming	5	500,00	SERVICES	VAT10	J00010
SEM0002	Solution Development	10	500,00	SERVICES	VAT10	J00010

Functional Design

No additional functions have been defined for these objects.

Table Design

The following tables will be required:

Table 123456701 Seminar Setup will hold the number series used for Seminars and Seminar Registrations.

Table 123456700 Seminar will hold seminar specific information such as name, maximum and minimum participants and posting groups.

Managing Seminars – Development

We are now ready to create the tables and forms to manage seminars.

Exercise 4 – Creating the Tables and Forms for Seminars

You must carry out the following tasks to create the seminar master files and forms:

1. In table 97 Comment Line, add the option Seminar to the options for the field **Table Name**.
2. In tables 279 Extended Text Header and 280 Extended Text Line, add the option Seminar to the options for the field **Table Name**.

3. Create **Table 123456701 Seminar Setup** with the following fields:

No.	Field Name	Type	Length	Comment
1	Primary Key	Code	10	
2	Seminar Nos.	Code	10	Relation to 308 No. Series table.
3	Seminar Registration Nos.	Code	10	Relation to 308 No. Series table.
4	Posted Sem. Registration Nos.	Code	10	Relation to 308 No. Series table.

Per Microsoft Navision standards for setup tables, the key for this table is the **Primary Key** field, which is left blank.

4. Create **Table 123456700 Seminar** with the following fields:

No.	Field Name	Type	Length	Comment
1	No.	Code	20	An alternate search field is the Search Name field.
2	Name	Text	50	
3	Seminar Duration	Decimal		Decimal places 0:1
4	Minimum Participants	Integer		
5	Maximum Participants	Integer		
6	Search Name	Code	30	
7	Blocked	Boolean		
8	Last Date Modified	Date		Must not be editable.
9	Comment	Boolean		FlowField; CalcFormula checks whether lines exist in the Comment Line table for the seminar. Must not be editable.
10	Job No.	Code	20	Relation to table 167 Job.
11	Seminar Price	Decimal		AutoFormat type is 1.

No.	Field Name	Type	Length	Comment
12	Gen. Prod. Posting Group	Code	10	Relation to table 251 Gen. Product Posting Group.
13	VAT Prod. Posting Group	Code	10	Relation to table 324 VAT Product Posting Group.
14	No. Series	Code	10	Must not be editable. Relation to table 308 No. Series.

The primary key for this table is the **No.** field with a secondary key of **Search Name**. Set the properties to specify form 123456701 as the lookup and drill down form for this table.

5. In the Seminar table, enter code to perform the following validation tasks:
 - When the user changes the **No.** value from what it was previously, the program gets the Seminar Setup record and uses the TestManual function of the NoSeriesManagement codeunit to test whether or not the number series is allowed to be manually changed. The program then sets the **No. Series** field to blank.

***HINT:** Use the xRec variable to test whether the No. field has been modified. You will need a record variable with a subtype of Seminar Setup and a Codeunit variable of subtype NoSeriesManagement to perform this task. Remember that there is only one record in the Seminar Setup table and the primary key field is left null.*

Solution:

```

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

- When the user enters or changes a value in the **Name** field, if the **Search Name** is still equal to the uppercase value of the previous Name or if the Search Name is blank, the program assigns the new Name to Search Name.

***HINT:** Use the function UPPERCASE when testing the Search Name field.*

Solution

```
IF ("Search Name" = UPPERCASE(xRec.Name)) OR  
("Search Name" = '') THEN  
"Search Name" := Name;
```

- When the user enters or changes a value in the **Job No.** field, the program retrieves the corresponding Job record and checks that the **Blocked** field is set to FALSE.

HINT: Use the TESTFIELD function to check a field's value.

Solution

```
Job.GET("Job No.");  
Job.TESTFIELD(Blocked, FALSE);
```

- When the user changes the Gen. Prod. Posting Group to a new value from what it was previously, the program checks that the function ValidateVatProdPostingGroup for the Gen. Product Posting Group table returns true. If it does, the program sets the VAT Prod. Posting Group to the Def. VAT Prod. Posting Group value from the Gen. Product Posting Group table.

Solution

```
IF xRec."Gen. Prod. Posting Group" <> "Gen. Prod. Posting  
Group" THEN  
IF  
GenProdPostingGrp.ValidateVatProdPostingGroup(GenProdPostin  
gGrp,"Gen. Prod. Posting Group") THEN  
VALIDATE("VAT Prod. Posting Group",GenProdPostingGrp."Def.  
VAT Prod. Posting Group");
```

6. In the Seminar table, create a new function named AssistEdit with a return type of Boolean. In this function, enter code that checks for a Seminar Nos. series in the Seminar Setup table. If it finds that there is one, the program uses the SelectSeries function in the NoSeriesManagement codeunit to check the series number. If this function returns true, the program uses the SetSeries function in the NoSeriesManagement codeunit to set the **No.** field, and the program then exits TRUE.

Solution

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

7. In the Seminar table, enter code in the appropriate table triggers to perform the following tasks:
- Document your code.
 - When a record is inserted, if the **No.** field is blank, the program gets the Seminar Setup record and run the InitSeries function of the NoSeriesManagement codeunit to initialize the series.

Solution: Enter the following code in the OnInsert trigger:

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

- When a record is modified or renamed, the program sets the **Last Date Modified** to the current date.

HINT: Use the TODAY function to get the current date.

Solution: Enter the following code in the OnModify and OnRename triggers:

```
"Last Date Modified" := TODAY;
```

- When a record is deleted, the program also deletes the corresponding records from the Comment Line table and the Extended Text Header table.

HINT: You created similar code for the Seminar Room comment and extended text records.

8. Create form 123456702 Seminar Setup as shown in the GUI design.
9. Enter code in the appropriate trigger of the Seminar Setup form so that when the user opens the form, the program resets the record, and if it does not get a record, it inserts a new one.

Solution: Enter the following code in the Form – OnOpenForm trigger:

```
RESET ;
IF NOT GET THEN
    INSERT ;
```

10. Create form 123456700 Seminar Card as shown in the GUI design.
 - Set the property to update the form when it is activated.
 - Add a menu button and menu items as follows:

Menu Button	Options	Comment
Seminar	List (F5)	Opens the lookup form.
	Comments	Opens the form 124 Comment Sheet for the selected entry.
	<Separator>	
	Extended Texts	Opens the form 386 Extended Texts for the selected entry.

- Add a command button next to the **No.** field to provide access to the Comment Sheet for the corresponding record.
- Set the RunFormLink property for this button so that only the comments corresponding to the selected Seminar are displayed.
- Enter code in the appropriate trigger so that when the user clicks the AssistEdit for the **No.** field, the program runs the AssistEdit function, and if it returns true, updates the current form.

Solution: Enter the following code in the No. - OnAssistEdit trigger:

```
IF AssistEdit THEN
    CurrForm.UPDATE ;
```

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

Solution: Enter the following code in the Form - OnAfterGetRecord trigger:

```
SETRANGE ( "No. " );
```

11. Create form 123456701 Seminar List according to the GUI design. Include the fields **No.**, **Name**, **Seminar Duration**, **Minimum Participants** (not visible), **Maximum Participants** (not visible), **Seminar Price**, **Gen. Prod. Posting Group**, **VAT Prod. Posting Group** and **Job No.**
 - Make the form not editable.
 - Glue the **Name** field to both sides of the form.
 - Remember to reset the width of fields with data type Code, Decimal and Integer to 1650.
 - The navigation from this form is be the same as the Seminar Card form, except that instead of a List option, there will be an option called Card (SHIFT + F5) from which the Seminar Card form will open for the selected seminar.

Testing Master Files

We have now completed the exercises to create the master files for our seminar management module. You have probably tested pieces as you worked on them, but now try to run this test script to check your work. If you run into areas that don't function as expected, try to figure out how to get them working. Remember not all pieces have been coded yet, so the test script may seem incomplete.

***NOTE:** In order to test the Email functionality, the system you are using will need to have Outlook running with a profile set up.*

1. Start by selecting form 123456702 Seminar Setup in the Object Designer and clicking **Run**. The fields in this form are blank as we haven't set them up yet. Click the lookup button for **Seminar Nos.** to open the No. Series form. We have setup our seminars to use the standard Microsoft Navision numbering functionality. Setup all three numbers with values of your choosing, set to Default automatically and close and re-open the Seminar Setup form. See that your values were saved.

2. Select form 123456700 Seminar Card in the Object Designer and click **Run**. Tab off the **No.** field and see that a number is assigned with the values you set up in step 1. Fill in the fields on the **General** tab and then move to the **Invoicing** tab. Use the lookup to select a Gen. Prod. Posting Group value with a Default VAT Prod. Posting Group set and see the VAT Prod. Posting Group fill in automatically. Click F5 to move to the Seminar List form. Close the Seminar List form.
3. Still on the Seminar Card form, use the menu items on the **Seminar Command** button to enter comments and extended texts. When you have finished close the Seminar Card form.
4. We have designed the Seminar module so that Instructors are set up as Resources and Contacts in our system. Set up a new Resource (type Person) and Contact that you will use to test the Instructor functionality. When you are done select form 123456705 Instructors in the Object Designer and click **Run**. Enter a new instructor using the **Resource** and **Contact** lookup buttons to select the Resource and Contact you set. Note that when you use the **Resource** lookup, all the Resources in the list should be of type Person. Try entering a **Resource No.** or **Contact No.** that doesn't exist to test your validation. When you have finished close the Instructors form.
5. We have designed the Seminar module so the Rooms are set up as Resources of type Machine and Contacts in our system. Set up a new **Resource** and **Contact** that you will use to test the Seminar Room functionality. When you are done select form 123456703 Seminar Room Card in the Object Designer and click **Run**. Create a new Seminar Room by entering a value in the **Code** field. We added code so that the **Name** field would be defaulted from the **Resource** or **Contact** if the **Name** field was blank. See if that code is functioning. On the **Communications** tab, try entering an E-Mail address and Home Page and use the command buttons next to them. Check the menu items under the Seminar Room command button.

You have now finished testing your master files!

Test Your Knowledge

Review Questions

1. Where is internal documentation located for new objects?
2. To ensure that Microsoft Navision's multilanguage functionality is properly enabled, which property must be set for controls?
3. Suppose that in the Seminar table, you want to make sure that the value in the **Minimum Participants** field is always less than the value in the **Maximum Participants** field. You want to perform this check whenever a record is inserted, whenever a record is changed and whenever a value is entered in the **Maximum Participants** field (and the **Minimum Participants** field is not empty). Which table and field event triggers would you use for these three checks? What code would you use to perform these checks?
4. What do the Rec and xRec record variables do?
5. What function would you use to retrieve a specific record using its primary key?
6. Using the table example Employee below, answer the following questions:
 - Assuming you are working with the record variable EmployeeVar, what code would you write to get the record for Employee No. 3475? What code would you write to jump to the fourth next record?
 - What code would you write to jump to the last record in the table?
 - What code would you use to define a record set to include the records between employee number 3475 and 6434?
 - What code would you use to define a record set to include the records greater than 5834?
 - What code would you use to define a record set for employees in the Purchasing department only? What code would you use to remove the filter on department?
 - What code would you use to change Ann Smith's department from Sales to Purchasing?
 - What code would you use to delete all Receivables employees?

Table: Employee (key = Employee No.)

Employee No.	Name	Department
5834	John Doe	Purchasing
3723	Ann Smith	Sales
3475	Bill Skaggs	Receivables
6434	Todd Lawrence	Sales
9482	Janet Davila	Receivables
0980	Susan Morris	Purchasing
9483	Rick Hamilton	Sales

7. What is the standard shortcut to open a Card form from the associated List form? What about to open the List form from the Card form?
8. What two triggers does a Codeunit have by default?
9. Name two ways to set the lookup form for a table field. What happens if both are set?

Conclusion

Chapter Summary

You have now created our master files and user interfaces. In doing so, you examined triggers and looked at how they might be used for stand-alone code. You learned about complex data types, specifically the record data type, and how to retrieve one or more records. You also looked at internal documentation for new objects and basic multi-language functionality.

Positioning – Where do you go from here?

Now that you have tables and user interfaces for our master data, you can begin to develop the forms and code necessary to carry out transactions.

Quick Interaction: Lessons Learned

Take a moment to write down three Key Points you have learned from this chapter:

1.

2.

3.

CHAPTER 3: MANAGING REGISTRATIONS

This chapter contains the following sections:

- Introduction
- Exporting Objects as Text Files
- Multi-Language Functionality in Text Messages
- Main/Sub forms
- Matrix Forms
- Types of Tables
- Additional Functions
- Test Your Skills
 - Diagnosis
 - Managing Seminar Registration
- Test Your Knowledge
- Conclusion

Microsoft Internal Use Only

Introduction

Positioning – What is the starting point?

You have finished creating our master files and now are ready to create the functionality to allow users to perform transactions with the master data.

Preconditions

To create the transactions in this chapter, the master files must be created.

Further preconditions are knowledge of the following areas:

- Internal documentation.
- Introduction to multi-language functionality.
- Working with event triggers, specifically table event triggers.
- Working with the record data type, including retrieving one or more records.

Business Goals

By the end of this chapter, you will have created the tables and forms necessary for performing transactions in the program.

Educational Goals

By completing this chapter, you will have learned or reacquainted yourself with the following:

- Exporting and importing objects as text files.
- Using multi-language functionality to create messages for the user.
- Using main and subforms.
- Creating matrix forms.
- Using virtual tables.
- Using temporary tables.

Exporting Objects as Text Files

Analyzing and sometimes even modifying objects can be much easier when the objects are in text file format. Any object in the Object Designer can be exported as text, and the text representation is complete. Once the object is in a text file, you can change the text file and import it back into Microsoft® Business Solutions–Navision® as an uncompiled object.

It is important to note that you cannot use the Import Worksheet when importing a text file. This means that you will not receive a warning about overwriting existing tables, you will not have the opportunity to skip the import of some objects and you will not have the opportunity to merge objects. You must therefore be careful when importing text files into Microsoft Navision.

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 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
```

The next section contains the object's date, time, and version properties, labeled with the title OBJECT-PROPERTIES.

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

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

The last section is the code section, labeled CODE. This section contains the global variables and functions for the object.

Once you have made changes to the text file and have saved it, you can import the text file back into Microsoft Navision.

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 will be imported. The objects will not be compiled, so you must compile them in Microsoft Navision before using them.

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 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 will be 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

To follow Microsoft Navision standards, we will build our Seminar Registration window in this chapter's exercises similar to Microsoft Navision's existing Sales Invoice windows. Let's take a look at some Microsoft Navision standards and features. For further detail on these topics, please refer to the Application Designer's Guide.

A main/subform is a combination of a card form with a tab control and a subform that contains a table box. For an example, look at the Sales Invoice form (Form 43), which is used to create, view, or modify sales invoice documents. As you can see, 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.

If you take a look at 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.

If you open the Sales Invoice Subform, notice 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, notice the following 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 Navision 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)

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 Navision, 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.

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 and 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. Step-by-step instructions follow on how to put the tables and the form controls together to create a simple multiplication table:

1. Create a new blank form and select as the source table, the table you want to use as the vertical table. In this example, use the Integer table.
2. Add a matrix box to the form from the toolbox.
3. In the Property window, set the Name property of the MatrixBox control to MultTable, 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, we have been using regular database tables to implement our seminar module. In the upcoming exercises, we will make use of some other kinds of tables.

Virtual Tables

A virtual table contains information provided by the system. In C/SIDE you have access to a number of virtual tables. They work in much the same way as normal database tables, but you cannot change the information in them. That is, you can only read the information. Another difference is that 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 just like an ordinary table, you can use the same methods to access information in virtual tables. For example, you can use filters to get subsets or ranges of integers or dates.

The most popularly used virtual tables in C/SIDE include:

- Date
- Integer

However, there are other virtual tables that are usable within Microsoft Navision:

- 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, you cannot view them directly. To view a virtual table, you can create a tabular form based on it. They are the tables with the highest numbers in the table list (2000000000+). The Application Designer's Guide has further information on a number of the virtual tables.

For our purposes, we will be using the Date virtual table. 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 can be regarded as a temporary variable that is used to hold a table. You can do almost anything with a temporary table that you can do with a normal database table; the only differences between a normal database table and a temporary table are that:

- Temporary tables are not stored in the database, but only held in memory on your workstation until the table is closed.
- The write transaction principle that applies to normal database tables does not apply to temporary tables. If you are not familiar with the transaction principle, refer to the Application Designer's Guide.

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. When you need to perform many operations on data in a specific table in the database, you can load the information into a temporary table while you modify it. Because all operations are local, this speeds up the process.

Creating a temporary table is much like 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 you want 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 just like regular tables, but they differ in that they are created automatically by the system. You can read, write, modify, and delete the information in these tables. 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

You were introduced to many of the following functions in Development I. As you will use them in the upcoming exercises, see how many you can remember. Look up the others in the C/SIDE Reference Guide.

- 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 just 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 you set the SaveRecord parameter to FALSE, this function will 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 if you want to use the same form for viewing and editing, but only allow some users to edit the records.

Control Functions

Microsoft Navision also has a number of functions available for form controls. To access the functions of a control, you must name the control first. You should then use the following syntax:

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

The following list gives some information about the most useful control functions, but it is, of course, 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 you make a text box 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, you have to call this function 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.

Test Your Skills – Managing Registrations – Diagnosis

Description

Now that our master data is complete, we must 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, we 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.

Use Cases

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

- Managing Seminar Registration



Implementation of Use Case 1 – Managing Seminar Registration

Managing Seminar Registration – Analysis

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

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

A 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 rental. In the registration information, we must also be able to specify how the seminar should be invoiced (for example, whether to include expenses or catering).

We should be able to set up additional comments for each seminar that would allow us to specify necessary equipment or any other particular requirements for the specific course.

Using this information, we can further define how the management of seminar registrations will be reflected in the module we are now creating.

Purpose

The registration of a participant involves ensuring that a place is reserved for the participant, that the participant's company can be invoiced, that appropriate facilities are reserved, and that seminar and participant information can be tracked.

Preconditions

Master data for the seminars, instructors, rooms, and participants must exist. Master data about the customers to which the participants are associated must exist.

Postconditions

Seminar instances are created with the ability to register participants and assign rooms, charges and instructors.

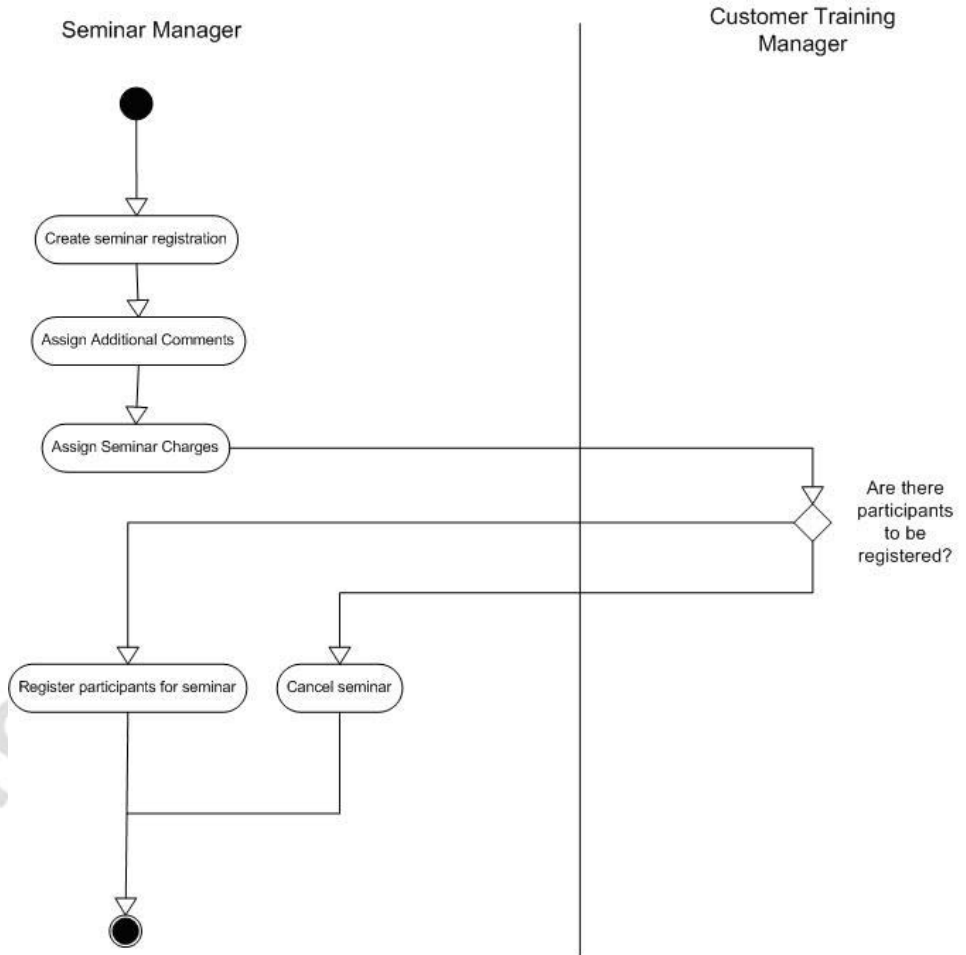
Main Scenario

When seminars are going to be held in the training academy, the seminar managers will create an "instance" of the seminar and specify the following details:

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

The seminar managers can then register participants for that instance of the seminar.

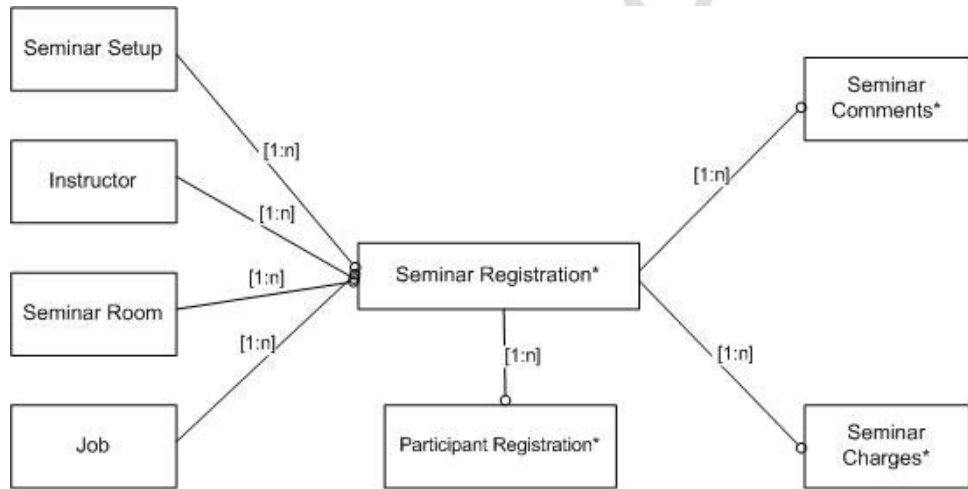
Activity Diagram



Managing Seminar Registration – Design

As can be seen from the analysis of the seminar registration management process, there are two main processes: the creation of seminar instances and the registration of participants. Within the first main process, create seminar instances, there are three subprocesses: define seminar registration details, assign additional comments and assign seminar charges.

The following diagram 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 subprocess tables are to the right.

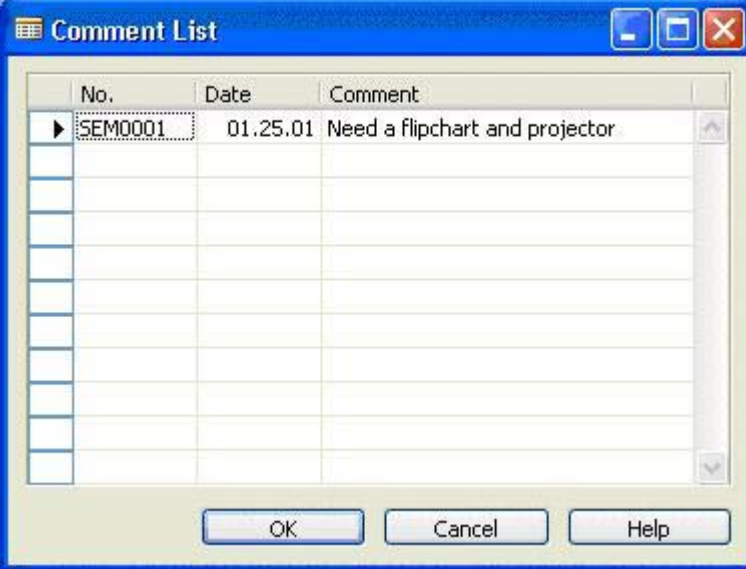


* new table

GUI Design

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

Seminar Comment List (Form 123456707): This form displays the comments for a seminar.

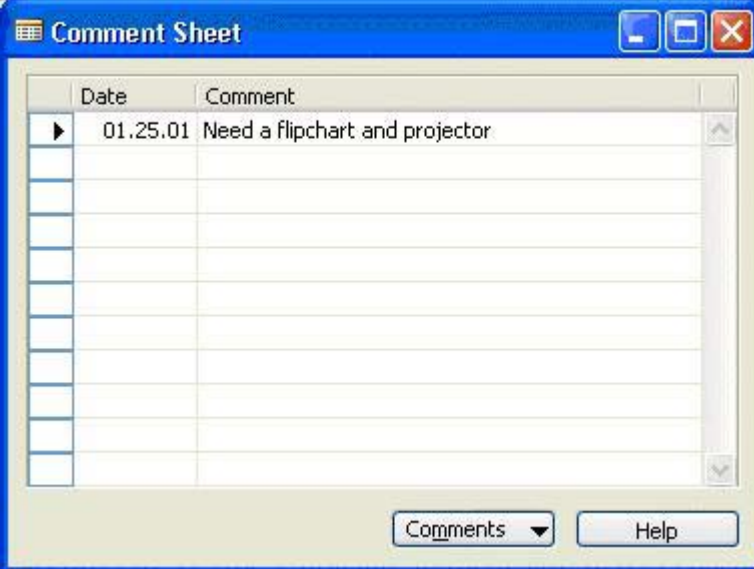


The screenshot shows a window titled "Comment List" with a table containing the following data:

No.	Date	Comment
SEM0001	01.25.01	Need a flipchart and projector

Buttons at the bottom: OK, Cancel, Help.

Seminar Comment Sheet (Form 123456706): This form enables the entry of comments for a seminar.

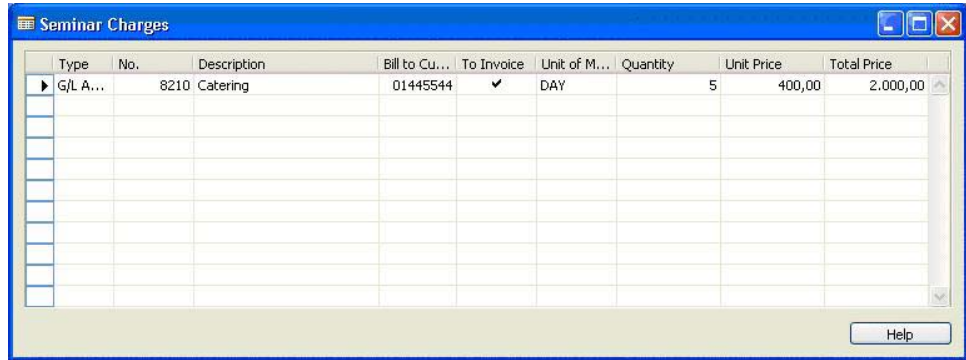


The screenshot shows a window titled "Comment Sheet" with a table containing the following data:

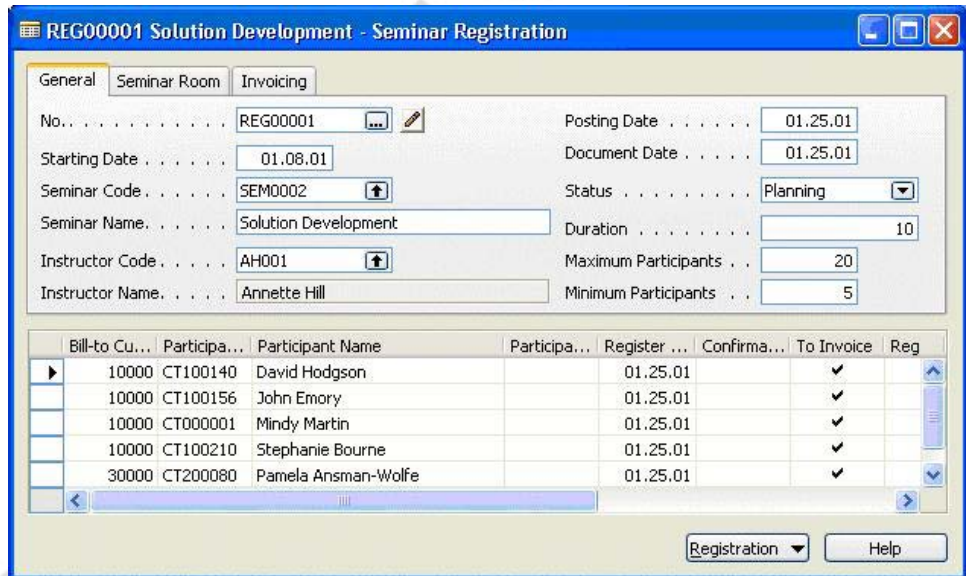
Date	Comment
01.25.01	Need a flipchart and projector

Buttons at the bottom: Comments (dropdown), Help.

Seminar Charges (Form 123456724): This form enables the entry of charges for a seminar.



Seminar Registration (Main Form 123456710, Subform 123456711): Since the seminar registration and participant registration are so closely linked, it would be best to handle them in one window with header and lines as shown in the following screenshot. This form actually consists of two forms, a header form and a subform that contains the lines.



GENERAL TAB

Bill-to Cu...	Participa...	Participant Name	Participa...	Register ...	Confirma...	To Invoice	Reg
10000	CT100140	David Hodgson		01.25.01		✓	
10000	CT100156	John Emory		01.25.01		✓	
10000	CT000001	Mindy Martin		01.25.01		✓	
10000	CT100210	Stephanie Bourne		01.25.01		✓	
30000	CT200080	Pamela Ansmann-Wolfe		01.25.01		✓	
30000	CT200079	Tina Gorenc		01.25.01		✓	

SEMINAR ROOM TAB

Bill-to Cu...	Participa...	Participant Name	Participa...	Register ...	Confirma...	To Invoice	Reg
10000	CT100140	David Hodgson		01.25.01		✓	
10000	CT100156	John Emory		01.25.01		✓	
10000	CT000001	Mindy Martin		01.25.01		✓	
10000	CT100210	Stephanie Bourne		01.25.01		✓	
30000	CT200080	Pamela Ansmann-Wolfe		01.25.01		✓	
30000	CT200079	Tina Gorenc		01.25.01		✓	

INVOICING TAB

Seminar Registration List (Form 123456713): This form displays the seminar registrations.

No.	Starting ...	Seminar ...	Seminar Name	Status	Duration	Maxim...	Room Code
REG00001	01.08.01	SEM0002	Solution Development	Planning	10	20	ROOM01
REG00002	01.10.01	SEM0001	Programming	Planning	5	12	ROOM02
REG00003	01.13.01	SEM0002	Solution Development	Planning	10	20	ROOM03

Functional Design

The user will eventually be able to post and print reports from the Seminar Registration, but these functions are designed and implemented later in the course.

Table Design

The following tables are required for registrations:

Table 123456710 Seminar Registration Header holds the information for one instance of a Seminar, which we refer 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.

Managing Seminar Registration – Development

Your first tasks are to create the tables and forms that we have designed. After creating the objects, we create code to allow them to function better.

Exercise 5 – Creating the Tables and Forms for Seminar Registration

Our first task will be to create the Seminar Comment Line table. 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 Navision.

1. Create table 123456704 Seminar Comment Line by exporting the standard table Sales Comment Line, modifying it and importing it back into Microsoft Navision.
 - To do this, first export the table 44 Sales Comment Line 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."
 - Wherever you find the word "Sales", replace it with the word "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.
 - Finally, save the text file, import it into Navision, 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	

- The key for the table is **Document Type, No., Line No.** This will also be imported with the table.
2. Create form 123456707 Seminar Comment List 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 form 123456706 Seminar Comment Sheet 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.

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. Must not be editable.
7	Status	Option		Options: Planning, Registration, Closed, Canceled.
8	Duration	Decimal		Decimal Places 0:1
9	Maximum Participants	Integer		
10	Minimum Participants	Integer		
11	Room Code	Code	20	Relation to Seminar Room table.
12	Room Name	Text	30	

No.	Field Name	Type	Length	Comment
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.

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.**

6. Create form 123456724 Seminar Charges 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 table 123456711 Seminar Registration Line 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.**

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 will automatically create a key for a newly inserted record.
9. Create form 123456710 Seminar Registration main form as shown in the GUI design. Include the three **General**, **Seminar Room** and **Invoicing** tabs, 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 will expand and contract 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 form 123456706 Seminar Comment Sheet. The link should run whenever there is an update.
	<Separator>	
	Charges	Opens form 123456724 Seminar Charges for the corresponding Seminar Registration No. The link should run whenever there is an update.

10. Create form 123456713 Seminar Registration List 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.

Exercise 6 – 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 should not be allowed to 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:
 - a. Tests that the corresponding Resource is not blocked.
 - b. Tests that the **Gen. Prod. Posting Group** field is filled on the Resource record.
 - c. Sets the **Description** field of the Seminar Charge table to the Name from the Resource.
 - d. 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:
 - a. Gets the corresponding G/L Account record and runs the CheckGLAcc function.
 - b. Tests that Direct Posting is TRUE for the G/L Account.
 - c. Sets the **Description** field of the Charge table to the Name of the G/L Account.
 - d. 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:
 - a. Gets the corresponding Resource record.
 - b. If the Unit of Measure Code is blank, the program sets it to the Base Unit of Measure Code from the Resource record.
 - c. 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.

