

Course 80437:

**C/SIDE Solution Development in
Microsoft Dynamics® NAV 2013**

This courseware is provided "as-is". Information and views expressed in this courseware, including URL and other Internet Web site references, may change without notice.

Unless otherwise noted, the examples depicted herein are provided for illustration only and are fictitious. No real association or connection is intended or should be inferred.

This courseware does not provide you with any legal rights to any intellectual property in any Microsoft product. Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this courseware may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means or for any purpose, without the express written permission of Microsoft Corporation.

Copyright © 2012 Microsoft Corporation. All rights reserved.

Microsoft®, Microsoft Dynamics®, Microsoft® PowerPoint®, Microsoft® SQL Server® data management software and Microsoft Dynamics® NAV are trademarks of the Microsoft group of companies. All other trademarks are property of their respective owners.

MICROSOFT LICENSE TERMS
MICROSOFT INSTRUCTOR-LED COURSEWARE

These license terms are an agreement between Microsoft Corporation (or based on where you live, one of its affiliates) and you. Please read them. They apply to your use of the content accompanying this agreement which includes the media on which you received it, if any. These license terms also apply to Trainer Content and any updates and supplements for the Licensed Content unless other terms accompany those items. If so, those terms apply.

**BY ACCESSING, DOWNLOADING OR USING THE LICENSED CONTENT, YOU ACCEPT THESE TERMS.
IF YOU DO NOT ACCEPT THEM, DO NOT ACCESS, DOWNLOAD OR USE THE LICENSED CONTENT.**

If you comply with these license terms, you have the rights below for each license you acquire.

1. DEFINITIONS.

- a. "Authorized Learning Center" means a Microsoft IT Academy Program Member, Microsoft Learning Competency Member, or such other entity as Microsoft may designate from time to time.
- b. "Authorized Training Session" means the instructor-led training class using Microsoft Instructor-Led Courseware conducted by a Trainer at or through an Authorized Learning Center.
- c. "Classroom Device" means one (1) dedicated, secure computer that an Authorized Learning Center owns or controls that is located at an Authorized Learning Center's training facilities that meets or exceeds the hardware level specified for the particular Microsoft Instructor-Led Courseware.
- d. "End User" means an individual who is (i) duly enrolled in and attending an Authorized Training Session or Private Training Session, (ii) an employee of a MPN Member, or (iii) a Microsoft full-time employee.
- e. "Licensed Content" means the content accompanying this agreement which may include the Microsoft Instructor-Led Courseware or Trainer Content.
- f. "Microsoft Certified Trainer" or "MCT" means an individual who is (i) engaged to teach a training session to End Users on behalf of an Authorized Learning Center or MPN Member, and (ii) currently certified as a Microsoft Certified Trainer under the Microsoft Certification Program.
- g. "Microsoft Instructor-Led Courseware" means the Microsoft-branded instructor-led training course that educates IT professionals and developers on Microsoft technologies. A Microsoft Instructor-Led Courseware title may be branded as MOC, Microsoft Dynamics or Microsoft Business Group courseware.
- h. "Microsoft IT Academy Program Member" means an active member of the Microsoft IT Academy Program.
- i. "Microsoft Learning Competency Member" means an active member of the Microsoft Partner Network program in good standing that currently holds the Learning Competency status.
- j. "MOC" means the "Official Microsoft Learning Product" instructor-led courseware known as Microsoft Official Course that educates IT professionals and developers on Microsoft technologies.
- k. "MPN Member" means an active silver or gold-level Microsoft Partner Network program member in good standing.

- I. "Personal Device" means one (1) personal computer, device, workstation or other digital electronic device that you personally own or control that meets or exceeds the hardware level specified for the particular Microsoft Instructor-Led Courseware.
 - m. "Private Training Session" means the instructor-led training classes provided by MPN Members for corporate customers to teach a predefined learning objective using Microsoft Instructor-Led Courseware. These classes are not advertised or promoted to the general public and class attendance is restricted to individuals employed by or contracted by the corporate customer.
 - n. "Trainer" means (i) an academically accredited educator engaged by a Microsoft IT Academy Program Member to teach an Authorized Training Session, and/or (ii) a MCT.
 - o. "Trainer Content" means the trainer version of the Microsoft Instructor-Led Courseware and additional supplemental content designated solely for Trainers' use to teach a training session using the Microsoft Instructor-Led Courseware. Trainer Content may include Microsoft PowerPoint presentations, trainer preparation guide, train the trainer materials, Microsoft One Note packs, classroom setup guide and Pre-release course feedback form. To clarify, Trainer Content does not include any software, virtual hard disks or virtual machines.
- 2. USE RIGHTS.** The Licensed Content is licensed not sold. The Licensed Content is licensed on a *one copy per user basis*, such that you must acquire a license for each individual that accesses or uses the Licensed Content.

2.1 Below are five separate sets of use rights. Only one set of rights apply to you.

a. **If you are a Microsoft IT Academy Program Member:**

- i. Each license acquired on behalf of yourself may only be used to review one (1) copy of the Microsoft Instructor-Led Courseware in the form provided to you. If the Microsoft Instructor-Led Courseware is in digital format, you may install one (1) copy on up to three (3) Personal Devices. You may not install the Microsoft Instructor-Led Courseware on a device you do not own or control.
- ii. For each license you acquire on behalf of an End User or Trainer, you may either:
 - 1. distribute one (1) hard copy version of the Microsoft Instructor-Led Courseware to one (1) End User who is enrolled in the Authorized Training Session, and only immediately prior to the commencement of the Authorized Training Session that is the subject matter of the Microsoft Instructor-Led Courseware being provided, **or**
 - 2. provide one (1) End User with the unique redemption code and instructions on how they can access one (1) digital version of the Microsoft Instructor-Led Courseware, **or**
 - 3. provide one (1) Trainer with the unique redemption code and instructions on how they can access one (1) Trainer Content,
- iii. **provided you comply with the following:**
 - iii. you will only provide access to the Licensed Content to those individuals who have acquired a valid license to the Licensed Content,
 - iv. you will ensure each End User attending an Authorized Training Session has their own valid licensed copy of the Microsoft Instructor-Led Courseware that is the subject of the Authorized Training Session,
 - v. you will ensure that each End User provided with the hard-copy version of the Microsoft Instructor-Led Courseware will be presented with a copy of this agreement and each End User will agree that their use of the Microsoft Instructor-Led Courseware will be subject to the terms in this agreement prior to providing them with the Microsoft Instructor-Led Courseware. Each individual will be required to denote their acceptance of this agreement in a manner that is enforceable under local law prior to their accessing the Microsoft Instructor-Led Courseware,
 - vi. you will ensure that each Trainer teaching an Authorized Training Session has their own valid licensed copy of the Trainer Content that is the subject of the Authorized Training Session,

- vii. you will only use qualified Trainers who have in-depth knowledge of and experience with the Microsoft technology that is the subject of the Microsoft Instructor-Led Courseware being taught for all your Authorized Training Sessions,
 - viii. you will only deliver a maximum of 15 hours of training per week for each Authorized Training Session that uses a MOC title, and
 - ix. you acknowledge that Trainers that are not MCTs will not have access to all of the trainer resources for the Microsoft Instructor-Led Courseware.
- b. **If you are a Microsoft Learning Competency Member:**
- i. Each license acquired on behalf of yourself may only be used to review one (1) copy of the Microsoft Instructor-Led Courseware in the form provided to you. If the Microsoft Instructor-Led Courseware is in digital format, you may install one (1) copy on up to three (3) Personal Devices. You may not install the Microsoft Instructor-Led Courseware on a device you do not own or control.
 - ii. For each license you acquire on behalf of an End User or Trainer, you may either:
 1. distribute one (1) hard copy version of the Microsoft Instructor-Led Courseware to one (1) End User attending the Authorized Training Session and only immediately prior to the commencement of the Authorized Training Session that is the subject matter of the Microsoft Instructor-Led Courseware provided, **or**
 2. provide one (1) End User attending the Authorized Training Session with the unique redemption code and instructions on how they can access one (1) digital version of the Microsoft Instructor-Led Courseware, **or**
 3. you will provide one (1) Trainer with the unique redemption code and instructions on how they can access one (1) Trainer Content,
- provided you comply with the following:**
- iii. you will only provide access to the Licensed Content to those individuals who have acquired a valid license to the Licensed Content,
 - iv. you will ensure that each End User attending an Authorized Training Session has their own valid licensed copy of the Microsoft Instructor-Led Courseware that is the subject of the Authorized Training Session,
 - v. you will ensure that each End User provided with a hard-copy version of the Microsoft Instructor-Led Courseware will be presented with a copy of this agreement and each End User will agree that their use of the Microsoft Instructor-Led Courseware will be subject to the terms in this agreement prior to providing them with the Microsoft Instructor-Led Courseware. Each individual will be required to denote their acceptance of this agreement in a manner that is enforceable under local law prior to their accessing the Microsoft Instructor-Led Courseware,
 - vi. you will ensure that each Trainer teaching an Authorized Training Session has their own valid licensed copy of the Trainer Content that is the subject of the Authorized Training Session,
 - vii. you will only use qualified Trainers who hold the applicable Microsoft Certification credential that is the subject of the Microsoft Instructor-Led Courseware being taught for your Authorized Training Sessions,
 - viii. you will only use qualified MCTs who also hold the applicable Microsoft Certification credential that is the subject of the MOC title being taught for all your Authorized Training Sessions using MOC,
 - ix. you will only provide access to the Microsoft Instructor-Led Courseware to End Users, and
 - x. you will only provide access to the Trainer Content to Trainers.

c. **If you are a MPN Member:**

- i. Each license acquired on behalf of yourself may only be used to review one (1) copy of the Microsoft Instructor-Led Courseware in the form provided to you. If the Microsoft Instructor-Led Courseware is in digital format, you may install one (1) copy on up to three (3) Personal Devices. You may not install the Microsoft Instructor-Led Courseware on a device you do not own or control.
 - ii. For each license you acquire on behalf of an End User or Trainer, you may either:
 1. distribute one (1) hard copy version of the Microsoft Instructor-Led Courseware to one (1) End User attending the Private Training Session, and only immediately prior to the commencement of the Private Training Session that is the subject matter of the Microsoft Instructor-Led Courseware being provided, **or**
 2. provide one (1) End User who is attending the Private Training Session with the unique redemption code and instructions on how they can access one (1) digital version of the Microsoft Instructor-Led Courseware, **or**
 3. you will provide one (1) Trainer who is teaching the Private Training Session with the unique redemption code and instructions on how they can access one (1) Trainer Content,
- provided you comply with the following:**
- iii. you will only provide access to the Licensed Content to those individuals who have acquired a valid license to the Licensed Content,
 - iv. you will ensure that each End User attending an Private Training Session has their own valid licensed copy of the Microsoft Instructor-Led Courseware that is the subject of the Private Training Session,
 - v. you will ensure that each End User provided with a hard copy version of the Microsoft Instructor-Led Courseware will be presented with a copy of this agreement and each End User will agree that their use of the Microsoft Instructor-Led Courseware will be subject to the terms in this agreement prior to providing them with the Microsoft Instructor-Led Courseware. Each individual will be required to denote their acceptance of this agreement in a manner that is enforceable under local law prior to their accessing the Microsoft Instructor-Led Courseware,
 - vi. you will ensure that each Trainer teaching an Private Training Session has their own valid licensed copy of the Trainer Content that is the subject of the Private Training Session,
 - vii. you will only use qualified Trainers who hold the applicable Microsoft Certification credential that is the subject of the Microsoft Instructor-Led Courseware being taught for all your Private Training Sessions,
 - viii. you will only use qualified MCTs who hold the applicable Microsoft Certification credential that is the subject of the MOC title being taught for all your Private Training Sessions using MOC,
 - ix. you will only provide access to the Microsoft Instructor-Led Courseware to End Users, and
 - x. you will only provide access to the Trainer Content to Trainers.

d. **If you are an End User:**

For each license you acquire, you may use the Microsoft Instructor-Led Courseware solely for your personal training use. If the Microsoft Instructor-Led Courseware is in digital format, you may access the Microsoft Instructor-Led Courseware online using the unique redemption code provided to you by the training provider and install and use one (1) copy of the Microsoft Instructor-Led Courseware on up to three (3) Personal Devices. You may also print one (1) copy of the Microsoft Instructor-Led Courseware. You may not install the Microsoft Instructor-Led Courseware on a device you do not own or control.

e. **If you are a Trainer.**

- i. For each license you acquire, you may install and use one (1) copy of the Trainer Content in the form provided to you on one (1) Personal Device solely to prepare and deliver an Authorized Training Session or Private Training Session, and install one (1) additional copy on another Personal Device as a backup copy, which may be used only to reinstall the Trainer Content. You may not install or use a copy of the Trainer Content on a device you do not own or control. You may also print one (1) copy of the Trainer Content solely to prepare for and deliver an Authorized Training Session or Private Training Session.

- ii. You may customize the written portions of the Trainer Content that are logically associated with instruction of a training session in accordance with the most recent version of the MCT agreement. If you elect to exercise the foregoing rights, you agree to comply with the following: (i) customizations may only be used for teaching Authorized Training Sessions and Private Training Sessions, and (ii) all customizations will comply with this agreement. For clarity, any use of "customize" refers only to changing the order of slides and content, and/or not using all the slides or content, it does not mean changing or modifying any slide or content.

2.2 Separation of Components. The Licensed Content is licensed as a single unit and you may not separate their components and install them on different devices.

2.3 Redistribution of Licensed Content. Except as expressly provided in the use rights above, you may not distribute any Licensed Content or any portion thereof (including any permitted modifications) to any third parties without the express written permission of Microsoft.

2.4 Third Party Programs and Services. The Licensed Content may contain third party programs or services. These license terms will apply to your use of those third party programs or services, unless other terms accompany those programs and services.

2.5 Additional Terms. Some Licensed Content may contain components with additional terms, conditions, and licenses regarding its use. Any non-conflicting terms in those conditions and licenses also apply to your use of that respective component and supplements the terms described in this agreement.

3. LICENSED CONTENT BASED ON PRE-RELEASE TECHNOLOGY. If the Licensed Content's subject matter is based on a pre-release version of Microsoft technology ("Pre-release"), then in addition to the other provisions in this agreement, these terms also apply:

- a. **Pre-Release Licensed Content.** This Licensed Content subject matter is on the Pre-release version of the Microsoft technology. The technology may not work the way a final version of the technology will and we may change the technology for the final version. We also may not release a final version. Licensed Content based on the final version of the technology may not contain the same information as the Licensed Content based on the Pre-release version. Microsoft is under no obligation to provide you with any further content, including any Licensed Content based on the final version of the technology.
- b. **Feedback.** If you agree to give feedback about the Licensed Content to Microsoft, either directly or through its third party designee, you give to Microsoft without charge, the right to use, share and commercialize your feedback in any way and for any purpose. You also give to third parties, without charge, any patent rights needed for their products, technologies and services to use or interface with any specific parts of a Microsoft software, Microsoft product, or service that includes the feedback. You will not give feedback that is subject to a license that requires Microsoft to license its software, technologies, or products to third parties because we include your feedback in them. These rights survive this agreement.
- c. **Pre-release Term.** If you are an Microsoft IT Academy Program Member, Microsoft Learning Competency Member, MPN Member or Trainer, you will cease using all copies of the Licensed Content on the Pre-release technology upon (i) the date which Microsoft informs you is the end date for using the Licensed Content on the Pre-release technology, or (ii) sixty (60) days after the commercial release of the technology that is the subject of the Licensed Content, whichever is earliest ("Pre-release term"). Upon expiration or termination of the Pre-release term, you will irretrievably delete and destroy all copies of the Licensed Content in your possession or under your control.

- 4. SCOPE OF LICENSE.** The Licensed Content is licensed, not sold. This agreement only gives you some rights to use the Licensed Content. Microsoft reserves all other rights. Unless applicable law gives you more rights despite this limitation, you may use the Licensed Content only as expressly permitted in this agreement. In doing so, you must comply with any technical limitations in the Licensed Content that only allows you to use it in certain ways. Except as expressly permitted in this agreement, you may not:
- access or allow any individual to access the Licensed Content if they have not acquired a valid license for the Licensed Content,
 - alter, remove or obscure any copyright or other protective notices (including watermarks), branding or identifications contained in the Licensed Content,
 - modify or create a derivative work of any Licensed Content,
 - publicly display, or make the Licensed Content available for others to access or use,
 - copy, print, install, sell, publish, transmit, lend, adapt, reuse, link to or post, make available or distribute the Licensed Content to any third party,
 - work around any technical limitations in the Licensed Content, or
 - reverse engineer, decompile, remove or otherwise thwart any protections or disassemble the Licensed Content except and only to the extent that applicable law expressly permits, despite this limitation.
- 5. RESERVATION OF RIGHTS AND OWNERSHIP.** Microsoft reserves all rights not expressly granted to you in this agreement. The Licensed Content is protected by copyright and other intellectual property laws and treaties. Microsoft or its suppliers own the title, copyright, and other intellectual property rights in the Licensed Content.
- 6. EXPORT RESTRICTIONS.** The Licensed Content is subject to United States export laws and regulations. You must comply with all domestic and international export laws and regulations that apply to the Licensed Content. These laws include restrictions on destinations, end users and end use. For additional information, see www.microsoft.com/exporting.
- 7. SUPPORT SERVICES.** Because the Licensed Content is "as is", we may not provide support services for it.
- 8. TERMINATION.** Without prejudice to any other rights, Microsoft may terminate this agreement if you fail to comply with the terms and conditions of this agreement. Upon termination of this agreement for any reason, you will immediately stop all use of and delete and destroy all copies of the Licensed Content in your possession or under your control.
- 9. LINKS TO THIRD PARTY SITES.** You may link to third party sites through the use of the Licensed Content. The third party sites are not under the control of Microsoft, and Microsoft is not responsible for the contents of any third party sites, any links contained in third party sites, or any changes or updates to third party sites. Microsoft is not responsible for webcasting or any other form of transmission received from any third party sites. Microsoft is providing these links to third party sites to you only as a convenience, and the inclusion of any link does not imply an endorsement by Microsoft of the third party site.
- 10. ENTIRE AGREEMENT.** This agreement, and any additional terms for the Trainer Content, updates and supplements are the entire agreement for the Licensed Content, updates and supplements.
- 11. APPLICABLE LAW.**
- a. United States. If you acquired the Licensed Content in the United States, Washington state law governs the interpretation of this agreement and applies to claims for breach of it, regardless of conflict of laws principles. The laws of the state where you live govern all other claims, including claims under state consumer protection laws, unfair competition laws, and in tort.

- b. Outside the United States. If you acquired the Licensed Content in any other country, the laws of that country apply.
- 12. LEGAL EFFECT.** This agreement describes certain legal rights. You may have other rights under the laws of your country. You may also have rights with respect to the party from whom you acquired the Licensed Content. This agreement does not change your rights under the laws of your country if the laws of your country do not permit it to do so.
- 13. DISCLAIMER OF WARRANTY. THE LICENSED CONTENT IS LICENSED "AS-IS" AND "AS AVAILABLE." YOU BEAR THE RISK OF USING IT. MICROSOFT AND ITS RESPECTIVE AFFILIATES GIVES NO EXPRESS WARRANTIES, GUARANTEES, OR CONDITIONS. YOU MAY HAVE ADDITIONAL CONSUMER RIGHTS UNDER YOUR LOCAL LAWS WHICH THIS AGREEMENT CANNOT CHANGE. TO THE EXTENT PERMITTED UNDER YOUR LOCAL LAWS, MICROSOFT AND ITS RESPECTIVE AFFILIATES EXCLUDES ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT.**
- 14. LIMITATION ON AND EXCLUSION OF REMEDIES AND DAMAGES. YOU CAN RECOVER FROM MICROSOFT, ITS RESPECTIVE AFFILIATES AND ITS SUPPLIERS ONLY DIRECT DAMAGES UP TO US\$5.00. YOU CANNOT RECOVER ANY OTHER DAMAGES, INCLUDING CONSEQUENTIAL, LOST PROFITS, SPECIAL, INDIRECT OR INCIDENTAL DAMAGES.**

This limitation applies to

- anything related to the Licensed Content, services, content (including code) on third party Internet sites or third-party programs; and
- claims for breach of contract, breach of warranty, guarantee or condition, strict liability, negligence, or other tort to the extent permitted by applicable law.

It also applies even if Microsoft knew or should have known about the possibility of the damages. The above limitation or exclusion may not apply to you because your country may not allow the exclusion or limitation of incidental, consequential or other damages.

Please note: As this Licensed Content is distributed in Quebec, Canada, some of the clauses in this agreement are provided below in French.

Remarque : Ce le contenu sous licence étant distribué au Québec, Canada, certaines des clauses dans ce contrat sont fournies ci-dessous en français.

EXONÉRATION DE GARANTIE. Le contenu sous licence visé par une licence est offert « tel quel ». Toute utilisation de ce contenu sous licence est à votre seule risque et péril. Microsoft n'accorde aucune autre garantie expresse. Vous pouvez bénéficier de droits additionnels en vertu du droit local sur la protection dues consommateurs, que ce contrat ne peut modifier. La ou elles sont permises par le droit locale, les garanties implicites de qualité marchande, d'adéquation à un usage particulier et d'absence de contrefaçon sont exclues.

LIMITATION DES DOMMAGES-INTÉRÊTS ET EXCLUSION DE RESPONSABILITÉ POUR LES DOMMAGES. Vous pouvez obtenir de Microsoft et de ses fournisseurs une indemnisation en cas de dommages directs uniquement à hauteur de 5,00 \$ US. Vous ne pouvez prétendre à aucune indemnisation pour les autres dommages, y compris les dommages spéciaux, indirects ou accessoires et pertes de bénéfices.

Cette limitation concerne:

- tout ce qui est relié au le contenu sous licence, aux services ou au contenu (y compris le code) figurant sur des sites Internet tiers ou dans des programmes tiers; et.
- les réclamations au titre de violation de contrat ou de garantie, ou au titre de responsabilité stricte, de négligence ou d'une autre faute dans la limite autorisée par la loi en vigueur.

Elle s'applique également, même si Microsoft connaissait ou devrait connaître l'éventualité d'un tel dommage. Si votre pays n'autorise pas l'exclusion ou la limitation de responsabilité pour les dommages indirects, accessoires ou de quelque nature que ce soit, il se peut que la limitation ou l'exclusion ci-dessus ne s'appliquera pas à votre égard.

EFFET JURIDIQUE. Le présent contrat décrit certains droits juridiques. Vous pourriez avoir d'autres droits prévus par les lois de votre pays. Le présent contrat ne modifie pas les droits que vous confèrent les lois de votre pays si celles-ci ne le permettent pas.

Revised September 2012

Table of Contents

Introduction

Microsoft Dynamics Courseware Overview.....	0-2
Student Objectives.....	0-3

Module 1: DATA AND PROCESS MODEL

Lesson 1: Table Types and Characteristics	1-3
Lesson 2: Standard Data Model.....	1-16
Lesson 3: Standard Process Model	1-26

Module 2: MASTER TABLES AND PAGES

Lesson 1: Prerequisite Knowledge	2-2
Lesson 2: Participants.....	2-3
Lesson 3: Instructors and Rooms.....	2-7
Lab 2.1: Customize Resource Tables and Pages	2-11
Lesson 4: Seminars.....	2-20
Lab 2.2: Creating Seminar Tables and Pages	2-30

Module 3: DOCUMENTS

Lesson 1: Prerequisite Knowledge	3-2
Lesson 2: Registrations	3-13
Lab 3.1: Importing, Reviewing and Completing Seminar Registration Tables.....	3-22
Lesson 3: Reviewing the Table Code.....	3-39
Lab 3.2: Create Seminar Registration Pages.....	3-53

Module 4: POSTING

Lesson 1: Prerequisite Knowledge	4-3
Lesson 2: Posting Seminar Registrations	4-20
Lab 4.1: Reviewing and Completing the Journal and Ledger Tables.....	4-29
Lab 4.2: Creating Codeunits and Pages for Seminar Journal Posting	4-41
Lab 4.3: Creating the Tables and Pages for Posted Registration Information	4-61
Lab 4.4: Modifying Tables, Pages, and Codeunits for Resource Posting.....	4-67
Lab 4.5: Creating the Codeunits for Document Posting	4-72

Module 5: FEATURE INTEGRATION

Lesson 1: Prerequisite Knowledge	5-2
Lesson 2: Seminar Feature Integration.....	5-5
Lab 5.1: Integrating Seminar Features.....	5-11
Lesson 3: Navigate Integration	5-14
Lab 5.2: Changing Objects to Integrate with Navigate	5-24

Module 6: REPORTING

Lesson 1: Prerequisite Knowledge	6-2
Lesson 2: Reporting Lab Overview	6-6
Lesson 3: Participant List Reporting	6-7
Lab 6.1: Creating the Seminar Participant List	6-11
Lesson 4: Invoice Posting Batch Job	6-23
Lab 6.2: Creating the Invoice Posting Batch Job	6-24
Lesson 5: Test Your Knowledge	6-35

Table of Contents

Module 7: STATISTICS

Lesson 1: Prerequisite Knowledge	7-2
Lesson 2: Seminar Statistics	7-3
Lab 7.1: Creating FlowFields for Sums.....	7-5
Lab 7.2: Creating the Seminar Statistics Page	7-7

Module 8: DIMENSIONS

Lesson 1: Prerequisite Knowledge	8-2
Lesson 2: Integrating Seminar Management with Dimensions	8-21
Lab 8.1: Integrating with Dimension Management	8-26

Module 9: ROLE TAILORING

Lesson 1: Prerequisite Knowledge	9-2
Lesson 2: Seminar Manager Role Center	9-13
Lab 9.1: Create the Seminar Manager Role Center	9-16
Lesson 3: MenuSuite Object Type.....	9-28
Lesson 4: Seminar Management Department Page	9-41
Lab 9.2: Create Seminar Management Department Page.....	9-44

Module 10: INTERFACES

Lesson 1: Prerequisite Knowledge	10-2
Lesson 2: Email Confirmation	10-13
Lab 10.1: Create Email Confirmations	10-17

Module 11: WEB SERVICES

Lesson 1: Prerequisite Knowledge	11-2
Lesson 2: Registration Web Service.....	11-11
Lab 11.1: Creating a Web Service	11-15
Lab 11.2: Create a Windows Forms Application to Test the Web Service.....	11-21

Module 12: TESTING AND DEBUGGING

Lesson 1: Prerequisite Knowledge	12-2
Lesson 2: Testing Seminar Management	12-32
Lab 12.1: Create Seminar Management Unit Tests	12-40
Lesson 3: Debugging	12-48

Module 13: SQL SERVER OPTIMIZATION

Lesson 1: SQL Server for Microsoft Dynamics NAV	13-2
Lesson 2: Representation of Microsoft Dynamics NAV Tables and Indexes in SQL Server.....	13-4
Lesson 3: Collation Options.....	13-6
Lesson 4: SQL Server Query Optimizer	13-8
Lesson 5: Optimizing a Microsoft Dynamics NAV Application.....	13-14
Lesson 6: Data Access Redesign	13-27
Lesson 7: C/AL Database Functions and Performance on SQL Server	13-29
Lesson 8: Bulk Inserts.....	13-34
Lesson 9: Locking, Blocking, and Deadlocks.....	13-36

Table of Contents

Lesson 10: SIFT Data Storage in SQL Server.....	13-43
Lesson 11: SQL Server Profiler.....	13-45
Lab 13.1: Analyze Index Usage.....	13-51
Lab 13.2: Optimize C/AL Code.....	13-63

Module 14: APPENDIX

Lesson 1: CRONUS International Ltd.	14-2
Lesson 2: Functional Requirements.....	14-4
Lesson 3: Content Structure	14-7
Lab 14.1: Function Testing.....	14-9

C/SIDE Solution Development in Microsoft Dynamics® NAV 2013

We created this additional Table of Contents to assist you in quickly finding out the areas that are new and, or changed from the Microsoft Dynamics NAV earlier version for this course.

These areas are identified with this icon  throughout the training material.

Module 8: DIMENSIONS

Topic 1: Dimension Types	8-4
Topic 3: Dimensions Data Model	8-8

Module 10: INTERFACES

Topic 1: Component Object Model (COM) Technologies	10-2
Topic 1: Solution Design	10-13

Module 11: WEB SERVICES

Topic 5: OData Web Services.....	11-8
---	------

Module 12: TESTING AND DEBUGGING

Topic 5: Transaction Model for Test Functions	12-7
Topic 8: Testing Pages	12-21
Lesson 3: Debugging	12-48

INTRODUCTION

Training is an important component of maintaining the value of a Microsoft Dynamics® investment. Quality training from industry experts keeps you up-to-date and helps you develop the skills necessary for fully maximizing the value of your solution. Microsoft Dynamics provides different kinds of training to meet everyone's needs, from online training, classroom training, or training materials. Select the training type that will best help you stay ahead of the competition.

Online Training

Online training delivers convenient, detailed training in the comfort of your own home or office. Online training provides immediate access to training 24 hours a day. It is perfect for the customer who does not have the time or budget to travel. Online training options combine the efficiency of online training with the thorough product coverage of classroom training.

Classroom Training

Classroom training provides, comprehensive learning through hands-on interaction. From demonstrations to presentations to classroom activities, you receive practical experience with instruction from our certified staff of experts.

Training Materials

Training materials help you learn at your own pace, in your own time, with information-packed training manuals. The many training manuals features many tips, tricks, and insights that you can reference continuously.

Microsoft Dynamics Courseware

The Microsoft Dynamics courseware consists of detailed training manuals that are designed from a training perspective. These manuals include advanced topics, in addition to training objectives, exercises, interactions, and quizzes.

Look for a complete list of manuals that are available for purchase on CustomerSource or PartnerSource.

Microsoft Dynamics Courseware Contents

Microsoft Dynamics courseware contains labs and quick interactions. These help

Lab

Within the Microsoft Dynamics training materials, you will find labs. These labs are typically offered in two levels to accommodate each student's variety of knowledge and expertise. We suggest that you try the High level steps first. If you need help completing the task, look to the information in the Detailed steps.

High level steps

High levels steps are the most challenging. These steps are designed for the experienced student who requires little instruction to complete the required task.

Detailed steps

Detailed steps are geared toward new users who require detailed instructions and explanations to complete the lab. Detailed steps guide you through the whole task. This includes navigation.

What's New Icon

This training material might include content for new features that is specific to this software version, and to any updated features. To assist in finding the content for the new features, an icon () is placed next to the heading. The icon identifies areas that are new and, or changed from the earlier version. However, it is important to review all content to make sure there is a thorough understanding of this information.

Student Objectives

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

List three main objectives here.

1.

2.

3.

MODULE 1: DATA AND PROCESS MODEL

Module Overview

Companies use Microsoft Dynamics NAV 2013 to manage their business operations and processes, such as creating and posting sales invoices, controlling the inventory, handling purchases, and so on. Microsoft Dynamics NAV 2013 rich functionality covers the following business processes:

- Financial Management
- Sales
- Marketing
- Purchases
- Warehouse
- Manufacturing
- Resource Planning
- Jobs
- Service Management
- Human Resources

However, most companies have very specific needs that are not covered by the Microsoft Dynamics NAV 2013 standard application. The gaps between the functionality that Microsoft Dynamics NAV 2013 provides and a company's needs can be as small as a simple change to an existing process, or as large as development of a whole new application area.

All application functionality in Microsoft Dynamics NAV 2013 is developed in Microsoft Dynamics NAV Development Environment and in C/AL programming language. Application functionality is contained in the objects that are stored in the database. As a developer, you can change any standard object or change the code. This lets you develop rich and powerful customizations, but also makes it easy to introduce bugs or cause the existing application to stop functioning as expected.

To develop solutions for Microsoft Dynamics NAV 2013, you must understand how the standard application works. You must become familiar with the basic principles that are consistently applied throughout Microsoft Dynamics NAV 2013 and across all its application areas.

C/SIDE Solution Development in Microsoft Dynamics® NAV 2013

Microsoft Dynamics NAV 2013 is consistent and intuitive in how it presents data and manages processes. When you customize Microsoft Dynamics NAV 2013, you may introduce new data structures and new processes. If you do not apply the principles from the standard application, you can easily produce an application that is difficult to use, maintain, and upgrade. Users should be unable to differentiate between standard processes and your custom processes. If your customizations violate standard principles and introduce new concepts or patterns, it is more difficult for users to use, and for you to maintain the application.

Objectives

The objectives are:

- Explain the different table types and their characteristics.
- Present the standard data model and introduce the data-related business logic.
- Present the standard process model that governs the transactions in Microsoft Dynamics NAV 2013.

Table Types and Characteristics

As a business management application, Microsoft Dynamics NAV 2013 manages and processes lots of data. All data is stored in tables. From a technical perspective, all tables are the same, as they all contain fields, keys, and triggers. From a functional perspective, there are different types of tables that serve different purposes.

Understanding different table types in Microsoft Dynamics NAV 2013 enables you to efficiently customize existing functionality and design new application areas.

The following table shows the most common table types and their examples.

Type	Remarks	Examples
Master	Table that contains information about the primary focus subject of a application area.	Customer, Vendor, Item
Supplemental	Table that contains information about a supplemental subject that is used in one or more application areas.	Language, Currency
Setup	Table that contains one record that holds general information about a application area.	General Ledger Setup, Sales & Receivables Setup
Register	Table that acts as a table of contents for its corresponding ledger table or tables.	G/L Register, Item Register
Subsidiary	Table that contains additional information about a master table or a supplemental table.	Item Vendor, FA Depreciation Book
Ledger	Table that contains the posted transactional information that is the primary focus of its application area.	Cust. Ledger Entry, Item Ledger Entry
Journal	Primary table that allows you to enter transactions for a application area.	Purchase Journal, Item Journal

Type	Remarks	Examples
Document	Secondary table that enables you to enter transactions for one or multiple application areas at the same time. Document tables are always pairs of tables: one table for document headers, and one table for document lines.	Sales Header/Sales Line, Finance Charge Memo Header/Finance Charge Memo Line, Reminder Header/Reminder Line
Document History	Table that contains the transaction history for documents that were posted.	Sales Invoice Header/Sales Invoice Line, Issued Fin. Charge Memo Header/Issued Fin. Charge Memo Line, Issued Reminder Header/Issued Reminder Line

Master Tables

A master table contains information about the subject of its application area. For example, the **Customer** table is a master table. This is the subject of the sales, marketing, and receivables application areas. A master table is somewhat static. Users regularly enter new master records, but rarely change existing master records.

All transactional tables in an application area are related to a master table. The master table itself is related to many other (usually supplemental) tables. There is at least one ledger table that is related to a master table. Master tables frequently contain many FlowFilters and FlowFields, most of which relate to its corresponding ledger tables. Most application areas have only one master table, although some master tables are shared between different application areas, and some application areas occasionally have more master tables.

Naming Master Tables

The name of a master table relates to the names of the records in the table. For example, the **Customer** table is named **Customer** because each record within it contains information about a customer.

Primary Key and Other Standard Fields

The primary key of a master table is named **No.**, is of type Code, and of length 20. The value of this field is assigned automatically through the number series functionality.

 **Note:** The **G/L Account** table is one important exception to this principle. It is the only master table in Microsoft Dynamics NAV 2013 where **No.** is not controlled by number series functionality.

The description field of a master table is named **Name** or **Description**, is of type Text, and is 50 characters long. This field, together with the **No.** field, is always included in the DataCaptionFields property of the table, so that these fields are displayed in the title bar of the table pages.

Many master tables contain a field named **Blocked**. This is typically of type Boolean. This field indicates whether users can use a master record in transactions. Instead of deleting a master record that is no longer used, users can mark it as **Blocked**. This makes sure that an attempt by any user or system action to use that master record fails. Sometimes this field is of type Option. This prevents the use of the master record in some specific transactions, but allows for use in others. For example, in **Customer** and **Vendor** tables, this field is of type option, and allows for several levels of blocking a customer or a vendor.

Associated Pages

There are always at least three pages that are associated with a master table. They are as follows:

- Card page
- List page
- Statistics page

The Card Page

Use the *card page* to view and edit single records in the master table. The name of the page is the name of the table followed by the word *card*. Therefore, the card page for the **Customer** table is named **Customer Card**.

The first group in the **Navigate** tab on the card page is always named as the master table. This group includes actions that call pages for related or subsidiary information about the master table. These actions can be the following:

- The action for the related ledger entries that you can also call by pressing CTRL+F7 on the keyboard.
- The action for the related statistics page that you can also call by pressing F7.

The List Page

Use the *list page* to view multiple records in the master table. Unlike the card page, you cannot use the list page to edit the master table.

The name of the page is the name of the table followed by the word *list*. Therefore, the list page for the **Customer** table is named **Customer List**. This page is set as the **LookupPageID** property and the **DrillDownPageID** property of the master table.

Similar to the card page, the list page also includes actions to show ledger entries and statistics with the same keyboard shortcuts. The list page has its **CardPageID** property set to the page ID of the corresponding card page. This makes sure that a corresponding card is always opened when users click **View**, **Edit**, or **New** actions, or double-click a row in a list.

The Statistics Page

Use the *statistics page* to view calculated information about the record in the master table. This information is separated from the card page for performance reasons because this information is calculated from a potentially large number of records in the database. This information might slow down data access if it is always displayed on the card page.

The name of this page is the name of the table followed by the word *statistics*. Therefore the statistics page for the **Customer** table is named **Customer Statistics**.

Supplemental Tables

A *supplemental table* contains information about a supplemental subject that is used in one or more application areas. For example, the **Currency** table is a supplemental table that contains information about currencies. This table is not the primary focus of any application area, however, it is important.

Many supplemental tables contain certain sets of defaults that are applied automatically to other types of records, such as master records, when a record from the supplemental table is used. For example, the **Item Category** table contains the following five default fields:

- **Def. Gen. Prod. Posting Group**
- **Def. Inventory Posting Group**
- **Def. Tax Group Code**
- **Def. Costing Method**
- **Def. VAT Prod. Posting Group**

Values from these fields are copied to the relevant fields in the **Item** table, when a user selects a value in the **Item Category Code** field for an item. Generally, supplemental tables are not related to other tables, although many other tables are related to supplemental tables.

Naming Supplemental Tables

The name of a supplemental table is the name of one of the records in the table. For example, the **Currency** table is named **Currency** because each record within it contains information about a currency.

Primary Key and Other Standard Fields

The primary key of a supplemental table is named **Code**, is of type Code and of length 10. The description field of this table is typically named **Description**, is of type Text, and is typically of length 50, even though it may sometimes have a different length. Some supplemental tables do not contain a description field; some supplemental tables contain a description field named **Name**.

Associated Pages

The page that is used for a supplemental table is a list page. The associated page has no or very few actions.

The name of the page is the plural of the name of the supplemental table. Therefore, the page that you use to edit the **Currency** table is named **Currencies**. This page is set as the **LookupPageID** property of the table.

Subsidiary Tables

A subsidiary table contains additional information about a master table or a supplemental table. For example, the **Item Vendor** table is a subsidiary table that contains additional information (vendor numbers) for the **Item** table in the inventory application area.

Naming Subsidiary Tables

The name of this table generally consists of the names of the table or tables for which it is a subsidiary or a close approximation. For example, the table that is subsidiary to both the **Vendor** table and the **Item** table is named **Item Vendor**. Usually, it is a singular name that describes one record that is contained within the table.

Primary Key and Other Standard Fields

The primary key for the **Subsidiary** table contains a field for each table for which it is a subsidiary, each of which is related to that table. For example, the primary key field for the **Item Vendor** table consists of the **Item No.** field (related to the **Item** master table), and the **Vendor No.** field (related to the **Vendor** master table).

The primary key also can contain an Integer as the last field (named **Line No.**) to differentiate multiple records with the same subsidiary relationship. For example, the **Employee Qualification** table has an Integer field in the primary key to differentiate multiple qualification records for the same employee.

Subsidiary tables are generally not related to other tables except for the master tables. Other tables are generally not related to a subsidiary table because each subsidiary table has multiple fields in the primary key. Subsidiary tables usually do not have description fields.

Associated Pages

A subsidiary table uses one page for editing and viewing purposes. This page is usually called from the master or supplemental page to which it is subsidiary. The name of the page is usually the plural of the name of the table, such as **Employee Qualifications** or something that is related to the information in the subsidiary table, such as **Item Vendor Catalog**.

The page that you use for a subsidiary table is either a worksheet page or a list page. The following guidelines help you select the correct type:

- If the primary key for the subsidiary table contains an Integer, the page is a worksheet page. It shows no primary key fields. The primary key fields (except for the Integer field) are included as filters so that they are set automatically when users enter information.
- If the primary key for the subsidiary table does not contain an Integer, the page is a list page. It does not contain the primary key field of the master table from which it is called. This primary key field is filtered so that it is set automatically. For example, if you call the **Vendor Item Catalog** page from **Item Card**, then the **Item No.** field is not displayed. If you call the **Vendor Item Catalog** page from the **Vendor Card** page, then the **Vendor No.** field is not displayed.

Ledger Tables

A ledger table contains the transactional information that is the primary focus of its application area. For example, the **Cust. Ledger Entry** table is a ledger table. It contains all transaction information that is the primary focus of the sales and receivables application area.

This table resembles a subsidiary table because it is related to the corresponding master table. However, it has different characteristics. It is related to many other tables, mostly supplemental tables. Register tables are related to ledger tables, but typically no other tables relate to a ledger table. Most application areas have at least one ledger table, but that depends on functionality (many application areas contain two or more ledger tables). Some ledger tables, such as **G/L Entry** or **Item Ledger Entry**, are shared between several application areas.

In order to maintain an audit trail for transactional information, ledger tables cannot be changed by users except for a few highly controlled exceptions. These exceptions exclude the ability to add or delete a record. Examples of such exceptions are **Cust. Ledger Entry** and **Vendor Ledger Entry** tables, which let users change certain fields, such as **On Hold**, or **Pmt. Disc. Tolerance Date**.

Naming Ledger Tables

The name of the ledger table is usually the name of the master table to which it is related, plus the words *ledger entry* describing one of the records in it (an entry). Because the name can be lengthy, the name is sometimes abbreviated. For example, the customer ledger entry table is actually named **Cust. Ledger Entry**. When there is more than one master table, the name is the application area followed by the words *Ledger Entry*, for example **G/L Entry**.

Primary Key and Other Standard Fields

The primary key of a ledger table is an Integer field named **Entry No.** This primary key is always generated automatically by the posting routine that controls this ledger table, and is always incremented by 1. There is always a field in the ledger table that has a table relationship with the master table that is associated with this ledger table. The description field of this table is a Text field of length 50, and is named **Description**.

In addition to the primary key, ledger tables generally have many secondary keys, many of which have SumIndexFields attached to them. These are used together with the FlowFields on the master table to calculate information for the user. Because of this, at least one of the secondary keys has a field that is related to the master table as the first field in the key.

Associated Pages

A list page is used to view the records in the Ledger table. The name of the page is the plural of the name of the ledger table. Therefore, the page that is used to display records from the **Cust. Ledger Entry** table is named **Customer Ledger Entries**.

This page is set as the **LookupPageID** property and the **DrillDownPageID** property of the table because they are used not only for viewing, but also for lookups and drill-downs into this table.

The list page can be displayed from the master table pages by pressing **CTRL+F7**.

Register Tables

A *register table* is a table of contents for its corresponding ledger table or tables. There is one record per posting process. The register table corresponds more closely to the posting routine instead of the application area. For example, the table that contains the list of entries that are made to the **Cust. Ledger Entry** table is the **G/L Register** table. This is because the customer ledger entries are posted from **General Journal** by using the general ledger posting procedures. The register table is related to its corresponding ledger table or tables.

Naming Register Tables

Register tables are named according to the posting function followed by the word *register*. Therefore, the register table that is updated by the general ledger posting function is named **G/L Register**. Users cannot change the register table.

Primary Key and Other Standard Fields

The primary key of a register table is an Integer field named **No**. The primary key is always automatically incremented by 1 by the posting routine that controls the register. Other standard fields for the register table include the following:

- **From Entry No.** and **To Entry No.** Integer fields that are related to the corresponding ledger table
- **Creation Date** field that specifies the date when that a transaction was posted
- **User ID** field that specifies which user has posted the transaction
- **Source Code** field that indicates the source of the transaction
- **Journal Batch Name** field that indicates the journal from which the transaction was posted

Register tables typically do not have description fields.

Associated Pages

A *list page* is used to view the records in the register table. The name of the page is the plural of the name of the register table. Therefore, the page that is used to display records from the **G/L Register** table is named **G/L Registers**.

The list page contains action links to other list pages that display the corresponding ledger entries.

Journal Tables

Journal tables enable transaction entry for a application area. All transactions, whether entered by a user directly or generated from another posting routine, pass through a journal table to eventually be posted to a ledger table.

Journal tables are related to many other tables including master tables, supplemental tables, subsidiary tables, and sometimes corresponding ledger tables.

Because of their use in transaction entries, journal tables have more trigger codes for validating data than most other table types.

Naming Journal Tables

The name of a journal table is the name of the transaction being posted, followed by the words *journal line*. For example, the table in which users enter transactions to the **Resource** application area is named **Resource Journal Line**. Each record in a journal table contains one line from the corresponding journal.

The journal table is usually related to two corresponding supplemental tables: the journal template table and the journal batch table. These tables let users split up data entry in various ways and let them set optional information that applies to the whole journal.

The names of these two tables are the same as the name of the journal table, except that they are followed by the words *journal batch* or *journal template* instead of *journal line*. Therefore, the two corresponding tables for the **Resource Journal Line** are named **Resource Journal Template** and **Resource Journal Batch**.

Primary Key and Other Standard Fields

The primary key of a journal table consists of three fields:

- **Journal Template Name** field that relates to the journal template table
- **Journal Batch Name** field that relates to the journal batch table
- **Line No.** Integer field

The **Description** field of this table is a Text field of length 50.

Associated Pages

Use a worksheet page to make entries to the journal table. The name of the page is the same as the journal table, except without the word *line*. Therefore, the worksheet page for the **Resource Journal Line** table is named **Resource Journal**. Sometimes, the page is named for the type of data that is entered. For example, one of the many worksheet pages that you use with the **Gen. Journal Line** table is named **Sales Journal**. None of the primary key fields are included on the page.

When the worksheet page is called, it is filtered by the **Journal Template Name** and **Journal Batch Name** fields. The AutoSplitKey property of the worksheet page automatically sets the **Line No.** field by incrementing the last Line No. by 10000, or by trying to “split” the **Line No.** fields of the record above and the record under the insertion of a new record.

The journal worksheet page always includes these actions:

- An action that looks up the card page for the master record that is used in the journal. This can also be called by pressing CTRL+F7.
- An action that shows all ledger entries for the master record that is used in the journal. This can also be called by pressing CTRL+F7.
- An action that posts the journal into the corresponding ledger or ledgers, that is named Post, and can also be called by pressing F9 on the keyboard.

The journal worksheet page usually includes other actions that let the user perform various processing functions.

Document Tables

Document tables are secondary transactional tables that enable entries for one or multiple application areas at the same time. They are secondary only in that their information is posted to ledgers through journal tables, and not directly.

For most users, document tables are the primary means of entering a transaction. Because they are used for transaction entries, document tables have more trigger codes than most other table types.

There are two kinds of document tables:

- Document header tables
- Document line tables

Document Header Table

A *document header table* holds the main transaction information that applies to all lines in the document. For example, for a sales transaction, the **Sales Header** table contains main information about order or invoices, such as the customer to whom the order or invoice belongs, posting dates, shipping information, and similar information that apply to all lines in the document.

Document Line Table

A *document line table* holds detailed information for the transaction. For example, for a sales transaction, the **Sales Line** table contains information about each line of the order or invoice. A document line table is a subsidiary table of the document header table.

Like journal tables, document tables are related to many other tables. This includes master, supplemental, and subsidiary tables. Other table types are rarely related to other document tables.

Naming Document Tables

The name of a document header table is the name of the transaction or document, plus the word *header*. For example, the document header table that contains sales transactions is named **Sales Header**. Each record contains one sale, such as order or invoice. For example, a document header table that contains finance charge memo transactions is named **Finance Charge Memo Header**. Each record contains one finance charge memo.

The name of a document line table is the name of the transaction or document, plus the word *line*. For example, the document line table that contains sales transactions is named **Sales Line**. Each record contains one line from a sale, such as order or invoice. For example, the document line table that contains finance charge memo transactions is named **Finance Charge Memo Line**. Each record contains one line from a finance charge memo.

Primary Key and Other Standard Fields

For most document header tables, the primary key is a Code field of length 20 named **No.** that contains the document number.

Some document header tables contain multiple kinds of documents. For example, the **Sales Header** table contains invoice documents, credit memo documents, sales order documents, and other document types. In these cases, the primary key has two fields: an Option field named **Document Type**, and a Code field of length 20 that is named **No.**

For most document line tables, the primary key has two fields: a Code field of length 20 that contains the document number, and an Integer field named **Line No.** The Code field relates to the document header table. It is typically named according to that table's name (without the word *header*), followed by that table's primary key field. For example, the Code field in the primary key of the **Finance Charge Memo Line** table is named **Finance Charge Memo No.**

When the document header table primary key includes a document type, the primary key of the document line table has the following three fields:

- **Document Type** Option field
- **Document No.** Code field of length 20
- **Line No.** Integer field

The Code field relates to the document header table.

Associated Pages

A document header table uses page of type Document to display one header record at a time to view and edit the information in the header table. The page name is also the name of the document that is displayed. For example, the page that shows finance charge memos from the **Finance Charge Memo Header** table is named **Finance Charge Memo**. This applies even if the table contains multiple types of documents because, in this case, the page is set up to only view information from one type. For example, the page that shows sales invoices from the **Sales Header** table is named **Sales Invoice**.

The page contains FastTabs to split the fields into logical groups. This makes it easier for the user to edit the information. A document page includes a page part that shows the document lines page.

A document line table uses a ListPart page to display multiple line records at a time. The name of the page is the name of the document followed by either words *lines* or *subform*. None of the primary key fields are included on this page. This page is filtered by using the SubPageLink property for all the primary key fields except for the **Line No.** field that is handled automatically by the AutoSplitKey property of the page.

The document header table also uses a list page to let users view multiple documents at the same time. The name of this page is the name of the document page followed by the word *list*. For example, the list page that shows the sales invoices is called **Sales Invoice List**.

The document header list page uses the CardPageID property to specify the document page that is shown when the user views, edits, or creates a new document, or double-clicks a document in the list.

Document History Tables

Document history tables have the same relationship to document tables that ledger tables have to journal tables. When a document is posted, part of that posting process is copying the document tables to their corresponding document history tables.

To aid with that copying process, the document history table has fields that have the same field numbers, names, and properties as the original document tables.

Because document history tables record posted transactions, they generally cannot be edited by the user. However, they can be deleted under certain circumstances. For example, you can delete a posted sales invoice if it was printed. Other than these few distinctions, document history tables and pages are the same as document tables and pages.

Setup Tables

Each application area has its own *setup* table. These setup tables hold only one record that contains fields to select options for the application area, or to hold data that applies to the whole company. No tables are related to setup tables, although setup tables are frequently related to other tables, usually supplemental tables.

Naming Setup Tables

The name of a setup table is usually the name of the application area it configures, followed by the word *setup*. For example, the table that contains setup information for the general ledger application area is named **General Ledger Setup**. One exception to this rule is the **Company Information** table.

Primary Key and Other Standard Fields

The primary key for this table is a Code field of length 10 named **Primary Key**. It is always left blank as only one record for each table is permitted. Setup tables do not have a description field.

Associated Setup Page

There is only one page that is used for setup tables, and it is of type Card. The page has the same name as the table. The primary key field is not included in this page. The page allows for changing information, but does not allow for inserting or deleting records. If there is not a record in the underlying table on this page, the code in the OnOpenPage trigger inserts a record.

 **Note:** Not every table that contains the word setup in its name is a setup table. There are some tables that contain the word setup in their name which have more than one record. These tables generally follow the rules of the subsidiary tables that were described in the "Subsidiary Tables" section, and do not follow the rules that are outlined in this section.

Standard Data Model

Every application area in Microsoft Dynamics NAV 2013 follows the same principles and has a similar data model. Master, subsidiary, document tables, journal, ledger, and other tables all have the same role. There are certain patterns that are applied consistently across all application areas.

Depending on their types, there are certain code patterns that you can follow in all tables of the same type.

The consistency of data model and data flow patterns is important for both users and developers. When users master one application area and understand data model principles, they can also quickly understand other application areas. As a developer, when you understand the principles of data models and patterns, you can customize the standard application. You can also build new application areas and maintain a consistent experience in the standard application. This makes sure that users are as productive as possible.

Data Model Diagram

In every application area, there are three groups of tables.

Group	Remarks
Configuration tables	Static or slowly changing table where users enter information one time, and then rarely, if ever, change it. The application uses these tables during creation, modification, or deletion of records in other tables, such as transaction tables. These tables are frequently checked by various processes, such as posting. Changing information in these tables changes the way that data is processed, or alter other aspects of the functionality of an application area.
Operational transaction tables	Primary work table for users. Users enter information in these tables regularly. Adding, changing, or deleting information in these tables typically does not affect the application, or the business itself.
Posted transaction tables	Table whose information is generated automatically by the application during posting and similar processes. Users cannot create new records in these tables, and cannot change or delete records. There are few examples in which users can change or delete information. All the examples have clear business logic justification.

C/SIDE Solution Development in Microsoft Dynamics® NAV 2013

The following "Data Model of a Functional Area" diagram explains the relationships of various tables in a typical application area.

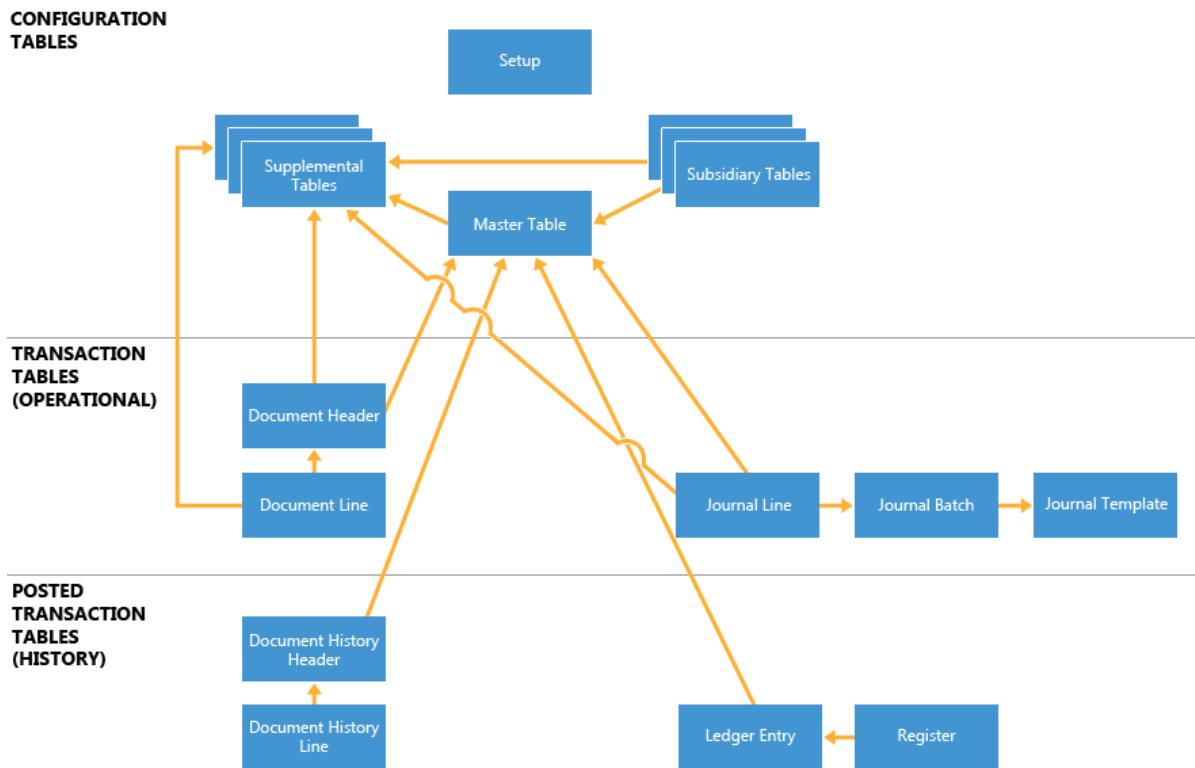


FIGURE 1.1: DATA MODEL OF AN APPLICATION AREA

Typical Data Triggers

In addition to maintaining data validity through data types and table relationships, Microsoft Dynamics NAV 2013 also contains data-related business logic. This makes sure that more complex data-related business rules are consistently applied as users insert, change, or delete information in the database.

These complex business rules are coded in table triggers as C/AL code. You can find the same patterns of code in the same types of tables, regardless of the application area where they belong.

You must understand these principles, and make sure that the code that you write in the existing objects does not violate those principles. You must also apply the same principles when you create completely new application areas.

OnInsert Trigger

This table trigger executes when a user inserts a new record into a table. The code in the trigger executes before the record is actually inserted into the table. If the code in the trigger causes a run-time error to occur, then the insert operation is canceled.

The OnInsert trigger has the following purposes in different table types.

Table Type	Purpose
Master	Assigns the No. field from the appropriate number series, if the user has not provided the value manually. Assigns the default dimensions for the account type, based on the configuration in the Default Dimension table.
Subsidiary	Checks whether all the necessary primary key fields are entered for tables with complex primary keys with three or more fields, or if any mutually exclusive or unacceptable primary key combinations are selected.
Journal	Validates the values in Shortcut Dimension 1 and Shortcut Dimension 2 fields.
Document (header)	Assigns the No. field from the appropriate number series if the user has not provided the value manually. Applies certain defaults, such as dates. Checks the filter on the field that is related to the primary master record of the application area. Assigns the value of the filter to the field if the field is filtered to a single value.
Document (line)	Makes sure that the status can accept new lines if the document supports different statuses.

For these table types, the OnInsert trigger may contain more code. For other table types, the OnInsert trigger may contain code that is specific to the applicable scenario, and no general principles are applied.

OnModify Trigger

This table trigger executes when a user changes an existing record in a table. The code in the trigger executes before the record is actually updated in the table. The application cancels the modifications if an error occurs in the trigger code.

The OnModify trigger has the following purposes in different table types:

Table Type	Purpose
Master	Sets the Last Date Modified field to the current system date.  Note: For master tables, the OnRename trigger must do the same action.
Journal	Makes sure that a modification does not violate business rules for the specific journal type.
Document (header)	Assigns the No. field from the appropriate number series, if the user has not provided the value manually. Applies certain defaults, such as dates. Checks the filter on the field that is related to the primary master record of the application area. Assigns the value of the filter to the field if the field is filtered to a single value.
Document (line)	Makes sure that the modification does not violate business rules for the specific document.

For other table types, the OnModify trigger may contain code that is specific to the applicable scenario, and then no general principles are applied.

OnDelete Trigger

This table trigger executes when a user deletes a record from a table. The code in the trigger executes before the record is physically deleted from the table. The record is not deleted if an error occurs in the trigger code.

The OnDelete trigger has the following purposes in different table types:

Table Type	Purpose
Master	Makes sure that there are no started, but uncompleted transactions (such as orders, jobs, and so on) that are related to the master record. This could leave the system in an inconsistent state or cause issues for transaction processing. Deletes all subsidiary information for the master record. This includes default dimensions and any open documents and transactions.
Supplemental	Deletes all subsidiary information for the supplemental record.
Journal	Makes sure that the deletion does not violate business rules for the specific type of journal.
Document (header)	Makes sure that the deletion does not violate any business rules for the specific document, and deletes all document lines and subsidiary document information.
Document (line)	Makes sure that the deletion does not violate any business rules for the specific document, and deletes all subsidiary document line information.
Document History (header)	Makes sure that the conditions under which a posted document can be deleted are met, and then deletes the posted document lines, and any subsidiary posted document information.
Document History (line)	Deletes any subsidiary posted document line information. It does not check whether the conditions for the deletion are met, because users can never directly delete a document history line.

Depending on the scenario, the OnDelete trigger may contain more code and achieve more goals than specified earlier.

OnValidate Trigger

This field trigger executes after the user enters a value in a field. The code in this trigger executes after the application executes a default validation behavior, such as data type validation.

This trigger is frequently defined on the fields which relate to other tables, such as master tables, subsidiary tables, and supplemental tables. When it is defined on such fields, it frequently performs the following important operations:

- Assigns the default dimensions, if applicable.
- Assigns certain default values to other fields.
- Validates any case-specific complex business rules.

For example, for the **Resource No.** field in the **Res. Journal Line** table, the OnValidate trigger performs the following tasks:

- Assigns the default dimensions from the selected resource to the resource journal line.
- Makes sure that the selected resource is not blocked.
- Assigns several fields from the selected resource to the resource journal line, such as **Description**, **Direct Unit Cost**, **Resource Group No.**, and **Gen. Prod. Posting Group**.
- Makes sure that the time sheet is specified for the resource which requires time sheets, if the resource line is not created automatically by the system.

Standard Data Flow

Users frequently enter data into one table. That data flows between various tables during different processes. For example, information that you enter into master or supplemental tables moves into document tables, journal tables, and then finishes in the ledger tables.

Master Table Data Flow

When users create master tables, most of information in the table is entered directly by the user, whereas some information may come from other tables. During master table creation, most of the default settings are assigned from supplemental tables. For example, when you select an **Item Category Code** for a record in the **Item** table, the application automatically assigns the default **Gen. Product Posting Group**, **VAT Prod. Posting Group**, **Inventory Posting Group**, and **Costing Method** to the item, as defined in the selected item category.

Module 1: Data and Process Model

The following figure "Data Flow during Master Record Creation" shows the flow of the data between tables during creation of master records.

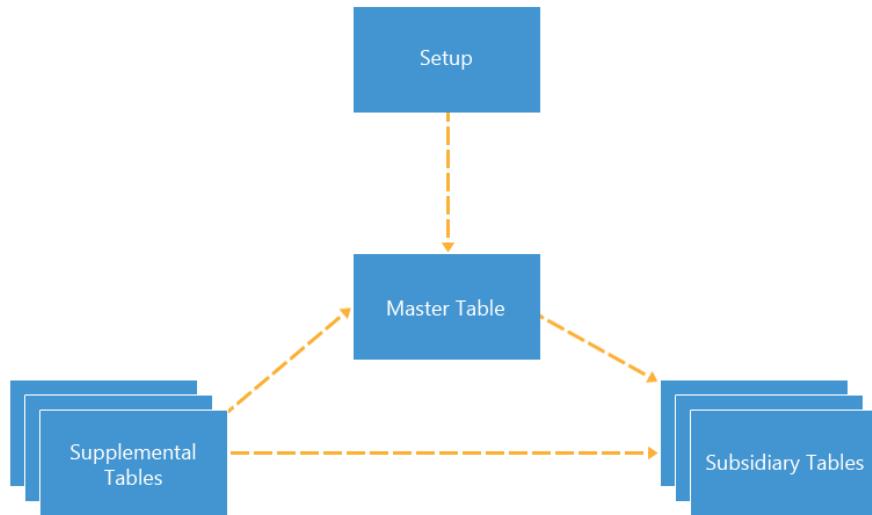


FIGURE 1.2: DATA FLOW DURING MASTER RECORD CREATION

Typically, master tables contain many fields that have relationships to other tables, such as supplemental and other master record tables. When you enter values in these fields, the application may take default values from related tables and assign them to the master record. In addition, when you enter subsidiary information for a master record, some master record fields may be taken into the subsidiary table.

During master record creation, certain defaults are checked in the setup table for the application area. At a minimum, this includes the number series, but may include many default checks or business rules validations.

Document Creation Data Flow

Documents are complex data structures that combine data from various types of tables. When users enter information into documents, most of the configuration tables for the application area are checked.

C/SIDE Solution Development in Microsoft Dynamics® NAV 2013

The following "Data Flow during Document Creation" figure shows data flow during document creation.

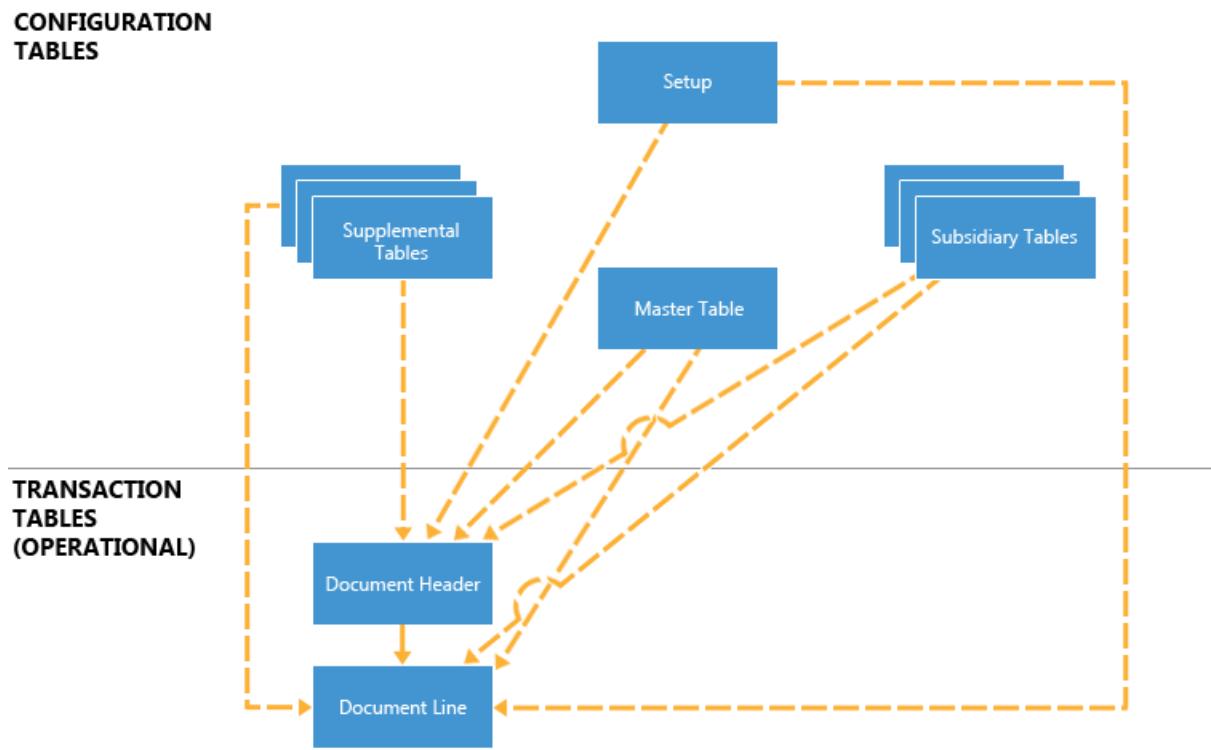


FIGURE 1.3: DATA FLOW DURING DOCUMENT CREATION

Journal Creation Data Flow

Users enter information into a journal line table in a journal. Although users enter most information directly, many fields are assigned automatically. In all journals, many fields, such as **Document No.** or **Reason Code**, are assigned from or based on the corresponding journal batch table. The journal batch table contains a series of other default values for the journal lines. For example, for the **General Journal Line** table, the **Bal. Account Type** and **Bal. Account No.** fields are assigned from the **Gen. Journal Batch** table.

In addition to defaults from the journal, and similar to the documents, many fields are assigned from the master, supplemental, and subsidiary tables that are associated with the journal transaction.

The following "Data Flow during Journal Creation" figure shows the data flow during journal creation.

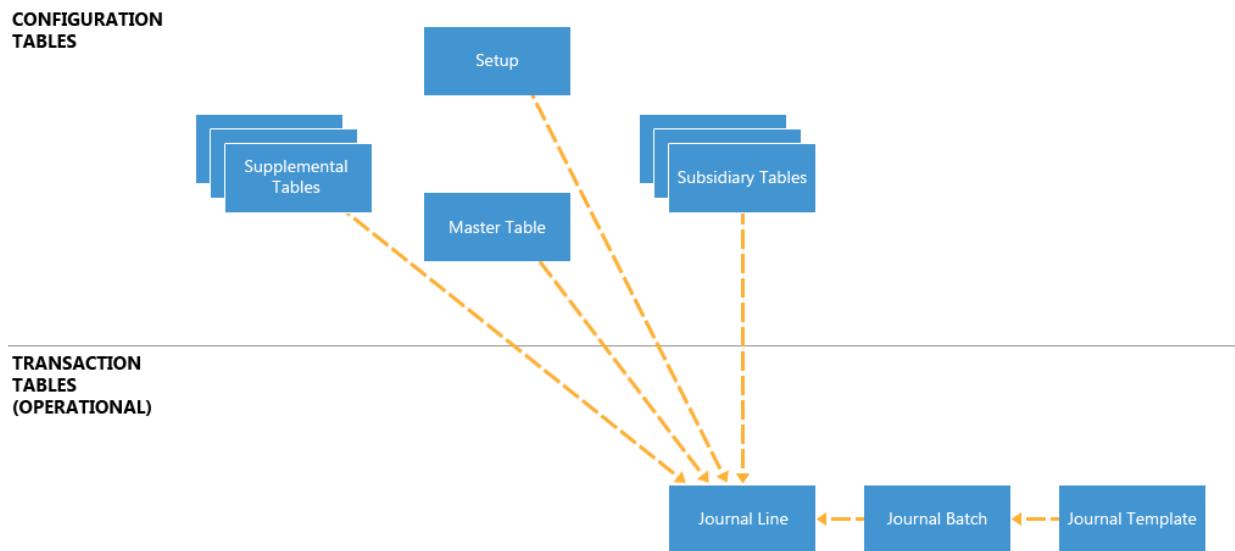


FIGURE 1.4: DATA FLOW DURING JOURNAL CREATION

Posting Data Flow

Posting is one of the most important processes in Microsoft Dynamics NAV 2013. It moves data from operational tables that are under user control (users can freely change them) to the posted transaction tables that are relevant from a business or financial perspective (users cannot freely change them).

There are two types of postings:

- Journal posting moves data from journal tables into a ledger entry tables.
- Document posting performs the following tasks:
 1. Moves data from document tables to document history tables.
 2. Moves data from document tables into journal tables.
 3. Invokes relevant journal posting routines.



Note: *Posting routines are very complex and involve much more processing than moving data between tables.*

C/SIDE Solution Development in Microsoft Dynamics® NAV 2013

The following "Data Flow during Posting" figure shows the data flow during document and journal posting processes.

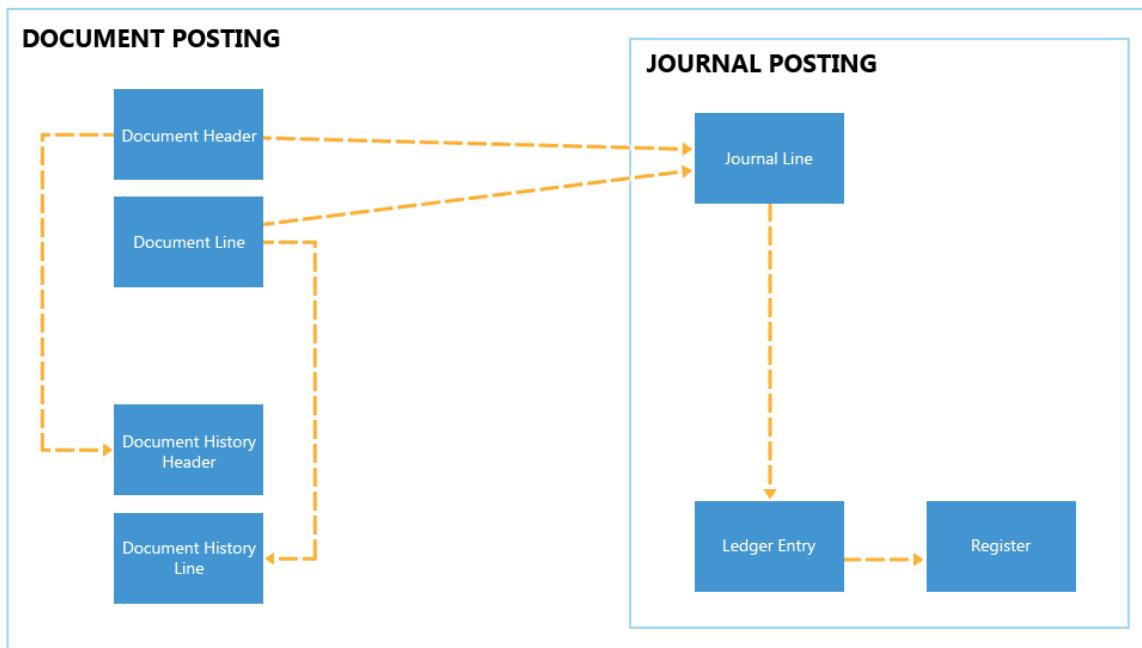


FIGURE 1.5: DATA FLOW DURING POSTING

Standard Process Model

Similar to consistent data structures, all application areas follow the same principle when it comes to processing. Because all application areas contain master, documents, journal, and ledger tables, there are also similar processes that guarantee consistent data flow among tables.

Posting is the most important process in Microsoft Dynamics NAV 2013. It accompanies almost every application area. Even though there are different posting routines for different application areas, all posting routines follow the same patterns and principles. Understanding those patterns and principles makes development simpler, and guarantees a consistent user experience.

Journal Posting

Journal posting is a process that creates ledger entries from journal lines in a application area. A single journal posting process may affect multiple application areas and result in different types of ledger entries. However, journal posting always reads a single set of journal tables. For example, a general journal posting routine always reads **Gen. Journal Line** and **Gen. Journal Batch** tables, but can create entries in the following tables:

- **G/L Entry**
- **Cust. Ledger Entry**
- **Vendor Ledger Entry**
- **Bank Account Ledger Entry**
- **FA Ledger Entry**

A journal posting routine consists of a group of codeunits, some of which are called directly by the user from a page. Other codeunits are called by other codeunits during posting processing.

The following codeunits are the core of the journal posting routine.

Codeunit	Remarks
Journal – Post Line	Reads information from a single journal line, and then writes corresponding ledger entry or entries.
Journal – Check Line	Reads information from a single journal line, and then checks it against various business rules, such as whether posting dates are valid, or if relevant fields contain values.
Journal – Post Batch	Reads information from all lines in a batch, and then calls the check line and post line for each line in the batch.

A user can never call any of these codeunits directly. They are always called by other journal or document posting codeunits.

C/SIDE Solution Development in Microsoft Dynamics® NAV 2013

Users can call the following codeunits by clicking a corresponding action in a journal page.

Codeunit	Available in page	Remarks
Journal – Post	Journal	Asks for a confirmation, and then calls Journal – Post Batch codeunit.
Journal – Post + Print	Journal	Asks for a confirmation, and then calls Journal – Post Batch codeunit, and then prints the corresponding register report.
Journal – Batch Post	Journal Batches	Asks for a confirmation, and then calls Journal – Post Batch for each selected batch.
Journal – Batch Post + Print	Journal Batches	Asks for a confirmation, and then calls Journal – Post Batch and then prints the corresponding register report for each selected batch.

Module 1: Data and Process Model

The following "Journal Posting Process" figure shows the process and data flow of journal posting codeunits.

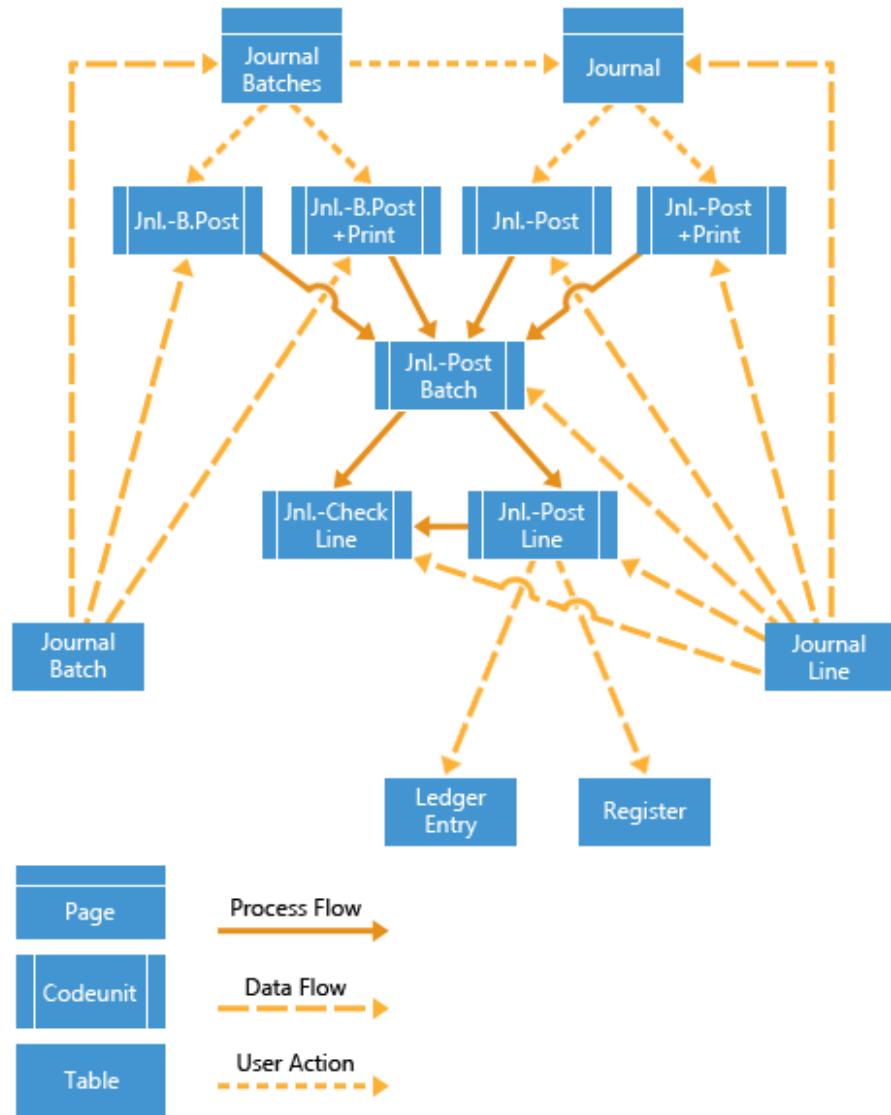


FIGURE 1.6: JOURNAL POSTING PROCESS

Document Posting

Documents resemble journals in the way that they also enable users to enter information before it is posted. Documents are more comprehensive and more intuitive than journals, because they frequently combine the functionality of several journals into a single, easy to use functionality.

C/SIDE Solution Development in Microsoft Dynamics® NAV 2013

Document posting is a process that creates posted documents from operational documents in an application area. Document posting also results in corresponding ledger entries and registers, frequently in multiple application areas. For example, a posting of a sales invoice, may result in ledger entries in the following tables:

- **G/L Entry**
- **Cust. Ledger Entry**
- **Item Ledger Entry**
- **Res. Ledger Entry**
- **FA. Ledger Entry**

A document posting routine consists of several codeunits.

Codeunit	Remarks
Post (Yes/No)	Asks for a confirmation, and then calls the Post codeunit. Users can call this codeunit from document and document list pages.
Post + Print	Asks for a confirmation, calls the Post codeunits, and then prints the posted document. Users can call this codeunit from document and document list pages.
Post	This codeunit is the central document posting codeunit. It copies the document into the posted document. It also analyzes the document and translates it into a series of journal lines from different journals. For each journal line calls the corresponding Journal – Check Line and Journal – Post Line codeunits. Users cannot call this codeunit directly.

Module 1: Data and Process Model

The following "Document Posting Process" figure shows the process and data flow of a document posting routine.

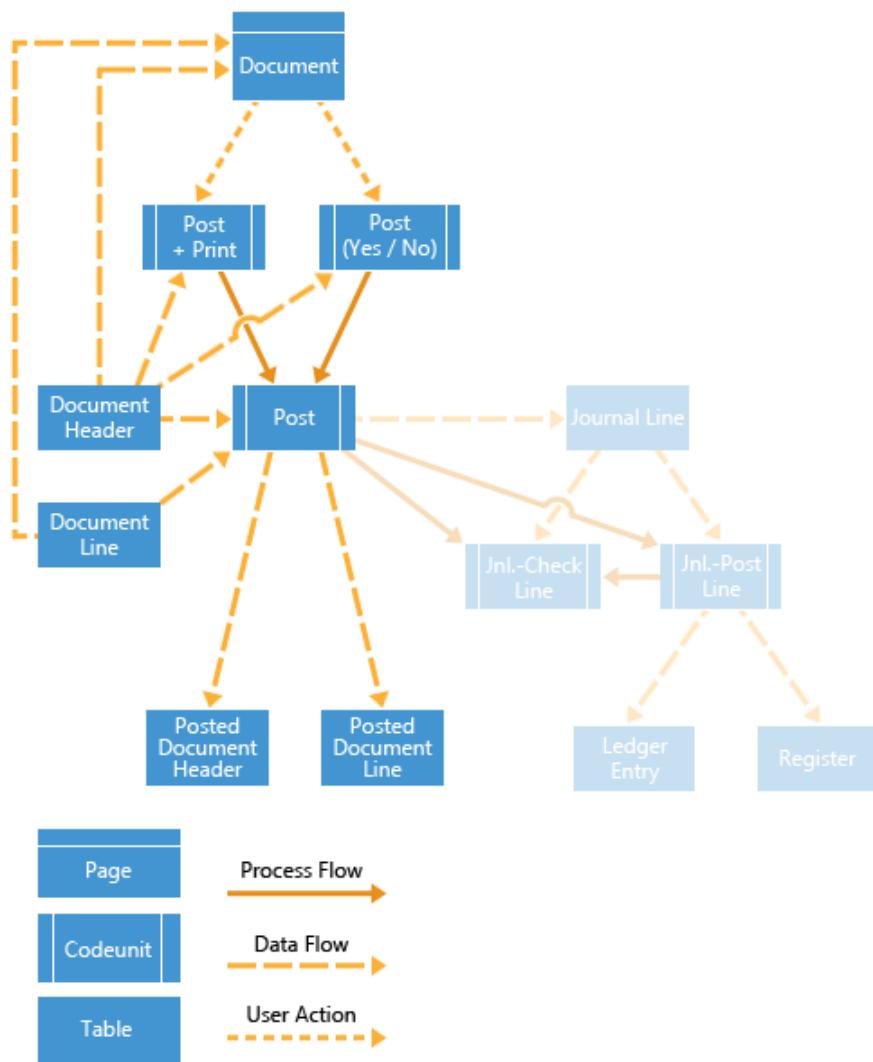


FIGURE 1.7: DOCUMENT POSTING PROCESS

Module Review

Module Review and Takeaways

Microsoft Dynamics NAV 2013 is a rich application that consists of many application areas, with data spread across a number of tables of different types with different purposes. Despite the apparent complexity of data and features, Microsoft Dynamics NAV 2013 applies basic principles to data and processes in all application areas. These principles guarantee that users have a simple, intuitive, and easy-to-learn experience across the application.

Even though Microsoft Dynamics NAV supports many business processes through rich standard functionality, companies frequently have different processes or even whole application areas, which are specific to their business or business vertical. Customizing existing application functionality or introducing new features or application areas into Microsoft Dynamics NAV 2013 is a common task for developers. Changes in such a complex application can easily result in bugs and inconsistencies which can degrade user experience and satisfaction.

Once you understand basic data principles, business logic patterns, and data and process flow in standard Microsoft Dynamics NAV 2013, you can apply them consistently to your own customizations. This guarantees that your changes blend seamlessly into standard functionality, and users have a consistent positive experience.

Test Your Knowledge

Test your knowledge with the following questions.

1. How do you call a table that holds relatively static information about subjects or objects of an application area?
 Master Table
 Subsidiary Table
 Register Table
 Setup Table

Module 1: Data and Process Model

2. What name, data type, and length is the primary key field for a master table?

() No. of type Code and length 10.

() No. of type Code and length 20.

() No. of type Code and length 30.

() Name of type Text and length 50.

3. The **Item Unit of Measure** table is an example of what kind of table?

() Supplemental Table

() Subsidiary Table

() Ledger Table

() Register Table

() Setup Table

4. At minimum, what pages do you have to provide for each master table? How are these pages named? Are these pages editable?

5. What is the primary key of a ledger table?

6. Which trigger makes sure that number series functionality is applied during master record or document creation?

7. A document posting routine always calls one or more journal posting routines.

True

False

Test Your Knowledge Solutions

Module Review and Takeaways

1. How do you call a table that holds relatively static information about subjects or objects of an application area?
 Master Table
 Subsidiary Table
 Register Table
 Setup Table
2. What name, data type, and length is the primary key field for a master table?
 No. of type Code and length 10.
 No. of type Code and length 20.
 No. of type Code and length 30.
 Name of type Text and length 50.
3. The **Item Unit of Measure** table is an example of what kind of table?
 Supplemental Table
 Subsidiary Table
 Ledger Table
 Register Table
 Setup Table
4. At minimum, what pages do you have to provide for each master table? How are these pages named? Are these pages editable?

MODEL ANSWER:

You must provide a list and a card page. The pages are named after the master record, and include words List and Card. For example, Customer List, and Customer Card. The list page must not be editable.

5. What is the primary key of a ledger table?

MODEL ANSWER:

It is an integer field named Entry No.

6. Which trigger makes sure that number series functionality is applied during master record or document creation?

MODEL ANSWER:

The OnInsert trigger.

7. A document posting routine always calls one or more journal posting routines.

() True

() False

MODULE 2: MASTER TABLES AND PAGES

Module Overview

Master tables contain key business information about subjects and objects of business processes. When you develop solutions, you typically first develop the master tables, because all transactional and detailed information in a application area depends on these tables. Master tables are very detailed and typically contain many fields. These fields describe all properties and characteristics of a subject, such as a customer or vendor, or an object, such as an item or a fixed asset, upon which the processes of a application area in Microsoft Dynamics NAV® 2013 depend.

Users cannot view or enter information directly in tables. This means that you must provide other objects that enable users to view and manage this data. In Microsoft Dynamics NAV 2013, *page* is the object type that provides this functionality. For every master table, you also have to provide at least two pages that are required for the basic process of managing the master data. These pages are a *card page* and a *list page*.

Master tables do not include only fields, but also some necessary C/AL logic that executes when certain system events occur. Some of this logic is mandated by Microsoft Dynamics NAV user experience standards, whereas some logic completely depends on the business requirements and processes that are relevant for the users. During development of the master tables and corresponding pages, you must make sure that you include all the necessary C/AL code in the relevant table or page triggers.

In this chapter, you develop the master tables and pages for the Seminar Management module, and make sure that they satisfy your customer's requirements.

Objectives

The objectives are:

- Explain the master table and page standards.
- Work with table event triggers.
- Work with the complex data types and their member functions.
- Explain multilanguage functionality.
- Define the strategy for implementing customers and participants,
- Create tables to manage the seminar rooms.
- Create instructor data management.
- Create seminar data management.

Prerequisite Knowledge

Before you design and develop the solution for managing master data, you must review Microsoft Dynamics NAV 2013 standard functionality. In each module, this "Prerequisite Knowledge" lesson presents Microsoft Dynamics NAV 2013 standards and principles that you can apply while customizing the solution for CRONUS International Ltd.

Triggers

Triggers are methods that execute when a system event occurs or when they are invoked by code. Following are two types of *active* triggers (triggers that contain executable code):

- Event Triggers have names that begin with "On..." and execute when specific events occur. For example, the OnInsert trigger executes when a user inserts a record in a table.
- Function Triggers are custom triggers that developers define. There are many function triggers that are defined as a part of the base application. You can add many more as part of your customization. These triggers execute when they are called by other C/AL code.

The *Documentation Trigger* is the third type of trigger. There is only one Documentation trigger per object, and anything that is written in it is not processed by the application, even if it contains C/AL code. It typically contains free-form documentation. This trigger lets you document the changes that you have made to objects.

This chapter demonstrates how to use these triggers in various scenarios.

Multi-Language Support

Microsoft Dynamics NAV 2013 is a multilanguage application. This means that a localized version of Microsoft Dynamics NAV 2013 can present itself in different languages. In other words, two or more RoleTailored clients that are connected to the same database can present their user interface in two or more languages at the same time. Users can change the language at any time, and the change is applied immediately.

Before you start to develop in a multilanguage-enabled database, set the working language to English (United States). To do this, follow this procedure:

1. On the **Tools** menu, click **Language**.
2. Select English (United States).
3. Click **OK**.

Module 2: Master Tables and Pages

To enable multilanguage functionality for your custom solutions, use the following guidelines:

- Always define the Name property of an object, a control, or a table field in English (United States), or ENU. You must make sure that this is never visible to the user, by specifying the Caption property.
- Always provide language-specific names in the CaptionML property of the objects, controls, or table fields.

 **Note:** A property that ends in *ML* means multilanguage. Make sure that you always set properties to take advantage of the multilanguage functionality of Microsoft Dynamics NAV 2013. Examples of the Multilanguage properties are *CaptionML*, *OptionCaptionML* and *InstructionalTextML*.

Participants

The core business of CRONUS International Ltd. is organizing and delivering seminars. The Seminar Management solution must enable users to schedule seminars and enroll participants in the seminars. Very frequently, companies send several employees to participate in seminars, and those companies pay for their employees' participation. Sometimes, participants are not related to a specific company, and they pay for seminar participation themselves.

Solution Design

CRONUS International Ltd. core functional requirements describe participants as persons who participate in seminars. For each participant, the solution must track the name, address, and contact details including at minimum, a telephone number and an email address. This means that the solution must give users a way to keep track of participants. This information is used for invoice processing.

One of the principles of solution design in Microsoft Dynamics NAV 2013 is to use standard functionality as much as you can. One of the customer's nonfunctional requirements also emphasizes this principle whenever possible. The solution must not duplicate functions that are already present in the application or cause redundant functionality.

C/SIDE Solution Development in Microsoft Dynamics® NAV 2013

Therefore, you must make sure that you determine whether there is existing functionality in Microsoft Dynamics NAV 2013 that CRONUS International Ltd. can use to represent participants. Following is the minimum information that you must track:

- Name
- Address
- Phone Number
- Email Address

There are many tables in Microsoft Dynamics NAV 2013 that include this kind of information about an entity. You must determine whether there is a table that you can use, or if you must develop one.

Following are two candidate tables in Microsoft Dynamics NAV 2013 that both relate to sales and already contain all required fields and functionality:

- Table 18, Customer
- Table 5050, Contact

You may be able to use either of these tables to satisfy the requirement, and may not have to develop new tables. To make sure that you select the correct table, you must consider all requirements that may help you decide. There is a requirement that participants receive invoices for seminar participation. If participants pay for participation themselves, they receive the invoice. If a participant's company pays for participation, the company receives the invoice. This means that there is a relationship between companies and participants. You have to make sure that the solution reflects that relationship.

The following figure represents the relationship between participants and companies:



FIGURE 2.1: PARTICIPANTS AND COMPANIES RELATIONSHIP

Module 2: Master Tables and Pages

Because there are no specific statements that describe companies, you are free to select an existing relationship between tables that are already present in the application. The only guideline that limits you is the following two assumptions from the requirements:

- Participants are obviously persons.
- Companies are obviously business entities, and not persons.

This means that you must look for a set of two tables that meet the following criteria:

- There must be a table that represents a person.
- There must be a table that represents a business entity.
- There must be a clear relationship between the two tables, so that the person belongs to the business entity.

In Microsoft Dynamics NAV 2013 the table that represents business entities in Sales area, is table **18, Customer**. The table that represents persons is **5050, Contact**. There is an existing relationship that links contacts to customers.

This means that the best solution at this point is to map a customer's requirements to the Microsoft Dynamics NAV 2013 standard functionality in the following way.

Required Functionality	Microsoft Dynamics NAV 2013 Standard Functionality
Participant	Contact
Company	Customer

However, the requirements state that both participants and companies can receive invoices. In Microsoft Dynamics NAV 2013 you can send only invoices to customers, but not to contacts. This leaves you with two design options:

- Redesign invoicing functionality to enable contacts to receive invoices.
- For participants who pay for participation themselves, always use both the customer record for invoicing, and the contact record for participation registration.

The first design choice is not a good one because it introduces significant changes to standard functionality. This typically causes many regression issues and other types of bugs. It also makes the application more difficult to develop, maintain, and upgrade to a future version of Microsoft Dynamics NAV. Unless there are other options, this approach should always be your last choice.

C/SIDE Solution Development in Microsoft Dynamics® NAV 2013

The second design choice is not directly covered in requirements. However, it does not directly contradict them either. Even though users do not send invoices to participants directly, when participants pay for participation themselves, the participants are represented by customer records in addition to contact records. This does not introduce any changes into the standard functionality of Microsoft Dynamics NAV 2013. It also does not conflict or contradict the requirements.

It is best practice to use standard functionality as much as you can. However, customer requirements also explain that as long as there are no other requirements that contradict the proposed solution, any solution that proposes a standard functionality or the best practices of Microsoft Dynamics NAV 2013 is preferred.

Therefore, instead of customizing the sales area, you should opt for the second solution, and use the standard Microsoft Dynamics NAV 2013 functionality.

 **Note:** In this lesson, the design process and the decisions that you must make during that process are followed in detail. This lesson introduces you to the best practices of Microsoft Dynamics NAV 2013 solution design. In future lessons and chapters most of the decisions are already made.

Development

After design decisions are made, you must develop the necessary tables and pages that represent the associated entities, and enable users to view and manage them.

Because you have decided to use the standard functionality of customers and contacts, and because both identified tables already have the necessary card and list pages that are typically required to maintain master data, you do not have to do any development.

 **Best Practice:** Do not try to change the names or captions of standard tables and pages to match your customer's terminology. Customers quickly adapt to Microsoft Dynamics NAV 2013 standards, and are not confused by a participant master record that is called Contact.

Instructors and Rooms

The next step is to create tables to manage the instructors who teach the seminars, and tables for the seminar rooms in which the seminars are held. Your goal is to design the solution that meets the requirements, but also takes advantage of any standard Microsoft Dynamics NAV functionality.

Solution Design

The CRONUS International Ltd. requirements for the management of instructors explain that each seminar is taught by an instructor, who is either a CRONUS International Ltd. employee or a subcontractor. For subcontractors, the subcontracting rate must be tracked.

The requirements for the management of seminar rooms explain that each seminar is held in a seminar room. Some seminars are held in-house, and some are held off-site. If a seminar occurs in-house, a room must be assigned. For off-site rooms, the rental rate must be tracked. For all room types, the solution must track the maximum number of participants that the room can accommodate.

Another CRONUS International Ltd. requirement applies to both instructors and rooms. The solution must be aware of and provide the tools to manage and plan for the availability and the capacity of both rooms and instructors.

The instructor and room entities are obviously very similar and have many common requirements. Both share the following characteristics:

- They require the same level of information to describe them.
- They can be internal or external.
- If they are external, they require tracking of additional costs.
- They require a level of availability and capacity management.

The only difference between instructors and rooms is that rooms have to keep track of the maximum number of participants.

In Microsoft Dynamics NAV 2013, there is a resource management application area which closely resembles this functionality. It keeps track of people and equipment (machines), manages their capacity and availability, and keeps track of their costs or prices.

C/SIDE Solution Development in Microsoft Dynamics® NAV 2013

The following table describes the map between resources and instructors and rooms.

Seminar Management Requirement	Resource Management Functionality
Instructor/Room	Type field, with options Person and Machine
Maximum number of participants for rooms	
Availability and capacity management	Availability and capacity management
Internal/External	
Additional costs for external instructors or rooms	Resource costs/Resource prices

There are many functional similarities, and choosing to customize the resource management application area is a good design decision.

 **Note:** Another design approach is to develop separate tables to keep track of instructors and rooms. However, that approach requires duplicating functionality between the standard solution and your custom solution. CRONUS International Ltd. functional requirements advise against this approach.

Development

The design specifies that the table **156, Resource** table represent the instructors and the rooms. Resources of type Person represent instructors, and resources of type Machine represent the rooms. You do not have to create any additional objects.

The following table summarizes the customizations that you must develop.

Object	Customization
Table 156, Resource	Add new fields: <ul style="list-style-type: none">• Internal/External• Maximum Participants
Page 76, Resource Card	Add the Internal/External field to the General FastTab. After Personal Data , add the Room FastTab. In the Room FastTab, add the Maximum Participants field.

Module 2: Master Tables and Pages

Object	Customization
Page 77, Resource List	<p>After the Name field, add the Internal/External and Maximum Participant fields.</p> <p>Make sure that you hide the Type field if the page is filtered by Type.</p> <p>Make sure that you hide the Maximum Participant field if the page is filtered to show only instructors.</p>

The “Page 76, Resource Card” figure shows pages after development work is completed.

The following figure shows the page 76, **Resource Card**, after the customizations.

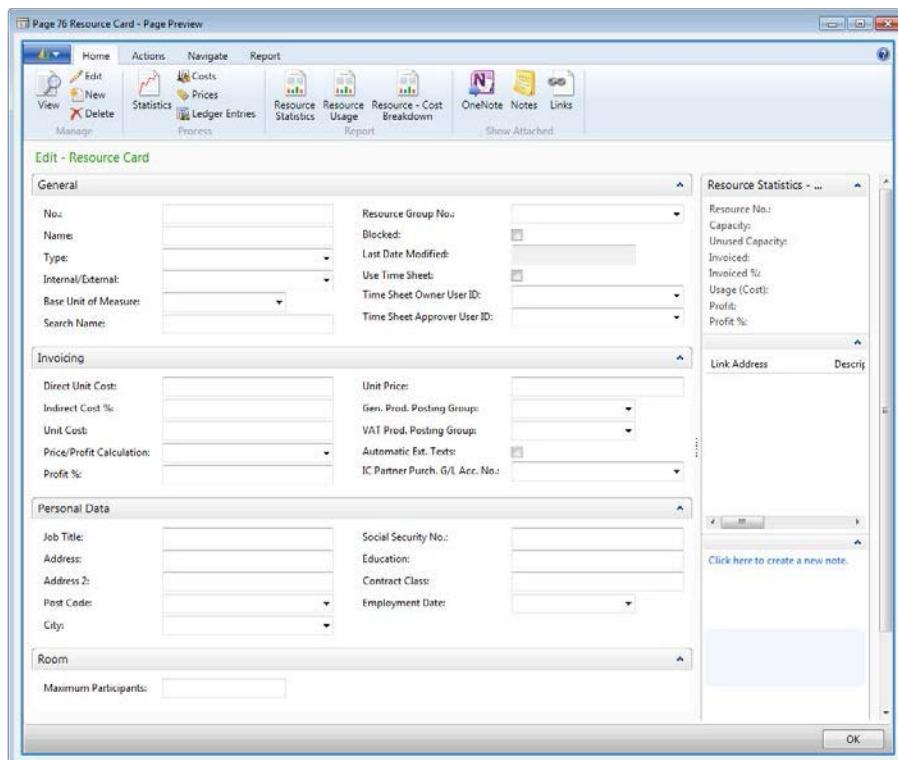


FIGURE 2.2: PAGE 76, RESOURCE CARD

The “Page 77, Resource List” figure shows the list after customization.

C/SIDE Solution Development in Microsoft Dynamics® NAV 2013

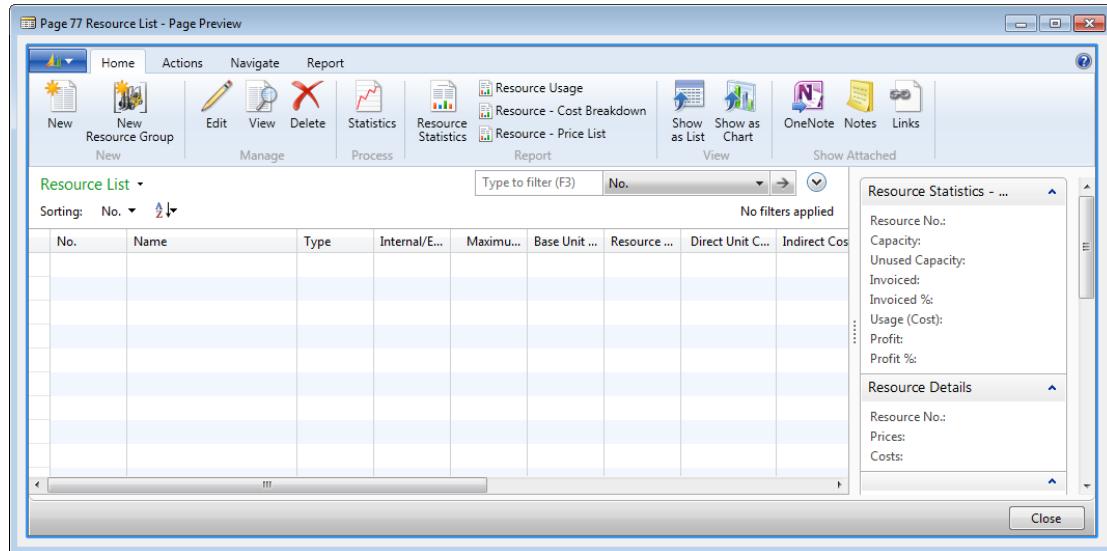


FIGURE 2.3: PAGE 77, RESOURCE LIST

Lab 2.1: Customize Resource Tables and Pages

Scenario

You are a developer who works on the implementation project of Microsoft Dynamics NAV 2013 at CRONUS International Ltd. You must customize the resource table to meet the requirements in the Functional Requirements Document for the project.

Exercise 1: Customize Resource Table

Exercise Scenario

Start by customizing the table to include fields that are specified in the design. These fields are **Internal/External** of type option, and **Maximum Participants** of type integer. You must make sure of the following:

- Your changes do not interfere with any future customizations your customer CRONUS International Ltd. or a third-party might make.
- You clearly document all changes that you make so that other developers can easily identify them.

Task 1: Add Fields to the Table

High Level Steps

1. Design the table **156, Resource**.
2. Add the fields **Internal/External**, of type option, and **Maximum Participants** of type integer.
3. Set the Caption, OptionString, and OptionCaption properties for the fields that you added in the previous step.
4. Save, and then compile the table.

Detailed Steps

1. Design the table **156, Resource**.
 - a. In **Object Designer**, click **Table**.
 - b. In the object list, find table **156, Resource**, and then click **Design** to open the **Table Designer**.
2. Add the fields **Internal/External**, of type option, and **Maximum Participants** of type integer.
 - a. At the end of the field list, add the following fields.

Field No.	Field Name	Data Type
123456701	Internal/External	Option
123456702	Maximum Participants	Integer

Field No.	Field Name	Data Type
123456703	Quantity Per Day	Decimal

3. Set the Caption, OptionString, and OptionCaption properties for the fields that you added in the previous step.
 - a. Select the **Internal/External** field.
 - b. Press SHIFT+F4 to open the **Properties** window.
 - c. In the Caption property, enter "Internal/External".
 - d. In the OptionString property, type "Internal,External". Make sure not to enter any spaces.
 - e. In the OptionCaption property, type "Internal,External".
 - f. Continue on the **Properties** window in **Table Designer**, and select the **Maximum Participants** field.
 - g. In the **Properties** window, in the Caption property, enter "Maximum Participants".
 - h. Close the **Properties** window.
4. Save, and then compile the table.
 - a. Click **File > Save**.
 - b. In the **Save** dialog box, make sure that **Compiled** is selected, and then click **OK**.

Task 2: Document the Changes

High Level Steps

1. Describe your changes in the **Documentation** trigger.
2. Set the Description property to indicate that the fields that you added are a part of the customization.
3. Append the version tag to the Version List property.

Detailed Steps

1. Describe your changes in the **Documentation** trigger.
 - a. Press **F9** to access **C/AL Editor**.
 - b. In the **Documentation** trigger, describe the changes that you made to the table.

Module 2: Master Tables and Pages

For example, the description trigger may contain the following code:

Documentation Trigger

```
CSD1.00 - 2012-06-15 - D. E. Veloper
```

```
Chapter 2 - Lab 1
```

- Added new fields:
- Internal/External
- Maximum Participants

c. Close the **C/AL Editor**.

2. Set the Description property to indicate that the fields that you added are a part of the customization.
 - a. In the **Table Designer**, in the **Description** column for the **Internal/External** and **Maximum Participants** fields, enter "CSD1.00".
 - b. Press CTRL+S to save the table.
 - c. In the **Save** dialog box, click **OK**.
 - d. Close **Table Designer**.
3. Append the version tag to the Version List property.
 - a. In **Object Designer**, make sure that table **156, Resource** is selected.
 - b. Select the Version List property, and press F2 to edit the value.
 - c. At the end of the version list, enter ",CSD1.00".

Exercise 2: Customize Resource Card

Exercise Scenario

After you add the fields to the table, you are ready to customize the pages. Each master table has two important pages that you must consider for customization every time that you add fields to a table: the card page and the list page.

You first customize the card page and add relevant fields and a FastTab, as specified in the design.



Note: Even though the steps to document the changes in the object are not the part of this exercise, make sure that you always document the changes to the documentation that you made to the table **156, Resource** in a similar manner.

Task 1: Add Controls to the Page

High Level Steps

1. Design page **76, Resource Card**.
2. Add the Internal/External and Quantity Per Day fields to the General FastTab.
3. Add the **Room** FastTab as the last FastTab in the page.
4. Add the **Maximum Participants** field to the **Room** FastTab.
5. Save, compile, and then close the page.

Detailed Steps

1. Design page **76, Resource Card**.
 - a. In **Object Designer**, click **Page**.
 - b. In the object list, find page **76, Resource Card**, then click **Design** to open the **Page Designer**.
2. Add the Internal/External and Quantity Per Day fields to the General FastTab.
 - a. Under the **General** group, position the cursor in the first row under the **Type** field.
 - b. Press F3 to insert a new row.
 - c. Click **View > Field Menu**.
 - d. In the **Field Menu** window, select the **Internal/External** field, and then click **OK**.
 - e. Under the **General** group, position the cursor in the first row under the **Base Unit of Measure** field.
 - f. Press F3 to insert a new row.
 - g. Click **View > Field Menu**.
 - h. In the **Field Menu** window, select the **Quantity per Day** field, and then click **OK**.
3. Add the **Room** FastTab as the last FastTab in the page.
 - a. In **Page Designer**, select the row of type Container, and subtype FactBoxArea.
 - b. Press F3 to insert a new row.
 - c. In the **Type** column, select **Group**.
 - d. Press SHIFT+ALT+LEFT to remove one level of indentation from the row.
 - e. In the **Caption** column, enter "Room".

4. Add the **Maximum Participants** field to the **Room** FastTab.
 - a. In **Page Designer**, select the row of type Container, and subtype FactBoxArea.
 - b. Press F3 to insert a new row.
 - c. Click **View > Field Menu**.
 - d. In the **Field Menu** window, select **Maximum Participants**, and then click **OK**.

5. Save, compile, and then close the page.
 - a. Click **File > Save**.
 - b. In the **Save** dialog box, make sure that the **Compiled** check box is selected, and then click **OK**.
 - c. Close **Page Designer**.

Exercise 3: Customize Resource List

Exercise Scenario

After you customize the **Resource Card** page, you are ready to customize the **Resource List** page. As specified in the design, you must add all of the fields that you added to the **Resource** table to this page.

List pages never include all of the fields from the underlying table, but only include sufficient information for users to quickly locate a record. Therefore, you must make sure that you exclude any irrelevant fields from the list tables. Only include those fields that users need.

Use the **Resource** table for both instructors and rooms. However, users do not want to see the combined list of instructors and rooms. Instead, they either want to see the list of instructors or the list of rooms. This indicates that any fields that are irrelevant to instructors or rooms should be hidden when the users view the list of instructors or rooms. You must write the code that achieves this goal.

Task 1: Add Variables

High Level Steps

1. Design page **77, Resource List**.
2. Add the following two Boolean variables: *ShowType*, and *ShowMaxParticipants*.
3. Set the variable properties to make sure that the variables are included in the page dataset.

Detailed Steps

1. Design page **77, Resource List**.
 - a. In **Object Designer**, click **Page**.
 - b. In the object list, find page **77, Resource List**, then click **Design** to open the **Page Designer**.
2. Add the following two Boolean variables: *ShowType*, and *ShowMaxParticipants*.
 - a. Click **View > C/AL Globals**.
 - b. Add the following variables.

Name	Data Type
ShowType	Boolean
ShowMaxParticipants	Boolean

3. Set the variable properties to make sure that the variables are included in the page dataset.
 - a. Select the **ShowType** variable.
 - b. Press SHIFT+F4 to open the **Properties** window.
 - c. Set the **IncludeInDataset** property to **Yes**.
 - d. In the **C/AL Globals** window, select the *ShowMaxParticipants* variable.
 - e. In the **Properties** window, set the **IncludeInDataset** property to **Yes**.
 - f. Close the **Properties** window.
 - g. Close the **C/AL Globals** window.

Task 2: Add Fields to the Page

High Level Steps

1. Under the Type field, add the Internal/External and Maximum Participants fields.
2. Set the Visible property expression of the **Type** field to **ShowType**.
3. Set the Visible property expression of the **Maximum Participants** field to *ShowMaxParticipants*.

Detailed Steps

1. Under the Type field, add the Internal/External and Maximum Participants fields.
 - a. In **Page Designer**, select the first field under the **Type** field.
 - b. Press F3 to insert a new row.

- c. Click **View > Field Menu**.
 - d. In the **Field Menu** window, select the **Internal/External** and **Maximum Participants** fields, and then click **OK**.
 - e. In the confirmation dialog box, click **Yes**.
2. Set the Visible property expression of the **Type** field to **ShowType**.
 - a. In **Page Designer**, select the **Type** field.
 - b. Press SHIFT+F4 to open the **Properties** window.
 - c. In the Visible property, type "ShowType".
 3. Set the Visible property expression of the **Maximum Participants** field to *ShowMaxParticipants*.
 - a. In **Page Designer**, select the **Maximum Participants** field.
 - b. In the **Properties** window, in the Visible property, enter "ShowMaxParticipants".
 - c. Close the **Properties** window.

Task 3: Add Code to the Page

High Level Steps

1. In the OnOpenPage trigger, add the code that sets the value of the *ShowType* and *ShowMaxParticipants* variables depending on the state of the filters in the view filter group. The *ShowType* variable must be *TRUE* if the list is not filtered on the **Type** field. The *ShowMaxParticipants* variable must be *TRUE* if the list is filtered to show the resources of type Machine.
2. Save, compile, and then close the page.

Detailed Steps

1. In the OnOpenPage trigger, add the code that sets the value of the *ShowType* and *ShowMaxParticipants* variables depending on the state of the filters in the view filter group. The *ShowType* variable must be *TRUE* if the list is not filtered on the **Type** field. The *ShowMaxParticipants* variable must be *TRUE* if the list is filtered to show the resources of type Machine.
 - a. In **Page Designer**, press F9 to access the **C/AL Editor**.
 - b. In the OnOpenPage trigger, add the comments block to indicate the changes that you are about to introduce.

C/SIDE Solution Development in Microsoft Dynamics® NAV 2013

The comments block may resemble the “Comments Block” example.

Comments block.

```
//CSD1.00>  
  
//CSD1.00<
```

- c. Within the comments block, add the code which sets the *ShowType* variable to *TRUE* if the page is filtered on the **Type** field.

The code may resemble the “Setting ShowType” example.

Setting ShowType

```
ShowType := GETFILTER(Type) = ";
```

- d. Add another line of code that sets the *ShowMaxParticipants* variable to *TRUE* if the filter on the **Type** field is equal to Machine.

The code may resemble the following “Setting ShowMaxParticipants” example.

Setting ShowMaxParticipants

```
ShowMaxParticipants := GETFILTER(Type) = FORMAT(Type::Machine);
```

- e. Around the two lines that set the values of variables, add code that first switches the filter group to 3, and then back to 0. Make sure that only filters that are set through the RunPageView property are considered.

The resulting code in the OnOpenPage trigger may resemble the following “OnOpenPage Trigger Code” example.

OnOpenPage Trigger Code

```
//CSD1.00>

FILTERGROUP(3);

ShowType := GETFILTER(Type) = "";

ShowMaxParticipants := GETFILTER(Type) = FORMAT(Type::Machine);

FILTERGROUP(0);

//CSD1.00<
```

2. Save, compile, and then close the page.
 - a. Click **File > Save**.
 - b. In the **Save** dialog box, make sure that the **Compiled** check box is selected, and then click **OK**.
 - c. Close **Page Designer**.

Seminars

There is no standard functionality in Microsoft Dynamics NAV 2013 that handles Seminar Management requirements. You must develop a completely new application area. A standard application area in Microsoft Dynamics NAV 2013 consists of a setup table, master tables and pages, and any necessary subsidiary or supplemental tables with their pages.

The core master record of the Seminar Management functionality is the seminar. All master tables share some common characteristics, and any new master tables that you develop must also have the same functionality.

All master pages are accompanied by a card page and a list page. Both pages provide a similar, intuitive, and consistent functionality that you must also provide in any master pages that you develop.

Setup Table and Page

Setup table is one of the core tables of a application area in Microsoft Dynamics NAV 2013. The setup table provides the settings that define the behavior of the business logic in a specific application area of the application. The business logic frequently depends on various configuration settings. Every time that the code makes a decision about how to handle a specific situation, it must check the appropriate setting in the setup table.

For example, when users post invoices, the business decision may be that the posting is only allowed in a specific period. Instead of hard coding the starting and ending dates in the code, you should add the **Allow Posting From** and **Allow Posting To** dates to a setup table. Then you can change the logic of the code to check whether the **Posting Date** of the invoice falls between the dates that are specified in the table. This makes it easy for users to change the periods of allowed posting. This is exactly how the **General Ledger Setup** table controls the posting behavior of any kinds of documents that affect the **General Ledger** application area.

At a minimum, setup tables contain the fields for configuring the number series that control the assignment of numbers to master records, documents, and posted documents of the application area. The more master record and document types, the more fields the setup table contains. There can only be one setup record for a application area. That is why the table has a single primary key field which is called **Primary Key**.

A setup table always has only one page of type card that shows the information from the table. The setup card must make sure that there can be only one record in the table. Therefore, this page always has the InsertAllowed and DeleteAllowed properties set to No. When the user opens the page and there is no record in the setup table, the page typically inserts the setup record.

Master Tables

Master tables contain key information about objects or subjects of business transactions. Examples of master tables are **Customer**, **Vendor**, **Item**, **G/L Account**, or for this solution, **Seminar**. Users must be able to uniquely identify the records in these tables. For that reason, users typically use consecutively assigned numbers. Therefore, master tables always have the primary key that consists of a single field named **No.**, of type code, and length 20.

Number Series Functionality

When a record is inserted, the application assigns a value to this field from a number series that is defined in the setup table for the application area. All master tables in Microsoft Dynamics NAV 2013 follow the same code pattern to implement this functionality. You must follow the same pattern in any master tables that you develop.

The following "Assigning a number from number series" code example from the **Resource** table shows how standard master tables perform this task.

Assigning a number from number series

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

This code first checks if the **No.** field is empty. This means that the user has not provided an explicit value in it. If there is no value, the code first retrieves the setup record for the application area and makes sure that the appropriate number series field is defined. Then, the code calls the standard number series functionality which assigns the next number that is based on the configuration in the **Number Series Line** table.

Every master table has a description field that is called either **Name** or **Description**, and is always of type text and length 50.

To support the number series functionality, you must provide a function that lets users select alternative number series. This function is called from the card page when users click the **AssistEdit** button in the **No.** field.

The following code example demonstrates the C/AL logic of the AssistEdit function of the **Customer** table.

Standard AssistEdit Code

```
WITH Cust DO BEGIN  
  
    Cust := Rec;  
  
    SalesSetup.GET;  
  
    SalesSetup.TESTFIELD("Customer Nos.");  
  
    IF NoSeriesMgt.SelectSeries(SalesSetup."Customer Nos.",OldCust."No.  
Series","No. Series") THEN BEGIN  
  
        NoSeriesMgt.SetSeries("No.");  
  
        Rec := Cust;  
  
        EXIT(TRUE);  
  
    END;  
  
END;
```

Finally, you must make sure that you observe the number series rules if users try to change the **No.** field. When numbers are assigned from a number series, the number series may dictate whether users can manually change the value which was assigned from the number series. There is the **No. Series** field in each master table that keeps track of the number series from which the **No.** field was assigned. To check whether the users can change the assigned number, you must write the code in the OnValidate trigger of the **No.** field.

The following code example demonstrates how the **Resource** table performs this check in the OnValidate trigger of the **No.** field.

No. – OnValidate Trigger in the Resource Table

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

Blocked Field

Master records are used in transactions. This means that after users have worked with the application for a while, many transactional records will relate to any master record. Many master records become obsolete at a specific time. For example, a customer may stop being your customer, or you may stop purchasing goods from a vendor, or an item may be discontinued. In these situations, you typically do not delete master records, but block them so that they remain available for any analysis or comparison purposes. However, you cannot use them in transactions any longer. The field which controls this functionality is called **Blocked**, it is of type Boolean, and it is present in all master tables.

 **Note:** Occasionally, the **Blocked** field is an Option field that enables several types of blocks, letting the master record be used in one set of transactions, and preventing it from being used in other transaction types. For example, the **Blocked** field in the **Customer** and **Vendor** tables is of type Option.

 **Note:** Not every table that contains the **Blocked** field is a master table. However, most of the tables that use this field use it in a similar manner.

Never write any logic that handles the **Blocked** field directly in the master table. But you must make sure that you check the **Blocked** field in the code of any tables that refer to or use the master table. For example, you check whether an item is blocked from the OnValidate trigger of the **No.** field in the **Sales Line** table, to make sure that users cannot use a blocked item in a sales transaction.

Master records are rarely changed, and users typically want to know when a specific master record was changed. Therefore, all master tables contain a noneditable field named **Last Date Modified** of type Date. Master tables also provide the code that sets the value of this field to the current system date when users change the record.

The following code example shows the code in the **OnModify** and **OnRename** triggers that sets the **Last Date Modified** field.

Setting Last Date Modified

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

Legacy Fields

Following are two fields that are typically present in all master tables:

- Comment
- Search Name or Search Description

Both of these fields are legacy fields and do not provide any useful functionality in Microsoft Dynamics NAV 2013. You should add these fields to any master table that you create to stay consistent with the standard layout of the master tables. They support any possible future use of these fields.

Posting Group Fields

Master records participate in business transactions, and business transactions typically have at least some financial aspect to them. The general ledger tracks all financial value of a company and can contain many accounts. In Microsoft Dynamics NAV 2013, the posting groups control the accounts that are used during different kinds of transactions. Each master record has relationships to the posting groups to facilitate selection of the appropriate accounts during the posting of master data transactions.

At the very minimum, all master data uses at least the general posting groups. General posting groups are one of several types of posting groups in Microsoft Dynamics NAV 2013. There are two types of general posting groups:

- *General business posting groups* that define the posting rules for the subjects of the transactions
- *General product posting groups* that define the posting rules for the objects of the transactions

When you design the master table, you must include either the **Gen. Bus. Posting Group** or the **Gen. Prod. Posting Group** field. You decide which one to include by first understanding how the master record is used. If the master record is the subject of your transactions (it represents who you are doing business with, such as customers or vendors), then select the **Gen. Bus. Posting Group** field. If the master record is the object of your transactions (it represents what you are operating with, such as items or resources), then select the **Gen. Prod. Posting Group** field.

Finally, if the master record has any tax relevance for your business, you must include either the **VAT Bus. Posting Group** or the **VAT Prod. Posting Group** field in the table. The rules for these two fields are the same as with the general posting groups.

Master Pages

All master tables have at least the following two pages that enable users to view and enter data in the table:

- List page is used in the list place of the RoleTailored client. It provides an overview of all records in the table.
- Card page is shown when users double-click a row in the list, or click **New, Edit, View, or Delete** actions in the list.

To enable standard navigation for master data pages in the RoleTailored client, you must set the following properties.

Object	Property	Remarks
Master Table	LookupPageID	Specifies which page provides the standard look-up behavior when users are looking up information from another table. For example, users look up information from the Item table when they enter lines in the Item Journal .
List Page	CardPageID	Specifies which page shows when users double-click lines in the list page, or when users click the New, Edit, View, or Delete actions.
List Page	Editable	Must be set to No , to prevent changes in the table directly from the list.

Solution Design

The CRONUS International Ltd. functional requirements are as follows:

- Seminars have a fixed duration, and a minimum and maximum number of participants.
- On occasion, seminars can be overbooked, depending on the capacity of the assigned room.
- Each seminar can be canceled if there are not enough participants.
- The price of each seminar is fixed.

C/SIDE Solution Development in Microsoft Dynamics® NAV 2013

It is obvious from the requirements description that seminars are a key business object for CRONUS International Ltd. Seminars are the master data for the Seminar Management application area that you are developing. Therefore, you must provide the following two tables:

- A setup table to configure the entire Seminar Management application area
- The master table to store information about all seminars

You also must provide all relevant pages for these tables so that users can move through the application and manage seminar information.

The following "Seminar Table Relationships" diagram shows the Seminar Setup and Seminar tables and their relationships to the other standard tables.

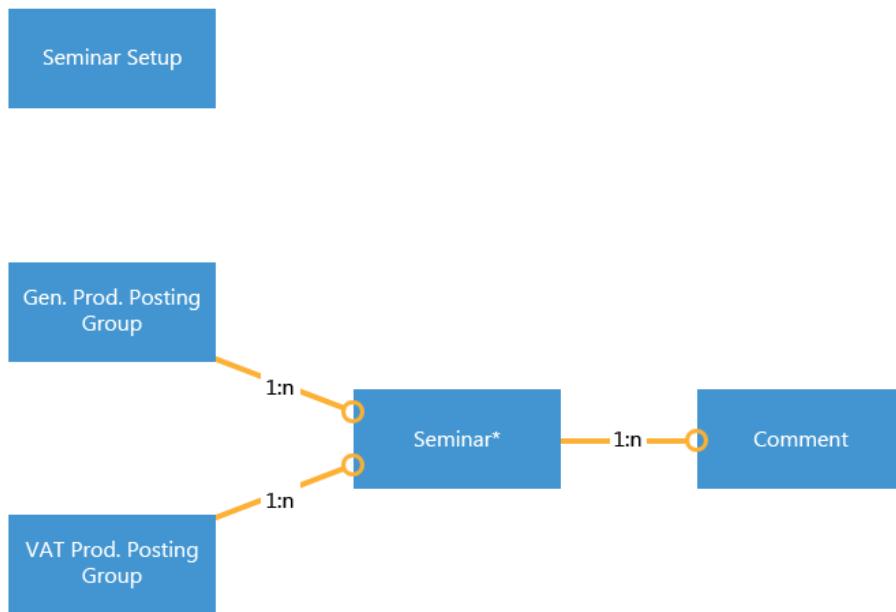


FIGURE 2.4: SEMINAR TABLE RELATIONSHIPS



Note: In this, and all additional entity relationship diagrams, the tables that are marked with an asterisk (*) are new tables that you must develop. All other tables are standard Microsoft Dynamics NAV 2013 tables.

Development

Your first development goal is to develop the tables and then the pages for your new Seminar Management application area. These tables and pages must follow standard Microsoft Dynamics NAV 2013 principles for the setup, master tables, and pages. These master tables and pages must provide the same functionality that users experience with other master tables and pages elsewhere in the application.

When you develop pages for Microsoft Dynamics NAV 2013, consider adding FactBoxes to the pages. FactBoxes enable users to see relevant information about data that is shown in the page. There are several system-provided FactBoxes that manage system-wide data, such as notes and links. Many application-provided FactBoxes display the application data.

At the very minimum, all card and list pages for master tables must have **Links** and **Notes** FactBoxes.

The **Seminar Setup** page as shown in the “Seminar Setup” figure, allows users to configure the Seminar Management application area.

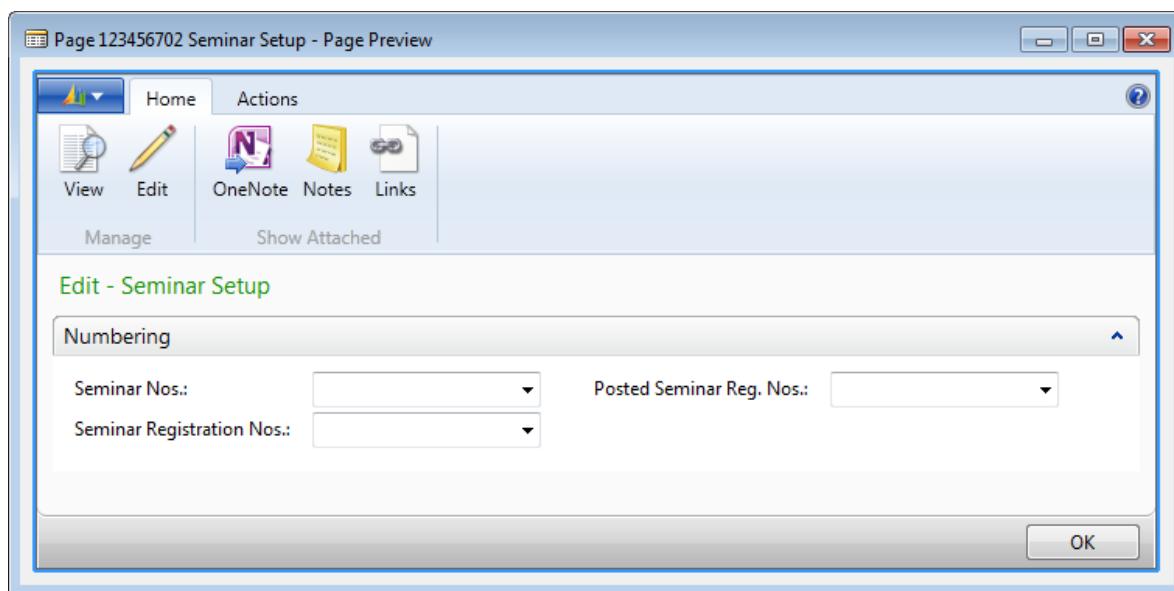


FIGURE 2.5: SEMINAR SETUP (PAGE 123456702)

The **Seminar Card** page as shown in the “Seminar Card” figure, allows users to enter and change the data in the **Seminar** table.

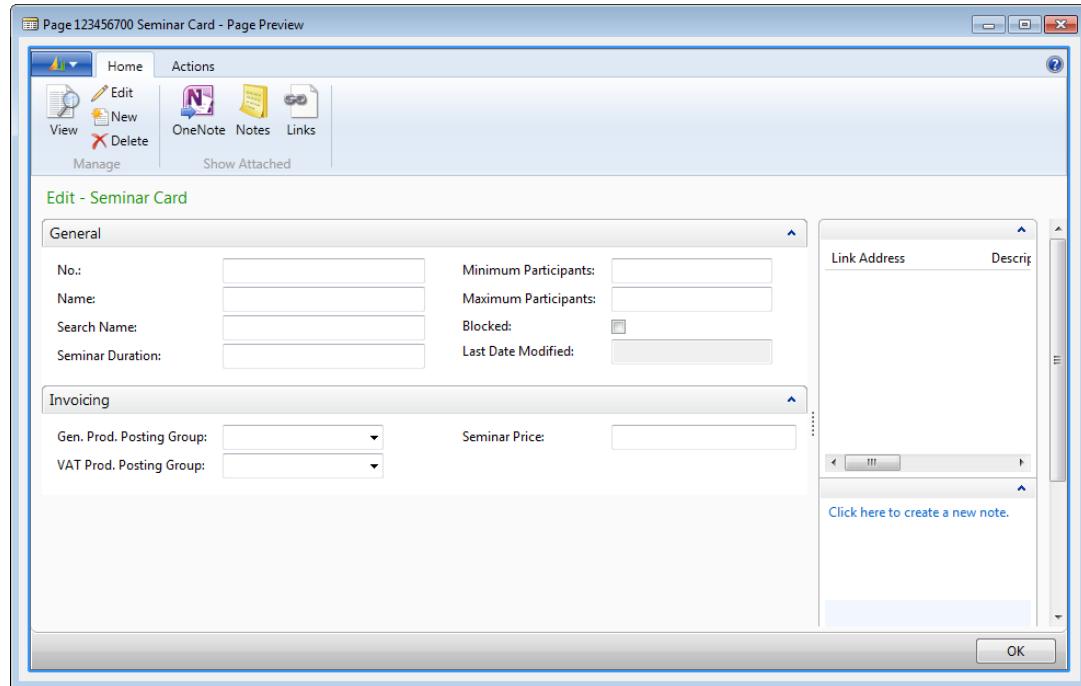


FIGURE 2.6: SEMINAR CARD (PAGE 123456700)

The **Seminar List** page, as shown in the "Seminar List" figure, provides an overview of the seminars, but does not let users change the data in the **Seminar** table.

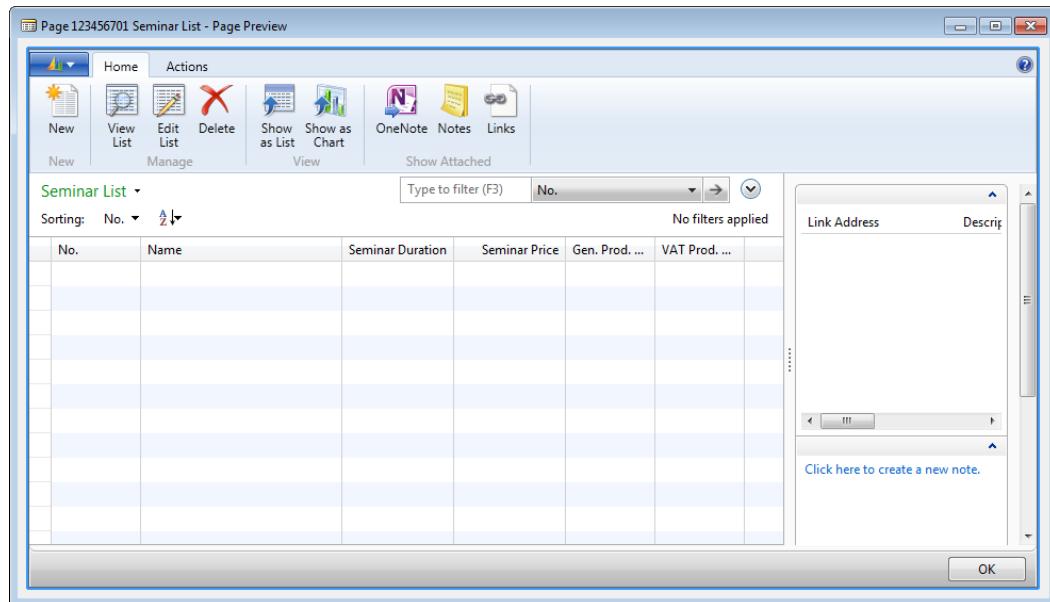


FIGURE 2.7: SEMINAR LIST (PAGE 123456701)

Lab 2.2: Creating Seminar Tables and Pages

Scenario

You must create tables for managing the seminar application area configuration and master data. Also, you must create pages that enable users to enter and maintain the data in these tables. These tables and pages must follow all the principles and Microsoft Dynamics NAV 2013 standards for setup tables, master tables, and card and list pages.

Exercise 1: Append the Table Name Option in the Comment Line table

Exercise Scenario

The *Comments* feature lets users enter arbitrary information about master records and documents. For master data, this information is kept in table **97, Comment Line**. This table has a composite primary key that consists of the following fields:

- **Table Name** that is an option field that specifies the master table to which a comment line relates
- **No.** that specifies the **No.** of the related record in the master table
- **Line No.** that specifies the unique line number of the comment line

To enable users to add comments to seminars, you must customize the **Table Name** option field to include the Seminar option.