- d. Calculates the Unit Price according to the Unit Price from the Resource record.
- If the **Type** is G/L Account:
 - a. Sets the **Qty. per Unit of Measure** to 1.
 - b. If the current field is the **Unit of Measure Code** field, the program validates the Quantity. Use CurrFieldNo to check the current field. We want to 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);
```

Exercise 7 – Adding Code to the Seminar Registration Header Table and Form

In the Seminar Registration Header table, enter code to perform the following tasks. Create your 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 you created for the Seminar table is very 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 the Status is Canceled and shows an error if the 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 the 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.

Exercise 8 – 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 gets 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 you have at least one Seminar Room, Instructor and Seminar 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 you set up in the last chapter should be automatically filled into the **No.** field. Note that when you start a new record 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 don't matter for now, but the **Job No.** becomes important in the next chapter when we post registrations.
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 didn't function as expected, make the necessary changes.

Microsoft Internal Use Only

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?

Conclusion

Chapter Summary

In this chapter, you created the tables and forms necessary to register participants in seminars. In doing so, you also created code to improve usability and data validation.

Positioning – Where do you go from here?

The next step is to take the transaction information and create a posting routine with which you can certify participants and create ledger entries for completed courses. You will also make it possible to post invoices to customers.

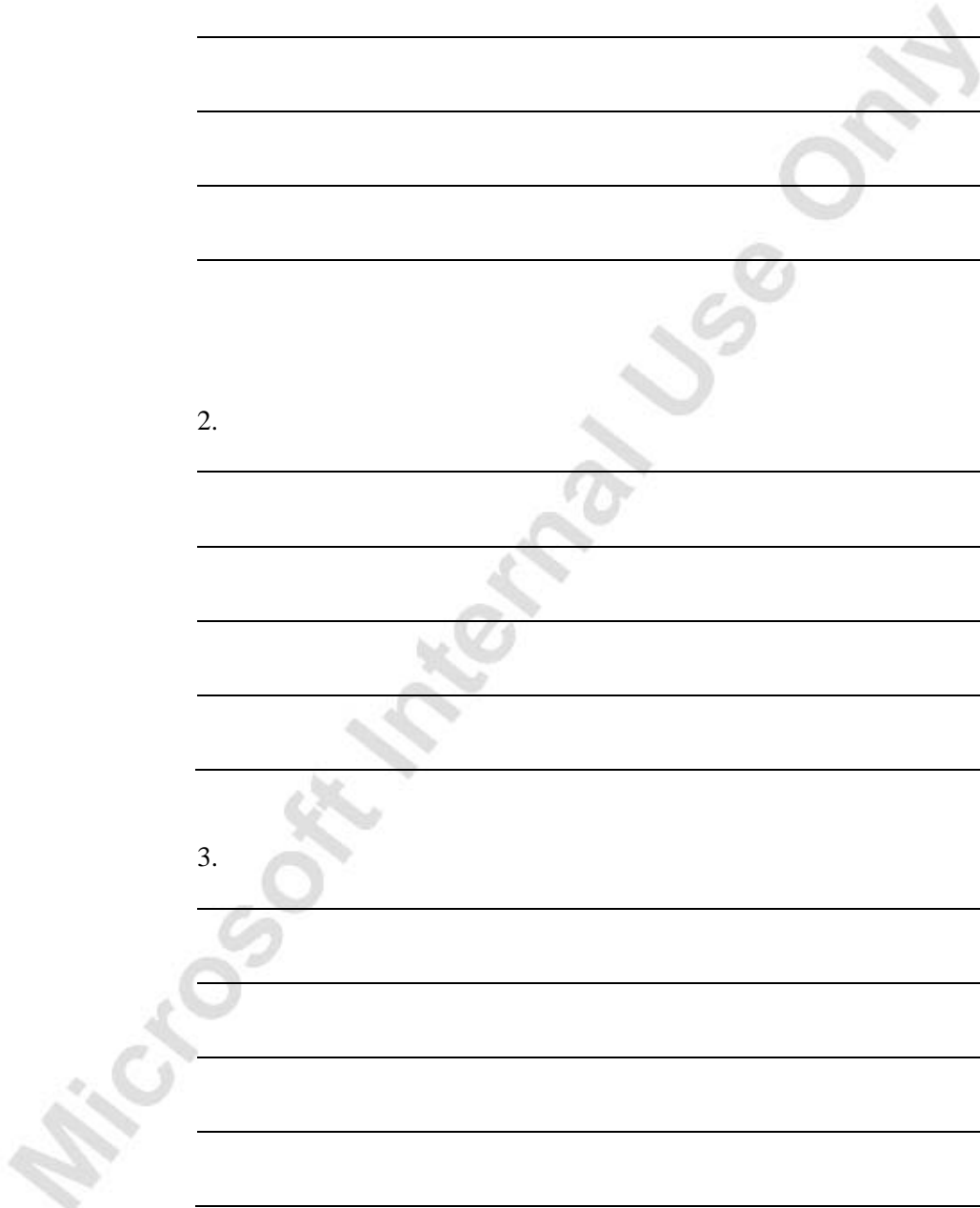
Quick Interaction: Lessons Learned

Take a moment to write down three Key Points you have learned from this chapter:

- 1.

- 2.

- 3.



CHAPTER 4: MANAGING POSTING

This chapter contains the following sections:

- Introduction
- Posting
 - Journal Tables
 - Ledger Tables
 - Posting Routines
 - Check Line Codeunit
 - Post Line Codeunit
 - Post Batch Codeunit
 - Posting Routine – Code Walkthrough
- Documentation in Existing Objects
 - Code Comments
- Document Posting Routines
 - Document Posting Routine – Code Walkthrough
- Performance Issues
 - Table Locking
 - Keys and Queries
 - Reducing Impact on the Server
 - Reducing Impact on Network Traffic
- Debugging Tools
 - Debugger
 - Code Coverage
 - Handling Runtime Errors
 - Client Monitor
- Test Your Skills
 - Diagnosis
 - Managing Seminar Registration Posting
- Test Your Knowledge
- Conclusion

Introduction

Positioning – What is our Starting Point?

Our seminar module now contains master files and a means of creating registrations. We can therefore begin to look at how we can use the registration information to create ledger entries for seminars through a posting routine.

Preconditions

The preconditions for this chapter are that the seminar, room, instructor and participant master files must exist. We must also have the seminar registration tables and forms.

Further preconditions are knowledge of the following areas:

- Writing internal documentation
- Enabling multilanguage functionality
- Exporting and importing objects
- Working with event triggers
- Working with complex data types and their member functions
- Using virtual tables
- Using temporary tables

Business Goals

In this chapter, our goal is to produce a posting routine that will post seminar registration information to ledger tables.

Educational Goals

Through completing this story, you will have learned or reacquainted yourself with the following:

- Creating journal posting routines
- Creating document posting routines
- Writing internal documentation for modifications to existing objects
- Debugging code
- Programming for low-impact on the application

Posting

Before we begin work on our seminar posting, let's examine some of the elements involved in posting.

Journal Tables

A Journal is a temporary work area for the user. Records can be inserted, modified and deleted as the user wishes. There are actually three tables that are used to make up the Journal. Journal Line is the main table. There are two supplemental tables called Journal Template and Journal Batch. These tables are used primarily as filters on the Journal Line table.

The primary key of the Journal Line table is a compound key (that is, it has more than one field). It consists of **Journal Template Name**, **Journal Batch Name** and **Line No.** The journal form that the user chooses sets the **Journal Template Name**, and this does not change unless the user goes into a different journal form. The **Journal Batch Name** may be changed at the top of the form (the options that are available depend upon the Template chosen), but only one can be viewed at a time. The **Line No.** keeps each record in the same template and batch unique. Line number is incremented automatically by the form (see the AutoSplitKey property). The user never sees most of the primary key fields on the form (other than the batch).

The journal form lets the user enter journal lines that will be added to the detail tables of the system (the ledgers), but nothing happens until the user decides to post. The user can leave the lines in the journal table as long as they wish without posting.

Ledger Tables

A ledger is a protected table that holds all the transactions for a particular functional area. These records are permanent and cannot be deleted or modified except through special objects. Also, records cannot be inserted directly into the table. Records cannot get into a ledger except through a Posting Routine, and Posting Routines only post journal lines.

The primary key is simply the **Entry No.** field. There are also many secondary keys, and most are compound. They are set up for reports, forms and FlowFields. The Ledger table holds the majority of detail information for the functional area.

A ledger table should never be modified directly, especially not by inserting or deleting records. There is a linkage between most of the ledger tables back to the General Ledger. Because of this linkage, any modifications made directly to the table can be disastrous. Usually, the only way to undo such changes is to restore the most recent backup of the database.

Posting Routines

A posting routine is a group of codeunits that is responsible for ensuring that all transactions that are put into the corresponding ledgers are correct per line and correct as a whole. The posting routine takes Journal Lines, checks them, converts them to ledger entries, inserts them into the Ledger table and ensures that all transactions made were consistent.

Although there are many different types of posting routines in Microsoft® Business Solutions–Navision®, there are some general rules that pertain to all of them. The primary codeunit that does the work of posting for a particular journal is simply called the posting routine. This codeunit is named after the Journal name with the addition of "-Post Line." Its main job is to transfer the information from the Journal record to the Ledger table, though it also does other things, such as calculations and data checking.

Posting Routine Companion Codeunits

The posting routine codeunit has two companion codeunits: one is named "-Check Line" and the other "-Post Batch." The Check Line routine is called by the Post Line routine to check each Journal line before sending it to the server for processing. Thus, all of its testing routines either do not touch the server at all or, at most, only once in each posting process.

The Post Batch routine repeatedly calls the Check Line routine to test all lines. It then repeatedly calls the Post Line routine to post all lines. The Post Batch routine is the only one that actually reads or updates the Journal table; the others simply use the Journal record passed into them. In this way, a programmer can call the Post Line routine directly (from another posting routine) without having to update the Journal table. The Post Batch routine is used only when the user selects Post within the Journal form.

Standardized Object Names

The last digits of the Object Numbers of these posting routines are standardized. The Check Line always ends with 1, the Post Line with 2 and the Post Batch with 3. As an example, Gen. Jnl.-Check Line is Codeunit 11, Gen. Jnl.-Post Line is Codeunit 12 and Gen. Jnl.-Post Batch is Codeunit 13.

Note that none of these routines have any interface that requires user input. This is so that they can be called from other applications without having to worry about messages popping up (except error messages). The Post Batch routine has a dialog that displays the progress of the posting and lets the user cancel. The rest of the user interface that has to do with posting is handled by another set of routines:

- Post routine (which just asks whether you want to post and then calls Post Batch) has an Object ID that ends with 1.

- Post and Print routine (which asks whether you want to post, calls Post Batch and then calls the Register Report) ends with 2.
- Batch Post (which asks whether you want to post the selected batches and then repeatedly calls Post Batch for each one) is called from the Journal Batches form and ends with 3.
- Batch Post and Print (which confirms that the user wants to post and then calls Post Batch for each batch and then calls the Register Report) ends with 4.

As an example, Gen. Jnl.-Post is Codeunit 231, Gen. Jnl.-Post+Print is Codeunit 232, Gen. Jnl.-B.Post is Codeunit 233 and Gen. Jnl.-B. Post+Print is Codeunit 234.

Check Line Codeunit

The name of this codeunit explains its function. It is designed to check the Journal Line that is passed to it. It does so without reading from the server, except perhaps the first time it is called.

Before checking any of the fields, this codeunit usually makes sure the journal line is not empty. It does so by calling the EmptyLine function in the Journal table. If the line is empty, the codeunit skips it by calling the EXIT function. There is no error, and the posting process will continue.

The last thing that the codeunit verifies is the validity of the dimensions that are passed into the function in a temporary Dimensions table. This is done by some simple calls to the DimensionManagement codeunit. We will discuss dimensions later in this course. If the codeunit does not stop the process with an error, then the journal line is accepted.

Post Line Codeunit

This codeunit is responsible for actually writing the journal line to the ledger. It only posts one Journal Line at a time. It does not look at previous or upcoming records.

You may notice that the code in the OnRun trigger of this codeunit is inserting records. The OnRun trigger of the Post Line codeunit is normally never called, but in prior versions of the product it was. For backward compatibility, the OnRun trigger loads the temporary Dimensions table with the two global dimensions and then calls the RunWithCheck function. This is the function that is normally called by other functions or triggers. It in turn calls the workhorse of the Post Line routine, the Code function.

Like Check Line, this codeunit skips any empty lines by exiting. This ensures that empty lines are not inserted into the ledger. The first thing the codeunit does, if the line is not empty, is to call Check Line to verify that all the needed journal fields are correct.

Next, the codeunit checks the important table relations. This requires reading the database (using GET), which is why it is done here rather than in Check Line.

Before writing to the ledger, Post Line writes to the register. The first time the program runs through the Post Line codeunit, it inserts a new record in the Register table. In every subsequent run through Post Line, the program modifies the record by incrementing the **To Entry No.**

Then the codeunit takes the next entry number and the values from the journal line and puts them into a ledger record. Finally, it can insert the ledger record.

Near the bottom of the Code function is the call to the DimensionManagement codeunit that copies the dimensions from the temporary Journal Line Dimension table to the real Ledger Entry Dimension table.

The entry number for this new ledger record is passed into the function to keep the dimension records associated with this ledger entry. The very last thing that the codeunit does is to increment the variable that holds the next entry number by one. Thus, when the codeunit is called again, the next entry number is ready.

When the Post Line codeunit is done, one journal line has been processed, but more than one ledger record may have been inserted into more than one ledger.

Post Batch Codeunit

This codeunit is responsible for posting the Template and Batch that were passed to it. Only one record variable for the journal is actually passed in, but the codeunit starts by filtering down to the template and batch of the record that was passed in. Then it finds out how many records are in the record set that the record variable represents. If the answer is none, the codeunit exits without an error, and it is up to the calling routine to let the user know that there was nothing to post.

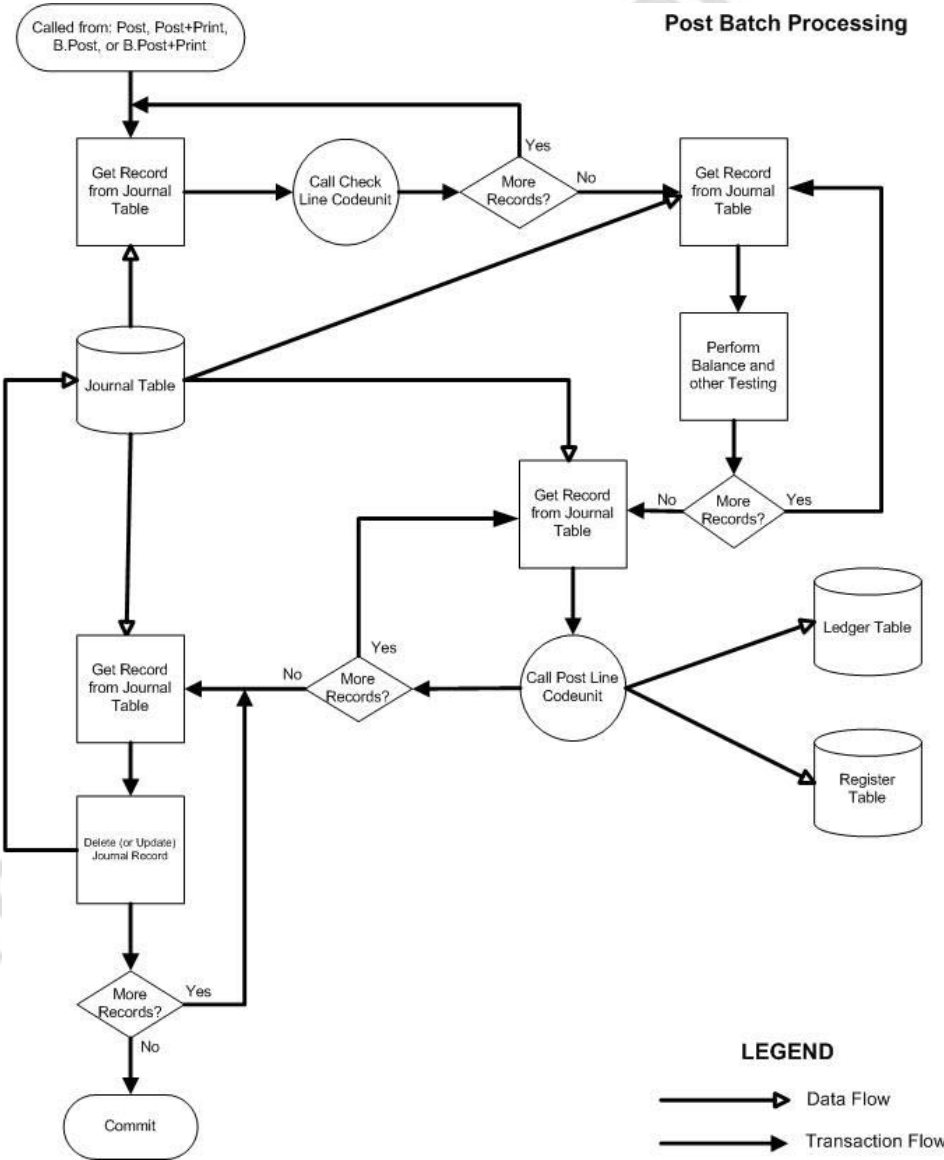
The Post Batch codeunit can then begin checking each journal line in the record set by calling the Check Line codeunit for each line. Once all the lines are checked, they can be posted by calling the Post Line codeunit for each line. By then, the codeunit has looped through all the records twice, once for Check Line and a second time for Post Line. To call the appropriate functions in Check Line and Post Line, Post Batch must fill in a temporary Journal Line Dimension table with all of the dimensions for the journal line. It can then pass this temporary record variable into the RunCheck function of Check Line (or the RunWithCheck function of Post Line) along with the journal line record variable.

Where Check Line checks consistency of a given line, the Post Batch Codeunit is responsible for checking the inter-relation of the lines. For example, if the codeunit is for general journal lines, it may also be responsible for making sure that the journal lines balance. This usually takes place after they are checked and before they are posted.

The codeunit may do other special things depending on the Journal Template. For recurring journals, the journal lines are updated with new dates based on the date formula. When a recurring journal line is posted, the codeunit must check the **Description** and **Document No.** fields and perhaps replace any replaceable parameters with the correct values (% 1 = day, %2 = week, %3 = month, and so on).

If the template is not recurring, the codeunit deletes all the journal lines.

The following diagram outlines the steps in the Posting Routine when Post Batch is called:



Posting Routine – Code Walkthrough

We are going to implement our own posting routine later in the seminar module using Microsoft Navision Standards. Let's quickly take a look at the Resource Journal Posting Routine (codeunits 211, 212, and 213), which is used when a user posts Resource Journals. Let's compare what we just learned about posting routines to what we see in these codeunits.

Check Line

Design codeunit 211-Res. Jnl.-Check Line. Notice that the OnRun trigger gets the GL Setup record and then sets a temporary record for the dimensions. Dimensions will be talked about later in this course.

In the RunCheck function trigger, the first thing you notice is that the codeunit checks to see if the line is empty. If the line is empty, the codeunit skips further checking and exits without error.

The codeunit also checks to see if the posting date is within the allowable posting date from the GL Setup table.

Finally, the Check Line codeunit calls functions from the DimensionManagement codeunit to verify the dimensions.

If this codeunit goes through without error, then the posting routine continues.

Post Line

Design codeunit 212-Res. Jnl.-Post Line. Notice that the OnRun trigger again gets the GL Setup record and also sets a temporary record for the dimensions. As mentioned previously in this chapter, the OnRun trigger contains code for backward compatibility. Most codeunits call the RunWithCheck function which then calls the Code function, where most of the posting is done in the posting routine.

Notice, like the Check Line codeunit, the Post Line codeunit skips any empty lines by exiting. This ensures that empty lines are not inserted into the ledger. Next, if the line is not empty, it calls Check Line to verify that all the needed journal fields are correct.

Next, the codeunit gets the next entry number from the Resource Ledger Entry table to be used with the resource register table. As mentioned above, before writing to the ledger, Post Line writes to the register. The codeunit first adds a new record into the Register table, and then every subsequent run through Post Line increments the **To Entry No.**

Then the codeunit takes the next entry number and the values from the journal line and puts them into a ledger record. Finally, it can insert the ledger record.

After the Ledger is inserted into the table, this codeunit calls the DimensionManagement codeunit to transfer the dimension lines from the temporary record to the Ledger entry table.

Post Batch

Design codeunit 213-Res. Jnl.-Post Batch. This codeunit is responsible for posting the Resource Template and Resource Batch that were passed to it.

The codeunit starts by filtering to the template and batch of the Resource Journal Line. If no records are found in this range, Post Batch exits and it is up to the calling routine (codeunit 271, 272, 273, or 274 in this case) to let the user know that there was nothing to post.

The Post Batch codeunit then loops and checks each journal line in the record set by calling the Check Line codeunit. Once all the lines are checked, they enter another loop which posts the records by calling the Post Line codeunit for each line. Post Batch must fill in a temporary Journal Line Dimension table with all of the dimensions for the journal line. It can then pass this temporary record variable into the RunCheck function of Check Line (or the RunWithCheck function of Post Line) along with the journal line record variable.

Unlike the General Journal, there are no interdependencies between Resource Journal lines, so no checks such as checking the balance need to be done here.

Lastly, this codeunit calls the UpdateAnalysisView codeunit to update all of the Analysis Views.

Document Posting Routines

A document in Microsoft Navision is an easy interface for a user to make many complicated transactions. In our case, the user uses the seminar registration document to tie a seminar registration to a number of customers and participants as well as to comments and seminar charges. The document posting routine for seminar registration translates this document information into journal entries which can then be posted to ledger entries.

To understand more clearly how a document posting routine works and what its components are, look at the example of the sales order posting routine.

Consider the example of a sales order with three sales lines. On Line 1, we are selling a G/L Account (perhaps this line adds some kind of surcharge or freight). On Line 2, we are selling an item (for example, a computer). On Line 3, we are selling a resource (here we are talking about the time that one of our employees has spent custom-building the computer).

When the user posts the document, the program generates an entry that debits the Accounts Receivable Account in the G/L, and each line could generate a separate G/L entry for the amount of that line. At the same time, entries are being made for the Item and Resource journals, as well as the General journal for the Customer.

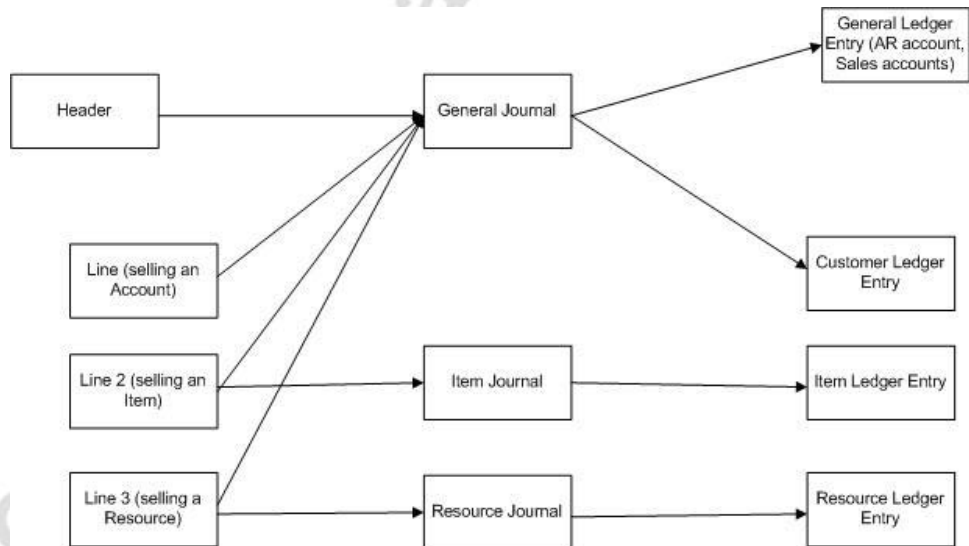
When these journal entries are posted, they are posted as if the user had entered them into the journals. The biggest difference is that the journal records are posted one at a time, which allows the Sales Post routine to bypass Post Batch and call Post Line directly.

In the example of a sales order posting, codeunit 12 Gen. Jnl.-Post Line would be called at least twice, codeunit 22 Item Jnl.-Post Line would be called once and codeunit 212 Res. Jnl.-Post Line would be called once.

A sales document is posted primarily by codeunit 80 Sales-Post. You can, however, post an entire batch of sales documents by calling report 297 Batch Post Sales Invoices. Note that this report is for invoices only. There is a separate report for each document type.

These reports call codeunit 80 repeatedly for each document. For this to work, codeunit 80 must not interact with the user. In fact, codeunit 80 is never called directly by a form. The form calls codeunit 81 Sales-Post (Yes/No) or 82 Sales-Post + Print or one of the reports mentioned previously. They in turn interact with the user (getting confirmation or other information) and then call codeunit 80 appropriately.

The following diagram illustrates the theory in document posting:



Document Posting Routine – Code Walkthrough

Let's focus on codeunit 80, and break it down into some of its major parts. Assume that the user is shipping and invoicing a Sales Order. Scroll through the codeunit and identify the sections that perform the following tasks:

- The codeunit starts by determining the document type and validates the information that appears on the sales header and lines. The codeunit determines what the posted document numbers are going to be and updates the header. This section ends with a COMMIT.
- The codeunit locks the appropriate tables.
- The codeunit inserts the Shipment Header and Invoice or Credit Memo Header.
- The codeunit processes the sales lines. This section starts by clearing the Posting Buffer (a temporary table based on the Invoice Post Buffer table) and ends with the UNTIL that goes with REPEAT, which loops through all the sales lines. Inside this section, each line is processed individually.
- Within the REPEAT loop, the codeunit checks each line with its matching Shipment line (if the line was previously shipped). If the line type is an Item or a Resource, it is posted through the correct journal. The line is then added to the posting buffer. It may be inserted or it may update a row already there. If the line is related to a job, it posts a journal line through the Job Journal. Then, if there is no shipment line, it inserts one. Finally, it copies the Sales Line to the Invoice Line or Credit Memo Line (the posted tables).
- The codeunit can now post all entries in the Posting Buffer to the General Ledger. These are the Credits that are created from the sale of the lines. Then the codeunit can post the Debit to the General Ledger. The customer entry is made to the Sales Receivables Account. It checks whether there is a balancing account for the header. This corresponds to an automatic payment for the invoice.
- Lastly, the codeunit updates and/or deletes the Sales Header and Lines and commits all changes.

Documentation in Existing Objects

When you make changes in an existing object, you must create a note in the Documentation trigger for each modification you make. Usually, this amounts to one note for each feature. The note heading should contain a reference number, the date when the modification was completed, the name of the developer responsible for the modification, the project name and a short description of the change. Here is an example:

```
Microsoft Business Solutions
-----
Project: Navision Solution Development II Training
jtd: John T. Doe

No.   Date       Sign   Description
-----
001   01.21.2004  jtd    Created
002   05.05.2004  jtd    Transfer new field
                                           Seminar Registration No.
```

Notice that documentation in an existing object looks much like the documentation in a new object, except that you create a new reference number for each modification.

Code Comments

Along with the general comments that you provide in the Documentation trigger, it is important to provide comments in the code at the lines where you have made a change. Only do this when you modify an existing object, not when you create a new object.

The key is to mark the changed code with the same reference number as used in the Documentation trigger of the object. For example, if you have modified a single line of code, you should mark it like this:

```
State := "Employee."Default Work State"; // jtd002

If you have added or modified an entire block of code, you
should mark your change as follows:

// - jtd:002 ---
    JobLedgEntry."Seminar Registration No." := "Seminar
Registration No.";
// + jtd:002 +++
```

If, as part of your modification, you want to remove a block of code, you should mark it as follows:

```
{ - jtd:002 --- Start Deletion
State := "Employee."Default Work State"; Locality :=
"Employee."Default Work Locality"; "Work Type Code" :=
Employee."Default Work Type Code";
+ jtd:002 +++ End Deletion}
```

Note that this keeps the old code in place, just commented out. **Never** delete Navision base code – comment it out instead.

Performance Issues

When writing large posting routines like this one, it is important to program with the idea of maximizing performance. There are a number of steps you can take while programming a solution in Microsoft Navision that helps to improve performance.

Table Locking

Normally, you do not need to be concerned with transactions and table locking when developing applications in C/SIDE. There are, however, some situations where you have to lock a table explicitly. For example, let us suppose that in the beginning of a function, you inspect data in a table and then use this data to perform various checks and calculations. Finally, you want to write back a record based upon the result of this processing. You will want the values that you retrieved at the beginning to be consistent with the data in the table now. In short, you cannot allow other users to update the table while your function is busy doing its calculations.

The solution is to lock the table yourself at the beginning of your function by using the LOCKTABLE function. This function locks the table until the write transaction is committed or aborted. This means that other users can read from the table, but they cannot write to it. Calling the COMMIT function unlocks the table.

The RECORDLEVELLOCKING property is used to detect whether record level locking is being used. This property is only used with the SQL Server Option for Microsoft Navision, which is currently the only server that supports record level locking.

Keys and Queries

When you write a query that searches through a subset of the records in a table, always carefully define the keys both in the table and in the query so that Microsoft Navision can quickly identify this subset. For example, the entries for a specific customer will normally be a small subset of a table containing entries for all the customers. If Microsoft Navision can locate and read the subset efficiently, the time it will take to complete the query will only depend on the size of the subset.

To maximize performance, you must define the keys in the table so that they facilitate the queries that you have to run. These keys must then be specified correctly in the queries.

Reducing Impact on the Server

Keeping the processing of transactions on the client as much as possible minimizes the load on the server and improves performance, particularly when there are a large number of users. There are several ways to achieve this.

Try to use the COMMIT function as little as possible. This function is handled automatically by the database for almost all circumstances, so you should almost never use it.

Limit the use of the LOCKTABLE function to where it is necessary. Remember that an insert, modify, rename, or delete locks the table automatically, so you do not need to lock a table for most modifications.

Try to structure your code so that you do not have to look at the return values of the INSERT, MODIFY or DELETE functions. When you use the return value, the server has to be notified right away so that a response can be obtained. If you do not look at them, the server is not notified until something else requires it.

Use the CALCSUMS and CALCFIELDS functions whenever possible to avoid going through records to add up values.

Reducing Impact on Network Traffic

Consider setting keys and filters and then using MODIFYALL or DELETEALL functions, which send only one command to the server, rather than getting and deleting or modifying each of many records, which will send more information back and forth through the network.

Since CALCSUMS and CALCFIELDS can both take multiple parameters, use these functions to perform calculations on several fields with one function call.

Debugging Tools

There are three categories of errors you can meet when you develop applications in C/AL code: syntax errors, runtime errors and program logic errors.

Syntax errors are errors where the program encounters an unknown identifier, or where the program expects keywords that are not there, such as an IF or THEN that is missing. Syntax errors are detected by the compiler.

Runtime errors are not detected by the compiler and only occur when the code is executed. A good example of this is when division by zero occurs in the code.

A program logic error occurs when the code compiles and runs but does not function as intended.

Microsoft Navision has tools that help you track down the source of errors and enable you to see what code has been run.

Debugger

Microsoft Navision provides an integrated debugging tool to help you check and correct code that does not run smoothly.

The debugger enables you to set breakpoints, which are marks that are set in the code to tell the debugger to stop execution so that you can examine what is happening. The Breakpoint on Triggers setting (SHIFT+CTRL+F12) is enabled by default when you activate the debugger for the first time. Otherwise the code would be executed normally because there are no breakpoints. The debugger therefore suspends execution of the code when it reaches the first trigger. At this point you can set other breakpoints and then disable the Breakpoint on Triggers option if so desired. If you do not disable the Breakpoint on Triggers setting, the debugger will suspend execution of the code at every trigger it reaches.

You can also set or remove breakpoints from the C/AL Editor by either pressing F9 or clicking TOOLS→DEBUGGER→TOGGLE BREAKPOINTS. Information about breakpoints is stored in the Breakpoints virtual table when you close the C/AL Editor.

You can activate the debugger from Microsoft Navision by clicking TOOLS→DEBUGGER→ACTIVE (SHIFT+CTRL+F11).

In the debugger interface, you find several windows and menus. From the Edit menu, you can access the Breakpoints dialog box (SHIFT+F9). It displays a list of the breakpoints that you have set for the object you are debugging. You can enable, disable, and remove breakpoints in the list. The **View** menu contains commands that display the various debugger windows, such as the Variables window and the Call Stack window. The Debug menu contains commands that start and control the debugging process, for example, Go, Step Into, Step Over and Show Next Statement.

The Go command executes code from the current statement until a breakpoint or the end of the code is reached, or until the application pauses for user input.

The Step Into command executes statements one at a time, and you can decide how to continue after each statement. The execution steps into any function that is called, which means that the debugger single-steps through the statements in the function.

The Step Over command executes statements one at a time, like Step Into, but if you use this command when you reach a function call, the function is executed without the debugger stepping through the function instructions.

The Show Next Statement command shows the next statement in your code.

Code Coverage

When you activate the code coverage functionality, the program logs the code and objects that are run during the time the coverage is activated. This can be useful when you are customizing Microsoft Navision and want to test your work. It provides a quick overview of the objects for which code has been executed and displays the code that has been logged.

To activate code coverage:

1. Click **TOOLS**→**DEBUGGER**→**CODE** coverage. The Code Coverage window opens.
2. Click **Start** to begin logging code.
3. When you have completed the transactions you want to monitor, return to the Code Coverage window, which now contains a list of any tables, forms, reports, dataports and codeunits that were used.
4. Click **Stop**.
5. Select an object for which you wish to view the code. Click **Code** to open the Code Overview window.

The Code Overview window displays code for the object that you selected in the Code Coverage window. Lines of code that were executed during the transaction(s) are shown in black. Lines of code that were not executed are shown in red.

Handling Runtime Errors

It is possible to avoid some runtime errors by writing code that handles possible errors. A typical example of this is the GET function. The return value of the GET function is a Boolean, so that if the program finds the record in the table, the function returns TRUE; if the program does not find the record, the function returns FALSE.

If the GET function is used as it is shown below, a runtime error occurs if the program cannot find the specified record:

```
Customer.GET("Customer Number");
```

If you create statements to handle a possible FALSE result of the GET with a message or error to the user, no runtime error occurs and the user is able to understand how to correct the problem. You would handle the runtime error by writing code in the ELSE branch of the code below:

```
IF Customer.GET("Customer Number") THEN  
....  
ELSE  
....
```

Client Monitor

The Client Monitor, which is accessed by clicking **Tools**, Client Monitor, is actually a virtual table called Monitor, which you can use to get an overview of the time consumption of specific operations. This tool can be quite valuable when you are optimizing the performance of your solution.

In the Client Monitor window, you can use the **Options** tab to specify the kind of information gathered by the Client Monitor. The **Options** tab also contains advanced parameters that are only available with the SQL Server Option. Refer to the Application Designer's Guide for more information on these parameters.

Test Your Skills – Managing Posting – Diagnosis

Description

Now that we have transaction data, our next task is to use the completed transaction data to create ledger entries from which we can view history, create statistics and create invoices. We will, of course, be doing this with a posting routine. Our client's functional requirements describe their posting needs in this way:

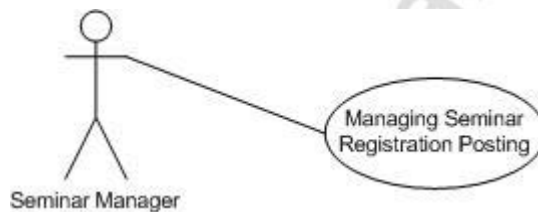
When a seminar is completed, the seminar should be posted as a job, with additional seminar-specific information.

Use Cases

We can describe the task of creating posting routines for the seminar module as the following use case:

- Managing Seminar Registration Posting

The following diagram illustrates the use case:



Implementation of Use Case 1 – Managing Seminar Registration Posting

Managing Seminar Registration Posting – Analysis

Our client's functional requirements describe the registration posting in the following way:

When a seminar is completed, the seminar should be posted as a job with additional seminar-specific information.

We therefore need to post seminars as jobs, but we must also have a separate ledger with seminar-specific information. We must also ensure that posting the registration enables us to invoice customers for participation in seminars.

Purpose

The purpose of posting seminar registration is to record that a seminar took place, to record the participants who took part and to enable us to invoice customers.

Preconditions

Information must exist for seminar registrations and customers.

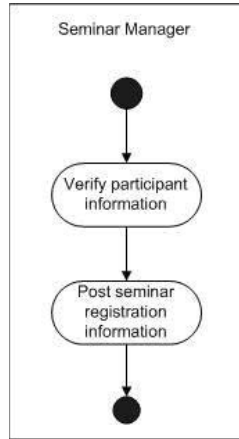
Postconditions

After a seminar registration posting, a posted registration document, ledger entries and registers will exist for the posted seminar registration.

Main Scenario

Upon completion of a seminar, seminar managers verify the participant list with the trainer and post the registration document.

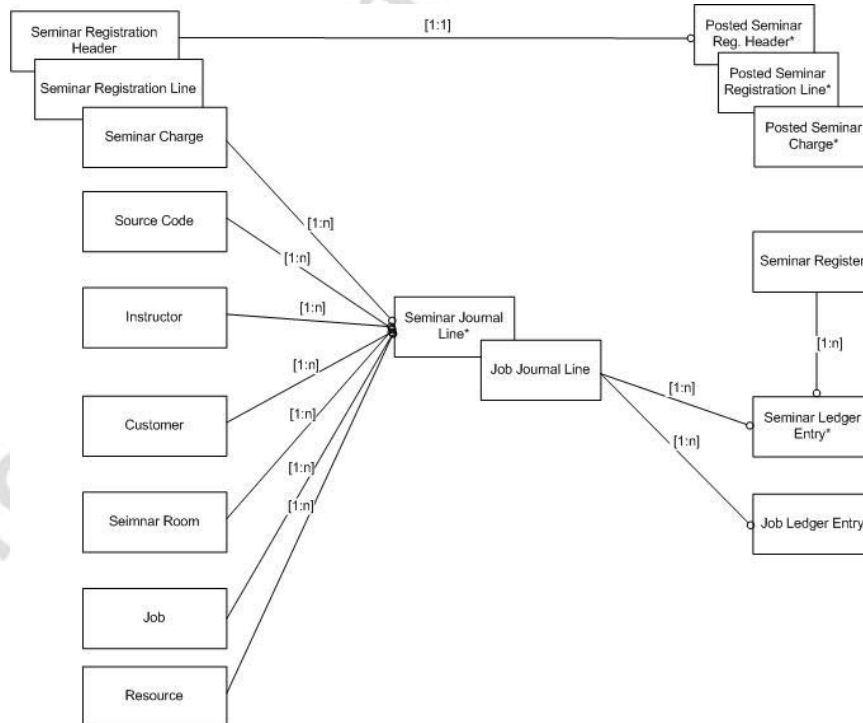
Activity Diagram



Managing Seminar Registration Posting – Design

The basic design for posting seminar registrations is similar to that of standard journal posting. Since we are also posting a document, we must create additional posted document tables and forms to contain historical information.

The following diagram illustrates how the tables interact.

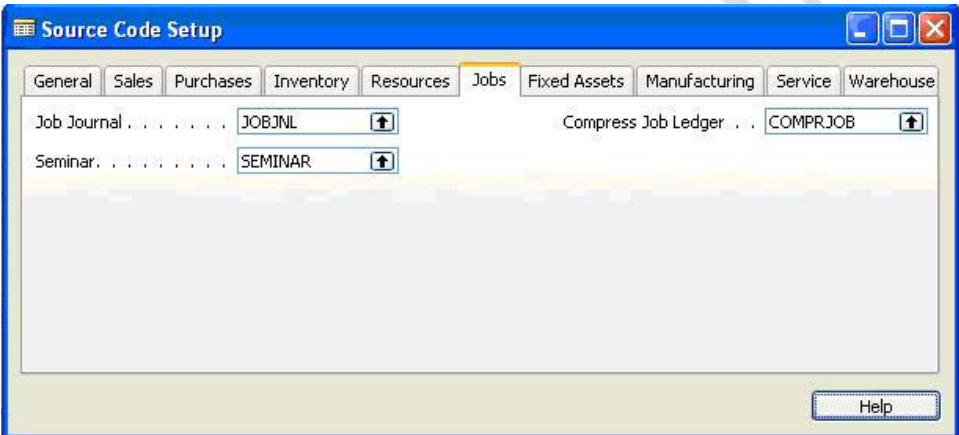


* new table

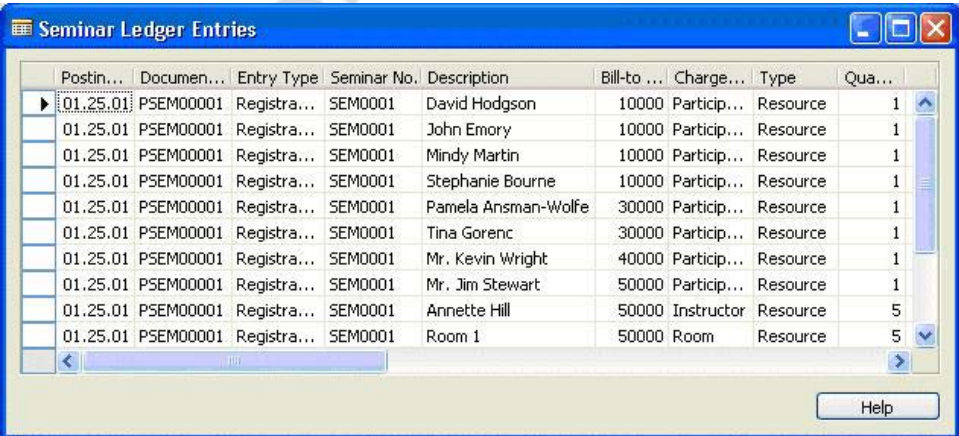
GUI Design

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

Source Code Setup (Form 279): Add one new field to the form, **Seminar**, as shown below.



Seminar Ledger Entries (Form 123456721): This form displays the seminar ledger entries.



Seminar Registers (Form 123456722): This form displays the registers created when seminar registrations are posted.

No.	Creation Date	User ID	Source Code	Journal Batch...	From Entry No.	To Entry No.
1	01.25.01		SEMINAR		1	10
2	01.25.01		SEMINAR		11	15

Buttons: Register, Help

Posted Seminar Charges (Form 123456739): This form shows the posted seminar charges from a posted seminar registration.

Type	No.	Description	Bill-to Cust...	To Invoice	Unit of Me...	Quantity	Unit P...	Total Price
G/L Account	8640	Miscellaneous		✓	DAY	5	450,00	2.250,00

Buttons: Help

Posted Seminar Registration (Form 123456734) and Posted Seminar Reg. Subform (Form 123456735): These forms show the posted seminar registration header and line information.

Bill-to Cus...	Participant ...	Participant Name	Participated	Register ...	Confirm...	To Invoice
10000	CT100140	David Hodgson	✓	01.25.01		✓
10000	CT100156	John Emory	✓	01.25.01		✓
10000	CT200136	Mindy Martin	✓	01.25.01		✓
10000	CT100210	Stephanie Bourne		01.25.01		✓
30000	CT200080	Pamela Anzman-Wolfe	✓	01.25.01		✓

GENERAL TAB

Bill-to Cus...	Participant ...	Participant Name	Participated	Register ...	Confirm...	To Invoice
10000	CT100140	David Hodgson	✓	01.25.01		✓
10000	CT100156	John Emory	✓	01.25.01		✓
10000	CT200136	Mindy Martin	✓	01.25.01		✓
10000	CT100210	Stephanie Bourne		01.25.01		✓
30000	CT200080	Pamela Anzman-Wolfe	✓	01.25.01		✓

SEMINAR ROOM TAB

This form shows the posted seminar registration line information.

The screenshot shows a window titled "PSEM00001 Programming - Posted Seminar Registration" with three tabs: "General", "Seminar Room", and "Invoicing". The "Invoicing" tab is active. At the top, there are two input fields: "Seminar Price" with the value "500,00" and "Job No." with the value "300010". Below these is a table with the following columns: "Bill-to Cus...", "Participant ...", "Participant Name", "Participated", "Register ...", "Confirm...", and "To Invoice".

Bill-to Cus...	Participant ...	Participant Name	Participated	Register ...	Confirm...	To Invoice
10000	CT100140	David Hodgson	✓	01.25.01		✓
10000	CT100156	John Emory	✓	01.25.01		✓
10000	CT200136	Mindy Martin	✓	01.25.01		✓
10000	CT100210	Stephanie Bourne	✓	01.25.01		✓
30000	CT200080	Pamela Anzman-Wolfe	✓	01.25.01		✓

At the bottom right of the window are two buttons: "Registration" (with a dropdown arrow) and "Help".

INVOICING TAB

Posted Seminar Reg. List (Form 123456736): This form displays a list of posted seminar registrations.

The screenshot shows a window titled "Posted Seminar Reg. List". It contains a table with the following columns: "No.", "Starting ...", "Seminar No.", "Seminar Name", "Duration", "Maximum...", and "Room Code".

No.	Starting ...	Seminar No.	Seminar Name	Duration	Maximum...	Room Code
PSEM00001	01.08.01	SEM0001	Programming	5	12	ROOM01
PSEM00002	04.01.01	SEM0001	Programming	5	12	ROOM01

At the bottom of the window are four buttons: "OK", "Cancel", "Registration" (with a dropdown arrow), and "Help".

Seminar Registration (Form 123456710): Add a **Posting** button to this form as shown below:

Bill-to Cu...	Participa...	Participant Name	Participa...	Register ...	Confirma...	To Invoice	Reg
10000	CT100140	David Hodgson		01.25.01		✓	
10000	CT100156	John Emory		01.25.01		✓	
10000	CT000001	Mindy Martin		01.25.01		✓	
10000	CT100210	Stephanie Bourne		01.25.01		✓	
30000	CT200080	Pamela Ansmann-Wolfe		01.25.01		✓	

Seminar Registration List (Form 123456713): Add a **Posting** button to this form as shown below:

No.	Starting ...	Seminar ...	Seminar Name	Status	Duration	Maximum Par...	Room Code
REG00001	01.08.01	SEM0002	Solution Development	Planning	10	20	ROOM01
REG00002	01.10.01	SEM0001	Programming	Planning	5	12	ROOM02
REG00003	01.13.01	SEM0002	Solution Development	Planning	10	20	ROOM03

Functional Design

As in all journal postings, you need journal posting codeunits to check Seminar Journal lines and to post them. However, unlike some posting codeunits (e.g. General Journal), you do not need a codeunit to post a batch of journal lines because you will not be posting batches.

Check Line: This codeunit helps ensure the data validity of a seminar journal line before it is sent to the posting routine. The codeunit should check that the journal line is not empty and that there are values for the Posting Date, Job No., Instructor Code and Seminar No. Depending on whether the line is posting an Instructor, a Room or a Participant, the codeunit should check that the key fields are not blank. The codeunit should check that the dates are valid.

Post Line: This codeunit performs the posting of the Seminar Journal Line. The codeunit should create a Seminar Ledger Entry per Seminar Journal Line and create a Seminar Register to track which entries were created during the posting.

You need to modify the Job Jnl.-Post codeunit to ensure that the Seminar Registration No. is recorded in the Job Ledger Entry.

You need two codeunits to enable the posting of the Seminar Registration document:

Seminar Post: This codeunit does the posting of an entire seminar registration, including the job posting, and seminar posting. The codeunit should transfer the comment records to new comment records corresponding to the posted document, and it should copy charges to new tables containing posted charges. The codeunit should create a new Posted Seminar Registration Header record as well as Posted Seminar Registration Lines. The codeunit should then run the job journal posting, and it should post seminar ledger entries for each participant, for the instructor and for the room. Finally, the codeunit should delete the records from the transaction tables, including the header, lines, comment lines, and charges.

Seminar Post (Y/N): This codeunit interacts with the user, asking whether they really want to post the registration. If the user answers Yes, the codeunit runs the Seminar Post codeunit.

Table Design

The following tables are required to implement our posting routines:

- Table 123456731 Seminar Journal Line
- Table 123456732 Seminar Ledger Entry
- Table 123456733 Seminar Register
- Table 123456721 Posted Seminar Charge
- Table 123456719 Posted Seminar Reg. Line
- Table 123456718 Posted Seminar Reg. Header

Add fields to the following Microsoft Navision tables:

- Table 210 Job Journal Line
- Table 169 Job Ledger Entry
- Table 242 Source Code Setup

Managing Seminar Registration Posting – Development

As with the previous use case, begin development of the posting by creating the tables and forms for the journal, ledger, register and posted documents.

Exercise 9 – Creating the Tables and Forms for Seminar Registration Posting

1. Source codes are used in posting tables to identify where entries originated. Add the additional field to table 242 Source Code Setup as follows:

No.	Field Name	Type	Length	Comment
123456 700	Seminar	Code	10	Relation to the Source Code table.

2. Modify form 279 Source Code Setup by adding the new **Seminar** field as shown in the GUI design.
3. Create table 123456731 Seminar Journal Line with the following fields:

No.	Field Name	Type	Length	Comment
1	Journal Template Name	Code	10	
2	Line No.	Integer		
3	Seminar No.	Code	20	Relation to Seminar table.
4	Posting Date	Date		
5	Document Date	Date		
6	Entry Type	Option		Options: Registration, Cancellation.
7	Document No.	Code	20	
8	Description	Text	50	
9	Bill-to Customer No.	Code	20	Relation to Customer table.
10	Charge Type	Option		Options: Instructor, Room, Participant, Charge.

No.	Field Name	Type	Length	Comment
11	Type	Option		Options: Resource,G/L Account
12	Quantity	Decimal		DecimalPlaces = 0:5
13	Unit Price	Decimal		AutoFormatType = 2
14	Total Price	Decimal		AutoFormatType = 1
15	Participant Contact No.	Code	20	Relation to Contact table.
16	Participant Name	Text	50	
17	Chargeable	Boolean		Initial value is Yes.
18	Room Code	Code	10	Relation to Seminar Room table.
19	Instructor Code	Code	10	Relation to Instructor table.
20	Starting Date	Date		
21	Seminar Registration No.	Code	20	
22	Job No.	Code	20	Relation to Job table.
23	Job Ledger Entry No.	Integer		Relation to Job Ledger Entry table.
24	Source Type	Option		Options: , Seminar.
25	Source No.	Code	20	If Source Type=Seminar, relation to Seminar table.
26	Journal Batch Name	Code	10	
27	Source Code	Code	10	Relation to Source Code table.
28	Reason Code	Code	10	Relation to Reason Code table.
29	Posting No. Series	Code	10	Relation to No. Series table.

The primary key for this table is **Journal Template Name, Journal Batch Name, Line No.**

4. Create a new function in table 123456731 Seminar Journal Line with a return type of Boolean, called EmptyLine. Enter one line of code in the function trigger so that the function returns TRUE if the **Seminar No.** is blank.

5. Create table 123456732 Seminar Ledger Entry with the following fields:

No.	Field Name	Type	Length	Comment
1	Entry No.	Integer		
2	Seminar No.	Code	20	Relation to Seminar table.
3	Posting Date	Date		
4	Document Date	Date		
5	Entry Type	Option		Options: Registration, Cancellation.
6	Document No.	Code	20	
7	Description	Text	50	
8	Bill-to Customer No.	Code	20	Relation to Customer table.
9	Charge Type	Option		Options: Instructor, Room, Participant, Charge.
10	Type	Option		Options: Resource,G/L Account.
11	Quantity	Decimal		DecimalPlaces = 0:5
12	Unit Price	Decimal		AutoFormatType = 2
13	Total Price	Decimal		AutoFormatType = 1
14	Participant Contact No.	Code	20	Relation to Contact table.
15	Participant Name	Text	50	
16	Chargeable	Boolean		Initial value is Yes.
17	Room Code	Code	10	Relation to Seminar Room table.
18	Instructor Code	Code	10	Relation to Instructor table.
19	Starting Date	Date		
20	Seminar Registration No.	Code	20	
21	Job No.	Code	20	Relation to Job table.

No.	Field Name	Type	Length	Comment
22	Job Ledger Entry No.	Integer		Relation to Job Ledger Entry table.
23	Remaining Amount	Decimal		FlowField; CalcFormula looks up the Remaining Amount in the corresponding Job Ledger Entry. AutoFormatType = 1. Must not be editable.
24	Source Type	Option		Options: , Seminar.
25	Source No.	Code	20	If Source Type=Seminar, relation to Seminar table.
26	Journal Batch Name	Code	10	
27	Source Code	Code	10	Relation to Source Code table.
28	Reason Code	Code	10	Relation to Reason Code table.
29	No. Series	Code	10	Relation to No. Series table.
30	User ID	Code	20	Relation to User table. Table relation should not be tested.

6. Enter code in the appropriate trigger so that when the user performs a lookup on the **User ID** field, the program runs the LookupUserID function from the LoginManagement codeunit.
7. Set the properties to specify form 123456721 as the lookup and drill down form for table 123456732.
8. The primary key for table 123456732 is **Entry No.**, with one secondary key of **Seminar No.**, **Posting Date** and a second secondary key of **Bill-to Customer No.**, **Seminar Registration No.**, **Charge Type**, **Participant Contact No.**
9. Create table 123456733 Seminar Register with the following fields:

No.	Field Name	Type	Length	Comment
1	No.	Integer		
2	From Entry No.	Integer		Relation to Seminar Ledger Entry table.
3	To Entry No.	Integer		Relation to Seminar Ledger Entry table.
4	Creation Date	Date		
5	Source Code	Code	10	Relation to Source Code table.

No.	Field Name	Type	Length	Comment
6	User ID	Code	20	Relation to User table. Table relation should not be tested.
7	Journal Batch Name	Code	10	

The primary key for table 123456733 is **No.**, with one secondary key of **Creation Date** and a second secondary key of **Source Code, Journal Batch Name, Creation Date**.

10. Enter code in the appropriate trigger so that when the user performs a lookup on the **User ID** field, the program runs the LookupUserID function from the LoginManagement codeunit.
11. Set the properties to specify form 123456722 as the lookup and drill down form for table 123456733.
12. Create form 123456721 Seminar Ledger Entries with the fields **Posting Date, Document No., Document Date** (not visible), **Entry Type, Seminar No., Description, Bill-to Customer No., Charge Type, Type, Quantity, Unit Price, Total Price, Remaining Amount, Chargeable, Participant Contact No., Participant Name, Instructor Code, Starting Date, Seminar Registration No., Job No., Job Ledger Entry No.** and **Entry No.** as shown in the GUI design. Set the property to make this form not editable.

With the new tables, you can now create the codeunits to post seminar-specific information from the seminar journal to the seminar ledger. These codeunits will be similar to other posting codeunits, except that we do not need a Post Batch codeunit for the document posting.

Exercise 10 – Creating the Codeunits and Form for Seminar Journal Posting

We will first create a codeunit to show the Seminar Ledger Entry form for a set of entries.

1. Create codeunit 123456734 Seminar Reg.-Show Ledger. Set the property to specify Seminar Register as the source table for this codeunit.
2. Enter code in the appropriate trigger so that when the program runs the codeunit, it runs the Seminar Ledger Entries form, showing only those entries between the From Entry No. and To Entry No. on the Seminar Register.

HINT: Look at Codeunit 275 Res.Reg.-Show Ledger.

The next step is to create the Check Line codeunit to help ensure data validity before posting.

3. Create codeunit 123456731 Seminar Jnl.-Check Line. Set the property to specify Seminar Journal Line as the source table for this codeunit.
4. In codeunit 123456731, create a function called RunCheck that takes a parameter that is passed by reference. This parameter is a record variable of the Seminar Journal Line table, called SemJnlLine.
5. Enter code in the appropriate trigger so that when the program runs codeunit 123456731, it runs the RunCheck function for the current record.
6. Enter code in the RunCheck function trigger so that the function performs the following tasks:

***HINT:** Look at the RunCheck function in Codeunit 211 Res.Jnl.-Check Line.*

- Tests whether the **Seminar Journal Line** is empty using the EmptyLine function. If the line is empty, the function exits.
- Tests that the **Posting Date**, **Job No.**, and **Seminar No.** fields are not empty.
- Depending on the value of the **Charge Type**, tests that the **Instructor Code**, **Room Code** and **Participant Contact No.** are not empty.
- If the line is Chargeable, tests that the **Bill-to Customer No.** is not blank.
- Shows an error if the **Posting Date** is a closing date.
- Tests that the **Posting Date** is between the **Allow Posting From** and **Allow Posting To** dates on the user's User Setup record; or if those fields are empty on the User Setup record, tests that the **Posting Date** is between the **Allow Posting From** and **Allow Posting To** dates on the G/L Setup record. The function shows an error if the **Posting Date** is not between the dates on either of these records.
- Shows an error if the **Document Date** is a closing date.

Now create the Post Line codeunit to post Seminar Journal lines.

7. Create codeunit 123456732 Seminar Jnl.-Post Line.
 - Set the property to specify Seminar Journal Line as the source table for this codeunit.
 - Define the following global variables for the codeunit:

Name	Data Type	Subtype	Length
SemJnlLine	Record	Seminar Journal Line	
SemLedgEntry	Record	Seminar Ledger Entry	
SemReg	Record	Seminar Register	
SemJnlCheckLine	Codeunit	Seminar Jnl.-Check Line	
NextEntryNo	Integer		

8. Create a function called GetSemReg that takes as a parameter a record variable for the Seminar Register table, called NewSemReg. This parameter is passed by reference.
9. Create a function called RunWithCheck that takes as a parameter a record variable for the Seminar Journal Line, called SemJnlLine2, which is passed by reference.
10. Create a function called Code.
11. Enter code in the appropriate trigger so that when the program runs codeunit 123456732 Seminar Jnl.-Post Line, it runs the RunWithCheck function for the current record.
12. Enter code in the GetSemReg function trigger so that the function sets the NewSemReg parameter to the SemReg record.
13. Enter code in the RunWithCheck function trigger so that the function copies the SemJnlLine from the SemJnlLine2 record, runs the Code trigger and copies the SemJnlLine2 record from the SemJnlLine.
14. Enter code in the Code function trigger so that the function performs the following tasks:

HINT: Look at the Code function in Codeunit 212 Res. Jnl.-Post Line.

- Tests whether the SemJnlLine is empty using the EmptyLine function. If it is empty, the function exits.
- Runs the RunCheck function of the SemJnlCheckLine codeunit.
- If the NextEntryNo is 0, the function locks the SemLedgEntry table and sets the NextEntryNo to one more than the **Entry No.** of the last record in the SemLedgEntry table.

- If the **Document Date** is empty, the function sets the **Document Date** to the **Posting Date**.
- Creates or updates the SemReg record, depending on whether or not the register record has been created for this posting. The function does this by checking whether the **No.** of the SemReg record is 0. If so, the function locks the SemReg table. If the function either cannot find the last record of the SemReg table, or if the **To Entry No.** is not 0, the function creates a new SemReg record, fills the fields as appropriate and inserts the new record. Regardless of whether the function creates a new record or not, the function sets the **To Entry No.** for the record to the NextEntryNo and modifies the record.
- Creates a new SemLedgEntry record, fills the fields as appropriate from the SemJnlLine record, sets the **Entry No.** to NextEntryNo, inserts the new record and increments NextEntryNo by 1.

Now create the Seminar Registers form from which the seminar registers can be viewed.

15. Create form 123456722 Seminar Registers with fields **No.**, **Creation Date**, **User ID**, **Source Code**, **Journal Batch Name**, **From Entry No.** and **To Entry No.** as shown in the GUI design.

- Set the property to specify that this form is not editable.
- Add a menu button and menu item to the form as follows:

Menu Button	Option	Comment
Register	Seminar Ledger	Runs the codeunit 123456734 Seminar Reg.-Show Ledger.

One of the requirements specified by the client was that the seminar registration should post as a job with additional seminar-specific information. We have just created the journals and codeunits to cover the seminar-specific information, so now we need to modify the tables, forms and codeunits for posting job journals to allow us to post seminar registrations as jobs.

Exercise 11 – Modifying the Tables, Forms and Codeunits for Job Posting

1. Add a new field to table 210 Job Journal Line as follows:

No.	Field Name	Type	Length	Comment
123456700	Seminar Registration No.	Code	20	

2. Add a new field to table 169 Job Ledger Entry as follows:

No.	Field Name	Type	Length	Comment
123456700	Seminar Registration No.	Code	20	

3. In codeunit 202 Job Jnl.-Post Line, enter code into the Code function trigger so that when the function is filling the JobLedgEntry fields, it fills the ledger's Seminar Registration No. field from the Job Journal Line.

Now create the tables and forms that will be used to store posted seminar registration information.

Exercise 12 – Creating the Tables and Forms for Posted Information

1. Create table 123456721 Posted Seminar Charge with the following fields:

No.	Field Name	Type	Length	Comment
1	Seminar Registration No.	Code	10	Relation to table 123456718. Must not be blank.
2	Line No.	Integer		
3	Job No.	Code	20	Relation to Job table.
4	Type	Option		Options: Resource, G/L Account.
5	No.	Code	20	If Type=Resource, relation to the Resource table. If Type=G/L Account, relation to the G/L Account table.
6	Description	Text	50	
7	Quantity	Decimal		Decimal Places=0:5
8	Unit Price	Decimal		AutoFormatType = 2 Minimum value is 0.
9	Total Price	Decimal		AutoFormatType=1 Must not be editable.
10	To Invoice	Boolean		Initial value is Yes.

No.	Field Name	Type	Length	Comment
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 the Resource Unit of Measure table, where the Resource No. = No.; otherwise, relation to the Unit of Measure table.
13	Gen. Prod. Posting Group	Code	10	Relation to Gen. Product Posting Group table.
14	VAT Prod. Posting Group	Code	10	Relation to VAT Product Posting Group table.
15	Qty. per Unit of Measure	Decimal		
16	Registered	Boolean		Must not be editable.

The primary key for table 123456721 Posted Seminar Charge is **Seminar Registration No., Line No.** with a secondary key of Job No.

2. Create form 123456739 Posted Seminar Charges with the fields **Type, No., Description, Bill-to Customer No., To Invoice, Unit of Measure Code, Quantity, Unit Price** and **Total Price** as shown in the GUI design. Set the property to specify that the form is not editable.
3. Create table **123456719 Posted Seminar Reg. Line** with the following fields:

No.	Field Name	Type	Length	Comment
1	Document No.	Code	20	Relation to table 123456718 Posted Seminar Reg. Header.
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 based on the Participant Contact No. Must not be editable.

No.	Field Name	Type	Length	Comment
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 for this table is **Document No., Line No.**

4. Create table 123456718 Posted Seminar Reg. Header with the following fields:

No.	Field Name	Type	Length	Comment
1	No.	Code	20	
2	Starting Date	Date		
3	Seminar No.	Code	10	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. Must not be editable.
7	Duration	Decimal		DecimalPlaces = 0:1
8	Maximum Participants	Integer		
9	Minimum Participants	Integer		

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

The primary key for table 123456718 Posted Seminar Reg. Header is **No.** with a secondary key of Room Code. The sum index field for the secondary key is Duration.

- Set the property to specify form 123456736 as the lookup form for the table.

***NOTE:** The field numbers in the Posted Seminar Reg. Header are set to match those of the Seminar Registration Header table even though they are not using all the same fields. This is so the TRANSFERFIELDS function, which relies on **Field No.**, can be used when copying.*

5. Create form 123456735 Posted Seminar Reg. Subform with the fields **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** as shown in the GUI design.
 - Set the width, height and positioning properties for the subform so that there is no empty space around the table box.
 - Set the properties for the **Line Discount %** and **Line Discount Amount** so that they are blank if the value is 0.
 - Set the property to specify that the form is not editable.
 - Set the property to specify that the program automatically creates a new key when a new line is inserted.

6. Create form 123456734 Posted Seminar Registration as shown in the GUI design.
 - Set the property to specify that the form is not editable.
 - Add a subform control to the form and set the control properties so that the subform control is the same size as the subform form. Set the subform properties to specify an ID for the form and the proper link to the table.

7. Add a menu button and menu items to form 123456734 as follows:

Menu Button	Option	Comment
Registration	List (F5)	Opens the lookup form.
	Comments	Opens the form 123456706 Seminar Comment Sheet showing the corresponding records. The link should run when the form is updated.
	<Separator>	
	Charges	Opens the form 123456739 Posted Seminar Charges showing the corresponding records. The link should run when the form is updated.

- Add a command button next to the **No.** field to provide access to the Comment Sheet for the corresponding record. Set the RunFormLink property for this button so that only the comments corresponding to the selected Posted Seminar Registration are displayed.

HINT: Copy the command button and picture box from the existing form 21 Customer Card and paste them into your new form.

- Enter code in the appropriate trigger so that after the form gets the record, the program releases the filter on the **No.** field of the Posted Seminar Header table.
8. Create form 123456736 Posted Seminar Reg. List with the fields **No.**, **Starting Date**, **Seminar No.**, **Seminar Name**, **Duration**, **Maximum Participants** and **Room Code** as shown in the GUI design.
- Set the property to specify that the form is not editable.
 - Add the menu button and menu item to the form as follows:

Menu Button	Option	Comment
Registration	Card (SHIFT + F5)	Opens the form 123456734 Posted Seminar Registration for the selected record.

You are now ready to create our document posting routine.

Exercise 13 – Creating the Codeunits for Document Posting

As shown in the functional design, you need two codeunits to handle the document posting. The first will be the codeunit that actually does the work of generating journal lines and running the posting routine, and the second will be the codeunit that interacts with the user.

The first codeunit will be the Seminar-Post codeunit.

1. Create codeunit 123456700 Seminar-Post.

2. Define the following global variables for the codeunit:

Name	Data Type	Subtype	Length
SemRegHeader	Record	Seminar Registration Header	
SemRegLine	Record	Seminar Registration Line	
PstdSemRegHeader	Record	Posted Seminar Reg. Header	
PstdSemRegLine	Record	Posted Seminar Reg. Line	
SemCommentLine	Record	Seminar Comment Line	
SemCommentLine2	Record	Seminar Comment Line	
SemCharge	Record	Seminar Charge	
PstdSemCharge	Record	Posted Seminar Charge	
SemRoom	Record	Seminar Room	
Instr	Record	Instructor	
Job	Record	Job	
Res	Record	Resource	
Cust	Record	Customer	
JobLedgEntry	Record	Job Ledger Entry	
SemLedgEntry	Record	Seminar Ledger Entry	
JobJnlLine	Record	Job Journal Line	
SemJnlLine	Record	Seminar Journal Line	
SourceCodeSetup	Record	Source Code Setup	
JobJnlPostLine	Codeunit	Job Jnl.-Post Line	
SemJnlPostLine	Codeunit	Seminar Jnl.-Post Line	
NoSeriesMgt	Codeunit	NoSeriesManagement	
ModifyHeader	Boolean		
Window	Dialog		
SrcCode	Code		10
LineCount	Integer		
JobLedgEntryNo	Integer		
SemLedgEntryNo	Integer		

3. Set the property to specify Seminar Registration Header as the source table for this codeunit.
4. Create a function called CopyCommentLines and set the property for this function to specify it as a local function. This function has the following four parameters:
 - An integer variable called FromDocumentType
 - An integer variable called ToDocumentType
 - A code variable with length 20 called FromNumber
 - A code variable with length 20 called ToNumber
5. Create a function called CopyCharges and set the property to specify it as a local function. This function has two parameters: a code variable with length 20 called FromNumber and a code variable with length 20 called ToNumber.
6. Create a function called PostJobJnlLine with a return type of Integer. Set the property to specify it as a local function. This function has one parameter of an option variable called ChargeType with the options Participant, Charge.
7. Create a function called PostSeminarJnlLine with a return type of integer. Set the property to specify it as a local function. This function has one parameter of an option variable called ChargeType with the options Instructor, Room, Participant, Charge.
8. Create a function called PostCharge and set the property to specify it as a local function.
9. Enter code in the CopyCommentLines function trigger so that this function finds records in the Seminar Comment Line table that correspond to the FromDocumentType and FromNumber values that were passed as parameters. For each record the function finds, the function creates a new Seminar Comment Line record that is a copy of the old record, except that the function sets the **Document Type** and **No.** to the ToDocumentType and ToNumber.
10. Enter code in the CopyCharges function trigger so that the function finds all Seminar Charge records that correspond to the FromNumber. For each record found, the function transfers the values to a new Posted Seminar Charge record, using the ToNumber as the **Seminar Registration No.**

11. Enter code in the PostJobJnlLine function trigger so that the function performs the following tasks:
 - Gets the Instructor, Resource, and Customer records that correspond to the Seminar Registration Header record.
 - Creates a new Job Journal Line record and fills the fields appropriately. The Gen. Bus. Posting Group comes from the Customer record. The **Entry Type** is Usage. The **Document No.** and the **Seminar Registration No.** are both the **No.** from the Posted Seminar Reg. Header. The **Source Code** is the value in the SrcCode variable. The **Source Currency Total Cost** is the Seminar Price from the Seminar Registration Line.
 - If the **ChargeType** is Participant, certain fields are filled as follows: the **Description** in the **Job Journal Line** is the participant's name, the **No.** is the instructor's Resource No., **Chargeable** is the **To Invoice** value from the registration line, the quantity fields are 1, the cost fields are 0 and the price fields are taken from the **Amount** field on the registration line.
 - If the **ChargeType** is Charge, certain fields are filled as follows: the **Description** in the Job Journal Line is the **Description** from the Seminar Charge record. If the **Type** from the Seminar Charge is Resource, the journal line's **Type** is Resource and the **Unit of Measure Code** and **Qty. per Unit of Measure** are the corresponding values from the Seminar Charge record. If the Type of the Seminar Charge is G/L Account, the journal line's **Type** is G/L Account, **Chargeable** is the **To Invoice** value on the Seminar Charge record, the **Quantity (Base)** is 1, the **Unit Cost** is 0, the **Total Cost** is 0, and the **No.**, **Quantity**, **Unit Price** and **Total Price** are the corresponding values from the Seminar Charge record.
 - Runs the Job Jnl.-Post Line codeunit with the newly created Job Journal Line.
 - Exits and returns the **Entry No.** of the last Job Ledger Entry record in the table.
12. Enter code in the PostSeminarJnlLine function trigger so that the function performs the following tasks:
 - Creates a new **Seminar Journal Line** and fills the fields as appropriate from the Seminar Registration Header, Posted Seminar Reg. Header and parameter values.
 - If the **ChargeType** is Instructor, certain fields are filled as follows: the **Description** is the instructor's Name, the **Type** is Resource, **Chargeable** is FALSE and the **Quantity** is the Duration from the registration header.

- If the **ChargeType** is Room, certain fields are filled in as follows: the **Description** is the room's Name, the **Type** is Resource, **Chargeable** is FALSE, and the **Quantity** is the Duration from the registration header.
 - If the **ChargeType** is Participant, certain fields are filled in as follows: the **Bill-to Customer No.**, **Participant Contact No.** and **Participant Name** values come from the corresponding fields on the registration line, the **Description** is the **Participant Name** on the registration line, the **Type** is Resource, **Chargeable** comes from the **To Invoice** field on the registration line and **Quantity** is 1. The **Unit Price** and **Total Price** come from the **Amount** field on the registration line.
 - If the **ChargeType** is Charge, certain fields are filled in as follows: the **Description**, **Bill-to Customer No.**, **Type**, **Quantity**, **Unit Price** and **Total Price** values all come from the corresponding fields on the Seminar Charge record. **Chargeable** comes from To Invoice on the Seminar Charge record.
 - Runs the Seminar Jnl.-Post Line codeunit with the newly created Seminar Journal Line.
 - Exits with the **Entry No.** of the last Seminar Ledger Entry record in the table.
13. Enter code in the PostCharge function trigger so that the function performs the following tasks:
- For each record in the Seminar Charge table that corresponds with the Seminar Registration Header, the function sets the JobLedgEntryNo to the result of the PostJobJnlLine function (run with a parameter of 1) and runs the PostSeminarJnlLine function (with a parameter of 3).
 - Sets the JobLedgEntryNo to 0.
14. Enter code in the appropriate trigger so that when the program runs this codeunit, the codeunit performs the following tasks:
- Clears all variables and sets the SemRegHeader variable to the current record.
 - Tests that the **Posting Date**, **Document Date**, **Starting Date**, **Seminar Code**, **Duration**, **Instructor Code**, **Room Code** and **Job No.** fields on the registration line are not empty. Tests that the Status is Closed.
 - Gets the Seminar Room and Instructor records that correspond to the registration header and tests for both of them that the **Resource No.** field is not empty.
 - Gets the Seminar Registration Line records that correspond to the registration header and shows an error message if no records are found.

- Opens a dialog window to keep the user informed of the progress.
- If the **Posting No.** is blank on the registration header, tests that the **Posting No. Series** is not blank and runs the GetNextNo function of the NoSeriesManagement codeunit.
- Modifies the header if necessary and performs a commit.
- If record level locking applies, locks the **Seminar Registration Line** and Seminar Ledger Entry tables and finds the last Seminar Ledger Entry record.
- Sets the SrcCode variable to the Seminar value in the Source Code Setup table.
- Creates a new Posted Seminar Reg. Header record and transfers fields from the Seminar Registration Header record to the new record.

HINT: Use the TRANSFERFIELDS function to do this.

- In the Posted Seminar Reg. Header record, sets the **No.** to the **Posting No.** from the registration header, sets the **Registration No. Series** to the **No. Series** of the registration header and sets the **Registration No.** to the **No.** of the registration header.
- Updates the dialog window.
- Sets the Source Code and **User ID** fields on the Posted Seminar Reg. Header record before inserting the record.
- Runs the CopyCommentLines and CopyCharges functions.
- Resets the filter on the Seminar Registration Line table to get the records corresponding to the Seminar Registration Header record.
- For each registration line found, the function updates the dialog window with an updated line count. The function then tests that the **Bill-to Customer No.** and **Participant Contact No.** are not blank. If **To Invoice** is FALSE, the function sets the **Seminar Price, Line Discount %, Line Discount Amount** and **Amount** to 0. The function then sets the JobLedgEntryNo variable to the result of the PostJobJnlLine function and the SemLedgEntryNo variable to the result of the PostSeminarJnlLine function. The function then creates a new Posted Seminar Reg. Line record and transfers the fields from the Seminar Registration Line record to the new posted record. The function sets the **Document No.** field of the new record to the **No.** of the Posted Seminar Reg. Header. Finally, the function inserts the new record and sets the JobLedgEntryNo and SemLedgEntryNo variables to 0.
- Runs the PostCharge, PostSeminarJnlLine (to post the instructor) and PostSeminarJnlLine (to post the room) functions.

- If record level locking does not apply, locks the Seminar Registration Line table.
- Deletes the Seminar Registration Header and all corresponding Seminar Registration Line records, Seminar Comment Line records and Seminar Charge records.
- Sets the current record to the Seminar Registration Header record.
- To enable the user to post the seminar registration, we need a codeunit that interacts with the user and runs the Seminar-Post codeunit based on the user's input.

15. Create codeunit 123456701 Seminar-Post (Yes/No).

16. Define the following global variables for the codeunit:

Name	Data Type	Subtype	Length
SemRegHeader	Record	Seminar Registration Header	
SeminarPost	Codeunit	Seminar-Post	

17. Set the property to specify Seminar Registration Header as the source table for this codeunit.
18. Create a function called Code and set the property to specify that it is a local function.
19. Enter code into the Code function trigger so that the function confirms that the user wants to post the registration. If the user answers No, the function exits. If the user answers Yes, the function runs the Seminar-Post codeunit with the Seminar Registration Header record. The function performs a commit at the end.
20. Enter code into the appropriate trigger so that when the program runs this codeunit, the codeunit copies the current record into the Seminar Registration Header record, runs the Code function and copies the Seminar Registration Header into the current record.

Finally, you need to modify two forms to enable the registration posting.

21. Add a **Posting** menu button and menu item to form 123456710 Seminar Registration as follows:

Menu Button	Option	Comment
Posting	Post (F11)	Runs codeunit 123456701 Seminar-Post (Yes/No). There should be ellipses on this button.

22. Add a **Posting** menu button and menu item to form 123456713 Seminar Registration List as follows:

Menu Button	Option	Comment
Posting	Post (F11)	Runs codeunit 123456701 Seminar-Post (Yes/No). There should be ellipses on this button.

Testing Managing Posting

It is assumed that some setup has been performed during previous test scripts and that there is some sample data of customers, contacts(participants), seminars, rooms and instructors when testing posting.

1. In order to test our posting routine, we first have to do some set up. In the Navigation Pane under FINANCIAL MANAGEMENT→SETUP→TRAIL CODES, open the Source Code Setup window. You should see the new **Seminar** field on the **Jobs** tab. Click the lookup to open the Source Codes window. Enter a new source code of SEMJNL with a description of Seminar Journal and click **OK**. Close the Source Code Setup window.
2. Under RESOURCE PLANNING→JOBS click **Jobs** to open the Job Card. Enter a new job to use with our testing. Most of the fields don't matter, but on the **General** tab you have to set the **Status** to Order to be able to post. On the **Posting** tab, specify a Job Posting Group. Close the Job Card.
3. Select form 123456710 Seminar Registration in the Object Designer and click **Run**. Enter a new Seminar Registration, filling out all fields and entering at least one line.

4. Select POSTING→POST to initiate the posting routine. You will probably encounter some errors. Some will be due to our validation code while others will likely be bugs in the code. Use the Debugger to help you find the sources of unexpected errors that occur when you run your code.
5. When you have eliminated all errors, verify that your posting worked as expected. Select form 123456734 Posted Seminar Registration in the Object Designer and click **Run**. You should be able to find the seminar registration you posted there. Check that:
 - All header fields were copied over
 - All lines and fields within them were copied over
 - Any related charges appear in the Posted Seminar Charges window
6. Create another Seminar Registration. Run form 123456713 Seminar Registration List and try posting from there.
7. Open the Job Card for the job you created and select Ledger Entries from the **Job** menu button. You should see records there with a Source Code of SEMJNL created from Seminar posting.
8. Run form 123456722 Seminar Registers from the Object Designer. You should see entries there created from our posting tests.
9. This is only an overview of the code we created for our posting routine. If you want to test more thoroughly, review the exercises and check on the validation and functionality we included.

Test Your Knowledge

Review Questions

1. What three tables make up a journal?
2. In which type of table is permanent transaction data stored? Can these tables be modified directly?
3. Which function would you use if you wanted to ensure that the data in a table would not be changed by another user until you had finished writing to the table? Is it always necessary to use this function?
4. What are the three standard posting routine codeunits? What does each of these codeunits do and how do they interrelate?
5. When is a document posting routine used? How does a document posting routine interact with other posting routines?
6. What is the standard Microsoft Navision shortcut for Posting?

Conclusion

Chapter Summary

In this chapter, you learned about posting in Microsoft Navision from journals and from transaction documents. In doing so, you learned about the different tables and codeunits that make up a standard posting routine. You learned about using the Microsoft Navision debugger and code coverage functionality. Finally, you learned what key aspects of programming to keep in mind to maximize performance.

Positioning – Where do you go from here?

You are now ready to integrate the different aspects of the solution that you have created and to integrate the solution into the standard Microsoft Navision interface.

Quick Interaction: Lessons Learned

Take a moment to write down three Key Points you have learned from this chapter:

1.

2.

3.

CHAPTER 5: MANAGING INTEGRATION

This chapter contains the following sections:

- Introduction
- Changing Tables That Contain Data
- Test Your Skills
 - Diagnosis
 - Managing Seminar Feature Integration
 - Managing Navigate Integration
- Test Your Knowledge
- Conclusion

Microsoft Internal Use Only

Introduction

Positioning – What is your starting point?

You now have a basic, functioning module, in which a user can input seminar master data, perform registrations and post completed seminar registrations. You are now ready to integrate these features both with one another and with the standard application.

Preconditions

The preconditions for this story are that the seminar, room, instructor, and participant master files must exist. Seminar registration tables and forms must exist. Posting routines for seminar registration posting must exist.

Further preconditions are knowledge of the following areas:

- Writing internal documentation
- Working with complex data types and their member functions
- Creating journal and document posting routines
- Debugging code
- Programming for low-impact on the application

Business Goals

In this story, your goal is to integrate the seminar management features with one another and with the standard Microsoft® Business Solutions–Navision® application.

Educational Goals

By completing this story, you should have learned or reacquainted yourself with the following:

- Using MenuSuite objects
- Implementing Microsoft Navigate functionality

Changing Tables that Contain Data

In creating the seminar module, you have created a number of tables and modified only a few. There are a few guidelines to keep in mind when changing tables that contain data.

C/SIDE is designed to ensure that you never lose data when you modify the design of a table that contains data. This means that while it is possible to change a table that contains data, there are some important general guidelines that specify which types of changes are allowed under certain conditions. These guidelines are as follows:

- You can always change the name of a field.
- You can change the data type for a field only if there is no data in the field for any of the records in the table. There is one exception to this rule: you can change the data type of a field from Code to Text even if the field contains data for some records.
- You can always add a field to a table.
- To delete a field, you must delete all data from the field in all records in the table. Furthermore you must remove all references to the field from other tables, forms and reports.
- You can always increase the length of a string field, meaning a field with a data type of Code or Text. Whether you can decrease the length of a string field depends on the contents of all the values in the column in the table. The minimum length of a string field is determined by the longest string in the column.
- Although the **Field No.** may be changed, you should do so only if absolutely required. Existing code has to be modified as code and table references use the **Field No.**

Test Your Skills – Managing Integration – Diagnosis

Description

In this chapter, you need to integrate the features created thus far with one another and with the standard application. This means that you will be adding navigation to some forms and creating a menu for the seminar module, as part of the main menu shown in the Navigation Pane.

We also want to integrate the standard Navigate functionality into the seminar module. The Navigate feature in Microsoft Navision allows you to see a summary of the number and type of entries posted for a specific document number or posting date. This feature can be quite useful in helping the user easily trace the ledger entries that result from certain transactions. To make the seminar module fit with the standard application and to improve the usability, we will integrate this feature into the module.

Use Cases

We can divide the integration tasks into two use cases:

- Managing Seminar Feature Integration
- Managing Navigate Integration

Because these are both one-time use cases rather than daily activities performed by a user, there are no use case or activity diagrams to illustrate the users' interaction with the system.

Implementation of Use Case 1 – Managing Seminar Feature Integration

Managing Seminar Feature Integration – Analysis

To integrate our seminar features with one another, we want the registrations and ledger entries that were developed later on to be available from the seminar forms that were developed earlier. We also want to create a MenuSuite object for the seminar module, which can be accessed from the standard Navigation Pane.

Purpose

The purpose of integration is to make all seminar features available from the appropriate places in the application. This makes the customized application as usable as possible.

Preconditions

The seminar master tables and forms and seminar registration tables and forms must exist so that they can be integrated.

Postconditions

The appropriate seminar registrations and ledger entries will be accessible from the Seminar Card and Seminar List forms. The seminar module will be accessible from the main menu.

Main Scenario

When the seminar managers look at a seminar definition in the Seminar Card or Seminar List form, they will be able to access the Seminar Registration for the seminar as well as the Seminar Ledger Entries that relate to the seminar. The seminar managers will typically access the features of the seminar module from the main menu.

Activity Diagram

There is no activity diagram for this use case.

Managing Seminar Feature Integration – Design

To integrate the seminar module as described in the Analysis section, we make modifications to some existing forms and tables and create a new Seminar Menu that we can add to the main Menu Suite.

GUI Design

Seminar Card (Form 123456700): Modify this form by adding a **Registration** menu button as shown below:

SEM0001 Programming - Seminar Card

General Invoicing

No. SEM0001

Name Programming

Search Name PROGRAMMING

Seminar Duration 5

Minimum Participants 6

Maximum Participants 12

Blocked

Last Date Modified 05.13.03

Seminar Registration Help

Seminar List (Form 123456701): Modify this form by adding a menu button as shown below:

Seminar List

No.	Name	Seminar ...	Seminar ...	Gen. Pro...	VAT Pro...	Job No.
SEM0001	Programming	5	500,00	SERVICES	VAT10	J00010
SEM0002	Solution Development	10	500,00	SERVICES	VAT10	J00010

Seminar Registration Help

Seminar Menu: This is a new Partner level menu accessible from the standard main menu suite. All seminar module features will be available from this menu.



Microsoft Navision Development II – C/SIDE Solution Development

The seminar tree for this menu is as follows:

Menu Type	Menu Name	Group	Comment
Group	Seminars		
Item	Contacts	Seminars	Opens form 5050 Contact Card.
Item	Instructors	Seminars	Opens form 123456705 Instructors.
Item	Seminars	Seminars	Opens form 123456700 Seminar Card.
Group	Periodic Activities	Seminars	
Item	Periodic Activities	Periodic Activities	Runs report 123456700 (to be created later in the course) that creates Seminar Invoices.
Group	Order Processing		
Item	Registrations	Order Processing	Opens form 123456710 Seminar Registration.
Item	Sales Invoices	Order Processing	Opens form 43 Sales Invoice.
Group	Reports	Order Processing	You will add reports under this group in a later chapter.
Group	Documents	Order Processing	You will add documents under this group in a later chapter.
Group	History	Order Processing	
Item	Posted Registrations	History	Opens form 123456734 Posted Seminar Registration.
Item	Posted Sales Invoices	History	Opens form 132 Posted Sales Invoice.
Item	Registers	History	Opens form 123456722 Seminar Registers.
Item	Navigate	History	Opens form 344 Navigate.
Group	Setup		
Item	Seminar Setup	Setup	Opens form 123456702 Seminar Setup.
Item	Seminar Rooms	Setup	Opens form 123456703 Seminar Room Card.

Functional Design

No functions are necessary for this use case.

Table Design

You will not create any new tables in this use case.

Managing Seminar Feature Integration – Development

Begin by making modifications to the Seminar Card and Seminar List forms. You can then create the new Seminar Menu.

Exercise 14 – Integrating Seminar Features

1. Add the new button and the new menu items to form 123456700 Seminar Card as follows:

Menu Button	Options	Comment
Seminar	Entries	Opens submenu with the following option: Ledger Entries (CTRL + F5): Opens form 123456721 Seminar Ledger Entries filtered to entries with a corresponding Seminar No. The link should be run whenever the form is updated. The lines in the form should be sorted by Seminar No. and Posting Date.
Registration	Registrations	Opens form 123456710 Seminar Registration for registrations with a Seminar Code that corresponds to the No. field. The link should be run whenever the form is updated.

2. Add the same menu button and menu items to form 123456701 Seminar List that you added to the Seminar Card above.
3. Create a new MenuSuite using the "Partner" level. See the Navigation Pane Designer open on the left.
4. Right-click on a menu on the Navigation Pane Designer and choose Create Menu. Set the caption to Seminars and the bitmap to 12.
5. Create the tree in the Seminar Menu as shown in the GUI Design.

Implementation of Use Case 2 – Managing Navigate Integration

Managing Navigate Integration – Analysis

To make the seminar module fit with the standard application and to improve the usability, integrate the Navigate feature into the module.

Purpose

The purpose of integrating the Navigate feature is to improve the traceability of transactions to their ledger entries.

Preconditions

The tables, forms, and codeunits that enable and are filled by the seminar posting routines must exist.

Postconditions

The user will be able to access the Navigate function from ledger entry forms and posted seminar documents as well as from the standard application.

Main Scenario

When the seminar managers want to look at a complete summary of the ledger entries that were created for a posted seminar registration or ledger entry, they will use Navigate.

Activity Diagram

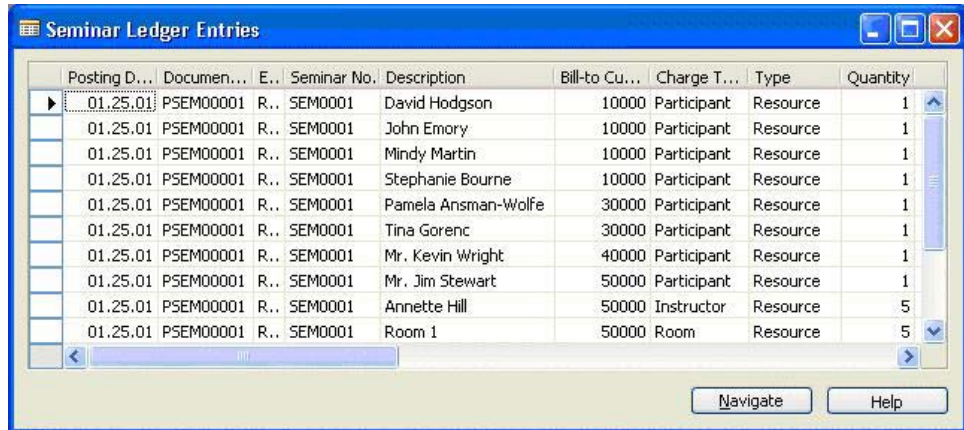
There is no activity diagram for this use case.

Managing Navigate Integration – Design

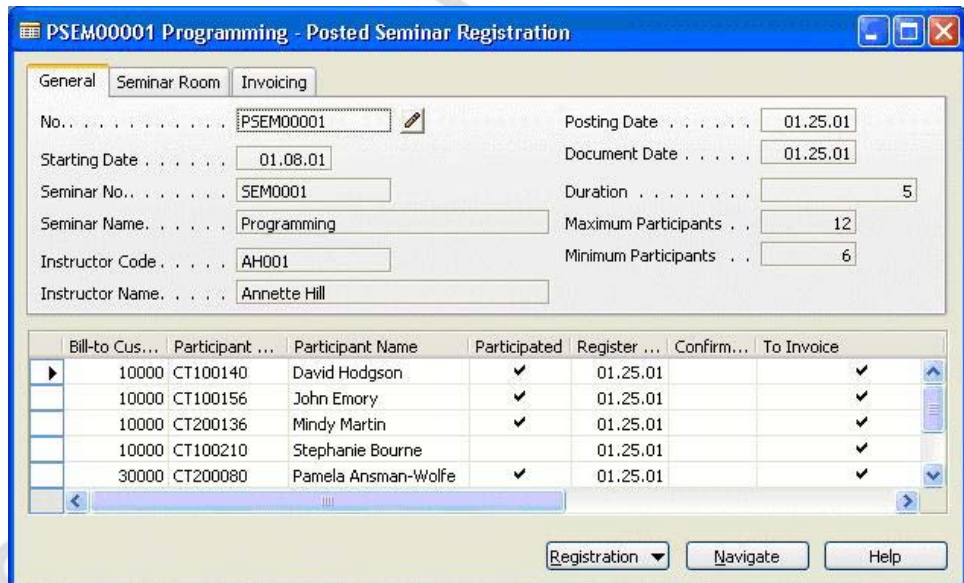
We want the Navigate feature to be available from the Seminar Ledger Entries form and the Posted Seminar Registration form.

GUI Design

Seminar Ledger Entries (Form 123456721): Modify this form by adding a **Navigate** command button:



Posted Seminar Registration (Form 123456734): Modify this form by adding a **Navigate** command button:



Functional Design

No functions are necessary for this use case.

Table Design

Do not create any new tables in this use case, but you will need to create new keys for two tables. The new keys will be specified in the development section.

Managing Navigate Integration – Development

The development steps to integrate the Navigate feature consist of adding keys to the ledger entry tables of the Seminar module, adding code to the Navigate form and adding command buttons to forms as shown in the GUI design.

Exercise 15 – Modifying Objects to Integrate Navigate

1. Add the secondary key **Document No.**, **Posting Date** to table 123456732 Seminar Ledger Entry.
2. In form 344 Navigate, add the following global variables:

Name	Data Type	Subtype	Length
SemRegHeader	Record	Seminar Registration Header	
PstdSemRegHeader	Record	Posted Seminar Reg. Header	
SemLedgEntry	Record	Seminar Ledger Entry	

3. Enter code in the FindRecords function trigger of the Navigate form so that the function performs the following tasks:
 - If the READPERMISSION property is TRUE on the Seminar Registration Header record, the program resets this record variable and filters the table to the records where the **No.** matches the DocNoFilter variable and the **Posting Date** matches the PostingDateFilter variable. Next, the program runs the function InsertIntoDocEntry.
 - The program then performs the same steps as above for the Posted Seminar Reg. Header table and the Seminar Ledger Entry table.
4. Enter code in the ShowRecords function trigger of the Navigate form so that the function performs the following tasks:
 - When the **Table ID** is that of the Seminar Registration Header, the function runs the lookup form for the Seminar Registration Header table.
 - When the **Table ID** is that of the Posted Seminar Reg. Header, the function runs the lookup form for the Posted Seminar Reg. Header table.

- When the **Table ID** is that of the Seminar Ledger Entry, the function runs the lookup form for the Seminar Ledger Entry table.
- 5. Add the **Navigate** command button to form 123456721 Seminar Ledger Entries.
- 6. Enter code in the appropriate trigger so that when the user clicks the **Navigate** button, the program runs the SetDoc function of the Navigate form and then runs the Navigate form.
- 7. Add the Navigate command button to form 123456734 Posted Seminar Registration.
- 8. Enter code in the appropriate trigger so that when the user clicks the **Navigate** button, the program runs the SetDoc function of the Navigate form and then runs the Navigate form.

Testing Managing Integration

Use the following script to test your development of the menu and navigate functionality.

1. Click the **Seminars** menu item in the Navigation Pane.
2. In the menu tree, click each of the items and check that the appropriate form or report opens. Remember that there are some items yet to be developed.
3. Use the Posted Registrations menu item to open the Posted Seminar Registrations form. Click the **Navigate** button and see that the Navigate form opens with the same **Document Number**.
4. Select Seminar Ledger Entry in the subform and click **Show**. The Seminar Ledger Entries form should open with the entries related to Posted Seminar Registration.

Test Your Knowledge

Review Questions

1. What is the standard Microsoft Navision shortcut for viewing Ledger Entries?
2. As a developer, at what level will you normally create MenuSuite objects?
3. From which types of forms is the Navigate feature typically available?

Conclusion

Chapter Summary

In this chapter, you integrated our previously created seminar module features with one another by adding navigation to forms. You added to the navigation pane so that the seminar module functionality is accessible in a standard way. Finally, we integrated the Navigate feature into the seminar module.

Positioning – Where do you go from here?

You are now ready to create reports for the seminar module.

Quick Interaction: Lessons Learned

Take a moment to write down three Key Points you have learned from this chapter:

1.

2.

3.

Microsoft Internal Use Only

Microsoft Internal Use Only

CHAPTER 6: MANAGING REPORTING

This chapter contains the following sections:

- Introduction
- Reporting
 - Report Triggers
 - Report Functions
 - Processing-only Reports
- Test Your Skills
 - Diagnosis
 - Managing Participant List Reporting
 - Managing Certification Confirmation
 - Managing Invoice Posting
- Test Your Knowledge
- Conclusion

Microsoft Internal Use Only

Introduction

Positioning – What is your starting point?

The seminar module includes so far:

- Master tables and forms.
- A means to create seminar registrations.
- Routines necessary to post the registrations.

These features are integrated into the standard application so that they can be accessed from the Main Menu. You are now ready to create reports for the module.

Preconditions

To create the reports for the seminar module, you must have master tables and forms, tables and forms for seminar registration, and the seminar registration posting routines.

Further preconditions are knowledge of the following areas:

- Writing internal documentation
- Enabling multilanguage functionality
- Exporting and importing objects
- Working with event triggers
- Working with complex data types and their member functions
- Using virtual tables
- Using temporary tables
- Creating journal and document posting routines
- Debugging code
- Programming for low-impact on the application

Business Goals

In this story, your goal is to create reports to improve the usability of the seminar module and to enable the posting of invoices.

Educational Goals

By completing this chapter, you should have learned or reacquainted yourself with the following:

- Using report event triggers
- Using special report functions
- Creating processing-only reports

Reporting

Report Triggers

Because there are so many events that happen when running a report, there are a number of event triggers for reports. The report itself, its data items, sections, request form and controls all have triggers. In this section, we do not describe all the triggers, but we will take a look at the order in which some of the more frequently used triggers fire when a report is run.

1. When the user initiates the running of the report, the `OnInitReport` trigger is called. This trigger can perform processing that is necessary before any part of the report is run. It can also stop the report.
2. If the `OnInitReport` does not end the processing of the report, the request form for the report is run, if it is defined. Here, the user can choose to cancel the report.
3. If the user chooses to continue, the `OnPreReport` trigger is called. At this point, no data has yet been processed.
4. When the `OnPreReport` trigger has been executed, the first data item is processed (provided that the processing of the report was not ended in the `OnPreReport` trigger).
5. Before the first record is retrieved, the `OnPreDataItem` trigger is called, and after the last record has been processed, the `OnPostDataItem` trigger is called.
6. Between these two triggers, the records of the data item are processed. Processing a record means executing the record triggers and outputting sections. `C/SIDE` also determines whether the current record should cause outputting of a special section: header, footer, group header or group footer.
7. If there is an indented data item, a data item run is initiated for this data item (data items can be nested 10 levels deep).
8. When there are no more records to be processed in a data item, control returns to the point from which the processing was initiated. For an indented data item this is the next record of the data item on the next higher level. If the data item is already on the highest level (indentation is zero), control returns to the report.
9. When the first data item has been processed, the next (if any) data item is processed in the same way.
10. When there are no more data items, the `OnPostReport` trigger is called. You can use this trigger to do any post processing that is necessary, for example, cleaning up by removing temporary files.

Report Functions

Certain functions can only be used in reports. These functions can be useful when you create complex reports.

CurrReport.SKIP: Use this function to skip the current record of the current data item. If a record is skipped, it is not included in totals and it is not printed. Skipping a record in a report is much slower than never reading it at all, so use filters as much as possible.

CurrReport.BREAK: Use this function to skip the rest of the processing of the data item that you are currently processing. The report resumes processing the next data item. All data items indented under the one that caused the break are also skipped.

CurrReport.QUIT: This function skips the rest of the report. It is not an error, however. It is a normal ending for a report.

CurrReport.PAGENO: Use this function to return the current page number of a report and/or to set a new page number.

CurrReport.CREATETOTALS: Use this function to maintain totals for a variable in the same way as totals are maintained for fields by using the TotalFields property. This function must be used in the OnPreDataItem trigger of the data item in the sections in which you will display the totals.

CurrReport.TOTALSCAUSEDBY: Use this function to determine which field caused a break to occur. The return value is the field number of the field that the data item is grouped on that changed and caused a Group Header or Group Footer section to print. This function is almost always used in the OnPreSection trigger of Group Header and Group Footer sections. This function must always be used when grouping more than one field for a single data item.

CurrReport.NEWPAGE: Use this function to force a page break when printing a report. This is usually found in the data item triggers.

CurrReport.PREVIEW: Use this function to determine whether a report is being printed in preview mode or not.

CurrReport.SHOWOUTPUT: Use this function to return the current setting of whether a section should be outputted or not, and to change this setting. This function should only be used in the OnPreSection trigger of a section. If TRUE is passed to the function, nothing changes and the section will print as normal. If FALSE is passed to the function, the section will not be printed for this iteration of the data item.

Processing-Only Reports

A processing-only report is one that does not print but instead changes table data. Printing reports can also change records. This section applies to those reports as well. You can specify a report to be "Processing Only" by changing the ProcessingOnly property of the Report object. The report functions just as it is supposed to (processing data items), but it does not print any sections.

The request form changes slightly as well by removing the **Print** and **Preview** buttons and replacing them with an **OK** button (**Cancel** and **Help** buttons stay).

Here are some helpful hints for writing a processing-only report:

- Change the ProcessingOnly property to Yes.
- Decide which tables should be read – these are the data items.
- Most of the code will go into the OnAfterGetRecord trigger.
- Do not forget the INSERT or MODIFY functions.
- Use a dialog to show the user the progress, and allow the user to cancel the report.

There are a few advantages to using a report to process data rather than a codeunit. One is that the request form functionality that allows the user to select options and filters for data items is readily available in a report, but difficult to program in a codeunit. Using the features of the Report Designer ensures consistency. Finally, instead of writing code to open tables and to retrieve records, you can simply use data items.

Test Your Skills – Managing Reporting – Diagnosis

Description

Our client's functional requirements describe their reporting needs in the following way:

You should be able to print a list of the participants registered for a seminar.

Upon completion of some seminars, participants receive a seminar certificate. We want to create such certificates from the system for individual participants.

When a seminar is finished, we want to invoice the customers for the participation of their registered participants. Invoicing will be by project and resources.

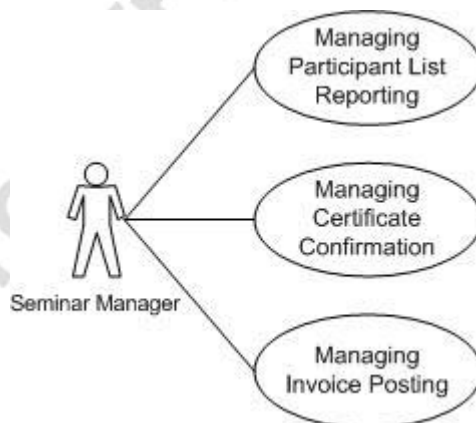
There are three main reports that can be created to fulfill these requirements: a participant list, a certificate confirmation and a processing-only report that posts invoices.

Use Cases

We can split the creation of reports for the seminar module into the following three use cases:

- Managing Participant List Reporting
- Managing Certificate Confirmation
- Managing Invoice Posting

The following diagram illustrates the use cases:



Implementation of Use Case 1 – Managing Participant List Reporting

Managing Participant List Reporting – Analysis

Our client's functional requirements describe the participant list report in the following way:

You should be able to print a list of the participants registered for a seminar.

The client requires a report that lists the participants who have registered for a seminar.

Purpose

This report will allow the user to see who has been registered for a seminar.

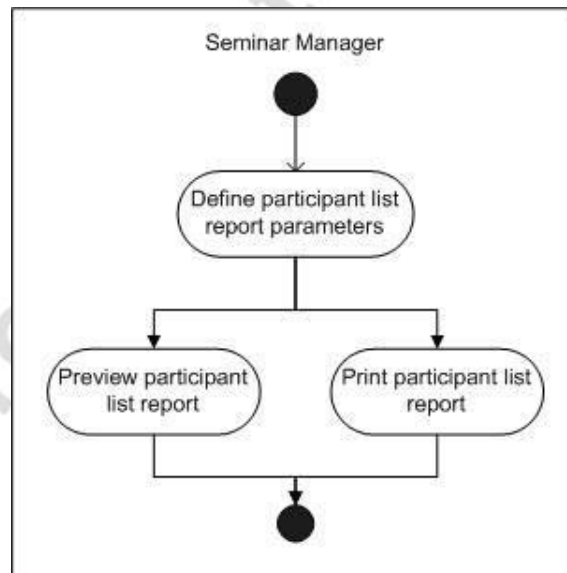
Postconditions

A report listing participants who are registered for a seminar can be printed from either the main Seminar Menu or from the Seminar Registration form.

Main Scenario

When seminar managers or instructors want to see or print a list of participants who are registered for a seminar, they will preview or print the report with this information from either the Seminar Menu, Reports option or the Seminar Registration form.

Activity Diagram



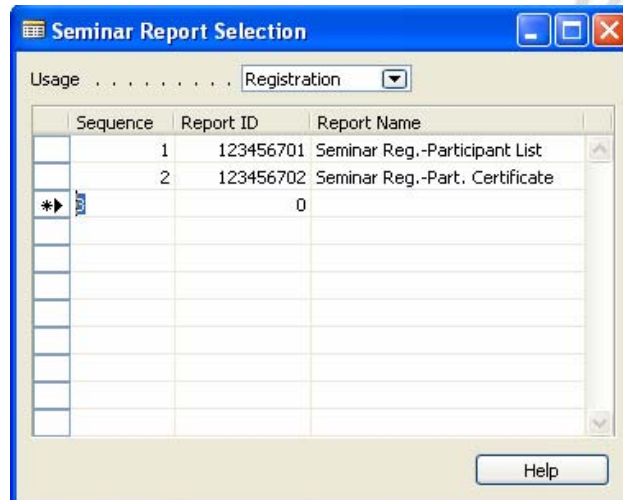
Managing Participant List Reporting – Design

To implement this report, create:

- The report itself.
- The request form to set the parameters of the report.
- The controls to access the report from forms.
- A form from which seminar reports can be selected.

GUI Design

Seminar Report Selection (Form 123456723): This form displays the available seminar reports.



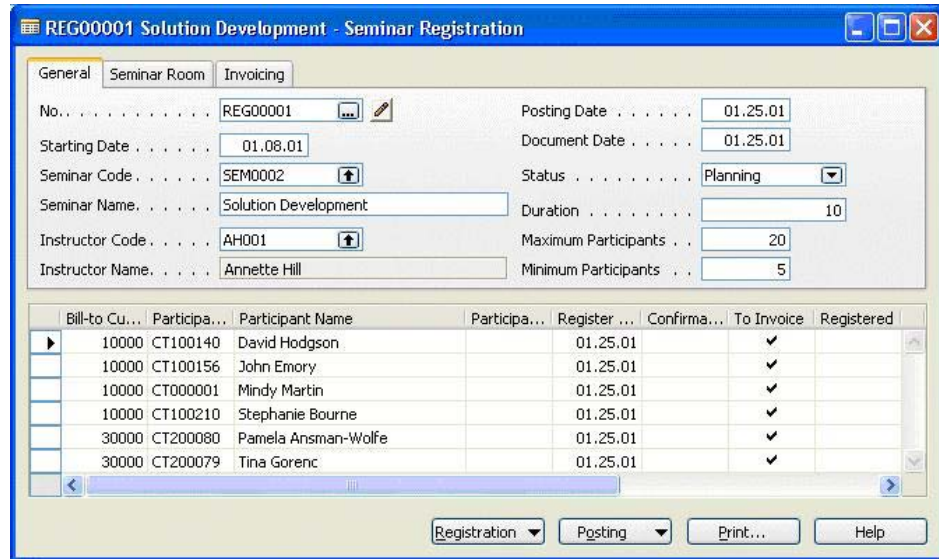
Seminar Menu: Modify this menu by adding the following menu item to the Reports Group (under Order Processing):

Menu Type	Menu Name	Group	Comment
Item	Seminar Reg.-Participant List	Reports	Opens report 123456701 Seminar Reg.-Participant List.
Item	Seminar Reg.-Part. Certificate	Reports	Opens report 123456702 Seminar Reg.-Part. Certificate.

Add the following menu item to the Setup Group:

Menu Type	Menu Name	Group	Comments
Item	Report Selections	Setup	Opens form 123456723 Seminar Report Selection.

Seminar Registration (Form 123456710): Modify this form by adding a **Print** command button from which to start the Participant List report.



Functional Design

To create this report, identify the tables from which the information should come and decide how to process the information.

The information for the report comes from the Seminar Registration Header and Seminar Registration Lines.

When running this report, the user should select which Seminar Registration Headers should be included. The program should then, for each Seminar Registration Header, print information from each corresponding Seminar Registration Line.

Table Design

Create one new table:

Table 123456705 Seminar Report Selections

Modify:

Table 123456710 Seminar Registration Header

Managing Participant List Reporting – Development

We begin development by creating the Participant List report and then we create the means by which seminar reports can be selected from a form.

Exercise 16 – Creating the Participant List Report

1. First, add a field to table 123456710 Seminar Registration Header as follows:

No.	Field Name	Type	Length	Comment
40	No. Printed	Integer		Must not be editable.

In the next tasks, create a codeunit to increment the **No. Printed** field you just added to the Seminar Registration Header table.

2. Create codeunit 123456702 Seminar Registration-Printed. Set the property to specify Seminar Registration Header as the source table for this codeunit.
3. Enter code in the appropriate trigger so that when the program runs this codeunit, it finds the Seminar Registration Header record, increases the **No. Printed** by 1, modifies and commits the table.

You are now ready to create the actual report in the next tasks.

4. Use the wizard to create form-type report 123456701 Seminar Reg.-Participant List based on table 123456710 Seminar Registration Header table. Add fields **No.**, **<Separator>**, **Seminar Code**, **Seminar Name**, **Starting Date**, **Duration**, **Instructor Name** and **Room Name** to the report. Save the report.
5. Set the properties for the Seminar Registration Header data item so that it is sorted by **No.** and so that the filter fields the user can fill in when running the report are **No.**, **Seminar Code**, **No. Printed**.
6. Add a data item called CopyLoop for the Integer virtual table.
 - Set the property so that the data item is indented to the first level.
 - Set the DataItemTableView property so that the table is sorted by Number.
 - Set the property so that a new page is created for each record.
7. Add a data item called PageLoop for the Integer virtual table.
 - Set the property so that the data item is indented to the second level.
 - Set the property so that the table is sorted by Number where the Number is 1.

8. Add a data item for the Seminar Registration Line table.
 - Set the property so that the data item is indented to the third level.
 - Set the property so that the data item is sorted by Document No., Line No.
 - Set the DataItemLinkReference and the DataItemLink properties so that the data item is linked to the Seminar Registration Header data item.
9. Set the property for the report so the caption is Seminar Reg.- Participant List.
10. Add the following sections to the report:
 - Two headers for the PageLoop data item.
 - A header and body section for the Seminar Registration Line data item.
11. Delete all sections except the ones you have just added.
12. Move the fields from the Seminar Registration Header header section to the first PageLoop header section and all the fields from the Seminar Registration Header body section to the second PageLoop header section.
 - Set the property for the Seminar Registration Line header section so that it prints on every page.
 - Set the property for the first PageLoop header section so that the section prints on every page.
 - Set the property for the second PageLoop header section so that the section prints on every page.
13. Add the fields **Bill-to Customer No.**, **Participant Contact No.** and **Participant Name** to the Seminar Registration Line body section. Move the labels for these fields to the Seminar Registration Line header section.
14. In the second PageLoop header section, you now have the text boxes that identify the seminar registration and seminar information. Because the fields have been moved from a different section, the program does not know from which table the data for the text boxes should be taken. Set the SourceExpr property for each text box in this section so that the table is specified along with the field name.

15. Define the following global variables for the report:

Name	Data Type	Subtype	Length
SeminarCountPrinted	Codeunit	Seminar Registration-Printed	
NoOfCopies	Integer		
NoOfLoops	Integer		

16. Enter code in the appropriate trigger so that after the program gets the record of the Seminar Registration Header table, the program calculates the **Instructor Name** field.
17. Enter code in the appropriate trigger so that before the CopyLoop data item is run, the program sets the NoOfLoops to one more than the absolute value of the NoOfCopies and filters the table to the records between 1 and the NoOfLoops.
18. Enter code in the appropriate trigger so that after the program gets the record for the CopyLoop data item, the program sets the page number of the current report to 1.
19. Enter code in the appropriate trigger so that after the CopyLoop data item, if the current report is not being previewed, the program runs the Seminar Registration-Printed codeunit for the Seminar Registration Header record.
20. On the request form for the report, add a text box with the caption "No. of Copies" with the source expression of the NoOfCopies variable.
21. Set the properties for the request form so that the form saves the values after it is closed.

Now that we have created the report, we can create the table and form from which the report can be selected and run.

22. Create Table 123456705 Seminar Report Selections with the following fields:

No.	Field Name	Type	Length	Comment
1	Usage	Options		Options: S.Registration.
2	Sequence	Code	10	Numeric field.
3	Report ID	Integer		Relation to the ID field of the Object table where the Type=Report.
4	Report Name	Text	80	FlowField; CalcFormula looks up the Object Caption field of the AllObjWithCaption table where the Object Type=Report and the Object ID= the field number of the Report ID. Must not be editable.

The primary key for this table is **Usage, Sequence**.

23. Enter code in the appropriate trigger so that when the user enters or changes the value in the **Report ID** field, the program calculates the Report Name value.
24. Define a new function called NewRecord.
25. Enter code in the function trigger so that the function filters the Seminar Report Selection table to the corresponding Usage. If the function can find the last record in the record set and if the **Sequence** for the record is not blank, the function increments the **Sequence**. If the function cannot find the last record or if the **Sequence** is blank, the function sets the **Sequence** to 1.
26. Create form 123456723 Seminar Report Selection with the fields **Sequence, Report ID** and **Report Name** as shown in the GUI design. Set the properties for the form so that it saves the values when the form is closed.
27. Set the properties for the **Report ID** text box so that the lookup form is the Objects form.
28. Set the properties for the **Report Name** text box so that there is no drill down and so that the lookup form is the Objects form.
29. Define a global variable called ReportUsage with the data type Option and a single option of Registration.

30. Define a function called SetUsageFilter. Enter code in the function trigger so that the function sets the filtergroup to 2, and if the ReportUsage is Registration, the function filters the table to where Usage is S. Registration. At the end, the function resets the filtergroup to 0.
31. Add a text box labeled Usage to the top of the form as shown in the GUI design. Set the properties of the text box so that the option is Registration and the source expression is the ReportUsage variable.
32. Enter code in the appropriate trigger so that the program runs the SetUsageFilter function when the user enters or changes the value in the **Usage** text box and when the program opens the form.
33. Enter code in the appropriate trigger so that after the program validates the **Usage** text box, the program updates the form.
34. Enter code in the appropriate trigger so that when the form opens a new record, the program runs the NewRecord function.

You can now add menu items to the main Seminar Menu to allow the user to run the reports.

35. Add the Report Selections menu item to the Setup menu item in the Seminar Menu as shown in the GUI design.
36. Add the reports as shown in the GUI design into the Partner Menu Suite under the Order Processing – Reports group.

You want the user to be able to run the participant list for a particular seminar registration from the registration form. To enable this, add a command button to the form that will run a codeunit to print the report. You will develop the codeunit first.

37. Create codeunit 123456703 Seminar Document-Print.
38. Define a function called PrintSeminarRegistrationHeader that takes one parameter. This parameter is a record variable of the Seminar Registration Header table, called SemRegHeader.
39. Enter code in the function trigger so that the function filters the SemRegHeader to the No. of the SemRegHeader record. The function filters the Seminar Report Selection table to those records with a **Usage** of S. Registration and a **Report ID** that is not 0. The function finds the first record in this set, and for each record in the set, modally runs the report corresponding to the **Report ID**.
40. Add the **Print** command button to form 123456710 Seminar Registration as shown in the GUI design. Enter code in the appropriate trigger so that when the user clicks this button, the program runs the PrintSeminarRegistrationHeader function of the Seminar Document-Print codeunit.

Managing Participant List Reporting – Testing

1. If you do not have any Seminar Registration records, create one by selecting SEMINARS→ORDER PROCESSING→REGISTRATIONS from the main menu. Fill in all of the fields in the **General** and **Seminar Rooms** tabs and add at least two participants.
2. Use the Seminar Report Selection window to set up the Seminar Reg.-Participant List as the report to be run for Registration. To do so, select SEMINARS→SETUP→REPORT SELECTIONS from the main menu. Select Registration as the **Usage** and 123456701 on the first line for the **Report ID**. The report name should be filled in automatically by our code.
3. Open the Seminar Registration form again and view the record you prepared in step 1. Click the **Print** button.
4. The request form should open with the No. set to the current registration. Select **Print** or **Preview** to print and check your report.
5. Try printing the report with different parameters and registration data to verify that it is working correctly.

Implementation of Use Case 2 – Managing Certificate Confirmation

Managing Certificate Confirmation – Analysis

Our client's functional requirements describe the certificate confirmation report in the following way:

Upon completion of some seminars, participants receive a seminar certificate. We want to create such certificates from the system for individual participants.

From this we can see that we need a printed certificate confirmation report that can be run for all the participants who completed a seminar.

Purpose

The certificate confirmation report allows the client to provide participants with a printed confirmation of successful participation in a seminar.

Preconditions

The precondition for this report is that a seminar registration must be posted.

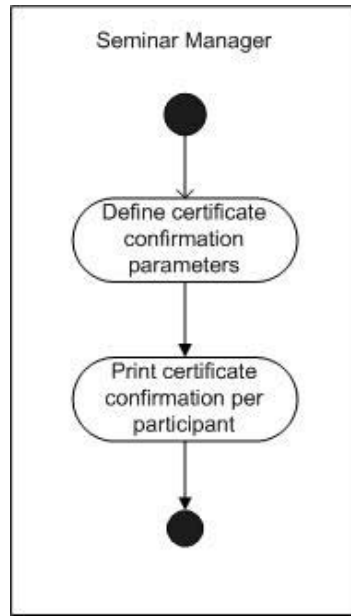
Postconditions

After this report has been run, there will be a printed certificate confirmation for each participant who attended the seminar.

Main Scenario

At the end of the seminar, the seminar managers will print certificate confirmations for each participant in the seminar.

Activity Diagram



Managing Certificate Confirmation – Design

GUI Design

No additional forms or navigation are necessary for this use case.

Functional Design

This report must create a page for each participant in a given seminar registration. Therefore, we need to use the Seminar Registration Header table and Seminar Registration Line table.

For each Seminar Registration Header selected by the user in the request form, the report should print one certificate for each participant, with information from the **Seminar Name**, **Starting Date** and **Instructor Name** fields on the Seminar Registration Header table and from the **Participant Name** field on the Seminar Registration Line table.

The client has provided a model for how this report should appear. This sample report is in the appendix of this manual.

Table Design

No additional tables or fields are necessary for this use case.

Managing Certificate Confirmation – Development

Exercise 17 – Creating the Certificate Confirmation Report

1. Create report 123456702 Seminar Reg.-Part. Certificate with the data items Seminar Registration Header and Seminar Registration Line.
2. Set the properties for the Seminar Registration Header data item so that the table is sorted by **No.** The user should be able to filter on the fields **No.**, **Starting Date** and **Seminar Code** when running the report. The data item should only print if there are records to be printed.
3. Set the properties for the Seminar Registration Line data item so that it is indented to the first level and so that the data item is sorted by **Document No.** and **Line No.** Set the link property and set the property so that there is a new page for each record.
4. Define a global record variable of the Company Information table, called CompanyInfo.
5. Enter code in the appropriate trigger so that before the program runs the Seminar Registration Header data item, it gets the CompanyInfo record and calculates the **Picture**.
6. Enter code in the appropriate trigger so that after the program gets the record for the Seminar Registration Header data item, it calculates the **Instructor Name**.
7. Enter code in the appropriate trigger so that after the program gets the record for the **Seminar Registration Line** data item, it skips to the next record if there is no **Participant Name** in this record.
8. The sections we use in this report are two body sections and one footer section, all for the **Seminar Registration Line** data item. Set the property for the footer section so that the section prints at the bottom of every page.
9. Place a picture box in the first body section as shown in the sample report in the appendix. The source for the picture box is the **Picture** field of the CompanyInfo record.
10. Place the following labels on the report for the fixed text of the report as shown in the sample report in Appendix A:
 - The "Participant Certificate" title, with a font size of 30 and bold.
 - The "has participated in seminar" phrase, with a font size of 15.

11. Set the captions for the labels so that they display the correct information.
12. Place text boxes on the report for the variable text of the report:
 - The participant name, with a font size of 25 and bold.
 - The seminar name, with a font size of 20 and bold.
 - The "on <date>" phrase, with a font size of 15.
 - The instructor's name, with a font size of 9.
13. Set the source expressions for the text boxes so that they display the correct information.
14. Add a shape to the report in the footer section. Set the properties for the shape so that the shape style is that of a horizontal line.

Managing Certificate Confirmation – Testing

1. Select SEMINARS→ORDER PROCESSING→REPORTS→SEMINAR REG.-PART. CERTIFICATION from the main menu.
2. Select a registration and Print or Preview to verify that the report is functioning correctly.

Implementation of Use Case 3 – Managing Invoice Posting

Managing Invoice Posting – Analysis

Our client's functional requirements describe the invoice posting in the following way:

When a seminar is finished, we want to invoice the customers for the participation of their registered participants. Invoicing will be by project and resources.

From this we can see that we need to create a report to produce invoices for customers where the lines are for one or more participants who have attended a seminar, as well as any charges.

Purpose

The purpose of this report is to enable customers to be invoiced for participation in seminars.

Preconditions

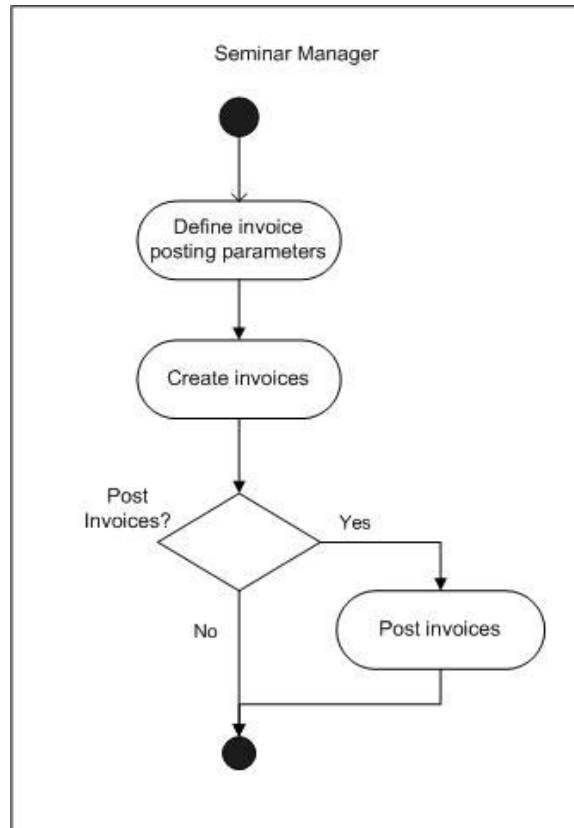
Seminar registration, posted seminar registration, customer and contact information must exist for the creation of this report.

Postconditions

The result of this use case will be a report that can be run to automatically generate invoices for one or more seminar registrations.

Main Scenario

After posting a seminar registration, the seminar managers will run the invoice posting report to create a batch of invoices for one or more customers, with one or more lines on each for the participants who registered in the seminar.

Activity Diagram**Managing Invoice Posting – Design****GUI Design**

No additional forms or navigation are necessary for this use case.

Functional Design

This report must create an invoice for each customer within a filter specified by the user, with lines for each participant in a registered seminar (that falls within the filter set by the user). The user should also have the option of either just creating the invoices or both creating and posting them at the same time.

Because we only want to create invoices for posted seminars, and because there can be additional charges, we should use the Seminar Ledger Entries to get the information to run this report. The report should create Sales Headers and Sales Lines as appropriate, and if the user has selected the option to post the invoices as well, the report should run the posting process for sales invoices.

Table Design

No additional tables or fields are necessary for this use case.

Managing Invoice Posting – Development

This report is different from the previous two in that it is not a report to be printed but is instead a processing-only report.

Exercise 17 – Creating the Invoice Posting Report

1. Create report 123456700 Create Seminar Invoices. Set the property to specify that this report is for processing-only.
2. Create a data item for the Seminar Ledger Entry table and set the properties for the data item as follows:
 - Specify the view of the data item to be sorted by **Bill-to Customer No., Seminar Registration No., Charge Type and Participant Contact No.**, where the **Remaining Amount** is not 0.
 - Specify that the user should be able to filter on the **Bill-to Customer No., Seminar No. and Posting Date.**
 - Specify the variable name of the data item as SeminarLedgerEntry.
3. Define the following global variables for the report:

Name	Data Type	Subtype	Length
SalesHeader	Record	Sales Header	
SalesLine	Record	Sales Line	
SalesSetup	Record	Sales & Receivables Setup	
GLSetup	Record	General Ledger Setup	
Cust	Record	Customer	
Job	Record	Job	
JobLedgEntry	Record	Job Ledger Entry	
ApplJobLedgEntry	Record	Job Ledger Entry	
JobPostingGr	Record	Job Posting Group	
CurrExchRate	Record	Currency Exchange Rate	
SalesCalcDisc	Codeunit	Sales-Calc. Discount	
SalesPost	Codeunit	Sales-Post	
JobLedgEntrySign	Code		10
Window	Dialog		
PostingDateReq	Date		

Name	Data Type	Subtype	Length
DocDateReq	Date		
CalcInvDisc	Boolean		
PostInv	Boolean		
NextLineNo	Integer		
NoOfSalesInvErrors	Integer		
NoOfSalesInv	Integer		

4. Define the following text constants for the codeunit:

Name	ConstValue
Text000	Please enter the posting date.
Text001	Please enter the document date.
Text002	Creating Seminar Invoices...\\
Text003	Customer No. #1#####\
Text004	Registration No. #2#####\
Text005	The number of invoice(s) created is %1.
Text006	Not all the invoices were posted. A total of %1 invoices were not posted.
Text007	There is nothing to invoice.

5. Define a function called FinalizeSalesInvHeader. Set the property to specify that this is a local function.
6. Define a function called InsertSalesInvHeader. Set the property to specify that this is a local function.
7. Enter code in the FinalizeSalesInvHeader function trigger so that the function performs the following tasks:
 - If the CalcInvDisc variable is TRUE, the function runs the Sales-Calc. Discount codeunit with the Sales Line record and finds the Sales Header.
 - The function then performs a commit.
 - The function clears the SalesCalcDisc and SalesPost variables and increments the NoOfSalesInv by one.
 - If the PostInv variable is TRUE, the function clears the SalesPost variable. If running the Sales-Post codeunit with the SalesHeader returns FALSE, the function should increment the NoOfSalesInvErrors by one.

8. Enter code into the InsertSalesInvHeader function trigger so that the trigger performs the following tasks:
 - Initializes a new Sales Header record with a **Document Type** of Invoice and a blank **No.** and inserts it into the database.
 - Validates that the **Sell-To Customer No.** of the new record equals the **Bill-to Customer No.** of the ledger entry.
 - If the **Bill-to Customer No.** value is not the same as that of the **Sell-To Customer No.**, the program should validate that the **Bill-to Customer No.** of the new record equals the **Bill-to Customer No.** of the ledger entry.
 - Validates that the **Posting Date** of the new record is the PostingDateReq.
 - Validates that the **Document Date** of the new record is the DocDateReq.
 - Validates that the **Currency Code** of the new record is blank.
 - Modifies and commits the new record.
 - Sets the NextLineNo variable to 10000.

9. Enter code into the appropriate trigger so that just before the data item is run, the program performs the following tasks:
 - Shows an error if the PostingDateReq or DocDateReq is empty.
 - Opens a dialog window with the text constants Text002, Text003 and Text004.

10. Enter code in the appropriate trigger so that after the program gets a record for the data item, the program performs the following tasks:
 - Gets the Job Ledger Entry corresponding to the Seminar Ledger Entry record.
 - If the **No.** of the Customer record is not the same as the **Bill-to Customer No.**, the program gets the Customer record corresponding to the **Bill-to Customer No.**
 - If the Customer is not blocked for All or is only blocked for Invoice, the program proceeds with the tasks that follow.
 - If the **Bill-to Customer No.** is different from that on the current Sales Header record, the program updates the dialog window with the new **Bill-to Customer No.** If the **No.** on the Sales Header is not blank, it runs the FinalizeSalesInvHeader function. The program then runs the InsertSalesInvHeader function and sets the **Document Type** and **Document No.** fields on the Sales Line to be those of the Sales Header.
 - Updates the dialog window with the **Sales Registration No.**
 - Sets the **Type** on the Sales Line according to the **Type** on the Job Ledger Entry.

- If the **Type** on the Job Ledger Entry is G/L Account, the program tests that the Job Posting Group on the corresponding Job record is not blank. The program tests that the **G/L Exp. Sales Acc.** field on the corresponding Job Posting Group record is not blank. The program sets the **No.** of the Sales Line to the G/L Exp. Sales Acc. value.
 - If the **Type** on the Job Ledger Entry is not G/L Account, the program sets the **No.** of the Sales Line to the **No.** on the Job Ledger Entry.
 - Fills the Sales Line fields as follows: **Document Type** and **Document No.** from the Sales Header, **Line No.** from the NextLineNo variable, **Description** from the Seminar Ledger Entry, **Work Type Code** and **Unit of Measure Code** from the Job Ledger Entry, and **Unit Price** from dividing the **Total Price** from the Job Ledger Entry by the **Quantity**.
 - If the **Currency Code** on the Sales Header is not blank, the program tests that the **Currency Factor** is blank and calculates the **Unit Price** for the Sales Line by running the ExchangeAmtLCYToFCY function of the Currency Exchange Rate table.
 - Fill the Sales Line fields as follows: **Unit Cost (LCY)** from the Total Cost of the Job Ledger Entry divided by the Quantity, and the **Quantity, Job No., Phase Code, Task Code, and Step Code** fields from the Job Ledger Entry.
 - If the **Total Price** on the JobLedgEntry is greater than 0, the program sets the JobLedgEntrySign to '+' and filters the ApplJobLedgEntry records to those where the **Applies-To ID** matches that of the JobLedgEntry, with a '-' sign at the end (meaning JobLedgEntry."Applies-To ID" + '-'). For these records, the program modifies the Applies-To ID to blank. If the Total Price is less than 0, the program performs the same steps with opposite signs.
 - Sets the **Applies-To ID** of the Job Ledger Entry record to the **Entry No.** plus the JobLedgEntrySign and modifies the record.
 - Sets the **Job Applies-To ID** to the **Applies-To ID** of the Job Ledger Entry record.
 - Sets the **Apply** and **Close (Job)** field on the SalesLine to TRUE.
 - Inserts the Sales Line record.
 - Increments the NextLineNo by 10000.
 - If the Customer record is blocked for All or Invoice, increments the NoOfSalesInvErrors by one.
11. Enter code in the appropriate trigger so that when the program posts the data item, it performs the following tasks:
- Closes the dialog window.

- If the **No.** of the Sales Header is not blank, runs the FinalizeSalesInvHeader function. The program then displays a message showing the number of errors that occurred, if any, or the number of invoices created.
 - If the **No.** of the Sales Header is blank, the program shows a message saying that there is nothing to invoice.
12. Place two new text boxes and labels on the request form. The sources for these text boxes are the variables PostingDateReq and DocDateReq.
 13. Place two new check boxes and labels on the request form. The sources for these check boxes are the variables CalcInvDisc and PostInv.
 14. Set the property for the request form so that the form saves the values after the form is closed.
 15. Enter code in the appropriate trigger so that when the request form is opened, if the PostingDateReq and DocDateReq variables are not filled, they are set to the work date. Set the CalcInvDisc value to the value of the **Calc. Inv. Discount** field in the Sales Setup.

Managing Invoice Posting – Testing

To test this portion of the seminar module, you need to have completed the testing and set up described in the previous exercises. In particular, it is important that the Job and Customer associated with the registrations you are testing have been set up correctly.

1. Select SEMINARS→PERIODIC ACTIVITIES→PERIODIC ACTIVITIES from the main menu.
2. The Create Seminar Invoices request form should open. Enter data so that one or more of your posted registrations meet the criteria. (If you do not have any posted registrations, create one.) Select **OK** to run the processing-only report.
3. Look at the Sales Invoices that were created. See that the header and lines were created correctly based on the Seminar information. You should expect to have one invoice per Bill-to Customer with a line for each contact registered. Check that the data flowed correctly from the registration (and the associated jobs, etc.) to the invoice.
4. Try running the Create Seminar Invoices process with the Post Invoices Option marked and unmarked and verify that the Sales Invoices are posted as indicated.

Test Your Knowledge

Review Questions

1. Place these triggers in the order in which they would "fire" in normal report processing: OnPostDataItem, OnPreReport, OnAfterGetRecord, OnPreDataItem, OnPostReport, OnInitReport.
2. What report function would you use to skip the processing of one record in a data item?
3. What are the advantages of using processing-only reports in some situations as opposed to codeunits?
4. What does the following line of code do and in what trigger would you expect it to appear?