Task 1: Add the Seminar Option

High Level Steps

1. Design the table **97, Comment line**.
2. Add the Seminar option to the list of options on the **Table Name** field.
3. Compile, save, and then close the table.

Detailed Steps

1. Design the table **97, Comment line**.
 - a. In **Object Designer**, click Table.
 - b. Select table **97, Comment Line**.
 - c. Click **Design** to open the **Table Designer** window.

2. Add the Seminar option to the list of options on the **Table Name** field.
 - a. Select the **Table Name** field.
 - b. Press SHIFT+F4 to open the **Properties** window.
 - c. In the OptionString property value, append six commas, and then enter "Seminar". Make sure that there are no spaces between the commas or before the word "Seminar".



Best Practice: When you append existing option fields with new options, you have to add several commas before the option that you add. This accommodates any options Microsoft may add to the field in a later version. If you do not add commas, any options that are added by Microsoft later will cause upgrade conflicts and result in possible bugs or issues.

- d. Repeat Step c for the OptionCaption property value.



Note: It is important not to add any spaces between the commas. If you add a space, then the option is not considered empty, and it is displayed as a blank option in the drop-down list. If you do not add a space, then the empty option is not displayed in the drop-down list.

- e. Close the **Properties** window.

3. Compile, save, and then close the table.
 - a. Click **File > Save**.
 - b. In the **Save** dialog box, make sure that the **Compiled** check box is selected.
 - c. Click **OK**.
 - d. Close **Table Designer**.

Exercise 2: Create the Seminar Tables

Exercise Scenario

You must create tables to support the Seminar Management functionality. You first create the **Seminar Setup** table, then the **Seminar** table, and finally you add the required C/AL code to the **Seminar** table to support standard master table behavior.

Task 1: Create the Seminar Setup Table

High Level Steps

1. Create the **Seminar Setup** table, and assign the ID of 123456701 to it.
2. Add fields to the **Seminar Setup** table.
3. Compile, save, and then close the table.

Detailed Steps

1. Create the **Seminar Setup** table, and assign the ID of 123456701 to it.
 - a. Make sure that the **Object Designer** shows tables.
 - b. Click **New** to open the **Table Designer** window.
 - c. Press SHIFT+F4 to access table properties.
 - d. In the **Properties** window, set the following properties.

Property	Value
ID	123456701
Name	Seminar Setup
Caption	Seminar Setup

 **Note:** Notice that when you set the *Caption* property, the *CaptionML* property is automatically set as well. You must always set the *Caption* property of all objects and fields to take advantage of the Microsoft Dynamics NAV 2013 multilanguage functionality.

- e. Close the **Properties** window.

2. Add fields to the **Seminar Setup** table.
 - a. Continue in the **Table Designer** window for the **Seminar Setup** table. Add the following fields to the **Seminar Setup** table.

No.	Field Name	Type	Length	Remarks
1	Primary Key	Code	10	
2	Seminar Nos.	Code	10	Set the TableRelation property to 308 (the No. Series table).
3	Seminar Registration Nos.	Code	10	Set the TableRelation property to 308 (the No. Series table).
4	Posted Seminar Reg. Nos.	Code	10	Set the TableRelation property to 308 (the No. Series table).

3. Compile, save, and then close the table.
 - a. Click **File > Save**.
 - b. In the **Save** dialog box, make sure that the **Compiled** check box is selected, and then click **OK**.
 - c. Close **Table Designer**.

Task 2: Create the Seminar Table

High Level Steps

1. Create the **Seminar** table, and assign the ID of 123456700.
2. Add fields to the **Seminar** table.
3. Configure the **Comment** field as a FlowField which shows if related records exist in the **Comment Line** table.
4. Add keys to the **Seminar** table. Set the primary key to the **No.** field, and add a secondary key for the **Search Name** field.
5. Compile, save, and then close the table.

Detailed Steps

1. Create the **Seminar** table, and assign the ID of 123456700.
 - a. Make sure that the **Object Designer** shows tables.
 - b. Click **New** to open the **Table Designer** window.
 - c. Press SHIFT+F4 to access table properties.
 - d. In the **Properties** window, set the following properties.

Property	Value
ID	123456700
Name	Seminar
Caption	Seminar
LookupPageID	123456700

- e. Close the **Properties** window.
2. Add fields to the **Seminar** table.
 - a. Continue in the **Table Designer** window for the **Seminar** table.
Add the following fields to the **Seminar** table.

No.	Field Name	Type	Length	Remarks
1	No.	Code	20	
2	Name	Text	50	
3	Seminar Duration	Decimal		Set the DecimalPlaces property to 0:1
4	Minimum Participants	Integer		
5	Maximum Participants	Integer		
6	Search Name	Code	50	
7	Blocked	Boolean		
8	Last Date Modified	Date		Set the Editable property to No.
9	Comment	Boolean		Set the Editable property to No.

No.	Field Name	Type	Length	Remarks
10	Seminar Price	Decimal		<p>Set the AutoFormatType property to 1. This makes sure that the value is always formatted as an amount.</p> <hr/>  Note: If you want to learn more about the AutoFormatType property, see Developer and IT Pro Help.
11	Gen. Prod. Posting Group	Code	10	Set the TableRelation property to 251 (the Gen. Product Posting Group table).
12	VAT Prod. Posting Group	Code	10	Set the TableRelation property to 324 (the VAT Product Posting Group table).
13	No. Series	Code	10	Set the Editable property to No and the TableRelation property to 308 (the No. Series Table).

3. Configure the **Comment** field as a FlowField which shows if related records exist in the **Comment Line** table.
 - a. Select the **Comment** field.
 - b. Press SHIFT+F4 to open the **Properties** window.
 - c. Set the FieldClass property to FlowField. This also displays the CalcFormula property that was not previously visible.
 - d. In the CalcFormula property, click the **AssistEdit** button to open the **Calculation Formula** window.
 - e. In the **Method** field, select "Exist".
 - f. In the **Table Field**, select "Comment Line".
 - g. In the **Table Filter** field, click the **AssistEdit** button to open the **Table Filter** window.

Module 2: Master Tables and Pages

- h. Enter the following information in the **Table Filter** window.

Field	Type	Value
Table Name	CONST	Seminar
No.	FIELD	No.

- i. Click **OK** to close the **Table Filter** window. Make sure that you click **OK** before closing the window; otherwise, your changes are not saved.
 - j. Click **OK** to close the **Calculation Formula** window. Make sure not to close this window without clicking **OK**, because you will lose any changes that you made.
 - k. Close the **Properties** window.
4. Add keys to the **Seminar** table. Set the primary key to the **No.** field, and add a secondary key for the **Search Name** field.
- a. Click **View > Keys** to open the **Keys** window.
 - b. On the first line, enter "No."
 - c. On the second line, enter "Search Name".
 - d. Close the **Keys** window.
5. Compile, save, and then close the table.
- a. Click **File > Save**.
 - b. In the **Save** dialog box, make sure that the **Compiled** check box is selected, and then click **OK**.
 - c. Close **Table Designer**.

Task 3: Add Code to the Seminar Table

High Level Steps

1. Add the variables for the **Seminar Setup**, **Comment Line**, **Seminar** and **Gen. Product Posting Group** tables, and the **NoSeriesManagement** codeunit.
2. Add the code to the **OnInsert** trigger, to perform the following logic: if there is no value in the **No.** field, assign the next value from the number series that is specified in the **Seminar Nos.** number series in the **Seminar Setup** table.
3. Add the code to the **OnModify** and **OnRename** triggers to set the **Last Date Modified** field to the system date.
4. Add the code to the **OnDelete** trigger to delete any comment lines for the seminar record being deleted.

5. In the OnValidate trigger of the **No.** field, enter the code to perform the following logic: when the user changes the **No.** value, validate that the number series that is used to assign the number allows manual numbers. Then set the **No. Series** field to blank.
6. In the OnValidate trigger of the **Name** field, enter the code that sets the **Search Name** field if it was equal to the uppercase of the previous value of the **Name** field.
7. In the OnValidate trigger of the **Gen. Prod. Posting Group** field, enter the code that performs the following logic: if the **ValidateVatProdPostingGroup** function of the **Gen. Product Posting Group** table returns *TRUE*, set the **VAT Prod. Posting Group** to the value of the **Def. VAT Prod. Posting Group** field from the **Gen. Product Posting Group** table.
8. In the **Seminar** table, create a new function named **AssistEdit** with a return type of Boolean. In this function, enter the code that makes sure there is a value in the **Seminar Nos.** field in the **Seminar Setup** table, and then calls the **SelectSeries** function in the **NoSeriesManagement** codeunit to check the series number. If this function returns *TRUE*, call the **SetSeries** function in the **NoSeriesManagement** codeunit to set the **No.** field, and then return *TRUE*.
9. Compile, save, and then close the table.

Detailed Steps

1. Add the variables for the **Seminar Setup**, **Comment Line**, **Seminar** and **Gen. Product Posting Group** tables, and the **NoSeriesManagement** codeunit.
 - a. Design table **123456700 Seminar**.
 - b. Click **View > C/AL Globals**.
 - c. Create the following variables.

Name	DataType	Subtype
SeminarSetup	Record	Seminar Setup
CommentLine	Record	Comment Line
Seminar	Record	Seminar
GenProdPostingGroup	Record	Gen. Product Posting Group
NoSeriesMgt	Codeunit	NoSeriesManagement

Module 2: Master Tables and Pages

2. Add the code to the OnInsert trigger, to perform the following logic:
if there is no value in the **No.** field, assign the next value from the number series that is specified in the **Seminar Nos.** number series in the **Seminar Setup** table.
 - a. Press F9 to open the **C/AL Editor** window.
 - b. In the OnInsert trigger, enter the following code:

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

3. Add the code to the OnModify and OnRename triggers to set the **Last Date Modified** field to the system date.
 - a. In the OnModify trigger, enter the following code:

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

- b. In the OnRename trigger, enter the same code.

4. Add the code to the OnDelete trigger to delete any comment lines for the seminar record being deleted.
 - a. In the OnDelete trigger, enter the following code:

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

5. In the OnValidate trigger of the **No.** field, enter the code to perform the following logic: when the user changes the **No.** value, validate that the number series that is used to assign the number allows manual numbers. Then set the **No. Series** field to blank.

C/SIDE Solution Development in Microsoft Dynamics® NAV 2013

- a. In the OnValidate trigger of the **No.** field, enter the following code:

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

6. In the OnValidate trigger of the **Name** field, enter the code that sets the **Search Name** field if it was equal to the uppercase of the previous value of the **Name** field.

- a. In the OnValidate trigger of the **Name** field, enter the following code:

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

7. In the OnValidate trigger of the **Gen. Prod. Posting Group** field, enter the code that performs the following logic: if the **ValidateVatProdPostingGroup** function of the **Gen. Product Posting Group** table returns *TRUE*, set the **VAT Prod. Posting Group** to the value of the **Def. VAT Prod. Posting Group** field from the **Gen. Product Posting Group** table.

- a. In the OnValidate trigger of the **Gen. Prod. Posting Group** field, enter the following code:

```
IF xRec."Gen. Prod. Posting Group" <> "Gen. Prod. Posting Group" THEN BEGIN  
  
    IF  
        GenProdPostingGroup.ValidateVatProdPostingGroup(GenProdPostingGroup,"VAT  
        Prod. Posting Group") THEN BEGIN  
  
            VALIDATE("VAT Prod. Posting Group",GenProdPostingGroup."Def. VAT Prod.  
            Posting Group");  
  
        END;  
  
    END;  
  
END;
```

Module 2: Master Tables and Pages

8. In the **Seminar** table, create a new function named **AssistEdit** with a return type of Boolean. In this function, enter the code that makes sure there is a value in the **Seminar Nos.** field in the **Seminar Setup** table, and then calls the **SelectSeries** function in the **NoSeriesManagement** codeunit to check the series number. If this function returns *TRUE*, call the **SetSeries** function in the **NoSeriesManagement** codeunit to set the **No.** field, and then return *TRUE*.
 - a. Click **View > C/AL Globals**.
 - b. In the **C/AL Globals** window, click the **Functions** tab.
 - c. In the first empty line, enter "AssistEdit".
 - d. Click **Locals**.
 - e. In the **C/AL Locals** window, click the **Return Value** tab.
 - f. In the **Return Type** field, select Boolean.
 - g. Close the **C/AL Locals** window.
 - h. Close the **C/AL Globals** window.
 - i. In the AssistEdit function trigger, enter the following code:

```
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  
  
        NoSeriesMgt.SetSeries("No.");  
  
        Rec := Seminar;  
  
        EXIT(TRUE);  
  
    END;  
  
END;
```

9. Compile, save, and then close the table.
 - a. Click **File > Save**.
 - b. In the **Save** dialog box, make sure that the **Compiled** check box is selected, and then click **OK**.
 - c. Close **Table Designer**.

Exercise 3: Create the Seminar Pages

Exercise Scenario

To fully support the master data functionality, you must define the **Seminar Setup** page to configure the Seminar Management application area, the **Seminar Card** page, and the **Seminar List** page. Then you must establish standard Microsoft Dynamics NAV 2013 master page functionality. This includes the RoleTailored client navigation and number series selection functionality.

Task 1: Create the Seminar Setup Page

High Level Steps

1. Create the **Seminar Setup** page as shown in the "Seminar Setup (Page 123456702)" figure, and assign the **ID** of 123456702 to it.
2. Add code to the OnOpenPage trigger to insert a new record, if there is not a record in the Seminar Setup table.
3. Compile, save, and then close the page.

Detailed Steps

1. Create the **Seminar Setup** page as shown in the "Seminar Setup (Page 123456702)" figure, and assign the **ID** of 123456702 to it.
 - a. In **Object Designer**, click **Page** to access the list of pages.
 - b. Click **New** to create a new page.
 - c. In the **New Page** window, in the **Table** field, enter "123456701", and then select the **Create a page using a wizard** option.
 - d. Click **OK** to start the **Card Page Wizard**.
 - e. In the **FastTab Name** column, replace the existing value with "Numbering", and then click **Next**.
 - f. In **Available Fields**, select and then click **>** for the following fields:
 - Seminar Nos.
 - Seminar Registration Nos.
 - Posted Seminar Registration Nos.
- g. Click **Next**, and then click **Finish**.
- h. In the **Page Designer** window, select the first empty line, and then press SHIFT+F4 to access the **Page - Properties** window.

Module 2: Master Tables and Pages

- i. Set the properties as follows.

Property	Value
ID	123456702
Name	Seminar Setup
Caption	Seminar Setup

- j. Close the **Page – Properties** window.
2. Add code to the OnOpenPage trigger to insert a new record, if there is not a record in the Seminar Setup table.
 - a. Click **View > C/AL Code**, or press F9 to open the **C/AL Editor** window.
 - b. In the OnOpenPage trigger, enter the following code:

```
IF NOT FINDFIRST THEN
```

```
    INSERT;
```

3. Compile, save, and then close the page.
 - a. Click **File > Save**.
 - b. In the **Save** dialog box, make sure that the **Compiled** check box is selected, and then click **OK**.
 - c. Close **Page Designer**.

Task 2: Create the Seminar Card Page

High Level Steps

1. Create the **Seminar Card** page as shown in the "Seminar Card (Page 123456700)" figure, and assign the **ID** of 123456700 to it.
2. Add the code to the OnAssistEdit trigger of the **No.** field control to call the **AssistEdit** function of the underlying table, and to update the page if the function returns *TRUE*.
3. Add an action to the page, to show the **Comment Sheet** page for the seminar that is currently displayed in the page.
4. Compile, save, and then close the page.

Detailed Steps

1. Create the **Seminar Card** page as shown in the "Seminar Card (Page 123456700)" figure, and assign the **ID** of 123456700 to it.
 - a. In **Object Designer**, click **New** to create a new page.
 - b. In the **New Page** window, in the **Table** field, enter "123456700", and then select the **Create a page using a wizard** option.
 - c. Click **OK** to start the **Card Page Wizard**.

- d. In the **FastTab Name** list, in the first empty line enter "Invoicing", and then click **Next**.
- e. Add the following fields to the **General** FastTab:
 - No.
 - Name
 - Minimum Participants
 - Maximum Participants
 - Search Name
 - Seminar Duration
 - Blocked
 - Last Date Modified
- f. Add the following fields to the **Invoicing** FastTab:
 - Gen. Prod. Posting Group
 - VAT Prod. Posting Group
 - Seminar Price
- g. Click **Next**.
- h. Click the **System** tab, and then add the **RecordLinks** and **Notes** FactBoxes in that order.
- i. Click **Finish**.
- j. In the **Page Designer** window, select the first empty line, and then press SHIFT+F4 to access the **Page - Properties** window.
- k. Set the properties as follows.

Property	Value
ID	123456700
Name	Seminar Card
Caption	Seminar Card

- l. Close the **Page – Properties** window.
2. Add the code to the OnAssistEdit trigger of the **No.** field control to call the **AssistEdit** function of the underlying table, and to update the page if the function returns *TRUE*.
 - a. In the OnAssistEdit trigger of the **No.** field control, enter the following code:

```
IF AssistEdit THEN
```

```
    CurrPage.UPDATE;
```

Module 2: Master Tables and Pages

3. Add an action to the page, to show the **Comment Sheet** page for the seminar that is currently displayed in the page.
 - a. Click **View > Page Actions** to open the Action Designer.
 - b. Enter the following information.

Type	SubType	Caption
ActionContainer	RelatedInformation	
ActionGroup		&Seminar
Action		Comments

- c. Select the **Comments** action, and then click SHIFT+F4 to open the **Properties** window.
 - d. In the Image property, enter "Comment".
 - e. In the RunObject property, enter "Page Comment Sheet".
 - f. In the RunPageLink property, click the **AssistEdit** button to open the **Table Filter** window.
 - g. Enter the following information in the **Table Filter** window.

Field	Type	Value
Table Name	CONST	Seminar
No.	FIELD	No.

- h. Click **OK** to accept the changes and close the **Table Filter** window.
 - i. Close the **Properties** window.
4. Compile, save, and then close the page.
 - a. Click **File > Save**.
 - b. In the **Save** dialog box, make sure that the **Compiled** check box is selected, and then click **OK**.
 - c. Close **Page Designer**.

Task 3: Create the Seminar List Page

High Level Steps

1. Create the **Seminar List** page as shown in the "Seminar List (Page 123456701)" figure, and assign the **ID** of 123456701 to it.
2. Add an action to the page to show the **Comment Sheet** page for the seminar that is selected in the list.
3. Compile, save, and then close the page.

Detailed Steps

1. Create the **Seminar List** page as shown in the "Seminar List (Page 123456701)" figure, and assign the **ID** of 123456701 to it.
 - a. In **Object Designer**, click **New** to create a new page.
 - b. In the **New Page** window, in the **Table** field, enter "123456700".
 - c. Select the **Create a page using a wizard** option, and then select the **List** option.
 - d. Click **OK** to start the **List Page Wizard**.
 - e. Add the following fields:
 - **No.**
 - **Name**
 - **Seminar Duration**
 - **Seminar Price**
 - **Gen. Prod. Posting Group**
 - **VAT Prod. Posting Group**
 - f. Click **Next**.
 - g. Click the **System** tab, and then add the **RecordLinks** and **Notes** FactBoxes in that order, and then click **Finish**.
 - h. In the **Page Designer** window, select the first empty line, and then press SHIFT+F4 to access the **Page - Properties** window.
 - i. Set the properties as follows.

Property	Value
ID	123456701
Name	Seminar List
Caption	Seminar List
Editable	No
CardPageID	Seminar Card

- j. Close the **Page – Properties** window.
2. Add an action to the page to show the **Comment Sheet** page for the seminar that is selected in the list.
 - a. Do not close the **Page Designer** for the **Seminar List** page.
 - b. Press SHIFT+F12 to go to the **Object Designer**.
 - c. Select page **123456700, Seminar Card** and click **Design**.
 - d. Click **View > Page Actions**.
 - e. In the **Action Designer** window, press CTRL+A, and then CTRL+C. This copies all actions from the **Seminar Card** page to clipboard.

Module 2: Master Tables and Pages

- f. Close the Action Designer and Page Designer windows for the **Seminar Card** page.
 - g. On the **Window** menu, click the option for the **Page Designer** window for the **Seminar List** page.
 - h. Click **View > Page Actions** to open the **Action Designer**.
 - i. Press CTRL+V to past the actions from the clipboard.
-



Note: You can copy actions from one page to another using the standard **CTRL+C** and **CTRL+V** keyboard shortcuts. This copies actions with all their properties.

- j. Close the **Action Designer**.
3. Compile, save, and then close the page.
 - a. Click **File > Save**.
 - b. In the **Save** dialog box, make sure that the **Compiled** check box is selected, and then click **OK**.
 - c. Close **Page Designer**.

Module Review

Module Review and Takeaways

In this chapter, you used two of the most fundamental object types, tables and pages, to establish the foundation for the Seminar Management solution. These tables define and store data for the solution. The pages also provide an intuitive interface where the user can interact with the table data.

You have seen and applied the master table principles of Microsoft Dynamics NAV 2013 standards to the master tables and pages to make sure that users have a consistent experience across the application.

Test Your Knowledge

Test your knowledge with the following questions.

1. What is the main difference between Documentation triggers and event triggers?

2. What is the main difference between event triggers and function triggers?

3. Which property do you set on a page to display a specific caption in Swedish language?

Name

NameML

Caption

CaptionML

CaptionSwedish

Module 2: Master Tables and Pages

4. When defining Multilanguage properties, you must always define a value in English (United States) language. Which language identifier represents English (United States)?

- () ENU
- () ENG
- () EN
- () ENML

Test Your Knowledge Solutions

Module Review and Takeaways

1. What is the main difference between Documentation triggers and event triggers?

MODEL ANSWER:

Documentation triggers contain free-text documentation, and their contents are not compiled or run. Event triggers contain C/AL code that must compile and that runs when the event occurs.

2. What is the main difference between event triggers and function triggers?

MODEL ANSWER:

Event triggers are defined by the system. They run when a predefined event occurs. Function triggers are defined by the developer. They run when a C/AL code line calls them.

3. Which property do you set on a page to display a specific caption in Swedish language?

() Name

() NameML

() Caption

() CaptionML

() CaptionSwedish

4. When defining Multilanguage properties, you must always define a value in English (United States) language. Which language identifier represents English (United States)?

() ENU

() ENG

() EN

() ENML

MODULE 3: DOCUMENTS

Module Overview

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

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

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

Objectives

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

Prerequisite Knowledge

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

Working with Objects as Text Files

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

You can export and import objects in the following formats.

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

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

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

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

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

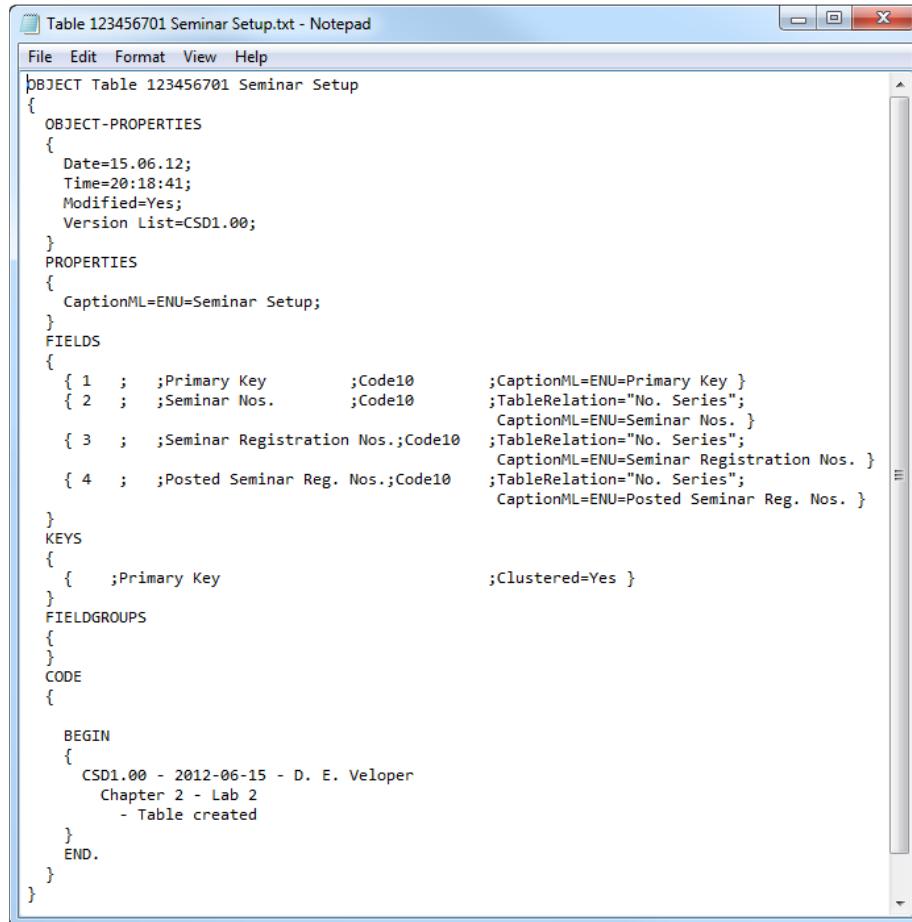
To export one or more objects as a text file, follow these steps:

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

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

C/SIDE Solution Development in Microsoft Dynamics® NAV 2013

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



The screenshot shows a Windows Notepad window titled "Table 123456701 Seminar Setup.txt - Notepad". The content of the window is the text representation of a Microsoft Dynamics NAV object (Table 123456701 Seminar Setup). The text is structured in sections, each starting with a brace {}, followed by the section name in uppercase. The sections include: OBJECT, OBJECT-PROPERTIES, PROPERTIES, FIELDS, KEYS, FIELDGROUPS, CODE, BEGIN, and END. The code uses semicolons as separators and includes various properties like Date, Time, Modified, and Version, along with field definitions and their properties.

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

        BEGIN
        {
            CSD1.00 - 2012-06-15 - D. E. Veloper
            Chapter 2 - Lab 2
            - Table created
        }
        END.
    }
}
```

FIGURE 3.1: TABLE 123456701, SEMINAR SETUP IN TEXT FORMAT

This file consists of several sections:

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

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



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

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

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

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

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

Multilanguage Functionality in Text Messages

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

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

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

Bad practice example

```
IF CONFIRM('Do you want to delete the customer?') THEN BEGIN  
    Cust.DELETE;  
    MESSAGE('Customer no. %1 is successfully deleted.',Cust."No.");  
END;
```

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

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

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

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

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

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

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

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

Using text constants to display multilanguage enabled texts

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

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

Document Pages

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

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

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

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

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

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

Module 3: Documents

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

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

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

Page Functions

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

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

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

Virtual Tables

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

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

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

Module 3: Documents

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

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

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

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

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

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

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

Temporary Tables

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

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

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

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

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

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

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

System Tables

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

- Security permissions
- Object metadata
- Record links
- Charts

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

Registrations

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

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

Solution Design

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

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

C/SIDE Solution Development in Microsoft Dynamics® NAV 2013

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

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

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

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

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

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

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

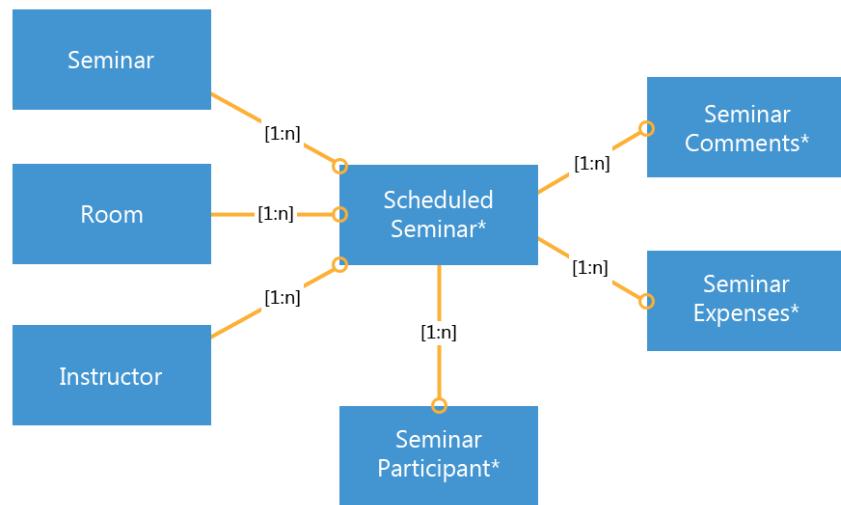


FIGURE 3.2: LOGICAL ENTITY RELATIONSHIP DIAGRAM

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

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

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

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

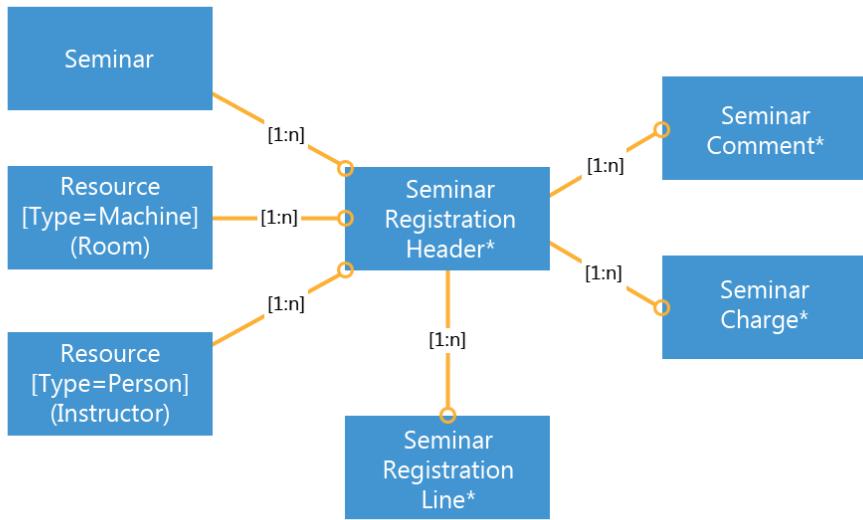


FIGURE 3.3: SEMINAR REGISTRATION TABLES

Development

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

Following is typical functionality for the documents:

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

Tables

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

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

Pages

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

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

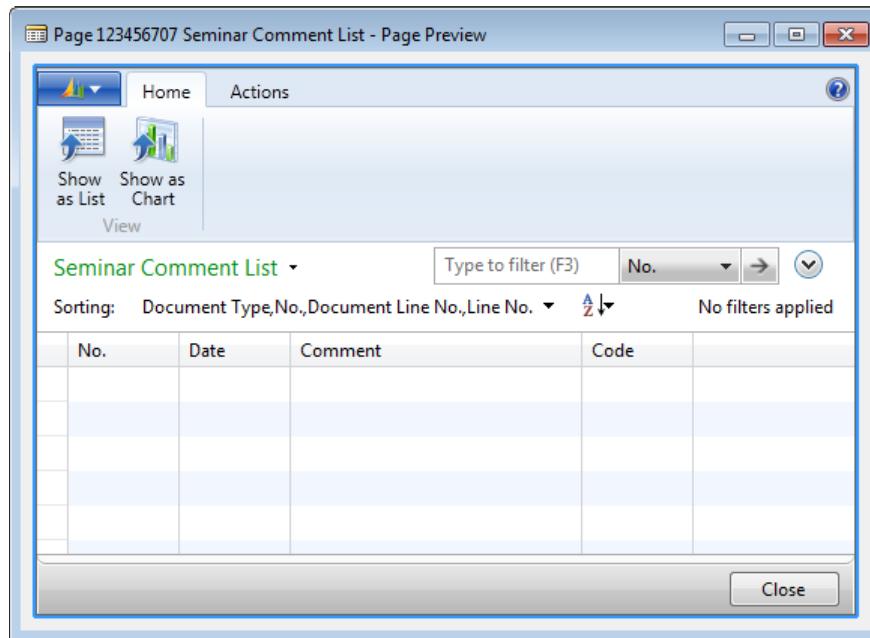


FIGURE 3.4: SEMINAR COMMENT LIST PAGE (123456707)

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

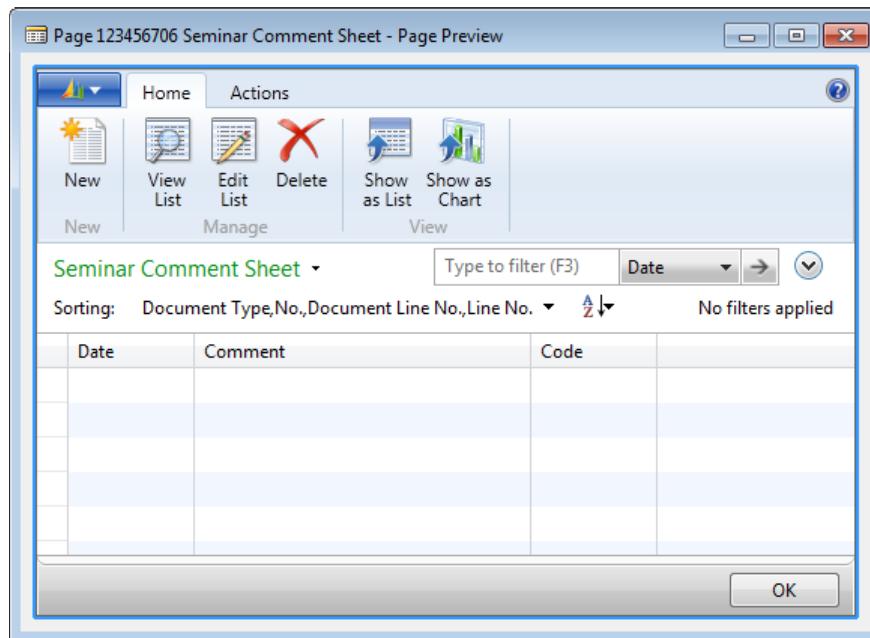


FIGURE 3.5: SEMINAR COMMENT SHEET PAGE (123456706)

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

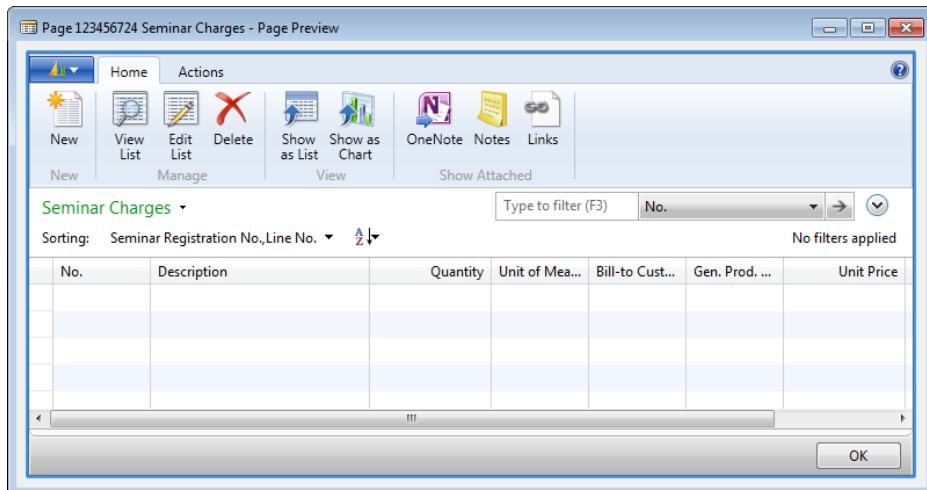


FIGURE 3.6: SEMINAR CHARGES PAGE (123456724)

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

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

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

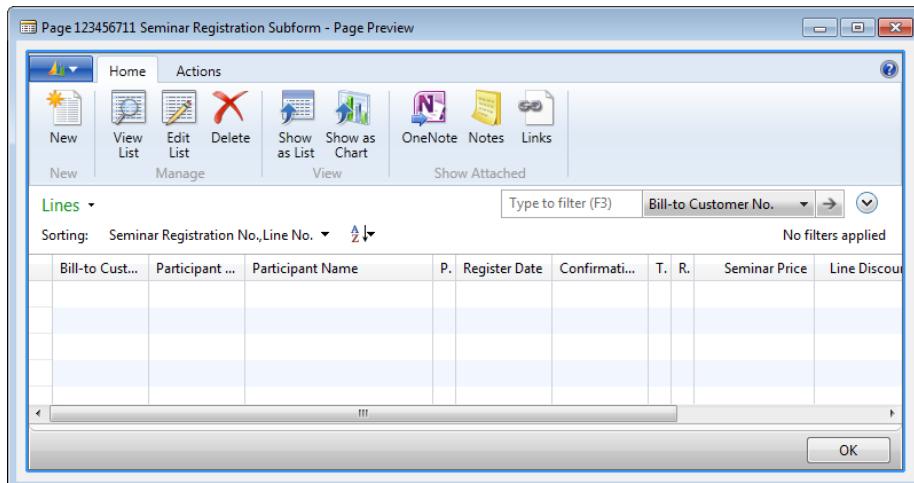


FIGURE 3.7: SEMINAR REGISTRATION SUBFORM PAGE (123456711)

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

C/SIDE Solution Development in Microsoft Dynamics® NAV 2013

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

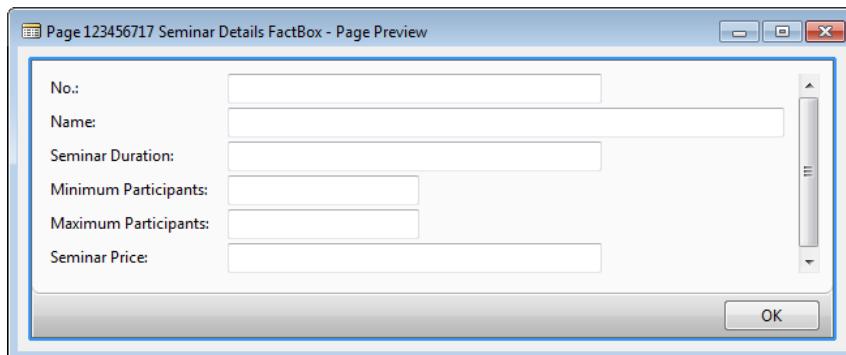


FIGURE 3.8: SEMINAR DETAILS FACTBOX PAGE (123456717)

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

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

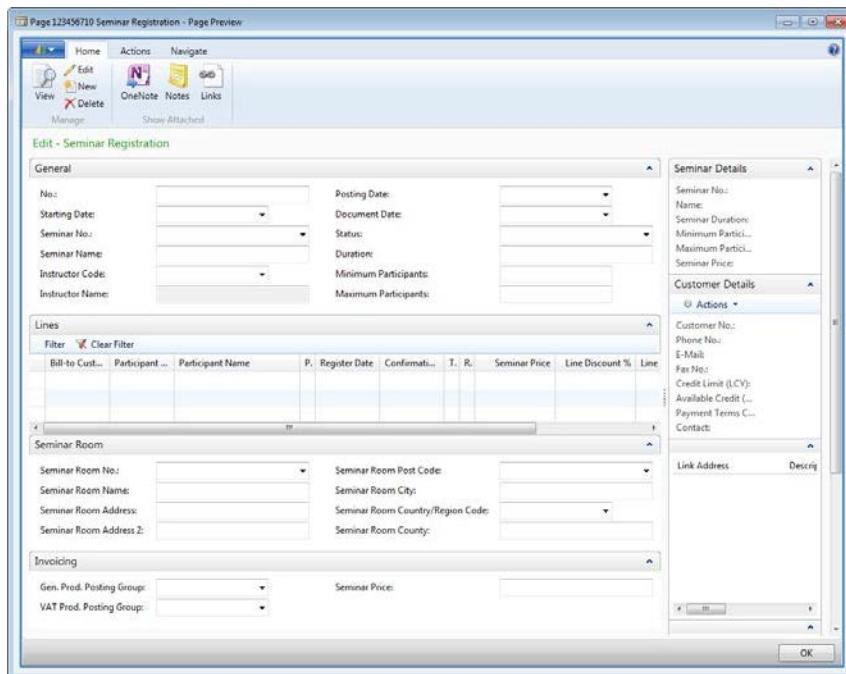


FIGURE 3.9: SEMINAR REGISTRATION PAGE (123456710)

Module 3: Documents

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

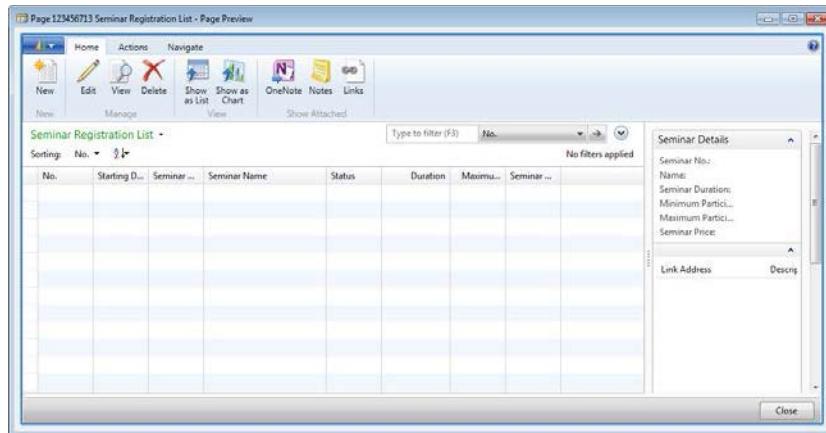


FIGURE 3.10: SEMINAR REGISTRATION LIST PAGE (123456713)

Lab 3.1: Importing, Reviewing and Completing Seminar Registration Tables

Scenario

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

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

Exercise 1: Import the Starter Objects

Exercise Scenario

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

Task 1: Review the Text File

High Level Steps

1. Open the Mod03\Labfiles\Lab 3.A - Starter.txt file.
2. Note down all objects and their types that are contained in the file.
3. Make sure that only the following objects are present in the file:
 - o **Seminar Comment Line** table (123456704)
 - o **Seminar Registration Header** table (123456710)
 - o **Seminar Registration Line** table (123456711)
 - o **Seminar Charge** table (123456712)
 - o **Seminar Comment Sheet** page (123456706)
 - o **Seminar Comment List** page (123456707)

The file should not contain any other objects.

4. Make sure that no imported objects exist in the database.

Detailed Steps

1. Open the Mod03\Labfiles\Lab 3.A - Starter.txt file.
 - a. Locate the Mod03\Labfiles\Lab 3.A - Starter.txt.
 - b. Double-click the file to open it in Notepad.
2. Note down all objects and their types that are contained in the file.
 - a. In Notepad, click **Edit > Find**.
 - b. In the **Find** dialog box, in **Find what**, type "OBJECT " (with space after the word "OBJECT").
 - c. Click **Find Next**.
 - d. Note down each object type, ID, and name.
 - e. Repeat steps c and d as long as you find objects.
3. Make sure that only the following objects are present in the file:
 - o **Seminar Comment Line** table (123456704)
 - o **Seminar Registration Header** table (123456710)
 - o **Seminar Registration Line** table (123456711)
 - o **Seminar Charge** table (123456712)
 - o **Seminar Comment Sheet** page (123456706)
 - o **Seminar Comment List** page (123456707)

The file should not contain any other objects.

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

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

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

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

Task 2: Importing and Compiling Objects

High Level Steps

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

Detailed Steps

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

Exercise 2: Review the Seminar Registration Header Table

Exercise Scenario

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

Task 1: Reviewing Table and Field Properties

High Level Steps

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

Detailed Steps

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

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

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



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

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



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

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

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

Task 2: Review Code

High Level Steps

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

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

Detailed Steps

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

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

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

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

Module 3: Documents

The code should look exactly like the following example:

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

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

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

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

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

C/SIDE Solution Development in Microsoft Dynamics® NAV 2013

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

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

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

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

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

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

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

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

```
SeminarRegLine.RESET;  
  
SeminarRegLine.SETRANGE("Document No.", "No.");  
  
SeminarRegLine.SETRANGE(Registered,TRUE);  
  
IF SeminarRegLine.FIND('-') THEN  
  
    ERROR(  
  
        Text001,
```

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

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

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

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

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

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

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

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

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

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

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

```
IF "Starting Date" <> xRec."Starting Date" THEN  
  
TESTFIELD(Status,Status::Planning);
```

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

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

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

```
SeminarRegLine.RESET;  
  
SeminarRegLine.SETRANGE("Document No.", "No.");  
  
SeminarRegLine.SETRANGE(Registered,TRUE);  
  
IF NOT SeminarRegLine.ISEMPTY THEN
```

```

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

```

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

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

```

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

```

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

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

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

```
IF "Room Resource No." = "" THEN BEGIN  
  
    "Room Name" := "";  
  
    "Room Address" := "";  
  
    "Room Address 2" := "";  
  
    "Room Post Code" := "";  
  
    "Room City" := "";  
  
    "Room County" := "";  
  
    "Room Country/Reg. Code" := "";  
  
END ELSE BEGIN  
  
    SeminarRoom.GET("Room Resource No.");  
  
    "Room Name" := SeminarRoom.Name;  
  
    "Room Address" := SeminarRoom.Address;  
  
    "Room Address 2" := SeminarRoom."Address 2";  
  
    "Room Post Code" := SeminarRoom."Post Code";  
  
    "Room City" := SeminarRoom.City;  
  
    "Room County" := SeminarRoom.County;  
  
    "Room Country/Reg. Code" := SeminarRoom."Country/Region Code";  
  
END;
```

Module 3: Documents

c. Make sure that there is code that does the following:

- Compares the maximum number of participants for the room with the maximum number of participants for the seminar.
- Asks the user to confirm whether the seminar can accept more participants.

```
IF (CurrFieldNo <> 0) THEN BEGIN  
  
    IF (SeminarRoom."Maximum Participants" <> 0) AND  
  
        (SeminarRoom."Maximum Participants" < "Maximum Participants")  
  
    THEN BEGIN  
  
        IF CONFIRM(Text004,TRUE,  
  
            "Maximum Participants",  
  
            SeminarRoom."Maximum Participants",  
  
            FIELDCAPTION("Maximum Participants"),  
  
            "Maximum Participants",  
  
            SeminarRoom."Maximum Participants")  
  
        THEN  
  
            "Maximum Participants" := SeminarRoom."Maximum Participants";  
  
        END;  
  
    END;
```

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

a. Make sure that the following code exists.

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



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

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

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

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

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

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

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

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

```
WITH SeminarRegHeader DO BEGIN  
    SeminarRegHeader := Rec;  
    SeminarSetup.GET;
```

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

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

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

Reviewing the Table Code

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

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

Demonstration: Reviewing the Seminar Registration Line Table Code

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

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

Demonstration Steps

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

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

GetSeminarRegHeader Function

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

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

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

UpdateAmount Function

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

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

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

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

OnInsert trigger

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

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

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

OnDelete Trigger

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

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

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

Bill-to Customer No. – OnValidate Trigger

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

```
END;
```

```
END;
```

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

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

Participant Contact No. – OnValidate Trigger

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

```
END;  
END;  
END;
```

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



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

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

Participant Contact No. – OnValidate

```
ContactBusinessRelation.RESET;  
  
ContactBusinessRelation.SETRANGE("Link to Table",ContactBusinessRelation."Link  
to Table":>Customer);  
  
ContactBusinessRelation.SETRANGE("No.", "Bill-to Customer No.");  
  
IF ContactBusinessRelation.FINDFIRST THEN BEGIN  
  
    Contact.SETRANGE("Company No.",ContactBusinessRelation."Contact No.");  
  
    IF PAGE.RUNMODAL(PAGE::"Contact List",Contact) = ACTION::LookupOK THEN  
    BEGIN  
  
        "Participant Contact No." := Contact."No.";
```

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

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

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

Seminar Price – OnValidate trigger

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

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

Line Discount % - OnValidate Trigger

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

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

Line Discount Amount – OnValidate Trigger

```

IF "Seminar Price" = 0 THEN BEGIN

    "Line Discount %" := 0;

END ELSE BEGIN

    GLSetup.GET;

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

END;

UpdateAmount;

```

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

Amount – OnValidate Trigger

```

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

TESTFIELD("Seminar Price");

GLSetup.GET;

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

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

IF "Seminar Price" = 0 THEN BEGIN

    "Line Discount %" := 0;

END ELSE BEGIN

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

END;

```

Demonstration: Reviewing the Seminar Charge Table

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

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

Demonstration Steps

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

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

Type – OnValidate Trigger

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

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



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

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

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

No. – OnValidate Trigger

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

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

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

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

Quantity – OnValidate and Unit Price – OnValidate Triggers

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

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

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

Unit of Measure Code – OnValidate Trigger

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

Demonstration: Reviewing the Seminar Comment Line Table and Pages

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

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

Demonstration Steps

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

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

Setup.NewLine Function

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

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

```
    Date := WORKDATE;
```

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

Lab 3.2: Create Seminar Registration Pages

Scenario

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

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

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

Exercise 1: Import and Review the Pages

Exercise Scenario

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

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

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

Task 1: Importing the Starter Objects

High Level Steps

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

Detailed Steps

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

Task 2: Reviewing the Objects

High Level Steps

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

Detailed Steps

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

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



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

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



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

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



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

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



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

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

Exercise 2: Completing the Document Pages

Exercise Scenario

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

Task 1: Completing the Seminar Registration Page

High Level Steps

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

Detailed Steps

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

2. Add the subpage for the **Seminar Registration Subform** page, and name the control SeminarRegistrationLines.
 - a. Select the Seminar Room group.
 - b. Press F3 to insert a new row.
 - c. In the Type column, select Part.
 - d. In the SubType column, select **Page**.
 - e. Click the **Left** button one time or press SHIFT+ALT+LEFT to reduce line indentation by one level.
 - f. Press SHIFT+F4 to open the **Properties** window for the part.
 - g. In the PagePartID property, type "Seminar Registration Subform".
 - h. In the **Name** property, type "SeminarRegistrationLines".
3. Link the **SeminarRegistrationLines** page part to the main page.
 - a. In the SubPageLink property click the **AssistEdit** button to open the **Table Filter** window.
 - b. In the Field column, type "Document No."
 - c. In the Type column select FIELD.
 - d. In the Value column, type "No."
 - e. Click **OK** to accept the changes and close the **Properties** window.
4. Add the **Seminar Details FactBox** to the page.
 - a. Select the row containing the **Customer Details FactBox** page part control.
 - b. Press F3 to insert a new row.
 - c. In the Type column, enter "Part". Verify that the SubType value is set to Page automatically.
 - d. Press SHIFT+F4 to show the **Properties** window for the new page part control.
 - e. Set the PagePartID property to "Seminar Details FactBox".
 - f. Do not close the **Properties** window.
5. Link the **Seminar Details FactBox** page part to the main page.
 - a. In the SubPageLink property, click the **AssistEdit** button to open the **Table Filter** window.
 - b. In the Field column, type "No."
 - c. In the Type column, select FIELD.
 - d. In the Value column, type "Seminar No."
 - e. Press **OK** to close the **Table Filter** window and apply the filter.
 - f. Close the **Properties** window.

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



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

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

Module Review

Module Review and Takeaways

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

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

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

Test Your Knowledge

Test your knowledge with the following questions.

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

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

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

Module 3: Documents

4. What kind of a table is table **200000001, Object**?

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

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

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

- () True
- () False

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

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

Test Your Knowledge Solutions

Module Review and Takeaways

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

True

False

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

True

False

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

MODEL ANSWER:

CurrPage.SETSELECTIONFILTER

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

Special table

Virtual table

Temporary table

System table

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

MODEL ANSWER:

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

Module 3: Documents

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

() True

() False

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

MODEL ANSWER:

ListPart

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

MODEL ANSWER:

ProviderID

MODULE 4: POSTING

Module Overview

Transactional systems, such as Microsoft Dynamics NAV 2013, record past business events or transactions, and must safeguard the integrity of that information. Some examples of these business events are as follows:

- Purchases from vendors
- Sales to customers
- Consumption of raw materials in production
- Output of finished goods in production
- Usage of resources
- Payments from bank accounts to vendors

To make sure that information about past business events is always intact, Microsoft Dynamics NAV 2013 distinguishes between working data and posted data. *Working data* represents information about current or future transactions. Users can insert, change, or delete that information as needed. *Posted data* represents information about past business transactions. Users cannot insert, change, or delete that information. *Posting* is a process that moves the data from working tables into posted tables.

All functional areas of Microsoft Dynamics NAV 2013 provide very similar features for enabling users to enter the transaction data and process it. This similarity exists at all levels: user interface, data model, and process level. When you develop a new functional area, you must follow the standard concepts as much as possible to maintain a consistent user experience across the application.

Working tables in Microsoft Dynamics NAV 2013 consist of the following:

- Document tables
- Journal tables

Posted tables in Microsoft Dynamics NAV 2013 consist of the following:

- Posted document tables
- Ledger entry tables
- Register tables

There are two posting routines that move the data between these tables:

- Document posting routine
- Journal posting routine

Each of these routine comprises several codeunits.

The Seminar module now contains master tables and document tables to create registrations. The next step is to use the registration information to create ledger entries for seminars through posting routines.

Objectives

The objectives are:

- Explain the working and posting tables.
- Explain posting routines and their relationships.
- Create journal posting routines.
- Create document posting routines.
- Present the best practices for documenting changes to existing objects.
- Program for low impact on the application.

Prerequisite Knowledge

Before you begin to work on posting, it is important to know about journal tables, ledger tables, and some of the elements that are involved in posting.

Journal, Ledger and Register Tables and Pages

Journal tables, ledger tables, and posting codeunits are at the core of every posting process in Microsoft Dynamics NAV 2013.

Journal Tables

A journal is a temporary work area for the user. Users can insert, change, and delete all records in journals. A journal consists of three tables.

Table	Remarks
Journal Template	Journal templates represent transaction types, such as sales, cash receipt, inventory, or reclassification. There is typically only one journal template per transaction type, but users may decide to define more.
Journal Batch	Batches may represent various logical subtypes of the same transaction type. For example, users may have different cash receipt batches for different bank accounts or customer groups, or different inventory batches for different locations or item types. Sometimes, batches represent different users who use them to physically separate transactions that are entered by different users.
Journal Line	Journal line tables store the information about the transaction itself.

Lines belong to batches, and batches belong to templates. The "Journal Structure" figure shows how journal tables are related to one another.

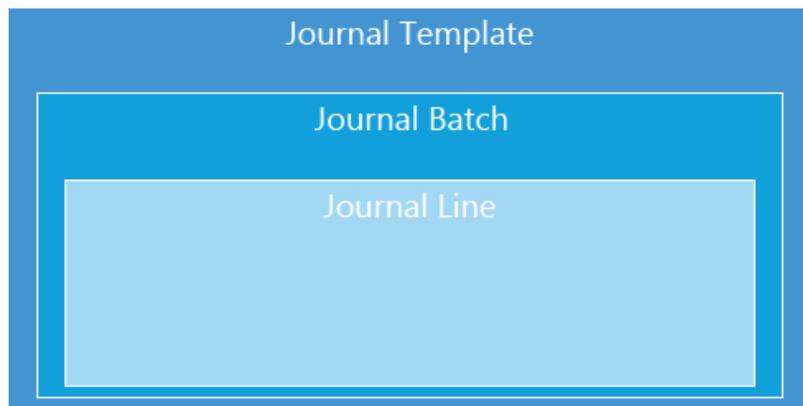


FIGURE 4.1: JOURNAL STRUCTURE

The Journal Page

The primary page to enter information into journals is called by the transaction type, and followed by the word *Journal*, for example: **Sales Journal**, **Cash Receipt Journal**, **Resource Journal**, or **Consumption Journal**. The page is of type worksheet, and uses **Journal Line** as its source table.

The primary key of a **Journal Line** table is a composite key, and consists of the **Journal Template Name**, **Journal Batch Name** and **Line No.** fields. The user never enters information into any of these fields directly. Instead, the **Journal** page sets the field according to the following rules:

- The **Journal** page that the user accesses sets the **Journal Template Name** field. If there are more templates for the same **Journal** page, then users must select the template when they start the **Journal** page. They cannot change the template unless they close and then reopen the **Journal** page.
- The journal page also sets the **Journal Batch Name** field. However, the user may change the **Batch Name** field at the top of the page.
- The **Line No.** field keeps each record in the same template and batch unique. The batch page sets the **Line No.** field automatically through the **AutoSplitKey** property.

The **Journal** page lets users enter and edit the journal lines that will later be posted into ledger tables. As long as the lines are in the journal, users can freely change or delete them, and they have no effect until the user posts the journal. Users can even leave the lines in the journal table indefinitely.

Ledger Tables

At the core of most functional areas in Microsoft Dynamics NAV 2013, there is a ledger table that keeps transaction history for that functional area. The ledger table is always called *Ledger Entry*. The **Ledger Entry** table is noneditable. Records in it are permanent and users cannot delete or change them, except in specific situations and by using special objects. You also cannot insert the entries directly into a ledger table. You can insert new entries into a ledger table only through a posting routine that moves the data from journal tables to ledger tables. After you post, the lines that were posted are deleted from the journal tables.

The primary key of every ledger table is the **Entry No.** field. There are many secondary keys, and most are compound. These keys are used by reports, pages, and FlowFields.

For most functional areas, there is a link between the **Ledger Entry** tables and the **General Ledger Entry** table. Because of this link, any modifications that you make directly to a ledger table can cause serious problems. Usually, the only way to undo such changes is to restore the most recent backup of the database.

The Ledger Entries Page

A page that shows the records from the **Ledger Entry** table is a List page, and is named after the ledger, followed by the words **Ledger Entries**, for example, **General Ledger Entries**, **Customer Ledger Entries** or **Item Ledger Entries**.

The **Ledger Entries** pages are typically noneditable, and do not allow insertions, modifications, or deletions. However, depending on the transaction type, they may allow certain changes that are typically related to business process specifics. For example, the **Customer Ledger Entries** and **Vendor Ledger Entries** pages allow changes to certain fields to provide putting entries on hold, or to manage the payment discounts after posting.

 **Note:** You do not protect the **Ledger Entry** tables directly by making the table fields noneditable. Instead, you must make sure that every page protects the table against unauthorized changes according to the business process requirements for the ledgers.

The Register Table and Page

Each functional area that includes a ledger also includes a register. A *register* is a table that keeps the history of all transactions. It is the core of the audit trail for the functional area. The table is always named after the ledger, followed by the word *Register*, for example **G/L Register**, **Item Register**, or **Resource Register**. The primary key is always the field **No.**

The **Register** table keeps the summary for the transaction, whereas the **Ledger Entry** table keeps the details for the transaction. There may be multiple ledger entries for each register line. For each transaction, the **Register** table always keeps track of the first and the last Ledger Entry record that is posted by the transaction. The **Register** table also keeps track of the **Creation Date**, **Source Code**, **User ID** and **Journal Batch Name** for the transaction.

For each **Register** table, there is a page that shows the records from the table. This is always named after the ledger, followed by the word *Registers*. The **Registers** page is a noneditable list page for the **Register** table, and always has the same name as its source table.

Every **Registers** page provides the quick means to show the ledger entries that result from the selected transaction in the register. The action is called after the ledger or the sub-ledger that it shows. For example, in the **Item Registers** page, there are **Item Ledger**, **Phys. Inventory Ledger**, **Value Entries**, and **Capacity Ledger** actions. Each of these actions runs a separate codeunit that receives the **Register** record, filters the ledger entries according to the **From Entry No.** and **To Entry No.** fields, and then shows the appropriate **Ledger Entries** page. This codeunit is always called after both the register page, and the ledger it shows. For example, clicking the **Item Ledger** action calls the Item Reg.-Show Ledger codeunit.

Journal Posting Codeunits

For each journal type, there is a group of codeunits that is responsible for moving the data from the journal tables into the ledger tables. These codeunits also make sure that all the data that is moved into the ledger is correct for each line and for the entire table. That group of codeunits is frequently called a *posting routine*. A posting routine performs the following tasks:

- Takes journal lines and checks them.
- Converts journal lines to ledger entries.
- Inserts journal lines into the ledger table.
- Makes sure that all posted transactions are consistent.

Although there are many types of posting routines in Microsoft Dynamics NAV 2013, they all follow the same data structure and architectural principles.

The Post Line Codeunit

The primary codeunit that does the work of posting for a particular journal is named after the journal name followed by the words *Post Line*, for example Gen. Jnl.-Post Line or Res. Jnl.-Post Line. The primary goal of a Post Line codeunit is to transfer the information from the **Journal Line** table into the **Ledger Entry** table, although it also performs other functions, such as calculations and data checking.

 **Note:** Depending on the business process that it handles, the Post Line codeunit may even post to multiple ledgers at the same time. For example, the Gen. Jnl.-Post Line codeunit posts information into general, customer, vendor, bank account, and fixed asset ledgers.

Journal Posting Companion Codeunits

For each type of posting routine (General Ledger, Item, Resource, and so on), the Post Line codeunit has two companion codeunits.

Codeunit	Purpose
Check Line	<p>Checks each journal line before it is posted. It receives the journal line as a parameter, and never reads it from the database. Check Line may read the related data from the database, however, it never writes any data back to the database. It checks for any conditions that may cause the posting to fail. It runs before the posting process starts to make sure that the posting process does not begin if there are any errors.</p> <p>The posting process in the Post Line codeunit performs many write operations. It also adds many locks, some of them explicit, so that the Check Line guarantees the highest possible concurrency between transactions. The posting process causes the problematic journal to fail before any locks are added.</p> <p>This codeunit is called by the Post Batch codeunit, but also by the Post Line codeunit.</p>
Post Batch	<p>The Post Batch codeunit repeatedly calls the Check Line codeunit to check all lines. If this check succeeds, then Post Batch repeatedly calls the Post Line codeunit to post all lines. The Post Batch codeunit is the only one that actually reads or updates the Journal table. The other codeunits use the Journal record that is passed into them. In this manner, you can call the Post Line codeunit directly from another posting codeunit without having to update the Journal table. The Post Batch codeunit is called only when the user clicks Post within the Journal page.</p>

C/SIDE Solution Development in Microsoft Dynamics® NAV 2013

By convention, the last digits of the object ID numbers of the posting codeunits are standardized.

Posting Codeunit	Ends with	Example
Check Line	1	11 Gen. Jnl.-Check Line
Post Line	2	12 Gen. Jnl.-Post Line
Post Batch	3	13 Gen. Jnl.-Post Batch

These codeunits do not require any user input. This is because they can be called from other objects that are part of larger batch processes, or from outside Microsoft Dynamics NAV using web services. In these situations, the user interface is either not desirable or not possible. The Post Batch codeunit displays a dialog that shows the posting progress and lets the user cancel the posting.

Journal Posting Starter Codeunits

Posting is a complex, and frequently time-consuming process that requires exclusive access to the data. Therefore, it must run without interruption so that posting codeunits do not allow any kind of user interaction. If there is any input that must be provided to the posting process, users must provide that input at the very beginning of the process.

Any user interaction during posting is handled by another set of codeunits:

Codeunit	Description	Object ID ends with	Example
Post	Asks the user whether to post, and then calls Post Batch.	1	231, Gen. Jnl.-Post
Post + Print	Asks the user whether to post, then calls Post Batch, and then calls the Register Report.	2	232 Gen. Jnl.- Post+Print
Batch Post	Asks whether to post the selected batches and then repeatedly calls Post Batch for each selected batch. Users can run this codeunit only from the Journal Batches page.	3	233, Gen. Jnl.-B.Post

Codeunit	Description	Object ID ends with	Example
Batch Post + Print	Confirms that the user wants to post the selected batches, then calls Post Batch for each selected batch, and then calls the Register Report.	4	234, Gen. Jnl.-B. Post+Print

The Journal Posting Process

The journal posting process involves one of the following starter codeunits:

- Post Batch
- Check Line
- Post Line

These three codeunits are the most important components of any posting routine, because they run the bulk of the business logic of transaction posting for a functional area.

Check Line Codeunit

As its name suggests, the Check Line codeunit checks the Journal Line that is passed to it. It does so without reading from the database server.

Before checking any of the fields, this codeunit makes sure that 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.

The last thing that the codeunit verifies is the validity of the dimensions for the journal line. The codeunit does so by calling the DimensionManagement codeunit. If the codeunit does not stop the process with an error, then the journal line is accepted, and the posting continues.

Post Line Codeunit

The Post Line codeunit is responsible for actually writing the journal line to the ledger. It only posts one journal line at a time, and it does not examine previous or upcoming records.

The function that runs the bulk of work in this codeunit is the **Code** function.

The OnRun trigger of the Post Line codeunit is usually never called, but it was called in earlier versions of the product, and is retained for backward compatibility. Instead, other codeunits call the **RunWithCheck** function that first calls the Check Line codeunit, and then calls the **Code** function.

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

Next, the codeunit checks the important table relations. This requires reading the database (by using the **GET** functions). This is why you do it here instead of in **Check Line**.

Before writing to the ledger, the Post Line writes to the register. The first time that the program runs through the Post Line codeunit, it inserts a new record in the **Register** table and sets the **From Entry No.** field to link to the first entry that is posted for the transaction. In every successive run through the Post Line codeunit, the program changes the record by incrementing the **To Entry No.** field.

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.

The last thing that the codeunit does is to increment the variable that holds the next entry number by one. Therefore, when the codeunit is called again, the next entry number is ready.

Post Batch Codeunit

The Post Batch codeunit is responsible for posting all the lines that belong to the same template and batch. Only one record variable for the journal is actually passed to this codeunit. However, the codeunit starts by filtering down to the template and batch of the record that is passed in. Then it determines how many records are in the batch. If there are no records, the codeunit exits without an error. The calling routine then notifies the user that there is nothing to post.

 **Note:** The Post Batch codeunit always respects any filters that the user has set on the **Journal Line** table in the **Journal** page. This allows users to only post sections of a batch, instead of the whole batch.

The Post Batch codeunit can then begin checking each journal line in the batch by calling the Check Line codeunit for each line. As soon as all 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 two times: one time for the Check Line codeunit, and again for the Post Line codeunit.

When the Check Line codeunit checks the validity of a single line, the Post Batch codeunit is responsible for checking the interrelation and consistency of all the lines that are being posted. For example, the Gen. Jnl-Post Batch codeunit also makes sure that the journal lines balance to zero. If a similar check is necessary, it usually occurs as a separate loop through the lines after the Check Line codeunit and before the Post Line loop.

Module 4: Posting

The codeunit may perform other functions, 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** field and the **Document No.** field and replace any parameters with the correct values, for example %1 = day, %2 = week, and %3 = month.

If the template is not recurring, the codeunit deletes all the journal lines after they are successfully posted.

The “Post Batch Process and Data Flow” diagram outlines the steps in the Posting Routine when the Post Batch codeunit is called.

The following diagram shows the logic of a Post Batch codeunit.

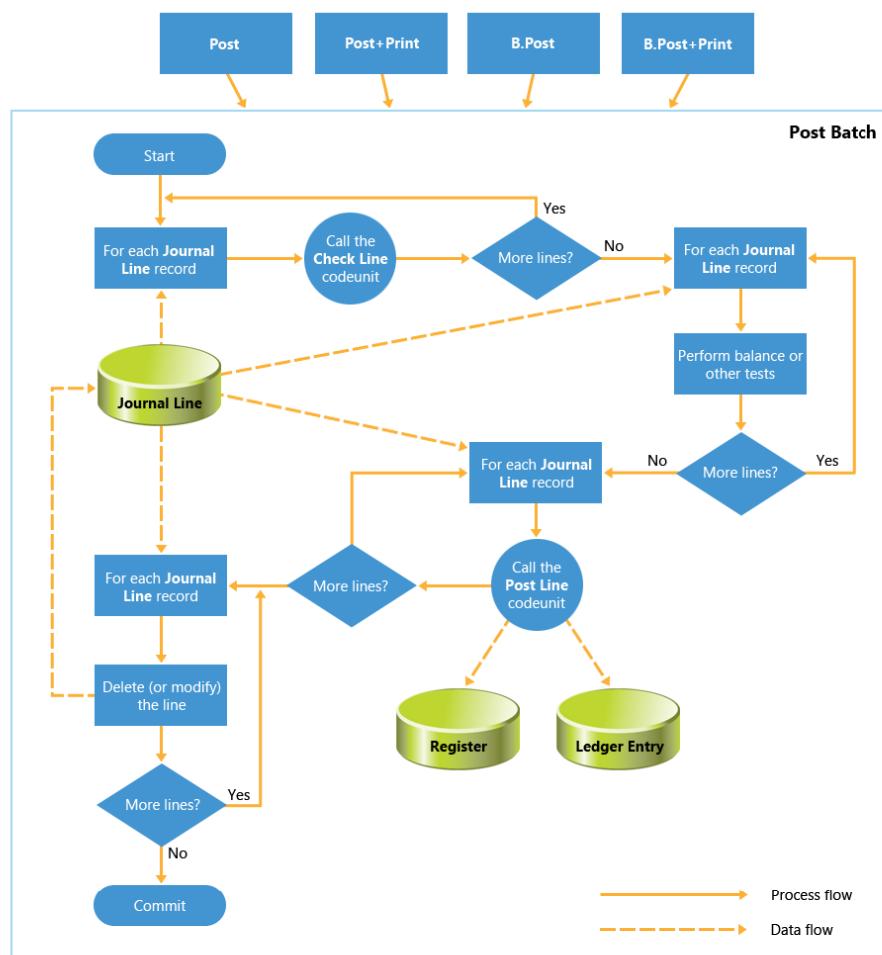


FIGURE 4.2: POST BATCH PROCESS AND DATA FLOW

Example Posting Routine

For a better understanding of how posting routines are written, you may review one of the simpler posting routines, such as the Resource Journal posting routine.

Check Line

Design the Res. Jnl.-Check Line codeunit (211). Notice that the OnRun trigger gets the **General Ledger Setup** record, and then calls the **RunCheck** function. The **RunCheck** function performs the following required checks:

- If the line is empty, the codeunit skips additional checking and exits without error.
- It checks if the posting date is within the allowed posting date range.
- If the line is related to a time sheet, the **RunCheck** function performs the time sheet checks by calling the Time Sheet Management codeunit.
- It calls functions from the DimensionManagement codeunit to check the dimension combinations and dimension posting rules.

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

Post Line

Design the Res. Jnl.-Post Line codeunit (212). Notice that the OnRun trigger gets the **General Ledger Setup** record, and then calls the **RunWithCheck** function. The **RunWithCheck** function then calls the **Code** function. Most of the posting work is performed in the **Code** function.

Like the Check Line codeunit, the Post Line codeunit also skips empty lines by exiting. This guarantees that empty lines are not inserted into the ledger. If the line is not empty, it calls **Check Line** to verify that all required 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. Before writing to the ledger, the Post Line codeunit writes to the register. On the first run, the codeunit adds a new record to the **Register** table. For every successive run through the Post Line codeunit, it increments the **To Entry No** field.

Then the codeunit takes the next entry number and the values from the journal line and puts them into a **Resource Ledger Entry** record. Finally, it inserts the **Resource Ledger Entry** record.

Post Batch

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

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

The Post Batch codeunit then loops and checks each journal line in the recordset by calling the Check Line codeunit. As soon as all lines are checked, they enter another loop which posts the records by calling the **RunWithCheck** function of the Post Line codeunit for each line.

Unlike the General Journal, there are no interdependencies between Resource Journal lines. Therefore no checks, such as checking the balance must be done.

Finally, this codeunit calls the UpdateAnalysisView codeunit to update any Analysis Views that require updates on posting.

Document Posting Routines

In Microsoft Dynamics NAV 2013, documents provide a simple way to process complex transactions. A document frequently combines multiple transactions into a single transaction. These transactions would be multiple individual transactions if posted from separate journals. By combining these transactions, documents not only simplify work for users, but also guarantee more transactional integrity than journals. This is known as *cross-functional transactional integrity*.

To better understand how a document posting routine works and what its components are, consider the following example of a sales order with three sales lines:

- Line 1: Selling a G/L account – for example, this line may add a surcharge or freight
- Line 2: Selling an item – for example, a computer
- Line 3: Selling a resource – for example, time that an employee spends custom building the computer

When the user posts the document, the program generates an entry that debits the Accounts Receivable Account in the general ledger (G/L). Each document line generates a separate G/L entry for that line. At the same time, the document posting routine generates an entry for the Item and Resource journals, and the General Journal for the Customer.

C/SIDE Solution Development in Microsoft Dynamics® NAV 2013

When these journal entries are posted, they are posted as if the user had entered them into the three separate journals. The biggest difference is that the journal records are posted individually. This enables the Sales-Post routine to bypass the Post Batch codeunit and call the Post Line codeunit directly.

In this example, the document posting routine calls the Gen. Jnl.-Post Line codeunit at least two times: one time for the Item Jnl.-Post Line codeunit, and one time for the Res. Jnl.-Post Line codeunit.

A sales document is posted primarily by the Sales-Post codeunit (80). The whole batch of sales documents can be posted by calling the **Batch Post Sales Invoices** report (297). Be aware that this report is for invoices only. There is a separate report for each document type, such as orders or credit memos.

These reports call the Sales-Post codeunit repeatedly for each document. For this to work, the Sales-Post codeunit must not interact with the user. In fact, the Sales-Post codeunit is never called directly by a page. The page calls the Sales-Post (Yes/No) codeunit (81), the Sales-Post + Print codeunit (82), or one of the reports that was mentioned previously. These other codeunits or reports in turn interact with the user. This is usually to obtain user confirmation before posting, and then to call the Sales-Post codeunit as appropriate.

The "Sales-Post Data Flow" diagram shows the data flow of the Sales-Post codeunit when a G/L Account, an Item, and a Resource line are included in a sales invoice.

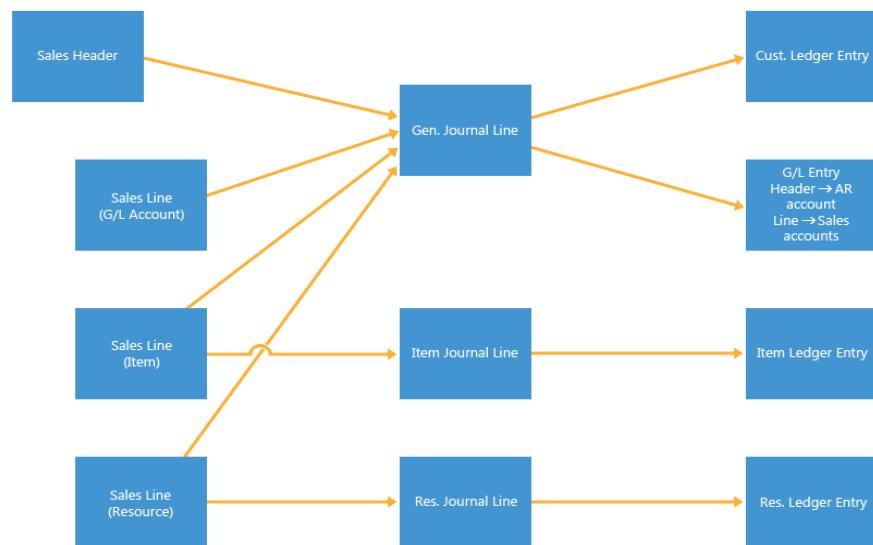


FIGURE 4.3: SALES-POST DATA FLOW

Document Posting Codeunit

Codeunit 80 posts sales documents. When a user ships and invoices a sales order, much of the work is performed in codeunit 80. This codeunit performs the following tasks:

- Determines the document type and validates the information on the sales header and lines.
 - The codeunit determines the posted document numbers and updates the header.
 - This section ends with a **COMMIT** function call.
- Locks the appropriate tables.
- Inserts the **Shipment Header** and **Invoice** or **Credit Memo Header**.
- Clears the **InvPostingBuffer**, a temporary table that is based on the **Invoice Post. Buffer** table.
- Within the main **REPEAT** loop, it iterates through all sales line, and checks each line with its matching **Sales Shipment Line** (if the line is previously shipped).
 - If the line type is Item or Resource, it is posted through the appropriate journal. The line is then added to the posting buffer. When you add to the posting buffer, a new line may be inserted, or you can update a current line. The buffer makes sure that there are as few **G/L Entry** records as possible that result from a single transaction. Therefore, it always combines lines that have the same values for the **G/L Account No.** field, the same dimension values, the same posting groups, and some more important fields.
 - If the line is related to a job, codeunit 80 posts a journal line through the Job Journal.
 - If there is no shipment line, codeunit 80 inserts one.
 - Finally, codeunit 80 copies the **Sales Line** to the **Invoice Line** or **Credit Memo Line** (the posted tables).
- Posts all entries in the **Posting Buffer** temporary table to the **General Ledger** table.
 - 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**.
- The routine then checks whether there is a balancing account for the header. This corresponds to an automatic payment for the invoice.
- Updates and deletes the **Sales Header** and **Lines** and commits all changes.

Documentation in Existing Objects

When you make changes to an existing object, enter a note in the Documentation trigger, in the same manner as for new objects that were discussed in earlier modules. The note should contain a reference number, the date the modification is completed, the name of the developer who made the modification, and a short description of the change.

Following is an example taken from the **Resource** table, with notes from Module 2, Lab 1.

Documentation Trigger Example

CSD1.00 - 2012-06-15 - D. E. Veloper

Module 2 - Lab 1

- Added new fields:
- Internal/External
- Maximum Participants

Notice that documentation in an existing object resembles the documentation in a new object. The details of these notes and the way that they are formatted can vary from one developer to another. They may also be the subject of an organization-wide set of standards. Understand that these notes are necessary to keep track of the changes that are made to objects over the life span of the objects in a Microsoft Dynamics NAV 2013 application.

Code Comments

Together with the general comments that are provided in the Documentation trigger, it is important to provide comments in the code at the lines where a change is made. Do this only when changing 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, mark a single changed line of code as follows.

Single Line Modification

```
//CSD1.00>  
  
//CheckCustBlockage("Sell-to Customer No.",TRUE);  
  
CheckCustBlockage("Sell-to Customer No.",FALSE);  
  
//CSD1.00<
```

If you add or change a whole block of code, mark the change as follows.

Multiple Lines Modification

```
//CSD1.00>  
  
//TESTFIELD("Document Type");  
  
//TESTFIELD("Sell-to Customer No.");  
  
//TESTFIELD("Bill-to Customer No.");  
  
//TESTFIELD("Posting Date");  
  
//TESTFIELD("Document Date");  
  
CheckDocumentTypeFields("Document Type");  
  
IF "Sell-to Customer No." <> "Bill-to Customer No." THEN  
  
    CompareCustomerDimensions("Sell-to Customer No.", "Bill-to Customer No.");  
  
//CSD1.00<
```

When removing standard code, mark the removed lines as follows.

Code removal

```
//CSD1.00>  
  
//SalesLine.SETFILTER(Quantity,'<>0');  
  
//SalesLine.SETFILTER("Return Qty. to Receive",'<>0');  
  
//CSD1.00<
```

Never delete the original Microsoft Dynamics NAV 2013 code. The goal of code comments is to preserve the original code, even when you change the business logic or remove the business logic. Preserving the original code performs the following three important tasks:

- It shows the original code before any customization.
- It makes any changes more obvious in the source code of the object.
- It streamlines the upgrade process by enabling the upgrade tools to match the original code with the new version code.

 **Note:** Even though braces are a valid way to comment out multiple lines of code, use them sparingly. Comments made with braces are not color-coded in green, and are inconspicuous when you are viewing the code. Also, during upgrade projects they make the changes less obvious than the changes that you make with // comments.

Performance Issues

When you write large posting routines, it is important to program to maximize performance. There are several steps to program a solution in Microsoft Dynamics NAV that will improve performance.

Table Locking

Most of the time, you do not have to be concerned about transactions and table locking when you develop applications in Microsoft Dynamics NAV Development Environment, because the SQL Server adds necessary locks to affected tables as soon as you start inserting, changing, or deleting data. However, there are some situations when you must explicitly lock a table to guarantee process or transaction integrity.

For example, suppose that in the beginning of a function, you read the data from a table, and then later use that data to perform various checks and calculations. Then, you write the record back to the database, based on the result of this processing. The values that you retrieved at the beginning must be consistent with the final data in the table. In short, other users must be unable to update the table while a function is busy doing the calculations.

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



Note: The **LOCKTABLE** function does not necessarily lock the table. The SQL Server may decide to lock only the rows that you read, or to escalate the locks to greater levels, such as a page or even a table level. Regardless of whether the SQL Server locks the table or only the sections of it, the **LOCKTABLE** function guarantees data consistency until you either call the **COMMIT** function or roll back the transaction.

Reducing Impact on the Server

Good code design minimizes the load on the server. There are several ways to achieve this including the following:

- Try to use the **COMMIT** function as little as possible. This function is handled automatically by the database for most circumstances.
- Use the **LOCKTABLE** function only when it is necessary. Remember that an insert, change, rename, or delete function automatically locks the table. So for most situations you do not have to lock the tables.
- Structure the code so that you do not have to examine the return values of the **INSERT**, **MODIFY**, or **DELETE** functions. When you use the return value, the server must be notified immediately to obtain a response. Therefore, if they are not necessary, do not examine the return values of **INSERT**, **MODIFY**, or **DELETE** functions.
- Use the **CALCSUMS** and **CALCFIELDS** functions when possible to avoid examining records to total values. Use the **SETAUTOCALCFIELDS** function when you must obtain the value of a FlowField for every single row in a loop.
- Always avoid too many round trips to the server.

Reducing Impact on Network Traffic

Consider setting keys and filters, and then use **MODIFYALL** or **DELETEALL** functions. These functions send only one command to the server, instead of getting and deleting or changing many records one by one using the **MODIFY** and **DELETE** functions.

Because **CALCSUMS** and **CALCFIELDS** can both take multiple parameters, you can use these functions to perform calculations on several fields that have a single function call.

Posting Seminar Registrations

In this section the client's functional requirements are analyzed and a solution is designed and implemented.

Solution Design

The CRONUS International Ltd. functional requirements outline that when a seminar is completed, users must be able to move the seminar registration information into the transaction history, and disable any further modification of this information. This requirement indicates that there must be a posting process that is involved with seminar registrations. When you apply the customer's language to the terminology of Microsoft Dynamics NAV 2013, this requirement states that users must be able to post the seminar registration information.

Another requirement further clarifies the customer's business need for a transaction history for seminars that must include the following:

- Details of participants, instructors, and rooms that are utilized during the seminars
- Information about additional charges

This information will be the basis for seminar cost analytical and statistical reporting. This indicates that the detailed information about posted transactions must include all the information that is contained in the seminar registration document. Microsoft Dynamics NAV terminology calls these transactions the *Ledger Entries*.

The final requirement details how the seminar registration information must integrate with the availability planning functionality for instructors and rooms. It also must provide the basis for automatic invoicing of customers.

When seminar registration is posted, the resource ledger entries should be generated for the instructor and room resources. The solution must provide the registration posting functionality that creates transaction data from which users can view history, analyze statistics, and create invoices.

When you introduce posting functionality, you must follow Microsoft Dynamics NAV 2013 standard conceptual, data model, and user interface principles. This means that you must provide at least the journal posting infrastructure that consists of the following:

- Journal tables
- Ledger tables
- Register tables
- Journal posting codeunits

Depending on the complexity of the processes that you must support, you may have to extend the functionality to also include the following features:

- Posted document tables
- Document posting codeunits

The “Data Flow in Seminar Registration Posting” figure shows the entities that are involved in the seminar registration posting process and the data flow between them.

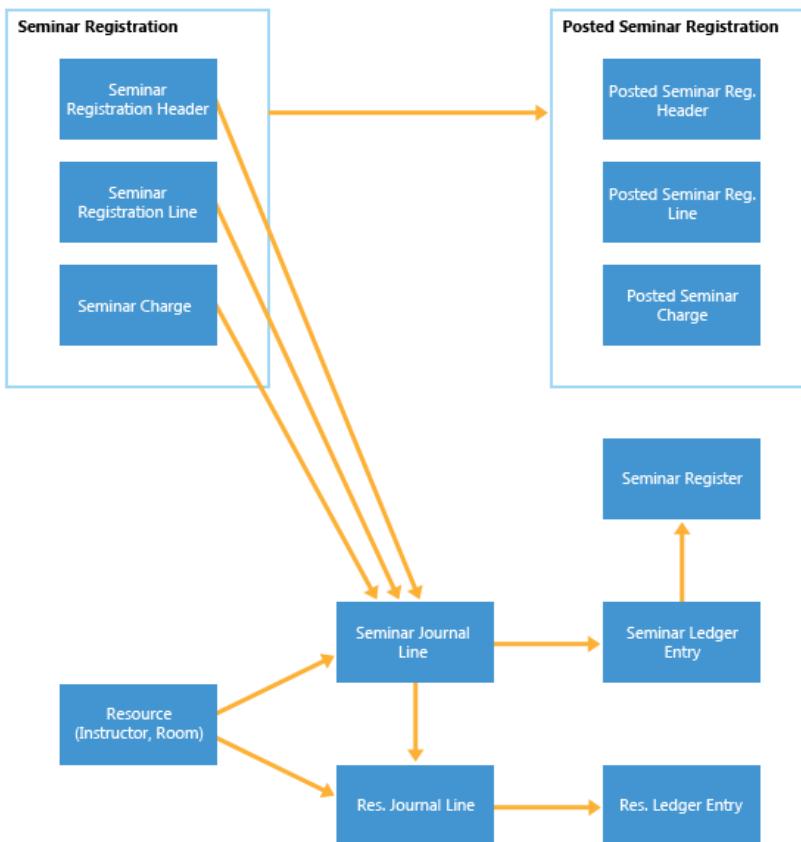


FIGURE 4.4: DATA FLOW IN SEMINAR REGISTRATION POSTING

Development

You must develop the tables, pages, and codeunits to enable the seminar registration posting process and keep the transaction history. All tables, pages, and codeunits must follow the standard Microsoft Dynamics NAV 2013 principles, and must provide all functionality that users experience with other posting routines and transactional history features.

Tables

To support the posting process and to keep the transaction history for the Seminar Management module, you must create the following new tables.

Table	Remarks
123456718 Posted Seminar Reg. Header	Holds the information for the completed (posted) seminar. Takes the data from the Seminar Registration Header table during posting.
123456719 Posted Seminar Reg. Line	Holds the detailed information for the completed (posted) seminar. Takes the data from the Seminar Registration Line table during posting.
123456721 Posted Seminar Charge	Holds charges that are related to the completed (posted) seminar.
123456731 Seminar Journal Line	Lets you post the seminar ledger entries.
123456732 Seminar Ledger Entry	Keeps the transaction details for all posted seminars.
123456733 Seminar Register	Keeps the transaction log for posted seminars.

You must also change the following tables.

Table	Remarks
203 Res. Ledger Entry	Add fields to link the resource ledger entries to the seminars and to the posted seminar registration documents.
207 Res. Journal Line	Add fields to support posting of seminar information during resource journal posting.
242 Source Code Setup	Add a field to support the audit trail source code for the seminar registration transaction history.

Pages

The pages for the seminar registration posting and the navigation between them reflect the relationships that are shown in the “Data Flow in Seminar Registration Posting” diagram. Design the simplest pages first and then integrate them with the more complex pages.

Add the Seminar Management group and the **Seminar** field to the **Source Code Setup** page:

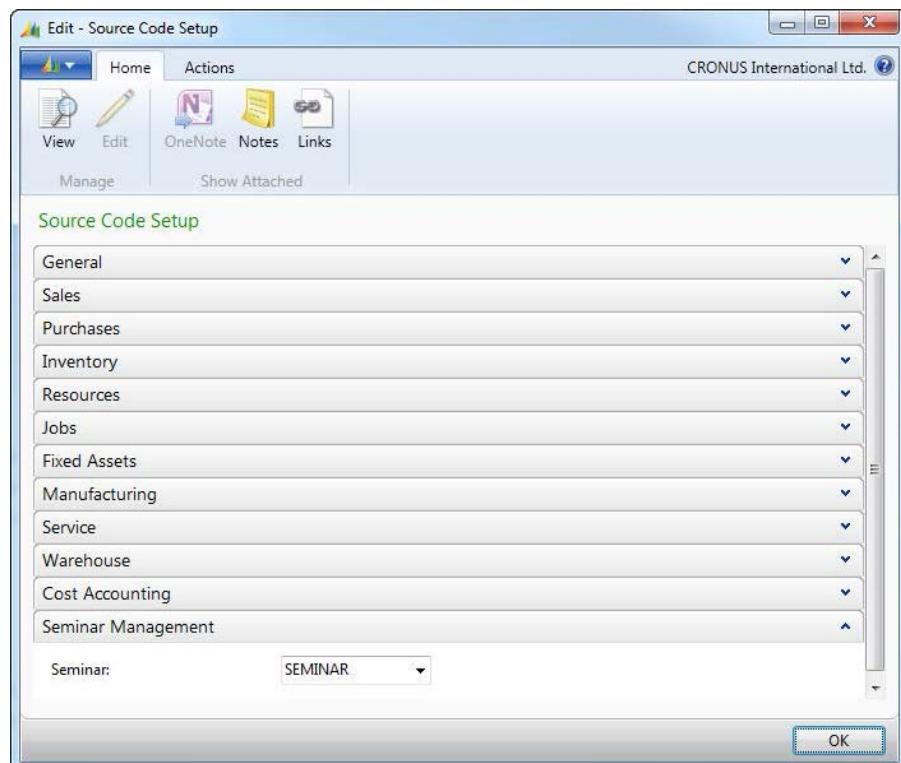


FIGURE 4.5: THE SOURCE CODE SETUP PAGE (279)

The **Seminar Ledger Entries** page displays the ledger entries:

C/SIDE Solution Development in Microsoft Dynamics® NAV 2013

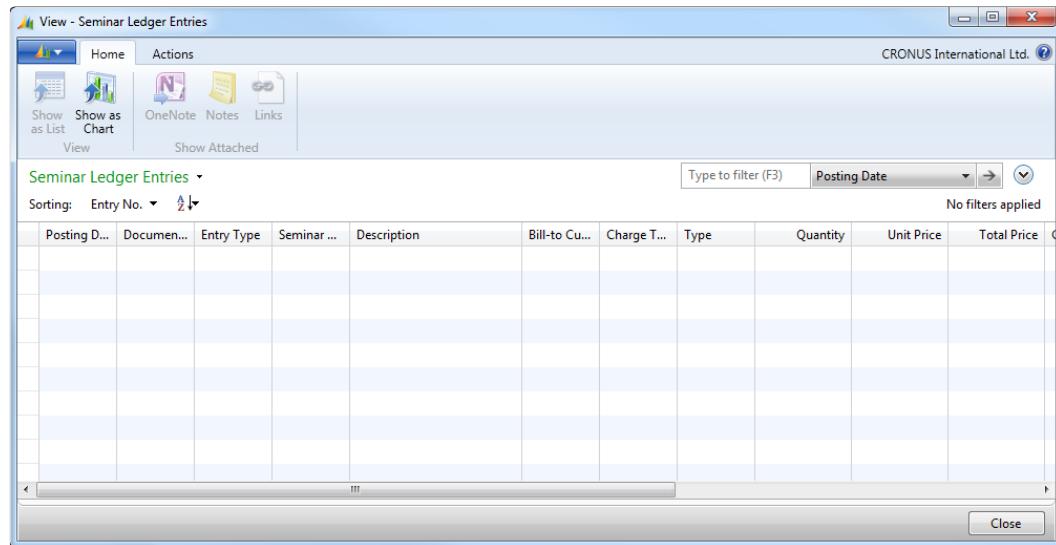


FIGURE 4.6: THE SEMINAR LEDGER ENTRIES PAGE (123456721)

The "Seminar Registers Page" figure displays the registers that are created when seminar registrations are posted.

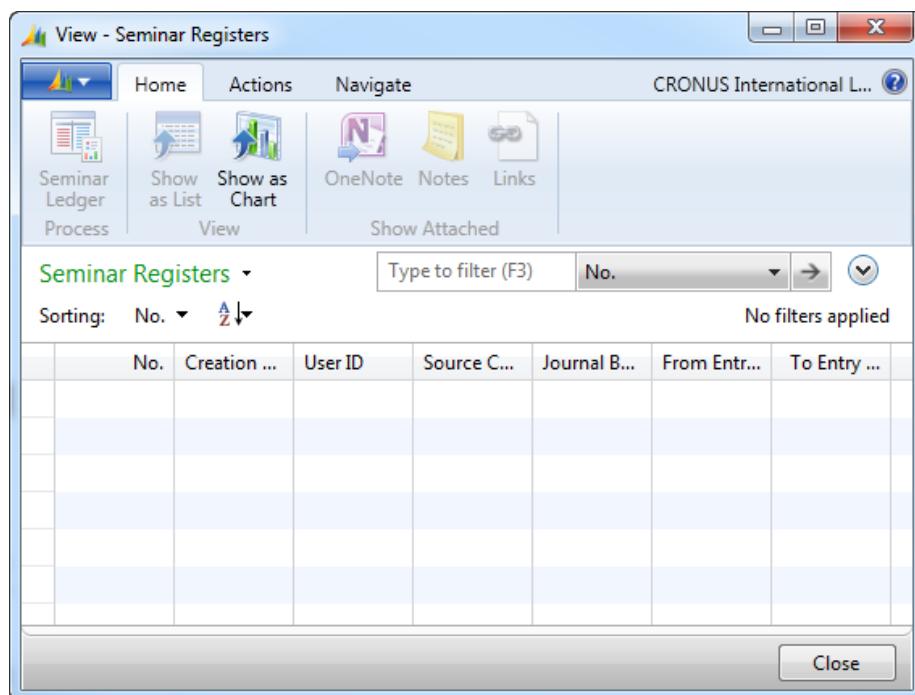


FIGURE 4.7: THE SEMINAR REGISTERS PAGE (123456722)

Module 4: Posting

The "Posted Seminar Charges Page" image shows the charges that are related to a posted seminar registration.

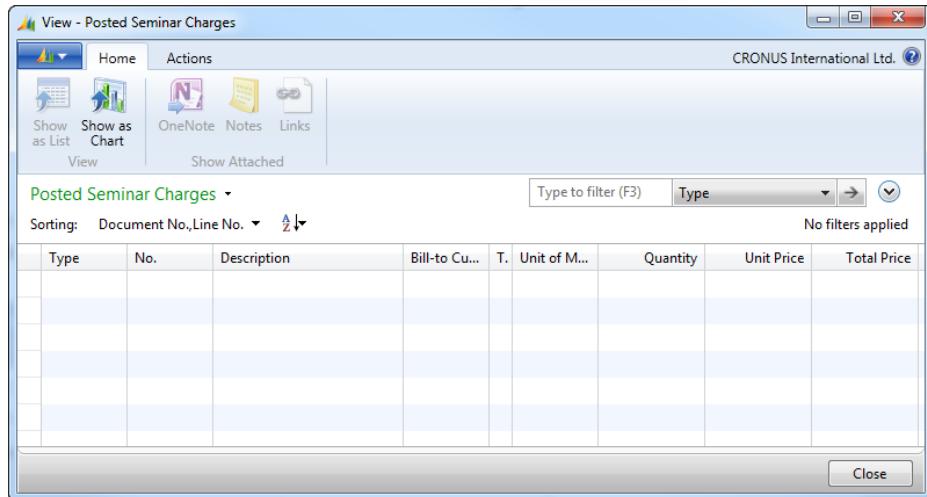


FIGURE 4.8: THE POSTED SEMINAR CHARGES PAGE (123456739)

The "Posted Seminar Reg. Subform page" image shows the lines page for the **Posted Seminar Registration** document page.

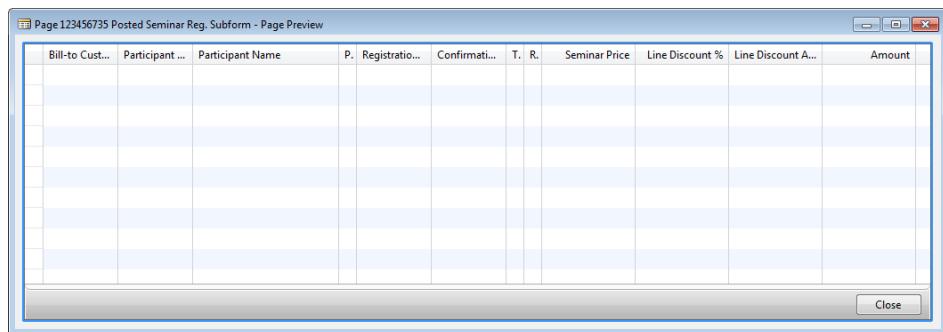


FIGURE 4.9: THE POSTED SEMINAR REG. SUBFORM PAGE (123456735) IN PAGE PREVIEW

The "Posted Seminar Registration Page" image shows the posted seminar registration document.

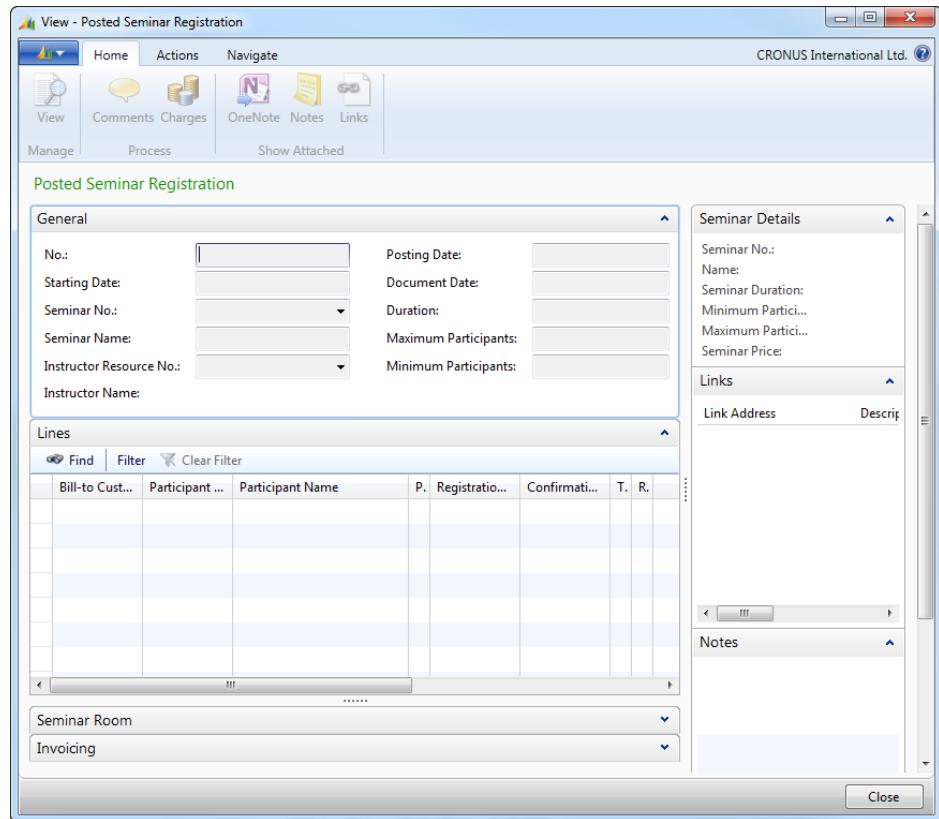


FIGURE 4.10: THE POSTED SEMINAR REGISTRATION PAGE (123456734)

The “Posted Seminar Reg. List Page” figure displays a list of posted seminar registrations.

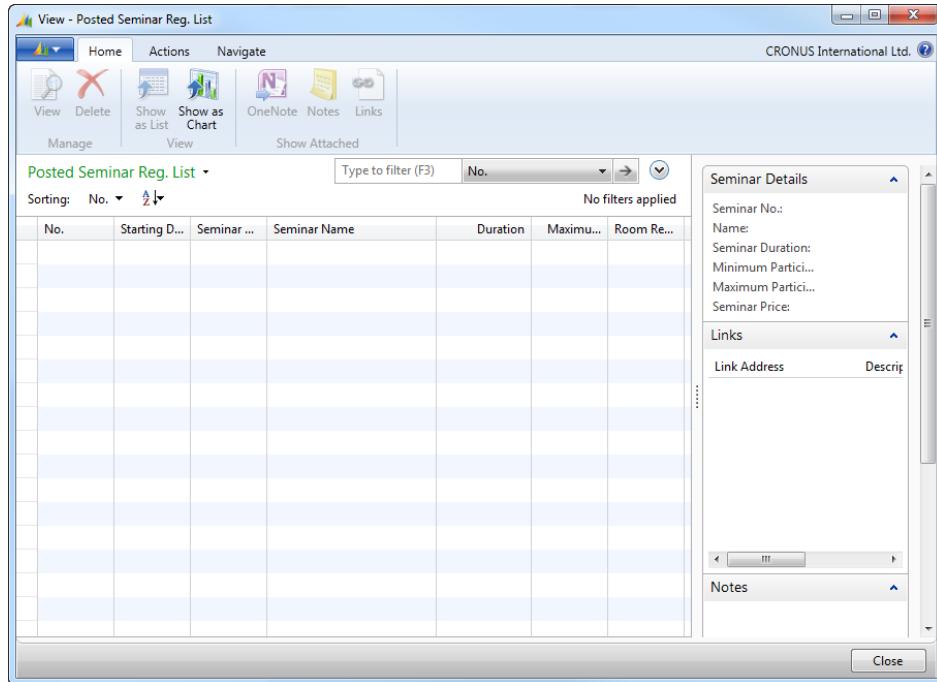


FIGURE 4.11: THE POSTED SEMINAR REG. LIST PAGE (123456736)

Codeunits

As in all journal postings, the journal posting codeunits check Seminar Journal lines and post them. However, unlike some posting codeunits (such as General Journal), a codeunit that posts a batch of these journal lines is not required because posting batches is not required for this solution.

You must develop the following journal posting codeunits.

Codeunit	Remarks
123456731 Seminar Jnl.-Check Line	<ul style="list-style-type: none"> • Verifies the data validity of a seminar journal line before the posting routine posts it by doing the following: • The codeunit checks that the journal line is not empty and that there are values for the Posting Date, Instructor Resource No., and Seminar No. fields. • Depending on whether the line is posting an Instructor, a Room, or a Participant, the codeunit checks that the applicable fields are not blank. • The codeunit also verifies that the dates are valid.

Codeunit	Remarks
123456732 Seminar Jnl.-Post Line	Performs the posting of the Seminar Journal Line . The codeunit creates a Seminar Ledger Entry for each Seminar Journal Line and creates a Seminar Register to track which entries are created during the posting

Change the Res. Jnl.-Post Line codeunit to make sure that the **Seminar No.**, and the **Seminar Registration No.** fields are recorded in the **Res. Ledger Entry** table.

Finally, you must develop the codeunits for the seminar registration document posting.

Codeunit	Remarks
123456700 Seminar-Post	<p>Posts the complete seminar registration that includes the resource posting and seminar posting.</p> <ul style="list-style-type: none"> The codeunit transfers the comment records to new comment records that correspond to the posted document. The codeunit also copies charges to new tables that contain posted charges. The codeunit creates a new Posted Seminar Reg. Header record and Posted Seminar Reg. Line records. The codeunit then runs the job journal posting, and posts seminar ledger entries for each participant, for the instructor, and for the room. Finally, the codeunit deletes the records from the document tables. This includes the header, lines, comment lines, and charges.
123456701 Seminar-Post (Yes/No)	Interacts with users, and confirms that they want to post the registration. If users confirm the posting, the codeunit runs the Seminar-Post codeunit.

Lab 4.1: Reviewing and Completing the Journal and Ledger Tables

Scenario

Isaac is a junior developer working on the team that is implementing Microsoft Dynamics NAV 2013 for CRONUS International Ltd. Isaac is in charge of developing the base version of tables and user interface objects.

When he developed the seminar registration tables and pages, he only provided you with the text file. This made it more difficult to review the contents of the input file, because you had to review the text file by using a text-editing tool, such as Notepad. You advised Isaac to always provide both the text and .fob versions of the object import file to make the review process simpler.

Isaac created the seminar journal and ledger tables and pages, and delivered both the .fob and text files that contain the objects. As Isaac's supervisor, you now must review the objects to make sure that they follow all Microsoft Dynamics NAV 2013 architectural principles and best practices. You must make any necessary corrections to the tables, their properties and fields, and their code.

By this point, you should already be familiar with the basics of the Microsoft Dynamics NAV 2013 Development Environment functions and features. Therefore, the detailed instructions only give you descriptions of the actions that you must do, instead of giving you detailed steps that are suitable for beginning users.

The following table contains some examples.

Instead of...	... the instructions state
<ol style="list-style-type: none"> On the View menu, click Properties. In the Editable property, enter "No". Close the Properties window. 	<ol style="list-style-type: none"> Set the Editable property to No.
<ol style="list-style-type: none"> On the View menu, click C/AL Locals. On the Return Value tab, in the Return Type field, enter "Boolean". Close C/AL Locals. 	<ol style="list-style-type: none"> Set the Return Value for the function to return Boolean type.
<ol style="list-style-type: none"> In Object Designer, click All. 	<ol style="list-style-type: none"> View all objects in Object Designer.

Instead of...	... the instructions state
1. In Object Designer, click Page . 2. Find and select the page 123456721, Seminar Ledger Entries . 3. Click Design .	1. Design the page 123456721, Seminar Ledger Entries .
1. On the File menu, click Save . 2. In the Save dialog window, make sure that the Compiled check box is selected, and then click OK . 3. Close the table.	1. Compile, save, and then close the table.

 **Note:** This applies only to the concepts that were covered earlier. If there is a new concept, the full detailed steps are provided. For any repeated concepts, only the descriptive instructions are provided. If you are still unsure about the detailed steps of a task, refer to the labs in earlier modules. If you have any questions, ask your instructor.

Exercise 1: Reviewing the Import File Contents and Importing the Objects

Exercise Scenario

Prior to importing objects, you should review the contents of the import file to make sure that no existing objects will be overwritten. Start the review process by importing a .fob file. This enables you to review the contents of the file before saving objects to the database.

 **Note:** You may notice that the import file does not include the **Seminar Journal Template** or **Seminar Journal Batch** table, or the **Seminar Journal** page. This is because the **Seminar Journal Template** is always posted in the background as a part of the document posting routine. For document posting, the template and the batch are always undefined. Therefore the tables are not needed.

Provide only the **Seminar Journal Template** and **Seminar Journal Batch** tables.
Provide a **Journal** page if users must access the journal directly from the client.

Task 1: Preview the .fob File Contents

High Level Steps

1. In Object Designer, open the Mod04\Labfiles\Lab 4.A - Starter.fob file in the Import Worksheet.
2. Make sure that all listed objects are new.
3. Close the Import Worksheet.

Detailed Steps

1. In Object Designer, open the Mod04\Labfiles\Lab 4.A - Starter.fob file in the Import Worksheet.
 - a. Open **Object Designer**.
 - b. On the **File** menu, click **Import**.
 - c. In the **Import Objects** dialog box, browse to Mod04\Labfiles\Lab 4.A - Starter.fob.
 - d. Click **Open**. The confirmation dialog box opens, and notifies you that no conflicts were found.
 - e. Click **No**, to open the Import Worksheet, as instructed by the dialog window.
2. Make sure that all listed objects are new.
 - a. Check the Action column for each row.
 - b. Make sure that the Action is set to Create for each row.
3. Close the Import Worksheet.
 - a. Click **Cancel** to close the Import Worksheet.



Note: Even though you could click **OK** to complete the import, you want to import the text file to review only the new objects and compile them manually. When you import from a text file, any new objects are obvious because they are not compiled.

Task 2: Import and Compile the Objects

High Level Steps

1. In Object Designer, import the Mod04\Labfiles\Lab 4.A - Starter.txt file.
2. Select and compile the imported objects.

Detailed Steps

1. In Object Designer, import the Mod04\Labfiles\Lab 4.A - Starter.txt file.
 - a. In Object Designer, on the **File** menu, click **Import**.
 - b. In the Import Objects dialog box, browse to Mod04\Labfiles\Lab 4.A - Starter.txt file.
 - c. Click **Open**.

2. Select and compile the imported objects.
 - a. View all objects in **Object Designer**.
 - b. Filter the view to only show the uncompiled objects.
 - c. Compile the objects.

Exercise 2: Reviewing the Seminar Journal Line Table

Exercise Scenario

After importing the objects, you must make sure that all objects follow the best practices and standard Microsoft Dynamics NAV 2013 principles. Start with the **Seminar Journal Line** table.

Task 1: Review Table and Field Properties

High Level Steps

1. Clear all filters in Object Designer.
2. Make sure that all important standard journal fields are present in the table 123456731, **Seminar Journal Line** table, and that the primary key consists of the **Journal Template Name**, **Journal Batch Name** and **Line No.** fields in the correct order.

Detailed Steps

1. Clear all filters in Object Designer.
 - a. In Object Designer, click **View > Show All**, or press SHIFT+CTRL +F7.

2. Make sure that all important standard journal fields are present in the table 123456731, **Seminar Journal Line** table, and that the primary key consists of the **Journal Template Name**, **Journal Batch Name** and **Line No.** fields in the correct order.
 - a. Design the table 123456731, **Seminar Journal Line**.

Module 4: Posting

- b. Make sure that the following fields are present in the table.

Field	Type	Remarks
Journal Template Name	Code[10]	
Journal Batch Name	Code[10]	This field must exist in all journal line tables, but is frequently located after all the fields that describe the transaction.
Posting Date	Date	Specifies the date for the entry. This is the date that the transaction occurred.
Document Date	Date	Specifies the date for the document. By default this equals the Posting Date. Users can change this date because the transaction may be entered and posted on different dates.
Source Code	Code[10]	Specifies the source for the entry. Sources map directly to journal templates. Therefore, they all map to transaction types. This field forms the basis for the audit trail that Microsoft Dynamics NAV 2013 leaves for every transaction. Users cannot change this field.
Reason Code	Code[10]	Specifies the reason why the entry was posted. Users may change this field.



Note: There are many more fields in the table. Most of the other fields are specific to the seminar journal transactions.

- c. On the **View** menu, click **Keys**.
- d. Verify that the first key in the list is "Journal Template Name,Journal Batch Name,Line No."
- e. Close the **Keys** window.

Task 2: Review Table Code

High Level Steps

1. Check whether the table contains the necessary functions for the journal line tables.
2. Create and define the **EmptyLine** function. The function must return a Boolean value.
3. Enter the code that sets the **Document Date** field to the value of the **Posting Date** field when users edit the **Posting Date** field. Then save and close the table.

Detailed Steps

1. Check whether the table contains the necessary functions for the journal line tables.
 - a. In the **C/AL Globals** window, check the **Functions** tab to determine whether there are any functions that were defined.

 **Note:** All Journal Line tables must contain the **EmptyLine** function, which is called during posting to make sure that only non-empty lines are posted.

2. Create and define the **EmptyLine** function. The function must return a Boolean value.
 - a. Create the **EmptyLine** function, configure its **Return Value** to return Boolean type.
 - b. Open the **C/AL Editor** window to view the code for the table.
 - c. In the **EmptyLine** function trigger, enter the following code.

```
EXIT(  
    ("Seminar No." = '')  
    AND (Quantity = 0));
```

 **Note:** The number of conditions in the **EmptyLine** function depends on the complexity of the journal transaction. For the seminar journal, if both the **Seminar No.** and **Quantity** fields are undefined, then the journal line is considered empty.

3. Enter the code that sets the **Document Date** field to the value of the **Posting Date** field when users edit the **Posting Date** field. Then save and close the table.
 - a. In the Posting Date – OnValidate trigger, enter the following code.

```
VALIDATE("Document Date","Posting Date");
```

- b. Compile, save, and then close the table.

Exercise 3: Reviewing Other Tables

Exercise Scenario

After reviewing the **Seminar Journal Line** table and correcting the issues that caused the gap between Isaac's work and Microsoft Dynamics NAV 2013 standards and best practices, you want to review the remaining journal posting tables: the **Seminar Ledger Entry** and **Seminar Register** tables.

Task 1: Review the Seminar Ledger Entry Table

High Level Steps

1. Review all the fields in the table 123456732, **Seminar Ledger Entry** to note any differences between the actual **Seminar Ledger Entry** table fields and the required set of fields.
2. Correct the issues that were noted in the previous step and save the table.

Detailed Steps

1. Review all the fields in the table 123456732, **Seminar Ledger Entry** to note any differences between the actual **Seminar Ledger Entry** table fields and the required set of fields.
 - a. Design the table 123456732, **Seminar Ledger Entry**.
 - b. Compare the fields in the table with the following list.

Field No.	Field Name	Data Type	Length
1	Entry No.	Integer	
2	Seminar No.	Code	20
3	Posting Date	Date	
4	Document Date	Date	
5	Entry Type	Option	
6	Document No.	Code	20
7	Description	Text	50
8	Bill-to Customer No.	Code	20

Field No.	Field Name	Data Type	Length
9	Charge Type	Option	
10	Type	Option	
11	Quantity	Decimal	
12	Unit Price	Decimal	
13	Total Price	Decimal	
14	Participant Contact No.	Code	20
15	Participant Name	Text	50
16	Chargeable	Boolean	
17	Room Resource No.	Code	20
18	Instructor Resource No.	Code	20
19	Starting Date	Date	
20	Seminar Registration No.	Code	20
23	Res. Ledger Entry No.	Integer	
24	Source Type	Option	
25	Source No.	Code	20
26	Journal Batch Name	Code	10
27	Source Code	Code	10
28	Reason Code	Code	10
29	No. Series	Code	10
30	User ID	Code	20



Note: Notice that the actual set of fields does not match this table. Apparently, Isaac has only created the **Seminar Ledger Entry** table by copying the **Seminar Journal Line** table. You now must correct these issues.

- c. Note the following differences.

Issue	Remarks
Entry No.	This field does not exist. Instead, there are the Journal Template Name and Line No. fields.
No. Series	This field does not exist. Instead, there is the Posting No. Series field.
User ID	This field does not exist.

Issue	Remarks
Primary key	Primary key must only include the Entry No. field.

2. Correct the issues that were noted in the previous step and save the table.
 - a. From the list of fields, delete the **Journal Template Name** and **Line No.** fields.
 - b. Create the **Entry No.** field as the first field in the table.
 - c. Create the **User ID** field as the last field in the table.
 - d. Compile, save, and close the table.
 - e. Design the **Seminar Ledger Entry Table** again.

 **Note:** When you add a new field to a table, you must close the table and then reopen it before you can reference the field in the code.

- f. Rename the **Posting No. Series** field to **No. Series**.
- g. Set the Caption properties for any fields that you created or changed to match the field's Name.
- h. Set the TableRelation property of the **User ID** field to relate to the **User Name** field of the table **User**.

 **Note:** If you are unsure how to set the TableRelation property directly, click the **AssistEdit** button for **TableRelation**, and establish the relation in the **Table Relation** window.

- i. Set the ValidateTableRelation and TestTableRelation properties of the **User ID** field to **No**.

 **Note:** This guarantees that if the user is deleted from the database later, no table relation tests fail.

- j. Set the primary key to **Entry No.**
- k. In the OnValidate trigger for the **User ID** field, enter the code that calls the **LookupUserID** function of the User Management codeunit.

UserMgt.LookupUserID("User ID");

 **Note:** Make sure that you define the UserMgt local variable for the User Management codeunit.

- I. Renumber the **Field No.** for all the fields incrementally from 1 to 28, starting with the **Entry No.** field and ending with the **User ID** field.
 - m. Compile, save, and then close the table.

Task 2: Review the Seminar Register Table

High Level Steps

1. Review all the fields in the table 123456733, **Seminar Register** to note any differences between the actual **Seminar Register** table fields and the required set of fields.
2. Correct the issues that you noted in the previous step and save the table.

Detailed Steps

1. Review all the fields in the table 123456733, **Seminar Register** to note any differences between the actual **Seminar Register** table fields and the required set of fields.
 - a. Design the table 123456733, **Seminar Register**.
 - b. Compare the fields in the table with the following list:

Field No.	Field Name	Data Type	Length
1	No.	Integer	
2	From Entry No.	Integer	
3	To Entry No.	Integer	
4	Creation Date	Date	
5	Source Code	Code	10
6	User ID	Code	20
7	Journal Batch Name	Code	10

- c. Note the following issues:

Issue	Remarks
No.	The field does not exist. Instead, the field Entry No. is there. By convention, Register tables always have the primary key field named No. .
Journal Template Name	This field is not necessary in the register. The source of the transaction is kept in the Source Code field.

2. Correct the issues that you noted in the previous step and save the table.
 - a. Change the Name and the Caption properties for the **Entry No.** field to "No."
 - b. Delete the **Journal Template Name** field.
 - c. Renumber the **Field No.** for all the fields incrementally from 1 to 7, starting with the **No.** field and ending with the **Journal Batch Name** field.
 - d. Compile, save, and then close the table.

Exercise 4: Customize the Source Code Setup Table and Page

Exercise Scenario

In Microsoft Dynamics NAV 2013 every transaction must leave a clear and obvious audit trail. The core feature for auditing transactions in Microsoft Dynamics NAV 2013 is the source codes feature. Source codes simplify locating transactions that originated from a specific application function, such as a journal or a batch job.

When customizing Microsoft Dynamics NAV 2013 to include new posting routines or batch jobs that result in posted entries, you must extend the **Source Code Setup** table and page by using a field that identifies any new transaction type that you are introducing. Then in your posting routines, you must make sure that you use that field to identify the transactions that originated from that new feature. Therefore, you establish an audit trail that is consistent with other features of Microsoft Dynamics NAV 2013.

 **Note:** Transaction source codes can only be defined in the **Journal Template** tables for journals, and in the **Source Code Setup** table for documents and batch jobs. Every transaction carries the **Source Code** field, and this field is always taken from either the appropriate **Journal Template** table, or from the **Source Code Setup** table. Users can never change the **Source Code** field in any of the transactions. The **Source Code** field is only shown in the **Ledger Entries** and **Register** pages, never in journals or documents.

After reviewing all posting tables, you now must make sure that the **Source Code Setup** table includes a source code configuration field for the seminar posting routine. Then, you must add the same field to the **Source Code Setup** page.

Task 1: Customize the Source Code Setup Table

High Level Steps

1. Add the **Seminar** field to the table 242, **Source Code Setup**.

Detailed Steps

1. Add the **Seminar** field to the table 242, **Source Code Setup**.
 - a. Design the table 242, **Source Code Setup**.
 - b. Add the following field.

No.	Field Name	Type	Length	Remarks
123456700	Seminar	Code	10	Set the table relation to the Source Code table.

- c. Set the Caption property for the field to "Seminar".
- d. Compile, save, and then close the table.

Task 2: Customize the Source Code Setup Page

High Level Steps

1. Add the **Seminar Management** FastTab and the **Seminar** field to the page 279, **Source Code Setup**.

Detailed Steps

1. Add the **Seminar Management** FastTab and the **Seminar** field to the page 279, **Source Code Setup**.
 - a. Design the page 279, **Source Code Setup**.
 - b. Add the Seminar Management group control as the last group control, just before the FactBoxArea container. Make sure that you indent the group control at the same level as the other group controls.
 - c. Add the **Seminar** field to the Seminar Management group. Make sure that it is indented under the group.
 - d. Compile, save, and then close the page.

Lab 4.2: Creating Codeunits and Pages for Seminar Journal Posting

Scenario

After you have reviewed all journal posting tables, and have completed the necessary customizations and corrections, you are now ready to develop the codeunits for the seminar journal posting routine. CRONUS International Ltd. only requires document posting functionality, and did not request journal posting functionality. Their users would find it unnatural to post any seminar registrations through a journal. Therefore, you do not have to develop all journal posting codeunits that you would normally provide if the customer required a full journal user interface.

 **Note:** Even though the customer never requested it, you must provide the journal posting functionality to enable documents to post ledger entries in a way that is consistent with both best practices and the application standards that are found in other functional areas.

The journal posting codeunits that you must develop are as follows.

Codeunit	Remarks
Seminar Jnl.-Check Line	This codeunit checks each line before posting. You must always provide this codeunit.
Seminar Jnl.-Post Line	This codeunit posts each line. You must always provide this codeunit.
Seminar Reg.-Show Ledger	This codeunit shows the ledger entries that result from a single journal posting.

No other journal posting codeunits are required because there is no user interface. Also, you do not have to provide the Seminar Jnl.-Post Batch codeunit, because document posting routines only call the Post Line codeunit.

Exercise 1: Create the Seminar Jnl.-Check Line Codeunit

Exercise Scenario

Create the Seminar Jnl.-Check Line codeunit, which is called from the Seminar Jnl.-Post Line codeunit. This codeunit must perform standard posting checks, such as allowed posting dates, presence of all required fields, and so on.



Note: If you are unsure how to structure the code in this codeunit, look at the codeunit 211, Res. Jnl-Check Line.

Task 1: Create the Codeunit

High Level Steps

1. Create the codeunit 123456731, Seminar Jnl.-Check Line, and set the properties to specify the **Seminar Journal Line** as the source table for this codeunit.

Detailed Steps

1. Create the codeunit 123456731, Seminar Jnl.-Check Line, and set the properties to specify the **Seminar Journal Line** as the source table for this codeunit.
 - a. In Object Designer, show the codeunits, and then click **New**.
 - b. Save the new codeunit as 123456731, Seminar Jnl.-Check Line.
 - c. Set the TableNo property for the codeunit to "Seminar Journal Line".

Task 2: Declare the Variables and Text Constants

High Level Steps

1. Declare the global variables for the **G/L Setup** and **User Setup** tables, and two date variables to keep track of allowed posting period starting and ending dates.
2. Declare the text constants that display errors if entries are posted on closing dates, or outside the allowed posting periods.

Detailed Steps

1. Declare the global variables for the **G/L Setup** and **User Setup** tables, and two date variables to keep track of allowed posting period starting and ending dates.
 - a. Declare the following global variables.

Name	DataType	Subtype
GLSetup	Record	General Ledger Setup
UserSetup	Record	User Setup

Name	DataType	Subtype
AllowPostingFrom	Date	
AllowPostingTo	Date	

2. Declare the text constants that display errors if entries are posted on closing dates, or outside the allowed posting periods.
 - a. Declare the following global text constants.

Name	ConstValue
Text000	cannot be a closing date.
Text001	is not within your range of allowed posting dates.

Task 3: Create the RunCheck Function

High Level Steps

1. Create the **RunCheck** function that receives a Seminar Journal Line record by reference. Make sure that any code that you add to this function later is enclosed in a WITH block for the record parameter that is passed to the function.
2. From the OnRun trigger, call the **RunCheck** function.

Detailed Steps

1. Create the **RunCheck** function that receives a Seminar Journal Line record by reference. Make sure that any code that you add to this function later is enclosed in a WITH block for the record parameter that is passed to the function.
 - a. In the **C/AL Globals** window, create a new function, and name it **RunCheck**.
 - b. Define the following parameters for this function.

Var	Name	DataType	Subtype
Yes	SemJnlLine	Record	Seminar Journal Line

- c. In the **RunCheck** function trigger, enter the following code.

```
WITH SemJnlLine DO BEGIN
```

```
END;
```

2. From the OnRun trigger, call the **RunCheck** function.
 - a. Enter the following code into the OnRun trigger.

```
RunCheck(Rec);
```

Task 4: Add Code to the RunCheck Function

High Level Steps

1. Enter code in the RunCheck function trigger to test whether the **Seminar Journal Line** is empty by using the **EmptyLine** function. If the line is empty, the function exits.
2. Make sure that the **Posting Date**, **Job No.**, and **Seminar No.** fields are not empty.
3. Depending on the value of the **Charge Type** field, make sure that the **Instructor Code**, **Seminar Room Code**, and **Participant Contact No.** fields are not empty.
4. If the line is Chargeable, make sure that the **Bill-to Customer No.** field is not blank.
5. Show an error if the Posting Date is a closing date.
6. Make sure that the **Posting Date** field is between the **Allow Posting From** field and the **Allow Posting To** field values in the **User Setup** table. If these fields are not defined there, then make sure that the **Posting Date** field is between the **Allow Posting From** field and **Allow Posting To** field values in the **G/L Setup** table.
7. Show an error if the **Document Date** field is a closing date, and then save the codeunit.

Detailed Steps

1. Enter code in the RunCheck function trigger to test whether the **Seminar Journal Line** is empty by using the **EmptyLine** function. If the line is empty, the function exits.
 - a. In the WITH block of the **RunCheck** function trigger, enter the following code.

```
IF EmptyLine THEN
```

```
    EXIT;
```



Note: For all other steps in this task, keep adding the code to the RunCheck function trigger, just before the END of the WITH block.

Module 4: Posting

2. Make sure that the **Posting Date**, **Job No.**, and **Seminar No.** fields are not empty.
 - a. Enter the following code.

```
TESTFIELD("Posting Date");  
  
TESTFIELD("Instructor Resource No.");  
  
TESTFIELD("Seminar No.');
```

3. Depending on the value of the **Charge Type** field, make sure that the **Instructor Code**, **Seminar Room Code**, and **Participant Contact No.** fields are not empty.
 - a. Enter the following code.

```
CASE "Charge Type" OF  
  
"Charge Type":::Instructor:  
  
TESTFIELD("Instructor Resource No.");  
  
"Charge Type":::Room:  
  
TESTFIELD("Room Resource No.");  
  
"Charge Type":::Participant:  
  
TESTFIELD("Participant Contact No.");  
  
END;
```

4. If the line is Chargeable, make sure that the **Bill-to Customer No.** field is not blank.
 - a. Enter the following code.

```
IF Chargeable THEN  
  
TESTFIELD("Bill-to Customer No.');
```

5. Show an error if the Posting Date is a closing date.

- a. Enter the following code.

```
IF "Posting Date" = CLOSINGDATE("Posting Date") THEN  
FIELDERROR("Posting Date",Text000);
```

6. Make sure that the **Posting Date** field is between the **Allow Posting From** field and the **Allow Posting To** field values in the **User Setup** table. If these fields are not defined there, then make sure that the **Posting Date** field is between the **Allow Posting From** field and **Allow Posting To** field values in the **G/L Setup** table.

- a. Enter the following code.

```
IF (AllowPostingFrom = 0D) AND (AllowPostingTo = 0D) THEN BEGIN  
  
IF USERID <> "" THEN  
  
IF UserSetup.GET(USERID) THEN BEGIN  
  
AllowPostingFrom := UserSetup."Allow Posting From";  
  
AllowPostingTo := UserSetup."Allow Posting To";  
  
END;  
  
IF (AllowPostingFrom = 0D) AND (AllowPostingTo = 0D) THEN BEGIN  
  
GLSetup.GET;  
  
AllowPostingFrom := GLSetup."Allow Posting From";  
  
AllowPostingTo := GLSetup."Allow Posting To";  
  
END;  
  
IF AllowPostingTo = 0D THEN  
  
AllowPostingTo := 12319999D;  
  
END;  
  
IF ("Posting Date" < AllowPostingFrom) OR ("Posting Date" > AllowPostingTo)  
THEN  
  
FIELDERROR("Posting Date",Text001);
```

 **Note:** This check is a standard check in all journal posting codeunits. You can take a look at the codeunit 211, Res. Jnl.-Check Line to see how it handles this particular check.

7. Show an error if the **Document Date** field is a closing date, and then save the codeunit.
 - a. Enter the following code.

```
IF ("Document Date" <> 0D) THEN  
  
    IF ("Document Date" = CLOSINGDATE("Document Date")) THEN  
  
        FIELDERROR("Document Date",Text000);
```

- b. Compile, save, and then close the codeunit.

Exercise 2: Create the Seminar Jnl.-Post Line Codeunit

Exercise Scenario

Now you must create the Seminar Jnl.-Post Line codeunit. This executes the core work of the seminar journal posting routine, and creates the **Seminar Ledger Entry** records for the journal posting transaction. This codeunit must handle the following:

- Run the Seminar Jnl.-Check Line codeunit.
- Increase the Entry No. of the ledger entries it creates by one.
- Make sure that one register record is created and maintained throughout the journal posting process to reflect the first and last entry number.
- Insert the ledger entry, and populate it from the fields of the **Seminar Journal Line** table.

 **Note:** You may want to view the codeunit 212, Res. Jnl.-Post Line to understand the structure and the logic of that codeunit, and then apply the same patterns and concepts in the Seminar Jnl.-Post Line codeunit.

Task 1: Create the Codeunit

High Level Steps

1. Create the codeunit 123456732, Seminar Jnl.-Post Line, and set the properties to specify the **Seminar Journal Line** as the source table for this codeunit.

Detailed Steps

1. Create the codeunit 123456732, Seminar Jnl.-Post Line, and set the properties to specify the **Seminar Journal Line** as the source table for this codeunit.
 - a. In Object Designer, show the codeunits, and then click **New**.
 - b. Save the new codeunit as 123456732, Seminar Jnl.-Post Line.
 - c. Set the TableNo property for the codeunit to "Seminar Journal Line".

Task 2: Declare the Variables

High Level Steps

1. Declare the global variables for the **Seminar Journal Line**, **Seminar Ledger Entry**, and **Seminar Register** tables. Then create a global variable for the Seminar Jnl.-Check Line codeunit, and an integer variable to keep track of the next available Entry No.

Detailed Steps

1. Declare the global variables for the **Seminar Journal Line**, **Seminar Ledger Entry**, and **Seminar Register** tables. Then create a global variable for the Seminar Jnl.-Check Line codeunit, and an integer variable to keep track of the next available Entry No.
 - a. Declare the following global variables.

Name	DataType	Subtype
SeminarJnlLine	Record	Seminar Journal Line
SeminarLedgerEntry	Record	Seminar Ledger Entry
SeminarRegister	Record	Seminar Register
SeminarJnlCheckLine	Codeunit	Seminar Jnl.-Check Line
NextEntryNo	Integer	

Task 3: Create the Functions

High Level Steps

1. Create the **RunWithCheck** and **Code** functions. The **RunWithCheck** function receives a Seminar Journal Line as a parameter by reference.
2. 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.
3. Enter code in the **RunWithCheck** function trigger so that the function copies the *SeminarJnlLine* from the *SeminarJnlLine2* record, runs the **Code** function, and then restores the *SeminarJnlLine2* record back from the *SeminarJnlLine* record.

Detailed Steps

1. Create the **RunWithCheck** and **Code** functions. The **RunWithCheck** function receives a Seminar Journal Line as a parameter by reference.
 - a. In the **C/AL Globals** window, create the following functions: **RunWithCheck** and **Code**.
 - b. For the **RunWithCheck** function, declare the following parameters.

Var	Name	DataType	Subtype
Yes	SeminarJnlLine2	Record	Seminar Journal Line

2. 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.
 - a. In the OnRun trigger, enter the following code.

```
RunWithCheck(Rec);
```

3. Enter code in the **RunWithCheck** function trigger so that the function copies the *SeminarJnlLine* from the *SeminarJnlLine2* record, runs the **Code** function, and then restores the *SeminarJnlLine2* record back from the *SeminarJnlLine* record.
 - a. In the **RunWithCheck** function trigger, enter the following code.