```
CurrReport.SHOWOUTPUT(CurrReport.TOTALSCAUSED BY =  
FIELDNO("Posting Date"));
```

5. What do you have to do to print FlowField data when running a report? What trigger would you use for this code?
6. In what trigger do you first have access to the values the user entered on the request form?

Conclusion

Chapter Summary

In this chapter, you created three reports. The first two, Participant List and Certificate Confirmation were normal reports. The third report, however, was a processing-only report to enable the creation of invoices for customers with participants in completed seminars.

Positioning – Where do you go from here?

You are now ready to create statistics for the seminar module.

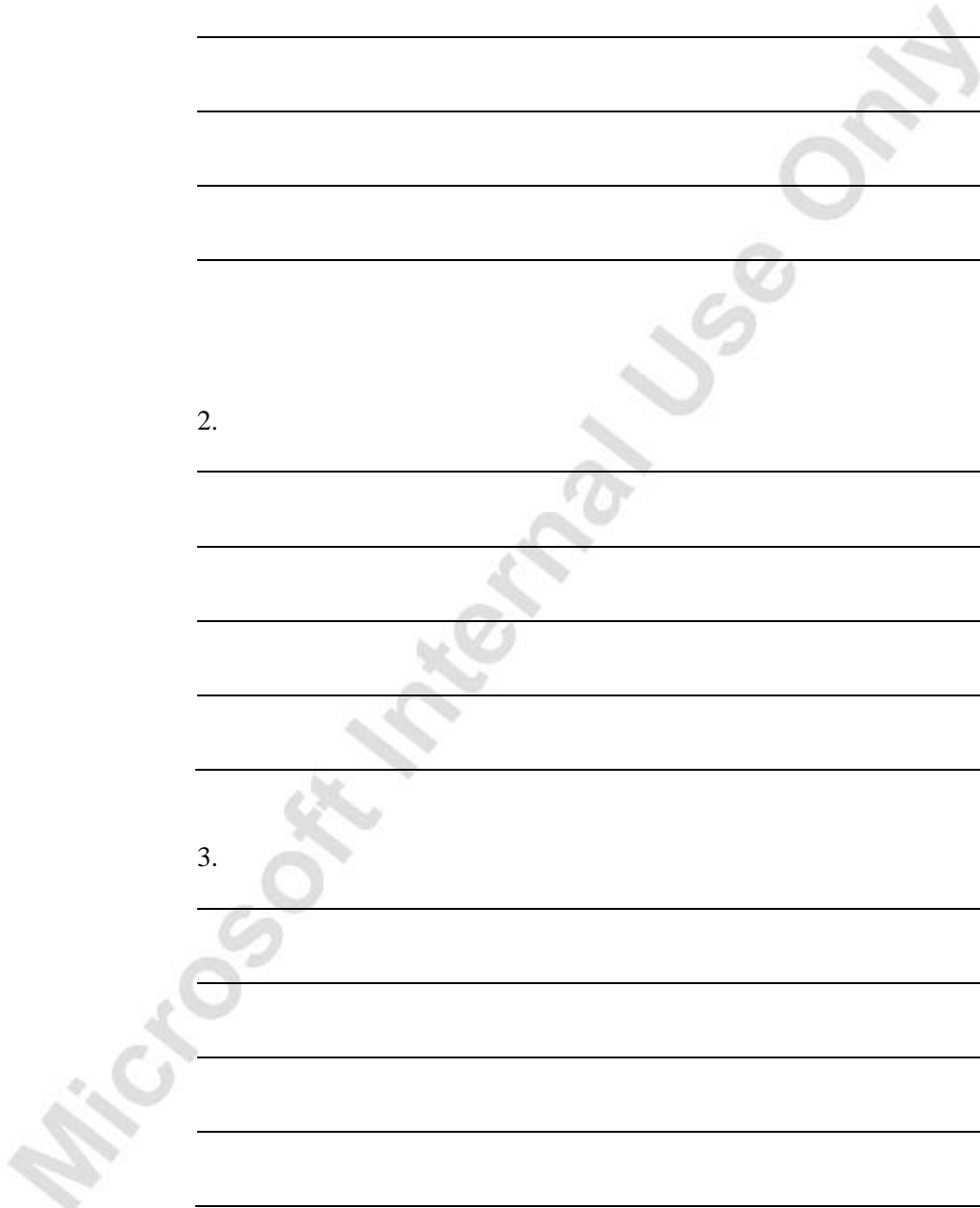
Quick Interaction: Lessons Learned

Take a moment to write down three Key Points you have learned from this chapter:

- 1.

- 2.

- 3.



CHAPTER 7: MANAGING STATISTICS

This chapter contains the following sections:

- Introduction
- Using FlowFilters in Calculation Formulas
- Test Your Skills
 - Diagnosis
 - Managing Seminar Statistics
- Test Your Knowledge
- Conclusion

Microsoft Internal Use Only

Introduction

Positioning – What is your starting point?

So far, the seminar module includes master tables and forms, a means to create seminar registrations and the routines necessary to post the registrations. These features are integrated into the standard application so that they can be accessed from the Navigation Pane. There are also reports for the module. You can now add statistics functionality.

Preconditions

To create the statistics functionality for the seminar module, you must have master tables and forms, tables and forms for seminar registration, and the seminar posting routines.

Further preconditions are knowledge of the following areas:

- Writing internal documentation
- Working with event triggers
- Working with complex data types and their member functions
- Creating journal and document posting routines
- Debugging code
- Programming for low-impact on the application

Business Goals

In this story, your goal is to add a feature to the seminar module: the ability to see statistical information.

Educational Goals

By completing this story, you will have learned or reacquainted yourself with the following:

- Using FlowFields and FlowFilters for calculations

Using FlowFilters in Calculation Formulas

End users may want to limit calculations so that they include only those values in a column that have some specific properties. For example, on our Seminar Statistics form, we want to sum up the total price of a seminar four different times, for four different, specific time periods. This is possible if the application has been designed to take advantage of SumIndexField Technology (SIFT) using FlowFilter fields in connection with the FlowFields. FlowFilters are used to determine how much information the system will include when it calculates the contents of FlowFields.

When defining a FlowField, you create a calculation formula that can consist of constants, of values from ordinary fields and of filters given as parameters in FlowFilter fields. FlowFilter fields are fields in which the end user can enter a filter value that will affect the calculation of a FlowField.

For an example of how to implement a statistics form using the FlowFields and FlowFilter fields in a master table, let's look at Table 18 Customer and Form 151 Customer Statistics.

The first line of the **Customer Statistics Sales** tab shows the Sales (LCY) for four different time periods: the current month, This Year, Last Year and To Date. The data shown in these fields is generated by means of a FlowField – Sales (LCY) – and a number of FlowFilters in the Customer table. The CalcFormula shown in the Sales (LCY) field properties uses a number of FlowFilters, but to simplify we'll just consider the Date Filter.

In the OnAfterGetRecord trigger of the Customer Statistics form, the Date Filter is set for each of the desired time periods using the Date Filter-Calc codeunit. The CALCFIELDS function is then used for each Date Filter to calculate a value for the Sales (LCY). We will use similar logic when creating the Seminar Statistics form.

For more information on SIFT, FlowFields and FlowFilters, please refer to the Application Designer's Guide.

Test Your Skills – Managing Seminar Statistics – Diagnosis

Description

Our client's functional requirements provide the following description of their statistical needs:

We would like to be able to see statistical information regarding the total price for each seminar, broken down into what is chargeable and what is not chargeable. We would like to see these statistics for different time periods such as for a month, for Last Year, for This Year and To Date.

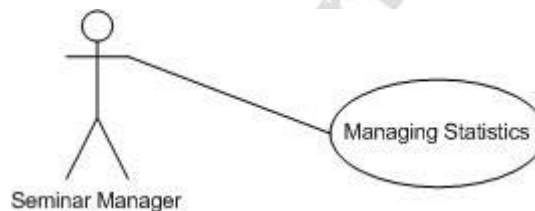
From this you can see that we need a new statistics feature that operates on the seminar detail information. To enable this feature, you must add FlowFields and FlowFilters to existing tables.

Use Cases

You can characterize the task of creating a statistics feature for the seminar module with the following use case:

- Managing Seminar Statistics

The following diagram illustrates the use cases:



Implementation of Use Case 1 – Managing Statistics

Managing Seminar Statistics – Analysis

Our client's functional requirements describe their need for a statistics feature in the following way:

You want to see statistical information regarding the total price for each seminar, broken down into what is chargeable and what is not chargeable. You want to see these statistics for different time periods such as for a month, Last Year, This Year and To Date.

From this you can see that the client wants to be able to open a statistics form from a seminar form. The form should instantly calculate the statistics for the total price and show it for the four time periods listed.

Purpose

The purpose of the seminar statistics feature is to allow the user to quickly and easily get an overview of the price statistics for a specific seminar.

Preconditions

A table containing posted seminar detail information must exist.

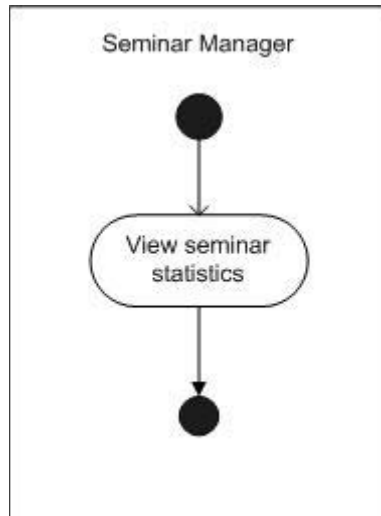
Postconditions

The result of this use case will be a form that calculates the total price statistics for a given seminar within four time periods: a month, Last Year, This Year and To Date.

Main Scenario

Seminar managers will access the seminar statistics from a seminar detail form. The form will calculate the total price statistics for the seminar for four time periods: the current month, Last Year, This Year, and To Date.

Activity Diagram



Managing Seminar Statistics – Design

The Seminar Statistics form will be accessible from the Seminar Card and Seminar List forms, and will automatically calculate the statistics for the selected seminar using the information from the Seminar Ledger Entry.

GUI Design

Create a Seminar Statistics form and make it accessible from the Seminar Card and the Seminar List forms.

Seminar Statistics (Form 123456714): This form displays statistical information for one seminar:

	January	This Year	Last Year	To Date
Total Price	1.500,00	1.500,00	0,00	5.500,00
Total Price (Chargeable) .	1.500,00	1.500,00	0,00	5.500,00
Total Price (Not Charg...	0,00	0,00	0,00	0,00

Seminar Card (Form 123456700): The Seminar menu button on this form will be modified to include a new menu item for seminar statistics.

Seminar List (Form 123456701): The Seminar menu button on this form will be modified to include a new menu item for seminar statistics.

Table Design

You need to add FlowFields and FlowFilters to the table that contains seminar information, table 123456700 Seminar. Also add a key of **Seminar No., Posting Date, Charge Type, Chargeable** to table 123456732 Seminar Ledger Entry.

Functional Design

To calculate the total prices, use the FlowFields and FlowFilters added to the Seminar table.

We want to calculate the Total Price, Total Price (Chargeable) and Total Price (Not Chargeable) in the Seminar table, using values from the Seminar Ledger Entry, for four different time periods. To store the totals, use an array of four dimensions for each of these three prices. Then use the standard DateFilter-Calc codeunit to calculate date filters for the time periods. With the date filters, we can filter the Seminar table and calculate the price fields to be shown on the form for each of the four time periods.

Managing Statistics – Development

Exercise 19 – Creating FlowFields for Sums

1. Add a secondary key of **Seminar No., Posting Date, Charge Type, Chargeable** to table 123456732 Seminar Ledger Entry. Set the property for the key so that the **Total Price** field is the sum index field.
2. Add fields to table 123456700 Seminar as follows.

No.	Field Name	Type	Length	Comment
20	Date Filter	Date		FlowFilter
21	Charge Type Filter	Option		FlowFilter; Options: Instructor,Room, Participant,Charge
25	Total Price	Decimal		FlowField; see step 3 below for the CalcFormula. Must not be editable. AutoFormatType=1
26	Total Price (Not Chargeable)	Decimal		FlowField; see step 4 below for the CalcFormula. Must not be editable. AutoFormatType=1
27	Total Price (Chargeable)	Decimal		FlowField; see step 5 below for the CalcFormula. Must not be editable. AutoFormatType=1

3. Set the CalcFormula for the **Total Price** field so that it calculates the sum of the **Total Price** field on the Seminar Ledger Entry table for the records where the **Seminar No.** corresponds to the **No.** field, where the **Posting Date** corresponds to the Date Filter and where the **Charge Type** corresponds to the Charge Type Filter.
4. Set the CalcFormula for the **Total Price (Not Chargeable)** field so that it calculates the sum of the **Total Price** field on the Seminar Ledger Entry table for the records where the **Seminar No.** corresponds to the **No.** field, where the **Posting Date** corresponds to the Date Filter, where the **Charge Type** corresponds to the Charge Type Filter and where the records are not Chargeable.
5. Set the CalcFormula for the **Total Price (Chargeable)** field so that it calculates the sum of the **Total Price** field on the Seminar Ledger Entry table for the records where the **Seminar No.** corresponds to the **No.** field, where the **Posting Date** corresponds to the Date Filter, where the **Charge Type** corresponds to the Charge Type Filter and where the records are Chargeable.

Exercise 20 – Creating the Seminar Statistics Form

1. Create form 123456714 Seminar Statistics, based on the Seminar table. Don't add any fields to this form yet.
2. Define the following global variables for the form:

Name	Data Type	Subtype	Length
DateFilterCalc	Codeunit	DateFilter-Calc	
SemDateFilter	Text		30
SemDateName	Text		30
CurrentDate	Date		
TotalPrice	Decimal		
TotalPriceNotChargeable	Decimal		
TotalPriceChargeable	Decimal		
i	Integer		

3. Set the property for all the variables except DateFilterCalc and CurrentDate so that each variable is an array of four dimensions.
4. Set the property for the form so that it is not editable.
5. Enter code in the appropriate trigger so that after the form gets the record, the program performs the following tasks:
 - Filters the table to the selected seminar.

- If the `CurrentDate` is not the work date, the program sets the `CurrentDate` variable to the work date. The program runs the `CreateAccountingPeriodFilter` function of the `DateFilter-Calc` codeunit with parameters of the first dimensions of the `SemDateFilter` and `SemDateName`, the `CurrentDate` and 0. It runs the `CreateFiscalYearFilter` function of the `DateFilter-Calc` codeunit with parameters of the second dimensions of the `SemDateFilter` and `SemDateName`, the `CurrentDate` and 0. The program then runs the `CreateFiscalYearFilter` function of the `DateFilter-Calc` codeunit with parameters of the third dimensions of the `SemDateFilter` and `SemDateName`, the `CurrentDate` and `i`.

HINT: There is similar code on the Customer Statistics form.

- For each of the dimensions, the program filters the table to records where the Date Filter corresponds to the value in the appropriate dimension of the `SemDateFilter` and calculates the **Total Price**, **Total Price (Not Chargeable)** and **Total Price (Chargeable)** fields. The program then sets the value for the appropriate dimension of the `TotalPrice`, `TotalPriceNotChargeable` and `TotalPriceChargeable` to the values in the corresponding fields.
 - Filters the table to those records where the Date Filter is before the `CurrentDate`.
6. Place the text labels for the **Total Price**, **Total Price (Chargeable)**, **Total Price (Not Chargeable)**, **This Year**, **Last Year** and **To Date** on the form as shown in the GUI design.
 7. Place a text box for the "month" label as shown in the GUI design. Set the property to specify that the source of the text box is the first dimension of the `SemDateName` variable. Set the `Border` and `Focusable` properties to `No`.
 8. Place four text boxes in the **Total Price** row. Set the properties to specify that the sources of the text boxes are the respective dimensions of the `TotalPrice` variable.
 9. Place four text boxes in the **Total Price (Chargeable)** row. Set the properties to specify that the sources of the text boxes are the respective dimensions of the `TotalPriceChargeable` variable.
 10. Place four text boxes in the **Total Price (Not Chargeable)** row. Set the properties to specify that the sources of the text boxes are the respective dimensions of the `TotalPriceNotChargeable` variable.

We have finished the Seminar Statistics form, so our last tasks are to make the form accessible from the two seminar forms.

11. Add the following menu item to form 123456700 Seminar Card between the Comments and the Extended Texts menu items.

Menu Button	Options	Comment
	<Separator>	
	Statistics (F9)	Opens form 123456714 Seminar Statistics for the selected seminar. The link should be run whenever the form is updated.
	<Separator>	

12. Add the same Statistics menu item to form 123456701 Seminar List as above.

Managing Statistics – Testing

In order to test the statistics form, you need to have entries in the Seminar Ledger Entry table with the related posted seminar registrations.

1. Open the Seminar Card and view a seminar that has some posted registrations. You can verify that the seminar has ledger entries by selecting SEMINAR→ENTRIES→LEDGER ENTRIES from the Seminar Card.
2. Select SEMINAR→STATISTICS to open the Seminar Statistics window. Verify that the totals shown are correct for the different rows and columns. You may find it convenient to do so by applying a table filter to the Seminar Ledger Entries window and comparing the results with the amounts shown in the Seminar Statistics window.

Test Your Knowledge

Review Questions

1. What is the purpose of a FlowFilter field?
2. How are FlowFilters used in calculations of FlowFields?
3. What are the advantages of using FlowFields to make calculations?
4. What is the Microsoft Navision standard shortcut for opening a Statistics form?

Conclusion

Chapter Summary

In this chapter, you created a statistics form for seminars to sum up the total price in four different time periods. You also made this form available from the seminar forms.

Positioning – Where do you go from here?

You are now ready to add dimensions to the existing features in our module.

Quick Interaction: Lessons Learned

Take a moment to write down three Key Points you have learned from this chapter:

1.

2.

3.

CHAPTER 8: MANAGING DIMENSIONS

This chapter contains the following sections:

- Introduction
- Dimensions
- Using Microsoft® Business Solutions–Navision® Developer's Toolkit
- Test Your Skills
 - Diagnosis
 - Managing Dimensions in Master Files
 - Managing Dimensions in Registration
 - Managing Dimensions in Seminar Posting
 - Managing Dimensions in Invoicing
- Test Your Knowledge
- Conclusion

Microsoft Internal Use Only

Introduction

Positioning – What is the Starting Point?

You have created a functioning seminar module with:

- Master tables and forms
- Registration functionality
- Posting routines
- Reports
- Statistics

You are now ready to add the standard dimensions functionality to the module.

Preconditions

To add dimensions functionality to the seminar module, you must have master tables and forms, tables and forms for seminar registration, and the posting routine to post seminar registrations.

Further preconditions are knowledge of the following areas:

- Writing internal documentation
- Exporting and importing objects
- Working with event triggers
- Working with complex data types and their member functions
- Creating journal and document posting routines
- Debugging code
- Programming for low-impact on the application
- Using report event triggers
- Creating processing-only reports

Business Goals

In this story, your goal is to improve data analysis capabilities in the seminar module by adding dimensions.

Educational Goals

By completing this story, you will have learned or reacquainted yourself with the following:

- Working with dimensions

Dimensions

Dimensions are data you add to an entry to act as a marker for the program, which can group entries with similar characteristics for analysis purposes. Every entry in the program can have dimensions: master files, transaction document headers and lines, journal lines, ledger entries, and posted documents and their lines.

We use the term "dimension" to describe how analysis occurs. A two-dimensional analysis, for example, could be analysis of sales for an area. However, by using more than two dimensions when creating an entry, you can carry out a more complex analysis later, for example, sales for each sales campaign for each customer group in each area.

Each dimension can have an unlimited number of dimension values. For example, a dimension called Department can have dimension values of Sales, Administration, Purchasing, and so on. The user defines and tailors these dimensions and values to every company's needs.

There are three types of dimensions:

Global: When you set up dimensions in the G/L Setup, you can set up two of them to be global dimensions. You can use this dimension throughout the program as filters for G/L Entries and on reports, account schedules and batch jobs.

Shortcut: You can enter shortcut dimensions on journal and document lines. These lines have eight fields that are designated for dimensions. The first two are always the global dimensions, but the remaining six can be selected from those set up as shortcut dimensions in the G/L Setup. You can also specify dimensions that are not set up as shortcut dimensions, but these must be set up in a separate Dimensions window for the header or line.

Budget: You can define four dimensions for each budget.

When you set up a global, shortcut or budget dimension, the program automatically renames all fields that use the dimension type with the code caption that you specify in the dimension setup.

Where dimensions are stored depends on the type of entry. The following table shows different tables that contain dimensions with the types of entries with which they are associated.

Dimension Table	Type of Entry
352 Default Dimension	Master file records
355 Ledger Entry Dimension	Ledger entries
356 Journal Line Dimension	Journal lines
357 Document Dimension	Document headers and lines
358 Production Document Dimension	Production orders, lines and components
359 Posted Document Dimension	Posted document headers and lines
361 G/L Budget Dimension	Budget entries

Code Walkthrough – Dimension Management Codeunit

Before you begin development, take a look at how the DimensionManagement codeunit works.

Codeunit 408 – Dimension Management contains functions that help populate the Dimension tables.

Look at the "TypeToTableID" functions (TypeToTableID1, TypeToTableID2, etc.). These functions take in one option parameter and then return the table ID of the table related to that parameter (if "G/L Account" was passed in, for example, the function would return table ID 15). These functions are used, through a series of other functions, to populate the dimension tables where the table ID is needed. Without them, we would have to write a similar case statement every time we needed to get the table ID of an option field. We add our own TypeToTableID function for table 95000 Seminar to return the table according value of the Type option.

Now look at the SetupObjectNoList function. This function sets a temporary record object with a list of tables that use dimensions. You can see this function used in Table 352, in the OnLookup trigger of the **Table ID** field. Modify SetupObjectNoList to include the master tables.

Other functions of note in this codeunit are ValidateDimValueCode and SaveDefaultDim. ValidateDimValueCode takes in two parameters, the field number and the dimension value. The function checks to see that the dimension is valid, and returns an error if it is not. SaveDefaultDim saves the dimension in the default dimension table. If these functions were not available, then you would need to create these functions in every table necessary.

You will be using a large number of the DimensionManagement functions in adding Dimensions functionality to the Seminar module. In general, there are functions that correspond to each type of Dimension table for getting default dimensions, updating, saving, deleting, inserting, and validating consistency and combinations of dimensions.

Using Navision Developer's Toolkit

When developing in Microsoft Navision, you often need information such as where and how often a certain table or field is used. This kind of analysis is not possible in the Object Designer, so we have the Microsoft Navision Developer's Toolkit to aid in analyzing databases, as well as comparing and merging databases.

In the Developer's Toolkit, you can import objects and analyze them by searching for certain words or phrases, finding out where a certain object, field or key is used, or finding the relations between objects. You can also compare two or three versions of a database to see the differences.

The Developer's Toolkit works with the text version of a Microsoft Navision database or object by parsing the different elements and storing the results in its own database. The Developer's Toolkit is read-only, although you can export objects and databases as text files. The database that the Toolkit uses is a Navision C/SIDE database, which is not the same as a Microsoft Navision database. To read from and write to the database, the Developer's Toolkit uses C/FRONT[®], which is the toolkit that makes it possible to develop applications in the C programming language that access a C/SIDE database. There is more information about C/FRONT in the appendix of this manual.

With Developer's Toolkit you can also directly access one or more Navision clients that are open on your computer. Through this access, you can quickly import objects, update from or to the client and export objects using an export worksheet.

Some of the features of the Developer's Toolkit include:

- The ability to view object relationships in tree or diagram form
- The ability to search the database for a word or phrase using Source Finder
- The ability to compare and merge databases or simply to compare two versions of a database
- The ability to list all a specific object's relations to or from other objects
- The ability to list all the places where a specific object (or an element of an object such as a field, key, trigger or procedure) is used
- The ability to view the method flow in C/AL

Test Your Skills – Managing Dimensions – Diagnosis

Description

The seminar module is missing a standard feature: dimensions.

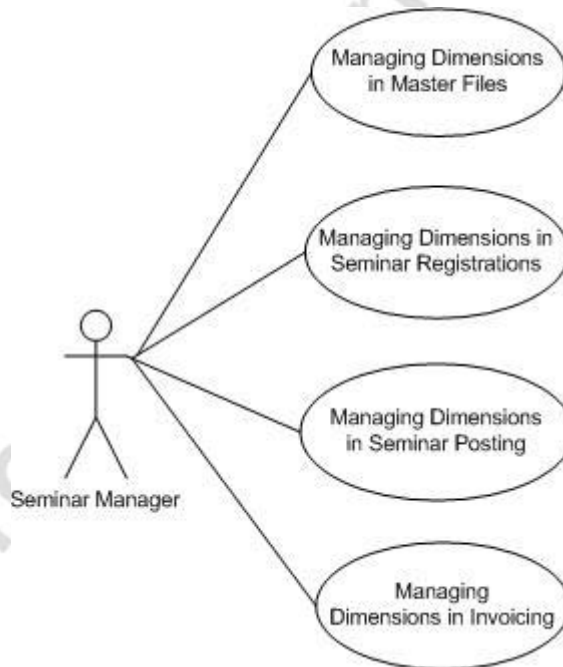
Your task for this story is to add dimensions functionality to the seminar module. This requires making dimensions available from master files, seminar registrations, seminar posting, and invoicing.

Use Cases

We can divide the analysis and implementation phases of this story into the following use cases:

- Managing Dimensions in Master Files
- Managing Dimensions in Seminar Registration
- Managing Dimensions in Seminar Posting
- Managing Dimensions in Invoicing

The following use case diagram illustrates how the use cases relate to one another:



Implementation of Use Case 1 – Managing Dimensions in Master Files

Managing Dimensions in Master Files – Analysis

In accordance with Microsoft Navision standards, the seminar master files should be associated with default dimensions. The master file dimensions then flow to the transactions in which the master data is used, and these document dimensions eventually flow to the ledger entries and posted document dimensions.

Define the following dimensions for seminars, rooms and instructors.

Purpose

The purpose of managing dimensions on the master file level is to enable analysis of master data.

Preconditions

Tables and forms for seminar master data must exist.

Postconditions

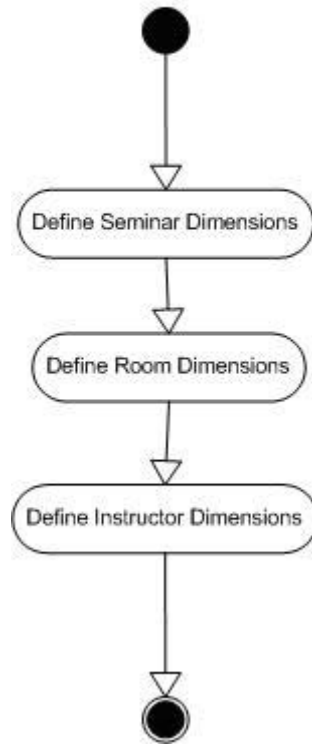
The result of this use case is that the user can define default dimensions for master data.

Main Scenario

When the seminar managers define master data, they will also define default dimensions for the data, as appropriate.

Activity Diagram

Seminar Manager



Managing Dimensions in Master Files – Design

To enable dimensions for master files, you need to make changes to the master tables and forms, as well as to the codeunits that manage dimensions.

GUI Design

Add menu items for dimensions to the Seminar Card (Form 123456700), Seminar Room Card (Form 123456703) and Instructors (Form 123456705) forms from which you define the master data.

Functional Design

Include dimension entries for master files into the Default Dimension table. We must make some modifications to this table and to the DimensionManagement codeunit so that they can accept entries from the Seminar, Seminar Room, and Instructor tables.

When the user creates a new record in a master table, the program should insert or update the Default Dimensions. Likewise, when the user deletes a record, the program should automatically delete the associated dimensions.

ValidateShortcutDimCode: When the user enters a value in a dimension field, we will use this function to validate the entry. The DimensionManagement codeunit has a function called ValidateDimValueCode that can help us do this.

Table Design

We will add dimension fields to the following tables:

- Table 123456700 Seminar
- Table 123456702 Seminar Room
- Table 123456703 Instructor
- We will modify the code in Table 352 Default Dimension so that the table can accept entries from the seminar master files.

Managing Dimensions in Master Files – Development

You can now begin development.

Exercise 21 – Modifying the DimensionManagement Codeunit

The first step in enabling dimensions for master files is to modify the DimensionManagement codeunit.

Open codeunit 408 DimensionManagement and look at the functions. Notice a number of generic functions that have parameters like TableID and DocType, so you can use them for any table. There are also functions like TypeToTableID2 that you can use for one specific table. We will begin modifying codeunit 408 by adding a table-specific function.

1. Create a function called TypeToTableID123456700 that takes a parameter called Type with a data type of Option and the options Resource, G/L Account. The function returns an integer.
2. Enter code in the function trigger so that the function returns the ID of the Resource table when the Type is Resource, and returns the ID of the G/L Account table when the Type is G/L Account.

Now modify the SetupObjectNoList function so that it takes into account the three tables Seminar, Seminar Room, and Instructor.

3. Change the Dimensions property of the TableIDArray variable from 20 to 23.
4. Comment out the existing code in the function trigger that sets the NumberOfObjects variable to 20 and add code so that the NumberOfObjects variable is set to 23. Set the values of TableIDArray[21]-[23] to correspond to the Seminar, Seminar Room, and Instructor tables.

Exercise 22 – Modifying the Tables and Forms for Dimensions in Master Files

We are now ready to modify the master file tables and forms to enable dimensions.

1. Add two new fields to table 123456700 Seminar as follows:

No.	Field Name	Type	Length	Comment
15	Global Dimension 1 Code	Code	20	Caption Class is 1,1,1. Relation to the Code field on the Dimension Value table where the Global Dimension No. is 1.
16	Global Dimension 2 Code	Code	20	Caption Class is 1,1,2. Relation to the Code field on the Dimension Value table where the Global Dimension No. is 2.

2. Create a new function for the table called ValidateShortcutDimCode. This function takes two parameters: an integer called FieldNumber and a code variable with a length of 20 called ShortcutDimCode, which is passed by reference.
3. Enter code in the function so that it runs the ValidateDimValueCode and SaveDefaultDim functions from the DimensionManagement codeunit and modifies the table.
4. Enter code in the appropriate triggers so that when the user enters or changes a value in the **Global Dimension Code 1** or **Global Dimension Code 2** fields, the program runs the ValidateShortcutDimCode function (the FieldNumber values are 1 and 2, respectively).
5. Enter code in the appropriate trigger so that when the user inserts a new record to the table, the program runs the UpdateDefaultDim function from the DimensionManagement codeunit.
6. Enter code in the appropriate trigger so that when the user deletes a record from the table, the program runs the DeleteDefaultDim function from the DimensionManagement codeunit.
7. Add the same global dimension fields to table 123456702 Seminar Room that you added to the Seminar table. Add these fields as the last two fields of the table. Add the same functions and code modifications as for the Seminar table.
8. Add the same global dimension fields to table 123456703 Instructor that you added to the Seminar and Seminar Room tables. Add these fields as the last two fields of the table. Add the same functions and code modifications as for the Seminar and Seminar Room tables.

When you add dimensions to any master file in Microsoft Navision, the dimensions are stored in table 352 Default Dimension. We will modify this table so that it can accept entries from the seminar master files.

9. Enter code in the UpdateGlobalDimCode function trigger in the Default Dimension table for each of the three master tables, Seminar, Seminar Room and Instructor, so that, depending on the TableID, the program gets the appropriate record from the appropriate table, sets the appropriate **Global Dimension Code** field to the NewDimValue and modifies the table.
10. Add the menu items for Dimensions to the Seminar Card and Seminar Room Card forms as follows:

Menu Button	Options	Comment
Seminar	Dimensions (SHIFT + CTRL + D)	Opens the form 540 Default Dimensions for the selected entry. The link should run whenever you update the form.

11. Add the menu button and menu items for Dimensions to the Instructors form as follows:

Menu Button	Options	Comment
Instructor	Dimensions (SHIFT + CTRL + D)	Opens the form 540 Default Dimensions for the selected entry. The link should run whenever you update the form.

Implementation of Use Case 2 – Managing Dimensions in Registration

Managing Dimensions in Registration – Analysis

The next step in enabling dimensions functionality for the seminar module is to set up the seminar registrations and posted registrations to accept dimensions as well.

Purpose

The purpose of managing dimensions on the registration tables and forms is so that users can perform analyses on the registration data using the dimensions to group the information properly.

Preconditions

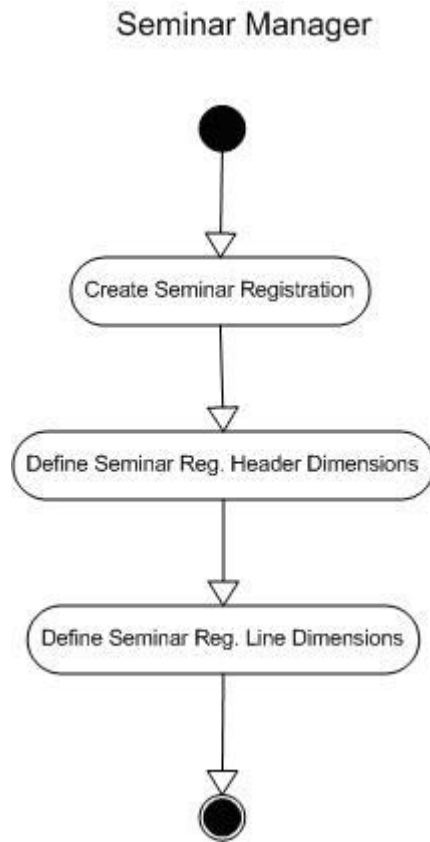
Tables and forms for seminar registrations must exist. Tables and forms for the posted seminar registrations must exist. Tables, forms and codeunits for the dimension functionality must exist.

Postconditions

The result of this use case is that the user will be able to define dimensions for seminar registration headers and lines, which then flow to the dimensions for the posted registration entries upon posting.

Main Scenario

When creating a seminar registration, the seminar managers enter dimension information for the header and lines. After posting the registration, the seminar managers can enter or change dimension information on the posted registration header and lines.

Activity Diagram**Managing Dimensions in Registration – Design**

As was described in the Diagnosis section, transaction documents and lines use shortcut dimension codes. Enabling dimensions in transaction documents means setting up the shortcut dimension codes for the header and lines, and writing code so that the dimensions from the seminar, seminar room and instructor chosen for the seminar registration are also included. You will also enable dimensions on the posted seminar registration tables and forms.

GUI Design

Add fields and menu items for dimensions to the forms where you define the master data.

Seminar Registration Form (Form 123456710) and Seminar Registration Subform (Form 123456711): Add the eight shortcut dimension code fields to the subform as shown in the following screenshot, starting with the **Department Code** field. These fields should not be visible until the user selects them using Show Column.

..	Amount	Departm...	Project Code	Custom...	Area Code	Busines...	Salesca...	Shortcut...	Shortcut...
▶	500,00	SALES			30				
	500,00	SALES			30				
	500,00	SALES			40				
	500,00	SALES			70				

Posted Seminar Registration (Form 123456734) and Posted Seminar Reg. Subform (Form 123456735): Add two shortcut dimension code fields to the subform, which are shown in the following screenshot as the **Project Code** and **Department Code** fields. These fields should not be visible until the user selects them using Show Column.

	To Invoice	Registered	Seminar Price	Line ...	Line Discount...	Amount	Project Code	Departm...
▶	✓		500,00			500,00		
	✓		500,00			500,00		
	✓		500,00			500,00		
	✓		500,00			500,00		
	✓		500,00			500,00		

Functional Design

You need functions in the Seminar Registration Header and Seminar Registration Line tables to get default dimensions, validate dimensions, insert dimensions and delete dimensions. We can use standard functions from the DimensionManagement codeunit to do most of the real work, but we will need functions to call these standard functions.

ValidateShortcutDimCode: As with the master files, we will create this function to validate and save the dimensions, using the SaveDefaultDim function from the DimensionManagement codeunit.

CreateDim: For each of the tables mentioned previously, this function should get the default dimensions for certain key dimension fields by running the GetDefaultDim function from the DimensionManagement codeunit. How many and for which fields the function retrieves default dimensions depends on the table. For the Seminar Registration Header, these fields will be **Seminar Code**, **Instructor Code**, **Room Code** and **Job No.** For the Seminar Registration Line, there will be only one field: **Bill-To Customer No.**

Table Design

Add new fields for dimensions to the following tables:

- Table 123456710 Seminar Registration Header
- Table 123456711 Seminar Registration Line
- Table 123456718 Posted Seminar Reg. Header
- Table 123456719 Posted Seminar Reg. Line

Modify the code in Table 357 Document Dimension so that the table can accept entries from the seminar registrations.

Managing Dimensions in Registration – Development

Exercise 23 – Modifying the Tables for Dimensions in Seminar Registrations

Begin by adding the necessary fields and code to the seminar registration table.

1. Add the shortcut dimension fields to table 123456710 Seminar Registration Header as follows:

No.	Field Name	Type	Length	Comments
35	Shortcut Dimension 1 Code	Code	20	Caption Class is 1,2,1. Relation to the Code field on the Dimension Value table, where the Global Dimension No. is 1.
36	Shortcut Dimension 2 Code	Code	20	Caption Class is 1,2,2. Relation to the Code field on the Dimension Value table, where the Global Dimension No. is 2.

2. Create a new function for the table called ValidateShortcutDimCode. This function takes two parameters: an integer called FieldNumber and a code variable with a length of 20 called ShortcutDimCode, which is passed by reference.
3. Enter code in the function trigger so that it runs the ValidateDimValueCode function from the DimensionManagement codeunit. If the **No.** is not blank, the function runs the SaveDocDim function from the DimensionManagement codeunit; otherwise, it runs the SaveTempDim function.
4. Enter code in the appropriate triggers so that when the user enters or changes a value in the new **Shortcut Dimension Code** fields, the program runs the ValidateShortcutDimCode function (the FieldNumber values are 1 and 2, respectively).

For seminar registrations, get the default dimensions from the Seminar, Job, Seminar Room and Instructor that the user assigns to the registration and assign these to the registration's Document Dimensions. Create a CreateDim function to do this for us.

5. Create a new function called CreateDim in the Seminar Registration Header table. This function takes eight parameters: four integers called Type1, Type2, Type3 and Type 4, and four code variables of length 20 called No1, No2, No3 and No4. The order of the parameters is Type1, No1, Type2, No2, and so forth.

6. In C/AL Locals, define an integer array variable named TableID with 10 dimensions and a code array variable named No of length 20 with dimensions. You will only use 4 of the 10 dimensions but you must specify 10 due to the parameters of the GetDefaultDim function in the Dimension Management codeunit.
7. Enter code in the CreateDim function trigger so that the function assigns the Type and No parameters to corresponding dimensions of the TableID and No variables. The function then clears both **Shortcut Dimension Code** fields and runs the GetDefaultDim function from the DimensionManagement codeunit. If the **No.** is not blank, the function runs the UpdateDocDefaultDim function of the DimensionManagement codeunit.

Now use the CreateDim function to assign the default dimensions from the Seminar, Job, Seminar Room and Instructor whenever you enter or change the default dimensions.

8. Enter code in the appropriate triggers so that when the user changes the **Seminar Code, Instructor Code, Room Code** or **Job No.** fields, the program runs the GetDimensions function of the Document Dimension table, runs the CreateDim function for the **Seminar Code, Instructor Code, Room Code** and **Job No.**, and runs the UpdateAllLineDim function from the Document Dimension table.

Next, add code to ensure that when the user creates or deletes a seminar registration, the program automatically creates or deletes dimensions as appropriate.

9. Enter code in the appropriate trigger so that when the user creates a new Seminar Registration Header, the program runs the InsertDocDim function from the DimensionManagement codeunit.
10. Enter code in the appropriate trigger so that when the user deletes a Seminar Registration Header, the program deletes all associated dimensions by running the DeleteDocDim function from the DimensionManagement codeunit.

The next modifications will be to the Seminar Registration Line table.

11. Add the shortcut dimension fields to table 123456711 Seminar Registration Line as follows:

No.	Field Name	Type	Length	Comments
15	Shortcut Dimension 1 Code	Code	20	Caption Class is 1,2,1. Relation to the Code field on the Dimension Value table, where the Global Dimension No. is 1.
16	Shortcut Dimension 2 Code	Code	20	Caption Class is 1,2,2. Relation to the Code field on the Dimension Value table, where the Global Dimension No. is 2.

12. Create a new function for table 123456711 called ValidateShortcutDimCode. As you did for the ValidateShortcutDimCode function in the Seminar Registration Header, add parameters and code so that this function validates and saves the document dimensions for the dimension field passed to it.
13. Enter code in the appropriate triggers so that when the user enters or changes a value in the **Shortcut Dimension Code** fields, the program runs the ValidateShortcutDim function (the FieldNumber values are 1 and 2, respectively).

For seminar registration lines, get the default dimensions from the Seminar Registration Header. Create a CreateDim function to do this for you.

14. Create a new function called CreateDim in the Seminar Registration Line table. As mentioned in the functional design, the Seminar Registration Line table has one key dimension field from which we must get default dimensions: the **Bill-To Customer No.** The CreateDim function will be similar to the CreateDim function in the Seminar Registration Header, except that it takes only two parameters: an integer called Type1 and a code variable of length 20 called No1.
15. Enter code in the function trigger so that the function assigns the parameters to a corresponding local integer variable and local code variable. The function clears both **Shortcut Dimension Code** fields and runs the GetPreviousDefaultDocDim and GetDefaultDim functions from the DimensionManagement codeunit. If the **Line No.** is not 0, the function runs the UpdateDocDefaultDim function of the DimensionManagement codeunit.
16. Enter code in the appropriate trigger so that when the user enters or changes the **Bill-To Customer No.**, the program runs the CreateDim function, passing the Customer table number as the first parameter.

Now create two functions, LookupShortcutDimCode and ShowShortcutDimCode, which shows information to the user.

17. Create a new function called LookupShortcutDimCode that takes two parameters: an integer called FieldNumber and a code variable of length 20 called ShortcutDimCode, which is passed by reference.
18. Enter code in the LookupShortcutDimCode function trigger so that the function runs the LookupDimValueCode function from the DimensionManagement codeunit. If the **Line No.** of the record is not 0. The function runs the SaveDocDim function from the DimensionManagement codeunit; otherwise, it runs the SaveTempDim function of the DimensionManagement codeunit.
19. Create a new function called ShowShortcutDimCode that takes one parameter: a code variable of length 20 that is passed by reference, called ShortcutDimCode. This parameter is an array of 8 dimensions.
20. Enter code in the ShowShortcutDimCode function trigger so that if the **Line No.** of the record is not 0, the function runs the ShowDocDim function of the DimensionManagement codeunit; otherwise, it runs the ShowTempDim function of the DimensionManagement codeunit.

You need a function that will display the line dimensions when requested by the user.

21. Create a new function called ShowDimensions.
22. Enter code in the ShowDimensions function trigger so that the function makes sure there is a **Document No.** and **Line No.** and then shows the corresponding lines from the Document Dimension table in the Document Dimensions form.

***HINT:** After you have filtered the Document Dimension table to the appropriate records, use the SETTABLEVIEW function for the Document Dimensions form before running it.*

As in the Seminar Registration Header table, you need to ensure that when the user creates or deletes a Seminar Registration Line, the program automatically handles the associated dimensions appropriately.

23. Enter code in the appropriate triggers so that when the user creates a new Seminar Registration Line, the program runs the InsertDocDim function from the DimensionManagement codeunit, and when the user deletes a record, the program runs the DeleteDocDim function from the DimensionManagement codeunit.

Lastly, before you can finish with the tables, you must modify the Document Dimension table so that it can accept dimensions from the seminar registrations.

24. Define local record variables for the UpdateGlobalDim function of table 357 Document Dimension, for the Seminar Registration Header and Seminar Registration Line tables. Set the IDs for these variables to 123456700 and 123456701, respectively.
25. Add to the existing code in the UpdateGlobalDim function trigger. Do this so that for each of the three record variables you just added, (if the TableID corresponds to the table ID for that table), the program gets the corresponding record. This sets either the Shortcut Dimension 1 Code or the Shortcut Dimension 2 Code (depending on the GlobalDimCodeNo value passed to the function) to the NewDimValue, and modifies the record.

Modify the UpdateLineDim function so that if the user changes a dimension on the Seminar Registration Header, the function changes the dimensions on any associated Seminar Registration Line.

26. In table 357 Document Dimension, modify the UpdateLineDim function so that if the Table ID is for the Seminar Registration Header, the function filters the document dimensions to those that have a Table ID for the Seminar Registration Line table. The function then looks for any associated Seminar Registration Line records. If it finds any, and the user wants to update those lines, the function deletes the old dimensions and inserts the new updated Document Dimension record that was passed to it, using the InsertNew function.

***HINT:** Look at how the function handles the case where the **Table ID** is Sales Header or Purchase Header.*

27. Modify the UpdateAllLineDim function in a similar way so that if the **TableNo** passed to it is for the Seminar Registration Header, the function updates the dimensions for associated Seminar Registration Line records.

Now that the seminar registration tables are ready for dimensions, you must enable dimensions on the forms.

Exercise 24 – Modifying the Forms for Dimensions in Seminar Registrations

Define functions in the forms to make it possible to show all the dimensions for a registration header or line, rather than just those defined as the global or shortcut dimensions on the tables.

1. In form 123456711 Seminar Registration Subform, create a global code variable `ShortcutDimCode` with a length of 20. Define this variable as an array of 8 dimensions.
2. Define a new function for the form called `ShowDimensions` for the form. Enter code in the function trigger so that this function simply runs the `ShowDimensions` function for the current record.
3. Define a new function called `UpdateForm` that takes a parameter of a Boolean variable called `SetSaveRecord`. Enter code in the function trigger so that this function simply updates the current form, running the form update trigger code if `SetSaveRecord` is `TRUE`.
4. Enter code in the appropriate triggers so that the program performs the following:
 - When the program gets a record for the form or the user enters or changes the **Bill-To Customer No.**, the program updates the `ShortcutDimCode` variable with the document dimensions by calling the `ShowShortcutDimCode` function.
 - After the program gets the record, it updates the controls on the form.
 - When the user goes to a new record, it clears the `ShortcutDimCode` variable.
5. Add the eight **Shortcut Dimension Code** fields to the form as shown in the GUI design. First, add the **Shortcut Dimension 1 Code** and **Shortcut Dimension 2 Code** fields to the form and then add six text boxes. The caption classes for these six text boxes are '1,2,x', where x is the respective number of the field (three through to eight). The sources for these fields are the respective dimensions of the `ShortcutDimCode` variable (three through to eight).
 - For each of these fields, enter code so that when the user enters or changes a value, the program runs the `ValidateShortcutDimCode` function for the corresponding `ShortcutDimCode` dimension value.
 - For each of these fields, enter code so that when the user performs a lookup on the field, the program runs the `LookupShortcutDimCode` function for the corresponding `ShortcutDimCode` dimension value.

6. In form 123456710 Seminar Registration, set the property so that the form is updated when it is activated.
7. Enter code in the appropriate trigger so that the form is updated when the user enters or changes a value in the **Seminar Code**, **Instructor Code**, **Room Code** or **Job No.** fields.
8. Add the menu button and menu items to form 123456710 Seminar Registration as shown below. Add code to the appropriate trigger so that when the user selects the Dimensions menu item, the program runs the ShowDimensions function of the subform.

Menu Button	Options	Comments
Registration	Dimensions	Opens the form 546 Document Dimensions for selected No., where the Line No. is 0. The link should run whenever the form is updated.
Line	Dimensions (CTRL + SHIFT + D)	Runs code in the OnPush trigger to show the line.

Now modify the posted registration tables and forms to enable dimensions. The user is not able to enter or change dimensions for posted documents, so you simply make the posted dimensions available for the user to view.

Exercise 25 – Modifying the Tables and Forms for Dimensions in Posted Seminar Registrations

1. Add the shortcut dimension fields to Table 123456718 Posted Seminar Reg. Header as follows:

No.	Field Name	Type	Length	Comments
35	Shortcut Dimension 1 Code	Code	20	Caption Class is 1,2,1. Relation to the Code field on the Dimension Value table, where the Global Dimension No. is 1.
36	Shortcut Dimension 2 Code	Code	20	Caption Class is 1,2,2. Relation to the Code field on the Dimension Value table, where the Global Dimension No. is 2.

2. Enter code in the appropriate trigger so that when the user deletes a Posted Seminar Reg. Header, the program deletes the associated posted document dimensions by calling the DeletePostedDocDim function from the DimensionManagement codeunit.
3. Add the shortcut dimension fields to Table 123456719 Posted Seminar Reg. Line as follows:

No.	Field Name	Type	Length	Comments
15	Shortcut Dimension 1 Code	Code	20	Caption Class is 1,2,1. Relation to the Code field on the Dimension Value table, where the Global Dimension No. is 1.
16	Shortcut Dimension 2 Code	Code	20	Caption Class is 1,2,2. Relation to the Code field on the Dimension Value table, where the Global Dimension No. is 2.

4. Create a new function called ShowDimensions. Enter code in the function trigger so that the function tests that a **Document No.** and **Line No.** exist. If so, the function filters the Posted Document Dimension table to the records corresponding to the Posted Seminar Reg. Line and shows those records in the Posted Document Dimension form.
5. As you did for the Posted Seminar Reg. Header table, enter code in the appropriate trigger so that when the user deletes a Posted Seminar Reg. Line record, the program deletes the associated Posted Document Dimension records.

6. In form 123456735 Posted Seminar Reg. Subform, create a function called ShowDimensions. Enter code in the function trigger so that it runs the ShowDimensions function of the current record.
7. Add the text boxes and labels for the new **Shortcut Dimension Code** fields as shown in the GUI design.
8. In form 123456734 Posted Seminar Registration, set the property to specify that the form should be updated when activated.
9. Add the new Dimensions menu item to the Registration menu button and the new Line menu button as follows:

Menu Button	Options	Comments
Registration	Dimensions	Opens the form 547 Posted Document Dimensions for the selected No. , where the Line No. is 0. The link should run whenever the form is updated.
Line	Dimensions (CTRL + SHIFT + D)	Runs code in the OnPush trigger to show the line dimensions (runs the ShowDimensions function of the Posted Seminar Reg. Subform form).

Implementation of Use Case 3 – Managing Dimensions in Seminar Posting

Managing Dimensions in Seminar Posting – Analysis

You have added dimension functionality to seminar registration tables and forms and to posted seminar registration tables and forms. What you are missing now is a means of bringing dimensions from seminar registrations to posted seminar registrations. You also need dimensions in the ledger entries. Therefore, you must now add and modify code in the seminar posting process to make this happen.

Purpose

The purpose of managing dimensions in the seminar posting process is to transfer the dimensions for seminar registrations to the posted seminar registrations.

Preconditions

The tables and forms for seminar registrations and posted seminar registrations must exist and must be set up for dimensions. The seminar journal lines and ledger entries and the seminar posting process must exist.

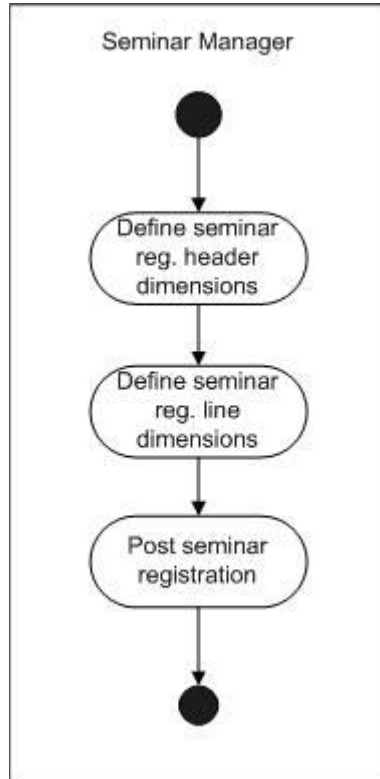
Postconditions

The result of this use case is a seminar posting process that transfers dimensions from seminar registrations to posted seminar registrations and carries the global dimensions to the seminar ledger entries.

Main Scenario

When the seminar managers post seminar registrations, the dimensions defined for those registrations are transferred automatically to the posted seminar registrations by the posting process. Furthermore, the global dimensions are transferred to the seminar ledger entries.

Activity Diagram



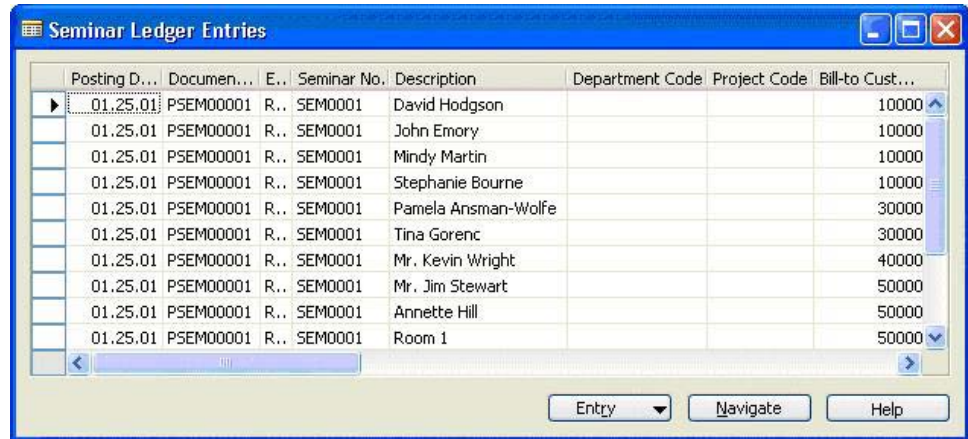
Managing Dimensions in Seminar Posting – Design

To enable the transfer of dimensions in the posting process, you add dimension fields to the journal and ledger entry tables. Then add the code to transfer the dimensions from seminar registration headers and lines to the posted registration headers and lines.

GUI Design

The only modifications to the user interface will be to the Seminar Ledger Entries form.

Seminar Ledger Entries (Form 123456721): To accommodate dimensions, add global dimension code fields. These fields appear in the following screenshot as the **Department Code** and **Project Code** fields.



Functional Design

In the Seminar Jnl.-Check Line codeunit, add code to validate the dimensions in the Seminar Journal Line.

In the Seminar Jnl.-Post Line codeunit, add code to transfer the journal line dimensions to the ledger entry dimensions.

In the Seminar-Post codeunit, add code to transfer the dimensions from the Document Dimension records to the **Posted Document Dimension**. For the Seminar Registration Header, the program should move the **Seminar Code**, **Job No.**, **Room Code** and **Instructor Code** dimensions. For the **Seminar Registration Line**, the program should move the **Bill-To Customer No.** dimension.

Table Design

Add fields for Shortcut Dimensions to Table 123456731 Seminar Journal Line.

Add fields for Shortcut Dimensions to Table 123456732 Seminar Ledger Entry.

Managing Dimensions in Seminar Posting – Development

Exercise 26 – Modifying the Tables, Codeunits and Forms for Dimensions in Seminar Posting

1. Add the following fields to Table 123456731 Seminar Journal Line:

No.	Field Name	Type	Length	Comments
25	Shortcut Dimension 1 Code	Code	20	Caption Class is 1,2,1. Relation to the Code field on the Dimension Value table, where the Global Dimension No. is 1.
26	Shortcut Dimension 2 Code	Code	20	Caption Class is 1,2,2. Relation to the Code field on the Dimension Value table, where the Global Dimension No. is 2.

2. Add the following fields to Table 123456732 Seminar Ledger Entry:

No.	Field Name	Type	Length	Comments
31	Global Dimension 1 Code	Code	20	Caption Class is 1,2,1. Relation to the Code field on the Dimension Value table, where the Global Dimension No. is 1.
32	Global Dimension 2 Code	Code	20	Caption Class is 1,2,2. Relation to the Code field on the Dimension Value table, where the Global Dimension No. is 2.

Now that you have the necessary fields, make the changes to the three main seminar posting codeunits. The first changes you make will be to the Seminar Jnl.-Check Line codeunit.

Before checking the lines, copy the shortcut dimension values from the **Seminar Journal Line** to a Journal Line Dimension record.

3. Delete the code in the OnRun trigger of codeunit 123456731 Seminar Jnl.-Check Line.

4. Enter code in the OnRun trigger so that if the Shortcut Dimension 1 Code in the Seminar Journal Line record is not empty, the program copies the **Seminar Journal Line table ID** and **Line No.** to a temporary Journal Line Dimension record variable. The program copies the **Global Dimension 1 Code** value from the G/L Setup to the **Dimension Code** field and the **Shortcut Dimension 1 Code** value from the **Seminar Journal Line** to the **Dimension Value Code** on the Journal Line Dimension record. The program then inserts the Journal Line Dimension record. Enter code to do the same with a new Journal Line Dimension record for the **Shortcut Dimension 2 Code** field on the Seminar Journal Line.
5. Add code in the OnRun trigger to run the RunCheck function with two parameters: the current record and the Journal Line Dimension record just created.

You are now ready to make the changes to the RunCheck function to check the dimensions passed to it.

6. As you can see from the previous step, you must add a parameter to the RunCheck function. Add a parameter (passed by reference) of a record variable for the Journal Line Dimension table.
7. Enter code in the function trigger so that after the function checks the **Document Date**, it loads the table IDs of the Seminar, Job, Seminar Room, Instructor and Customer tables into a local integer array variable (with ten dimensions). The function then loads the **Seminar Code**, **Job No.**, **Room Code**, **Instructor Code** and **Bill-To Customer No.** values into the corresponding dimensions of a local code array variable (with ten dimensions). Using these two array variables, the function checks the dimensions by running the CheckJnlLineDimValuePosting function from the DimensionManagement codeunit. If this function returns false, the function shows an error. If the **Line No.** is not 0, this can be a specific error to help the user see exactly where the error occurred.

***HINT:** You can use the GetDimValuePostingErr function from the DimensionManagement codeunit to help create the error message.*

Now modify codeunit 123456732 Seminar Jnl.-Post Line to transfer the dimensions to the ledger entries.

8. In the OnRun trigger, enter the code you entered into the OnRun trigger of the Seminar Jnl.-Check Line codeunit. This code loads and inserts two Journal Line Dimension records with information from the **Shortcut Dimension Code** fields into a temporary global Journal Line Dimension record variable. At the end of the code, call the RunWithCheck function with two parameters: the current record and the temporary Journal Line Dimension record variable.

9. As you did in the Seminar Jnl.-Check Line codeunit, add a parameter (passed by reference) to the RunWithCheck function. The parameter is a record variable of the Journal Line Dimension table.
10. Enter code in the RunWithCheck function trigger so that after the function performs the first copy of the Seminar Journal Line record, the function resets and deletes all records in the original global Journal Line Dimension variable. The function then copies the records from the Journal Line Dimension parameter variable to the global Journal Line Dimension variable. It does so by calling the CopyJnlLineDimToJnlLineDim function of the DimensionManagement codeunit.
11. In the Code function trigger, change the call of the RunCheck function of the Seminar Jnl.-Check Line codeunit so that it calls the function with the two parameters instead of only one.
12. In the Code function trigger, you want to ensure that the journal line dimensions are moved to the new ledger entries' dimensions. The DimensionManagement codeunit makes this easy with a function called MoveJnlLineDimToLedgEntryDim. Enter code to call this function after the Seminar Ledger Entry record is inserted.

Now add code to codeunit 123456700 Seminar-Post to move the document dimensions to the posted document dimensions.

13. Create a new local function called CheckDimValuePosting that takes four parameters: an integer variable called TableID and two record variables that are passed by reference for the Seminar Registration Header and Seminar Registration Line.
14. Enter code in the function trigger so that depending on the table ID sent to the function as the TableID parameter, the function will load an integer array variable (with ten dimensions) and a code array variable (of length 20 with ten dimensions) with the table IDs and field values of the key dimension fields for the table. (See the functional design for a list of the key dimension fields for the Seminar Registration Header and Seminar Registration Line tables.) In each case, the function calls the CheckDocDimValuePosting function of the DimensionManagement codeunit, and shows an error if it returns FALSE.
15. Create a new local function called CheckDimComb that takes two parameters: an integer called TableID and an integer called LineNo.
16. Enter code in the function trigger so that if the CheckDocDimComb function of the DimensionManagement codeunit returns FALSE when run with the document dimensions, the function will show one of three errors: one error if the **Line No.** is 0 or one of the other two errors depending on whether the TableID is for the Seminar Registration Header or Seminar Registration Line.

You now need a function that checks the document dimensions and passes them to a temporary TempDocDim variable.

17. Create a new local function called CopyAndCheckDocDimToTempDocDim.
18. Enter code in the function trigger so that the function copies the appropriate records from the Document Dimension table to the TempDocDim variable. The function runs the CheckDimComb function for the Seminar Registration Header table ID and **Line No.** 0. The function then runs the CheckDimValuePosting function. If there are associated Seminar Registration Line records, the function runs the CheckDimComb and CheckDimValuePosting functions for each of the records.
19. Enter code in the OnRun trigger so that after testing the **Resource No.** field of the Instructor record, the program runs the CopyAndCheckDocDimToTempDocDim function.
20. Enter code in the OnRun trigger so that after the Posted Seminar Reg. Header is inserted, the program copies the document dimensions to the posted document dimensions by calling the MoveOneDocDimToPostedDocDim function from the DimensionManagement codeunit for the Seminar Registration Header dimensions in the DocDim variable. Do the same for the Seminar Registration Line dimensions after the Posted Seminar Reg. Line record is inserted.

In addition to posting the document dimensions to the Seminar Ledger Entry records, you must also post the dimensions to the Job Ledger Entry records.

21. Enter code in the PostJobJnlLine function trigger so that the function performs the following tasks:
 - After copying the relevant fields to the Job Journal Line record when the **ChargeType** is Participant, the function copies the records for the corresponding Seminar Registration Line from the TempDocDim to a TempJnlLineDim variable. It does so by calling the CopyDocDimToJnlLineDim function from the DimensionManagement codeunit.
 - Further on in the function, instead of running the Job Jnl.-Post Line codeunit, the function should run the RunWithCheck function of the Job Jnl.-Post Line codeunit.

You now need to add similar code to the PostSeminarJnlLine function.

22. As done previously in the PostJobJnlLine function, enter code in the PostSeminarJnlLine function trigger so that depending on the ChargeType, the function copies the document dimensions for a corresponding record to a TempJnlLineDim variable, using the following guidelines:
 - For a ChargeType of Instructor or Room, the corresponding record is the Seminar Registration Header.
 - For a ChargeType of Participant, the corresponding record is the Seminar Registration Line.
 - In addition, instead of running the Seminar Jnl.-Post Line codeunit, the function runs the RunWithCheck function of the Seminar Jnl.-Post Line codeunit.

You are now finished modifying the codeunits. All that remains is to make the dimensions visible in the Seminar Ledger Entries form.

23. Add the new **Global Dimension Code** fields to form 123456721 Seminar Ledger Entries as shown in the GUI design.
24. Add the new Entry menu button and Dimensions menu item as follows:

Menu Button	Options	Comments
Entry	Dimensions (SHIFT+CTRL+D)	Opens the form 544 Ledger Entry Dimensions for the selected entry. The link should run whenever the form is updated.

Implementation of Use Case 4 – Managing Dimensions in Invoicing

Managing Dimensions in Invoicing – Analysis

When we create invoices for seminars, we must carry the associated dimensions to the invoices so that the invoice lines can be analyzed by dimension.

Purpose

The purpose of managing dimensions in the invoice posting process is to transfer the dimensions for seminar registrations to the sales invoices.

Preconditions

The tables and forms for seminar registrations and posted seminar registrations must exist and must be set up for dimensions. The invoice creation process must exist.

Postconditions

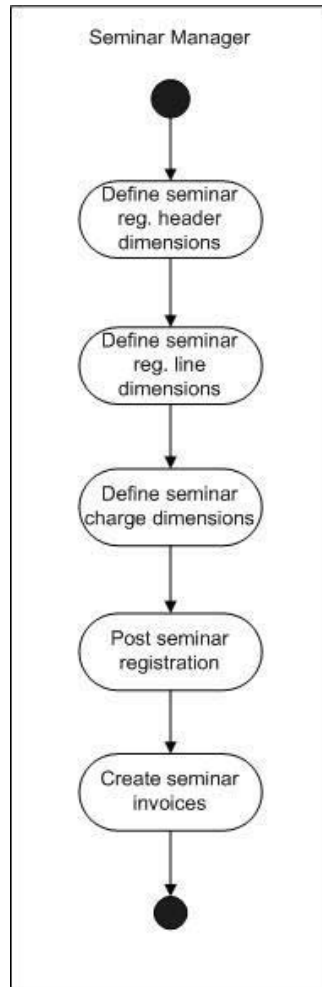
The result of this use case is an invoice creation and posting process that transfers dimensions from seminar registrations to the associated sales invoices.

Main Scenario

When the seminar managers create and post the sales invoices for seminar registrations, the invoice creation process automatically transfers the dimensions defined for those registrations to the sales invoices.

Microsoft Internal Use Only

Activity Diagram



Managing Dimensions in Invoicing – Design

You will modify the Create Seminar Invoices report to enable the transfer of dimensions when creating invoices.

GUI Design

No changes to the user interface will be necessary for this use case.

Functional Design

For every Seminar Ledger Entry record that we use to create a sales invoice, transfer the corresponding Ledger Entry Dimension records to new Document Dimension records for the invoice.

Table Design

No changes to tables will be necessary for this use case.

Managing Dimensions in Invoicing – Development

Exercise 27 – Modifying the Create Seminar Invoices Report for Dimensions

1. In report 123456700 Create Seminar Invoices, create two new local record variables: one for the Document Dimension table and one for the Ledger Entry Dimension table.
2. Enter code in the appropriate trigger so that after the program gets the Seminar Ledger Entry record and inserts the Sales Line, it deletes all Document Dimension records that correspond to the Sales Line. Then, for every Ledger Entry Dimension record that corresponds to the Job Ledger Entry, the program should create a new Document Dimension record for the Sales Line with the Dimension Code and Dimension Value Code of the Ledger Entry Dimension record.

Testing Managing Dimensions

You will now test the flow of dimensions from the master tables through the registration and posting processes right up to the creation of invoices. This test script assumes that you have successfully completed the setup and testing of those processes, and that some data exists.

1. Open the Seminar Card and view a seminar. Select Dimensions from the **Seminar** menu button and enter at least one dimension for this seminar.
2. Open the Seminar Room Card and view a room. Select Dimensions from the Seminar menu button and enter at least one dimension for this seminar room.

3. Open the Instructors List and select an instructor. Select Dimensions from the **Instructor** menu button and enter at least one dimension for this instructor.
4. Open the Seminar Registration form and create a new registration using the seminar, room and instructor that you define for dimensions. Because we set the line dimensions invisible, you will have to use the Show Column feature to see the dimension fields in the subform.
5. View the header dimensions by selecting the **Dimensions** menu item from the **Registration** menu button. Verify that they are what you expected based on the values from the master records. View the line dimensions by selecting a line and then selecting Dimensions from the **Line** menu button. Verify that the values are what you expected.
6. Post the registration and open the Posted Seminar Registration form. Verify that the dimensions are the same as those you specified before posting.
7. Open the Seminar Ledger Entries form and view the entries for the registration you just posted. Use the Show Column feature to see the dimension columns and verify that they are what you specified.

Test Your Knowledge

Review Questions

1. What is a dimension and what is its purpose in Microsoft Navision?
2. In what tables are the dimensions for the following entry types stored:
 - "Master" data entries (like Customer or Seminar)
 - Transaction document header and lines (like Sales Line or Seminar Registration Header)
 - Journal lines
 - Ledger entries
 - Posted documents
3. Describe the flow of information and how dimensions are transferred from the master data through transaction documents to ledger entries and posted documents.

Conclusion

Summary

In this chapter, we modified our existing seminar module objects so that dimensions can be defined and carried throughout the module.

Positioning – Where to go from Here

This training module has all the features of a standard Microsoft Navision application. You can now add some interfaces for additional features.

Quick Interaction: Lessons Learned

Take a moment to write down three Key Points you have learned from this chapter:

- 1.

- 2.

- 3.

Microsoft Internal Use Only

CHAPTER 9: MANAGING INTERFACES

This chapter contains the following sections:

- Introduction
- Using an Automation Server
- Using OCX or Custom Controls
- XMLPort Triggers
- File Handling
- Test Your Skills
 - Diagnosis
 - Managing E-mail Confirmation
 - Managing XML Participant List
- Test Your Knowledge
- Conclusion

Microsoft Internal Use Only

Introduction

Positioning – What is the starting point?

You have created a functioning seminar module with:

- Master tables and forms
- Registration functionality
- Posting routines
- Reporting
- Statistics
- Dimensions

You can now add some interfaces to the module to extend its functionality.

Preconditions

You need to have master tables and forms, tables and forms for seminar registration, and the posting routines to post seminar registration information.

Further preconditions are knowledge of the following areas:

- Writing internal documentation
- Enabling multilanguage functionality
- Exporting and importing objects
- Working with event triggers
- Working with complex data types and their member functions
- Using virtual and temporary tables
- Creating journal and document posting routines
- Debugging code
- Programming for low-impact on the application
- Using report event triggers
- Using special report functions
- Creating processing-only reports
- Using FlowFields for calculations
- Working with dimensions

Business Goals

In this story, our goal is to add features involving interfaces to the module.

Educational Goals

By completing this story, you will have learned or reacquainted yourself with the following:

- Using Automation
- Using OCX
- Working with an XMLPort
- File handling

Using an Automation Server

In C/SIDE, you can implement COM (Component Object Model) technologies in one of two ways: by using custom controls (OCXs) or by using Automation (C/SIDE in the role of an automation controller or client) to perform tasks with other applications. To create the e-mail confirmations, you will be using C/SIDE as an automation server.

Using an automation server consists of five basic steps:

1. Declare the creatable (top-level) interface (class) of the automation server as a variable of type Automation.
2. Declare all the other interfaces (classes) as variables of type Automation.
3. Use the C/AL function CREATE on the variable declared in step 1. Do not use CREATE on any other variables.
4. Use the methods and properties of the automation server in your C/AL code. The syntax and semantics for these methods and properties are documented in the documentation for each automation server.
5. You can CLEAR (destroy) the top-level object if you want. Otherwise, it will be destroyed automatically when the variable goes out of scope.

It is recommended that you create a separate codeunit for code that uses automation because of performance issues and because an object using automation can only be compiled on a machine on which the automation server is installed.

The data being transferred must be in text format.

Using Custom (or OCX) Controls

You can use custom controls (which are also known as OLE controls or ActiveX® controls) in Microsoft® Business Solutions–Navision®. However, Microsoft Navision only supports non-visual controls.

To use a custom control, it must be physically installed on the target machine and it must be registered with the operating system.

When you are ready to use a custom control, declare it as a global or local variable with a data type of OCX. Then select the custom control you want to use from a list by clicking the lookup button in the **Subtype** field. If the control you want to use is not in the list, you can install and register a control yourself. You can find information on how to do this in the Application Designer's Guide.

Once you have declared the control as a variable, you can access its methods and properties as normal. To see the methods and properties available, use the C/AL Symbol Menu. Methods can be run just as you would run any other function in C/AL, and properties can be read and set just as any other object properties.

XMLPort Triggers

There are three events that fire for an XMLPort object.

- **OnInitXMLPort:** This trigger fires when the XMLport is loaded and before any table views and filters are set.
- **OnPreXMLPort:** This trigger fires after the table views and filters are set and before the XMLport is run.
- **OnPostXMLPort:** This trigger fires after all the data items in an XMLPost have been processed.

There are also event triggers for every field in an XMLPort which are dependant on whether the XMLPort is importing or exporting. For more information on those triggers, consult the online C/SIDE Reference Guide.

File Handling

Use file-handling to create the XML file. Through File Variables, you can import data from or export data to any file accessible through your operating system.

The File Data Type

In order to gain access to an external file from within C/SIDE, you must first declare a variable of type File. This is a complex data type, which has numerous methods (functions) used to open, read, write and close external files.

Declare one File variable for each file you wish to access at the same time. You would be able to use one File variable to handle multiple files, but you have to access them one at a time, since each File variable can be open to only one file at a time.

Opening Files for Import or Export

When you are ready to use a file, you must open it. Before you open it you must set it up properly. To set it up properly, you should determine first whether you are going to use it for reading (import) or for writing (export). Although you could theoretically open a file for writing and then read it as well, this is normally not done in Microsoft Navision.

Below are the File Methods used to prepare a file for opening and for actually opening (and closing) it. As with other methods, these are called through the File variable. For example, if the File variable is named "ImportFile", then to use the Open method, you would enter:

```
ImportFile.OPEN(Name) ;
```

We will skip the "ImportFile." in front of each of these methods:

- **WRITEMODE(NewWriteMode)**: Sets the Read/Write status of a File variable before opening the file. If NewWriteMode is TRUE, you are able to write to the file; if NewWriteMode is FALSE, then you are only able to read from the file. Note that another way to use this method is like this: `IsWriteMode := WRITEMODE;`. By using this method after the file is open, you can tell whether a file is available for writing or not.
- **TEXTMODE(NewTextMode)**: Sets the type of data to be read or written. If TRUE, the file is opened in Text mode and if FALSE, it is opened in Binary mode. These will be explained a little later. This method can only be used before opening the file. Note that another way to use this method is like this: `IsTextMode := TEXTMODE;`. With this method, used only after the file is open, you can tell whether a file is being read or written in text mode or binary mode.

- **QUERYREPLACE(NewQueryReplace):** Use this method if you are going to open a file using the CREATE method. If NewQueryReplace is TRUE, and the file already exists, C/SIDE asks the user before a replace will be allowed. If NewQueryReplace is FALSE and the file already exists, C/SIDE erases the old file and replaces it with a new (empty) one, without asking the user. If the file does not already exist, this method has no effect.
- **OPEN(FileName):** Use this method to open an already existing file. Be sure to set the WRITEMODE and the TEXTMODE before opening a file. FileName should be the full path name of the file. If you use OPEN as a statement (without looking at the return value) and the file indicated by FileName does not exist, a run-time error occurs. If you use OPEN as an expression (looking at the return value) and the file indicated by FileName does not exist, OPEN returns FALSE, but if the file does exist, it is opened, and OPEN returns TRUE.
- **CREATE(FileName):** Use this method to create and open a file. Be sure to set the WRITEMODE, the TEXTMODE and QUERYREPLACE before creating a file. FileName should be the full path name of the file. If you use CREATE as a statement (without looking at the return value), and the file indicated by FileName cannot be created (say the path does not exist), a run-time error occurs. If you use CREATE as an expression (looking at the return value) and the file indicated by FileName cannot be created, CREATE returns FALSE, but if the file is created, it is opened, and CREATE returns TRUE. If the file already exists, it is cleared and opened. Whether the user is warned about this or not depends on the parameter to the QUERYREPLACE method called before calling CREATE.
- **CLOSE:** Use this method to close access to the file through this file variable. Once this is called, you cannot use this file variable again to access a file unless it is re-opened or re-created. If CLOSE is called and the file is not opened, a run-time error occurs.

Methods of Reading Files

There are two methods provided for reading or writing data in external files. The method is set using the TEXTMODE method described before. There are two possible settings:

- **TEXT (TEXTMODE=TRUE):** Each file access will read or write a line of text. The variable used can be of any type and it will be converted to (if writing) or from (if reading) text during the processing.
- **BINARY (TEXTMODE=FALSE):** Each file access will read or write a single variable. The variable will be read or written in its internal format. For example, a Text is written out as a null terminated string whose length is the defined length of the Text variable. Since Text Mode is limited to one variable per line, note that you cannot have more than 250 characters per line in the file. Binary mode does not have this limitation, since it does not have lines. However, the internal format can really only be read if it was written in the same manner from Microsoft Navision. The one variable type that is very useful when using Binary mode is the Char type. Using this, you can read or write one byte at a time. In the SkeletonFileImport object, we use Binary Mode and the Char type to import text files that can be of any length.

Reading or Writing Data in External Files

The following methods are used to read or write data in external files (once the file variable is open):

- **[Read :=] READ(variable):** Read a variable from the file. The optional return value indicates the number of bytes read. If you are using Text mode, a line of text is read and the text is evaluated into the variable passed in as a parameter. If you are using Binary mode, the variable is read in its internal format, and thus the number of bytes read depends on the size of the variable.
- **WRITE(variable):** Write a variable to the file. If you are using Text mode, the variable is formatted to text and written out as a line of text. If you are using Binary mode, the variable is written using its internal format.

There are other File Handling methods that can also be used.

Test Your Skills – Managing Interfaces – Diagnosis

Description

In the functional requirements, our client expressed the desire for two features that involve interfaces:

You should be able to send an e-mail notification to the customer's participants in several types of situations, such as registration confirmation.

You should be able to print a list of the participants registered for a seminar as an XML file.

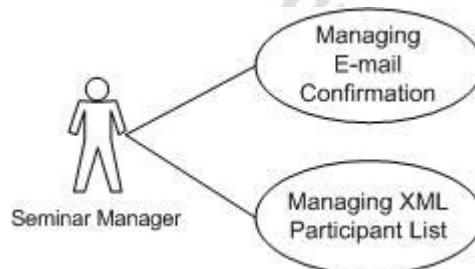
Note that you will need two interfaces: one that handles e-mail and one that creates an XML file.

Use Cases

We can split the analysis and implementation phases of this story into the following use cases:

- Managing E-mail Confirmation
- Managing XML Participant List

The following diagram illustrates how the use cases relate:



Implementation of Use Case 1 – Managing E-mail Confirmation

Managing E-mail Confirmation – Analysis

Our client describes the e-mail confirmation in this way:

We should be able to send an e-mail notification to the customer's participants in several types of situations, such as registration confirmation.

You will create a feature that will produce and send an automatic confirmation e-mail message to each of the registered participants for a seminar.

Purpose

The purpose of managing e-mail confirmation is to provide a confirmation message for participants, using e-mail as the medium.

Preconditions

Tables and forms for seminar master data and registrations must exist.

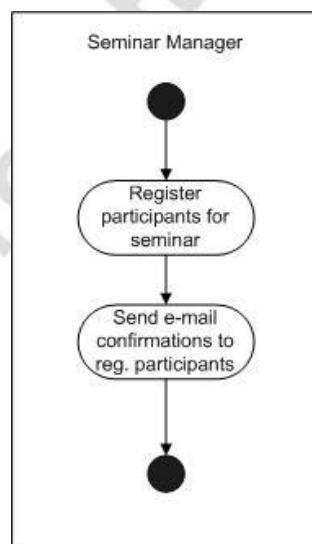
Postconditions

The result of this use case is that an e-mail confirmation feature will be available to the user.

Main Scenario

At a certain date before the beginning of a seminar, seminar managers send out e-mail confirmations to all those participants who are registered for the seminar to inform them that they are registered and to remind them of the starting date for the seminar.

Activity Diagram



Managing E-mail Confirmation – Design

GUI Design

We want the e-mail confirmation to be available from the seminar registration.

Seminar Registration (Form 123456710): Add a new **Functions** menu button as shown below:

Bill-to Cu...	Participa...	Participant Name	Participa...	Register ...	Confirma...	To Invoice	Reg
10000	CT100140	David Hodgson		01.25.01		✓	
10000	CT100156	John Emory		01.25.01		✓	
10000	CT000001	Mindy Martin		01.25.01		✓	
10000	CT100210	Stephanie Bourne		01.25.01		✓	
30000	CT200080	Pamela Ansmann-Wolfe		01.25.01		✓	

Seminar Registration List (Form 123456713): Add a new **Functions** menu button as shown below:

No.	Starting ...	Seminar ...	Seminar Name	Status	Duration	Maximu...	Room Coc
REG00001	01.08.01	SEM0002	Solution Development	Planning	10	20	ROOM01

Functional Design

To send e-mails, use an Automation variable with a subtype of an automation server class that can create and send e-mails. For you, this automation server is the Navision Attain ApplicationHandler, and the class you will use is MAPIHandler.

Create a codeunit that can create and send the e-mails. In this codeunit, use the same principle used to post lines – that is, have one function (or codeunit, in the case of posting) that handles one line, and a separate function that handles multiple lines by calling the first function for each line.

Table Design

No tables will be created or modified for this use case.

Managing E-mail Confirmation – Development & Testing

Exercise 28 – Creating E-mail Confirmations

Begin by creating a codeunit to create the e-mail.

1. Create a codeunit called 123456705 SeminarMail.
2. In the C/AL Globals, declare record variables for the Seminar Registration Header, Seminar Registration Line, Customer and Contact tables. Declare an integer variable to hold the number of errors. Finally, declare a variable, called MAPIHandler, of type Automation. In the Subtype, select Navision Attain ApplicationHandler as the Automation Server and MAPIHandler as the class.
3. Define text constants for the different sections of your e-mail confirmation, including the subject line, the greeting, the confirmation sentence and the signature.

Now create the NewConfirmationMessage function that does the work of creating one e-mail message.

4. Define a function called NewConfirmationMessage that has a parameter of a record variable for the Seminar Registration Line table, which is passed by reference.
5. Enter code in the function trigger so that the function performs the following tasks:
 - Creates an instance of the MAPIHandler (using the CREATE function).
 - Creates the e-mail message by assigning the different elements of the message to the properties of the MAPIHandler variable.
 - a. ToName is the E-Mail from the Contact record
 - b. CCName is the E-Mail from the Customer record
 - c. Subject is the subject line text

Use the AddBodyText method of the MAPIHandler to create each line of the e-mail message.

***NOTE:** To see all the properties and methods available for the MAPIHandler, open the C/AL Symbol Menu by pressing F5 or clicking View, C/AL Symbol Menu.*

- "Sends" the message using the Send method of the MAPIHandler.
 - Uses the ErrorStatus property of the MAPIHandler to check the number of errors.
 - If there are no errors, sets the Confirmation Date on the **Seminar Registration Line** to today's date.
6. Define a function called SendAllConfirmations with a parameter of a record variable for the Seminar Registration Header.
 7. Enter code in the function trigger so that the function runs the NewConfirmationMessage function for each **Seminar Registration Line** associated with the Seminar Registration Header that was passed to it.

Now that you have code to create and send e-mail confirmations, you need to make the code available to the user.

8. Add a new menu button and menu item to form 123456710 Seminar Registration as follows:

Menu Button	Options	Comments
Functions	Send E-mail Confirmations	Runs code to Send E-mail Confirmations.

9. Enter code in the appropriate trigger so that when the user selects the Send E-mail Confirmations menu item, the program runs the SendAllConfirmations function.

10. Add a new menu button and menu item to form 123456713 Seminar Registration List as follows:

Menu Button	Options	Comments
Functions	Send E-mail Confirmations	Runs code to Send E-mail Confirmations.

11. Enter code in the appropriate trigger so that when the user selects the Send E-mail Confirmations menu item, the program runs the SendAllConfirmations function.

Microsoft Internal

Implementation of Use Case 2 – Managing XML Participant List

Managing XML Participant List – Analysis

Our client's functional requirements described the need for a participant list in XML format:

You want to be able to export the participant list for a seminar as an XML file. Therefore, you need to create a participant list report, similar to the report we created in a previous story, which is exported in XML format to a file instead of being printed.

Purpose

The purpose of managing the XML participant list is to create a participant list in XML format.

Preconditions

The tables and forms for seminar registrations must exist.

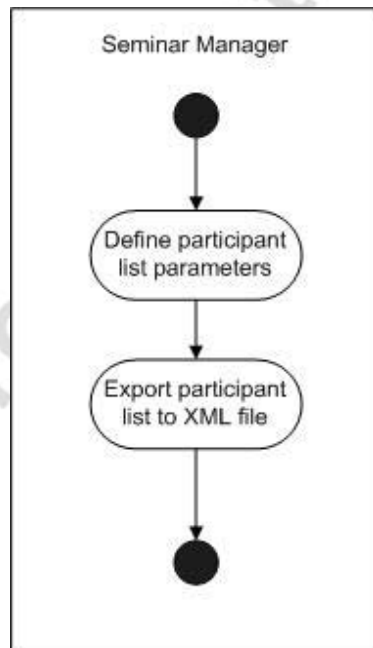
Postconditions

The result of this use case is an XML file that lists participants in a seminar.

Main Scenario

The seminar managers create participant lists for one or more seminars in an XML file.

Activity Diagram



Managing XML Participant List – Design

GUI Design

Partner Menu (MenuSuite 80): Add the following menu item to the Seminar menu:

Menu Type	Menu Name	Group	Comments
Item	Create XML List	Periodic Activities	Runs report 123456705 XML Sem. Reg.-Participant List.

Functional Design

The XMLPort has three basic sections:

- A title, Seminar Registration – Participant List.
- Heading information from the Seminar Registration Header including the **No.**, **Seminar Code**, **Seminar Name**, **Starting Date**, **Duration**, **Instructor Name** and **Room Name**.
- Line information for each registered participant from the Seminar Registration Line, including **Customer No.**, **Contact No.** and **Name**.

The following table shows the basic sections you will create:

Section	Sample Section
Title	<Seminar_Registration_-_Participant_List>
Heading for Heading information	<Seminar>
Heading information	<No>REG0001</No>
Heading for Line information	<Participant>
Line information	<Customer_No>10000</Customer_No>

In the appendix, you can see how the XML file should look when it is exported.

Table Design

You will not modify or create tables for this use case.

Managing XML Participant List – Development & Testing

Exercise 29 – Creating the XML Sem. Reg.-Participant List

Begin by creating the XMLport.

1. Create a new XMLport called 123456700 Sem. Reg.-XML Participant List.
2. Define the data items for the XMLport, using the functional design as a reference for which tables we will draw information from. Add code to calculate the FlowFields so that the **Instructor Name** and **Participant Name** appear in the XML file.

Now implement a way that the user can run this XMLport. To give the user the ability to select filters, we are going to create a new report:

3. Create a new Processing Only report called 123456705 Sem. Reg.-XML Participant List.
4. This report has one data item, the Seminar Registration Header table. Set the name of this datatype to SEMREGHEADER.
5. Create global variables for TestOutputStream (datatype Outstream) and TestFile (datatype File).
6. In the OnPreReport trigger insert this code, substituting your preferred file path:

```
TestFile.CREATE('C:\XML_Part_List.XML');  
TestFile.CREATEOUTSTREAM(TestOutputStream);  
XMLPORT.EXPORT(123456700,TestOutputStream,SEMREGHEADE  
R);  
TestFile.CLOSE;  
MESSAGE('XML Completed');
```

You are now finished with the XMLPort, so all that remains is to add the menu item to the Seminar main menu.

7. Add the new menu item to the Periodic Activities folder in the Partner MenuSuite.

Conclusion

Chapter Summary

In this chapter, you created interfaces by using an automation server to send e-mail messages to participants and an XMLport to create an XML file of participants in a seminar.

Positioning – The Next Step

Our seminar module is complete. Before finishing, we will examine some deployment issues.

Microsoft Internal Use Only

Test Your Knowledge

Review Questions

1. In what two ways can you use COM objects in C/SIDE?
2. When you want to use a variable for an automation server, what data type do you give the variable?
3. What C/AL function do you use to create an instance of an automation server class?
4. Can you transfer picture files to an automation server?
5. Where should you ideally place code that uses automation?
6. What event triggers exist for XMLports?

Quick Interaction: Lessons Learned

Take a moment to write down three Key Points you have learned from this chapter:

1.

2.

3.

Microsoft Internal Use Only

Microsoft Internal Use Only

CHAPTER 10: DEPLOYMENT

This chapter contains the following sections:

- Introduction
- Deployment Tasks
- Ongoing Support
- Test Your Knowledge
- Conclusion

Microsoft Internal Use Only

Introduction

The deployment phase marks the end of the implementation phase. During deployment, the final preparations are made and the solution is installed at the client site.

There are a number of preconditions for this phase:

- The new solution must be completely developed and tested.
- Any errors must have been identified and resolved.
- The various analysis and design documents must be updated for any changes.
- The infrastructure must be in place.
- The users must be trained and knowledgeable about the new module they will be using after deployment.
- The user documentation must be prepared and ready for use.

Once these preconditions are in place, you are ready to implement our seminar module at the CRONUS International Training Academy site.

Deployment Tasks

Although the most important, and most exciting, deployment task is "going live" with the new module at the client site, there are a few tasks that must be accomplished first.

Steering Committee Approval

Conduct a final walk-through of the new module with the Steering Committee to obtain their final approval for delivery. Once you have this approval, you can prepare the system for going live.

Configuration Checklist

Go through a checklist of all the configuration elements that must be in place before the module can be used. In this case, you must be sure that the various new number series have been created and that the seminar setup is properly completed.

Data Conversion

During data conversion, you will perform the transfer of existing customer data into the new seminar module. This includes master file data as well as ledger entries and transaction documents.

Importing Document and Ledger Data

There are generally three types of data that you import into Microsoft® Business Solutions–Navision®: master or supplemental table data, transaction document data and ledger data.

Importing the master data is fairly straightforward, and you will most likely not need to write any code.

Importing transaction documents, however, is more complicated. Because of the complex nature of the header and line data and because the sheer amount of data per line can sometimes be beyond the limits of a dataport, it is not recommended to import documents. Historical data stored in the ledger entries is usually sufficient. However, if it is necessary to import open orders, you can use a dataport to import two files, one for the headers and one for the lines, making sure to validate the fields. If some orders are partially shipped, you should only import the portion of the order that is still open.

If necessary, you can import historical, or posted, documents. The process for doing this is a bit simpler because you do not need to validate the fields since they are for viewing only, and because you can enter line numbers sequentially, since the user cannot insert new posted document records.

***NOTE:** When importing ledger or balance data, the most important thing to remember is to never import directly into a Ledger table! Instead, import the data into an appropriate Journal table. You can either leave the entries in the Journal table to be reviewed and posted later or, if you perform more testing up front to ensure that the data is correct, post the Journal entries as you import them.*

One other thing to note is that the validation of some fields changes the values of other fields. For instance, when you validate the **Account No.** field, the validation code fills in the **Description** field. If the **Description** had already been imported, the imported value would be lost. To solve this problem, create a global variable and put that variable's identifier into the SourceExpr property of the **Dataport** field rather than the actual field name. Then, after you call the validation of the **Account No.** field, you can transfer the **Description** from the variable to the field.

You should always fill in the **Source Code** field with a unique value for each import routine. By doing this, you can always see where certain data in the journals and ledger entries came from.

When importing document or ledger data, write code to check that the data follows Microsoft Navision business rules, to initialize the records and to fill in any missing data.

Dataport Event Triggers

The events for dataport triggers revolve around the processing of data items and the import and export of records. The initial triggers that run before the data items are processed are `OnInitDataPort`, `OnPreDataPort` and `OnPreDataItem`. `OnPreDataPort` can be used to validate any options entered by the user on the request form. `OnPreDataItem` can be used to insert code that will be run once for the entire Dataport.

To write code that fires when records are exported, use the `OnBeforeExportRecord` and `OnAfterExportRecord` triggers. Likewise, to work with importing records, use the import triggers. Use `OnBeforeImportRecord` to initialize the record to be imported. Use `OnAfterImportRecord` to validate the information being imported, fill in missing data, and to either store the record or post it. To write code that fires after a data item has been processed, use the `OnPostDataItem` trigger, and to write code to fire after all data items have been processed, use the `OnPostDataPort` trigger.

Exercise 30 – Creating Contact Dataport

The client would like to import contacts into our new seminar module. You will create a simple Dataport to perform this task.

1. Create a dataport called 123456700 Import Contact with the Contact table as the DataItem.
2. In the Dataport fields, choose to import the fields:

No.
Name
Type
Address
Address 2
City
Country Code
Phone No.
Company No.

3. Write code to update the **Currency Code** based on the **Country Code**. The countries that you should set up are detailed in the table below.

Country Code	Currency Code
AT	EUR
BE	EUR
CA	CAD
DK	EUR
NL	EUR
US	USD

Go Live

At go live, the system becomes available to users for normal daily transactions. After the system is up and running, you can hand the administration of the module over to a client manager.

Project Evaluation

As your final deployment task, perform a project evaluation with the project team to assess the key aspects of the project planning and implementation. Some of the factors you evaluate will be the following:

- Cooperation with the client and within the project team.
- Utilization of the project team skills.
- Project planning.
- "Highlights" and "lowlights" of the different project phases.
- Client satisfaction.

Ongoing Support Phase

After deployment, the project enters the ongoing support phase. As a Microsoft Navision developer, there are two primary activities that you perform during this phase: implementing new customer requirements and updating for new releases. Each new development project should follow the Microsoft Navision Implementation methodology we have used in developing the seminar module. Let's examine what is involved in the upgrade process.

Reasons to Upgrade

There are two important reasons to upgrade your customer's Microsoft Navision installation.

Your Customer has already paid for it: Many of your customers will have an Upgrade Agreement in effect from the time of sale. Thus, many of your customers have paid for and will expect an upgrade.

Your Customer will pay for it: Some Upgrades will be important enough that your customer will want them even if they have not paid for them in advance with an Upgrade Agreement. Also, even with the Upgrade Agreement, the customer is only purchasing the product upgrades and not the installation of the product upgrades. The installation fees could be worth while, if upgrades are planned for and implemented properly.

Benefits to the Customer

- **Gain access to new features:** First, they may gain access to new features. For example, in version 4.0 we have implemented Intercompany Postings. If your customer does not upgrade to version 4.0, they will not be able to use this feature.
- **Improvements to existing features:** In version 4.0, there are improvements to financial reporting, budgets, sales documents, dimensions and human resources among others. If your customer wants any of these improvements, they need to upgrade to version 4.0.
- **Improved performance and removal of problems:** Performance is improved and problems (read "bugs") are removed with every single upgrade.
- **Allow upgrades to new operating systems and hardware:** For example when Microsoft Navision Financials version 2.60 was released, it added support for Microsoft® Windows 2000, didn't include support for SQL 2000.
- **Better support when on the current version:** Your customer will tend to receive better support when they are on the current version. As versions become obsolete, fewer and fewer people have expertise in that version.

Benefits to the Solution Center

- **Better customer relations:** Customer Relations are built on providing services to the customer. An Upgrade is an opportunity to provide a service to an existing customer that is NOT in response to a customer complaint. It also gives you a chance to talk with your customer and find out what new needs they may have which you could provide for them. Finally, it keeps you in their minds.
- **Easier to support when the customer is using the current version:** In many cases, the solution to a problem your customer has is found in a newer version. In other cases, you may find it hard to support an old product when nobody at your organization can remember it.

- **Remove problems before the customer notices them:** When your customer finds a problem that turns out to be caused by a bug, they often want you to fix the problem for nothing, and even if you do (you shouldn't, but...), they may be irritated with you and with the product. If an upgrade fixes a bug that the customer has not run into yet, they pay for it with the upgrade, and there is one less opportunity for them to be disappointed.
- **Opportunities for additional sales:** Microsoft Navision often includes new features that your customer may be interested in using. However, in order to get this new feature, the customer must upgrade, which generates service revenue for you.

Definitions

Before discussing the types of upgrades, we need to explain a few definitions.

- **Executables:** The Microsoft Navision programs that run under the operating system and include the client, the server, the C/ODBC drivers and so on. They can be identified by the fact that you see them in the Windows or NT Explorer, and they are files that end with ".exe" or ".dll", and so on.
- **Application:** The programs that run within the Executables, more specifically within the client, which is known as C/SIDE. The Application is made of Navision Objects.
- **Functional Area:** A major division within the Application, like General Ledger or Inventory, that can be identified by the fact that there is a Navigation Pane Menu item that identifies it and by the sub-menu that appears when this button is pressed.
- **Granule:** A set of objects that is purchased separately which contains a set of features.
- **Feature:** A small set of functions that are part the Application. The General Ledger is a Functional Area within the Application, Account Schedules is a Granule, and Date Comparison is a Feature of Account Schedules.
- **Bug:** A piece of the program that does not work as intended. Note that if a piece of the program is not working the way you would like does not make it a bug. Only if it does not work as designed by Microsoft Navision is something considered a bug.
- **Enhancement:** An "enhancement" is when an existing feature is made to work better in some way, easier to use, faster, performing additional functions, and so on. If a feature is merely made to work as it was originally intended, then that is a bug fix, not an enhancement.

Types of Updates

There are different categories of Software Updates which you may be required to install:

- **Hotfix:** A single cumulative package composed of one or more files used to address a problem in a product. Hotfixes address a specific customer situation and may not be distributed outside that customer organization.
- **Release:** Normally scheduled product release that includes the upgrade toolkit.
- **Update:** A broadly released fix for a specific problem addressing a non-critical, non-security related bug. Can include critical updates and feature packs.
- **Critical Update:** A broadly released fix for a specific problem that addresses a critical, non-security related bug.
- **Service Pack:** A tested, cumulative set of all hotfixes, security updates, critical updates, and updates, as well as additional fixes for problems found internally since the release of the product. Service packs may also contain a limited number of customer-requested design changes or features.
- **Feature Pack:** New product functionality that is first distributed outside the context of a product release, and is usually included in the next full product release.

Planning for Upgrades

The most important step in the upgrade process is to plan for it in advance. When you install Microsoft Navision on your customer's system, you know that you will upgrade eventually, so you might as well plan for it.

Start this planning when you are making your initial implementation plans. Many times, an upgrade may be made very difficult because of a decision made when you first implement the customer's system. The largest area is in customizations. Other areas are also important. For example, an upgrade of the executables usually requires changes on every single client machine. Throughout the implementation process, keep you customer informed of upgrade considerations. For example, if they do not know that upgrades will bring their system down for a while, they will be irritated when an upgrade is done. If they know in advance, they can help plan by telling you when might be the best time to upgrade.

There is no overstating the importance of keeping good records. You must know what is on every customer installation, so you can:

- Make appropriate decisions about whether to install an improvement.
- Make correct plans about how long an upgrade will take.
- Know when to ask your customer about whether they want certain new features.
- Know when an upgrade will override a customization that you have done.

Customizations

The most important planning for upgrading is when you are doing customizations. Consider how you will upgrade your customizations whenever you modify your customer's software. The best way to do this is to use the recommended methodology.

We recommend including considerations for Upgrading with every Design Specification. In addition to helping you plan ahead, it also helps the customer to make intelligent decisions. If they know that a particular design will mean extensive upgrade problems every time, they may opt for an alternate design, even if it costs more money at first.

Also, using the Low Impact programming techniques are part of the Attain Development course. Only make changes to the base application when necessary, and then make them in a way that is easy to upgrade. In addition, document every change. This is not optional – it is essential. You cannot have a smooth upgrade without knowing what is on the customer's system.

If your customer has development tools, you have additional problems. Teach your customer to use the same documentation and modification techniques that you have learned when programming. You need to be copied on all of their documentation.

Scheduling

When it is time to perform the upgrade, the planning starts with scheduling. Once you decide that an upgrade will be done, you must start planning a schedule with your customer well in advance. There are several reasons for this.

The Microsoft Navision Server (or SQL Server) must be taken down every time you do an upgrade, no matter how small. Even if there isn't an Executables upgrade, there is an object cache on every client. In addition, a backup must be performed before every upgrade, no matter how small. Treat every upgrade as though you were implementing a new system, since in some ways, you are.

Because of the need to take down the server for a period of time, you may need to install the upgrade during off-hours, or even over a weekend, if there is a data conversion as part of the upgrade.

There are several things you can do to relieve some of these time issues and make the upgrade less painful for the customer, but they require planning and scheduling. First of all, enlist the support of your customer's IT department for anything that they can do for you. For example, the customer's IT department can see that the server is taken down and that backups are performed before you get there. Furthermore, they can also distribute Executables changes, and communicate with the users about the upgrade. Second, do as much as possible at your own site. For example upgrade all objects, write and test the data upgrade routines, and finally test the installation. When you've done all this, you are ready to bring everything to your customer's site for execution. However, make sure you plan for contingencies – what if things fail.

Upgrading the Executables

If there is an upgrade to the executables (the Microsoft Navision programs that run on the Operating System), this part of the upgrade is done first. It can be scheduled separately from any other steps. For example, you could upgrade the Executables portion of a Release one weekend, and not schedule the upgrades for the Objects or Data for another two or three months.

Upgrading the Executables is very easy, normally just copying the new files over the old ones. However, when you have 50 or even more Client machines to upgrade, this can take a lot of time.

Sometimes, the internal database structure can change. When this happens, you will be told, but all you need to do is make a Microsoft Navision Backup under the old system, and then restore from the backup after you have installed the new system.

The first step is to determine what is being upgraded. This can be an upgrade of the Microsoft Navision Server, the Client, or both. It can also involve other Executables, like N/ODBC. Sometimes, a new Customer License is required. If so, be sure to order one before you proceed. Read the Upgrade Toolkit documentation carefully to determine what exactly is upgraded. Note that there are special instructions when upgrading the SQL Server Option.

Navision Server

If you need to upgrade the Microsoft Navision Server, follow these instructions. First, if you need to do a Navision Backup. Make the backup while the old system is still running. This must be a complete backup – all companies, data common to all companies, and application objects. Then, you can have everybody log off the system and bring down the Microsoft Navision Server. Now, make a complete backup copy of all Navision files (executables and database) using tape or another fixed disk drive. This backup will be used should anything go wrong with the upgrade.

Now, install the new Server components. This normally involves copying the new files over the old ones. Sometimes it requires a new installation, but just install the new version over the old version. Normally, there is a Client also located on the server machine. If so, and if there is an upgrade to the Client Executables, then do this now. If there is a new License involved in the upgrade, copy it into both the server directory (the one which contains server.exe) and the client directory (the one which contains fin.exe).

If a Navision Backup was required due to a database structure change, delete the old database, bring up the Microsoft Navision Client on the Server Machine, and create the new database (using the same name and location as before). Restore the complete Navision Backup into this new database. Bring down the Server Machine Client.

Now, bring up the Server, and bring up the Server Machine Client. Log onto the Server from the Server Machine Client. Run a few simple tests to make sure that the system has been properly restored and that you can still run basic functionality.

If there are Executable upgrades for Clients, bring down the Server and upgrade all of the clients. This will be covered in the next section. Once this is done, you can bring the server back up, and you are back on line.

Navision Clients

If there are Executable Changes for a Client, the steps are quite simple. Bring down the client (exit Microsoft Navision). Copy the new Client files into the client directory, or re-install the client; whatever the upgrade kit instructions say.

If there are other Executable Upgrades, like N/ODBC, C/FRONT, and so on, then install them now as well. The upgrade kit includes specific instructions.

If a new Customer License is required, copy it into the Client directory at this time. Then bring the client up.

You need to repeat the above steps for each Client in the installation. If there are many Clients, you may want to enlist the customer's help in doing this part of the upgrade. The steps are very simple, but must be repeated on each system.

If the Server Executables are also being upgraded, you must first bring down all Clients, upgrade the Server, upgrade all Clients, bring the Server back up, and then bring all Clients back up.

Upgrading the Objects

The Object Upgrade is the most difficult part of any upgrade, no matter what the category of software update. What makes this difficult? Well, first Microsoft Navision a/s in Denmark made worldwide changes to the application objects. Next, the NTR makes changes to these same application objects. Then, you made custom changes to these same application objects. In fact, if your customer has the design tools, they may have made their own changes to these same application objects. All without consulting each other!

Each NTR is responsible for merging Microsoft Navision a/s changes with their changes (if applicable). This is why it takes time between the worldwide release of a version and the local NTR release of that same version. It is your job to merge these changes with the changes that you made during customization. There are many things that help, a lot of them depend on you following the rules for documenting customizations. We will review the Version Tags and Modification Flag rules as we go through the upgrade steps. (For a review of versioning, please consult the Development I manual). We will review the basic process that you must go through for each individual application object. For this example, use Table 18, the Customer Table, as a stand-in for any application object.

For each object, you have a possibility of up to three versions. First, the old Base version, which you can get from the CD that you installed from. It is very important that you document what version you installed from. Second, you have the new Base version and also you have the old Customized version, which is what is currently running on your customer's system. What we want when we are finished, is a new Customized version of the object, which includes all of the customizations, plus all of the changes made for this upgrade.

There are four possibilities here:

Possibility One

First, neither you nor we made any changes to this object. In this case, the object on your customer's system does not change during this upgrade.

Possibility Two

Second, you customized the object, but we do not change it for this upgrade. Again, in this case, the object on your customer's system remains unchanged for this upgrade.

Possibility Three

The third possibility is that we change the object for the upgrade, so there is a new base version, but you did not customize this object. In this case, the new object replaces the object on your customer's system.

Possibility Four

Finally, the possibility that causes the problems – you made customizations to the object, and we also changed the object as part of the upgrade. In this case, you must first determine which version changed the object the most, either your changes, or our changes. Whichever one changed the object the most, you should use that object. Then, you must change that object so that it includes all of the changes that were made by the other.

For example, the upgrade you obtained from us contains more changes than you made during customization. Therefore, get the object that you obtained from us, re-do your customizations on that object, and then replace the object on your customer's system with the new, customized object.

Another example, the customizations you made contain more changes than we did for the upgrade. Therefore, get the object from the customer system, re-do all of our changes on that object, and then replace the object on your customer's system with the new customized object.

A Typical Upgrade

Let's run through the typical steps to upgrade objects. In this situation, you have made a few small changes for your customer.

Assume that you have followed all of the recommendations for Version Tags, that is, that in the customer's system all modified objects have been tagged with an additional Version Tag (we will use CR01), and that all Modification flags are turned off. In addition, your customer has no design tools; therefore, you are not required to go to the customer site to pick up the objects, since the customer has not changed them. You kept a copy of the customer's database in house. This copy has just objects, either no data or minimal (Cronus) test data. You also have a copy of the target version application

Step by Step

1. Make a working copy of the customer's database (objects only). Simply do a Navision backup, create a new database, and then restore Application Objects only.
2. Get the Improved Objects from the target version database. To do this, go into the Object Designer and click the **All** button. Then put your cursor in the **Version Tag** column, and set a filter to your NTR's code for the new version. This filters out all but the improved objects. Then select all objects (CTRL+A) and export. Export as a Navision Object file, with an extension of .fob.
3. Go back into your Working Copy database and import the Navision Object file into the Object Designer. Normally, there will be conflicts, but even if there are not, bring up the Object Import Worksheet anyway. Look over all of the objects, find the conflicts, and write down the object type and ID on a Conflict List. Skip any objects that have a conflict for this step. For the remaining objects, there is no conflict; the new objects simply replace the old ones since the old ones were not customized.

4. Review the four possibilities again. First, an object might not have changed in either the customized version or the new base version. In this case, we would not have imported the object, since only the changed objects were imported. These do not show up in the Import Worksheet. Second, the object might only have changed in the customized version, but not in the new base version. In this case, the object is not imported, since only new base objects are imported. Again, these do not show up in the Import Worksheet. Third, the object might have changed only in the new base version, but not in the customized version. These objects are in the Import Worksheet, but they show up with no conflicts. They can just be imported. Fourth, the object might have been changed in both the customized version and the new base version. These objects are in the Import Worksheet. In this case, they show up with a conflict. Skip these objects (do not import them) and write them down to create a Conflicts List.
5. Review the Conflicts list. You should look up each object in the Conflicts List in the Change Log that came with the new version. Remember to check all the improvements, since it might have been modified more than once. If you look at the Change Log in the Latest Improvements page, where they are all together in one log, and you start at the bottom with the version you see in the Import Worksheet, you can find all of the changes more easily.
6. For each object in the Conflicts List, ask yourself if the upgrade modification is small? More to the point, is it smaller than (or even almost as small as) the change you made for your customization?

If the answer to this question is Yes, then use the Change Log that came with the new version to implement the improvement on the customized object in the Working Copy database.

If the answer to this question is No, then import the new base object into the Working Copy database, and then use your Change Log to implement the customization on the New Base object.

Do you not have a change log? Then use the Compare tool to create one. This makes it easier for you to decide which change is smaller.

Either way, set the version tag to reflect both new versions. This shows both the new NTR Version Tag and the Customization Version Tag.

There is one exception to these rules. If this is a table object, we may have added a new field in a number range that you cannot add. In this situation, if at all possible, import the object and then implement your customization on the new object using your change log. If this is not practical, the only alternative is to set the action to "Merge Existing<-New" and import the new base object. This brings in the new field(s). Then, apply the remainder of the Change Log.

7. Complete step 6 for every object on the Conflicts List. The hard part is over.
8. As an optional step, you can change the Version that is displayed on the Help About screen. Note that this step is optional only for an improvement. If this is a Service Pack or a Minor or Major release, you must perform this step.

In the Object Designer, open Codeunit 1 and look for the function called ApplicationVersion. An example is shown here. Simply change the displayed version to the new version you wish to display, reflecting the latest base version tag in all the object. Just change the text between the single quote marks to this text. After you update Codeunit 1 and save it, don't forget to update the version tag of this object as well, to the same version as you put in those quotes.

9. The next step is to compile all objects. You do this to make sure there are no conflicts with existing objects that show up. In the Object Designer, click the **All** button, select all objects, and press F11 to compile them. When it is done, any objects that did not compile will be marked, so you can View, Marked Only, in order to check them out and fix any problems.
10. Now, you must create the Upgraded Objects file. In the Object Designer, click the **All** button, and then filter the **Version Tag** column. Once this is done, clear all the **Modification Flags**.
11. Select all the filtered objects (use CTRL+A) and export them as a Navision Object File, a file that has the extension .fob. This is the Upgraded Objects file.
12. Now you can install the Upgraded Objects on your customer site. First, bring this file to your customer site. Schedule a non-busy time to do this. When you load objects, especially table objects, sometimes tables are automatically converted or re-keyed, which takes a long time when there is a lot of data.

At the Customer Site, bring down the server. Make a backup copy of the database (enlist the support of the customer's IT department for this step). Start the client up which is located on the Server Machine, and open the customer's database in Local mode. Now go into the Object Designer and Import the Upgraded Objects .fob file.

Remember, even if there are now conflicts, you have already resolved all of them. Therefore, when you bring up the Import Worksheet, all you need to do is click the **Replace All** button, and then do the import.

Once this is completed (it might be 30 seconds, it might be 3 hours or more), compile all objects, then bring down the local client. Bring the Server back up, and your customer can continue working on their newly upgraded system.

Tools for Upgrading the Objects

Microsoft Navision gives you several tools you can use to help you upgrade. Each one may be best for different circumstances.

Import Worksheet

You have seen how the Import Worksheet can be a useful tool. It is the best tool to use to rapidly determine object conflicts. It can also be used to add new Objects and Fields in the base number ranges that you may not have permissions for. Finally, as you can see, it is the best tool to use to install the upgraded objects on the customer's system.

Editing Text Objects

C/SIDE allows you to import Text Objects as well as Microsoft Navision Objects. This is the best if you have manually applied an automatic change log to some objects. It is also used when importing the Merge Tool results.

Compare Tool

The Compare Tool is the best tool to use when you are generating, and printing an Automatic Change Log. It can also export an automatic change log, which is in a good format to use for manual updates. Remember that if you use an automatic change log, it is best to make the change in a text object file and import it as text. Also, the Compare tool helps you find the relative size of each modification by reporting what percent of each object changed.

Merge Tool

The Merge Tool is the best tool for automatically merging changes. The Merge Tool is used only to merge the objects that have conflicts, in other words, only load the objects (from both the Customized and the New Base versions) into the Merge Tool that you need to compare and merge. The Merge Tool is also very useful to track and store the various versions that exist for your various customers. For it to do this properly, you need to disregard previous instructions and load all changed objects from both the Customized and New Base versions.

Using the Change Log

Many times when an improvement is made, a manually generated change log is published. This is the best tool for documenting modifications that will be implemented many times. However, if you are only implementing the modification once or twice, it is too much work to create. Remember that if you use a manual change log, it is best to make the changes directly in C/SIDE.

What if?

In this section, you explore the answers to many "What if" questions. These questions usually pop up during the upgrade process, especially for difficult upgrades.

A Major Release

Now, we discuss some "What If," situations that may occur in your particular upgrade which require some additional information. First, what if this is a major release with extensive customization. Since there are more objects, there will be more chance of conflicts. Since the changes are more extensive, there is more of a chance that you will have to re-implement your customization, rather than use a change log to re-implement the upgrade.

One suggestion – in your planning, include some time to test the upgraded system, sometime after recompiling all objects but before you set the version tags, reset the modified flags, and export. Make sure that there are no obvious problems. Use the Customer License to do the test, in case there is a problem with it. Once you do the export, test the object upgrade on a copy of the complete customer database (including data).

One other possibility is that there could be a major functionality change. This usually means a change in the way something is done. In this case, the change is so major that you cannot merge the changes, either with the merge tool or even manually. This is because there is a whole new way of doing something, not just a change in the details. If this is the case, you must re-implement your customization from scratch in any object in which there is a conflict. In some cases, your customization may need to be changed even in unchanged objects, should it depend on the old way of doing things.

It is also important to re-evaluate the relevance of your customization when moving to a new version. Perhaps Microsoft Navision has added a feature that could replace all or some of your customization. Or perhaps the customer's business processes have changed. Before you upgrade, you should review the customization's functionality and discuss with your customer which parts will be upgraded.

Field Name Changes

What if you made some major field name changes? The problem here is that these changes will ripple down into every other object that refers to the changed field, at least in any automatic change log you generate. This can create huge change logs where in fact there is no actual change to many objects. For example, suppose you changed Department to Division. Consider the effect on the Customer Table. When you change the field name there, note that it has now changed automatically on the card form, on the list form, on the Customer List report, and so on, all without you having to do anything. However, if you now generate an automatic change log, it shows that the card form changed, the list form changed, the reports changed, and so on. This makes it harder for you to do upgrades.

The solution is to make the field name changes first. Before you extract the New Base objects from the new database, make your customized field name changes to the tables only. Any object that you update, be sure the Modified Flag is checked, even if you just change the table name. When you extract the new objects, you must get all objects whose Version Tag OR Modified Flag has been changed. Use the Marking Facility to do this. First, filter all objects whose Modified Flag is on, select all of them, and press CTRL+F1. Then, remove that filter, filter on all objects with the new Version Tag, select all of them, and press CTRL+F1. Now, remove that filter, use View, Marked Only, and then select all filtered objects.

Customer has Design Tools

What if my customer has Design Tools? The problem here is that your customer probably does not follow the Microsoft Navision development rules. These include rules about setting the Version Tags and Modification Flags, Internal object documentation, developing on line and not using a Project Log, so you cannot tell why they made a change.

The solution has two parts; first, better planning is necessary. You can try to teach your customer the rules, explain how it makes upgrading easier. Or, you can just get them to follow one rule – leave the Modification Flags alone. This rule is easy to follow, since it does not require them to do anything.

When you do an object upgrade, you must schedule a time to pick up the customer database from the customer site before you can upgrade. This is because your copy is not up to date. Just use a Navision Backup. Make sure when you pick up the backup from your client, you inform them that they should not make any changes until you install the upgrade. Furthermore, reiterate that any changes that they make will not be included in the upgrade. You might also want to identify a customer contact who helped with the development and who can answer any questions about the changes made.

The second part of the solution is that certain steps in the Upgrade Process will be changed. In step 1, when you make the Working Copy of the customer database, be sure to create a new database and restore the Application Objects ONLY from the Navision Backup you obtained from the customer site. After this, go through and give your customer's changed objects (the ones with the Modification flag on) their own additional Version Tag, something different from your version tag. Use new letters, based on the customer name. For example, if Acme Tools is your customer, make their Version tag AT01 (assuming this is your first upgrade), and then add an "AT01" at the end of every modified object with an existing version tag. If the customer created a new object, set the Version tag to "AT01" alone. When you reach step 6, (updating each object on the Conflicts list), take into account customer changes. At the least, you will need to generate a new change log from the Working Copy. When you have finished compiling all objects, before you export, you might want to schedule some time for your customer contact to come in and check out the upgraded system. It will be easier to see if a customer change was lost this way.

Finally, when you do export the objects, be sure to get all objects with either the New Base Version Tag, or the Customer's new Version Tag that you created. You can use the marking technique to do this. The rest of the steps are the same.

We are not Following Microsoft Navision's Methodology

What if you, the Solution Center, are not following the rules? For example, what if you went to the customer site and made some changes on the fly? The answer – see the previous section on what if the customer has the design tools.

In addition, you should start following the rules from now on. Upgrades are difficult enough when you follow the rules.

More Than One Upgrade to Do

Here is a common situation. The customer has not upgraded for a while, and you have skipped a Release and two Service Packs. Now, with the latest Release, the customer sees a feature he just has to have, and he realizes that now he wants to be upgraded. The problem, there is no direct upgrade path except from one version to the next version. You have to do multiple upgrades at once. Object Upgrades can be combined at your site. Simply do the upgrade steps 2 through 10 for each upgrade version in between. When you have upgraded to the final version, remember to export all objects with any Version Tag higher than what the customer had, so they get all changed objects.

However, Data Upgrades are usually too complex to combine. Therefore, you should schedule a separate upgrade for each Data Upgrade you need. This usually corresponds to a Release. Thus, when you combine Object Upgrades, you must still export the modified objects for each Data Upgrade. Note that technically, you could do all the Data Upgrades at once, even if they are not combined. It is highly recommended that you test thoroughly between Data Upgrades during testing if you plan to combine Data Upgrades at the customer site.

Upgrading the Data

Now, we are finally ready to upgrade the data. You usually need to do this every time there is a Release.

We include Data Conversion routines (usually codeunits) in a Upgrade Toolkit which is generally released about one month after we release the product. You should take advantage of this time lag to familiarize yourself with the new product and any oddities about it. When you get the Data Conversion routines, try not to modify them. It would be best if any data conversion routines you need could be written separately, and then run in series after ours. However, occasionally, you need to modify our Data Conversion routines because otherwise they will not work with your customizations. This is OK, but be careful. You must write (or modify) Data Conversion routines at the same time as you are creating the Upgraded Objects on your site. However, these routines must be run on the customer site, in order to upgrade their live data.

Preparation

There are some additional steps you must do to prepare for a data upgrade. First, when you make your Working Copy, make an additional copy to use to create the Phase One Conversion objects. Phase One objects must be created using the old, un-upgraded database. When you finish Upgrading the Objects, make another copy of the database, this one to be used to create the Phase Two Conversion Objects. Phase Two objects must be created on a new, upgraded database.

One other step you should do, is to create a complete copy of the customer's database. This is so you can test the upgrade process. This means you must get the customer database from the customer site so that you can have complete, current data.

The Conversion Process

The conversion process is highly dependent on the particular configuration of the system and the old and target versions involved. The Upgrade Toolkit guides you through the steps involved for each possibility.

Please note that all of these steps outlined in the Upgrade Toolkit can take a significant amount of time, especially if your customer has substantial data. You will be able to gauge the amount of time when you do your test run at your site. Be sure that the customer can be down long enough for the entire database and object upgrade. This work will probably be done only on weekends or holidays.

Conclusion

We have successfully deployed our seminar module and are prepared for the ongoing maintenance phase. Our client has identified some additional requirements which can be found in the Additional Exercises section of this manual, but our initial project is now complete.

Quick Interaction: Lessons Learned

Take a moment to write down three Key Points you have learned from this chapter:

1.

2.

3.

Microsoft Internal Use Only

Microsoft Internal Use Only

CHAPTER 11: COURSE SUMMARY

This chapter contains the following section:

- Course Summary

Course Summary

We have covered a wide range of technical and theoretical topics in this course through creating the seminar module for our client. The following is a brief summary of the topics we have covered.

Introduction

In the Introduction, we discussed the "rules," "style," and "methodology" with which you should develop solutions in Microsoft® Business Solutions–Navision®.

Business Case Diagnosis & Analysis

In this section, we discussed the business case and analyzed our client's requirements.

Managing Master Files

In Managing Master Files, we discussed the different types of triggers and how to use event triggers. We also discussed the specific triggers available for tables, forms, dataports and codeunits. In order to add more functionality to the tables and forms we were creating, we discussed complex data types, specifically the record data type, and its member functions that allow us to retrieve individual records and sets of records. Finally we looked at enabling objects for Multilanguage functionality and table relations considerations specific to Microsoft® SQL Server®.

Managing Registrations

In Managing Registrations, we discussed how to export objects as text files, modify the text files if necessary and import the files back into Microsoft Navision. We discussed how to write text messages to the user, such as error messages and confirmation questions, so that they are enabled for multilanguage functionality.

We looked at the Microsoft Navision Sales Invoice as an example of how to implement a form with a subform. We defined what a matrix form is and how it is typically used in Microsoft Navision. We then went through some of the basic steps to create a matrix form. We discussed the different types of tables in Microsoft Navision, including virtual, system and temporary tables, and how to use them.

We also discussed additional form and control functions as well as reviewing some useful functions from the Development 1 course.

Managing Posting

In Managing Posting, we began by defining Journal and Ledger Entry tables and the roles that they play in the posting process. We discussed how and in what situations you would lock tables in Microsoft Navision.

We described briefly how posting routines are written in Microsoft Navision. We then looked at the different codeunits that make up a posting routine by discussing the elements that are generally included in the Check Line, Post Line and Post Batch codeunits.

We discussed how and where to add documentation when you make changes to existing objects. We then described how transaction documents such as sales orders are posted in Microsoft Navision. We also described some of the performance issues to keep in mind while programming and how to program to keep the impact on a server and network traffic to a minimum.

After creating a posting routine for our seminar registrations, we described the Debugger and Code Coverage tools that are included with Microsoft Navision and looked at how to use them.

Managing Integration

In Managing Integration, we integrated our module elements with the standard application by adding a Seminar menu to the Navigation Pane and integrating the Navigate feature. We also discussed some guidelines for changing tables that contain data.

Managing Reporting

In Managing Reporting, before beginning development on our seminar module reports, we discussed the various report triggers and the order in which they "fire." We briefly described some of the more commonly used functions in reports. Finally, we described processing-only reports and how they are used.

Managing Statistics

In Managing Statistics, we created a seminar statistics form that summarized the total price for seminars that had been posted. Before creating this feature, we discussed the role of FlowFilters in calculation formulas.

Managing Dimensions

In Managing Dimensions, we implemented dimensions functionality in our seminar module. We then briefly discussed the Navision Developer's Toolkit.

Managing Interfaces

In our final story, Managing Interfaces, we added functionality to create e-mail messages to be sent to participants and to generate a participant list as an XML file. In doing so, we discussed how to use C/SIDE as an automation server, using custom controls and XMLPorts.

Deployment

Having completed development and testing of the seminar module, we looked at the tasks we will perform in the Deployment phase. We also discussed upgrades as part of the Ongoing Support Phase.

Microsoft Internal Use Only

CHAPTER 12: REVIEW QUESTIONS

This chapter contains the following section:

- Review Questions

Review Questions

The following questions review the topics covered in this course and will help to prepare you for the certification test in Microsoft® Business Solutions–Navision® solution development. When you have finished answering all of the questions, you can check your answers in Appendix C, Answers to Review Questions.

1. When running a report, the code in two of the following data item triggers will be executed, even if there are no records in the data item: OnPreDataItem, OnAfterGetRecord, OnPostDataItem. Which two?
2. Our client, CRONUS, has decided that they would like to use the same general product posting group for every seminar. Therefore, on the Seminar Card form, the Gen. Product Posting Group should always default to the same value for every seminar. Would you add this code to the OnInsert trigger of the Seminar table or to the OnInsertRecord trigger of the Seminar Card form?
3. You are writing a function in which you are passed a seminar number parameter called SemNum. In the function, you have a piece of code that you want to run only if a Seminar record exists where the No. is the same as SemNum. What code would you write to test whether or not the record exists?
4. You are writing code in which you want to open the Instructors form to allow the user to select one or more instructors, so that the code can make changes to those records. What function of the Form variable would you use to run the form so that a user can make a selection to be used by the code?
5. You are writing code for the Seminar Registration Header table in which, if the **Minimum Participants** value is greater than the **Maximum Participants** value, you will show an error. How would you write this error message so that when users read it, they know how to fix the error?

6. You are writing code in a function in the Seminar table with the variable NewRoomCode. You want to assign the NewRoomCode to the **Room Code** field and run the code in the OnValidate trigger for this field. What function can you use to accomplish both of these tasks in one line?
7. What type of information does a virtual table in C/SIDE typically contain?
8. Which C/AL function is used to calculate a FlowField?
9. If the **Amount** field on the Seminar Registration Line table was a SumIndexField for the table, which C/AL function would be used to calculate the sum of the Amount values on the table?
10. You are writing code in which you want to select all the Seminar Registration Header records where the **Starting Date** is between two variables called FirstDate and SecondDate. How would you define this record set using the SETRANGE function? How would you define this record set using the SETFILTER function?
11. You are writing code in which you must retrieve a record from the Seminar Registration Header table using values from the table's secondary key. Which C/AL function will you use to retrieve the record?
12. Which C/AL function unlocks a table after the LOCKTABLE function has been called?
13. If you have modified the code in an existing, standard codeunit, in which places in the codeunit will you document your changes, and what information will you include in this documentation?
14. Of the three main posting codeunits, Check Line, Post Line and Post Batch, which codeunit writes to the Ledger Entry table? Which codeunit reads the journal lines and calls the other two codeunits?
15. Why should you exercise caution when importing objects such as text files?

16. In the OnValidate trigger of the **Room Code** field on the Seminar table, you are writing code that you want run only if the value entered by the user is different from what it was before. What code would you write to compare the two values?
17. You are adding a menu item to a menu button on the Seminar Registration form to run the Seminar Charges for the seminar registration. Which properties will you set to ensure that if the user changes to a new Seminar Registration record, the Seminar Charges form will update to show the charges for the new Seminar Registration record?
18. What is the purpose of the Check Line function in a posting routine?
19. What does the RECORDLEVELLOCKING property tell you?
20. When defining the data items that will make up a report, what are two of the important factors in ensuring that the performance of the report is as efficient as possible?
21. What is the standard shortcut key to start a posting routine for a document?
22. You are writing a function that is passed a record variable (by reference) called SemRoom. In the function, you want to write code that checks the filters that the user placed on the record. Is this possible?
23. Which trigger executes when the user clicks the lookup button for a field?
24. In a function, you write the following line of code:

```
SemRegHeader . TRANSFERFIELDS ( PostedSemRegHeader ) ;
```

What does this code do?

25. In the Seminar Registration form, if the **Starting Date** is less than three days away from the work date, you want the **Starting Date** to appear in bold. What code would you write to make this happen and in what trigger would it appear?

26. You are performing data conversions into the new module. You are responsible for performing the conversion of closed seminar registration data into the Seminar Ledger Entry table. What method would you use to perform this conversion, assuming that the data has been thoroughly tested?
27. What are the three types of dimensions?
28. What property do you set to specify the values shown in a form's title bar?
29. What properties should you always set for menu buttons on a form?
30. For Multilanguage functionality what functions can you use with text constants to show the user the table or field where an error occurred?
31. What field property do you set to create a FlowFilter in a table?
32. What code would you write to set the value of "Minimum Participants" to five for all records in the Seminar table?

The client has requested that you write a new report for the seminar module. The report should list all the seminars that are going to take place between dates specified by the user on the request form. The user should also be able to specify that the list of participants in each seminar should show on the report by clicking a check box on the request form. Based on this information from the client, answer the following questions:

33. Assume that your data items are Seminar Registration Header (SemRegHeader) and Seminar Registration Line (SemRegLine), with the Seminar Registration Line data item being indented. If you call CurrReport.SKIP from the OnAfterGetRecord trigger of the SemRegLine data item, which record will the program process next? And if you call CurrReport.BREAK instead, which record will be processed?
34. Which report function would you call to stop the entire report?

35. You have added two text boxes to the request form for the user to enter the starting date and ending date for the report. In which trigger can you write code to validate the dates entered and that the ending date is after the starting date?

36. To allow the user to choose whether or not to print the list of participants in the report, you have added a check box to the request form, with its source being a variable called ShowList. You will add code to the PreSection trigger for the body section showing the registration lines. What code will you write to control whether the section prints or not?

Microsoft Internal Use Only

Microsoft Internal Use Only

CHAPTER 13: ADDITIONAL EXERCISES

This chapter contains the following sections:

- Introduction
- Adding Seminar Translations
- Adding Multiple Dimensions
- Managing Seminar Planning
- Conclusion

Microsoft Internal Use Only

Introduction

Positioning – What is the starting point?

You have finished phase one of the seminar module and now are ready to add some additional functionality.

Preconditions

All aspects of phase one have been completed and there is data in the master files.

Business Goals

By the end of this chapter, you will have created the tables and forms necessary for translating seminars, using multiple dimensions with the seminar module and managing seminar planning.

Educational Goals

By completing this chapter, you should have learned or reacquainted yourself with the following:

- Implementing standard Microsoft® Business Solutions–Navision® multi-language functionality.
- Implementing standard Microsoft Navision dimension functionality.
- Creating matrix forms.

Adding Seminar Translations

The client needs the ability to enter translations of the seminar name for Multilanguage functionality. This is a change to Use Case 4 – Managing Seminars from Managing Master Files. This is accomplished by creating a Seminar Translations form and table that is very similar to those implemented for some Microsoft Navision master files. To see an example in Microsoft Navision, view Form 30 Item Card, Form 35 Item Translations and Table 30 Item Translation.

GUI Design

Seminar Translations (Form 123456709): This form enables the entry of translations of the seminar name.

Language Code	Description
ENU	Programming

There is no navigation from this form.

Seminar Card (Form 123456700): Add 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 form 123456709 Seminar Translations for the selected entry.

Seminar List (Form 123456701): Add the same **Translations** menu item as the Seminar Card above.

Functional Design

No additional functions have been defined.

Table Design

You need to create one table for the Seminar Translations change request.

Table 123456706 Seminar Translations contains 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 A1 – Adding Seminar Translations

You must carry out the following tasks to add the seminar translations to the seminar master files.

1. Create table 123456706 Seminar Translation 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 form 123456709 Seminar Translations. 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 form 123456700 Seminar Card as shown in the GUI design.
5. Add the Translations menu item to form 123456701 Seminar List as shown in the GUI design.

Adding Multiple Dimensions

You added dimensions functionality to the seminar modules but the client would like to increase the usability of this feature by defining multiple dimensions. This is a change to Use Case 1 – Managing Dimensions in Master Files from Managing Dimensions.

To see a sample of how this works in Microsoft Navision, open the Customer Card and select **Dimensions** from the **Customer** menu button to open the Default Dimensions form. This is the same kind of dimensions functionality that you have already 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. You want to add the multiple dimensions functionality to 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 form 123456700 Seminar Card).

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

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 will be changed to open a submenu with the Dimensions – Single and Dimensions – Multiple options as shown previously for the Seminar List form.

Exercise A2 – Adding Multiple Dimensions

From the list form of a master file, like 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. You will make some modifications to the Default Dimensions-Multiple form to enable this functionality for the seminar module.

1. In form 542 Default Dimensions – Multiple, 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 more 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 that you have just created.
4. Add the dimensions menu items to the Seminar List, Seminar Room List and Instructors forms as shown in the GUI design.

Managing Seminar Planning

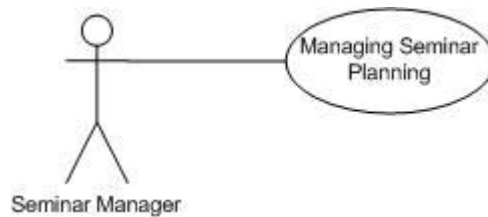
Diagnosis

You need a calendar system that gives an overview of seminar dates to help in seminar planning.

Use Cases

Based on the results of the diagnosis, we can describe this requirement with the following use case:

- Managing Seminar Planning



Analysis

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

We need a calendar system that will give us an overview of our seminar dates to help in seminar planning. We 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.

Using this information, we can further define how the management of the seminar planning overview will be reflected in the program we are now creating.

Purpose

A seminar planning overview will provide a summary of all scheduled seminars registered for any given period of time.

Preconditions

Calendar date information and seminar registration information must exist.

Postconditions

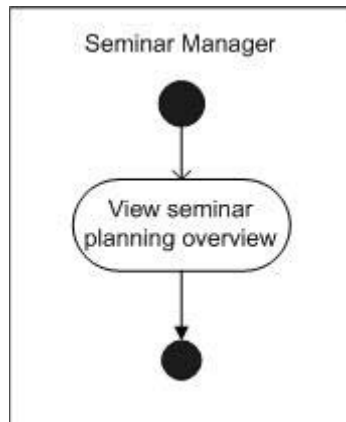
A seminar planning overview form will exist.

Main Scenario

When seminar managers want to gain an overview of which seminars are scheduled for certain dates, they will 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).

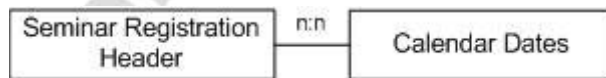
The program will use the company's Base Calendar (found by clicking GENERAL LEDGER→SETUP→BASE CALENDAR) to schedule the seminar dates so they are not held on weekends or on scheduled holidays.

Activity Diagram



Managing Seminar Planning Overview – Design

The processes and activities for this use case are quite simple, but the interaction of the data that we will use to achieve this planning overview is somewhat complex. The following diagram shows the relationship of the seminar registrations to the dates that we want to show in the form:



The seminar planning overview will not be a table of information but only 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, we require a "matrix" type representation of the data.

For a review of matrix forms, please see Chapter 4.

GUI Design

You only need to create one form for the Seminar Planning overview, which should appear as shown in the following screenshot.

Seminar Planning Form (Form 123456733):

No.	Seminar ...	Seminar Name	01.16.01	01.17.01	01.18.01
▶ REG00001	SEM0002	Solution Development	1	1	1
REG00002	SEM0001	Programming	1		
REG00003	SEM0002	Solution Development	1	1	1

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

The buttons along the bottom will be 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

You need the following function for the Seminar Planning overview:

CheckDate: A function that helps ensure that you do not schedule seminars for "nonworking" days like 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 return true or false depending on the answer.

GetSelectionFilter: You need a function 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 the following screenshot:



You then want the GetSelectionFilter function to produce the string 'ROOM01..ROOM03.' On the other hand, if the user had selected only the first and third records, you would want the string to be 'ROOM01|ROOM03.'

Table Design

To make this form work, create a Seminar Registration Buffer table. This will be a simple table in which you can temporarily store information for this form.

Table 123456730 Seminar Registration Buffer 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 A3 – Creating the Seminar Planning Form

1. Begin by creating form 123456733 Seminar Planning as a basic form with a matrix box.
 - The source table for the form will be 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 make things simple, copy these buttons from a standard form. You can use form 157 Item Availability by Periods for this purpose. After you have copied them and pasted them to your 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 create it yourself. 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 will be 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 you want the starting date of each period 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 will be 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 that we calculate 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 A4 – Adding Code to the Seminar Planning Form

You first need to make some changes to existing forms.

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 what you created for the GetSelectionFilter function in the Seminar Room List form, except that you use different variables 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 (shown in the table design) as a temporary table. You need a temporary table because you do not want to fill a table with permanent data every time the seminar planning overview is used. Instead, you want a table that is filled and emptied with the use of the form.

- 3. Create table 123456730 Seminar Registration Buffer 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 your Seminar Planning form.

Now that you have created the temporary table and declared it as a variable, create the code in the Seminar Planning form that will bring the form to life.

- 4. To begin with, you need to 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 and 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 in the PeriodFormManagement codeunit (returns an integer).

6. In the code for the matrix box, set the value for the MatrixHeader variable. You do this in the OnAfterGetRecord trigger of the matrix box. For this, use the function CreatePeriodFormat from the standard codeunit PeriodFormManagement, which will return the value for the MatrixHeader. As parameters, this function expects a period type and a date. In this case, the period type will be that which was selected by the user (with the **Trendscape option** buttons), and the date parameter will be 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 your code by running the form and testing the period option buttons.

You now need a function to check whether dates are "working" or "nonworking", as described in the functional design.

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 table 7601 Base Calendar Changes, to see whether the date is a "Nonworking" date. The function will get the Base Calendar Code from the Company Information table and find 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, you want the buffer table to be filled with the dates on which the seminars will run, the buffer 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.

You now need to 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 your Seminar Planning form.

Conclusion

In this section, you added some additional functionality to the seminar module to conform with Microsoft Navision standards for dimensions and multilanguage. You also created a seminar planning overview that brings together information from our seminar registrations and the system's virtual date table.

Microsoft Internal Use Only

APPENDIX A: SAMPLE REPORTS

This appendix contains samples of the following reports:

- Sample Participant List
- Sample Certificate Confirmation

Microsoft Internal Use Only

Sample Participant List

Seminar Registration - Participant List
CRONUS International Ltd.

May 23, 2004
Page 1

No. REG00001
Seminar Code SEM0002
Seminar Name Solution Development
Starting Date 01.08.03
Duration 10
Instructor Name Annette Hill
Room Name Room 1

Bill-to Customer No.	Participant Contact No.	Participant Name
10000	CT100140	David Hodgson
10000	CT100156	John Emory
10000	CT000001	Mindy Martin
10000	CT100210	Stephanie Bourne
30000	CT200080	Pamela Ansmann-Wolfe
30000	CT200079	Tina Gorenc

Sample Certificate Confirmation



Participant Certificate

David Hodgson
has participated in seminar
Solution Development
on January 8, 2004

Annette Hill

Microsoft Internal Use Only

Microsoft Internal Use Only

APPENDIX B: SAMPLE XML PARTICIPANT LIST

This appendix contains an example of the output from the XML Sem. Reg.-Participant List report.

Microsoft Internal Use Only

Sample XML Sem. Reg.-Participant List