```
SeminarJnlLine.COPY(SeminarJnlLine2);
```

```
Code;
```

```
SeminarJnlLine2 := SeminarJnlLine;
```

 **Note:** The SeminarJnlLine global variable is the main record variable that must be available to all functions in the Seminar Jnl.-Post Line codeunit. By copying it from the by-reference parameter, the whole codeunit has access to the same **Seminar Journal Line** record.

When the **Code** function is finished, the by-reference parameter is set to the SeminarJnlLine global variable to pass its latest state to the caller. This is a necessary convention because the OnRun trigger is never called directly. It provides backward compatibility only.

For this codeunit, it is present for convention reasons. The Rec variable is never initialized, and cannot be used as the global **Seminar Journal Line** record variable available throughout the codeunit. Therefore, the SeminarJnlLine is declared, and the pattern that you see in the **RunWithCheck** function guarantees the same behavior that you would usually achieve if you called the OnRun trigger directly, and used the Rec variable instead.

Task 4: Add Code to the Code Function

High Level Steps

1. In the **Code** function, enter the WITH code block for the *SeminarJnlLine* record variable.
2. Check whether the *SeminarJnlLine* is empty by using the **EmptyLine** function. If it is empty, the function exits.
3. Runs the **RunCheck** function of the *SeminarJnlCheckLine* codeunit.
4. If the *NextEntryNo* is 0, lock the *SeminarLedgEntry* record, then set the *NextEntryNo* to the **Entry No.** of the last record in the *SeminarLedgEntry* table, if it can be found. Then, increase the *NextEntryNo* by one.
5. If the **Document Date** is empty, the set the **Document Date** to the **Posting Date**.
6. Create or update the **SeminarRegister** record, depending on whether the register record was previously created for this posting. When you create the register record, initialize all fields according to their meaning.
7. Create a new **SeminarLedgerEntry** record, populate the fields from the **SeminarJnlLine** record, set the **Entry No.** field to the *NextEntryNo* variable, insert the new record, and then increment the *NextEntryNo* variable by one. Finally, save the codeunit.

Detailed Steps

1. In the **Code** function, enter the WITH code block for the *SeminarJnlLine* record variable.
 - a. In the **Code** function trigger, enter the following code.

```
WITH SeminarJnlLine DO BEGIN  
  
END;
```

2. Check whether the *SeminarJnlLine* is empty by using the **EmptyLine** function. If it is empty, the function exits.
 - a. In the WITH block of the **Code** function trigger, enter the following code.

```
IF EmptyLine THEN  
  
EXIT;
```

 **Note:** For all successive steps in this task, keep adding the code to the **Code** function trigger, just before the END of the WITH block.

3. Runs the **RunCheck** function of the *SeminarJnlCheckLine* codeunit.
 - a. Enter the following code.

```
SeminarJnlCheckLine.RunCheck(SeminarJnlLine);
```

4. If the NextEntryNo is 0, lock the SeminarLedgEntry record, then set the NextEntryNo to the **Entry No.** of the last record in the SeminarLedgEntry table, if it can be found. Then, increase the NextEntryNo by one.
 - a. Enter the following code.

```
IF NextEntryNo = 0 THEN BEGIN  
  
SeminarLedgerEntry.LOCKTABLE;  
  
IF SeminarLedgerEntry.FINDLAST THEN  
  
NextEntryNo := SeminarLedgerEntry."Entry No.";  
  
NextEntryNo := NextEntryNo + 1;  
  
END;
```

 **Note:** If the NextEntryNo variable is equal to zero, it means that the **Code** function was called for the first time during the posting process. In this case, to maintain the transaction integrity, the **Seminar Ledger Entry** table must be locked.

Then, the next entry number must be calculated. If there are any other entries in the **Seminar Ledger Entry** table, then the NextEntryNo is set to the last **Entry No.** used, and if there are no other entries, then it remains at zero.

Finally, the NextEntryNo is increased by one, to make sure that it either starts at one for the very first ledger entry or at the next available value if there are other ledger entries already in the table.

-
5. If the **Document Date** is empty, then set the **Document Date** to the **Posting Date**.

- a. Enter the following code.

```
IF "Document Date" = 0D THEN  
    "Document Date" := "Posting Date";
```

6. Create or update the **SeminarRegister** record, depending on whether the register record was previously created for this posting. When you create the register record, initialize all fields according to their meaning.

 **Note:** If the **No.** field of the **SeminarRegister** record is zero, then the register record has not yet been created.

- a. Enter the following code.

```
IF SeminarRegister."No." = 0 THEN BEGIN  
    SeminarRegister.LOCKTABLE;  
  
    IF (NOT SeminarRegister.FINDLAST) OR (SeminarRegister."To Entry No." <> 0)  
    THEN BEGIN  
  
        SeminarRegister.INIT;  
  
        SeminarRegister."No." := SeminarRegister."No." + 1;  
  
        SeminarRegister."From Entry No." := NextEntryNo;  
  
        SeminarRegister."To Entry No." := NextEntryNo;  
  
        SeminarRegister."Creation Date" := TODAY;
```

```
SeminarRegister."Source Code" := "Source Code";  
  
SeminarRegister."Journal Batch Name" := "Journal Batch Name";  
  
SeminarRegister."User ID" := USERID;  
  
SeminarRegister.INSERT;  
  
END;  
  
END;  
  
SeminarRegister."To Entry No." := NextEntryNo;  
  
SeminarRegister.MODIFY;
```



Note: If the register has not yet been initialized, then the table is first locked to maintain the transaction integrity.

If there are no register records in the table at all, or if this is the first time in this transaction that the **Code** function is called (the **To Entry No.** field is zero only for the first call), then a register record is initialized and populated with relevant data. **From Entry No.** is set to the NextEntryNo., which at this point is the first entry for the transaction.

Finally, for every call to the **Code** function, the **To Entry No.** field is set to the **NextEntryNo.** field. This increases by one every time that the function is called. This ensures that at the end of the transaction, the **From Entry No.** field of the register record is set to the **Entry No.** field of the first ledger entry. Also, the **To Entry No.** field is set to the **Entry No.** field of the last ledger entry in the transaction.

7. Create a new **SeminarLedgerEntry** record, populate the fields from the **SeminarJnlLine** record, set the **Entry No.** field to the *NextEntryNo* variable, insert the new record, and then increment the *NextEntryNo* variable by one. Finally, save the codeunit.
 - a. Enter the following code.

```
SeminarLedgerEntry.INIT;  
  
SeminarLedgerEntry."Seminar No." := "Seminar No. ";  
  
SeminarLedgerEntry."Posting Date" := "Posting Date";  
  
SeminarLedgerEntry."Document Date" := "Document Date";  
  
SeminarLedgerEntry."Entry Type" := "Entry Type";
```

```
SeminarLedgerEntry."Document No." := "Document No.;"  
SeminarLedgerEntry.Description := Description;  
SeminarLedgerEntry."Bill-to Customer No." := "Bill-to Customer No.;"  
SeminarLedgerEntry."Charge Type" := "Charge Type";  
SeminarLedgerEntry.Type := Type;  
SeminarLedgerEntry.Quantity := Quantity;  
SeminarLedgerEntry."Unit Price" := "Unit Price";  
SeminarLedgerEntry."Total Price" := "Total Price";  
SeminarLedgerEntry."Participant Contact No." := "Participant Contact No.;"  
SeminarLedgerEntry."Participant Name" := "Participant Name";  
SeminarLedgerEntry.Chargeable := Chargeable;  
SeminarLedgerEntry."Room Resource No." := "Room Resource No.;"  
SeminarLedgerEntry."Instructor Resource No." := "Instructor Resource No.;"  
SeminarLedgerEntry."Starting Date" := "Starting Date";  
SeminarLedgerEntry."Seminar Registration No." := "Seminar Registration No.;"  
SeminarLedgerEntry."Res. Ledger Entry No." := "Res. Ledger Entry No.;"  
SeminarLedgerEntry."Source Type" := "Source Type";  
SeminarLedgerEntry."Source No." := "Source No.;"  
SeminarLedgerEntry."Journal Batch Name" := "Journal Batch Name";  
SeminarLedgerEntry."Source Code" := "Source Code";  
SeminarLedgerEntry."Reason Code" := "Reason Code";
```

```
SeminarLedgerEntry."No. Series" := "Posting No. Series";  
SeminarLedgerEntry."User ID" := USERID;  
SeminarLedgerEntry."Entry No." := NextEntryNo;  
SeminarLedgerEntry.INSERT;  
NextEntryNo := NextEntryNo + 1;
```

- b. Compile, save, and then close the codeunit.

Exercise 3: Create the Seminar Ledger Entries Page

Exercise Scenario

Now you must create the page to show the ledger entries. The page must be of list type, and must be noneditable.

Task 1: Create the Page

High Level Steps

1. Create a noneditable list page for the **Seminar Ledger Entry** table by using a wizard and adding the fields to the page.
2. Save the page as **123456721, Seminar Ledger Entries**, and close it.

Detailed Steps

1. Create a noneditable list page for the **Seminar Ledger Entry** table by using a wizard and adding the fields to the page.
 - a. In Object Designer, show the pages, and then click **New**.
 - b. Choose the **Seminar Ledger Entry** table, and then start the List Page Wizard.
 - c. Add the following fields to the page:
 - Posting Date
 - Document No.
 - Document Date
 - Entry Type
 - Seminar No.
 - Description
 - Bill-to Customer No.
 - Charge Type
 - Type
 - Quantity
 - Unit Price

- Total Price
 - Chargeable
 - Participant Contact No.
 - Participant Name
 - Instructor Resource No.
 - Room Resource No.
 - Starting Date
 - Seminar Registration No.
 - Entry No.
- d. Add the **Record Links** and **Notes** system FactBoxes.
 - e. Finish the wizard.
 - f. Set the Visible property for the Document Date field to *FALSE*.
 - g. Set the Editable property for the page to **No**.
 - h. Set the Caption property for the page to "Seminar Ledger Entries".
2. Save the page as **123456721, Seminar Ledger Entries**, and close it.
 - a. Save the page as **123456721, Seminar Ledger Entries**.
 - b. Close the **Page Designer** window.

Exercise 4: Create the Seminar Reg.-Show Ledger Codeunit

Exercise Scenario

The last codeunit you must create is the Seminar Reg.-Show Ledger codeunit. The sole purpose of this codeunit is to show the records from the **Seminar Ledger Entry** table that are filtered to only a single transaction. The transaction is defined by the **Seminar Register** record that this function receives through the *Rec* parameter of the OnRun trigger.

Task 1: Create the Codeunit

High Level Steps

1. Create the codeunit **123456734, Seminar Reg.-Show Ledger**, and set the properties to specify the **Seminar Register** as the source table for this codeunit.
2. Declare a global variable for the **Seminar Ledger Entry** table.

Detailed Steps

1. Create the codeunit **123456734, Seminar Reg.-Show Ledger**, and set the properties to specify the **Seminar Register** as the source table for this codeunit.
 - a. In Object Designer, show the codeunits, and then click **New**.

- b. Save the new codeunit as **123456734, Seminar Reg.-Show Ledger.**
 - c. Set the TableNo property for the codeunit to "Seminar Register".
2. Declare a global variable for the **Seminar Ledger Entry** table.
- a. Create the following global variable.

Name	DataType	Subtype
SeminarLedgerEntry	Record	Seminar Ledger Entry

Task 2: Add Code to the OnRun Trigger

High Level Steps

1. Enter code in the appropriate trigger so that when the program runs the codeunit, the codeunit runs the **Seminar Ledger Entries** page. This shows only those entries between the **From Entry No.** field and the **To Entry No.** field on the **Seminar Register**.

Detailed Steps

1. Enter code in the appropriate trigger so that when the program runs the codeunit, the codeunit runs the **Seminar Ledger Entries** page. This shows only those entries between the **From Entry No.** field and the **To Entry No.** field on the **Seminar Register**.
 - a. In the OnRun trigger, enter the following code.

```
SeminarLedgerEntry.SETRANGE("Entry No.", "From Entry No.", "To Entry No.");  
PAGE.RUN(PAGE::"Seminar Ledger Entries", SeminarLedgerEntry);
```

- b. Compile, save, and then close the codeunit.

Exercise 5: Create the Seminar Registers Page

Task 1: Create the Page

High Level Steps

1. Create a noneditable list page for the **Seminar Register** table by using a wizard and add the fields to the page.
2. Add an action to the RelatedInformation action container to run the Seminar Reg.-Show Ledger codeunit.
3. Save the page as 123456722, **Seminar Registers**, and close it.

Detailed Steps

1. Create a noneditable list page for the **Seminar Register** table by using a wizard and add the fields to the page.
 - a. In Object Designer, show the pages, and then click **New**.
 - b. Choose the **Seminar Register** table, and then start the List Page Wizard.
 - c. Add the following fields to the page:
 - No.
 - Creation Date
 - User ID
 - Source Code
 - Journal Batch Name
 - From Entry No.
 - To Entry No.
 - d. Add the **Record Links** and **Notes** system FactBoxes.
 - e. Finish the wizard.
 - f. Set the Editable property for the page to **No**.
 - g. Set the Caption property for the page to "Seminar Registers".
2. Add an action to the RelatedInformation action container to run the Seminar Reg.-Show Ledger codeunit.
 - a. Open the **Page – Action Designer** window.
 - b. Define the RelatedInformation action container.
 - c. Define the Register action group.
 - d. Add the **Seminar Ledger** action, and set RunObject property to run the Seminar Reg.-Show Ledger codeunit.
 - e. Assign the WarrantyLedger image to the action and promote it as a large action to the Process category.
3. Save the page as 123456722, **Seminar Registers**, and close it.
 - a. Save the page as 123456722, **Seminar Registers**.
 - b. Close the **Page Designer** window.

Lab 4.3: Creating the Tables and Pages for Posted Registration Information

Scenario

CRONUS International Ltd. wants to post the Seminar Registration documents after the seminars are completed. You must create the tables and pages that will store and show the posted seminar registration information.

When a user posts a document in Microsoft Dynamics NAV 2013, the structure of the posted information must match the structure of the original information in all relevant aspects. This means that the data model for posted documents must always match the data model for open documents. It is both a convention and a requirement in Microsoft Dynamics NAV 2013 that posted document table fields match the document table fields. If a field is relevant for the posted document, then it must have the same **Field No.** as the same field in the document table. This makes it easier for users to match the posted document information to the information they originally entered into the system. It also simplifies development because you can use the **TRANSFERFIELDS** function to copy the field values between open and posted document tables.

 **Note:** The **TRANSFERFIELDS** function copies all fields that have the same **Field No.** from the source table to the destination table. The fields must have the same data type for the copying to succeed (text and code are convertible, other types are not). There must be room for the actual length of the contents of the field to be copied in the field to which it is to be copied. If any one of these conditions is not fulfilled, a run-time error occurs.

Therefore, the simplest way to create the posted document tables is by saving the original tables under a different ID and Name. Then make any necessary changes, such as removing unnecessary fields, or appending those fields that are not relevant for the document, but are relevant for the posted document tables.

Exercise 1: Create the Posted Registration Tables

Exercise Scenario

You start by creating the tables for posted registration information. The best approach is to design each document table, and save it under a new ID and name. Then, you must remove all the code from the tables, and make any necessary corrections to table and field properties to meet the requirements and best practices for posted document tables.

Task 1: Create the Posted Seminar Reg. Header Table

High Level Steps

1. Create the **Posted Seminar Reg. Header** table by saving the **Seminar Registration Header** table under ID 123456718.
2. Remove all the code from the table.
3. Delete and rename fields to match the standards for posted document header tables, and add the **User ID** and **Source Code** fields.
4. Add code to the OnLookup trigger of the **User ID** field to run the **LookupUser** function of the User Management codeunit.
5. Correct the calculation formula for the **Comment** field.
6. Set the Caption property for the table to match its Name, and then save and close the table.

Detailed Steps

1. Create the **Posted Seminar Reg. Header** table by saving the **Seminar Registration Header** table under ID 123456718.
 - a. Design the table 123456710, **Seminar Registration Header**.
 - b. Click **File > Save As**. Save the table as 123456718, **Posted Seminar Reg. Header**.
2. Remove all the code from the table.
 - a. In the **C/AL Globals** window, delete all variables, text constants, and functions.
 - b. In the **C/AL Editor** window, delete all C/AL code. To do this, click the header bar for the Documentation trigger (or any other trigger), then press CTRL+A, then press DEL.
 - c. Confirm the deletion by clicking **Yes**.
3. Delete and rename fields to match the standards for posted document header tables, and add the **User ID** and **Source Code** fields.
 - a. Delete the **Posting No.** field.
 - b. Rename the **Posting No. Series** field to **Registration No. Series**, and modify its Caption accordingly.
 - c. Add the following fields.

Field No.	Field Name	Data Type	Length
29	User ID	Code	20
30	Source Code	Code	10

- d. Set the TableRelation property for the **User ID** field to the **User Name** field of the **User** table, and set its ValidateTableRelation and TestTableRelationship properties to **No**.
 - e. Set the **TableRelation** property for the **Source Code** field to the **Source Code** table.
4. Add code to the OnLookup trigger of the **User ID** field to run the **LookupUser** function of the User Management codeunit.
- a. In the User ID – OnLookup trigger, view the C/AL Locals.
 - b. Define the following local variable.

Name	DataType	Subtype
UserMgt	Codeunit	User Management

- c. Enter the following code.

```
UserMgt.LookupUserID("User ID");
```

5. Correct the calculation formula for the **Comment** field.
 - a. In the CalcFormula property for the **Comment** field, correct the table filter so that the **Document Type** field is filtered on Posted Seminar Registration value instead of Seminar Registration value.

 **Note:** Click the **AssistEdit** button in the **CalcFormula** to access the **Calculation Formula** window, and then click the **AssistEdit** button in the **Table Filter** field to access the **Table Filter** window.

6. Set the Caption property for the table to match its Name, and then save and close the table.
 - a. Set the Caption property for the table to "Posted Seminar Reg. Header".
 - b. Compile, save, and then close the table.

Task 2: Create the Posted Seminar Reg. Line Table

High Level Steps

1. Create the **Posted Seminar Reg. Line** table by saving the **Seminar Registration Line** table under ID 123456719.
2. Correct the table relation for the **Document No.** field.
3. Remove all code from the table.
4. Set the Caption property for the table to match its Name, and then save and close the table.

Detailed Steps

1. Create the **Posted Seminar Reg. Line** table by saving the **Seminar Registration Line** table under ID 123456719.
 - a. Design the table 123456711, **Seminar Registration Line**.
 - b. Click **File > Save As**. Save the table as 123456719, **Posted Seminar Reg. Line**.
2. Correct the table relation for the **Document No.** field.
 - a. For the **Document No.** field, set the TableRelation property to the **Posted Seminar Reg. Header** table.
3. Remove all code from the table.
 - a. In the **C/AL Globals** window, delete all variables, text constants, and functions.
 - b. In the **C/AL Editor** window, delete all C/AL code. To do this, click the header bar for the Documentation trigger (or any other trigger), then press CTRL+A, then press DEL.
 - c. Confirm the deletion by clicking **Yes**.
4. Set the Caption property for the table to match its Name, and then save and close the table.
 - a. Set the Caption property for the table to "Posted Seminar Reg. Line".
 - b. Compile, save, and then close the table.

Task 3: Create the Posted Seminar Charge Table

High Level Steps

1. Create the **Posted Seminar Charge** table by saving the **Seminar Charge** table under ID 123456721.
2. Correct the table relation for the **Document No.** field.
3. Remove all code from the table.
4. Set the Caption property for the table to match its Name, and then save and close the table.

Detailed Steps

1. Create the **Posted Seminar Charge** table by saving the **Seminar Charge** table under ID 123456721.
 - a. Design the table 123456712, **Seminar Charge**.
 - b. Click **File > Save As**. Save the table as 123456721, **Posted Seminar Charge**.

2. Correct the table relation for the **Document No.** field.
 - a. For the **Document No.** field, set the TableRelation property to the **Posted Seminar Reg. Header** table.
3. Remove all code from the table.
 - a. In the **C/AL Globals** window, delete all variables, text constants, and functions.
 - b. In the **C/AL Editor** window, delete all C/AL code. To do this, click the header bar for the Documentation trigger (or any other trigger), then press CTRL+A, then press DEL.
 - c. Confirm the deletion by clicking **Yes**.
4. Set the Caption property for the table to match its Name, and then save and close the table.
 - a. Set the Caption property for the table to "Posted Seminar Charge".
 - b. Compile, save, and then close the table.

Exercise 2: Import the Posted Registration Pages

Exercise Scenario

After you have created the tables, continue to the most important functionality of the seminar posting feature: the document posting routine. In the meantime, you assign the task to Isaac to develop the following pages for accessing the posted document information: **Posted Seminar Registration**, **Posted Seminar Reg. List**, and **Posted Seminar Charges**.

Task 1: Import the Objects

High Level Steps

1. In Object Designer, import the Mod04\Labfiles\Lab 4.C - Starter - Posted Registration Pages.fob file.
2. Select and compile the imported objects.

Detailed Steps

1. In Object Designer, import the Mod04\Labfiles\Lab 4.C - Starter - Posted Registration Pages.fob file.
 - a. In Object Designer, click **File > Import**.
 - b. In the **Import Objects** dialog box, browse to Mod04\Labfiles\Lab 4.C - Starter - Posted Registration Pages.fob file.
 - c. Click **Open**.

2. Select and compile the imported objects.
 - a. View all objects in Object Designer.
 - b. Filter the view to only show the uncompiled objects.
 - c. Compile the objects.
 - d. Clear all filters in the Object Designer by pressing SHIFT+CTRL+F7.

Task 2: Review the Objects

High Level Steps

1. Review the imported objects.

Detailed Steps

1. Review the imported objects.
 - a. Design the page 123456734, **Posted Seminar Registration**.
 - b. Review the page contents in the **Page Preview** window.
 - c. Review the page properties.
 - d. Close the page 123456734.
 - e. Design the page 123456736, **Posted Seminar Reg. List**.
 - f. Review the page contents in the **Page Preview** window.
 - g. Review the page properties.
 - h. Close the page 123456736.
 - i. Design the page 123456739, **Posted Seminar Charges**.
 - j. Review the page contents in the **Page Preview** window.
 - k. Review the page properties.
 - l. Close the page 123456739.

Lab 4.4: Modifying Tables, Pages, and Codeunits for Resource Posting

Scenario

When you create new modules, such as Seminar Management, you frequently have to integrate those custom modules with existing features and functionality. Seminars integrate with Resource Management functionality. You use resources to represent instructors and rooms. For auditing and reporting, you want to attach the seminar information to all records in the **Resource Ledger Entry** table. This performs the following goals:

- You have a more robust trail record because you know which resource ledger entries are related to a seminar.
- You can easily and efficiently calculate totals for combinations of instructors, rooms, and seminars.
- You enable seamless user interface flow between Resource Management and Seminar Management functional areas, because all entries are related on the data model level.

You decide to add the following fields to the **Resource Ledger Entry** table.

Field	Remarks
Seminar No.	Lets you keep track of which instructor or room is connected with a seminar.
Seminar Registration No.	Link the posted seminar registration so that users can easily move to all instructor or room resource ledger entries from a posted seminar registration.

 **Note:** The **Seminar No.** field seems redundant, because it can be retrieved through the Seminar Registration No. field. However, if you omit the **Seminar No.** field, you cannot directly filter on resource ledger entries for specific seminars. This reduces the user experience, and adds processing demands when you might have to report or total instructor or room ledger entries by seminar. Therefore, while adding an additional field is not an elegant solution from the data normalization perspective, it is the most efficient solution from the user experience and data processing perspective.

To make sure that the **Seminar No.** and **Seminar Registration No.** fields are always posted into the **Resource Ledger Entry** table, you must also change the following existing objects.

.Type	ID	Name	Remarks
Table	203	Res. Ledger Entry	
Table	207	Res. Journal Line	To post any new fields to a Ledger Entry table, you must first add those fields to the matching Journal Line table.
Codeunit	212	Res. Jnl.-Post Line	To move the fields from a Journal Line table to the matching Ledger Entry table, you must add the appropriate code to the Post Line codeunit for the journal.
Page	202	Resource Ledger Entries	You typically want to show the new fields in the Ledger Entries page.

Exercise 1: Modify the Objects

Exercise Scenario

You start by adding and changing the necessary fields to the **Res. Ledger Entry** and **Res. Journal Line** tables, and then modify the **Resource Ledger Entries** page, and the Res. Jnl.-Post Line codeunit.

Task 1: Modify the Res. Ledger Entry Table***High Level Steps***

1. Add the **Seminar No.** and **Seminar Registration No.** fields to the table 203, **Res. Ledger Entry**.

Detailed Steps

1. Add the **Seminar No.** and **Seminar Registration No.** fields to the table 203, **Res. Ledger Entry**.
 - a. Design the table 203, **Res. Ledger Entry**.
 - b. Add the following fields.

No.	Field Name	Type	Length	Remarks
123456700	Seminar No.	Code	20	Set the table relation to the Seminar table.
123456701	Seminar Registration No.	Code	20	Set the table relation to the Posted Seminar Reg. Header table.

- c. Set the Caption property for both these fields to match their Name.
- d. Compile, save, and then close the table.

Task 2: Modify the Res. Journal Line Table***High Level Steps***

1. Add the **Seminar No.** and **Seminar Registration No.** fields to the table 207, **Res. Journal Line**.

Detailed Steps

1. Add the **Seminar No.** and **Seminar Registration No.** fields to the table 207, **Res. Journal Line**.
 - a. Design the table 207, **Res. Journal Line**.
 - b. Add the following fields.

No.	Field Name	Type	Length	Remarks
123456700	Seminar No.	Code	20	Set the table relation to the Seminar table.

No.	Field Name	Type	Length	Remarks
123456701	Seminar Registration No.	Code	20	Set the table relation to the Posted Seminar Reg. Header table.

- c. Set the Caption property for both these fields to match their **Name**.
- d. Compile, save, and then close the table.



Note: When the fields with same Nos. and Names exist in multiple tables, you can copy them from one table to another. Here, instead of creating fields, you can copy the fields from the **Res. Ledger Entry** table.

Task 3: Modify the Resource Ledger Entries Page

High Level Steps

1. Add the **Seminar No.** and **Seminar Registration No.** fields to the page 202, **Resource Ledger Entries**.

Detailed Steps

1. Add the **Seminar No.** and **Seminar Registration No.** fields to the page 202, **Resource Ledger Entries**.
 - a. Design the page 202, **Resource Ledger Entries**.
 - b. Above the **Job No.** field, insert the **Seminar No.** and **Seminar Registration No.** fields.
 - c. Compile, save, and then close the page.

Task 4: Modify the Res. Jnl.-Post Line Codeunit

High Level Steps

1. In codeunit 212, Res Jnl.-Post Line, enter code into the **Code** function trigger so that when the function is populating the **ResLedgEntry** fields, it also assigns the **Seminar No.** and **Seminar Registration No.** fields from the **Res. Journal Line** table.

Detailed Steps

1. In codeunit 212, Res Jnl.-Post Line, enter code into the **Code** function trigger so that when the function is populating the **ResLedgEntry** fields, it also assigns the **Seminar No.** and **Seminar Registration No.** fields from the **Res. Journal Line** table.
 - a. Design the codeunit 212, Res. Jnl.-Post Line.

Module 4: Posting

- b. In the Code function trigger, below the assignment of the **Qty.** **per Unit of Measure** field, and above the GetGLSetup line, enter the following code.

```
//CSD1.00>

ResLedgEntry."Seminar No." := "Seminar No.';

ResLedgEntry."Seminar Registration No." := "Seminar Registration No.';

//CSD1.00<
```

- c. Compile, save, and then close the codeunit.

Lab 4.5: Creating the Codeunits for Document Posting

Scenario

Isaac has started developing the codeunits for seminar registration posting, but the task was too complex for him. He completed the Seminar-Post (Yes/No) codeunit. For the Seminar-Post codeunit he declared variables and functions, and then decided to hand over the task to you. Therefore, you complete the Seminar-Post and Seminar-Post (Yes/No) codeunits that Isaac started developing.

When you have completed the development of these codeunits, you must modify the document pages to let users call the posting routine from the RoleTailored client.

Because you have not yet developed any reports for the Seminar Management functional area, you do not have to provide the Post + Print codeunit.

Exercise 1: Complete the Seminar-Post Codeunit

Exercise Scenario

The Seminar-Post codeunit is the central codeunit of seminar registration posting. It takes the Seminar Registration Header record as a parameter, and processes the information that is contained in it to produce a Posted Seminar Registration document. It must also create the seminar ledger entries for the participants, the instructor, the room, any seminar charges, and the resource ledger entries for the instructor and the room.

Task 1: Import the File

High Level Steps

1. Import the starter objects.

Detailed Steps

1. Import the starter objects.
 - a. In Object Designer, click **File** > **Import**.
 - b. Locate the Mod04\Labfiles\ Lab 4.E - Starter.fob object file and click **OK**.
 - c. Click **Yes** to complete the import.
 - d. Close the **Import Objects** window.

Task 2: Complete the **CopyCommentLines** Function

High Level Steps

1. In the **CopyCommentLines** function trigger, enter the code that finds records in the **Seminar Comment Line** table that matches the specified *FromDocumentType* and *FromNumber*, and for each record inserts a copy of the old record, with the **Document Type** and **No.** set to the *ToDocumentType* and *ToNumber*.

Detailed Steps

2. In the **CopyCommentLines** function trigger, enter the code that finds records in the **Seminar Comment Line** table that matches the specified *FromDocumentType* and *FromNumber*, and for each record inserts a copy of the old record, with the **Document Type** and **No.** set to the *ToDocumentType* and *ToNumber*.
 - a. In the **CopyCommentLines** function trigger, enter the following code.

```
SeminarCommentLine.RESET;  
  
SeminarCommentLine.SETRANGE("Document Type",FromDocumentType);  
  
SeminarCommentLine.SETRANGE("No.",FromNumber);  
  
IF SeminarCommentLine.FINDSET(FALSE,FALSE) THEN BEGIN  
  
    REPEAT  
  
        SeminarCommentLine2 := SeminarCommentLine;  
  
        SeminarCommentLine2."Document Type" := ToDocumentType;  
  
        SeminarCommentLine2."No." := ToNumber;  
  
        SeminarCommentLine2.INSERT;  
  
    UNTIL SeminarCommentLine.NEXT = 0;  
  
END;
```

Task 3: Complete the CopyCharges Function

High Level Steps

1. In the **CopyCharges** function trigger, enter the code that finds all Seminar Charge records that correspond to the specified *FromNumber*. For each record found, the function transfers the values to a new **Posted Seminar Charge** record, by using the **ToNumber** as the **Seminar Registration No.**.

Detailed Steps

1. In the **CopyCharges** function trigger, enter the code that finds all Seminar Charge records that correspond to the specified *FromNumber*. For each record found, the function transfers the values to a new **Posted Seminar Charge** record, by using the **ToNumber** as the **Seminar Registration No.**.

 **Note:** Because the SeminarCharge and the PstdSeminarCharge variables are based on different tables, you cannot assign the record variables directly. All field values must be assigned individually. If the **PstdSeminarCharge** table field number and types are the same as the **SeminarCharge** table, you can use the **TRANSFERFIELDS** function to transfer all the field values at one time.

- a. In the **CopyCharges** function trigger, enter the following code.

```
SeminarCharge.RESET;  
  
SeminarCharge.SETRANGE("Document No.",FromNumber);  
  
IF SeminarCharge.FINDSET(FALSE,FALSE) THEN BEGIN  
  
    REPEAT  
  
        PstdSeminarCharge.TRANSFERFIELDS(SeminarCharge);  
  
        PstdSeminarCharge."Document No." := ToNumber;  
  
        PstdSeminarCharge.INSERT;  
  
    UNTIL SeminarCharge.NEXT = 0;  
  
END;
```

Task 4: Complete the PostResJnlLine Function

High Level Steps

1. In the **PostResJnlLine** function trigger, enter WITH code block for the *SeminarRegHeader* record variable.
2. In the WITH code block, enter the code that does the following:
 - o Makes sure that the **Quantity Per Day** field on the **Resource** record is not empty,
 - o Initializes a **Resource Journal Line** record.
 - o Sets its **Entry Type** to Usage.
 - o Assigns the **Document No.** from the *PstdSeminarRegHeader* record variable.
 - o Assigns the **Resource No.** from the *Resource* record parameter.
3. In the WITH code block, append the code that assigns the following field values from the seminar Registration Header record:
 - o Posting Date
 - o Reason Code
 - o Description
 - o Gen. Prod. Posting Group
 - o Posting No. Series

Assign these from the fields that have the same name, except for the **Description** field. Assign this from the **Seminar Name** field. Assign the **Source Code** field from the *SourceCode* global variable. Assign the **Resource No., Unit of Measure Code** and **Unit Cost** fields from the *Resource* record parameter. Set the **Qty. per Unit of Measure** field to 1.
4. In the WITH code block, append the code that calculates the **Quantity** field as the product of the **Duration** field from the *SeminarRegHeader* record variable and the **Quantity Per Day** field from the *Resource* record parameter. Then, calculate the **Total Cost** field as the product of the **Unit Cost** and **Quantity** field values. Then, assign values to **Seminar No.** and **Seminar Registration No.** fields. Finally, call the **RunWithCheck** function of the Res. Jnl.-Post Line codeunit.
5. After the WITH block, find the last **Resource Ledger Entry**, and return its **Entry No.** field value as the function return value.

Detailed Steps

1. In the **PostResJnlLine** function trigger, enter WITH code block for the *SeminarRegHeader* record variable.
 - b. In the **PostResJnlLine** function trigger, enter the following code.

```
WITH SeminarRegHeader DO BEGIN  
      
END;
```

2. In the WITH code block, enter the code that does the following:
 - o Makes sure that the **Quantity Per Day** field on the **Resource** record is not empty,
 - o Initializes a **Resource Journal Line** record.
 - o Sets its **Entry Type** to Usage.
 - o Assigns the **Document No.** from the *PstdSeminarRegHeader* record variable.
 - o Assigns the **Resource No.** from the *Resource* record parameter.
 - a. In the WITH block, enter the following code.

```
Resource.TESTFIELD("Quantity Per Day");  
  
ResJnlLine.INIT;  
  
ResJnlLine."Entry Type" := ResJnlLine."Entry Type"::Usage;  
  
ResJnlLine."Document No." := PstdSeminarRegHeader."No.";  
  
ResJnlLine."Resource No." := Resource."No.";
```

3. In the WITH code block, append the code that assigns the following field values from the seminar Registration Header record:
 - o Posting Date
 - o Reason Code
 - o Description
 - o Gen. Prod. Posting Group
 - o Posting No. Series

Assign these from the fields that have the same name, except for the **Description** field. Assign this from the **Seminar Name** field. Assign the **Source Code** field from the *SourceCode* global variable. Assign the **Resource No.**, **Unit of Measure Code** and **Unit Cost** fields from the *Resource* record parameter. Set the **Qty. per Unit of Measure** field to 1.

Module 4: Posting

- a. In the WITH code block, append the following code.

```
ResJnlLine."Posting Date" := "Posting Date";  
  
ResJnlLine."Reason Code" := "Reason Code";  
  
ResJnlLine.Description := "Seminar Name";  
  
ResJnlLine."Gen. Prod. Posting Group" := "Gen. Prod. Posting Group";  
  
ResJnlLine."Posting No. Series" := "Posting No. Series";  
  
ResJnlLine."Source Code" := SourceCode;  
  
ResJnlLine."Resource No." := Resource."No.";  
  
ResJnlLine."Unit of Measure Code" := Resource."Base Unit of Measure";  
  
ResJnlLine."Unit Cost" := Resource."Unit Cost";  
  
ResJnlLine."Qty. per Unit of Measure" := 1;
```

4. In the WITH code block, append the code that calculates the **Quantity** field as the product of the **Duration** field from the *SeminarRegHeader* record variable and the **Quantity Per Day** field from the *Resource* record parameter. Then, calculate the **Total Cost** field as the product of the **Unit Cost** and **Quantity** field values. Then, assign values to **Seminar No.** and **Seminar Registration No.** fields. Finally, call the **RunWithCheck** function of the Res. Jnl.-Post Line codeunit.

- a. In the WITH code block, append the following code.

```
ResJnlLine.Quantity := Duration * Resource."Quantity Per Day";  
  
ResJnlLine."Total Cost" := ResJnlLine."Unit Cost" * ResJnlLine.Quantity;  
  
ResJnlLine."Seminar No." := "Seminar No.";  
  
ResJnlLine."Seminar Registration No." := PstdSeminarRegHeader."No.";  
ResJnlPostLine.RunWithCheck(ResJnlLine);
```

5. After the WITH block, find the last **Resource Ledger Entry**, and return its **Entry No.** field value as the function return value.
 - a. After the WITH block, enter the following code.

```
ResLedgEntry.FINDLAST;  
  
EXIT(ResLedgEntry."Entry No.");
```

Task 5: Complete the PostSeminarJnlLine Function

High Level Steps

1. In the **PostSeminarJnlLine** function trigger, enter WITH code block for the *SeminarRegHeader* record variable.
2. In the WITH block, enter the code that initializes the *SeminarJnlLine* record variable, and then assigns the following fields from the *SeminarRegHeader* and *PstdSeminarRegHeader* record variables, as appropriate:
 - Seminar No.
 - Posting Date
 - Document Date
 - Document No.
 - Charge Type
 - Instructor Resource No.
 - Starting Date
 - Seminar Registration No.
 - Room Resource No.
 - Source Type
 - Source Code
 - Reason Code
 - Posting No.
3. To the WITH code block, append the code that compares the *ChargeType* parameter to all possible option values that it can have.
4. If the *ChargeType* is Instructor, retrieve the appropriate **Resource** record, and then on the *SeminarJnlLine* record variable, assign **Description** from the instructor **Name**, set **Type** to **Resource**, set **Chargeable** to FALSE, and set **Quantity** to the **Duration** field from the **SeminarRegHeader**. Finally, call the **PostResJnlLine**, and assign its return value to the **Res. Ledger Entry No.** field of the *SeminarJnlLine* record variable.
5. If the *ChargeType* is Room, retrieve the appropriate **Resource**, and then on the *SeminarJnlLine* record variable, assign **Description** from the room **Name**, set **Type** to **Resource**, set **Chargeable** to FALSE, and set **Quantity** to the **Duration** field from the **SeminarRegHeader**. Finally, call the **PostResJnlLine**, and assign its return value to the **Res. Ledger Entry No.** field of the *SeminarJnlLine* record variable.

6. If the *ChargeType* is Participant, assign the fields to the *SeminarJnlLine* record variable from the *SeminarRegLine* record variable. Assign the following fields:

- o Bill-to Customer No.
- o Participant Contact No.
- o Participant Name
- o Description
- o Chargeable
- o Unit Price
- o Total Price

Description is set from **Participant Name**, **Chargeable** is set from **To Invoice**, and **Unit Price** and **Total Price** are set from **Amount**. Set the **Type** to **Resource** and **Quantity** to 1.

7. If *ChargeType* is Charge, then assign the fields to the *SeminarJnlLine* record variable from the *SeminarCharge* record variable. Assign the following fields:

- o Description
- o Bill-to Customer No.
- o Type
- o Quantity
- o Unit Price
- o Total Price
- o Chargeable

Chargeable is set from **To Invoice**.

8. After the CASE block, post the *SeminarJnlLine* through the Seminar Jnl.-Post Line codeunit.

Detailed Steps

1. In the **PostSeminarJnlLine** function trigger, enter WITH code block for the *SeminarRegHeader* record variable.
 - a. In the **PostSeminarJnlLine** function trigger, enter the following code.

```
WITH SeminarRegHeader DO BEGIN  
  ...  
END;
```

C/SIDE Solution Development in Microsoft Dynamics® NAV 2013

2. In the WITH block, enter the code that initializes the *SeminarJnlLine* record variable, and then assigns the following fields from the *SeminarRegHeader* and *PstdSeminarRegHeader* record variables, as appropriate:
 - o Seminar No.
 - o Posting Date
 - o Document Date
 - o Document No.
 - o Charge Type
 - o Instructor Resource No.
 - o Starting Date
 - o Seminar Registration No.
 - o Room Resource No.
 - o Source Type
 - o Source Code
 - o Reason Code
 - o Posting No.
- a. In the WITH block, enter the following code.

```
SeminarJnlLine.INIT;  
  
SeminarJnlLine."Seminar No." := "Seminar No.;"  
  
SeminarJnlLine."Posting Date" := "Posting Date";  
  
SeminarJnlLine."Document Date" := "Document Date";  
  
SeminarJnlLine."Document No." := PstdSeminarRegHeader."No.";  
  
SeminarJnlLine."Charge Type" := ChargeType;  
  
SeminarJnlLine."Instructor Resource No." := "Instructor Resource No.";  
  
SeminarJnlLine."Starting Date" := "Starting Date";  
  
SeminarJnlLine."Seminar Registration No." := PstdSeminarRegHeader."No.";  
  
SeminarJnlLine."Room Resource No." := "Room Resource No.";  
  
SeminarJnlLine."Source Type" := SeminarJnlLine."Source Type"::Seminar;  
  
SeminarJnlLine."Source No." := "Seminar No.";  
  
SeminarJnlLine."Source Code" := SourceCode;
```

```
SeminarJnlLine."Reason Code" := "Reason Code";  
SeminarJnlLine."Posting No. Series" := "Posting No. Series";
```

3. To the WITH code block, append the code that compares the *ChargeType* parameter to all possible option values that it can have.
 - a. In the WITH block, append the following code.

```
CASE ChargeType OF  
  
    ChargeType::Instructor:  
  
        BEGIN  
  
        END;  
  
    ChargeType::Room:  
  
        BEGIN  
  
        END;  
  
    ChargeType::Participant:  
  
        BEGIN  
  
        END;  
  
    ChargeType::Charge:  
  
        BEGIN  
  
        END;  
  
    END;
```

4. If the *ChargeType* is Instructor, retrieve the appropriate **Resource** record, and then on the *SeminarJnlLine* record variable, assign **Description** from the instructor **Name**, set **Type** to **Resource**, set **Chargeable** to FALSE, and set **Quantity** to the **Duration** field from the **SeminarRegHeader**. Finally, call the *PostResJnlLine*, and assign its return value to the **Res. Ledger Entry No.** field of the *SeminarJnlLine* record variable.
 - a. In the Instructor block, enter the following code.

```
Instructor.GET("Instructor Resource No.");  
  
SeminarJnlLine.Description := Instructor.Name;
```

```
SeminarJnlLine.Type := SeminarJnlLine.Type::Resource;  
  
SeminarJnlLine.Chargeable := FALSE;  
  
SeminarJnlLine.Quantity := Duration;  
  
SeminarJnlLine."Res. Ledger Entry No." := PostResJnlLine(Instructor);
```

5. If the *ChargeType* is Room, retrieve the appropriate **Resource**, and then on the *SeminarJnlLine* record variable, assign **Description** from the room **Name**, set **Type** to **Resource**, set **Chargeable** to FALSE, and set **Quantity** to the **Duration** field from the **SeminarRegHeader**. Finally, call the **PostResJnlLine**, and assign its return value to the **Res. Ledger Entry No.** field of the *SeminarJnlLine* record variable.
 - a. In the Room block, enter the following code.

```
Room.GET("Room Resource No.");  
  
SeminarJnlLine.Description := Room.Name;  
  
SeminarJnlLine.Type := SeminarJnlLine.Type::Resource;  
  
SeminarJnlLine.Chargeable := FALSE;  
  
SeminarJnlLine.Quantity := Duration;  
  
// Post to resource ledger  
  
SeminarJnlLine."Res. Ledger Entry No." := PostResJnlLine(Room);
```

6. If the *ChargeType* is Participant, assign the fields to the *SeminarJnlLine* record variable from the *SeminarRegLine* record variable. Assign the following fields:
 - o Bill-to Customer No.
 - o Participant Contact No.
 - o Participant Name
 - o Description
 - o Chargeable
 - o Unit Price
 - o Total Price

Description is set from **Participant Name**, **Chargeable** is set from **To Invoice**, and **Unit Price** and **Total Price** are set from **Amount**. Set the **Type** to **Resource** and **Quantity** to 1.

Module 4: Posting

- a. In the Participant block, enter the following code.

```
SeminarJnlLine."Bill-to Customer No." := SeminarRegLine."Bill-to Customer No.";  
  
SeminarJnlLine."Participant Contact No." := SeminarRegLine."Participant Contact  
No.";  
  
SeminarJnlLine."Participant Name" := SeminarRegLine."Participant Name";  
  
SeminarJnlLine.Description := SeminarRegLine."Participant Name";  
  
SeminarJnlLine.Type := SeminarJnlLine.Type::Resource;  
  
SeminarJnlLine.Chargeable := SeminarRegLine."To Invoice";  
  
SeminarJnlLine.Quantity := 1;  
  
SeminarJnlLine."Unit Price" := SeminarRegLine.Amount;  
  
SeminarJnlLine."Total Price" := SeminarRegLine.Amount;
```

7. If *ChargeType* is Charge, then assign the fields to the *SeminarJnlLine* record variable from the *SeminarCharge* record variable. Assign the following fields:

- o Description
- o Bill-to Customer No.
- o Type
- o Quantity
- o Unit Price
- o Total Price
- o Chargeable

Chargeable is set from **To Invoice**.

- a. In the Charge block, enter the following code.

```
SeminarJnlLine.Description := SeminarCharge.Description;  
  
SeminarJnlLine."Bill-to Customer No." := SeminarCharge."Bill-to Customer No.";  
  
SeminarJnlLine.Type := SeminarCharge.Type;  
  
SeminarJnlLine.Quantity := SeminarCharge.Quantity;  
  
SeminarJnlLine."Unit Price" := SeminarCharge."Unit Price";  
  
SeminarJnlLine."Total Price" := SeminarCharge."Total Price";
```

```
SeminarJnlLine.Chargeable := SeminarCharge."To Invoice";
```

8. After the CASE block, post the *SeminarJnlLine* through the Seminar Jnl.-Post Line codeunit.
 - a. After the CASE block, enter the following code.

```
SeminarJnlPostLine.RunWithCheck(SeminarJnlLine);
```

Task 6: Complete the PostCharges Function

High Level Steps

1. In the **PostCharges** function trigger, enter the code that calls the **PostSeminarJnlLine** function for every **Seminar Charge** for the current *SeminarRegHeader*.

Detailed Steps

1. In the **PostCharges** function trigger, enter the code that calls the **PostSeminarJnlLine** function for every **Seminar Charge** for the current *SeminarRegHeader*.
 - a. In the PostCharges function trigger, enter the following code.

```
SeminarCharge.RESET;  
  
SeminarCharge.SETRANGE("Document No.",SeminarRegHeader."No.");  
  
IF SeminarCharge.FINDSET(FALSE,FALSE) THEN BEGIN  
  
    REPEAT  
  
        PostSeminarJnlLine(3); // Charge  
  
    UNTIL SeminarCharge.NEXT = 0;  
  
END;
```

Task 7: Add Code to the OnRun Trigger

High Level Steps

1. In the OnRun trigger, enter the code that clears all variables and sets the *SeminarRegHeader* record variable to the current record. Then create a WITH block for the *SeminarRegHeader* variable. After the WITH block, set the current record to the *SeminarRegHeader* record variable.
2. In the WITH block, make sure that the following fields are not empty and that the **Status** field value is Closed:
 - o Posting Date
 - o Document Date

- Seminar No.
 - Duration
 - Instructor Resource No.
 - Room Resource No.
3. If there are no lines for the current document, throw an error.
 4. Open a dialog box to show the posting progress.
 5. If the **Posting No.** is blank on the registration header, make sure that the **Posting No. Series** is not blank. Then assign the **Posting No.** to the next number from the posting number series, as indicated on the header. Then, modify the header and perform a commit. Finally, lock the **Seminar Registration Line** table.
 6. Assign the *SourceCode* variable from the **Seminar** field of the **Source Code Setup** table.
 7. Initialize a new **Posted Seminar Reg. Header** record, and then transfer the fields from the registration header. Assign **No.** and **No. Series** to the **Posting No.** and **Posting No. Series** fields from the registration header. Assign **Source Code** from the *SourceCode* variable, and **User ID** from the **USERID** function. Finally, insert the **Seminar Reg. Header** record.
 8. Update the dialog box.
 9. Copy the comment lines and charges from the registration header to the posted registration header, by calling the **CopyCommentLines** and **CopyCharges** functions.
 10. Set the *LineCount* variable to zero, and prepare the loop for the registration lines of the current registration header.
 11. For each registration line, increase the *LineCount* variable by one, update the dialog window, and make sure that **Bill-to Customer No.** and **Participant Contact No.** are not empty. If the line should not be invoiced, reset its **Seminar Price**, **Line Discount %**, **Line Discount Amount** and **Amount** fields to zero. Post the participant line by calling the **PostSeminarJnlLine** function. Finally, initialize and insert a new posted registration line by transferring the fields from the registration line, and assigning the appropriate **Document No.** value.
 12. Post the charges by calling the **PostCharges** function. Then post the seminar ledger entry for the instructor and the room by calling the **PostSeminarJnlLine** function.
 13. Delete the registration header, lines, comments, and charges.
 14. Save the codeunit.

Detailed Steps

1. In the OnRun trigger, enter the code that clears all variables and sets the *SeminarRegHeader* record variable to the current record. Then create a WITH block for the *SeminarRegHeader* variable. After the WITH block, set the current record to the *SeminarRegHeader* record variable.
 - a. In the OnRun trigger, enter the following code.

```
CLEARALL;  
  
SeminarRegHeader := Rec;  
  
WITH SeminarRegHeader DO BEGIN  
  
END;  
  
Rec := SeminarRegHeader;
```

2. In the WITH block, make sure that the following fields are not empty and that the **Status** field value is Closed:
 - o Posting Date
 - o Document Date
 - o Seminar No.
 - o Duration
 - o Instructor Resource No.
 - o Room Resource No.

 **Note:** For all remaining steps in this task, always append the code to the end of the WITH block.

- a. In the WITH block, enter the following code.

```
TESTFIELD("Posting Date");  
  
TESTFIELD("Document Date");  
  
TESTFIELD("Seminar No.");  
  
TESTFIELD(Duration);
```

```
TESTFIELD("Instructor Resource No.");
TESTFIELD("Room Resource No.");
TESTFIELD(Status,Status::Closed);
```

3. If there are no lines for the current document, throw an error.

- a. Enter the following code.

```
SeminarRegLine.RESET;
SeminarRegLine.SETRANGE("Document No.", "No.");
IF SeminarRegLine.ISEMPTY THEN
  ERROR(Text001);
```

4. Open a dialog box to show the posting progress.

- a. Enter the following code.

```
Window.OPEN(
  '#1#####
  Text02);
Window.UPDATE(1,STRSUBSTNO('%1 %2',Text03,"No."));
```

5. If the **Posting No.** is blank on the registration header, make sure that the **Posting No. Series** is not blank. Then assign the **Posting No.** to the next number from the posting number series, as indicated on the header. Then, modify the header and perform a commit. Finally, lock the **Seminar Registration Line** table.

- a. Enter the following code.

```
IF SeminarRegHeader."Posting No." = "" THEN BEGIN
  TESTFIELD("Posting No. Series");
  "Posting No." := NoSeriesMgt.GetNextNo("Posting No. Series", "Posting Date", TRUE);
  MODIFY;
  COMMIT;
```

```
END;
```

```
SeminarRegLine.LOCKTABLE;
```

6. Assign the *SourceCode* variable from the **Seminar** field of the **Source Code Setup** table.

- a. Enter the following code.

```
SourceCodeSetup.GET;
```

```
SourceCode := SourceCodeSetup.Seminar;
```

7. Initialize a new **Posted Seminar Reg. Header** record, and then transfer the fields from the registration header. Assign **No.** and **No. Series** to the **Posting No.** and **Posting No. Series** fields from the registration header. Assign **Source Code** from the *SourceCode* variable, and **User ID** from the **USERID** function. Finally, insert the **Seminar Reg. Header** record.

- a. Enter the following code.

```
PstdSeminarRegHeader.INIT;
```

```
PstdSeminarRegHeader.TRANSFERFIELDS(SeminarRegHeader);
```

```
PstdSeminarRegHeader."No." := "Posting No.:";
```

```
PstdSeminarRegHeader."No. Series" := "Posting No. Series";
```

```
PstdSeminarRegHeader."Source Code" := SourceCode;
```

```
PstdSeminarRegHeader."User ID" := USERID;
```

```
PstdSeminarRegHeader.INSERT;
```

8. Update the dialog box.

- a. Enter the following code.

```
Window.UPDATE(1,STRSUBSTNO(Text004,"No.",
```

```
PstdSeminarRegHeader."No."));
```

9. Copy the comment lines and charges from the registration header to the posted registration header, by calling the **CopyCommentLines** and **CopyCharges** functions.

- a. Enter the following code.

```

CopyCommentLines(
    SeminarCommentLine."Document Type"::"Seminar Registration",
    SeminarCommentLine."Document Type"::"Posted Seminar Registration",
    "No.",PstdSeminarRegHeader."No.");
CopyCharges("No.",PstdSeminarRegHeader."No.");

```

10. Set the *LineCount* variable to zero, and prepare the loop for the registration lines of the current registration header.

- a. Enter the following code.

```

LineCount := 0;
SeminarRegLine.RESET;
SeminarRegLine.SETRANGE("Document No.", "No.");
IF SeminarRegLine.FINDSET THEN BEGIN
    REPEAT
        UNTIL SeminarRegLine.NEXT = 0;
END;

```

11. For each registration line, increase the *LineCount* variable by one, update the dialog window, and make sure that **Bill-to Customer No.** and **Participant Contact No.** are not empty. If the line should not be invoiced, reset its **Seminar Price**, **Line Discount %**, **Line Discount Amount** and **Amount** fields to zero. Post the participant line by calling the **PostSeminarJnlLine** function. Finally, initialize and insert a new posted registration line by transferring the fields from the registration line, and assigning the appropriate **Document No.** value.

- a. In the REPEAT block, enter the following code.

```

LineCount := LineCount + 1;
Window.UPDATE(2,LineCount);

SeminarRegLine.TESTFIELD("Bill-to Customer No.");
SeminarRegLine.TESTFIELD("Participant Contact No.");

```

```
IF NOT SeminarRegLine."To Invoice" THEN BEGIN  
    SeminarRegLine."Seminar Price" := 0;  
    SeminarRegLine."Line Discount %" := 0;  
    SeminarRegLine."Line Discount Amount" := 0;  
    SeminarRegLine.Amount := 0;  
END;  
  
// Post seminar entry  
PostSeminarJnlLine(2); // Participant  
  
// Insert posted seminar registration line  
PstdSeminarRegLine.INIT;  
PstdSeminarRegLine.TRANSFERFIELDS(SeminarRegLine);  
PstdSeminarRegLine."Document No." := PstdSeminarRegHeader."No.";  
PstdSeminarRegLine.INSERT;
```

12. Post the charges by calling the **PostCharges** function. Then post the seminar ledger entry for the instructor and the room by calling the **PostSeminarJnlLine** function.

- a. After the REPEAT block, enter the following code.

```
// Post charges to seminar ledger  
PostCharges;  
  
// Post instructor to seminar ledger  
PostSeminarJnlLine(0); // Instructor
```

```
// Post seminar room to seminar ledger  
  
PostSeminarJnlLine(1); // Room
```

13. Delete the registration header, lines, comments, and charges.

a. Enter the following code.

```
DELETE;  
  
SeminarRegLine.DELETEALL;  
  
  
SeminarCommentLine.SETRANGE("Document Type",  
    SeminarCommentLine."Document Type"::"Seminar Registration");  
  
SeminarCommentLine.SETRANGE("No.", "No.");  
  
SeminarCommentLine.DELETEALL;  
  
  
SeminarCharge.SETRANGE(Description);  
  
SeminarCharge.DELETEALL;
```

14. Save the codeunit.

a. Compile and save the codeunit, and close the **C/AL Editor** window.

Exercise 2: Enable Posting from the Seminar Registration Pages

Exercise Scenario

After you complete the development of the seminar registration posting routine, you must enable users to start the routine from the relevant pages. In Microsoft Dynamics NAV 2013, users must be able to start posting from the Document and the List pages for documents.

To meet the Microsoft Dynamics NAV 2013 user experience standards, you must add the Post action to the **Seminar Registration** and **Seminar Registration List** pages.

Task 1: Modify the Pages

High Level Steps

1. Add an action to the **Seminar Registration** page that runs the Seminar-Post (Yes/No) codeunit.
2. Add an action to the **Seminar Registration List** page that runs the Seminar-Post (Yes/No) codeunit.

Detailed Steps

1. Add an action to the **Seminar Registration** page that runs the Seminar-Post (Yes/No) codeunit.
 - a. Design the page 123456710, **Seminar Registration**.
 - b. Add the ActionItems action container.
 - c. Add the Posting action group to ActionItems.
 - d. Add the **Post** action to Posting group.
 - e. Define the following properties:

Property	Value
Caption	P&ost
Image	PostDocument
Promoted	Yes
PromotedCategory	Process
ShortCutKey	F9
RunObject	Codeunit Seminar-Post (Yes/No)

- f. Compile, save, and then close the page.
2. Add an action to the **Seminar Registration List** page that runs the Seminar-Post (Yes/No) codeunit.
 - a. Design the page 123456710, **Seminar Registration**.
 - b. From the **Page – Action Designer**, select the rows you created in the step 1 and copy them.
 - c. Close the **Action Designer**, and the **Page Designer** windows.
 - d. Design the page 123456713, **Seminar Registration List**.
 - e. In the **Page – Action Designer** page, paste the actions copied in step b.
 - f. Close the **Action Designer**.
 - g. Compile, save, and then close the page.

Module Review

Module Review and Takeaways

There are two types of posting routines in Microsoft Dynamics NAV 2013: journal posting and document posting routines. These types of posting routines always use the same data model and processing principles, and apply a series of recognizable design patterns. To successfully customize Microsoft Dynamics NAV 2013 and extend it with the new functional areas that support posting routines, you must have a thorough understanding of these standards, and follow them consistently.

A journal in Microsoft Dynamics NAV 2013 consists of at least one of the following:

- The **Journal Line** table if it exists only to support the posting routine.
- The **Journal Batch** and **Journal Template** tables if they enable users to enter information into them from the RoleTailored client.

A journal posting routine in Microsoft Dynamics NAV 2013 consists of at least the Check Line and Post Line codeunits if journals are only posted by the system. You can have several more starter codeunits to handle the user interaction and batch posting, if users manage the journals directly.

Document posting data models consist of the same set of tables as the open (working) documents. At a minimum, this is the Header and the Line table, but may also include any other subsidiary table.

A document posting routine in Microsoft Dynamics NAV 2013 copies the open documents into posted documents, and depends on the **TRANSFERFIELDS** function to simplify the development and maintenance of the posting process. A document posting routine also translates the document information into at least one but frequently many journals, and posts them as an important part of the document posting process. A posted document therefore results not only in the posted document tables, but also in ledger entries.

This module covered the following subjects:

- Posting in Microsoft Dynamics NAV 2013 from journals and from documents
- Tables and codeunits of a standard posting routine
- Key aspects of programming to be aware of to maximize performance

Test Your Knowledge

Test your knowledge with the following questions.

1. Which three tables make up a journal?

2. Which codeunit from the journal posting routine makes sure that data journal lines is complete and valid, before actual posting starts and before any locking occurs?

3. What is the difference between the Post Batch and Batch Post codeunits?

4. Multiline comments can be nested. When using multiline comments you must make sure that each open comments ({) sign is followed by a properly nested close comments (}) sign.

() True

() False

5. A table is automatically locked when you start writing data to it.

() True

() False

Module 4: Posting

6. Which C/AL function enables you to lock a table immediately and explicitly, even if you make no write access to it?

7. When are the locks released from a locked table?

Test Your Knowledge Solutions

Module Review and Takeaways

1. Which three tables make up a journal?

MODEL ANSWER:

Journal Template, Journal Batch, and Journal Line.

2. Which codeunit from the journal posting routine makes sure that data journal lines is complete and valid, before actual posting starts and before any locking occurs?

MODEL ANSWER:

Check Line

3. What is the difference between the Post Batch and Batch Post codeunits?

MODEL ANSWER:

Post Batch posts lines from a single batch. Batch Post calls Post Batch for each batch selected in the **Journal Batches** page.

4. Multiline comments can be nested. When using multiline comments you must make sure that each open comments ({) sign is followed by a properly nested close comments (}) sign.

() True

() False

5. A table is automatically locked when you start writing data to it.

() True

() False

6. Which C/AL function enables you to lock a table immediately and explicitly, even if you make no write access to it?

MODEL ANSWER:

LOCKTABLE

7. When are the locks released from a locked table?

MODEL ANSWER:

The locks are released at the end of a transaction. A transaction ends automatically when the code execution completes, when you explicitly end it by using the COMMIT function, or when you abort the transaction by using the ERROR function.

MODULE 5: FEATURE INTEGRATION

Module Overview

At this stage the Seminar Management application area is a combination of individual functions that CRONUS International Ltd. can use to input seminar master data, perform registrations, and post completed seminar registrations. The next step is to integrate these features with one another and with the standard application. This makes it easier for users to move through the application.

This module addresses the integration of solution functionality with the user interface (UI) of the application.

Objectives

- Integrate previously created Seminar Management features with one another.
- Explain the architecture of the Navigate feature.
- Enable easier searches by adding Navigate functionality to Seminar Management pages.
- Enable looking up Seminar Management information from standard application areas.

Prerequisite Knowledge

Making changes to standard or existing functionality frequently involves making structural changes to tables that already contain data. These structural changes may include any of the following:

- Adding new fields
- Deleting existing fields
- Changing the data type or length of existing fields
- Changing other table or field properties, such as TableRelation or DataPerCompany

Client data is valuable. Making any structural changes to that data requires planning and accuracy. Microsoft Dynamics NAV 2013 safeguards the data by only allowing specific types of changes to tables that contain data. This prevents any accidental data loss, corruption, or other kinds of problems that can arise from modifications.

By understanding the kind of changes that you can make to existing tables, you can plan implementation and development activities.

Changing Tables that Contain Data

When you developed the Seminar Management functionality, you created several tables, and changed several existing ones. Microsoft Dynamics NAV Development Environment makes sure that no data is lost when you change the structure of a table. This means that when you change a table that contains data, there are important guidelines that specify the changes that are allowed under certain conditions.

Changes to Fields

You can always make the following changes to table fields:

- Change the name.
- Change any properties that only control how data is displayed or formatted.
- Change the TableRelation, ValidateTableRelation, and TestTableRelation properties.
- Change a FlowField back to a regular field.
- Change the CalcFormula on a FlowField.
- Increase the length of a text or a code field.
- Add a new field.



Note: When you increase the length of a Text or a Code field, or add a new field, the only limitation is the maximum record size that is imposed by Microsoft SQL Server. This is 8,060 bytes per record.

You can make the following changes if the field does not contain data in any of the records for any companies in the database:

- Change the Data Type.
 - Change the Field No.
 - Change a normal field into a FlowField.
 - Disable a field by setting Enabled to **No**.
 - Delete the field.
-



Note: The system lets you delete a field even if there are remaining references to the field, such as from CalcFormula of a FlowField, TableRelation, page control, report column, a code reference, or any other type of reference. After you delete a field, you manually must remove all references to the field from other objects. When you run an object that has an invalid field reference, a run-time error occurs.



Best Practice: Changing the Field No. has the same effect on all field references as deleting the field completely. You should change the Field No. of an existing field only if it is necessary.

In addition to these rules, you can also decrease the length of a Text or a Code field as long as the length of all values in that field in all companies in the database is equal to or smaller than the target size.

If you try to make a prohibited change, Microsoft Dynamics NAV Development Environment warns you, and then prevents the change.

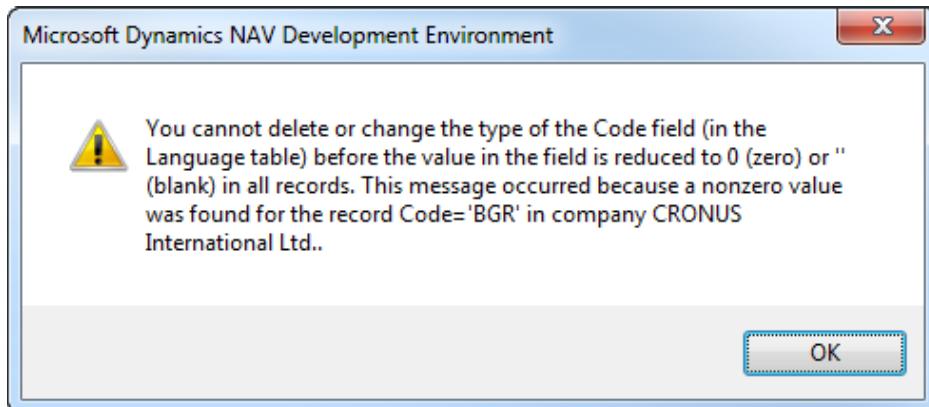


FIGURE 5.1: INVALID FIELD CHANGE WARNING WINDOW

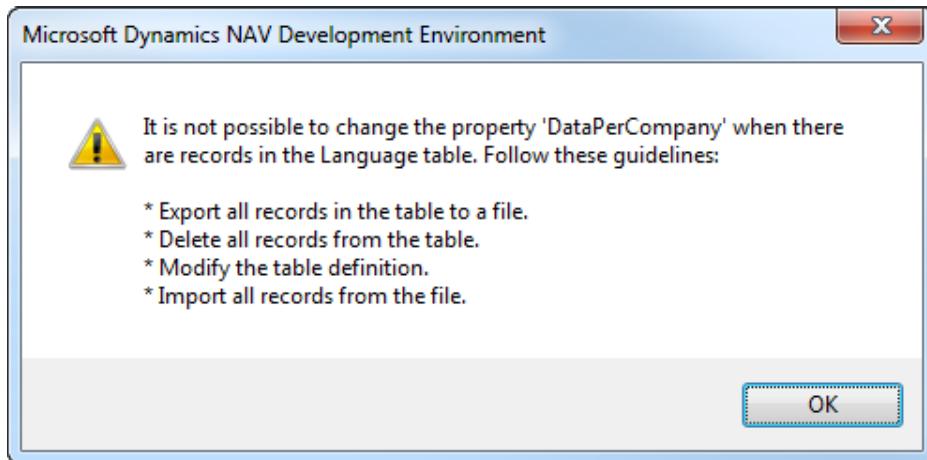
Changes to Tables

Even though changes to fields are the most common structural changes that you make to tables, there are certain changes that affect tables directly. Typically you can make most changes, including changing the Name, the ID, and most of the other properties.

The DataPerCompany table property is the only property that has specific rules for making changes. The following guidelines explain the conditions under which you can change this property:

- If there is only one company in the database, you can always change DataPerCompany to either value.
- If there are more companies in the database, you can change DataPerCompany to **Yes**. You can do this only if there is no data in the table in any of the companies, or if there is data in the table in only one of the companies. The table is empty in all other companies.
- If there are more companies in the database, and there is data in the table of any company, you cannot change this property to **No**.

If you try to change the DataPerCompany property in any of these prohibited situations, Microsoft Dynamics NAV Development Environment warns you and then prevents the change.



Seminar Feature Integration

You are now ready to integrate the Seminar Management features with the standard application and to one another. You have already integrated some features earlier, such as linking lists to card pages, or enabling users to call posting routines from document pages. Except for these simple and most common feature integration steps, you must provide deeper integration to maintain a consistent user experience across the application.

The following table summarizes the typical features that integrate different application functionalities.

Category	Features
Trail Record	<ul style="list-style-type: none">Enables the Navigate feature to search for new ledger entry and posted document types.
Master Pages	<ul style="list-style-type: none">Create new documents from master pages.Access open documents from master pages.Access ledger entries from master pages.
Transactions	<ul style="list-style-type: none">Access Navigate from posted document pages.Access Navigate from ledger entries.

Integrating these features enables users to be more productive for the following reasons:

- It reduces the number of clicks for users to access related features that are available in all relevant pages.
- It reduces time that you spend on data entry and filtering by defaulting master data and document number fields.
- It reduces errors that are created by typing incorrect information by defaulting field values.

Solution Design

CRONUS International Ltd. provided no specific functional requirements about the feature integration. Most of the work that relates to feature integration belongs to the domain of Microsoft Dynamics NAV 2013 standards and best practices.

However, the following two nonfunctional requirements address customer productivity and ease-of-use issues:

- The solution must be consistent, user-friendly, and easy to learn and to use.
- Any custom-built functionality must follow the standards, principles, and best practices of Microsoft Dynamics NAV 2013, and must seamlessly integrate into the standard application. The solution must enable users to be productive and spend as little time as possible searching and filtering.

You can address these requirements by integrating features. Microsoft Dynamics NAV 2013 standard functionality provides many examples of integrating features when you develop your custom application functionality.

The three most common user tasks in Microsoft Dynamics NAV 2013 are as follows:

- Creating new transactions
- Maintaining and processing existing transactions
- Analyzing transaction history

The most common user task is entering transactional information. This includes documents and journals. Because documents and journals are always related to a master record, the majority of master pages in Microsoft Dynamics NAV 2013 provide a quick way to enter a document or a journal for a master record. For example, you can create a new quote or an invoice for a customer directly from the **Customer Card** or **Customer List**, or you can access the **Item Journal** directly from the **Item Card** or **Item List**.

Module 5: Feature Integration

In addition to creating new documents for a master record, you must quickly access any existing documents that are related to that master record. For example, from **Customer Card** or **Customer List**, you can quickly access any quotes, invoices, or other document types that are related to the selected customer. This provides an intuitive user experience, and makes it easy for users to quickly access related information.

Finally, users typically access transaction history for a master record. Therefore, users can click **Ledger Entries** on each master record card. This provides a standard and consistent way to access the ledger entries for the master record.

Ledger Entries are a mandatory action on every master page. You can also press CTRL+F7. This is the system-wide shortcut for accessing related ledger entries from any master page. For example, to access **Customer Ledger Entries** from **Customer Card**, you can press CTRL+F7, or click **Ledger Entries**.

As a general principle, any information that is related to a record must be available from a page that displays that record. Therefore, to stay consistent with Microsoft Dynamics NAV 2013 standards, you must provide similar functionality for Seminar Management: table **Seminar** is the master record, **Seminar Registration** is the document, and Seminar Ledger Entries are ledger entries. Therefore, you must enable the following functionality from the **Seminar Card** and **Seminar List** pages:

- Creation of new documents
- Access to existing Seminar Registration documents
- Access to Seminar Ledger

Development

To integrate Seminar Management, you must change several existing pages and one table to be consistent with standard Microsoft Dynamics NAV 2013 functionality.

Creating New Documents from Master Pages

The standard way to create a new document from a master page is to click the appropriate action in the New group on the **Home** tab, or in the New Document group in the **Actions** tab. This action always runs the document page in the Create mode. It then sets the record link between the master record and the related field in the document table.

When a user inserts a new record, the code in the OnInsert trigger of the document header table checks whether there is a filter on the master record field that filters to a single master record value. If the code in the OnInsert trigger finds such a filter, it validates the master record field to the value in the filter. This makes sure that a new document is assigned automatically to the master record that the master page passed as the record link to the document page.

For example, when a user clicks **Sales Invoice** in the New group on the **Home** tab in the **Customer Card** page, the action runs the **Sales Invoice** page in Create mode. This sets the record link on the **Sell-to Document No.** field of the **Sales Header** table to the value of the **No.** field for the selected customer. Users finish creating the **Sales Invoice** page by either leaving the **No.** field, or selecting a number series by clicking the **AssistEdit** button. The code in the OnInsert trigger of the **Sales Header** table then checks whether there is a filter on the **Sell-to Customer No.** field for a single customer **No.**. If there is such a filter, the code validates the **Sell-to Customer No.** field to the value in the filter.

The following code example from the OnInsert trigger in the **Sales Header** table is responsible for applying the record link filter from the **Customer Card** page to every new sales document.

Applying the Record Link Filter in the Sales Header Table