```
- <Seminar_Registration_-_Participant_List>
- <Seminar>
  <No>REG00001</No>
  <Seminar_Code>SEM0002</Seminar_Code>
  <Seminar_Name>Solution Development</Seminar_Name>
  <Starting_Date>01.08.01</Starting_Date>
  <Duration>10</Duration>
  <Instructor_Name>Annette Hill</Instructor_Name />
  <Room_Name>Room 1</Room_Name>
- <Participant>
  <Customer_No>10000</Customer_No>
  <Contact_No>CT100140</Contact_No>
  <Name>David Hodgson</Name>
</Participant>
- <Participant>
  <Customer_No>10000</Customer_No>
  <Contact_No>CT100156</Contact_No>
  <Name>John Emory</Name>
</Participant>
- <Participant>
  <Customer_No>10000</Customer_No>
  <Contact_No>CT000001</Contact_No>
  <Name>Mindy Martin</Name>
</Participant>
- <Participant>
  <Customer_No>10000</Customer_No>
  <Contact_No>CT100210</Contact_No>
  <Name>Stephanie Bourne</Name>
</Participant>
</Seminar>
</Seminar_Registration_-_Participant_List>
```

APPENDIX C: USING C/Front

This appendix contains the following sections:

- Introduction
- Using C/Front
- Two Interfaces – DLL and OCX
- Accessing Data From the Database Using C/Front and Microsoft® Visual Basic®
- Limitations of C/Front

Microsoft Internal Use Only

Introduction

This appendix provides information on C/Front. However, this course is not a substitute for reading the C/Front Reference Guide. For information on C/Front installation and setup, please refer to the C/Front Reference Guide or the Microsoft® Business Solutions–Navision® Development I manual.

Using C/Front

C/Front is an application programming interface that can be used to access a C/SIDE database. C/Front facilitates high-level interaction with the C/SIDE database manager, and allows C developers to manipulate any C/SIDE database.

The central component of C/Front is a library of C functions. These functions give you access to every aspect of data storage and maintenance, and allow you to integrate both standard and custom applications with your C/SIDE database.

Since C/Front is a front end to Microsoft® Business Solutions–Navision® Attain, you may want to use C/Front to add, delete or view data in your own applications. Furthermore, you may need to create and manipulate table objects or to find out information from tables, which is something you cannot do when using the C/ODBC driver. C/Front is much faster than using the C/ODBC driver and is as close to the native connection as you can get with Microsoft Navision Attain. However, in order to use C/Front, you must use some other programming language such as Delphi, Kylix, C#, C++, C, or Visual Basic. C/Front provides two methods of accessing Microsoft Navision Attain – either through a C style DLL or through the use of the OCX interface. Our examples will use Microsoft® Visual Basic® 6.0 and the OCX.

Two Interfaces – DLL and OCX

Differences Between the C/Front API DLL and OCX

There are a number of differences between the C/Front API DLL and the OCX. The C/Front API DLL is primarily intended for C programmers, but can be used with any language and compiler that can load and use DLLs and that use the _cdecl calling convention. The OCX is primarily used for gaining easy access to Navision Attain databases from environments like Microsoft® Excel and Microsoft® Word or Microsoft Visual Basic.

The C/Front Reference Guide is really intended for the DLL and for the C programmer but can be used with the OCX as well. It provides some background information, and contains a considerable number of examples. If you are acquainted with the DLL, it is recommended that you read the online Help for a function before you use it through the OCX – there are a number of differences. Furthermore, some functions in the DLL have no equivalent in the OCX, and there are a couple of new functions in the OCX. You should also note that errors are handled differently. For example, you cannot install your own error or message handler to replace the default one when you use the OCX.

General Differences

- Where all the functions in the DLL are named DBL_*, the corresponding methods in the OCX do not use the DBL_ prefix. For example, the method that corresponds to the DBL_OpenDatabase function is OpenDatabase.
- The GetLastErrorCode in the DLL has an equivalent in the LastError method in the OCX.
- The Field_2_Str function in the DLL has an equivalent in the FieldToStr method in the OCX.
- Closing dates cannot be used in the OCX. If a closing date is retrieved from the database, it will be considered a normal date, and there is no way to put a closing date into the database from the OCX.
- There are no hundredths of seconds in Time values in the OCX.
- The Allow function is replaced by four methods – AllowKeyNotFound, AllowRecordExists, AllowRecordNotFound and AllowTableNotFound.

Functions Without Equivalents

There is no longer any need for the conversion functions, because functions such as GetFieldData and AssignField use the Variant data type. The following functions therefore have no equivalents in the OCX:

- Alpha_2_Str
- BCD_2_Str
- Date_2_Str
- Date_2_YMD
- HMST_2_Time
- Str_2_Alpha
- Str_2_BCD
- Str_2_Date
- Str_2_Time
- Time_2_HMST
- Time_2_Str
- YMD_2_Date

Error handling is different in the OCX (see Error Handling in the C/Front manual) and you cannot replace the default error or message handler with a function of your own. However, we recommend that you do make sure that you include plenty of error checking because the majority of errors that occur happen because the wrong values have been set. The following functions have therefore no equivalents in the OCX:

- SetExceptionHandler
- SetMessageShowHandler

When the OCX is used, there is no need for explicit initialization and deinitialization. Therefore, the following functions have no equivalents in the OCX:

- Exit
- Init

The following two functions have been omitted because the changed functionality of GetFieldData makes them less necessary and also because the environments from where the OCX will typically be used do not support the use of pointers to the same degree as C does.

- GetFieldDataAddr
- GetFieldDataSize

The following function is still retained in the DLL in order to make it easier to port applications from the C-Toolkit for Navision 3.XX. This function is not needed in the OCX.

- FieldDataOffset

Accessing Data From the Database Using C/Front in Visual Basic

The following example that we review, assumes that you want to open a predefined database, the customer table, and that you wish to display the customers in a list form. The example does not explain how to use Visual Basic.

1. Open Visual Basic and choose the type of project (Standard.Exe) and then click **Open**.
2. Select Project1 inside the Project Window and right-click and select **Rename**. Rename your project to Cfront_Test.

3. Click on the Form1 in the Project Window and then select the properties. Set the following properties:
Name = MainForm
Height = 6450
Width = 6650
Startup Position = 2 - CenterScreen
4. Click **Projects** and select components or use the shortcut key sequence of (Ctrl + T). Next, select Cfront OLE Control Module, Microsoft Common Dialog SP3, and Microsoft FlexGrid Control 6.0 (SP3). Make sure that there is a check mark in the box beside each component name.
5. Declare some variables that we will need. Click **View** from the main menu and select **Code**. The code window appears. Select (General) from the left drop-down list and (Declarations) from the right drop-down list and then create the following public variables.

```
Public NavisionPath As String
Public LicenseFileName As String
Public DatabaseFileName As String
Public CompanyName As String
Public UserId As String
Public TableNo As Long
Public TableHandle As Long
Public hRec As Long
Public GridRow As Integer
Public GridCol As Integer
```

6. We need to create our own menu. Click **Tools** from the main menu and select Menu Editor or press CTRL + E. Add the following menu options – File, Open, Exit, View, and Write with Open and Exit indented under file. Furthermore, note the each menu option must have a caption and a name. Therefore, give the menu options previously described and create the captions as shown in the picture above.
7. Double-click on the **OCX** icon in the Components Window, this places the OCX (CFront) on the form. Select the **OCX** icon and move it up to the top left of the form. Next, set the following properties:
Visible = No
(Name) = Cfront

NOTE: You may want to shrink the size of the icon to where it's no larger than the command button.

8. Double-click on the CommonDialog icon in the Components Window; this places the CommonDialog component icon on the form. Select the CommonDialog component icon and move it up to the top of the form on the right side of the Cfront icon. Change the name property from CommonDialog1 to CommonDialog.
9. Double-click on the MSFlexGrid icon in the components Window, this places the MSFlexGrid component on the form. Set the following properties:

```
(Name) = TableBox
Top = 520
Height = 5655
Width = 6495
Rows = 0
TabIndex = 0
```

10. Now it's time to add some code. Select **File** and then **Open** from the menu that we've created. This should automatically open the Code window and create a procedure Open_Click. Add the following code to the Open_Click procedure.

```
'If connecting to a Navision database
DriverName = "NDBCN"
ServerName = ""
NetType = "tcp"
CacheSize = 1000
UseCommitCache = 0
NTAuthentication = 0
UserId = ""
Password = ""
'If connecting to a SQL Server database
'DriverName = "NDBCS"
'NetType = "Named Pipes"
'CacheSize = 0
'UseCommitCache = 0
'NTAuthentication = 1 'Yes
```

NOTE: If you are using Microsoft® SQL Server® you still have to enter a zero value for CacheSize or UseCommitCache even though they only apply to the Navision Server.

```
CommonDialog.ShowOpen
DatabaseFileName = CommonDialog.FileName
NavisionPath = Mid(CommonDialog.FileName, 1,
InStr(1, CommonDialog.FileName,
CommonDialog.FileTitle, vbTextCompare) - 2)
DatabaseFileName = CommonDialog.FileTitle

'Set the license file to be in the location where
'the database is located
LicenseFileName = NavisionPath + "\fin.flf"
CFRONT.SetNavisionPath NavisionPath
CFRONT.LoadLicenseFile LicenseFileName
CFRONT.CheckLicenseFile (9110)
Call
CFRONT.ConnectServerAndOpenDatabase(DriverName,
ServerName, NetType, DatabaseFileName, CacheSize,
UseCommitCache, NTAuthentication, UserId,
Password)
CompanyName = "CRONUS International Ltd."
CFRONT.OpenCompany CompanyName
```

The driver name is very important. Please note the code above uses NDBCN for the Microsoft Navision server and NDBCS for the SQL Server. If the open/connect operation fails, the function raises an exception that terminates the application. Please note that only one database/server connection can be open at a time.

Only one company can be open at a time. You must open a company before the application can access the data in the database tables. However, you can have any number of tables within a single company open at the same time. A table is identified by a unique number; therefore, when a table is opened, the database manager returns a unique handle, which remains valid until the table is closed. This handle must be passed to all operations that are carried out on the table. Note the code above also checks to ensure that the license file includes permissions for C/FRONT. If not, an error message appears.

Next, in the left drop-down list of the Code Window, select Exit and add the following code:

```
If TableHandle <> 0 Then
    CFront.CloseTable (TableHandle)
End If
If CompanyName <> "" Then
    CFront.CloseCompany
End If
If Connected Then
    CFront.DisconnectServer
End If
CFront.ReleaseAllObjects
MainForm.Hide
```

Note that we must put some error checking so that we do not receive errors when trying to exit our application if we have not opened the company or viewed the records. If you do not have the code for the TableHandle, you receive Error 12000 – Invalid Handle. If you do not have the code to check for the CompanyName, you receive Error 1046 No Company Selected. Finally, notice that after disconnecting from the server, make sure that you release all objects to ensure that you've freed any memory used by C/FRONT.

11. Next, in the left-hand drop-down list of the Code Window select **View** and add the following code:

```
Dim CurRow As Long
TableNo = 23 'Vendor
If Not MainForm.CFront.OpenTable(TableHandle,
TableNo)
Then
    MsgBox "Unable to Open Table 23 – Vendor."
    Return
End If
```

If the table number is not found, an error message that the table cannot be found will be displayed and then you will receive Error 1001 – Table does not exist.

```

GridRow = 0
NumColumns = 1
FieldNo = MainForm.CFRONT.NextField(TableHandle,
0)
While FieldNo > 0
    NumColumns = NumColumns + 1
    TableBox.Cols = NumColumns
    TableBox.Row = GridRow
    TableBox.Col = NumColumns - 1
    TableBox.Text =
MainForm.CFRONT.FieldName(TableHandle, FieldNo)
    FieldNo = MainForm.CFRONT.NextField(TableHandle,
FieldNo)
Wend
'load the data
hRec = MainForm.CFRONT.AllocRec(TableHandle)
If MainForm.CFRONT.FindRec(TableHandle, hRec, "-")
Then
    GridRow = 1
    Do
        RecFields = "" & GridRow
        GridRow = GridRow + 1
        FieldNo =
MainForm.CFRONT.NextField(TableHandle,0)
        While FieldNo > 0
            RecFields = RecFields & Chr(9) &
MainForm.CFRONT.FieldToStr(TableHandle, hRec,
FieldNo)
            FieldNo =
MainForm.CFRONT.NextField(TableHandle, FieldNo)
        Wend
        TableBox.AddItem RecFields
        Loop Until MainForm.CFRONT.NextRec(TableHandle,
hRec, 1) = 0
    End If
MainForm.CFRONT.FreeRec hRec

```

If you use a row value that does not exist, you receive error message 3009 – Invalid Row Value.

Writing Data Back to the Database Using C/FRONT in Microsoft Excel

Microsoft Navision Attain comes with an excellent example that demonstrates both reading and writing to the Budget Table. Furthermore, the example shows you how you can create your own error routine, unlike the OCX example that we used in Visual Basic, which is not allowed to have a custom error routine.

Limitations of C/FRONT

- Windows do not go in to standby or hibernation if there is an open server connection from C/FRONT.
- Entries made by C/FRONT will not fire triggers to execute any code validation.
- C/FRONT is very particular and looks in different places at different times for a license. Check and make sure you have the license loaded everywhere it needs to be, that is client, server, and so on.
- Besides the limitations, a common misconception that many have is that C/FRONT can be used in place of multiple sessions. However, this is not the case. Every connection in C/FRONT is treated as a session.
- As with regular Microsoft Navision, the client and ODBC, the versions of C/FRONT must match the version of the SERVER that you are trying to connect to.
- Keys can be active or inactive. You cannot activate keys from within C/FRONT; therefore, only active keys are available to C/FRONT.

APPENDIX D: ANSWERS TO REVIEW QUESTIONS

This appendix contains answers to the Review Questions to Chapter 12.

Microsoft Internal Use Only

Answers to Review Questions

1. OnPreDataItem and OnPostDataItem.
2. OnInsert of the table.
3. The code will be as follows:

```
IF Seminar.GET(SeminarNo) THEN...
```

4. The code will be as follows:

```
Form.RUNMODAL
```

5. The error will appear as follows:

```
ERROR('%1 cannot be greater than %2.',FIELDNAME("Minimum  
Participants"), FIELDNAME("Maximum Participants"));
```

6. VALIDATE.
7. A virtual table contains information provided by the system.
8. CALCFIELDS.
9. CALCSUMS.
10. The SETRANGE will appear as follows:

```
SETRANGE("Starting Date", FirstDate, SecondDate);
```

The SETFILTER will appear as follows:

```
SETFILTER("Starting Date", '%1..%2', FirstDate, SecondDate);
```

11. FIND.
12. COMMIT.
13. In the codeunit's Documentation trigger, you will include the date, your initials, a reference number and a brief description of the changes made. At the position where you made the changes, you will mark the beginning and the end of the changes with your initials and the reference number from the Documentation trigger.
14. Post Line; Post Batch.

15. You must exercise caution when importing objects as text files because the program does not run a check to compare the imported object with the existing object. Therefore, the Import Worksheet will not open when importing these files, and the old objects will be automatically overwritten.

16. The code will appear as follows:

```
IF xRec."Room Code" <> Rec."Room Code" THEN...
```

17. RunFormLink and RunFormLinkType.

18. The Check Line codeunit checks the validity of the Journal Line that is passed to it. It checks whether the line is empty, and it checks that certain key fields like the **Posting Date** and **Document No.** are not blank.

19. If the RECORDLEVELLOCKING property is TRUE, it means that the SQL Server Option is being used.

20. When writing a report, you should be careful to create data items from the appropriate tables and in the right order, to use the appropriate keys and to filter the data items properly.

21. F11.

22. Yes, because the record variable represents the records as well as the filters and key from the associated table.

23. OnLookup.

24. Transfers the fields from the Seminar Registration Header record to the Posted Seminar Registration Header record where the field numbers and types are the same between the two records.

25. The following code will appear in the OnFormat trigger:

```
IF ("Starting Date" <= WORKDATE + 3) THEN  
    CurrForm."Starting Date".UPDATEFONTBOLD;
```

26. You would write a dataport to import the data into the Seminar Journal Line table and use it to call the Seminar-Post codeunit.

27. Global, Shortcut, and Budget.

28. DataCaptionFields.

29. HorzGlue and VertGlue.

30. TABLECAPTION and FIELDCAPTION.

31. FieldClass = FlowFilter.

32. Seminar.MODIFYALL("Minimum Participants",5);

33. If CurrReport.SKIP is called, the next record in the Seminar Registration Line data item will be processed. If CurrReport.BREAK is called, the next record in the Seminar Registration Header data item will be processed.
34. CurrReport.QUIT.
35. OnPreReport.
36. CurrReport.SHOWOUTPUT(ShowList);

Microsoft Internal Use Only

INDEX

- ActiveX..... 240
- Analysis Phase 4
- Application Designer's Guide..... 3, 11, 109, 240
- ApplicationHandler, Navision Attain..... 247
- Audience, Target..... 2
- Automation data type 247
- Automation Data Type 239
- Automation Server, Using an 239
- Base Calendar Changes (Table) 302
- BREAK, Using the CurrReport Function 162
- Breakpoint on Triggers..... 107
- Breakpoints 107
- Business Case, Executive Summary 8
- C/Front®..... 201
- C/SIDE Reference Guide..... 3, 4, 19, 64, 65
- CALCFIELDS, Using the Method..... 24
- CaptionML..... 24, 59
- Certificate Confirmation,
 Creating the Report for 175
- Certificate Confirmation,
 Use Case 2—Managing 173
 Analysis 173
 Design 174
 Development and Testing 175
- Certification, Navision Solution Development 3
- Check Line Codeunit..... 97
- Client Monitor 109
- CLOSE, Using the CurrForm Function 64
- Code Comments 104
- Code Coverage..... 108
- Codeunit Triggers 19
 Documentation 19
 OnRun 19
- COM..... See Component Object Model
- Comment Line (Table) 32, 44
- COMMIT Function..... 103, 105, 106, 322
 Using the 105
- Complex Data Types 20
- Component Object Model (COM) 239
- Configuration Checklist 258
- Confirmation
 Certificate See Certificate Confirmation
 E-mail See E-mail Confirmation
- Contact Card (Form) 28
- Contact List (Form) 28
- Content, Course..... 3
- COPY, Using the Function..... 24
- COUNT, Using the Function 24
- Courseware..... 3
- Create Seminar Invoices (Report) 178, 233
- CREATETOTALS,
 Using the CurrReport Function 162
- Custom Controls, Using 240
- Customer Card (Form)..... 34, 123, 124, 125, 132
- Data Conversion..... 258
- Data Model 7, 11, 311
- Data Types
 Complex..... 20
 Record 20
- Dataport Triggers..... 260
 OnAfterExportRecord 260
 OnAfterImportRecord..... 260
 OnBeforeExportRecord 260
 OnInitDataPort 260
 OnPostDataItem 260
 OnPostDataport 260
 OnPreDataport 260
- Date Virtual Table..... 62
- Debugger 107
- Debugging Tools 107
- Default Dimension (Table)..... 206
- Defining a Record Set 22
- DELETE, Using the Function..... 23
- Demonstration Data..... 3
- Deployment Phase 4, 257
- Design Phase 4
- Developer's Toolkit 201
- Development & Testing Phase..... 4
- Diagnostic Phase..... 4
- DimensionManagement (Codeunit)..... 97, 206,
 207, 208, 213, 221
- Dimensions..... 199
 Budget..... 199
 Global..... 199
 Shortcut..... 199
 Tables 200
- Dimensions in Invoicing,
 Modifying the Report for 233
- Dimensions in Master Files
 Modifying Tables and Forms for 208
 Modifying the
 DimensionManagement Codeunit for 207
- Dimensions in Registration
 Modifying Tables for 214
 Modifying the Forms for 219
 Modifying the Tables and Forms
 in Posted Seminar Registrations for 221
- Dimensions, Story 7—Managing
 Diagnosis 203
 Review Questions..... 235
 Use Cases 203
- Dimensions,
 Use Case 1—Managing in Master Files 204
 Analysis..... 204
 Design 205
 Development & Testing 206
- Dimensions,
 Use Case 2—Managing in Registration..... 210
 Analysis..... 210

Design.....	211	Seminar Registration.....	9
Dimensions,		Seminars.....	9
Use Case 3–Managing in Seminar Posting.	223	GET, Using the Function.....	21
Analysis.....	223	Go Live.....	261
Design.....	224	Historical Data, Importing.....	259
Development & Testing.....	226	Implementation Methodology.....	4
Dimensions,		Import Contact (Dataport).....	260
Use Case 4–Managing in Invoicing.....	231	Importing Document and Ledger Data.....	259
Analysis.....	231	INIT, Using the Function.....	23
Design.....	232	InMatrix Property.....	61
Development & Testing.....	233, 234	InMatrixHeading Property.....	61
Document Dimension (Table).....	200, 213, 218	INSERT, Using the Function.....	23
Document Posting Routines.....	101	Instructor (Table).....	40, 206
Documentation		Instructors	
Code Comments.....	104	Creating Tables and Forms for.....	40
In Existing Objects.....	104	Functional Requirements.....	9
Documentation Trigger.....	17, 18, 19, 104, 322	Instructors (Form).....	40, 293
Duration, Course.....	3	Instructors, Use Case 3–Managing.....	38
EDITABLE		Analysis.....	38
Using the CurrForm Function.....	65	Design.....	39
Using the Form Control Function.....	65	Development & Testing.....	40
E-mail Confirmation, Use Case 1–Managing.	245	Integration, Story 4–Managing	
Analysis.....	245	Diagnosis.....	146
Design.....	246	Review Questions.....	156
Development & Testing.....	247	Use Cases.....	146
Errors		Interfaces, Functional Requirements.....	10
Program Logic.....	107	Interfaces, Story 8–Managing	
Runtime.....	107	Diagnosis.....	244
Syntax.....	107	Review Questions.....	254
Event Triggers.....	17	Use Cases.....	244
Exporting Objects as Text Files.....	57	Invoice Posting, Creating the Report for.....	178
Extended Text Header (Table).....	32, 37, 44	Invoice Posting, Use Case 3–Managing.....	176
Extended Text Line (Table).....	32, 44	Analysis.....	176
FIELDCAPTION function.....	58, 59	Design.....	177
FIELDCAPTION Function.....	58, 59	Development & Testing.....	178
FIND, Using the Function.....	21	Invoicing, Functional Requirements.....	10
FlowFields, Creating FlowFields for Sums.....	191	Item Availability by Periods (Form).....	299
FlowFields, Use Case 1–Managing.....	189	Job Jnl.-Post Line (Codeunit)..	127, 133, 135, 229
Development & Testing.....	191	Job Journal Line (Table).....	118
FlowFilters, Using.....	187	Job Ledger Entry (Table).....	118
Form and Control Functions, Additional.....	64	Journal Tables.....	95
Form Triggers		Keys and Queries, Performance Issues with ..	106
OnActivateForm.....	19	Ledger Data, Importing.....	259
OnAfterGetCurrRecord.....	19	Ledger Entry Tables.....	95
OnAfterGetRecord.....	19	LOCKTABLE Function.....	105, 106, 284
OnBeforePutRecord.....	19	Using the.....	105
OnCloseForm.....	19	Main Menu.....	147
OnDeactivateForm.....	19	Maintain Relationships Setting.....	25
OnDeleteRecord.....	19	MAPIHandler.....	247, 248
OnFindRecord.....	19	Master Data, Importing.....	259
OnInit.....	19	Master Files, Story 1–Managing	
OnInsertRecord.....	19	Diagnosis.....	26
OnModifyRecord.....	19	Review Questions.....	52
OnOpenForm.....	19	Use Cases.....	26
Function Trigger.....	17	Matrix Forms.....	60
Functional Requirements.....	9	MatrixBox Form Control.....	60, 61
Instructors.....	9	Methodology, Implementation.....	4
Interfaces.....	10	Microsoft SQL Server, Table Relations.....	25
Invoicing.....	10	MODIFY, Using the Function.....	23
Participants.....	9	Modifying the Database Record.....	23
Reporting and Statistics.....	10		

- Multilanguage Functionality 24, 52, 58
 In Objects 24
 In Text Messages 58
 Navigate 146, 152, 154
 Navigate (Form) 150, 154
 Navigate Integration, Modifying Objects for 154
 Navigate Integration, Use Case 2–Managing . 152
 Analysis 152
 Design 152
 Development & Testing 154, 155
 Navision Developer's Toolkit, Using 201
 NEWPAGE, Using the CurrReport Function ... 162
 NEXT, Using the Function 21
 NoSeriesManagement (Codeunit) . 46, 47, 48, 82,
 83, 84, 86, 137
 Objectives, Training 2
 OCX
 Controls, Using 240
 OnActivateForm Trigger 19
 OnAfterExportRecord Trigger 260
 OnAfterGetCurrRecord Trigger 19, 60
 OnAfterGetRecord Trigger 19, 60, 163,
 183, 283, 286, 302
 OnAfterImportRecord Trigger 260
 OnBeforeExportRecord Trigger 260
 OnBeforeImportRecord Trigger 260
 OnBeforePutRecord Trigger 19, 60
 OnCloseForm Trigger 19
 OnDelete Trigger 18, 23, 240
 OnDeleteRecord Trigger 19
 OnFindRecord Trigger 19, 60, 301
 On-going Support Phase 4
 OnInit Trigger 19
 OnInitDataPort Trigger 260
 OnInitReport Trigger 161, 183
 OnInsert Trigger 17, 18, 19, 23, 240, 283, 322
 OnInsertRecord Trigger 19, 283
 OnLookup Trigger 19, 323
 OnModify Trigger 18, 19, 23, 240
 OnModifyRecord Trigger 19
 OnOpenForm Trigger 19
 OnPostDataItem Trigger . 161, 183, 260, 283, 322
 OnPostReport Trigger 161
 OnPreDataItem Trigger 161, 162, 183,
 260, 283, 322
 OnPreDataport Trigger 260
 OnPreReport Trigger 161, 183, 324
 OnPreSection Trigger 162
 OnRename Trigger 18, 19, 23
 OnRun Trigger 19, 97, 226, 227, 229
 OnValidate Trigger 19, 284, 285
 OnValidate Ttrigger 24
 PAGENO, Using the CurrReport Function 162
 Participant List Report
 Creating the Report for 168
 XML See XML Participant List
 Participant List Reporting,
 Use Case 1–Managing 165
 Analysis 165
 Design 166
 Development & Testing 167
 Participants, Functional Requirements 9
 Participants, Use Case 4–Managing 27
 Analysis 27
 Design 28
 Development & Testing 28
 Performance Issues 105
 Keys and Queries 106
 Reducing Impact on Network Traffic 106
 Reducing Impact on the Server 106
 PeriodFormManagement (Codeunit) 301, 302
 Post Batch Codeunit 98
 Post Line Codeunit 97
 Posted Seminar Charge (Table) 118
 Posted Seminar Charges (Form) 114
 Posted Seminar Reg. Header (Table) ... 118, 213
 Posted Seminar Reg. Line (Table) 213
 Posted Seminar Reg. List (Form) 116
 Posted Seminar Reg. Subform (Form) ... 115, 212
 Posted Seminar Registration (Form) 115,
 153, 212
 Posting
 Invoice See Invoice Posting
 Seminar Registration See Seminar
 Registration Posting
 Posting Routines 96
 Companion Codeunits 96
 Document 101
 Standardized Object Names 96
 Posting, Story 3–Managing
 Diagnosis 110
 Review Questions 141
 Use Cases 110
 PREVIEW, Using the CurrReport Function 162
 ProcessingOnly Property 163
 Processing-only Reports 163
 Program Logic Errors 107
 Project Evaluation 261
 Project Plan 12
 QUIT, Using the CurrReport Function 162
 READPERMISSION property 154
 Record Data Type 20
 Defining a Record Set 22
 Modifying 23
 Retrieving a Record 21
 Record Set, Defining a 22
 RECORDLEVELLOCKING Property 105
 Reducing Impact on Network Traffic 106
 Reducing Impact on the Server 106
 Registrations, Story 2–Managing
 Diagnosis 66, 291, 293, 294
 Review Questions 91
 Use Cases 66, 295
 RENAME, Using the Function 23
 Report Functions 162
 Report Triggers 161
 Reporting, Functional Requirements 10
 Reporting, Story 5–Managing
 Diagnosis 164

Review Questions	183	Creating the Tables and Forms for.....	74
Use Cases.....	164	Functional Requirements	9
Requirements		Seminar Registration (Form).....	71, 117, 167, 246
Functional.....	9	Seminar Registration Buffer (Table).....	298
Other	10	Seminar Registration Form (Form).....	212
Retrieving a Record	21	Seminar Registration Header (Table).....	73,
Review Questions		167, 213
For Story 1–Managing Master Files.....	52	Seminar Registration Line (Table).....	73, 213
For Story 2–Managing Registrations	91	Seminar Registration List (Form)	73, 117, 246
For Story 3–Managing Posting	141	Seminar Registration Posting	
For Story 4–Managing Integration	156	Creating the Document	
For Story 5–Managing Reporting	183	Posting Codeunits for	132
For Story 6–Managing Statistics.....	195	Creating the Seminar Journal Posting	
For Story 7–Managing Dimensions	235	Codeunits and Form for.....	123
For Story 8–Managing Interfaces	254	Creating the Tables and Forms for.....	119
Rooms		Seminar Registration Posting,	
Adding Code for	35	Use Case 2–Managing.....	111
Creating Tables and Forms for	32	Analysis	111
Rooms, Use Case 2–Managing.....	30	Design	112
Analysis.....	30	Seminar Registration Subform (Form)	71, 212
Design	31	Seminar Registration-Printed (Codeunit)	168
Development & Testing.....	32	Seminar Registrations	
Rules.....	4	Adding Code for Line Table and Form	87
Runtime Errors.....	107	Adding Code to the Header Table and Form	82
Handling.....	109	Seminar Registrations,	
Seminar (Table).....	44, 45	Use Case 1–Managing	67
Seminar Card (Form).....	43, 148, 190, 291	Analysis	67
Seminar Charge (Table).....	73	Design	69
Seminar Charges (Form).....	71	Seminar Registrations,	
Seminar Charges, Adding Code for	80	Use Case 1–Managing	
Seminar Comment Line (Table).....	74	Development & Testing	73, 89
Seminar Comment List (Form)	70	Seminar Report Selection (Form).....	166
Seminar Comment Sheet (Form).....	70	Seminar Report Selections (Table).....	167
Seminar Document-Print (Codeunit).....	172	Seminar Room Card (Form).....	31
Seminar Feature Integration,		Seminar Room List (Form)	32, 293
Use Case 1–Managing	147	Seminar Setup (Form).....	43
Analysis.....	147	Seminar Setup (Table)	44, 45
Design.....	147	Seminar Statistics (Form).....	190
Seminar Jnl.-Check Line (Codeunit).....	124, 125,	Seminar Translations (Form).....	291
.....	225, 226, 227, 228	Seminar Translations (Table).....	292
Seminar Jnl.-Post Line (Codeunit) .	125, 133, 136,	SeminarMail (Codeunit).....	247
.....	225, 227, 230	Seminar-Post (Codeunit).....	132, 228
Seminar Journal Line (Table)	118, 225	Seminar-Post (Yes/No) (Codeunit).....	138, 139
Seminar Ledger Entries (Form)	113, 153, 225	Seminars	
Seminar Ledger Entry (Table)	118	Creating the Tables and Rooms for	44
Seminar List (Form)	44, 148, 191, 291, 293	Creating the Tables and Rooms for	292
Seminar Menu (Form).....	149, 166, 251	Functional Requirements	9
Seminar Planning		Seminars, Use Case 5–Managing.....	41
Adding Code to the	300	Analysis	41
Creating the Form for	299	Design	42
Seminar Planning Form (Form)	297	Development & Testing	44
Seminar Planning Overview,		SETCURRENTKEY, Using the Function.....	22
Use Case 2–Managing		SETFILTER, Using the Function.....	22
Analysis.....	295	SETRANGE, Using the Function.....	22
Design.....	296	SHOWOUTPUT,	
Seminar Reg.-Part. Certificate (Report).....	175	Using the CurrReport Function.....	162
Seminar Reg.-Participant List (Report).....	168	SKIP, Using the CurrReport Function	162
Seminar Register (Table).....	118	Source Code Setup (Form)	113, 119
Seminar Registers (Form)	114	Source Code Setup (Table).....	118, 119
Seminar Registration		SQL Server	

- Table Relations on Microsoft SQL Server 25
- Statistics
- Creating the Form for Seminar 192
 - Functional Requirements 10
 - Story 6–Managing
 - Diagnosis 188
 - Use Cases 188
 - Statistics, Story 6–Managing
 - Review Questions 195
 - Statistics, Use Case 2–Managing Seminar
 - Analysis 189
 - Design 190
 - Steering Committee Approval 258
 - Structure, Course 2
 - Support Phase 4
 - Syntax Errors 107
 - Table Locking 105
 - Table Relations on Microsoft SQL Server 25
 - TableBox Control 60
 - TABLECAPTION Function 59
 - Temporary Tables 63
 - Text Files, Exporting Objects as 57
 - Text Messages,
 - Using Multilanguage Functionality in 58
 - TOTALSCAUSED BY,
 - Using the CurrReport Function 162
 - Transaction Documents, Importing 259
- Triggers
- Codeunit
 - Documentation 19
 - OnRun 19
 - Codeunit 19
 - Dataport 260
 - OnAfterExportRecord 260
 - OnAfterImportRecord 260
 - OnBeforeExportRecord 260
 - OnInitDataPort 260
 - OnPostDataItem 260
 - OnPostDataport 260
 - OnPreDataport 260
 - Documentation 17, 18, 19, 104
 - Documentation 322
 - Form
 - OnActivateForm 19
 - OnAfterGetCurrRecord 19
 - OnAfterGetRecord 19
 - OnBeforePutRecord 19
 - OnCloseForm 19
 - OnDeactivateForm 19
 - OnDeleteRecord 19
 - OnFindRecord 19
 - OnInit 19
 - OnInsertRecord 19
 - OnModifyRecord 19
 - OnOpenForm 19
 - Report 161
 - Table Event 18, 240
 - OnDelete 18, 240
 - OnInsert 18, 240
 - OnModify 18, 240
 - OnRename 18
 - Table Field Event
 - OnLookup 19
 - OnValidate 19
 - Triggers, Event 17
 - UPDATE, Using the CurrForm Function 64
 - UPDATEEDITABLE, Using the Function 65
 - Use Cases
 - For Story 1–Managing Master Files 26
 - For Story 2–Managing Registrations 66, 295
 - For Story 3–Managing Posting 110
 - For Story 4–Managing Integration 146
 - For Story 5–Managing Reporting 164
 - For Story 6–Managing Statistics 188
 - For Story 7–Managing Dimensions 203
 - For Story 8–Managing Interfaces 244
 - VALIDATE, Using the Function 24
 - Virtual Tables 61
 - VISIBLE, Using the Function 65
 - What You Need to Know About
 - Automation Server, Using an 239
 - Check Line Codeunit 97
 - Complex Data Types 20
 - Debugging Tools 107
 - Defining a Record Set 22
 - Document Posting Routines 101
 - Documentation in Existing Objects 104
 - Event Triggers 17
 - Exporting Objects as Text Files 57
 - FlowFilters, Using 187
 - Importing Document and Ledger Data 259
 - Journal and Ledger Entry Tables 95
 - Matrix Forms 60
 - Multilanguage Functionality in Objects 24
 - Multilanguage Functionality in Text Messages 58
 - Performance Issues 105
 - Post Batch Codeunit 98
 - Post Line Codeunit 97
 - Posting Routines 96
 - Processing-only Reports 163
 - Record Data Type 20
 - Report Functions 162
 - Report Triggers 161
 - Retrieving a Record 21
 - Table Event Triggers 18
 - Table Locking 105
 - temporary tables 63
 - Virtual Tables 61
 - XML Participant List,
 - Use Case 2–Managing 250
 - Analysis 250
 - Design 251
 - Development & Testing 252
 - XML Sem. Reg.-
 - Participant List (Report) 252, 309, 310

Microsoft Internal Use Only