```
IF GETFILTER("Sell-to Customer No.") <> "" THEN  
    IF GETRANGEMIN("Sell-to Customer No.") = GETRANGEMAX("Sell-to Customer No.") THEN  
        VALIDATE("Sell-to Customer No.",GETRANGEMIN("Sell-to Customer No."));
```

Tables

To enable creation of new Seminar Registration documents from the **Seminar Card** or **Seminar List** pages, add code to the OnInsert trigger of the **Seminar Registration Header** table. This code performs the following logic:

- Checks whether there is a filter on the **Seminar No.** field.
- If there is a filter on the **Seminar No.** field, the code checks whether the filter is to a single Seminar No. value.
- If the filter is to a single value, the code validates the **Seminar No.** field to the value in the filter on the **Seminar No.** field.

Pages

You must change the **Seminar Card** and **Seminar List** pages by adding the following action structure:

- Related Information (container)
 - Seminar (group)
 - Seminar Ledger Entries (action)
 - Registrations (group)
 - Registrations (action)

Module 5: Feature Integration

- New Document Items (container)
 - Seminar Registration (action)

The “Seminar Card Page (123456700) with the Home tab Selected” figure shows the **Seminar Card** page after customization, with the **Seminar Registration** action in the New group on the **Home** tab.

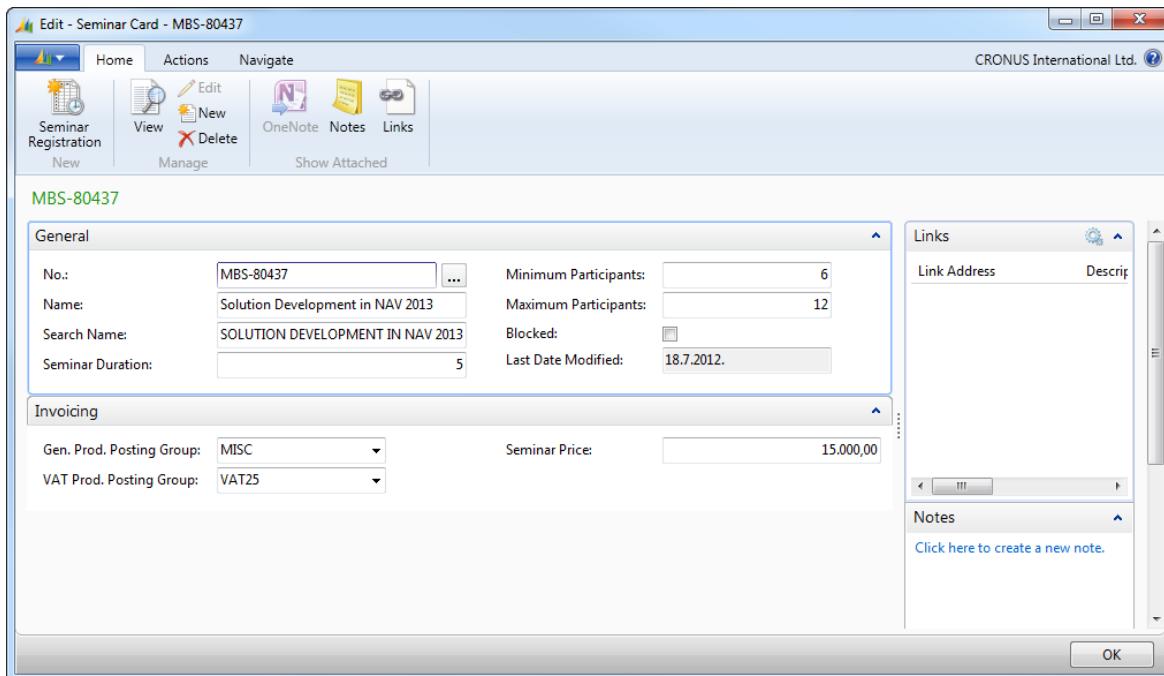


FIGURE 5.3: THE SEMINAR CARD PAGE (123456700) WITH THE HOME TAB SELECTED

The “Seminar List Page (1234565701) with the Navigate tab Selected” figure shows the **Seminar List** page after customization, with the **Ledger Entries** action in the **Seminar** group, and the **Registrations** action in the **Registrations** group on the **Navigate** tab.

C/SIDE Solution Development in Microsoft Dynamics® NAV 2013

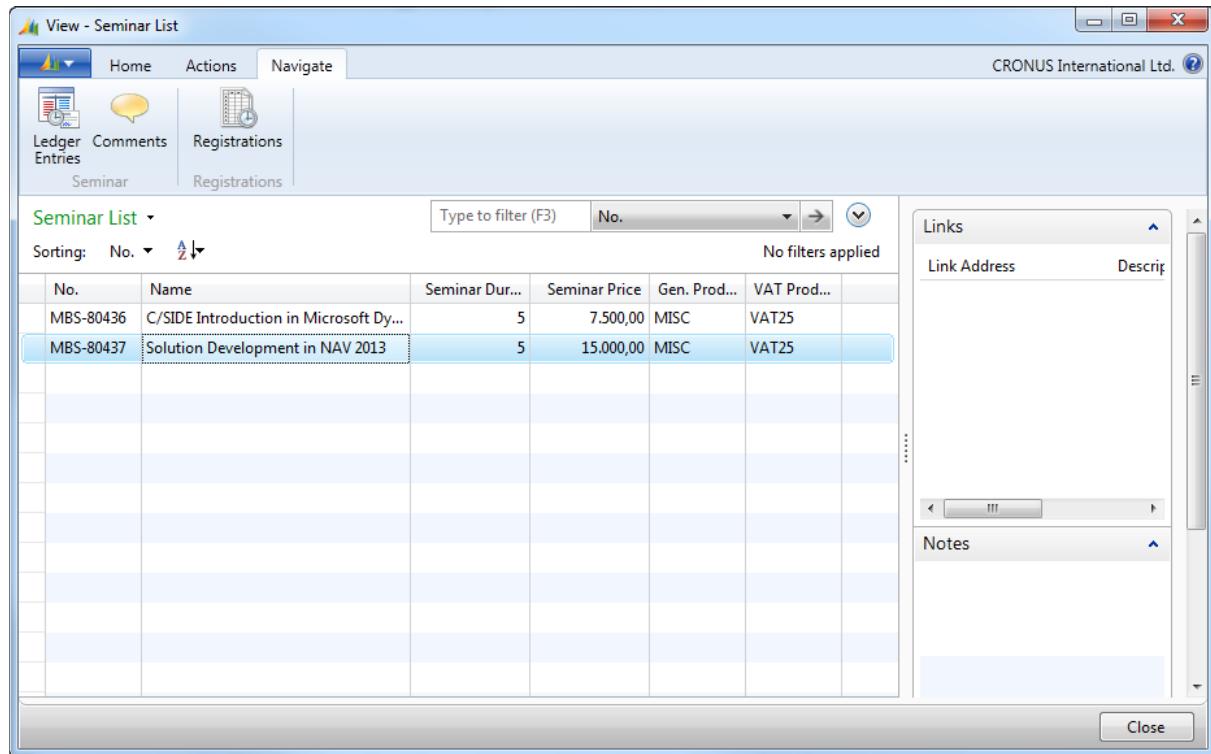


FIGURE 5.4: THE SEMINAR LIST PAGE (1234565701) WITH THE NAVIGATE TAB SELECTED.

Lab 5.1: Integrating Seminar Features

Scenario

Isaac is a developer working on the implementation of Microsoft Dynamics NAV 2013 for CRONUS International Ltd. He is responsible for developing page customizations for Seminar Management integration.

He first must integrate Seminar Management features to enable standard navigation between master data, documents, and posted information.

Exercise 1: Customize Seminar Registration Master Pages

Exercise Scenario

Isaac must add actions to the **Seminar Registration Card** and **Seminar Registration List** pages to do the following:

- Enable creation of new Seminar Registrations.
- Enable access to existing Seminar Registrations.
- Enable access to **Seminar Ledger Entries** for the seminar that is shown in the card or selected in the list.

To create new seminar registrations for a specific seminar, Isaac also must customize the **Seminar** table. If there is a filter on the **Seminar No.** field, and the filter applies to a single **Seminar No.**, Isaac must make sure that the **Seminar No.** field validates to the value in the filter.

Task 1: Customize the Seminar Registration Header Table

High Level Steps

1. Create code in the **Seminar Registration** table to apply the record link filter to the **Seminar No.** field.

Detailed Steps

1. Create code in the **Seminar Registration** table to apply the record link filter to the **Seminar No.** field.
 - a. Design the table 123456710, **Seminar Registration Header**.
 - b. In the **OnInsert** trigger, append the following code:

Appended code in the OnInsert Trigger

```
IF GETFILTER("Seminar No.") <> "" THEN  
    IF GETRANGEMIN("Seminar No.") = GETRANGEMAX("Seminar No.") THEN  
        VALIDATE("Seminar No.",GETRANGEMIN("Seminar No."));
```

- c. Compile, save, and then close the table.

Task 2: Customize the Seminar Card Page

High Level Steps

1. Add an action to create a new **Seminar Registration** from the **Seminar Card** page.
2. Add a new action as the first action in the **Seminar** group, to run the **Seminar Ledger Entries** page for the current **Seminar** record.
3. Add a new group and an action to run the **Seminar Registration List** page for the current **Seminar** record.

Detailed Steps

1. Add an action to create a new **Seminar Registration** from the **Seminar Card** page.
 - a. Design the page 123456700, **Seminar Card**.
 - b. Open the **Action Designer** for **Page Actions**.
 - c. To the list of actions, append an ActionContainer of the NewDocumentItems **SubType**. Make sure that it has no indentation, by clicking the **Left** button as many times as needed.
 - d. Under the NewDocumentItems ActionContainer, add a new action, and set the **Caption** property to "Seminar Registration".
 - e. Set the following properties on the action.

Property	Value
RunPageMode	Create
Image	NewTimesheet
Promoted	Yes
PromotedCategory	New
PromotedIsBig	Yes
RunObject	Page Seminar Registration
RunPageLink	Seminar No.=FIELD(No.)

2. Add a new action as the first action in the **Seminar** group, to run the **Seminar Ledger Entries** page for the current **Seminar** record.
 - a. Under the **Seminar** ActionGroup, and above the Comments action, insert a new action, and set the Caption property to "Ledger Entries".

Module 5: Feature Integration

- b. Set the following properties on the action.

Property	Value
Image	WarrantyLedger
PromotedCategory	Process
PromotedIsBig	Yes
ShortCutKey	Ctrl+F7
RunObject	Page Seminar Ledger Entries
RunPageLink	Seminar No.=FIELD(No.)

3. Add a new group and an action to run the **Seminar Registration List** page for the current **Seminar** record.
- Under the RelatedInformation ActionContainer, and above the NewDocumentItems ActionContainer, add a new ActionGroup, and set its caption to "&Registrations". Make sure that it is the last group in the RelatedInformation ActionContainer, and that it is indented one level under the ActionContainer.
 - To the **Registrations** group, add a new action, and set the **Caption** to "&Registrations". Indent it one level under the **Registrations** group.
 - On the **Registrations** action, set the following properties.

Property	Value
Image	Timesheet
PromotedCategory	Process
RunObject	Page Seminar Registration List
RunPageLink	Seminar No.=FIELD(No.)

- Compile, save, and then close the page.

Task 3: Customize the Seminar List Page

High Level Steps

- Make the same changes to the actions on the **Seminar List** page that you made to the **Seminar Card** page.

Detailed Steps

- Make the same changes to the actions on the **Seminar List** page that you made to the **Seminar Card** page.
 - Design page 123456700, **Seminar Card**.
 - Open the **Action Designer** for **Page Actions**.
 - Select all actions (press CTRL+A), and then copy them to the clipboard (press CTRL+C).

- d. Close **Action Designer** and **Page Designer** for the **Seminar Card** page.
- e. Design page 123456701, **Seminar List**.
- f. Open the **Action Designer** for **Page Actions**.
- g. Select and delete any existing actions (press CTRL+A, then press F4, and then confirm the deletion).
- h. Paste the actions from the clipboard (press CTRL+V).
- i. Compile, save, and then close the page.



Note: You can repeat the same steps you did for the **Seminar Card** page.

Navigate Integration

The Navigate feature lets users view a summary of the number and type of entries with the same document number or posting date. This feature is very useful for finding the ledger entries or other types of posted information that result from certain transactions. The Navigate feature is one of the central traceability features in Microsoft Dynamics NAV 2013, and one of its most versatile features.

When users post a transaction, they rely on the Navigate feature when they must trace the results of the transaction. Users access the Navigate feature from any page that displays any type of posted entries or documents. The Navigate feature displays every resulting entry.

This makes it important to fully integrate any custom functionality that includes a posting process with the Navigate feature. Integrating the Navigate feature improves the traceability of your own transactions and the resulting posted documents and ledger entries.

Navigate Feature Architecture

The architecture of this feature is simple, even though the Navigate feature performs the complex task of searching and then displaying database records to the user in the appropriate page. When you search, the Navigate feature takes advantage of simple filtering mechanisms. When it shows pages, it uses the default lookup forms.

The Navigate feature is completely contained within the page 344, **Navigate**.

Module 5: Feature Integration

The "Navigate Page (344)" figure shows the **Navigate** page.

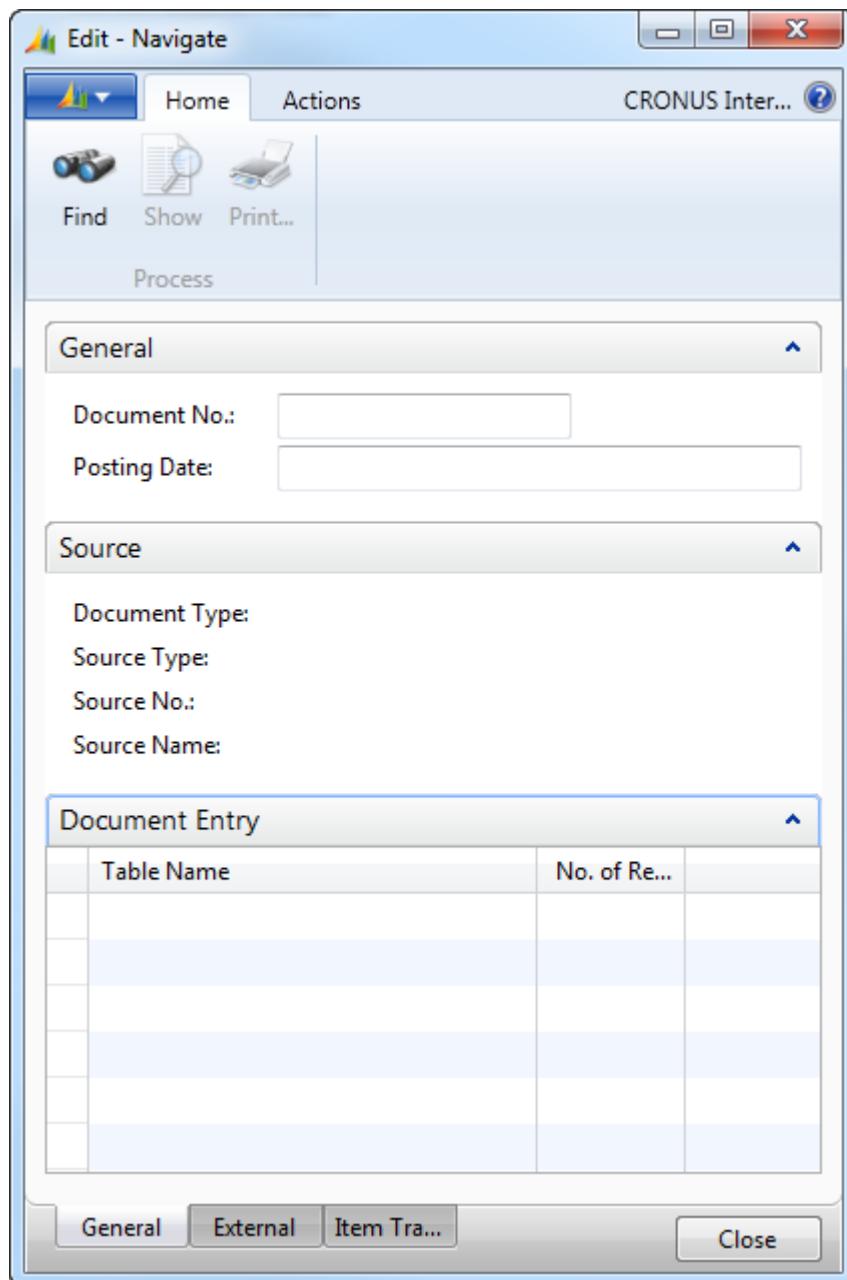


FIGURE 5.5: THE NAVIGATE PAGE (344)

The page uses table **265, Document Entry** as its source, and sets the property to **Yes**, to only work with temporary in-memory data.

The "The Document Entry Table (265)" figure shows the fields in the **Document Entry** table.

E..	Field No.	Field Name	Data Type	Length	Description
▶	1	Table ID	Integer		
✓	2	No. of Records	Integer		
✓	3	Document No.	Code	20	
✓	4	Posting Date	Date		
✓	5	Entry No.	Integer		
✓	6	Table Name	Text	100	
✓	7	No. of Records 2	Integer		
✓	8	Document Type	Option		
✓	9	Lot No. Filter	Code	20	
✓	10	Serial No. Filter	Code	20	

FIGURE 5.6: THE DOCUMENT ENTRY TABLE (265)

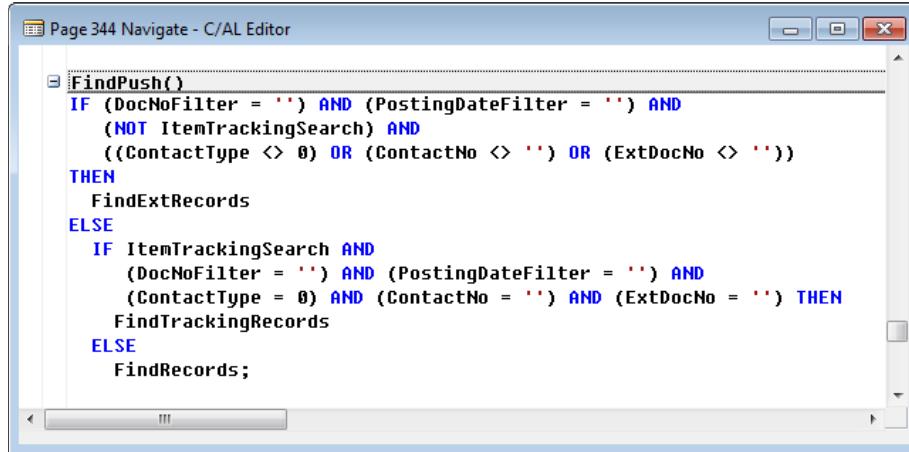
The following functions control the majority of the work in the **Navigate** feature:

- FindPush
- FindRecords and FindExtRecords
- InsertIntoDocEntry
- ShowRecords

FindPush

The most common way to use the **Navigate** page is to specify the **Document No.**, **Posting Date** or both, and then click **Find**. The field controls for the **Document No.** and Posting Date filters are bound to the DocNoFilter and PostingDateFilter variables. When the user clicks **Find**, it calls the **FindPush** function.

The following illustration shows the C/AL code in the FindPush function trigger.



```

Page 344 Navigate - C/AL Editor
[FindPush()]
IF (DocNoFilter = '') AND (PostingDateFilter = '') AND
    (NOT ItemTrackingSearch) AND
    ((ContactType <> 0) OR (ContactNo <> '') OR (ExtDocNo <> ''))
THEN
    FindExtRecords
ELSE
    IF ItemTrackingSearch AND
        (DocNoFilter = '') AND (PostingDateFilter = '') AND
        (ContactType = 0) AND (ContactNo = '') AND (ExtDocNo = '') THEN
        FindTrackingRecords
    ELSE
        FindRecords;

```

FIGURE 5.7: THE FINDPUSH FUNCTION TRIGGER

The **FindPush** function calls either the **FindExtRecords** or **FindRecords** function, depending on which filter fields the user populated with values.

Note: The **FindExtRecords** function looks for transactions that are based on the **External Document No.** field. This is the document number that is assigned by a third-party, such as a customer or a vendor. The **FindRecords** function looks for transactions that are based on the **Document No.** This is the document number that is assigned by Microsoft Dynamics NAV 2013, or by a Microsoft Dynamics NAV 2013 user. There is no basic difference in the C/AL code structure of either of these functions.

FindRecords

The **FindRecords** function follows this simple algorithm:

1. Empties the **Document Entry** temporary source table.
2. Repeats the following C/AL code pattern for each type of table.

The FindRecords Table Search Pattern

```

IF VendLedgEntry.READPERMISSION THEN BEGIN
    VendLedgEntry.RESET;
    VendLedgEntry.SETCURRENTKEY("Document No.");
    VendLedgEntry.SETFILTER("Document No.",DocNoFilter);
    VendLedgEntry.SETFILTER("Posting Date",PostingDateFilter);
    InsertIntoDocEntry();

```

```
DATABASE::"Vendor Ledger  
Entry",0,VendLedgEntry.TABLECAPTION,VendLedgEntry.COUNT);  
  
END;
```

- a. The READPERMISSION line checks whether the user has sufficient permissions to read from the table that is being searched. If this is the case, the search continues. Otherwise, it skips to the next table with the same pattern.
 - b. Resets the appropriate transaction (posted document or ledger entry) table.
 - c. Sets the appropriate table key to make the search easier.
 - d. Sets the filter to the **Document No.** and **Posting Date** fields to the values of the DocNoFilter and PostingDate variables. These variables are sources for the **Document No.** and **Posting Date** field controls in the page. The user enters the value into those variables directly.
 - e. Calls the **InsertIntoDocEntry** function by passing the table ID, document type (if relevant), the table caption, and the number of records in the filter.
3. Now all the necessary tables are searched. If any relevant posted document or ledger entry was found, the Rec variable is not empty. The following line of code checks for this condition.

```
DocExists := FINDFIRST;
```

4. If any records are found, then the source information is retrieved and shown in the page through several variables. Otherwise, the system informs the user that no records could be found.
5. Finally, the system updates the page.

Now the page shows the records it has found by using this algorithm.

InsertIntoDocEntry

The **InsertIntoDocEntry** function's task is to store information about any found records into the **Document Entry** temporary table.

The InsertIntoDocEntry function trigger contains the following C/AL code.

InsertIntoDocEntry Function

```
IF DocNoOfRecords = 0 THEN  
    EXIT;  
  
INIT;  
  
"Entry No." := "Entry No." + 1;  
  
"Table ID" := DocTableID;  
  
"Document Type" := DocType;  
  
"Table Name" := COPYSTR(DocTableName,1,MAXSTRLEN("Table Name"));  
  
"No. of Records" := DocNoOfRecords;  
  
INSERT;
```

The system first checks if there are any records. If there are none, the system exits.
The system then performs the following tasks:

1. Initializes a new record.
2. Assigns the information that is passed as parameters into the fields of the **Document Entry** table by using the implicit Rec variable.
3. Inserts the new record.

ShowRecords

The power of the **Navigate** feature is not only its capability to search for records in the database, but its ability to show the relevant page for each record type that it finds. The **ShowRecords** function controls that part of the functionality.

Users can call the **ShowRecords** function by either clicking **Show**, or drilling down any of the rows that represent the found record types.

The **ShowRecords** function is large, however, it follows the same simple pattern to display records. This pattern consists of one large CASE block. Based on the table ID of the selected row, the default lookup page runs over the same table that was filtered earlier in the **FindRecords** or **FindExtRecords** function. The user can quickly access the details of any posted transaction by knowing only its posting date or document number.

Calling Navigate from Other Pages

When using **Navigate**, the users don't have to memorize the document numbers for the transactions. Instead, they can call **Navigate** from any posted document or ledger entry pages, and then Navigate automatically filters by the **Document No.** and the **Posting Date** of the transaction.

To enable this functionality, the **Navigate** page includes the **SetDoc** function. Other pages can call this function to set the filters before running the **Navigate** page. Then, when the **Navigate** page runs, it first checks if there are any filters already set by other objects. If there are, it immediately finds records depending on the type of filters that are set by other objects.

Solution Design

Users can access the **Navigate** feature from all ledger entry pages, posted documents, and from their Role Center. **Navigate** finds all posted entries and documents in Microsoft Dynamics NAV 2013 that have the same **Document No.**, **Posting Date**, or both, as specified by the user. Extend the **Navigate** feature to also look for records in the **Seminar Ledger Entry** and **Posted Seminar Reg. Header** tables.

Seminar managers can use the **Navigate** feature to view a complete summary of the ledger entries that are related to a Posted Seminar Registration or a Seminar Ledger Entry. Therefore, you must add an action to access the **Navigate** feature to all Seminar Management posted information pages. These pages are as follows:

- Posted Seminar Registration
- Posted Seminar Reg. List
- Seminar Ledger Entries

Development

Your primary task is to enable the Navigate feature to search for posted seminar registration documents and seminar ledger entries. This requires changes to the Navigate page. This page is responsible for searching through the tables and displaying the search results. The changes include appending the code to the functions that are responsible for searching for records, and displaying appropriate pages for each record type that the Navigate feature can find.

Tables

To maximize the table search performance, you must also add a secondary key to the **Seminar Ledger Entry** table and **Seminar Ledger Entry** table. The **Navigate** page searches for records by the **Document No.** field and the Posting Date field. Therefore, you must always add a key that includes these two fields to any ledger entry table that the Navigate feature uses.

Pages

You must add an action to run the **Navigate** page from the **Seminar Ledger Entries** page and the **Posted Seminar Registration** page. Change the **Seminar Ledger Entries** and **Posted Seminar Registration** pages by adding the **Navigate** action to the Actions tab.

Lab 5.2: Changing Objects to Integrate with Navigate

Scenario

After integrating Seminar Management features with one another, Isaac is now ready to integrate these features with standard application functionality. The most important standard feature that he must integrate is Navigate. He must enable Navigate to search for Seminar Management transaction records, and make Navigate available in Seminar Management pages.

Exercise 1: Customize Tables

Exercise Scenario

To enable **Navigate** to search for Seminar Management transactions in the most efficient way, Isaac must add a new key to the **Seminar Ledger Entry** table.

Task 1: Modify the Seminar Ledger Entry Table

High Level Steps

1. Add the key with **Document No.** and **Posting Date** fields to the **Seminar Ledger Entry** table.

Detailed Steps

1. Add the key with **Document No.** and **Posting Date** fields to the **Seminar Ledger Entry** table.
 - a. Design table 123456732, **Seminar Ledger Entry**.
 - b. Open the **Keys** window for the table.
 - c. In the first empty line enter "Document No.,Posting Date"
 - d. Close the **Keys** window.
 - e. Compile, save, and then close the table.

Exercise 2: Customize the Navigate Page

Exercise Scenario

Isaac changes the **Navigate** page so that it can search for **Posted Seminar Reg. Header** and **Seminar Ledger Entry** records.

Task 1: Add Global Variables

High Level Steps

1. In the **Navigate** page, add global variables for the **Posted Seminar Reg. Header** and **Seminar Ledger Entry** tables.

Detailed Steps

1. In the **Navigate** page, add global variables for the **Posted Seminar Reg. Header** and **Seminar Ledger Entry** tables.
 - a. Design page 344, **Navigate**.
 - b. Open the **C/AL Globals** window.
 - c. Add the following variables to the end of the **Variables** list.

Name	DataType	Subtype
PostedSeminarRegHeader	Record	Posted Seminar Reg. Header
SeminarLedgEntry	Record	Seminar Ledger Entry

 **Best Practice:** When customizing standard Microsoft Dynamics NAV 2013 objects, it is a best practice to always visibly separate your variables from existing variables. You do this by adding a separator variable, with a discernible name, such as ---CSD1.00 Variables---. Add your variables after this separator. Because there is no other way to clearly mark your custom variables, this is the only way to make your variables stand out. This simplifies object maintenance.

Task 2: Customize the FindRecords Function

High Level Steps

1. Add code to the **FindRecords** function to look for **Posted Seminar Reg. Header** and records similar to the way that the function searches for other entry types. The function must first check whether the user has permission to read the **Posted Seminar Reg. Header** table. The function then set filters for the **No.** field and the **Posting Date** field to the *DocNoFilter* variable values and the *PostingDateFilter* variable value. The **FindRecords** function finally calls the **InsertIntoDocEntry** function to store the information about any found records.
2. Repeat the same search algorithm for the **Seminar Ledger Entry** table. Make sure that you utilize the appropriate table key.

Detailed Steps

1. Add code to the **FindRecords** function to look for **Posted Seminar Reg. Header** and records similar to the way that the function searches for other entry types. The function must first check whether the user has permission to read the **Posted Seminar Reg. Header** table. The function then set filters for the **No.** field and the **Posting Date** field to the *DocNoFilter* variable values and the *PostingDateFilter* variable value. The **FindRecords** function finally calls the **InsertIntoDocEntry** function to store the information about any found records.
 - a. Show the **C/AL Editor** window for the **Navigate** page.
 - b. Locate the **FindRecords** function trigger.
 - c. In the **FindRecords** function trigger, before the *DocExists := FINDFIRST* line, enter the following code.

```
//CSD1.00>

IF PostedSeminarRegHeader.READPERMISSION THEN BEGIN

    PostedSeminarRegHeader.RESET;

    PostedSeminarRegHeader.SETFILTER("No.",DocNoFilter);

    PostedSeminarRegHeader.SETFILTER("Posting Date",PostingDateFilter);

    InsertIntoDocEntry(
        DATABASE::"Posted Seminar Reg.
        Header",0,PostedSeminarRegHeader.TABLECAPTION,
        PostedSeminarRegHeader.COUNT);

END;

//CSD1.00<
```

2. Repeat the same search algorithm for the **Seminar Ledger Entry** table. Make sure that you utilize the appropriate table key.
 - a. In the **FindRecords** function trigger, enter the following code immediately after the code that you added in the previous step, but still within the comment block.

```

IF SeminarLedgEntry.READPERMISSION THEN BEGIN

    SeminarLedgEntry.RESET;

    SeminarLedgEntry.SETCURRENTKEY("Document No.", "Posting Date");

    SeminarLedgEntry.SETFILTER("Document No.", DocNoFilter);

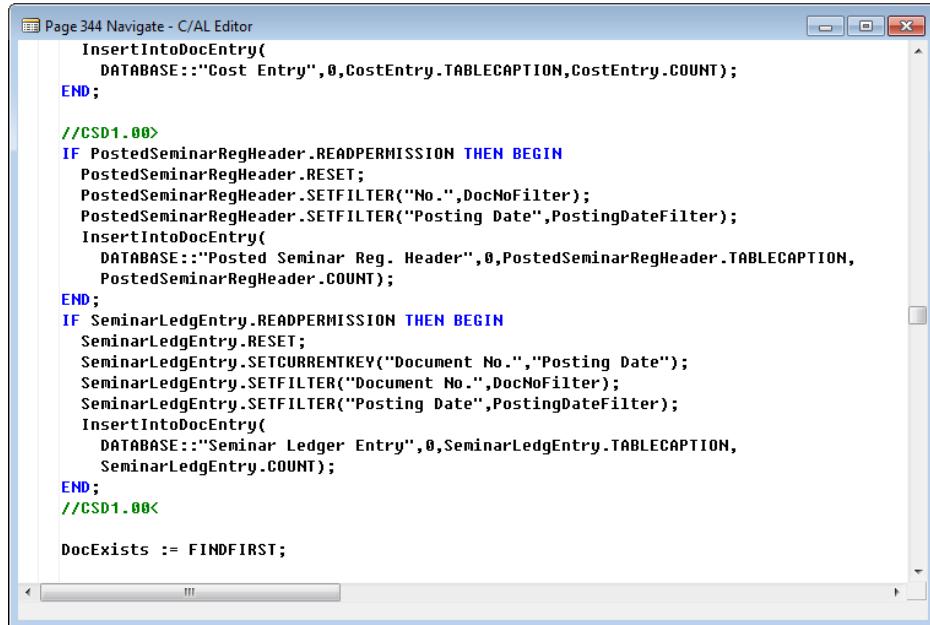
    SeminarLedgEntry.SETFILTER("Posting Date", PostingDateFilter);

    InsertIntoDocEntry(
        DATABASE::"Seminar Ledger Entry", 0, SeminarLedgEntry.TABLECAPTION,
        SeminarLedgEntry.COUNT);

END;

```

The “C/AL Logic for Finding the Posted Seminar Reg. Header and Seminar Ledger Entry Records” figure shows the completed code in the context of the **FindRecords** function trigger.



The screenshot shows the C/AL Editor window titled "Page 344 Navigate - C/AL Editor". The code is as follows:

```

Page 344 Navigate - C/AL Editor
InsertIntoDocEntry(
    DATABASE::"Cost Entry", 0, CostEntry.TABLECAPTION, CostEntry.COUNT);
END;

//CSD1.00
IF PostedSeminarRegHeader.READPERMISSION THEN BEGIN
    PostedSeminarRegHeader.RESET;
    PostedSeminarRegHeader.SETFILTER("No.", DocNoFilter);
    PostedSeminarRegHeader.SETFILTER("Posting Date", PostingDateFilter);
    InsertIntoDocEntry(
        DATABASE::"Posted Seminar Reg. Header", 0, PostedSeminarRegHeader.TABLECAPTION,
        PostedSeminarRegHeader.COUNT);
END;
IF SeminarLedgEntry.READPERMISSION THEN BEGIN
    SeminarLedgEntry.RESET;
    SeminarLedgEntry.SETCURRENTKEY("Document No.", "Posting Date");
    SeminarLedgEntry.SETFILTER("Document No.", DocNoFilter);
    SeminarLedgEntry.SETFILTER("Posting Date", PostingDateFilter);
    InsertIntoDocEntry(
        DATABASE::"Seminar Ledger Entry", 0, SeminarLedgEntry.TABLECAPTION,
        SeminarLedgEntry.COUNT);
END;
//CSD1.00K

DocExists := FINDFIRST;

```

FIGURE 5.8: C/AL LOGIC FOR FINDING THE POSTED SEMINAR REG. HEADER AND SEMINAR LEDGER ENTRY RECORDS

Task 3: Customize the ShowRecords Function

High Level Steps

1. Add code to the **ShowRecords** function to include two more case switches, for the **Posted Seminar Reg. Header** and **Seminar Ledger Entry** tables. For each case switch, run the lookup page for the appropriate record variable.

Detailed Steps

1. Add code to the **ShowRecords** function to include two more case switches, for the **Posted Seminar Reg. Header** and **Seminar Ledger Entry** tables. For each case switch, run the lookup page for the appropriate record variable.
 - a. Locate the **ShowRecords** function trigger.
 - b. In the **ShowRecords** function trigger, at the end of the CASE block, and immediately after the DATABASE::"Cost Entry": case switch, enter the following code:

```
//CSD1.00>

DATABASE::"Posted Seminar Reg. Header":

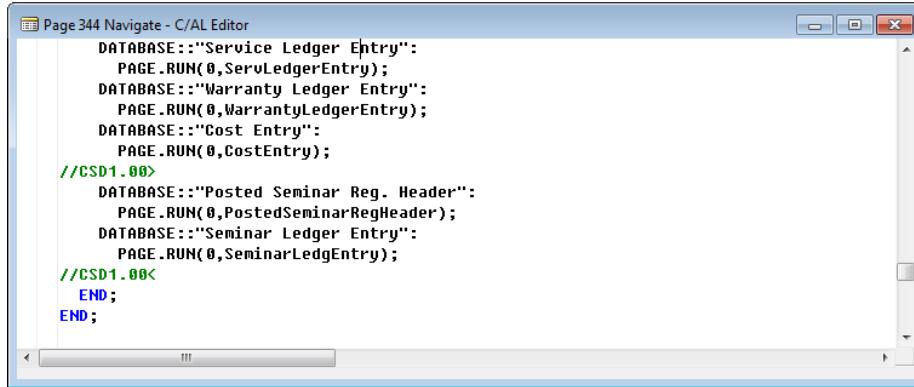
PAGE.RUN(0,PostedSeminarRegHeader);

DATABASE::"Seminar Ledger Entry":

PAGE.RUN(0,SeminarLedgEntry);

//CSD1.00<
```

The "ShowRecords Function Trigger Modification" figure shows the completed code in the context of the ShowRecords function trigger.



```

Page 344 Navigate - C/AL Editor
DATABASE::"Service Ledger Entry";
PAGE.RUN(0,ServLedgerEntry);
DATABASE::"Warranty Ledger Entry";
PAGE.RUN(0,WarrantyLedgerEntry);
DATABASE::"Cost Entry";
PAGE.RUN(0,CostEntry);
//CSD1.00>
DATABASE::"Posted Seminar Reg. Header";
PAGE.RUN(0,PostedSeminarRegHeader);
DATABASE::"Seminar Ledger Entry";
PAGE.RUN(0,SeminarLedgEntry);
//CSD1.00<
END;
END;

```

FIGURE 5.9: SHOWRECORDS FUNCTION TRIGGER MODIFICATION

 **Note:** The call to open the page passes the value 0 (zero) into the **PAGE.RUN** function. This causes the system to open the default lookup page for the table. Always make sure that the table that is included in the Navigate feature has the lookup page defined, or explicitly define the page to run.

- c. Compile, save, and then close the page.

Task 4: Define Default Lookup and Drilldown Pages for Tables

High Level Steps

1. Define the default lookup and drilldown pages for the **Posted Seminar Reg. Header** table.
2. Define the default lookup and drilldown pages for the **Seminar Ledger Entries** table.

Detailed Steps

1. Define the default lookup and drilldown pages for the **Posted Seminar Reg. Header** table.
 - a. Design table 123456718, **Posted Seminar Reg. Header**.
 - b. In the **Table – Properties** window, set the **LookupPageID** property to "Posted Seminar Registration".
 - c. Set the **DrillDownPageID** property to the same value.
 - d. Compile, save, and then close the table.
2. Define the default lookup and drilldown pages for the **Seminar Ledger Entries** table.
 - a. Design table 123456732, **Seminar Ledger Entry**.
 - b. In the **Table – Properties** window, set the **LookupPageID** property to "Seminar Ledger Entries".
 - c. Set the **DrillDownPageID** property to the same value.
 - d. Compile, save, and then close the table.

Exercise 3: Customize Pages

Exercise Scenario

Finally, Isaac changes the relevant Seminar Management pages, by adding the **Navigate** action to run the **Navigate** page to each of them.

Task 1: Customize the Posted Seminar Registration Page

High Level Steps

1. Add the **Navigate** action to the Actions group on the **Posted Seminar Registration** page.
2. Add code to the **Navigate** action **OnAction** trigger, to set the default document and posting date filters in the **Navigate** page, and run the **Navigate** page.

Detailed Steps

1. Add the **Navigate** action to the Actions group on the **Posted Seminar Registration** page.
 - a. Design page 123456734, **Posted Seminar Registration**.
 - b. Show the Action Designer for Page Actions.
 - c. Define the ActionItems ActionContainer, and make sure that you remove all indentation.
 - d. Under the ActionItems ActionContainer, add a new action, and set the Caption property to "&Navigate".
 - e. Set the following properties on the **Navigate** action.

Property	Value
Image	Navigate
Promoted	Yes
PromotedCategory	Process

2. Add code to the **Navigate** action **OnAction** trigger, to set the default document and posting date filters in the **Navigate** page, and run the **Navigate** page.
 - a. Define a global variable for page 344, **Navigate**, and name it **Navigate**.
 - b. In the OnAction trigger for the **Navigate** action, enter the following code.

```
Navigate.SetDoc("Posting Date","No.");
```

```
Navigate.RUN;
```

- c. Compile, save, and then close the page.



Note: Repeat the same steps for the **Posted Seminar Reg. List** page.

Task 2: Customize the Seminar Ledger Entries Page

High Level Steps

1. Add the **Navigate** action to the Actions group on the **Posted Seminar Registration** page.
2. Change code in the **Navigate** action **OnAction** trigger, to set the default document and posting date filters in the **Navigate** page. Run the **Navigate** page.

Detailed Steps

1. Add the **Navigate** action to the Actions group on the **Posted Seminar Registration** page.
 - a. Design page 123456734, **Posted Seminar Registration**.
 - b. Show the **Action Designer** for **Page Actions**.
 - c. Select the ActionItems ActionContainer and Navigate action, and copy them to clipboard (CTRL+C).
 - d. Close Action Designer, and then close the page.
 - e. Design page 123456721, **Seminar Ledger Entries**.
 - f. Show the **Action Designer** for **Page Actions**.
 - g. Paste the actions from the clipboard.
2. Change code in the **Navigate** action **OnAction** trigger, to set the default document and posting date filters in the **Navigate** page. Run the **Navigate** page.
 - a. Define a global variable for page 344, **Navigate**, and name it *Navigate*.
 - b. In the OnAction trigger for the **Navigate** action, replace the existing code with the following code.

```
    Navigate.SetDoc("Posting Date","Document No.");
```

```
    Navigate.RUN;
```

- c. Compile, save, and then close the page.

Module Review

Module Review and Takeaways

Seminar Management individual features are now working together. Users can easily move through pages by clicking actions. To do this, you added appropriate actions to Seminar Management pages that you created earlier.

You also extended the functionality of pages that display the posted transaction data for Seminar Management, by letting users call Navigate from those pages.

Finally, you extended the Navigate feature to search for posted seminar registration information and seminar ledger entries. Navigate is a central traceability feature Microsoft Dynamics NAV 2013. Users frequently depend on this feature to search for specific transactions, or analyze the results of a single transaction.

Test Your Knowledge

Test your knowledge with the following questions.

1. Which of the following actions fails if there is data in the table or a field?

- () Changing table name.
- () Changing the properties that control how data is displayed or formatted.
- () Changing TableRelation property of a field.
- () Change the type of a field from Code to Text.
- () Increase the length of a Text or a Code field.

2. Which conditions must be met if you want to change a normal field to a FlowField?

Module 5: Feature Integration

3. Which object contains the complete Navigate functionality?

- Page 344, Navigate
- Page 433, Navigate
- Codeunit 344, NavigateManagement
- Codeunit 433, Navigate
- Report 344, Navigate

4. Which functions in the Navigate page do you have to customize to enable searching and showing new documents or ledger entries?

Test Your Knowledge Solutions

Module Review and Takeaways

1. Which of the following actions fails if there is data in the table or a field?
 - () Changing table name.
 - () Changing the properties that control how data is displayed or formatted.
 - () Changing TableRelation property of a field.
 - (✓) Change the type of a field from Code to Text.
 - () Increase the length of a Text or a Code field.

2. Which conditions must be met if you want to change a normal field to a FlowField?

MODEL ANSWER:

The field must be of the data type compatible with the FlowField calculation type, and there must be no data in the field in any of the companies in the database.

3. Which object contains the complete Navigate functionality?

- (✓) Page 344, Navigate
- () Page 433, Navigate
- () Codeunit 344, NavigateManagement
- () Codeunit 433, Navigate
- () Report 344, Navigate

4. Which functions in the Navigate page do you have to customize to enable searching and showing new documents or ledger entries?

MODEL ANSWER:

FindRecords and ShowRecords

MODULE 6: REPORTING

Module Overview

The Seminar module to this point includes the following:

- Master tables and pages.
- A way to create new seminar registrations.
- Routines to post the registrations.

These features are integrated into the standard application so that you can access them from a new **Seminar Management** menu in the Departments area. The next step is to create reports for the module.

Objectives

The objectives are:

- Use report event triggers.
- Use special report functions.
- Create reports for the RoleTailored client.
- Create a seminar participant list.
- Create a ProcessingOnly report that posts invoices.

Prerequisite Knowledge

Before analyzing and implementing the report functionality that is covered in this module, we will review the following concepts :

- Report request pages
- Report triggers
- Report functions
- ProcessingOnly reports

Lesson Objectives

- Use report event triggers.
- Apply report functions.

Report Request Pages

In Microsoft Dynamics NAV 2013, a report is initialized with a **Request** Page. A request page runs before a report executes. Request pages enable end-users to specify options and filters for a report.

Report Triggers

Each report object consists of several elements that can contain the following triggers:

- The report itself
- One or more data items
- A **Request** page that has an optional FastTab for each data item and an optional **Options** FastTab
- Columns and Labels that display data

Each of these elements has a fixed number of event triggers that execute during report execution. You must understand the order that some frequently used triggers execute. The following list details the order in which these common event triggers execute:

1. When a user starts the report, the OnInitReport trigger is called. This is the first trigger that runs. It performs processing that is required before any part of the report can run. If the **Request** page is needed, the OnInitReport trigger runs before the **Request** page is displayed. Use the OnInitReport trigger to initialize variables and to populate default values on the **Request** page.

Module 6: Reporting

2. If the OnInitReport trigger does not end the processing of the report, the **Request** page for the report runs if it is defined. The user can decide to cancel the report from the **Request** page.
3. If the user continues, the OnPreReport trigger is called. At this point, no data is processed. Similar to the OnInitReport trigger, the OnPreReport trigger ends report processing. Use the OnPreReport trigger to process values that the user entered on the **Request** page.
4. As long as the processing of the report is not ended in the OnPreReport trigger, the data items are processed. Each data item has its own OnPreDataItem, OnAfterGetRecord, and OnPostDataItem triggers.
5. Before any records are retrieved, the OnPreDataItem trigger is called. In the same manner, the OnPostDataItem trigger is called after the last record is processed.
6. Between the OnPreDataItem trigger and the OnPostDataItem trigger, the records of the data item process. Processing a record means executing the OnAfterGetRecord trigger for each record that the data item retrieves, and outputting the values of the records by using the report's layout.
7. If there is an indented data item, a data item run is initiated for this data item and for each record in its parent data item. You can nest data items up to ten levels deep.
8. When all records are processed in a data item, control returns to the point from which the processing initiates. 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. After the first data item at indentation level zero processes, the next data item at indentation level zero (if one exists) processes in the same manner.

When there are no more data items, the OnPostReport trigger is called. Use this trigger to do any necessary post processing, for example, cleaning up by removing temporary files.

C/SIDE Solution Development in Microsoft Dynamics® NAV 2013

The "Report Trigger Execution Flow" figure shows the trigger execution flow.

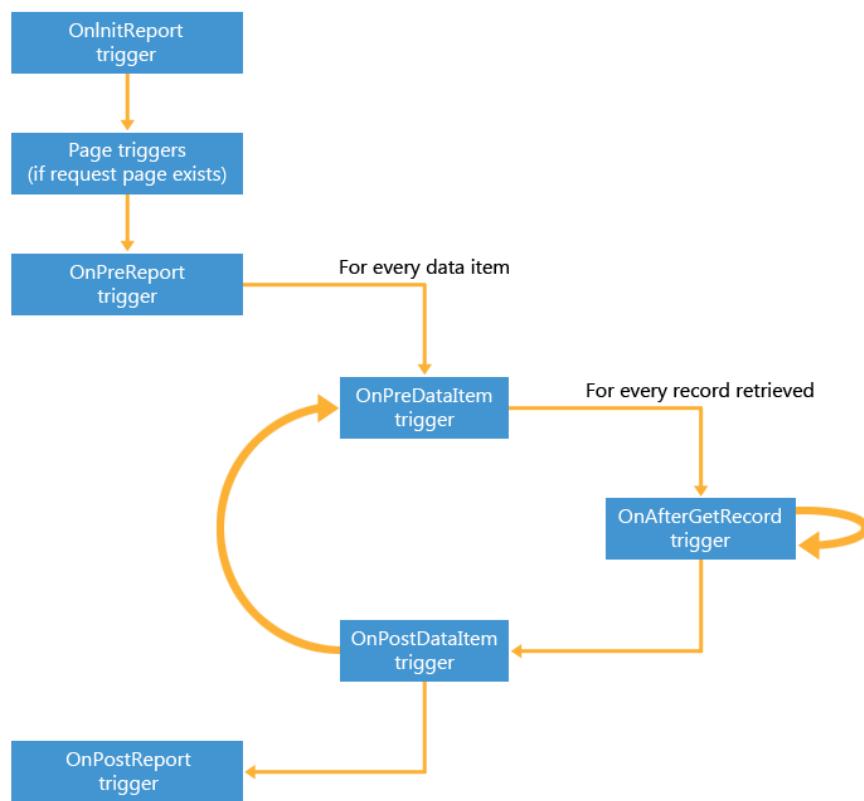


FIGURE 6.1: REPORT TRIGGER EXECUTION FLOW

Report Functions

You can use only certain functions in reports. Use these functions in complex reports. For a full listing of report functions, refer to the Microsoft Dynamics NAV 2013 Developer and IT Pro Help on the installation media or on MSDN. This developer Help file also contains useful walkthroughs to assist you in familiarizing yourself with Report Designer.

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. Therefore, use filters as much as you can.

A typical situation where you can use **SKIP** is to retrieve records from a related table by using values in the current record to form a filter. If the values in the current record already indicate that no records from the related table will be retrieved, you do not have to perform this processing and you can use **SKIP** to avoid the processing.

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

CurrReport.QUIT: This function skips the rest of the report; however, it is not an error. It is a typical ending for a report. When you use the **QUIT** function, the report exits without committing any changes that were made to the database during the execution. The OnPostReport trigger will not be called.

CurrReport.PREVIEW: Use this function to determine whether a report is printing in preview mode.

 **Note:** If you run a report in preview and the **CurrReport.PREVIEW** function is called, then the **Print** and **Save As** functionality is not available in the Microsoft Dynamics NAV 2013 client for Windows, Microsoft Dynamics NAV Web client, or in an application that runs on Microsoft Dynamics NAV Portal Framework for Microsoft SharePoint 2010.

This makes sure that any functionality that depends on the **CurrReport.PREVIEW** function being FALSE is called correctly when doing the actual print. If you run a client report definition (RDLC) report layout in preview mode and do not call the **CurrReport.PREVIEW** function, then you can print from the **Print Preview** window.

ProcessingOnly Report

A processing-only report is a report object that does not print, but only processes table data. Processing table data is not limited to processing-only reports. Reports that print can also change records. This section also applies to those reports. You can specify a report to be "Processing Only" by changing the ProcessingOnly property of the Report object. The report functions as it is supposed to (processing data items); however, it does not generate any printed output.

When the ProcessingOnly property is set, the **Request** page for the report changes slightly, as the **Print** and **Preview** buttons are replaced with an **OK** button. The **Cancel** and **Help** buttons remain unchanged.

C/SIDE Solution Development in Microsoft Dynamics® NAV 2013

As soon as the ProcessingOnly property is set, use the following guidelines to develop processing-only reports:

- Decide which tables are read. These are the data items.
- Design the report so that most of the code goes into the OnAfterGetRecord trigger.
- Use the **INSERT** or **MODIFY** functions in the tables, as appropriate.
- Use a dialog to show the user the progress and let the user cancel the report.

There are advantages in using a report instead of a codeunit to process data:

- The **Request** page functionality that lets the user select options and filters for data items is easily available in a report, but it is difficult to program in a codeunit.
- The Report Dataset Designer helps you visualize the program execution flow.
- Instead of writing code to open tables and to retrieve records, use report data items to provide a declarative way to access data.



Note: Processing Only reports have the advantage of built-in user interactivity by using the Request page designer. When the process requires user interactivity, choose a Processing Only report instead of a codeunit. Processing Only reports are also easier to maintain because of the way the data items visualize the flow of the code.

Reporting Lab Overview

The customer's functional requirements describe the need for the following reports and statistics:

- A list of the participants registered for a seminar.
- The total costs for each seminar, segregated into chargeable and nonchargeable costs to the customer.
- Statistics for different time periods, such as a month, last year, this year, or to the current date.

You can create two main reports to fulfill these requirements:

- A participant list
- A processing-only report that posts invoices

Use a lab in this module to implement each of these reports.

Lesson Objectives

Create reports for the RoleTailored client.

Participant List Reporting

A review of the client's specifications shows that reporting is required by the Seminar Management module. Begin the Analysis and Design of the needed reports.

Lesson Objectives

Create a seminar participant list.

Solution Analysis

The client's functional specifications require a **Participant List** report. This is a list of enrolled participants for a seminar. This report should be available from both the main **Seminar** menu and the **Seminar Registration** page.

Solution Design

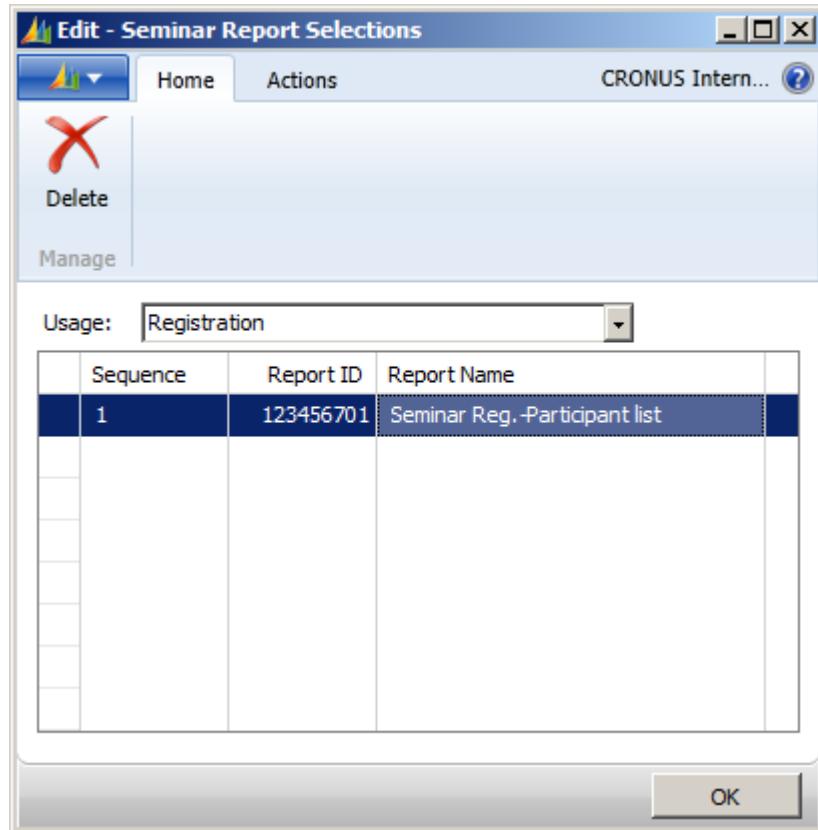
To implement this report, you must create the following items:

- The report itself
- The **Request** page to set the parameters of the report
- The controls to access the report from a page
- A page where you can select seminar reports

GUI Design

The **Seminar Report Selection** page displays the available seminar reports.

THE SEMINAR REGISTRATION PAGE (123456710)



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

Module 6: Reporting

Add the following menu item to the Setup Group:

Menu Type	Menu Name	Group	Comment
Item	Report Selections	Setup	Opens page 123456723 Seminar Report Selection

Change the **Seminar Registration** page by adding a promoted Print command to the **Actions** menu. This starts the **Participant List** report. See the "Seminar Registration Page" figure for an example.

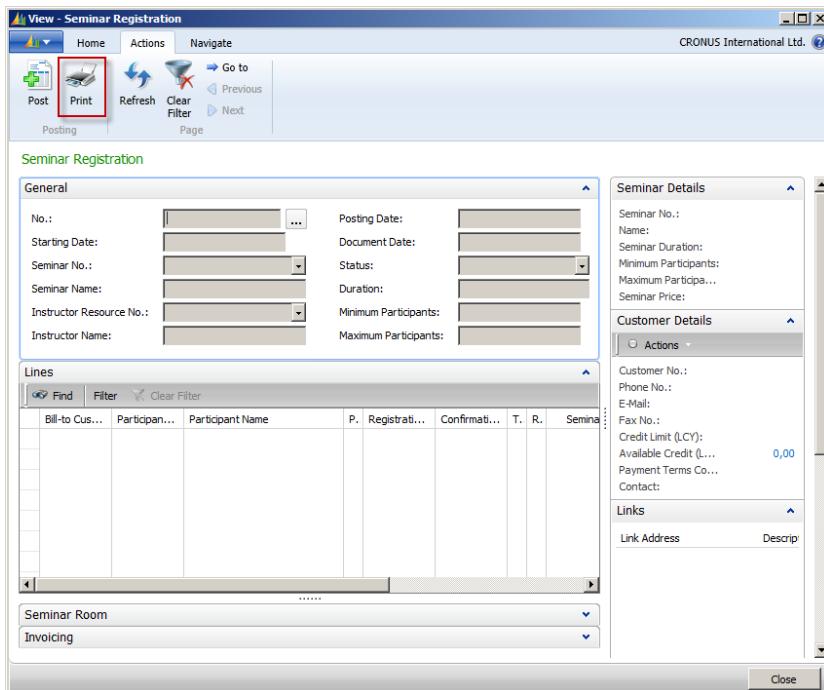


FIGURE 6.3: THE SEMINAR REGISTRATION PAGE (123456710)

Functional Design

The information for the report originates from the **Seminar Registration Header** table and the **Seminar Registration Line** table.

When users run this report, they select which **Seminar Registration Headers** to include. For each **Seminar Registration Header** table, the program then prints information from each corresponding **Seminar Registration Line** table.

Table Design

Implementation of the **Participant List** report requires the creation of one new table that is called the **Seminar Report Selections** table. You must also change the Seminar Registration Header table .

Lab 6.1: Creating the Seminar Participant List

Scenario

A **Participant List** report is required for the RoleTailored client, along with a way to select and run seminar reports. Create a **Participant List** report that lists all registrations by seminar. Use the typical Header/Detail list format and logic. Then create a way to selected seminar reports from a page.

Exercise 1: Part A: The Report Dataset

Task 1: Add a field to the Seminar Registration Header table

High Level Steps

1. Add a field to the **Seminar Registration Header** table (123456710).

Detailed Steps

1. Add a field to the **Seminar Registration Header** table (123456710).

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

Task 2: Create a codeunit to increment the No. Printed field just added to the Seminar Registration Header table.

High Level Steps

1. Create the Seminar Registration-Printed codeunit (123456702).

Detailed Steps

1. Create the Seminar Registration-Printed codeunit (123456702).
 - a. Set the property to specify the **Seminar Registration Header** table as the source table for this codeunit.
 - b. Enter code in the appropriate trigger so that when the program runs this codeunit, it does the following tasks:
 - Finds the **Seminar Registration Header** record.
 - Increases the No. Printed by 1.
 - Changes and commits the table.

```

FIND;
"No. Printed" := "No. Printed" + 1;

MODIFY;
COMMIT;
```

Task 3: Create the actual dataset in Report Dataset Designer.

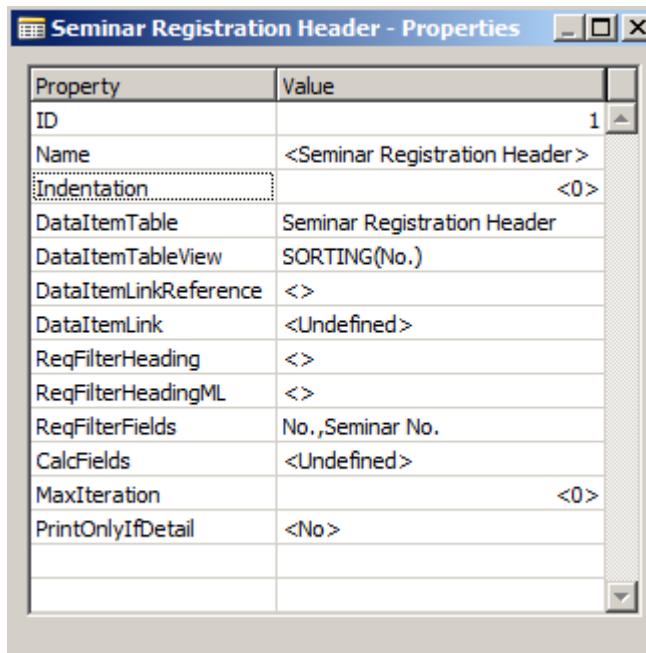
High Level Steps

1. Create a new report.
2. Add a data item for the **Seminar Registration Header** (123456710) table.
3. Add sorting and filtering to the Seminar Registration Header data item.
4. Add a data item for the **Seminar Registration Line** table.
5. Add the following fields to the Seminar Registration Line data item.
6. Use the property of the columns so that captions are available for the fields of both data items.
7. Set the property for the report so that the caption is Seminar Reg.- Participant List.
8. 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.
9. Add a Label to the Report.

Detailed Steps

1. Create a new report.
 - a. In the Object Designer, click New to create the Seminar Reg.- Participant List report (123456701).
2. Add a data item for the **Seminar Registration Header** (123456710) table.
 - a. Add the following fields to the DataItem:
 - No.
 - Seminar No.
 - Seminar Name
 - Starting Date
 - Duration
 - Instructor Name
 - Room Name

3. Add sorting and filtering to the Seminar Registration Header data item.
 - a. Set the properties for the Seminar Registration Header data item so that it is sorted by No., and the filter fields are **No.**, and **Seminar No.** as shown in the figure PROPERTIES FOR THE SEMINAR REGISTRATION HEADER DATA ITEM:



The screenshot shows a Windows-style dialog box titled "Seminar Registration Header - Properties". It contains a table with two columns: "Property" and "Value". The properties listed are: ID (Value: 1), Name (<Seminar Registration Header>), Indentation (<0>), DataItemTable (Seminar Registration Header), DataItemTableView (SORTING(No.)), DataItemLinkReference (<>), DataItemLink (<Undefined>), ReqFilterHeading (<>), ReqFilterHeadingML (<>), ReqFilterFields (No., Seminar No.), CalcFields (<Undefined>), MaxIteration (<0>), and PrintOnlyIfDetail (<No>). The dialog has standard window controls (minimize, maximize, close) at the top right.

Property	Value
ID	1
Name	<Seminar Registration Header>
Indentation	<0>
DataItemTable	Seminar Registration Header
DataItemTableView	SORTING(No.)
DataItemLinkReference	<>
DataItemLink	<Undefined>
ReqFilterHeading	<>
ReqFilterHeadingML	<>
ReqFilterFields	No., Seminar No.
CalcFields	<Undefined>
MaxIteration	<0>
PrintOnlyIfDetail	<No>

FIGURE 6.4: PROPERTIES FOR THE SEMINAR REGISTRATION HEADER DATA ITEM

4. Add a data item for the **Seminar Registration Line** table.
 - a. Set the property so that the data item indents to the first level.
 - b. Set the property so that the data item sorts by **Document No.** and **Line No..**
 - c. Set the DataItemLinkReference and the DataItemLink properties so that the data item links to the **Document No.** in the line data item is equal to the **No.** field in the header data item . as shown in the figure PROPERTIES FOR THE SEMINAR REGISTRATION LINE DATA ITEM:

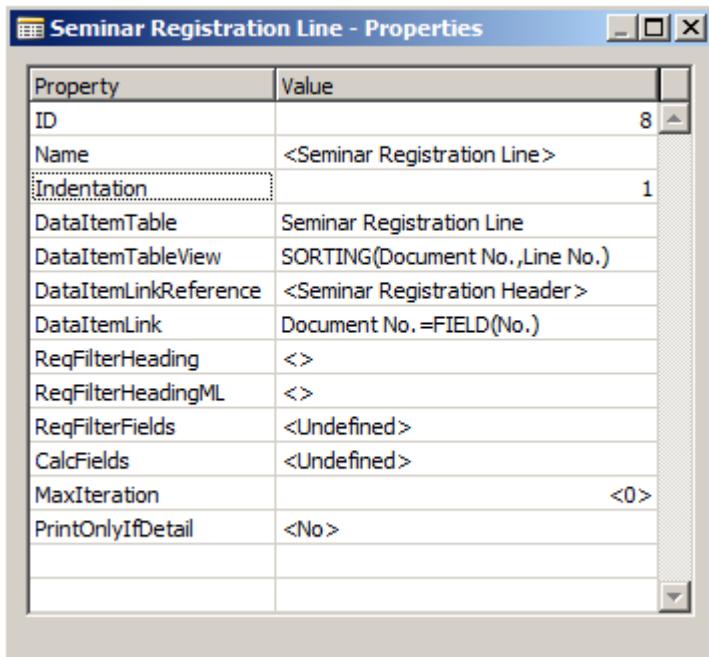


FIGURE 6.5: PROPERTIES FOR THE SEMINAR REGISTRATION LINE DATA ITEM

5. Add the following fields to the Seminar Registration Line data item.
 - a. Bill-to Customer No.
 - b. Participant Contact No.
 - c. Participant Name

6. Use the property of the columns so that captions are available for the fields of both data items.
 - a. Set the Includecaption property to True for all columns.

7. Set the property for the report so that the caption is Seminar Reg.- Participant List.
 - a. In the Report Dataset Designer goto the last row and click the down arrow.
 - b. Open the Properties window, the report properties are displayed.
 - c. Set the Caption property to Seminar Reg.- Participant List.

8. 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.
 - a. Select the Seminar Registration Header data item and click F9.
 - b. In the OnAfterGetRecord trigger type the following code:

```
CALCFIELDS("Instructor Name");
```

9. Add a Label to the Report.
 - a. Open the Report Label Designer on the **View** menu.
 - b. Add a label to the report with a caption of Seminar Registration Header:
 - i. Select View, Labels.
 - ii. In the Report Label Designer window on the first line type SeminarRegistrationHeader in the **Name** and in the **Caption**.
 - iii. Close the Report Label Designer window.
 - c. The Report Dataset now looks similar to the "Report Dataset Designer" figure.

The screenshot shows the 'Report Dataset Designer' window for a report titled 'Report 123456701 Seminar Reg.- Participant list'. The window displays a grid of data items. The columns are labeled 'E.', 'Data Type', 'Data Source', and 'Name'. There are two main sections under 'Data Type': 'DataItem' and 'DataItem'. Under the first 'DataItem' section, there are seven columns corresponding to fields from the 'Seminar Registration Header' dataset. Under the second 'DataItem' section, there are three columns corresponding to fields from the 'Seminar Registration Line' dataset. All columns have a checked checkbox in the last column.

E.	Data Type	Data Source	Name	I..
	▶ DataItem	Seminar Registration Header	<Seminar Registration Header>	
	Column	"Seminar Registration Header". "No."	No_SeminarRegistrationHeader	✓
	Column	"Seminar Registration Header". "Starting Date"	StartingDate_SeminarRegistrationHeader	✓
	Column	"Seminar Registration Header". "Seminar No."	SeminarNo_SeminarRegistrationHeader	✓
	Column	"Seminar Registration Header". "Seminar Name"	SeminarName_SeminarRegistrationHeader	✓
	Column	"Seminar Registration Header". "Instructor Name"	InstructorName_SeminarRegistrationHeader	✓
	Column	"Seminar Registration Header". "Duration"	Duration_SeminarRegistrationHeader	✓
	Column	"Seminar Registration Header". "Room Name"	RoomName_SeminarRegistrationHeader	✓
	▶ DataItem	Seminar Registration Line	<Seminar Registration Line>	
	Column	"Seminar Registration Line". "Bill-to Customer No."	BilltoCustomerNo_SeminarRegistrationLine	✓
	Column	"Seminar Registration Line". "Participant Contact..."	ParticipantContactNo_SeminarRegistrationLine	✓
	Column	"Seminar Registration Line". "Participant Name"	ParticipantName_SeminarRegistrationLine	✓

FIGURE 6.6: REPORT DATASET DESIGNER

Exercise 2: Part B: The Report Layout

Exercise Scenario

To render a report from inside the RoleTailored client, you must create a Visual Studio client report definition (RDLC) report layout.

Task 1: Use Visual Studio Report Designer to create a layout for the report.

High Level Steps

1. Open Visual Studio Report Designer.
2. Add a table to the body of the report.
3. Add fields to the table.
4. Add group header rows to the table.
5. Add fields to the group header.
6. Format the table.

Detailed Steps

1. Open Visual Studio Report Designer.
 - a. Select **View > Layout** on the Tools menu. This creates an empty RDLC layout and opens it in Visual Studio Report Designer.
 2. Add a table to the body of the report.
 - a. Select **View > Toolbox** in Visual Studio Report Designer if the toolbox is not visible by default.
 - b. In the toolbox, select the **Table** control and drag it onto the body of the report.
 - c. Position the table in the upper-left corner in the body of the report.
 3. Add fields to the table.
 - a. Select **View > Report Data** to show the report data window.
 - b. In the report data window, open the Parameters folder.
 - c. From the Parameters folder, select the field **[@BilltoCustomerNo_SeminarRegistrationLineCaption]**, and then drag it onto the table on the text box on the first row, first column (in the Header Row).
 - d. Repeat this for the following fields:
 - **[@ParticipantContactNo_SeminarRegistrationLineCaption]**
 - **[@ParticipantName_SeminarRegistrationLineCaption]**

The Header row in the table now contains the caption fields for the fields from the **Seminar Registration Line** table.
 - e. In the second row of the table, in the first text box, right-click the text box, and then select the following field from the shortcut menu: **[BilltoCustomerNo_SeminarRegistrationLine]**
 - f. Repeat this for the following fields:
 - **[ParticipantContactNo_SeminarRegistrationLine]**
 - **[ParticipantName_SeminarRegistrationLine]**
4. Add group header rows to the table.
 - a. In the **Row Groups** window at the bottom of the screen, select **Details > Add Group > Parent Group**.
 - b. In the Tablix group window, enter **[No_SeminarRegistrationHeader]** in the **Group By** field.
 - c. Delete the first column of the table.
 - d. Right-click the detail row of the table, and then select **Insert Row, Outside Group, Above**.

Module 6: Reporting

- e. Repeat this step until there are seven empty group header rows in the table.
- f. In the first row of the table, select the three text boxes.
- g. Right-click, and then click **Cut**.
- h. On the first text box on the header row just above the Details row, right-click, and then click **Paste**.

The table should look like the “Table with Group Header Rows” figure.

	[@BilltoCustomer]	[@ParticipantC]	[@ParticipantN:]
≡	BilltoCustomer	ParticipantCon	ParticipantNam

FIGURE 6.7: TABLE WITH GROUP HEADER ROWS

5. Add fields to the group header.
 - a. In the first column of the table, in the empty group header rows, add the following fields:
 - i. =Parameters!No_SeminarRegistrationHeaderCaption.Value
 - ii. =Parameters!SeminarNo_SeminarRegistrationHeaderCaption.Value
 - iii. =Parameters!SeminarName_SeminarRegistrationHeaderCaption.Value
 - iv. =Parameters!StartingDate_SeminarRegistrationHeaderCaption.Value
 - v. =Parameters!Duration_SeminarRegistrationHeaderCaption.Value
 - vi. =Parameters!InstructorName_SeminarRegistrationHeaderCaption.Value
 - vii. =Parameters!RoomName_SeminarRegistrationHeaderCaption.Value

- b. In the second column of the table, in the empty group header rows add the following fields:
 - i. =Fields!No_SeminarRegistrationHeader.Value
 - ii. =Fields!SeminarNo_SeminarRegistrationHeader.Value
 - iii. =Fields!SeminarName_SeminarRegistrationHeader.Value
 - iv. =Fields!StartingDate_SeminarRegistrationHeader.Value
 - v. =Fields!Duration_SeminarRegistrationHeader.Value
 - vi. =Fields!InstructorName_SeminarRegistrationHeader.Value
 - vii. =Fields!RoomName_SeminarRegistrationHeader.Value
6. Format the table.
 - a. Increase the Width of the columns in the table.
 - b. Change the Color property of the line captions to Blue.
 - c. Change the FontSize property of the text boxes in the table to 8pt.

Task 2: Add a Page Header

High Level Steps

1. Add a page header to the report.
2. Format the title.

Detailed Steps

1. Add a page header to the report.
 - a. In the Page Header, add four textboxes: one on the left side and three on the right side.
 - b. In the text box on the left side of the page header, enter the following value expressions:

```
=Parameters!SeminarRegistrationHeader.Value
```

- c. In the three text boxes on the right side, add the following expressions:

```
=Globals!ExecutionTime
```

```
=Globals!PageNumber & " of " & Globals!TotalPages
```

```
=User!UserID
```

2. Format the title.
 - a. In the text box on the Page Header that contains the title, set the Color to blue, and then make the text Bold.

Task 3: Add the Company Name to the Report

High Level Steps

1. To add the Company Name to the report, the dataset must be updated.
2. Add an integer data item to the bottom of the data item designer.
3. Add a Global Variable.
4. Add the Company Name as a column to the Integer DataItem.
5. Add the Company Name to the Page Header.

Detailed Steps

1. To add the Company Name to the report, the dataset must be updated. Add an integer data item to the bottom of the data item designer.
 - a. Make sure that the Integer data item is not indented.
 - b. Filter the Integer data item so that it only displays one record.
 - i. In the DataItemTableView propertie of the Integer data item enter the following Value:

```
WHERE(Number=CONST(1))
```

2. Add a Global Variable.
 - a. In the menu, select **View > Globals**, and then add a variable of type record from the **Company Information** (79) table.
3. Add the Company Name as a column to the Integer DataItem.
 - a. Change the name of the column to CompanyInformation_Name. See the "Report Dataset with Company Name" figure.

Report 123456701 Seminar Reg.-Participant list - Report Dataset Designer			
E.	Data Type	Data Source	Name
	DataItem	Seminar Registration Header	<Seminar Registration Header>
	Column	"Seminar Registration Header".No."	No_SeminarRegistrationHeader
	Column	"Seminar Registration Header"."Starting Date"	StartingDate_SeminarRegistrationHeader
	Column	"Seminar Registration Header"."Seminar No."	SeminarNo_SeminarRegistrationHeader
	Column	"Seminar Registration Header"."Seminar Name"	SeminarName_SeminarRegistrationHeader
	Column	"Seminar Registration Header"."Instructor Name"	InstructorName_SeminarRegistrationHeader
	Column	"Seminar Registration Header".Duration	Duration_SeminarRegistrationHeader
	Column	"Seminar Registration Header"."Room Name"	RoomName_SeminarRegistrationHeader
	DataItem	Seminar Registration Line	<Seminar Registration Line>
	Column	"Seminar Registration Line".Bill-to Customer No."	BilltoCustomerNo_SeminarRegistrationLine
	Column	"Seminar Registration Line"."Participant Contact..."	ParticipantContactNo_SeminarRegistrationLine
	Column	"Seminar Registration Line"."Participant Name"	ParticipantName_SeminarRegistrationLine
	DataItem	Integer	<Integer>
▶	Column	"Company Information".Name	CompanyInformation_Name

FIGURE 6.8: REPORT DATASET WITH COMPANY NAME

4. Add the Company Name to the Page Header.
 - a. Add a text box in the Page Header to display the Company Name. Use the following expression:

```
=Last(Fields!CompanyInformation_Name.Value, "DataSet_Result")
```

Task 4: Filter the table

High Level Steps

1. Because of the new integer data item the dataset has changed. It now has one additional row that contains the company name. Filter out this row in the table.

Detailed Steps

1. Because of the new integer data item the dataset has changed. It now has one additional row that contains the company name. Filter out this row in the table.
 - a. Add a Filter to the Group in the Tablix to make sure that only the following lines are allowed:

```
=Len(Fields!No_SeminarRegistrationHeader.Value) > 0
```

Task 5: Run the Report

High Level Steps

1. Save and Run the Report.

Detailed Steps

1. Save and Run the Report.
 - a. Save the report layout and close Visual Studio.
 - b. Save the report in the NAV Report Designer, and then accept the changed layout. See the "Report 123456701 Rendered in RoleTailored client" figure for an example.

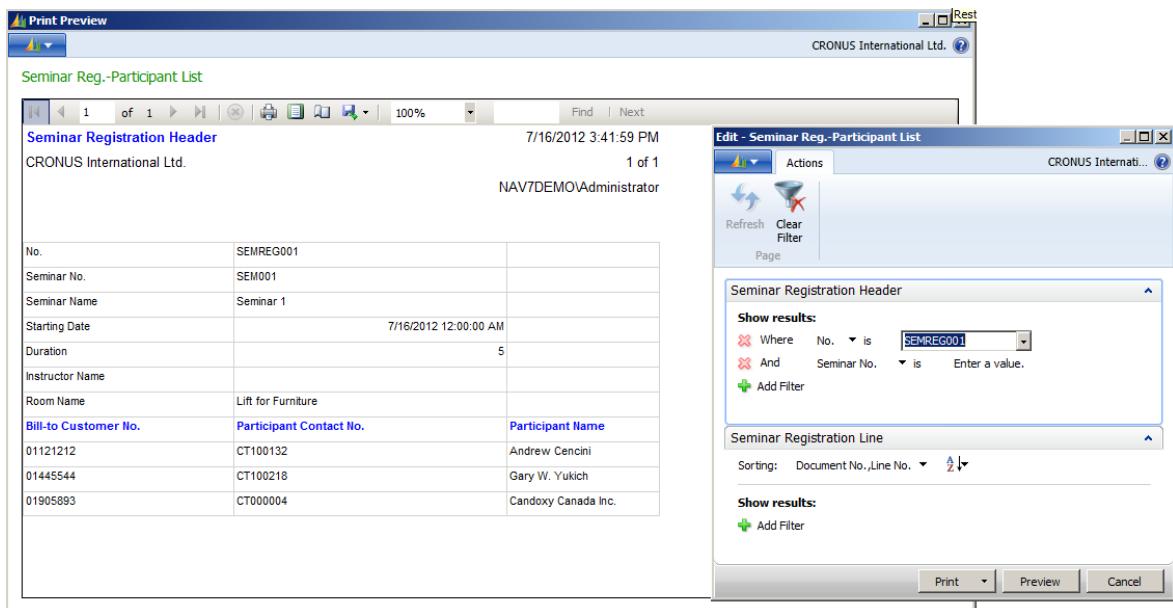


FIGURE 6.9: REPORT 123456701 RENDERED IN ROLETAILORED CLIENT

Exercise 3: Part C: Report Selections Table and Page

Exercise Scenario

Import the report selection objects.

Task 1: Import the Seminar Report Selections Table and Page

High Level Steps

1. Import the starter object.

Detailed Steps

1. Import the starter object.
 - a. In Object Designer, click **File > Import**.
 - b. Browse to the file Mod06\Labfiles\Lab 6.B - Starter - Report Selection.fob.

- c. Click **Open**.
- d. Click **Yes**, and then click **OK** to complete the import.

Exercise 4: Part D: Testing

Task 1: Test Report Selections

High Level Steps

1. Test the application.

Detailed Steps

1. Test the application.
 - a. If there are no Seminar Registration records, create one by selecting **Seminars > Order Processing > Registrations** from the main menu. Complete all fields in the General and Seminar Rooms FastTabs and add at least two participants.
 - b. Use the **Seminar Report Selection** window to set up the Seminar Reg.-Participant List as the report to run for Registration. To do this, 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.
 - c. Open the **Seminar Registration** page again, and then view the record prepared in Step 1. Click **Print**.
 - d. The **Request** page should open with the No. set to the current registration. Select **Print** or **Preview** to print and check the report.
 - e. Try to print the report by using different parameters and different registration data to verify that it works correctly.

Invoice Posting Batch Job

When a seminar concludes, customers are invoiced for their registered participants. These invoices are implemented as Batch Jobs. Batch jobs should allow for multiple participants and can include project and resource information.

Lesson Objectives

Create a Processing Only report that posts invoices.

Solution Analysis

The process to create invoices is implemented as a ProcessingOnly Report. Use this type of report object to process and modify data without rendering any report for display.

This ProcessingOnly Report creates an invoice for each customer in a filter that is specified by the user. The report has 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 creating the invoices, or creating and posting them at the same time.

This type of report is known in Microsoft Dynamics NAV as a batch job. A batch job in Microsoft Dynamics NAV is a routine that processes data in batches, such as the Adjust Exchange Rates Batch Job, or Batch Posting Sales Orders, Invoices, and Credit Memos.

If you set up numerous orders, invoices, or credit memos, you can post them with a batch job. You can post them at night or at another convenient time. Batch posting invoices and credit memos functions exactly like orders.

Solution Design

Use the **Seminar Ledger Entries** to access all charges to invoice for posted seminars. The report creates Sales Headers and Sales Lines as appropriate. If the user selected the option to also post the invoices, the report runs the posting process for sales invoices.

Lab 6.2: Creating the Invoice Posting Batch Job

Scenario

As soon as a seminar concludes, the registrations must be posted in preparation for billing. Create a ProcessingOnly report to generate Microsoft Dynamics NAV standard format invoices that are ready for billing and posting.

Exercise 1: Create the Invoice Posting Batch Job

Exercise Scenario

This report differs from the previous two reports because it is not a print report, but is a ProcessingOnly report.

Task 1: Import the Report

High Level Steps

1. Import the report object.

Detailed Steps

1. Import the report object.
 - a. In the development environment, open the Object Designer using **Tools > Object Designer**.
 - b. Locate the Mod06\LabfilesLab 6.B - Starter.fob.
 - c. Select **File > Import** to import the report.

Task 2: Complete the C/AL code of the imported report

High Level Steps

1. In Object Designer, locate report **Create Seminar Invoices** (123456700), and then complete the C/AL code according to these guidelines.
2. Verify that these global variables are defined for the report.
3. Verify that these text constants are defined.
4. Enter code in the FinalizeSalesInvHeader function trigger so that the function performs these tasks.
5. Enter code into the InsertSalesInvHeader function trigger so that the trigger performs these tasks.
6. Enter code into the appropriate trigger so that just before the data item is run, the program performs these tasks.

Module 6: Reporting

7. Enter code in the appropriate trigger so that after the program gets a record for the DataItem, the program performs these tasks.
8. Enter code in the appropriate trigger so that when the program posts the DataItem, it performs these tasks.
9. Access the Request Page Designer by clicking **View > Request Page**.

Detailed Steps

1. In Object Designer, locate report **Create Seminar Invoices** (123456700), and then complete the C/AL code according to these guidelines.
 - a. Verify the property to specify that this report is for processing-only.
 - b. Verify a data item for the **Seminar Ledger Entry** table and set the properties for the data item as follows:
 - i. Specify the Key in DataItemTableView property of the data item to sort by the following fields:
 - **Bill-to Customer No.**
 - **Document No.**
 - **Charge Type**
 - **Participant Contact No.**
 - ii. Specify that the user can filter on the **Bill-to Customer No., Seminar No., and Posting Date**.
 - iii. Specify the variable name of the data item as SeminarLedgerEntry.

2. Verify that these global variables are defined 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	
Customer	Record	Customer	
CurrencyExchRate	Record	Currency Exchange Rate	
SalesCalcDiscount	Codeunit	Sales-Calc. Discount	
SalesPost	Codeunit	Sales-Post	

Name	Data Type	Subtype	Length
Window	Dialog		
PostingDateReq	Date		
DocDateReq	Date		
CalclnvoiceDiscount	Boolean		
PostInvoices	Boolean		
NextLineNo	Integer		
NoOfSalesInvErrors	Integer		
NoOfSalesInv	Integer		

3. Verify that these text constants are defined.

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 are posted. A total of %1 invoices are not posted.
Text007	There is nothing to invoice.

4. Enter code in the FinalizeSalesInvHeaderfunction trigger so that the function performs these tasks.
- If the CalclnvoiceDisc variable is TRUE (which means the user selected this option), the function runs the Sales-Calc. Discount codeunit with the Sales Line record and finds the Sales Header record.
 - The function then performs a commit.
 - The function clears the SalesCalcDiscount and SalesPost variables and increments the NoOfSalesInvoice by one.
 - If the PostInvoice variable is TRUE (which means the user selected this option), 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.

Code Example

```
WITH SalesHeader DO BEGIN  
    IF CalcInvoiceDiscount THEN  
        SalesCalcDiscount.RUN(SalesLine);  
        GET("Document Type","No.");  
        COMMIT;  
        CLEAR(SalesCalcDiscount);  
        CLEAR(SalesPost);  
        NoOfSalesInv := NoOfSalesInv + 1;  
        IF PostInvoices THEN BEGIN  
            CLEAR(SalesPost);  
            IF NOT SalesPost.RUN(SalesHeader) THEN  
                NoOfSalesInvErrors := NoOfSalesInvErrors + 1;  
            END;  
        END;  
    END;
```

5. Enter code into the InsertSalesInvHeader function trigger so that the trigger performs these tasks.
 - a. Initializes a new Sales Header record with a Document Type of Invoice and a blank No. and inserts it into the database.
 - b. Validates that the Sell-To Customer No. of the new record equals the Bill-to Customer No. of the ledger entry.
 - c. If the Bill-to Customer No. value differs from that of the Sell-To Customer No., the program validates that the Bill-to Customer No. of the new record equals the Bill-to Customer No. of the ledger entry.
 - d. Validates that the Posting Date of the new record is the PostingDateReq.
 - e. Validates that the Document Date of the new record is the DocDateReq.
 - f. Validates that the Currency Code of the new record is blank.
 - g. Changes and commits the new record.
 - h. Sets the NextLineNo variable to 10000.

Code Example

```
WITH SalesHeader DO BEGIN  
  
    INIT;  
  
    "Document Type" := "Document Type"::Invoice;  
  
    "No." := "";  
  
    INSERT(TRUE);  
  
    VALIDATE("Sell-to Customer No.", "Seminar Ledger Entry"."Bill-to Customer No.");  
  
    IF "Bill-to Customer No." <> "Sell-to Customer No." THEN  
  
        VALIDATE("Bill-to Customer No.", "Seminar Ledger Entry"."Bill-to Customer No.");  
  
    VALIDATE("Posting Date", PostingDateReq);  
  
    VALIDATE("Document Date", DocDateReq);  
  
    VALIDATE("Currency Code", "");  
  
    MODIFY;  
  
    COMMIT;  
  
    NextLineNo := 10000;  
  
END;
```

6. Enter code into the appropriate trigger so that just before the data item is run, the program performs these tasks.
 - a. In the OnPreDataItem trigger enter code that:
 - i. Shows an error if the PostingDateReq or DocDateReq is empty by using the text constants Text000 and Text001.
 - ii. Opens a dialog window with the text constants Text002, Text003, and Text004.

Code Example

```
IF PostingDateReq = 0D THEN  
  
    ERROR(Text000);  
  
IF DocDateReq = 0D THEN
```

```

ERROR(Text001);

Window.OPEN(
    Text002 +
    Text003 +
    Text004);

```

7. Enter code in the appropriate trigger so that after the program gets a record for the DataItem, the program performs these tasks.
 - a. In the OnAfterGetRecord trigger enter code that:
 - i. If the No. of the Customer record differs from the Bill-to Customer No., the program gets the Customer record that corresponds to the Bill-to Customer No.
 - ii. If the Customer record is blocked for All or Invoice, the NoOfSalesInvErrors increments by 1 (one).
 - iii. If the Customer is not blocked for All or is only blocked for Invoice, the program proceeds with the following tasks:
 - If the Bill-to Customer No. differs 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 **FinalizeSalesInvoiceHeader** function.
 - iv. The program then runs the **InsertSalesInvoiceHeader** function and sets the **Document Type** field and the **Document No.** field on the Sales Line to be those of the Sales Header.
 - v. Updates the dialog window with the Sales Registration No.
 - vi. Sets the No. of the Sales Line depending on whether the Charge Type that is being posted is for an instructor, room, or participant.
 - vii. Completes the following 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**

- Unit Price from the Seminar
 - If the **Currency Code** on the **Sales Header** is not blank, the program test 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.
- viii. Completed the **Sales Line** fields as follows: Quantity from the Quantity of the Seminar Ledger Entry.
- ix. Inserts the Sales Line record.
- x. Increments the **NextLineNo** by **10000**.

Code Example

```
IF "Bill-to Customer No." <> Customer."No." THEN  
  
    Customer.GET("Bill-to Customer No.");  
  
    IF Customer.Blocked IN [Customer.Blocked::All,Customer.Blocked::Invoice] THEN  
        BEGIN  
  
            NoOfSalesInvErrors := NoOfSalesInvErrors + 1;  
  
        END ELSE BEGIN  
  
            IF "Seminar Ledger Entry"."Bill-to Customer No." <> SalesHeader."Bill-to Customer No." THEN BEGIN  
  
                Window.UPDATE(1,"Bill-to Customer No.");  
  
                IF SalesHeader."No." <> "" THEN  
  
                    FinalizeSalesInvoiceHeader;  
  
                    InsertSalesInvoiceHeader;  
  
                END;  
  
                Window.UPDATE(2,"Seminar Registration No.");  
  
                CASE Type OF  
  
                    Type::Resource:  
  
                    BEGIN  
  
                        SalesLine.Type := SalesLine.Type::Resource;  
  
                    CASE "Charge Type" OF
```

```
"Charge Type"::Instructor:  
  
    SalesLine."No." := "Instructor Resource No.;"  
  
"Charge Type"::Room:  
  
    SalesLine."No." := "Room Resource No.;"  
  
"Charge Type"::Participant:  
  
    SalesLine."No." := "Instructor Resource No.;"  
  
    END;  
  
    END;  
  
    END;  
  
SalesLine."Document Type" := SalesHeader."Document Type";  
  
SalesLine."Document No." := SalesHeader."No.";  
  
SalesLine."Line No." := NextLineNo;  
  
SalesLine.VALIDATE("No.");  
  
Seminar.GET("Seminar No.");  
  
IF "Seminar Ledger Entry".Description <> "" THEN  
  
    SalesLine.Description := "Seminar Ledger Entry".Description  
  
ELSE  
  
    SalesLine.Description := Seminar.Name;  
  
SalesLine."Unit Price" := "Unit Price";  
  
IF SalesHeader."Currency Code" <> "" THEN BEGIN  
  
    SalesHeader.TESTFIELD("Currency Factor");  
  
    SalesLine."Unit Price" :=  
  
    ROUND(  
  
        CurrencyExchRate.ExchangeAmtLCYToFCY(  
  
        WORKDATE,SalesHeader."Currency Code",
```

```
SalesLine."Unit Price",SalesHeader."Currency Factor"));

END;

SalesLine.VALIDATE(Quantity,Quantity);

SalesLine.INSERT;

NextLineNo := NextLineNo + 10000;

END;
```

8. Enter code in the appropriate trigger so that when the program posts the DataItem, it performs these tasks.

- a. In the OnPostDataItem trigger enter code that:
 - i. Closes the dialog window.
 - ii. If the No. of the Sales Header is blank, the program displays a message that states there is nothing to invoice.
 - iii. If the No. of the Sales Header is not blank, runs the **FinalizeSalesInvoiceHeader** function.
 - iv. The program displays a message that displays the number of errors that occurred, if any, or the number of invoices that were created.

Code Example

```
Window.CLOSE;

IF SalesHeader."No." = " THEN BEGIN

MESSAGE(Text007);

END ELSE BEGIN

FinalizeSalesInvoiceHeader;

IF NoOfSalesInvErrors = 0 THEN

MESSAGE(

Text005,

NoOfSalesInv)

ELSE

MESSAGE(
```

```
Text006,  
NoOfSalesInvErrors)  
END;
```

9. Access the Request Page Designer by clicking **View > Request Page**.
 - a. Add an Options group to the **Request** page and include these variables:
 - PostingDateReq
 - DocDateReq
 - CalclnvoiceDiscount
 - PostInvoices
 - b. Set the property for the **Request** page so that the page saves the values after the page is closed.
 - c. Enter code in the appropriate trigger so that when the **Request** page is opened, if the **PostingDateReq** and **DocDateReq** variables are not filled, they are set to the work date and set the CalclnDisc value to the value of the **Calc. Inv. Discount** field in the **Sales Setup**.
 - i. In the OnOpenPage trigger of the Request Page enter the following code:

Code Example

```
IF PostingDateReq = 0D THEN  
  
    PostingDateReq := WORKDATE;  
  
IF DocDateReq = 0D THEN  
  
    DocDateReq := WORKDATE;  
  
SalesSetup.GET;  
  
CalclnvoiceDiscount := SalesSetup."Calc. Inv. Discount";
```

- d. The Request Page will now resemble the Figure: REQUEST PAGE OF CREATE SEMINAR INVOICES

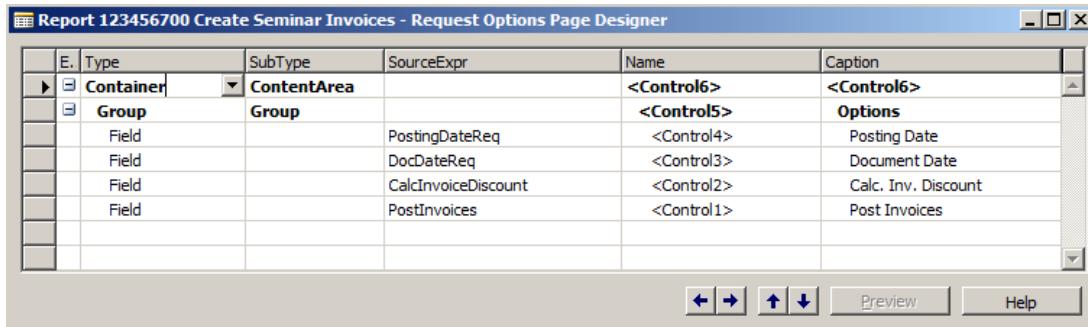


FIGURE 6.10: REQUEST PAGE OF CREATE SEMINAR INVOICES

Task 3: Testing

High Level Steps

1. To test this part of the seminar module, complete the steps that are described in the previous exercises. In particular, make sure that the Customer who is associated with the registrations that are being tested is set up correctly.

Detailed Steps

1. To test this part of the seminar module, complete the steps that are described in the previous exercises. In particular, make sure that the Customer who is associated with the registrations that are being tested is set up correctly.
 - a. Click **Seminars > Periodic Activities > Periodic Activities** from the main menu. The **Create Seminar Invoices** Request page opens.
 - b. Enter data so that one or more posted registrations meet the criteria. If there are no posted registrations, create one.
 - c. Click **OK** to run the ProcessingOnly Report.
 - d. Look at the Sales Invoices that were created. See that the header and lines were created correctly based on the Seminar information.
 - e. Expect to have one invoice per Bill-to Customer with a line for each contact registered.
 - f. Check that the data flowed correctly from the registration to the invoice.
 - g. 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.

Module Review

Module Review and Takeaways

In this module, you created two reports. The first, Participant List, was a typical report. The second report was a processing-only report that enabled you to create invoices for customers with participants in completed seminars.

In the next module, you will add statistics to the Seminar module.

Test Your Knowledge

Test your knowledge with the following questions.

Sequencing Activity

Put the following steps in order by numbering each to indicate the correct order.

	Steps
	OnPreDataItem
	OnPreDataItem
	OnPostDataItem
	OnPostReport
	OnPreReport
	OnAfterGetRecord
	OnAfterGetRecord
	OnInitReport
	OnInitReport
	OnPostReport
	OnPreReport
	OnPostDataItem

1. What report function would you use to skip the processing of one record in a DataItem?

C/SIDE Solution Development in Microsoft Dynamics® NAV 2013

2. What are the advantages of using processing-only reports in some situations instead of codeunits?

3. In what trigger do you first have access to the values that the user entered on the **Request** page?

Test Your Knowledge Solutions

Module Review and Takeaways

Sequencing Activity

	Steps
5	OnPreDataItem
5	OnPreDataItem
9	OnPostDataItem
11	OnPostReport
3	OnPreReport
7	OnAfterGetRecord
7	OnAfterGetRecord
1	OnInitReport
1	OnInitReport
11	OnPostReport
3	OnPreReport
9	OnPostDataItem

1. What report function would you use to skip the processing of one record in a Dataitem?

MODEL ANSWER:

The CorrReport.SKIP function is used to skip one record in a Dataitem.

2. What are the advantages of using processing-only reports in some situations instead of codeunits?

MODEL ANSWER:

In Codeunits, you must write all of the code to build the data record looping mechanism. Report objects have this mechanism built in by default. It is much easier to obtain user input by using a report object.

3. In what trigger do you first have access to the values that the user entered on the **Request** page?

MODEL ANSWER:

The first trigger that runs after the user has completed the **Request** page is the OnPreReport trigger. This trigger is used to evaluate the user's input before it continues with the data items.

MODULE 7: STATISTICS

Module Overview

In this module you add statistics to the Seminar Management module. Seminar managers at CRONUS International Training Academy require immediate and flexible financial reporting. Microsoft Dynamics NAV 2013 meets this business need by using FlowFilters and FlowFields.

Objectives

The objectives are:

- Create a page that calculates price sums efficiently.
- Make the page available from the Seminar pages.
- Use FlowFilters to easily calculate statistics for different time periods.

Prerequisite Knowledge

FlowFilters are useful for limiting calculations so that they include only the values in a column that have specific properties. For example, on the **Seminar Statistics** page, users may want to sum the total price of a seminar four times, for four time periods (such as week, month, current year, and prior year). This is possible if the application takes advantage of SumIndexField Technology (SIFT) by using FlowFilter fields with the FlowFields. Use FlowFilters to determine how much information the system includes when it calculates the contents of FlowFields. Use a FlowFilter to filter a FlowField.

Using FlowFilters and FlowFields

A FlowField is defined by setting the FieldClass property of the field to FlowField. The CalcFormula property defines the functionality of the FlowField. The CalcFormula can include filter values that are based on constant and/or variable parameters. A FlowFilter is a specific type of filter field that is defined in the same table that contains the associated FlowField. Therefore, you can apply FlowFilters to the source tables on which FlowFields are based.

To implement a statistics page by using the FlowFields and FlowFilter fields in a master table, view Table 18 **Customer** and Page 151 **Customer Statistics**. The columns of the **Customer Statistics Sales** FastTab show the Sales (LCY) field for the following four time periods:

- This Period
- This Year
- Last Year
- To Date

The data that is shown in these fields is generated by a FlowField, Sales (LCY), and several FlowFilters in the **Customer** table. The CalcFormula that is shown in the Sales (LCY) field properties uses several FlowFilters, but just consider the Date Filter now.

In the OnAfterGetRecord trigger of the **Customer Statistics** page, the Date Filter is set for each desired time period by using the Date Filter-Calc codeunit. Then use the **CALCFIELDS** function for each Date Filter to calculate a value for the Sales (LCY). Use similar logic when you create the **Seminar Statistics** page.

For more information about SIFT, FlowFields, and FlowFilters, refer to the Microsoft Dynamics NAV 2013 Developer and IT Pro Help.

Seminar Statistics

Seminar statistics lets the user quickly and easily retrieve an overview of the price statistics for a specific seminar.

Solution Analysis

You can use FlowFields and FlowFilters on the seminar solution to provide statistics as described in the functional requirements in Module 1, Business Case Diagnosis and Analysis.

The client requires statistics for different time periods, such as for a month, for last year, for this year, and up to the current date.

This requirement indicates that the client wants to open a **Statistics** page from a **Seminar** page. This page calculates the statistics for the total price and shows it for the four time periods that are listed.

Seminar managers should have access to the seminar statistics from a **Seminar Detail** page. This page calculates the total price statistics for the seminar for the time periods that are indicated in the function requirements.

Solution Design

The **Seminar Statistics** page must be available from the **Related Information** menu on the **Seminar Card** and **Seminar List** pages. The **Seminar Statistics** page automatically calculates the statistics for the selected seminar by using the information from the **Seminar Ledger Entry** page.

The **Seminar Statistics** page displays statistical information for one seminar, as shown in the "Seminar Statistics Page" figure.

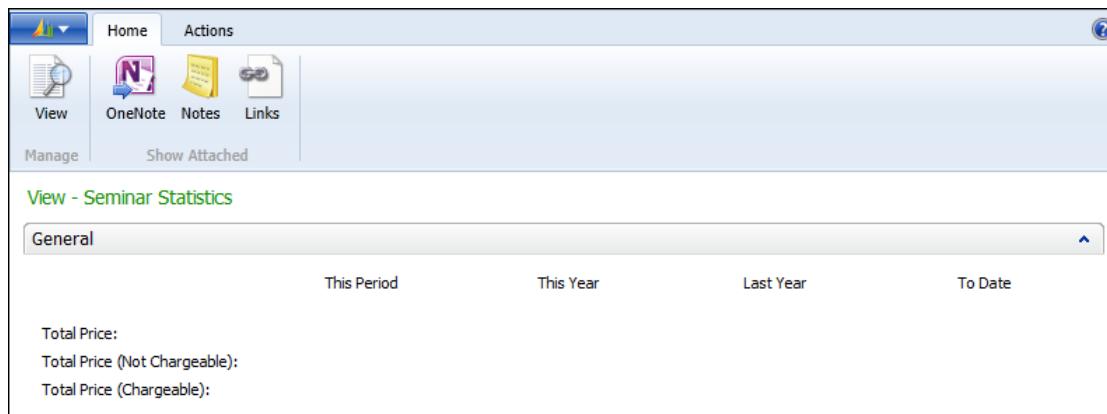


FIGURE 7.1: THE SEMINAR STATISTICS PAGE (123456714)

Seminar Card (Page 123456700): The **Seminar** menu selection on the **Related Information** button on this page is changed to include a new command for seminar statistics.

Seminar List (Page 123456701): The **Seminar** menu selection on the **Related Information** button on this page is changed to include a new command for seminar statistics.

The solution requires the addition of FlowFields and FlowFilters to the **Seminar** table (123456700). Also, a key of Seminar No., Posting Date, Charge Type, Chargeable is added to the **Seminar Ledger Entry** table 123456732.

Use values from the **Seminar Ledger Entry** page to calculate the Total Price, Total Price (Chargeable) and Total Price (Not Chargeable) in the **Seminar** table, for four time periods. Use an array of four dimensions to store the totals for each of these three prices. Use the standard DateFilter-Calc codeunit to calculate date filters for the time periods. With the date filters, you can filter the Seminar table and use it to calculate the price fields that are shown on the page for each time period.

Lab 7.1: Creating FlowFields for Sums

Scenario

You need FlowFields to support the planned **Statistics** page.

Exercise 1: Implement FlowFields for sums in the Seminar solution.

Task 1: Add Flowfields to the solution.

High Level Steps

- Follow these steps to implement FlowFields for sums in this solution.

Detailed Steps

- Follow these steps to implement FlowFields for sums in this solution.
 - Add a secondary key of Seminar No., Posting Date, Charge Type, Chargeable to the **Seminar Ledger Entry** table (1123456732). Set the property for the key so that the **Total Price** field is the sum index field.
 - Add fields to the **Seminar** table (123456700) as shown in the following table.

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; refer to Step 3 in this topic for the CalcFormula. Must be noneditable. AutoFormatType=1
26	Total Price (Not Chargeable)	Decimal		FlowField; refer to Step 4 in this topic for the CalcFormula. Must be noneditable. AutoFormatType=1

No.	Field Name	Type	Length	Comment
27	Total Price (Chargeable)	Decimal		FlowField; refer to Step 5 in this topic for the CalcFormula. Must be noneditable. AutoFormatType=1

- c. Set the CalcFormula property 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 following occurs:
 - **Seminar No.** field corresponds to the **No.** field.
 - **Posting Date** corresponds to the **Date Filter**.
 - **Charge Type** corresponds to the **Charge Type Filter**.
- d. Set the CalcFormula property 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 following occurs:
 - **Seminar No.** corresponds to the **No.** field.
 - **Posting Date** corresponds to the **Date Filter**.
 - **Charge Type** corresponds to the **Charge Type Filter**.
 - Records are Chargeable.
- e. Set the CalcFormula property 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 following occurs:
 - **Seminar No.** corresponds to the **No.** field.
 - **Posting Date** corresponds to the **Date Filter**.
 - **Charge Type** corresponds to the **Charge Type Filter**.
 - Records are Chargeable.

Lab 7.2: Creating the Seminar Statistics Page

Scenario

A **Seminar Statistics** page is required.

Exercise 1: Implement a Statistics Page

Exercise Scenario

Follow the approach that you used for other statistics pages and use the table changes that you just completed to create a **Seminar Statistics** page. Make it accessible from the **Seminar Card** and **List** pages.

Task 1: Create the Seminar Statistics Page

High Level Steps

1. Follow these steps to implement the **Statistics** page.
2. Enter code in the appropriate trigger so that after the page retrieves the record, the program performs the following tasks.
3. Add a ContentArea Container to the page. Add an indented Group with label General to the ContentArea.
4. Add a **Group** without a label and with a subtype of FixedLayout. This is the frame for the statistics area of this page. The values are populated by defining columns for all four (4) dimensions.
5. Repeat step 5 for three more groups. Label these groups as follows: This Year, Last Year, and To Date. Make sure each of these three groups has appropriately increasing dimension values.
6. Make the page available from the two seminar pages. Add the selections that are described in the table of the **Related Information** menu of the **Seminar Card** page (123456700) after the **Comments** command.
7. Add the same menu selections to the **Seminar List** page (123456701).

Detailed Steps

1. Follow these steps to implement the **Statistics** page.
 - a. Create the **Seminar Statistics** page (123456714), based on the **Seminar** table. A statistics page has a PageType of Card, and has the LinksAllowed property set to No. Do not add any controls to this page at this time.

- b. Define the following global variables for the page.

Name	DataType	SubType	Length
DateFilterCalc	Codeunit	DateFilter-Calc	
SeminarDateFilter	Text		30
SeminarDateName	Text		30
CurrentDate	Date		
TotalPrice	Decimal		
TotalPriceNotChargeable	Decimal		
TotalPriceChargeable	Decimal		
I	Integer		

- c. Set the Dimensions property for all the variables except DateFilterCalc, CurrentDate, and I so that each variable is an array of four dimensions.
- d. Set the property for the page so that it is noneditable.
2. Enter code in the appropriate trigger so that after the page retrieves 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.
 - Runs the **CreateAccountingPeriodFilter** function of the DateFilter-Calc codeunit with parameters of the first dimensions of the SeminarDateFilter and SeminarDateName, the CurrentDate, and **0** (zero).
 - Runs the **CreateFiscalYearFilter** function of the DateFilter-Calc codeunit with parameters of the second dimensions of the **SeminarDateFilter** and **SeminarDateName**, the **CurrentDate**, and **0**.
 - Runs the **CreateFiscalYearFilter** function of the DateFilter-Calc codeunit with parameters of the third dimensions of the **SeminarDateFilter** and **SeminarDateName**, the **CurrentDate**, and the value **-1**. **HINT:** There is similar code on the Customer Statistics page.
 - For each dimension, filters the table to records where the Date Filter corresponds to the value in the appropriate dimension of the SeminarDateFilter and calculates the **Total Price**, **Total Price (Not Chargeable)**, and **Total Price (Chargeable)** fields.
 - Sets the value for the appropriate dimension of the TotalPrice, TotalPriceNotChargeable, and TotalPriceChargeable to the values in the corresponding fields.

- h. Filters the table to those records where the Date Filter is before the CurrentDate.
- i. In the OnAfterGetRecord trigger verify the code resembles:

Code Example

```
SETRANGE("No.", "No.");  
  
IF CurrentDate <> WORKDATE THEN BEGIN  
  
    CurrentDate := WORKDATE;  
  
    DateFilterCalc.CreateAccountingPeriodFilter(SeminarDateFilter[1], SeminarDateName[1], CurrentDate, 0);  
  
    DateFilterCalc.CreateFiscalYearFilter(SeminarDateFilter[2], SeminarDateName[2], CurrentDate, 0);  
  
    DateFilterCalc.CreateFiscalYearFilter(SeminarDateFilter[3], SeminarDateName[3], CurrentDate, -1);  
  
    END;  
  
FOR i := 1 TO 4 DO BEGIN  
  
    SETFILTER("Date Filter", SeminarDateFilter[i]);  
  
    CALCFIELDS("Total Price", "Total Price (Not Chargeable)", "Total Price (Chargeable)");  
  
    TotalPrice[i] := "Total Price";  
  
    TotalPriceNotChargeable[i] := "Total Price (Not Chargeable)";  
  
    TotalPriceChargeable[i] := "Total Price (Chargeable)";  
  
    END;  
  
SETRANGE("Date Filter", 0D, CurrentDate);
```

3. Add a ContentArea Container to the page. Add an indented Group with label General to the ContentArea.
 - a. In the Page Designer, on the first line set Type to Container and SubType to ContentArea

- b. Goto the second line and set Type to Group and Subtype to Group and Caption to General and make sure it is indented one step to the right using the right arrow button.

- 4. Add a **Group** without a label and with a subtype of FixedLayout. This is the frame for the statistics area of this page. The values are populated by defining columns for all four (4) dimensions.

 - a. Goto the third line and set Type to Group and Subtype to FixedLayout.

- 5. Repeat step 5 for three more groups. Label these groups as follows: This Year, Last Year, and To Date. Make sure each of these three groups has appropriately increasing dimension values.

 - a. Complete the design in the Page designer so that it looks like the figure Seminar Statistics Page (123456714) In Page Designer:

Page 123456714 Seminar Statistics - Page Designer					
E.	Type	SubType	SourceExpr	Name	Caption
▶	Container	ContentArea		<Control1000000000>	<Control1000000000>
▶	Group	Group		<Control1000000001>	General
▶	Group	FixedLayout		<Control1000000002>	<Control1000000002>
▶	Group	Group		<Control1000000007>	This Period
	Field		SeminarDateName[1]	<Control1000000003>	<Control1000000003>
	Field		TotalPrice[1]	<Control1000000004>	Total Price
	Field		TotalPriceNotChargeable[1]	<Control1000000005>	Total Price (Not Chargeable)
	Field		TotalPriceChargeable[1]	<Control1000000006>	Total Price (Chargeable)
▶	Group	Group		<Control1000000008>	This Year
	Field		SeminarDateName[2]	<Control1000000009>	<Control1000000009>
	Field		TotalPrice[2]	<Control1000000012>	Total Price
	Field		TotalPriceNotChargeable[2]	<Control1000000011>	Total Price (Not Chargeable)
	Field		TotalPriceChargeable[2]	<Control1000000010>	Total Price (Chargeable)
▶	Group	Group		<Control1000000013>	Last Year
	Field		SeminarDateName[3]	<Control1000000017>	<Control1000000017>
	Field		TotalPrice[3]	<Control1000000016>	Total Price
	Field		TotalPriceNotChargeable[3]	<Control1000000015>	Total Price (Not Chargeable)
	Field		TotalPriceChargeable[3]	<Control1000000014>	Total Price (Chargeable)
▶	Group	Group		<Control1000000022>	To Date
	Field		SeminarDateName[4]	<Control1000000021>	<Control1000000021>
	Field		TotalPrice[4]	<Control1000000020>	Total Price
	Field		TotalPriceNotChargeable[4]	<Control1000000019>	Total Price (Not Chargeable)
	Field		TotalPriceChargeable[4]	<Control1000000018>	Total Price (Chargeable)

FIGURE 7.2: SEMINAR STATISTICS PAGE (123456714) IN PAGE DESIGNER

6. Make the page available from the two seminar pages. Add the selections that are described in the table of the **Related Information** menu of the **Seminar Card** page (123456700) after the **Comments** command.

- a. Add the following additional **Seminar** actions:

Options	Comment
&Statistics (F7)	Opens page 123456714 Seminar Statistics for the selected Seminar . Set the Image property to Statistics.

7. Add the same menu selections to the **Seminar List** page (123456701).

- a. Add the following additional **Seminar** actions:

Options	Comment
&Statistics (F7)	Opens page 123456714 Seminar Statistics for the selected Seminar . Set the Image property to Statistics.

Task 2: Solution Testing

High Level Steps

1. To test the statistics page, you must have entries in the **Seminar Ledger Entry** table with the related posted seminar registrations. Follow these steps to verify the solution implementation:

Detailed Steps

1. To test the statistics page, you must have entries in the **Seminar Ledger Entry** table with the related posted seminar registrations. Follow these steps to verify the solution implementation:
 - a. Open the **Seminar Card** page and view a seminar that has posted registrations. Verify that the seminar has ledger entries by selecting **Seminar**, and then click **Entries**.
 - b. Click **Ledger Entries** on the **Seminar Card** page.
 - c. Select **Seminar**, and then click **Statistics** to open the **Seminar Statistics** window.
 - d. Verify that the totals that are shown are correct for the different rows and columns. It may be convenient to apply a table filter to the **Seminar Ledger Entries** window and compare the results with the amounts that are shown in the **Seminar Statistics** window.

Module Review

Module Review and Takeaways

This module focused on creating a statistics page for seminars that sums the total price in four time periods. This statistics page is made available from the **Seminar** page **Related Information** menu. Add this functionality by using FlowFields and FlowFilters. FlowFilters and FlowFields use SIFT technology to provide filtered statistics quickly and easily.

Test Your Knowledge

Test your knowledge with the following questions.

1. How are FlowFilters used in calculations of FlowFields?

2. What are the advantages of using FlowFields to make calculations?

3. What is the purpose of a **FlowFilter** field?

4. What is the Microsoft Dynamics® NAV standard shortcut for opening a Statistics page?

Test Your Knowledge Solutions

Module Review and Takeaways

1. How are FlowFilters used in calculations of FlowFields?

MODEL ANSWER:

FlowFilters are set as filter values in the CalcFormula of FlowFields. They are used to limit the information that is retrieved when the system calculates the value of a FlowField.

2. What are the advantages of using FlowFields to make calculations?

MODEL ANSWER:

The advantages of using FlowFields are ease in developing and using fields that have calculated values and faster calculation processing. Calculating FlowFields takes several lines of C/AL code, because the filters and parameters are defined in the field's CalcFormula property. Writing code to calculate those values directly from their source tables takes significantly more effort to develop, and in some cases, may take much more processing of data to calculate.

3. What is the purpose of a **FlowFilter** field?

MODEL ANSWER:

A FlowFilter is a special type of filter field that you define in the same table that contains the associated FlowField. You can include FlowFilters in the CalcFormula of the FlowField to enable user-defined filters to be applied in the calculation of the FlowField. When you set a filter in a FlowFilter, the system includes the FlowFilter value as a filtering constraint when calculating the contents of related FlowFields.

4. What is the Microsoft Dynamics® NAV standard shortcut for opening a Statistics page?

MODEL ANSWER:

The standard keyboard shortcut for opening a Statistics page is Ctrl+Shift+J.

MODULE 8: DIMENSIONS

Module Overview

In Microsoft Dynamics NAV® 2013, dimensions are configurable attributes that accompany all master records, documents, journals, posted documents, and ledger entries. Users can define their own dimension structures. Many reporting features use dimensions to enable simple multidimensional analysis of data as it is needed.

Dimensions are a core feature of Microsoft Dynamics NAV 2013. You must integrate dimensions into any new module that you create that involves any of the processes where dimensions are present in the standard application.

When you add new master records, documents, journals, or ledgers, you must make sure that those new record types provide the same dimension functionality as other application areas, such as Sales & Receivables, or General Ledger. This not only safeguards a consistent user experience across the application, but enables users to analyze and understand the data that is generated by a custom module, as they do with the standard application.

Seminar Management application uses many standard application concepts where dimensions are usually present. Therefore, you must extend the Seminar Management functionality to include dimensions functionality in the following areas: master records, documents, and ledger entries.

Objectives

- Describe Global, Shortcut, and Budget dimension types and their functions.
- List the basic rules of Dimension Setup.
- Present the dimension management data and process models.

Implement dimensions on the master record level.

Prerequisite Knowledge

A *dimension* is additional information that is attached to records, primarily for analysis. Table fields are static, unchangeable attributes of records and are predetermined by the developer. Whereas, dimensions are dynamic, configurable attributes that the user selects and sets up.

A company's accounts consist of many entries from many sources. Accounts are associated with many activities within the company. It is frequently necessary to create statements, statistics, and analyses that are extracts of the complete financial statements. These extracts can be created by using individual dimensions or combinations of dimensions.

If you set up a dimension called Department, and then use this dimension and a dimension value when you post an entry, you can retrieve information later, such as items that were sold and the departments that sold them. If more than one dimension was used on posted entries, the user can create a richer analysis of a company's activities. For example, a single sales entry can include multiple dimension information about the account to which the item sale was posted, where the item was sold, who sold it, and the type of customer who made the purchase.

By using dimensions, you can analyze trends and compare various characteristics across a range of entries. The analysis view functionality is especially effective for this purpose. However, you can also use filters, account schedules, and reports to create informative dimensions analyses.

You can also use dimensions to support business rules by influencing how a user can combine dimensions and how dimensions can be posted. This may be useful if certain departments cannot use particular accounts or sell to particular customers.

 **Additional Reading:** For more information about supporting business rules through dimensions, see *Dimension Combination Table* and *Default Dimension Table* topics in Microsoft Dynamics NAV help.

Many different types of records in Microsoft Dynamics NAV 2013 can have dimensions. The following types of records in Microsoft Dynamics NAV 2013 always have dimensions:

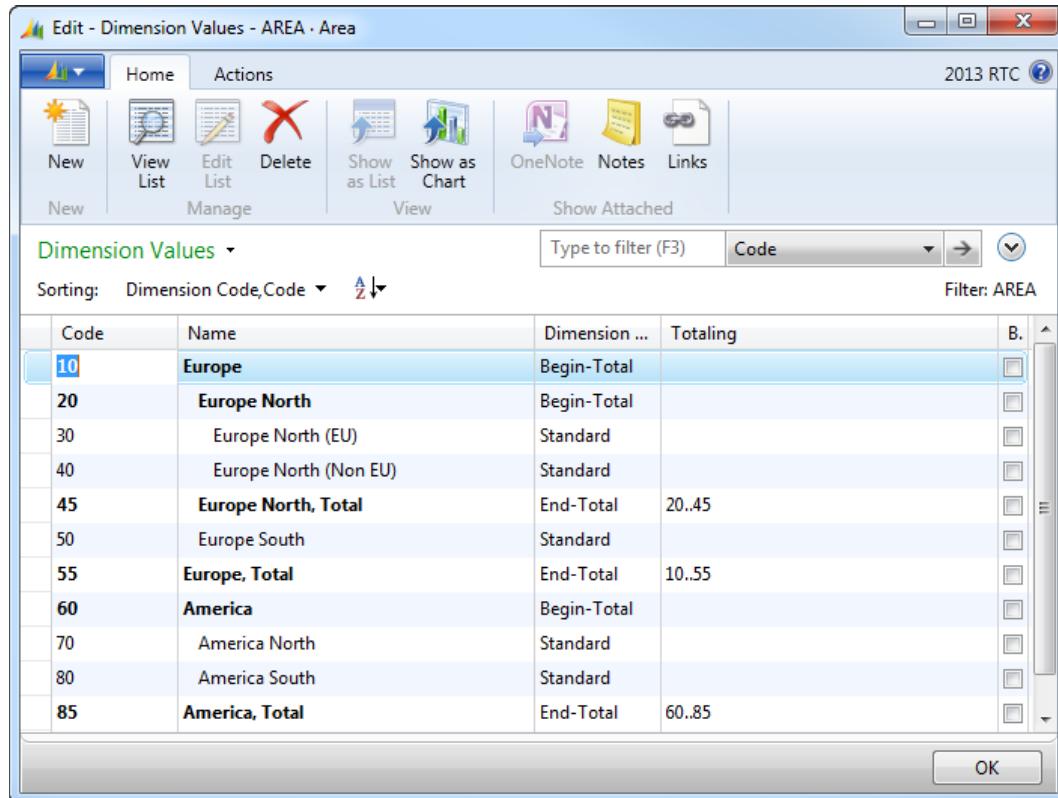
- Master records
- Document headers and lines
- Posted document headers and lines
- Journal lines
- Ledger entries

 **Note:** There are few exceptions to this rule. Two important exceptions are the Reminders and Finance Charge Memos, where dimensions are only used at the document header level, not in the lines. This is because neither reminders nor finance charge memos are a core process. They both require analysis only at the header level. Any detailed analysis is available through other tables that are related to the process.

Nevertheless, when providing new functionality, such as supporting a core process of a company, such as Seminar Management, you should provide full dimension management functionality.

Each dimension can have an unlimited number of dimension values. For example, a dimension called Department can have dimension values of Sales, Administration, and so on. Dimensions can also be hierarchical with multiple levels of values organized in parent/child relationships. Dimensions can also include totaling groups that have headers and footers.

For example, the AREA dimension in the demo database for Microsoft Dynamics NAV 2013 has the following value structure. See the "Dimension Values for the AREA Dimension figure."



Dimension Values				
Sorting:		Type to filter (F3)	Code	Filter: AREA
10	Europe	Begin-Total		
20	Europe North	Begin-Total		
30	Europe North (EU)	Standard		
40	Europe North (Non EU)	Standard		
45	Europe North, Total	End-Total	20.45	
50	Europe South	Standard		
55	Europe, Total	End-Total	10.55	
60	America	Begin-Total		
70	America North	Standard		
80	America South	Standard		
85	America, Total	End-Total	60.85	

FIGURE 8.1: DIMENSION VALUES FOR THE AREA DIMENSION

Users define and customize dimensions and their values to meet their company's needs and business process requirements.



Dimension Types

Users can define as many dimensions as they need in their company. There can be an unlimited number of dimension values for each dimension. You define the values to use on entries in journals and documents and in dimensions-related reports and batch jobs.

Following are the three most important dimension types:

- Global dimensions
- Shortcut dimensions
- Budget dimensions

Global Dimensions

You can configure two dimensions as global dimensions. Users configure the global dimensions in the General Ledger Setup card. The global dimensions are available throughout Microsoft Dynamics NAV 2013, and are stored as physical attributes (fields) on the tables that they describe. This lets users easily filter the entries, or use global dimension filters on reports, batch jobs, account schedules, and so on.



Note: The tables that use global dimensions always have the **Global Dimension 1 Code** and **Global Dimension 2 Code** fields. Users never see these captions. Instead, Microsoft Dynamics NAV automatically translates the captions for global dimensions into the captions that the user has configured in the dimension setup, by calling the **CaptionClassTranslate** function of the codeunit 1, ApplicationManagement.

CaptionClassTranslate function then calls the codeunit 42, CaptionManagement. This reads the dimension caption from the **Dimension** table, and shows it to the user. Dimension captions are cached, and you must restart the application if you change the dimension caption in the setup.

Global dimensions are typically a company's most used and important dimensions because of their availability throughout the program. Because these dimensions are present as actual fields in tables, users cannot change these dimensions directly. There is a batch job that changes the global dimensions and then updates all records. This process can take some time.

Shortcut Dimensions

Two global and up to six additional dimensions (a total of eight dimensions) can be shortcut dimensions. Unlike global dimensions, these dimensions are not present as actual fields in tables, but are displayed as fields in document and journal pages. This enables users to enter dimension values directly on the lines in journals or documents. This saves time and increases productivity.

When users enter dimension values for records, they must do the following: open additional pages, enter the dimensions in multiple lines, and then close the page to return to the original record. For example, to enter dimensions on a sales order line, you must click **Line > Dimensions**, or press SHIFT+CTRL+D. Then in the **Edit Dimension Set Entries** page, you must configure multiple dimension values as multiple different lines by first selecting the dimension, and then the dimension value. It is time consuming to enter multiple dimensions for many records. Shortcut dimensions make users more productive by displaying additional dimension columns in journals and document lines pages. Users can enter dimension values for up to eight configured shortcut dimensions.

 **Note:** *The first two shortcut dimensions, that are also global dimensions, exist as actual fields in tables. These dimensions are called **Shortcut Dimension 1 Code** and **Shortcut Dimension 2 Code**. The other six shortcut dimensions do not exist as fields in tables. However, they are present as field controls in pages that use shortcut dimensions. These field controls are bound to elements of an array that is usually called `ShortcutDimCode`. Trigger code on all shortcut dimension field controls makes sure that when users enter values into the controls, that an actual dimension set is stored in the **Dimension Set Entry** table.*

Captions for shortcut dimension field controls are translated exactly like global dimensions.

Because shortcut dimensions 3 through 8 are not present as actual fields in tables, users can only filter on the first two shortcut dimensions.

Budget Dimensions

When you enter budgets, you typically bind budget entries to specific dimensions, such as projects, or departments. Each budget can use up to four dimensions. Budget dimensions do not have relationships with global dimensions, and users can use any four dimensions as budget dimensions, regardless of which dimensions are configured as global dimensions.

Dimension Setup

Users must first configure dimensions in Microsoft Dynamics NAV 2013. Configuring dimensions involves two steps: configuring the dimensions and dimension values, and configuring the global and shortcut dimensions for the application.

Configuring Dimensions

Users configure dimensions in the Administration area in Departments, under **Administration > Application Setup > Financial Management > Dimensions > Dimensions**. For each dimension, users must provide the **Code**, **Name**, **Code Caption**, and **Filter Caption** fields.

The “Dimensions List Page” figure shows the Dimensions list in the Microsoft Dynamics NAV 2013 client for Windows.

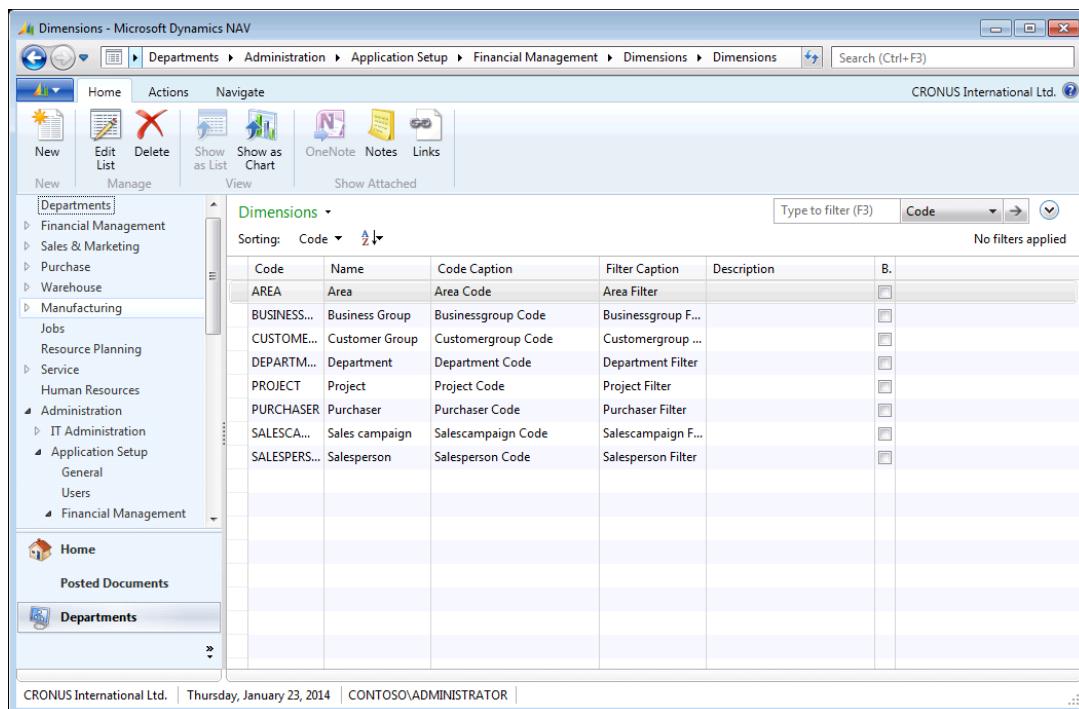


FIGURE 8.2: THE DIMENSIONS LIST PAGE

Each dimension contains a series of dimension values. These can be organized as lists, or as hierarchies. To configure the dimension values, in the **Dimension List** page, click **Navigate > Dimension Values** to open the **Dimension Values** page. For each dimension value, users must provide the **Code**, **Name** and **Dimension Value Type** fields.

The “Dimension Values Page” figure shows the **Dimension Values** page for the DEPARTMENT dimension.

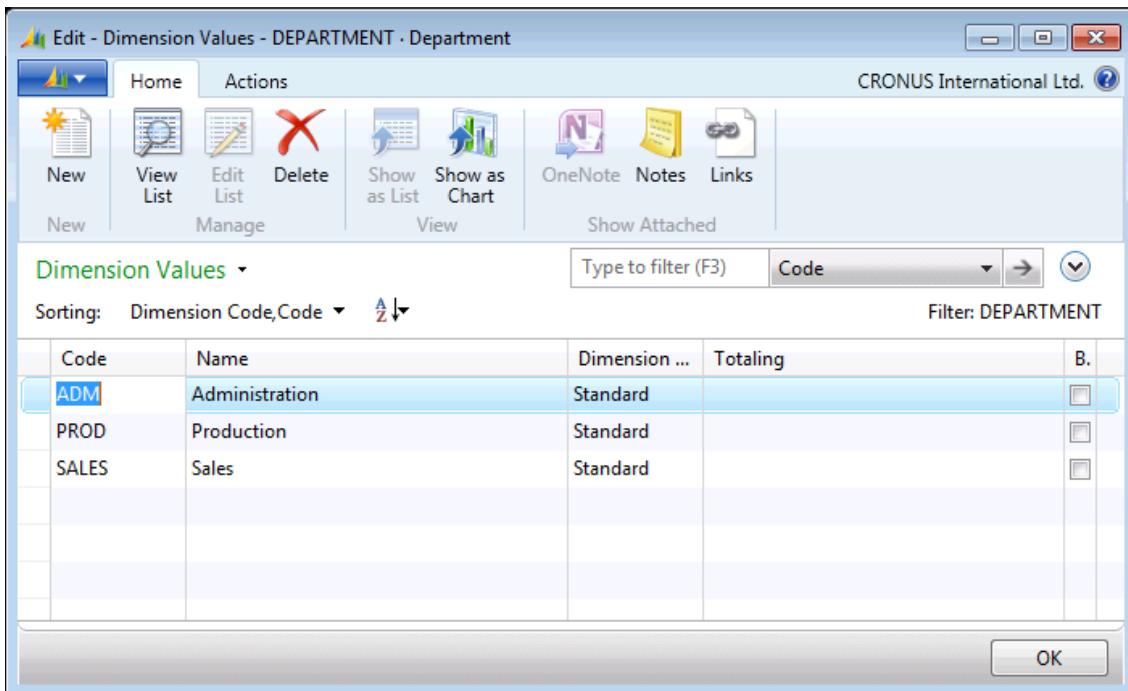


FIGURE 8.3: THE DIMENSION VALUES PAGE

 **Additional Reading:** For more information about how to configure dimensions, dimension values, and dimension value hierarchies, refer to the How to: Set Up Dimensions and Dimension Values topic in the Microsoft Dynamics NAV help, or to the Setup Dimensions chapter of the 80435 Application Setup in Microsoft Dynamics NAV 2013 course.

Configuring Global and Shortcut Dimensions

Users configure global dimensions and shortcut dimensions on the **Dimensions** FastTab of the **General Ledger Setup** page, under **Departments > Administration > Application Setup > Financial Management > Finance > General Ledger Setup**.

For the Microsoft Dynamics NAV 2013 demonstration database, the global dimensions are set to the DEPARTMENT and PROJECT dimensions, as shown in the “Dimensions FastTab on the General Ledger Setup Page” figure.

If Global Dimension 1 and Global Dimension 2 are set up in the system, they now link to these two dimensions. For example, the **Global Dimension 1 Code** field in the **Customer** table will now have a caption of Department Code, because this dimension is set up as the Global Dimension 1 Code in General Ledger Setup.

C/SIDE Solution Development in Microsoft Dynamics® NAV 2013

To change the two global dimensions, in the **General Ledger Setup** page, click **Actions > Change Global Dimensions**. This opens the Change Global Dimensions batch job that changes the values in the **General Ledger Setup** page. It also updates the values of the **Global Dimension 1 Code** and **Global Dimension 2 Code** fields in all tables where they are present.

 **Note:** Because the first two shortcut dimensions are always equal to the global dimensions, you cannot change the values of the **Shortcut Dimension 1 Code** and **Shortcut Dimension 2 Code** fields manually, and running the Change Global Dimensions batch job also changes the values of these two fields.

Dimensions Data Model

Microsoft Dynamics NAV 2013 uses several tables to store dimensions, their values, and the relationships between dimensions and various types of records. Regardless of its functionality and the business processes it supports, every application module that requires dimensions uses the same underlying data model to support dimension management.

The "Dimension Management Data Model" figure shows the data model of the dimension management functionality in Microsoft Dynamics NAV 2013.

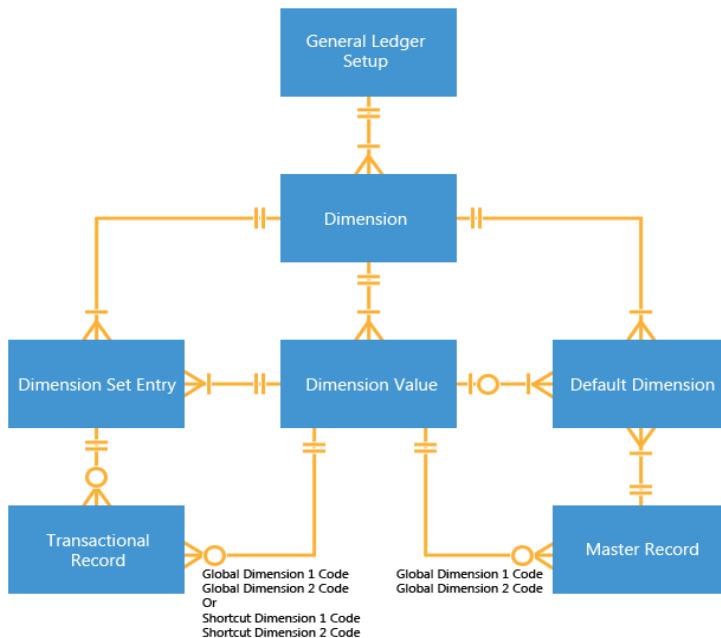


FIGURE 8.4: DIMENSION MANAGEMENT DATA MODEL

Dimension and Dimension Value Tables

The **Dimension** table stores the dimensions that you can use, and the **Dimension Value** table stores the values for those dimensions. If users want to enable analysis by an attribute that is not present in the data model, they can define a new dimension, and its values.

Suppose that a company wants to track sales by sales channel, such as retail, wholesale, or Internet. The company may define a new dimension called CHANNEL, with values of RETAIL, WHOLESALE, and INTERNET. This information is stored in the **Dimension** and **Dimension Value** tables.

Default Dimension Table

The **Default Dimension** table stores the default dimension values that are attached to a specific master record, such as a Customer, a G/L Account, or an Item. When you specify default dimensions for master records, the program suggests these dimensions and dimension values when the account is used in a transaction, for example on a journal line or a document. This makes entry posting easier for the user, as the program completes the dimension fields automatically. These dimension values are the default settings. Users can always change the default dimension values that are suggested by the program.

 **Note:** The **Default Dimension** table also specifies how a dimension value is used during posting of transactions.

The **Default Dimension** table uses the **Table ID** and **No.** fields to specify to which table and record in that table that a default dimension applies. If the **No.** field is empty, then the configured dimension and dimension value are the default for all records in the table that is specified in the **Table ID** field. This feature is called Account Type Default Dim. Users can access it from the Account Type Default Dim. action in the **Navigate** tab of the ribbon on the **Dimensions** list page.

 **Note:** When you configure the Account Type Default Dim., the default dimensions are not assigned to all new records of the specified account type (master record type) when you create new master records. Instead, when you use an account type inside journals or documents, the default dimensions are copied from both the specified master record, such as Customer, and from the default dimensions for the master record type. If the master record type is customer, then those default dimensions have **Table ID** field of 18, and empty **No.** field

Dimension Set Entry Table

The **Dimension Set Entry** table stores combinations of dimension values. Each combination of dimension values is assigned a **Dimension Set ID** field. Any transactional record, such as a document header, document line, journal line, posted document header, posted document line, or a ledger entry, that uses the same combination of dimension values, has the same value in the **Dimension Set ID** field (typically, this field has Field No. of 480).

For example, a journal line may use the following combination of dimension values.

Dimension Code	Dimension Value Code
AREA	30
CUSTOMERGROUP	MEDIUM
DEPARTMENT	SALES

This combination of dimensions may be stored in the Dimension Set Entry table as follows.

Dimension Set ID	Dimension Code	Dimension Value Code	Dimension Value ID	Dimension Name	Dimension Value Name
3	AREA	30	17	Area	Europe North (EU)
3	CUSTOMERGROUP	MEDIUM	9	Customer Group	Medium Business
3	DEPARTMENT	SALES	2	Department	Sales

The journal line then stores the value of 3 in the **Dimension Set ID** field. If any other journal line, ledger entry, or any other transactional record requires the same combination, it will reuse the existing one, and set its **Dimension Set ID** field to the same value of 3.



Note: By assigning IDs to dimension sets, and reusing those IDs when specific combinations of dimension values are used, Microsoft Dynamics NAV 2013 improves the application performance by reducing storage requirements and posting time.

Master Tables

All master tables have the **Global Dimension 1 Code** and **Global Dimension 2 Code** fields, which have table relation to the **Dimension Value** table.

Transactional Record Tables

Transactional record tables for open (not-yet-posted) transactions, including document header, document line, journal line tables, and posted document header and lines tables, have the **Shortcut Dimension 1 Code** and **Shortcut Dimension 2 Code** fields. These fields have the same relation to the **Dimension Value** table as global dimension fields have to master tables. For all tables except posted document tables, the OnValidate trigger on these fields calls the appropriate functionality in the DimensionManagement codeunit. This makes sure that values that are entered by users are valid, and it calculates appropriate dimension sets from dimension value combinations that are entered for the record.

All transactional record tables, including all not-yet-posted tables and ledger entry tables, have the additional **Dimension Set ID** field. This field contains the value of the dimension set in the **Dimension Set Entry** table that corresponds to the combination of dimension values that the user has entered for the record.

Dimension Process Model

In Microsoft Dynamics NAV 2013, the dimension process model comprises a broad set of functionality for maintaining dimension data. The model includes page actions, table and page triggers and functions, and the codeunit 408, DimensionManagement.

When you create new master, journal, document, or ledger tables and pages, you must make sure that you integrate your processes with dimension management. Do this by calling the DimensionManagement codeunit in all locations where a standard application would call it.

Master Tables

The OnInsert trigger in master tables calls the **UpdateDefaultDim** function of DimensionManagement to make sure that any default dimensions that are assigned to a master record type are inserted for a new master record. Global dimension fields are updated accordingly.

In a similar fashion, the OnDelete trigger calls the **DeleteDefaultDim** function of the DimensionManagement codeunit to delete all default dimensions for a master record when the master record is deleted.

All master tables contain the ValidateShortcutDimCode function. This function validates the user's entry in the field, and stores default dimension values for the master record into the **Default Dimension** table. It does this by calling the **ValidateDimValueCode** and **SaveDefaultDim** functions of the DimensionManagement codeunit.

This function is called from the OnValidate trigger of the **Global Dimension 1 Code** and **Global Dimension 2 Code** fields.

Document Header Tables

The document header tables contain comprehensive business logic for managing dimensions. Document header tables define the following functions to manage dimensions.

Function	Remarks
CreateDim	<p>Specifies up to ten master record types and numbers for which dimensions should be copied into the document. The CreateDim function then calls the GetDefaultDimID function of the DimensionManagement codeunit, and updates the value of the Dimension Set ID field.</p> <p>For example, in the Sales Header table, it takes the default dimensions from the Customer, Salesperson, Campaign, Responsibility Center, and Customer Template that are contained in the sales document to calculate the Dimension Set ID field for the sales document.</p> <p>This function is typically called from the OnValidate trigger of the field that is related to the most important master record of the document, such as Bill-to Customer No. in the Sales Header table.</p>
ValidateShortcutDimCode	<p>Calls the ValidateShortcutDimValues function of DimensionManagement codeunit to calculate the new Dimension Set ID field, and then changes the record if it is necessary. It also calls the UpdateAllLineDim function.</p> <p>It is called from the OnValidate trigger of the Shortcut Dimension 1 Code and Shortcut Dimension 2 Code fields.</p>

Function	Remarks
ShowDocDim	<p>Calls the EditDimensionSet2 function of the DimensionManagement codeunit, then updates the Dimension Set ID field if it is necessary, and calls the UpdateAllLineDim function.</p> <p>This function is called from the OnLookup trigger of the Dimension Set ID field, and from the OnAction trigger for the Dimensions action on the document page.</p>
UpdateAllLineDim	<ul style="list-style-type: none"> Asks the user for the confirmation, and then does the following for each line: Calls the GetDeltaDimSetID function of the DimensionManagement codeunit. Updates the Dimension Set ID field. Calls the UpdateGlobalDimFromDimSetID function of the DimensionManagement codeunit.

Document Line Tables

Similar to document header tables, the document line tables have a similar set of functions for managing dimensions that call the DimensionManagement codeunit when users change or access dimension information that is related to the line.

The following table shows the dimension management functions that are typically defined in a document line table.

Function	Remarks
CreateDim	Performs the same logic as the CreateDim function on document header tables, except it is only for the document line.
ShowDimensions	Achieves the same goal as the ShowDocDim function on document header tables.
ValidateShortcutDimCode	Calls the ValidateShortcutDimValues function of the DimensionManagement codeunit to calculate the new Dimension Set ID field.

Function	Remarks
LookupShortcutDimCode	<p>Calls the LookupDimValueCode function of the DimensionManagement codeunit, and then calls the ValidateShortcutDimCode function.</p> <p>This function is called from the OnLookup triggers of shortcut dimension field controls on document subpages, such as Sales Order Subform.</p>
ShowShortcutDimCode	<p>Calls the GetShortcutDimensions function of the DimensionManagement codeunit.</p> <p>It is called from the OnAfterGetRecord trigger on the page that displays the document line.</p>

Journal Line Tables

Journal line tables resemble document line tables. There are very few differences between the two table types. Documents have headers, whereas journals do not.

Posted Document and Ledger Entry Tables

Users cannot usually change tables that contain posted information, which includes posted documents and ledger entries. Therefore, the tables do not contain much business logic. ShowDimensions is the only function that calls the ShowDimensionSet function of the DimensionManagement codeunit. This function is called when users look up the value in the Dimension Set ID field, or click the Dimensions action in a relevant posted document, posted document lines, or ledger entries page.

Master Record Pages

All card pages for master records contain an action named Dimensions. This action is located in the **Navigate** tab in the ribbon and shows the dimensions that are attached to the master record.

 **Note:** Even though master record tables always include the **Global Dimension 1 Code** and **Global Dimension 2 Code** fields, these fields are not shown on the master card pages in standard Microsoft Dynamics NAV 2013. If users require these fields, you can add them to master card pages.

On list pages for master records, the following two actions are located in the **Navigate** tab in the ribbon:

- **Dimensions-Single** – Shows and lets the user edit dimensions that are attached to the single selected master record.
- **Dimensions-Multiple** – Shows and lets the user edit dimensions that are attached to all master records in the selection.

 **Note:** To enable the Dimensions-Multiple feature for a new master record type, you must customize the page 542, **Default Dimensions-Multiple**, by adding a new function that copies the default dimensions from the new master record type.

Document and Posted Document Pages

Similar to card type pages for master records, all document and list type pages for documents contain an action named Dimensions. This action is located in the **Navigate** tab in the ribbon and shows the dimensions that are attached to the master record.

Document subpages contain the following dimension management functionality:

- An action named Dimensions in the Line action group shows the dimensions for the line in the edit mode.
- The OnValidate and OnLookup triggers of the Shortcut Dimension field controls call the **ValidateShortcutDimCode** and **LookupShortcutDimCode** functions of the underlying document line table.
- The OnAfterGetRecord trigger and the OnValidate trigger of fields that are related to the primary master record for the line (such as **No.** in the **Sales Line** table), call the **ShowShortcutDimCode** function of the underlying document line table.
- The OnNewRecord trigger clears the ShortcutDimCode array that keeps values for the shortcut dimensions of the current document line record.

Pages that show posted documents include the **Dimensions** action, which shows the dimensions for the posted document, or document line. Posted document pages show the action in the **Navigate** tab in the ribbon, whereas document subpages show the action in the Line action group.

Journal and Ledger Entry Pages

The journal pages contain the same functionality as the document subpages, except that they show the **Dimensions** action in the **Navigate** tab of the ribbon. Ledger entry pages only contain the **Dimensions** action in the **Navigate** tab of the ribbon, and do not have any other functionality that is related to dimensions.

DimensionManagement Codeunit

The codeunit 408, DimensionManagement is at the core of the dimension management process model in Microsoft Dynamics NAV 2013. This is where most of the dimension data processing occurs. Functions in this codeunit handle several groups of processes, such as managing dimension sets, checking posting rules, managing default dimensions, and so on. All tables and pages that include global or shortcut dimension fields, or enable users to show, edit, or look up dimension information, reference this codeunit and call its functions.

You must also call the following important functions from any custom-built master, document, journal, or ledger page.

Function	Remarks
UpdateDefaultDim	Makes sure that any default dimensions that are assigned to a specific table (master record type) are inserted for a new master record. It also updates any affected global dimension fields.
DeleteDefaultDim	Deletes all default dimensions for a master record when the master record is deleted.
ValidateDimValueCode	Validates whether a value that is specified for a dimension is a valid value. If it cannot find the value that the user has entered, it tries to find the first dimension value that begins with the user's entry. If it finds it successfully, it substitutes the user's entry with the actual dimension value. In all other cases, it fails with an error. This function is called from all OnValidate triggers for global and shortcut dimension fields on tables and field controls on pages.



Note: For master record tables, the OnValidate trigger actually calls the **ValidateShortcutDimCode** function of the table. This in turn calls the **ValidateDimValueCode** function of the codeunit 408.

Module 8: Dimensions

Function	Remarks
SaveDefaultDim	<p>Inserts or updates an entry in the Default Dimension table for a master record when a user enters a value in any of the global dimension fields.</p> <p>It is called from the ValidateShortcutDimCode function in the master table, which is called from the OnValidate trigger of the Global Dimension 1 Code and Global Dimension 2 Code fields.</p>
GetDefaultDimID	<p>Uses up to ten master record types and numbers and an existing dimension set ID. Then it calculates the new dimension set ID from the combination of the existing dimension set and any of the ten dimension values that are not already included in the set. Finally, it returns the new dimension set ID to the caller.</p> <p>This function is typically called from CreateDim functions of document header tables.</p>
ValidateShortcutDimValues	<p>Calls the ValidateDimValueCode function, and then calculates the new dimension set.</p> <p>It is called from the ValidateShortcutDimCode function on document header tables when users change the value for one of the shortcut dimensions.</p>
ShowDimensionSet	<p>Runs the page 479, Dimension Set Entries to show all dimension values that consist of a dimension set in the read-only mode.</p> <p>This function is typically called from the ShowDimensions function on ledger entry or posted document line tables.</p>

Function	Remarks
EditDimensionSet	Shows the page 480, Edit Dimension Set Entries to show all dimension values that consist of a dimension set in the edit mode. Then it calculates the new dimension set ID for the dimension values that were edited by the user. This function is typically called from the ShowDimensions function on document line tables.
EditDimensionSet2	Does the same as the EditDimensionSet function, except that it also updates the global dimension values through a call to the UpdateGlobalDimFromDimSetID function. This function is typically called from the ShowDocDim function on document header tables.
GetDeltaDimSetID	Combines the document header dimensions with document line dimensions and calculates and returns a new resulting dimension set ID. It is called from the UpdateAllLineDim function of document header tables when users update header dimensions and want to apply the changes to all document lines.
UpdateGlobalDimFromDimSetID	Updates values for Global Dimension 1 Code and Global Dimension 2 Code fields (or the first two shortcut dimensions) for a record by reading the values from the specified dimension set. This function is typically called from the UpdateAllLineDim function in document header tables.
LookupDimValueCode	Shows the available dimension values for the specified dimension to let the user select a value for a shortcut dimension field. This function is called from the LookupShortcutDimCode function on document line tables.

Module 8: Dimensions

Function	Remarks
GetShortcutDimensions	<p>Populates the array of values for eight shortcut dimensions from a dimension set ID.</p> <p>It is called from the ShowShortcutDimCode function of document line tables.</p>
SetupObjectNoList	<p>Populates the list of all account types (master record types) in the application. This lets users select the table type in the Account Type Default Dim page.</p> <p> Note: Whenever you add new master record types, you must customize this function to enable users to define default account type dimensions on your custom master tables.</p>
CheckDimIDComb	<p>Checks dimension value combinations of a dimension that is set for invalid combinations according to the configuration in the Dimension Combination and Dimension Value Combination tables.</p> <p>This function is called from Jnl.-Check Line codeunits of journal posting routines, and Post codeunits of document posting routines.</p>
CheckDimValuePosting	<p>Checks dimension values of a dimension set against the Value Posting field settings for default dimensions of master records that are involved in the posting.</p> <p>This function is called from Jnl.-Check Line codeunits of journal posting routines, and Post codeunits of document posting routines.</p>

Function	Remarks
GetDimCombErr	Returns the message that describes the dimension combination error. This function is called from Jnl.-Check Line codeunits of journal posting routines, and Post codeunits of document posting routines.
GetDimValuePostingErr	Returns the message that describes the dimension posting value error. This function is called from Jnl.-Check Line codeunits of journal posting routines and Post codeunits of document posting routines.

Journal and Document Posting Codeunits

Several codeunits that are connected with posting routines perform specific dimension-related tasks and call functions of the DimensionManagement codeunit.

The following table explains the dimension management features of posting-routine codeunits.

Codeunit	Dimension Management Features
Jnl.-Check Line	Calls the CheckDimIDComb and CheckDimValuePosting functions of the DimensionManagement codeunit on the dimension set of the journal line.
Jnl.-Post Line	Copies values of Shortcut Dimension 1 Code , Shortcut Dimension 2 Code , and Dimension Set ID fields from the journal line table to Global Dimension 1 Code , Global Dimension 2 Code , and Dimension Set ID fields in the ledger entry table.

Codeunit	Dimension Management Features
Post	<p>Calls the CheckDimIDComb and CheckDimValuePosting functions of the DimensionManagement codeunit on the dimension sets of the document header and document lines.</p> <p>Copies values of Shortcut Dimension 1 Code, Shortcut Dimension 2 Code, and Dimension Set ID fields from the document lines to the appropriate journal lines, before it starts the journal posting routine for the journal line.</p>

Integrating Seminar Management with Dimensions

In Microsoft Dynamics NAV 2013, dimensions serve two purposes: they enable the expansion of the existing data model with additional attributes that meet a company's business need for simple multi-dimensional analysis of data; and they support simple data-driven business rules that relate to data validity during posting.

When you extend the application by using new application areas or functionality such as master records, documents, or journals, you should take advantage of the standard dimension management functionality to give users a consistent user experience.

You have already developed many features for the new Seminar Management application area. You now must integrate the dimension management features with Seminar Management functionality.

Solution Design

The CRONUS International Ltd. functional requirements do not include specific details about dimensions. The legacy application that they used prior to the implementation of Microsoft Dynamics NAV 2013 has no similar functionality, and CRONUS International Ltd. is unfamiliar with the concept of dimensions.

However, two of the following functional requirements for CRONUS International Ltd. can apply to dimensions:

- The solution must use the standard functionality of Microsoft Dynamics NAV 2013 wherever possible. The solution must not duplicate functions that are already present in the application or cause redundant functionality.

- The solution must be consistent, user-friendly, easy to learn, and to use. Any custom-built functionality must follow the standards, principles, and best practices of Microsoft Dynamics NAV 2013, and must seamlessly integrate into the standard application.



Best Practice: Whether your customers explicitly ask for standard Microsoft Dynamics NAV 2013 functionality in a custom-built module, or if any nonfunctional requirements request full dependency on standard functionality, you must always provide a level of integration with standard features that is consistent with any other functionally similar application area. Even though users might not know what dimensions are before they implement Microsoft Dynamics NAV 2013, they will find dimension functionality elsewhere in the application. Any functionality that you develop must not be an exception.

Solution Development

To provide a consistent user experience and to integrate with dimension management features of standard Microsoft Dynamics NAV 2013, you must change many of the objects that you have previously developed, including tables, pages, and codeunits. You must extend both your data model to integrate with the standard dimension model, and your application features to integrate with the standard dimension process model, as explained in the Prerequisite Knowledge lesson.

The following table summarizes the changes that you must make to the existing standard and custom objects.

Object	Summary of Changes
Table 352, Default Dimension	<ul style="list-style-type: none">Add a new function to update the global dimension code on a seminar record.Change the UpdateGlobalDimCode function to call the new function when it is necessary.
Table 123456700 Seminar	<ul style="list-style-type: none">Add Global Dimension 1 Code and Global Dimension 2 Code fields.Add the ValidateShortcutDimCode function to validate a user's entry and to store the default dimension values.Change the OnInsert and OnDelete triggers to call DimensionManagement codeunit where you can.

Module 8: Dimensions

Object	Summary of Changes
Table 123456710 Seminar Registration Header	<ul style="list-style-type: none"> • Add Shortcut Dimension 1 Code, Shortcut Dimension 2 Code, and Dimension Set ID fields. • Add the SeminarRegLinesExist function. • Add the CreateDim function. • Add the ValidateShortcutDimCode function. • Add the ShowDocDim function. • Add the UpdateAllLineDim function.
Table 123456711 Seminar Registration Line	<ul style="list-style-type: none"> • Add Shortcut Dimension 1 Code, Shortcut Dimension 2 Code, and Dimension Set ID fields. • Add the ShowDimensions function. • Add the CreateDim function. • Add the ValidateShortcutDimCode function. • Add the LookupShortcutDimCode function. • Add the ShowShortcutDimCode function.
Table 123456718 Posted Seminar Reg. Header	<ul style="list-style-type: none"> • Add Shortcut Dimension 1 Code, Shortcut Dimension 2 Code, and Dimension Set ID fields. • Add the ShowDimensions function.
Table 123456719 Posted Seminar Reg. Line	<ul style="list-style-type: none"> • Add Shortcut Dimension 1 Code, Shortcut Dimension 2 Code, and Dimension Set ID fields. • Add the ShowDimensions function.
Table 123456731 Seminar Journal Line	<ul style="list-style-type: none"> • Add Shortcut Dimension 1 Code, Shortcut Dimension 2 Code, and Dimension Set ID fields.

Object	Summary of Changes
Table 123456732 Seminar Ledger Entry	<ul style="list-style-type: none"> • Add Shortcut Dimension 1 Code, Shortcut Dimension 2 Code, and Dimension Set ID fields. • Add the ShowDimensions function.
Page 542 Default Dimensions-Multiple	<ul style="list-style-type: none"> • Add a new function to select default dimensions for specified seminar records.
Page 123456700 Seminar Card	<ul style="list-style-type: none"> • Add the Dimensions action to the Navigate tab of the ribbon.
Page 123456701 Seminar List	<ul style="list-style-type: none"> • Add the Dimensions-Single and Dimensions-Multiple actions to the Navigate tab of the ribbon.
Page 123456710 Seminar Registration	<ul style="list-style-type: none"> • Add the Dimensions action to the Navigate tab of the ribbon.
Page 123456711 Seminar Registration Subform	<ul style="list-style-type: none"> • Add the Dimensions action to the Line action group. • Add field controls for the eight shortcut dimensions. • Change the OnAfterGetRecord and OnNewRecord page triggers. • Change the OnValidate trigger for the Bill-to Customer No. field control.
Page 123456713 Seminar Registration List	<ul style="list-style-type: none"> • Add the Dimensions action to the Navigate tab of the ribbon.
Page 123456721 Seminar Ledger Entries	<ul style="list-style-type: none"> • Add the Dimensions action to the Navigate tab of the ribbon.
Page 123456734 Posted Seminar Registration	<ul style="list-style-type: none"> • Add the Dimensions action to the Navigate tab of the ribbon.
Page 123456735 Posted Seminar Reg. Subform	<ul style="list-style-type: none"> • Add the Dimensions action to the Line action group.

Module 8: Dimensions

Object	Summary of Changes
Page 123456736 Posted Seminar Reg. List	<ul style="list-style-type: none"> • Add the Dimensions action to the Navigate tab of the ribbon.
Codeunit 123456700 Seminar-Post	<ul style="list-style-type: none"> • Add the CheckDim function. • Add the CheckDimComb function. • Add the CheckDimValuePosting function. • Copy the values from the Shortcut Dimension 1 Code, Shortcut Dimension 2 Code, and Dimension Set ID fields from the document lines to the appropriate journal lines.
Codeunit 123456731 Seminar Jnl.-Check Line	<ul style="list-style-type: none"> • Call the CheckDimIDComb and CheckDimValuePosting functions of the DimensionManagement codeunit.
Codeunit 123456732 Seminar Jnl.-Post Line	<ul style="list-style-type: none"> • Copy the values from the Shortcut Dimension 1 Code, Shortcut Dimension 2 Code, and Dimension Set ID fields from the journal line to the Global Dimension 1 Code, Global Dimension 2 Code, and Dimension Set ID fields in the ledger entry.



Note: When you develop the customizations in this chapter's lab, you implement some of the changes that are specified in the previous table and import several changes from external files.

Lab 8.1: Integrating with Dimension Management

Scenario

Viktor is the business systems developer who works for the Microsoft Dynamics partner company that is implementing Microsoft Dynamics NAV 2013 for CRONUS International Ltd. His task is to integrate existing seminar management module features with dimension management.

Viktor reuses existing code from other objects. He makes sure to extend the master records, documents, journal and the ledger entry tables of the Seminar Management module by mimicking the functionality that he finds in the **Customer, Sales Header, Sales Line, Res. Journal Line, and Res. Ledger Entry** tables, the **Sales Invoice, Posted Sales Invoice, and Resource Ledger Entries** pages, and the Res. Jnl.-Check Line, Res. Jnl-Post Line, and Sales Post codeunits.

Because there are many objects that require customization, Isaac helps Viktor by customizing a subset of the Seminar Management objects. Isaac also prepares the set of object files that Viktor will import and review later.

 **Note:** Even though the labs contain the steps to develop most of the dimension management functionality in Sales Management module from the ground up, you may take a look at the objects that are listed here to see how they integrate with the dimension management features of the standard Microsoft Dynamics NAV 2013 application.

Exercise 1: Extending Master Data with Dimensions

Exercise Scenario

Viktor starts by changing several standard objects to make them ready for integration of the Seminar Management module with dimension management features. Then he extends the master tables and pages by using required dimension management features.

Task 1: Modify the DimensionManagement Codeunit

High Level Steps

1. Add the Seminar option to the list of available account types, in the **SetupObjectNoList** function of the DimensionManagement codeunit.

Detailed Steps

1. Add the Seminar option to the list of available account types, in the **SetupObjectNoList** function of the DimensionManagement codeunit.
 - a. Design codeunit 408 DimensionManagement.
 - b. Locate the **SetupObjectNoList** function.
 - c. In the local variables for the function, on the TableIDArray variable set the Dimensions property to 100.
 - d. In the SetupObjectNoList function trigger, at the end of the block where TableIDArray values are assigned and before the Object.SETRANGE statement, enter the following code.

```
//CSD1.00>  
  
TableIDArray[100] := DATABASE::Seminar;  
  
//CSD1.00<
```

- e. Compile, save, and then close the codeunit.

Task 2: Modify the Seminar Table

High Level Steps

1. Add the **ValidateShortcutDimCode** function to the **Seminar** table to call the **ValidateDimValueCode** and **SaveDefaultDim** functions of the DimensionManagement codeunit.
2. Add the **Global Dimension 1 Code** and **Global Dimension 2 Code** fields to the table 123456700 **Seminar**. Make sure that you establish correct table relations to the **Dimension Value** table.
3. Add code to the OnInsert and OnDelete triggers to call the UpdateDefaultDim and DeleteDefaultDim triggers of the DimensionManagement codeunit.

Detailed Steps

1. Add the **ValidateShortcutDimCode** function to the **Seminar** table to call the **ValidateDimValueCode** and **SaveDefaultDim** functions of the DimensionManagement codeunit.
 - a. Design table 123456700, **Seminar**.
 - b. Add a new function and name it **ValidateShortcutDimCode**. The function receives the following parameters.

Var	Name	DataType	Length
No	FieldNumber	Integer	
Yes	ShortcutDimCode	Code	20

- c. Add a new global variable for the DimensionManagement codeunit, and name it DimMgt.
- d. In the ValidateShortcutDimCode function trigger, enter the following C/AL code.

```
DimMgt.ValidateDimValueCode(FieldNumber,ShortcutDimCode);
DimMgt.SaveDefaultDim(DATABASE::Customer,"No.",FieldNumber,ShortcutDimCode);
MODIFY;
```

2. Add the **Global Dimension 1 Code** and **Global Dimension 2 Code** fields to the table 123456700 **Seminar**. Make sure that you establish correct table relations to the **Dimension Value** table.
 - a. Add the following fields.

Field No.	Field Name	Data Type	Length
21	Global Dimension 1 Code	Code	20
22	Global Dimension 2 Code	Code	20

- b. Set the TableRelation property on the **Global Dimension 1 Code** field to the **Code** field of the **Dimension Value** table, where **Global Dimension No.** is CONST 1.



Note: You can use the Table Relation window to set the relation in a more user-friendly way, or you may set the property to the following text: "Dimension Value".Code WHERE (Global Dimension No.=CONST(1))

Module 8: Dimensions

- c. Set the TableRelation property on the **Global Dimension 2 Code** field to the **Code** field of the **Dimension Value** table, where **Global Dimension No.** is CONST 2.



Note: "Dimension Value".Code WHERE (Global Dimension No.=CONST(2))

- d. Set the CaptionClass property on the following fields.

Field	CaptionClass
Global Dimension 1 Code	'1,1,1'
Global Dimension 2 Code	'1,1,2'

3. Add code to the OnInsert and OnDelete triggers to call the UpdateDefaultDim and DeleteDefaultDim triggers of the DimensionManagement codeunit.
- a. At the end of the OnInsert trigger, add the following code.

```
DimMgt.UpdateDefaultDim(  
    DATABASE::Seminar,"No."  
    "Global Dimension 1 Code","Global Dimension 2 Code");
```

- b. At the end of the OnDelete trigger, add the following code.

```
DimMgt.DeleteDefaultDim(DATABASE::Seminar,"No.");
```

- c. Compile, save, and then close the table.

Task 3: Modify the Default Dimension Table

High Level Steps

1. Add a new function to the table 352, **Default Dimension**, and name it **UpdateSeminarGlobalDimCode**. This function assigns the value to the **Global Dimension 1 Code** and **Global Dimension 2 Code** fields of the specified **Seminar** record.
2. Add code to the **UpdateGlobalDimCode** function to call the **UpdateSeminarGlobalDimCode** function when you can.

Detailed Steps

1. Add a new function to the table 352, **Default Dimension**, and name it **UpdateSeminarGlobalDimCode**. This function assigns the value to the **Global Dimension 1 Code** and **Global Dimension 2 Code** fields of the specified **Seminar** record.
 - a. Design table 352, **Default Dimension**.
 - b. In the **C/AL Globals** window for the table, create a new function, and name it **UpdateSeminarGlobalDimCode**.
 - c. Define the following parameters on the **UpdateSeminarGlobalDimCode** function.

Name	DataType	Length
GlobalDimCodeNo	Integer	
SeminarNo	Code	20
NewDimValue	Code	20

- d. Define a local record variable for the function. Set the its Subtype to Seminar.
- e. In the function trigger for the function, enter the following code.

```
//CSD1.00>

IF Seminar.GET(SeminarNo) THEN BEGIN

CASE GlobalDimCodeNo OF

1:
    Seminar."Global Dimension 1 Code" := NewDimValue;

2:
    Seminar."Global Dimension 2 Code" := NewDimValue;

END;
END;

//CSD1.00<
```

2. Add code to the **UpdateGlobalDimCode** function to call the **UpdateSeminarGlobalDimCode** function when you can.
 - a. In the function trigger of the **UpdateGlobalDimCode** function, within the CASE block after the last case, enter the following code.

```
//CSD1.00>

DATABASE::Seminar:

UpdateSeminarGlobalDimCode(GlobalDimCodeNo,"No.",NewDimValue);

//CSD1.00<
```

- b. Compile, save, and then close the table.

Task 4: Import the Default Dimensions-Multiple Page

High Level Steps

1. Import the Mod08\Labfiles\Lab 8.A - Starter - Page 542.fob object file.

Detailed Steps

1. Import the Mod08\Labfiles\Lab 8.A - Starter - Page 542.fob object file.
 - a. In Object Designer, click **File > Import**.
 - b. Browse to the Mod08\Labfiles\Lab 8.A - Starter - Page 542.fob file, and then click **Open**.
 - c. Click **Yes** to complete the import.
 - d. Click **OK** to complete the object import.
 - e. In Object Designer, click **File > Import**.
 - f. Browse to the Mod08\Labfiles\Lab 8.A - Starter - Page 542.fob file, and then click **Open**.
 - g. Click **Yes** to complete the import.
 - h. Click **OK** to complete the object import.

Task 5: Modify the Seminar Card and Seminar List Pages

High Level Steps

1. Add the Dimensions action to the **Navigate** tab on the **Seminar Card** page.
2. Add the Dimensions action group to the **Navigate** tab on the **Seminar List** page.
3. Add the Dimensions-Single and Dimensions-Multiple actions to the Dimensions action group on the **Seminar List** page.

Detailed Steps

1. Add the Dimensions action to the **Navigate** tab on the **Seminar Card** page.
 - a. Design page 123456700, **Seminar Card**.
 - b. In the Action Designer for the page, under the Seminar action group, add a new action, and set its caption to "Dimensions".
 - c. Set the following properties on the new action.

Property	Value
Image	Dimensions
ShortCutKey	Shift+Ctrl+D
RunObject	Page Default Dimensions
RunPageLink	Table ID=CONST(123456700),No.=FIELD(No.)

- d. Compile, save, and then close the page.
2. Add the Dimensions action group to the **Navigate** tab on the **Seminar List** page.
 - a. Design page 123456701, **Seminar List**.
 - b. In the Action Designer for the page, under the Seminar action group, add a new action group, and set its caption to "Dimensions".
3. Add the Dimensions-Single and Dimensions-Multiple actions to the Dimensions action group on the **Seminar List** page.
 - a. Under the Seminar action group, add a new action, and set its caption to "Dimensions-Single".
 - b. Set the following properties on the Dimensions-Single action.

Property	Value
Image	Dimensions
ShortCutKey	Shift+Ctrl+D
RunObject	Page Default Dimensions
RunPageLink	Table ID=CONST(123456700),No.=FIELD(No.)

- c. Under the Seminar action group, add a new action, and set its caption to "Dimensions-Multiple".

Module 8: Dimensions

- d. In the OnAction trigger for the Dimensions-Multiple action, define the following local variables.

Name	DataType	Subtype
Seminar	Record	Seminar
DefaultDimMultiple	Page	Default Dimensions-Multiple

- e. In the OnAction trigger for the Dimensions-Multiple action, enter the following code.

```
CurrPage.SETSELECTIONFILTER(Seminar);  
  
DefaultDimMultiple.SetMultiSeminar(Seminar);  
  
DefaultDimMultiple.RUNMODAL;
```

- f. Set the Image property on the Dimensions-Multiple action to "DimensionSets".
g. Compile, save, and then close the page.

Exercise 2: Extending Documents with Dimensions

Exercise Scenario

Viktor extends the seminar registration tables and pages with required dimension management functionality. He then imports the posted seminar registration tables and pages from an external file that was prepared by Isaac, another developer working on the implementation project for CRONUS International Ltd.

Task 1: Modify the Seminar Registration Header and Line Tables

High Level Steps

1. In the Seminar Registration Line table, add the **Shortcut Dimension 1 Code** and **Shortcut Dimension 2 Code**, and **Dimension Set ID** fields.
2. Add a global variable for the DimensionManagement codeunit.
3. Add the **ShowDimensions** function, and call it from the appropriate trigger.
4. Add the **CreateDim** function and call it from the appropriate trigger.
5. Add the **ValidateShortcutDimCode** function, and call it from the appropriate triggers.
6. Add the **LookupShortcutDimCode** function.
7. Add the **ShowShortcutDimCode** function.

8. In the **Seminar Registration Header** table, add the **Shortcut Dimension 1 Code** and **Shortcut Dimension 2 Code**, and **Dimension Set ID** fields.
9. In the **Seminar Registration Header** table, add a global variable for the DimensionManagement codeunit.
10. In the **Seminar Registration Header** table, add the **SeminarRegLinesExist** function.
11. Add the **CreateDim** function and call it from the appropriate trigger.
12. Add the **ValidateShortcutDimCode** function, and call it from the appropriate triggers.
13. Add the **ShowDocDim** function, and call it from the appropriate trigger.
14. Add the **UpdateAllLineDim** function.
15. Compile the Seminar Registration Line table.

Detailed Steps

1. In the Seminar Registration Line table, add the **Shortcut Dimension 1 Code** and **Shortcut Dimension 2 Code**, and **Dimension Set ID** fields.
 - a. Design table 123456711, **Seminar Registration Line**.
 - b. Add the following fields to the table.

Field No.	Field Name	Data Type	Length
51	Shortcut Dimension 1 Code	Code	20
52	Shortcut Dimension 2 Code	Code	20
480	Dimension Set ID	Integer	

- c. Set the TableRelation property on the shortcut dimension fields to relate to the **Code** field of the matching record in the **Dimension Value** table.

 **Note:** You may copy the values from the same property on the **Global Dimension 1 Code** and **Global Dimension 2 Code** fields in the **Seminar** table.

Otherwise, you may enter "Dimension Value".Code WHERE (Global Dimension No.=CONST(1)). Make sure that you replace "1" with "2" for the **Shortcut Dimension 2 Code** field.

- d. Set the TableRelation property on the **Dimension Set ID** field to the **Dimension Set Entry** table.

Module 8: Dimensions

2. Add a global variable for the DimensionManagement codeunit.
 - a. In the **C/AL Globals** window, define a new variable, and name it DimMgt. Set Type to Codeunit, and subtype to DimensionManagement.
3. Add the **ShowDimensions** function, and call it from the appropriate trigger.
 - a. In the C/AL Globals window, create a new function, and name it **ShowDimensions**.
 - b. In the function trigger code for the function, enter the following code.

```
"Dimension Set ID" :=  
  
DimMgt.EditDimensionSet(  
  
    "Dimension Set ID",  
  
    STRSUBSTNO('%1 %2',  
  
    "Document No.",  
  
    "Line No.");  
  
DimMgt.UpdateGlobalDimFromDimSetID(  
  
    "Dimension Set ID",  
  
    "Shortcut Dimension 1 Code",  
  
    "Shortcut Dimension 2 Code");
```

- c. In the OnLookup trigger for the **Dimension Set ID** field, enter the following code.

```
ShowDimensions;
```

4. Add the **CreateDim** function and call it from the appropriate trigger.
 - a. In the **C/AL Globals** window, create a new function, and name it **CreateDim**.
 - b. Define the following parameters for the **CreateDim** function.

Name	DataType	Length
Type1	Integer	
No1	Code	20

C/SIDE Solution Development in Microsoft Dynamics® NAV 2013

- c. Define the following local variables for the **CreateDim** function.

Name	DataType	Subtype	Length
SourceCodeSetup	Record	Source Code Setup	
TableID	Integer		
No	Code		20

- d. Set the Dimensions property on the *TableID* and *No* local variables to 10.
e. In the function trigger for the **CreateDim** function, enter the following code.

```
SourceCodeSetup.GET;  
  
TableID[1] := Type1;  
  
No[1] := No1;  
  
"Shortcut Dimension 1 Code" := ";"  
  
"Shortcut Dimension 2 Code" := ";"  
  
GetSeminarRegHeader;  
  
"Dimension Set ID" :=  
  
DimMgt.GetDefaultDimID(  
  
TableID,No,SourceCodeSetup.Seminar,  
  
"Shortcut Dimension 1 Code",  
  
"Shortcut Dimension 2 Code",  
  
SeminarRegHeader."Dimension Set ID",  
  
DATABASE::Seminar);  
  
DimMgt.UpdateGlobalDimFromDimSetID(  
  
"Dimension Set ID",  
  
"Shortcut Dimension 1 Code",  
  
"Shortcut Dimension 2 Code");
```

- f. At the end of the OnValidate trigger for the **Bill-to Customer No.** field, append the following code.

Module 8: Dimensions

```
CreateDim(DATABASE::Customer,"Bill-to Customer No.");
```

5. Add the **ValidateShortcutDimCode** function, and call it from the appropriate triggers.
 - a. In the **C/AL Globals** window, create a new function, and name it **ValidateShortcutDimCode**.
 - b. Define the following parameters for the **ValidateShortcutDimCode** function.

Var	Name	DataType	Length
	FieldNumber	Integer	
Yes	ShortcutDimCode	Code	20

- c. In the function trigger for the **ValidateShortcutDimCode** function, enter the following code.

```
DimMgt.ValidateShortcutDimValues(  
    FieldNumber,  
    ShortcutDimCode,  
    "Dimension Set ID");
```

- d. In the OnValidate trigger for the **Shortcut Dimension 1 Code** field, enter the following code.

```
ValidateShortcutDimCode(1,"Shortcut Dimension 1 Code");
```

- e. In the OnValidate trigger for the **Shortcut Dimension 2 Code** field, enter the following code.

```
ValidateShortcutDimCode(2,"Shortcut Dimension 2 Code");
```

6. Add the **LookupShortcutDimCode** function.
 - a. In the **C/AL Globals** window, create a new function, and name it **LookupShortcutDimCode**.
 - b. Define the following parameters for the **LookupShortcutDimCode** function.

Var	Name	DataType	Length
	FieldNumber	Integer	
Yes	ShortcutDimCode	Code	20

- c. In the function trigger for the **LookupShortcutDimCode** function, enter the following code.

```
DimMgt.LookupDimValueCode(  
    FieldNumber,  
    ShortcutDimCode);  
  
ValidateShortcutDimCode(  
    FieldNumber,  
    ShortcutDimCode);
```

7. Add the **ShowShortcutDimCode** function.
 - a. In the **C/AL Globals** window, create a new function, and name it **ShowShortcutDimCode**.
 - b. Define the following parameters for the **ShowShortcutDimCode** function.

Var	Name	DataType	Length
Yes	ShortcutDimCode	Code	20

- c. Set the Dimensions property on the ShortcutDimCode parameter to 8.
- d. In the function trigger for the **ShowShortcutDimCode** function, enter the following code.

```
DimMgt.GetShortcutDimensions(  
    "Dimension Set ID",  
    ShortcutDimCode);
```

- e. Save the table without compiling, and then close the Table Designer.

Module 8: Dimensions

8. In the **Seminar Registration Header** table, add the **Shortcut Dimension 1 Code** and **Shortcut Dimension 2 Code**, and **Dimension Set ID** fields.

- Design table 123456710, **Seminar Registration Header**.
- Add the following fields to the table.

Field No.	Field Name	Data Type	Length
51	Shortcut Dimension 1 Code	Code	20
52	Shortcut Dimension 2 Code	Code	20
480	Dimension Set ID	Integer	

- Set the TableRelation property on the shortcut dimension fields to relate to the **Code** field of the matching record in the **Dimension Value** table.
- Set the TableRelation property on the **Dimension Set ID** field to the **Dimension Set Entry** table.

9. In the **Seminar Registration Header** table, add a global variable for the DimensionManagement codeunit.

- In the **C/AL Globals** window, define a new variable, and name it DimMgt. Set Type to Codeunit, and subtype to DimensionManagement.

10. In the **Seminar Registration Header** table, add the **SeminarRegLinesExist** function.

- Create a new function, name it **SeminarRegLinesExist**.
- On the **Return Value** tab of the **C/AL Locals** window for the function, set the Return Type to Boolean.
- In the function trigger for the **SeminarRegLinesExist** function, enter the following code.

```
SeminarRegLine.RESET;  
  
SeminarRegLine.SETRANGE("Document No.", "No.");  
  
EXIT(SeminarRegLine.FINDFIRST);
```

C/SIDE Solution Development in Microsoft Dynamics® NAV 2013

11. Add the **CreateDim** function and call it from the appropriate trigger.
 - a. In the **C/AL Globals** window, create a new function, and name it **CreateDim**.
 - b. Define the following parameters for the **CreateDim** function.

Name	DataType	Length
Type1	Integer	
No1	Code	20
Type2	Integer	
No2	Code	20
Type3	Integer	
No3	Code	20

- c. Define the following local variables for the **CreateDim** function.

Name	DataType	Subtype	Length
SourceCodeSetup	Record	Source Code Setup	
TableID	Integer		
No	Code		20
OldDimSetID	Integer		

- d. Set the Dimensions property on the *TableID* and *No* local variables to 10.
- e. In the function trigger for the **CreateDim** function, enter the following code.

```
SourceCodeSetup.GET;  
  
TableID[1] := Type1;  
  
No[1] := No1;  
  
TableID[2] := Type2;  
  
No[2] := No2;  
  
TableID[3] := Type3;  
  
No[3] := No3;  
  
"Shortcut Dimension 1 Code" := ";
```

```
"Shortcut Dimension 2 Code" := ";

OldDimSetID := "Dimension Set ID";

"Dimension Set ID" :=

DimMgt.GetDefaultDimID(TableID,No,

SourceCodeSetup.Seminar,

"Shortcut Dimension 1 Code",

"Shortcut Dimension 2 Code",0,0);

IF (OldDimSetID <> "Dimension Set ID") AND

SeminarRegLinesExist

THEN BEGIN

MODIFY;

UpdateAllLineDim("Dimension Set ID",OldDimSetID);

END;
```

- f. At the end of the OnValidate trigger for the **Seminar No.** field, append the following code.

```
CreateDim(
DATABASE::Seminar,"Seminar No.",

DATABASE::Resource,"Instructor Resource No.",

DATABASE::Resource,"Room Resource No.");
```

- g. At the end of the OnValidate trigger for the **Instructor Resource No.** field, append the following code.

```
CreateDim(
DATABASE::Seminar,"Seminar No.",

DATABASE::Resource,"Instructor Resource No.",

DATABASE::Resource,"Room Resource No.");
```

C/SIDE Solution Development in Microsoft Dynamics® NAV 2013

- h. At the end of the OnValidate trigger for the **Room Resource No.** field, append the following code.

```
CreateDim(  
    DATABASE::Seminar,"Seminar No.",  
    DATABASE::Resource,"Instructor Resource No.",  
    DATABASE::Resource,"Room Resource No.");
```

12. Add the **ValidateShortcutDimCode** function, and call it from the appropriate triggers.
 - a. In the **C/AL Globals** window, create a new function, and name it **ValidateShortcutDimCode**.
 - b. Define the following parameters for the **ValidateShortcutDimCode** function.

Var	Name	DataType	Length
	FieldNumber	Integer	
Yes	ShortcutDimCode	Code	20

- c. Define a local integer variable for the **ValidateShortcutDimCode** function, and name it **OldDimSetID**.
- d. In the function trigger for the **ValidateShortcutDimCode** function, enter the following code.

```
OldDimSetID := "Dimension Set ID";  
  
DimMgt.ValidateShortcutDimValues(  
    FieldNumber,  
    ShortcutDimCode,  
    "Dimension Set ID");  
  
IF "No." <> " THEN  
  
    MODIFY;  
  
IF OldDimSetID <> "Dimension Set ID" THEN BEGIN  
  
    MODIFY;  
  
    IF SeminarRegLinesExist THEN
```

```

UpdateAllLineDim(
    "Dimension Set ID",
    OldDimSetID);
END;

```

- e. In the OnValidate trigger for the **Shortcut Dimension 1 Code** field, enter the following code.

```
ValidateShortcutDimCode(1,"Shortcut Dimension 1 Code");
```

- f. In the OnValidate trigger for the **Shortcut Dimension 2 Code** field, enter the following code.

```
ValidateShortcutDimCode(2,"Shortcut Dimension 2 Code");
```

13. Add the **ShowDocDim** function, and call it from the appropriate trigger.
- In the **C/AL Globals** window, create a new function, and name it **ShowDocDim**.
 - Define a local integer variable for the **ShowDocDim** function, and name it OldDimSetID.
 - In the function trigger of the **ShowDocDim** function, enter the following code.

```

OldDimSetID := "Dimension Set ID";
"Dimension Set ID" :=

DimMgt.EditDimensionSet2(
    "Dimension Set ID","No.",
    "Shortcut Dimension 1 Code",
    "Shortcut Dimension 2 Code");

IF OldDimSetID <> "Dimension Set ID" THEN BEGIN
    MODIFY;
    IF SeminarRegLinesExist THEN
        UpdateAllLineDim(
            "Dimension Set ID",

```

```
OldDimSetID);
```

```
END;
```

- d. In the OnLookup trigger for the **Dimension Set ID** field, enter the following code.

```
ShowDocDim;
```

14. Add the **UpdateAllLineDim** function.

- In the **C/AL Globals** window, create a new function, and name it **UpdateAllLineDim**.
- Define the following parameters for the **UpdateAllLineDim** function.

Name	DataType
NewParentDimSetID	Integer
OldParentDimSetID	Integer

- Define a local integer variable for the **UpdateAllLineDim** function, and name it NewDimSetID.
- Define a global text constant, name it Text009, and enter the following ConstValue for it: "You may have changed a dimension.\Do you want to update the lines?"
- In the function trigger for the function, enter the following code.

```
IF NewParentDimSetID = OldParentDimSetID THEN  
    EXIT;  
  
IF NOT CONFIRM(Text009) THEN  
    EXIT;  
  
SeminarRegLine.RESET;  
  
SeminarRegLine.SETRANGE("Document No.", "No.");  
  
SeminarRegLine.LOCKTABLE;  
  
IF SeminarRegLine.FIND('-') THEN  
    REPEAT  
        NewDimSetID := DimMgt.GetDeltaDimSetID(
```

```
SeminarRegLine."Dimension Set ID",  
  
NewParentDimSetID,  
  
OldParentDimSetID);  
  
IF SeminarRegLine."Dimension Set ID" <> NewDimSetID  
  
THEN BEGIN  
  
    SeminarRegLine."Dimension Set ID" := NewDimSetID;  
  
    DimMgt.UpdateGlobalDimFromDimSetID(  
  
        SeminarRegLine."Dimension Set ID",  
  
        SeminarRegLine."Shortcut Dimension 1 Code",  
  
        SeminarRegLine."Shortcut Dimension 2 Code");  
  
    SeminarRegLine.MODIFY;  
  
END;  
  
UNTIL SeminarRegLine.NEXT = 0;
```

- f. Compile, save, and then close the table.
15. Compile the Seminar Registration Line table.
 - a. In Object Designer, select the table **123456711, Seminar Registration Line**.
 - b. Click **Tools > Compile** (or press F11).

Task 2: Modify the Seminar Registration Pages

High Level Steps

1. Add the Dimensions action to the **Seminar Registration** and **Seminar Registration List** pages.
2. Add field controls for eight shortcut dimensions to the **Seminar Registration Subform** page.
3. Add the Dimension actions to the Line action group.
4. Modify the OnAfterGetRecord and OnNewRecord page triggers.
5. Modify the OnValidate trigger for the **Bill-to Customer No.** field.

Detailed Steps

1. Add the Dimensions action to the **Seminar Registration** and **Seminar Registration List** pages.
 - a. Design page 123456710, **Seminar Registration**.
 - b. In the Seminar Registration action group, add a new page action, set its caption to "Dimensions", and set the following properties.

Property	Value
Image	Dimensions
ShortCutKey	Shift+Ctrl+D

- c. In the OnAction trigger for the Dimensions action, enter the following code.

```
ShowDocDim;  
  
CurrPage.SAVERECORD;
```

- d. Compile, save, and then close the **Seminar Registration** page.
- e. Design page 123456713, **Seminar Registration List**.
- f. In the Seminar Registration action group, add a new page action, set its caption to "Dimensions", and set the following properties.

Property	Value
Image	Dimensions
ShortCutKey	Shift+Ctrl+D

- g. In the OnAction trigger for the Dimensions action, enter the following code.

```
ShowDocDim;
```

- h. Compile, save, and then close the **Seminar Registration List** page.

2. Add field controls for eight shortcut dimensions to the **Seminar Registration Subform** page.
 - a. Design page 123456711, **Seminar Registration Subform**.
 - b. Create the ShortcutDimCode global variable of type Code and length 20, and set its Dimensions property to 8.

Module 8: Dimensions

- c. Design page 46, **Sales Order Subform**.
- d. Copy to the clipboard all field controls with the following SourceExpr values: "Shortcut Dimension 1 Code", "Shortcut Dimension 2 Code", ShortcutDimCode[3], ShortcutDimCode[4], ShortcutDimCode[5], ShortcutDimCode[6], ShortcutDimCode[7], ShortcutDimCode[8].



Note: Select the specified controls, then press CTRL+C.

- e. Close the **Sales Order Subform** page.
- f. In the page designer for the **Seminar Registration Subform** page, at the first empty line, paste the field controls from the clipboard (press CTRL+V).
3. Add the Dimension actions to the Line action group.
 - a. In the Action Designer of the **Seminar Registration Subform** page, define the following actions.

Type	SubType	Caption
ActionContainer	ActionItems	<Action9>
ActionGroup		&Line
Action		Dimensions

- b. For the Dimensions action, define the following properties.

Property	Value
Image	Dimensions
ShortCutKey	Shift+Ctrl+D

- c. In the OnAction trigger for the Dimensions action, enter the following code.

```
ShowDimensions;
```

4. Modify the OnAfterGetRecord and OnNewRecord page triggers.
 - a. In the OnAfterGetRecord trigger, enter the following code.

```
ShowShortcutDimCode(ShortcutDimCode);
```

- b. In the OnNewRecord trigger, enter the following code.

```
CLEAR(ShortcutDimCode);
```

5. Modify the OnValidate trigger for the **Bill-to Customer No.** field.
 - a. In the OnValidate trigger for the **Bill-to Customer No.** field, enter the following code.

```
ShowShortcutDimCode(ShortcutDimCode);
```

- b. Compile, save, and then close the **Seminar Registration Subform** page.

Task 3: Import Posted Seminar Registration Tables and Pages

High Level Steps

1. Import the Mod08\Labfiles\Lab 8.A - Starter.fob object file.

Detailed Steps

1. Import the Mod08\Labfiles\Lab 8.A - Starter.fob object file.
 - a. In Object Designer, click **File > Import**.
 - b. Browse to the Mod08\Labfiles\Lab 8.A - Starter.fob file, and then click **Open**.
 - c. Click **OK** to open the Import Worksheet.
 - d. Click **Replace All**, and then click **OK**.
 - e. Click **OK** to complete the object import.

Exercise 3: Extending Ledger Entries and Posting Process with Dimensions

Exercise Scenario

Viktor extends the journal posting routines and imports the changes that were made by Isaac to ledger tables and pages and the document posting routine.

Task 1: Modify the Seminar Jnl.-Check Line Codeunit

High Level Steps

1. In the Seminar Jnl.-Check Line codeunit, enter code to call the **CheckDimIDComb** and **CheckDimValuePosting** functions of the DimensionManagement codeunit.

Detailed Steps

1. In the Seminar Jnl.-Check Line codeunit, enter code to call the **CheckDimIDComb** and **CheckDimValuePosting** functions of the DimensionManagement codeunit.
 - a. Design codeunit 123456731, Seminar Jnl.-Check Line.
 - b. Create a global variable named DimMgt for the DimensionManagement codeunit.
 - c. Create the following global text constants.

Name	ConstValue
Text002	The combination of dimensions used in %1 %2, %3, %4 is blocked. %5
Text003	A dimension used in %1 %2, %3, %4 has caused an error. %5

- d. In the **RunCheck** function, create the following local variables.

Name	DataType	Length
TableID	Integer	
No	Code	20

- e. On both variables, set the Dimensions property to 10.
f. In the end of the **RunCheck** function, before the final END of the WITH block, enter the following code.

```

IF NOT DimMgt.CheckDimIDComb("Dimension Set ID") THEN
    ERROR(
        Text002,
        TABLECAPTION,"Journal Template Name",
        "Journal Batch Name","Line No.",
        DimMgt.GetDimCombErr);
    TableID[1] := DATABASE::Seminar;
    No[1] := "Seminar No.";
    TableID[2] := DATABASE::Resource;
    No[2] := "Instructor Resource No.";
    TableID[3] := DATABASE::Resource;
    No[3] := "Room Resource No.";
    IF NOT DimMgt.CheckDimValuePosting(TableID,No,"Dimension Set ID") THEN
        IF "Line No." <> 0 THEN
            ERROR(

```

```
Text003,  
  
TABLECAPTION,"Journal Template Name",  
  
"Journal Batch Name","Line No.",  
  
DimMgt.GetDimValuePostingErr)  
  
ELSE  
  
ERROR(DimMgt.GetDimValuePostingErr);
```

- g. Compile, save, and then close the codeunit.

Task 2: Modify the Seminar Jnl.-Post Line Codeunit

High Level Steps

1. In the Seminar Jnl.-Post Line codeunit, add code to copy **Shortcut Dimension 1 Code**, **Shortcut Dimension 2 Code**, and **Dimension Set ID** fields from the journal line to the **Global Dimension 1 Code**, **Global Dimension 2 Code**, and **Dimension Set ID** fields in the ledger entry.

Detailed Steps

1. In the Seminar Jnl.-Post Line codeunit, add code to copy **Shortcut Dimension 1 Code**, **Shortcut Dimension 2 Code**, and **Dimension Set ID** fields from the journal line to the **Global Dimension 1 Code**, **Global Dimension 2 Code**, and **Dimension Set ID** fields in the ledger entry.
 - a. Design codeunit 123456732, Seminar Jnl.-Post Line.
 - b. In the Code function trigger, after assignment block of **SeminarLedgerEntry** table, and before the SeminarLedgerEntry.INSERT statement, enter the following code.

```
SeminarLedgerEntry."Global Dimension 1 Code" := "Shortcut Dimension 1 Code";  
  
SeminarLedgerEntry."Global Dimension 2 Code" := "Shortcut Dimension 2 Code";  
  
SeminarLedgerEntry."Dimension Set ID" := "Dimension Set ID";
```

- c. Compile, save, and then close the codeunit.

Task 3: Import and Review the Seminar-Post Codeunit

High Level Steps

1. Import the Mod08\Labfiles\Lab 8.A - Starter - Seminar-Post.fob object file.
2. Review the code in the Seminar-Post codeunit.

Detailed Steps

1. Import the Mod08\Labfiles\Lab 8.A - Starter - Seminar-Post.fob object file.
 - a. In Object Designer, click **File > Import**.
 - b. Browse to the Mod08\Labfiles\Lab 8.A - Starter - Seminar-Post.fob file and then click **Open**.
 - c. Click **OK** to open the **Import Worksheet**.
 - d. Click **Replace All**, and then click **OK**.
 - e. Click **OK** to complete the object import.
2. Review the code in the Seminar-Post codeunit.
 - a. Design codeunit 123456700, Seminar-Post.
 - b. Review code in the **CheckDim** function, and see where it is called from.
 - c. Review code in the **CheckDimComb** and **CheckDimValuePosting** functions, and see where they are called from.
 - d. Review code in the **PostResJnlLine** and **PostSeminarJnlLine** functions, and how these functions assign dimension fields from the document lines to resource and seminar journals.
 - e. Close the Seminar-Post codeunit.

Module Review

Module Review and Takeaways

Dimensions let users extend the data models of master tables, documents, journals, and ledgers, with additional attributes that meet a company's business needs. They also let users establish rules that posting routines check before creating any posting entries. These posting routines make sure that any data that is entered by users complies with the business rules.

This chapter explored the role of dimensions in Microsoft Dynamics NAV 2013 and demonstrated how to integrate the dimensions management functionality in the existing features of the Seminar Management module. Therefore, CRONUS International Ltd. users can now work with dimensions in the Seminar Management module, in the same manner as they use dimensions in any other application area of Microsoft Dynamics NAV 2013.

Test Your Knowledge

Test your knowledge with the following questions.

1. What is a dimension and what is its purpose in Microsoft Dynamics NAV 2013?

2. Which of the following types is not a dimension type in Microsoft Dynamics NAV 2013?

- () Global Dimension
() Shortcut Dimension
() Budget Dimension
() Department Dimension

Module 8: Dimensions

3. What table stores information about dimensions users define on master records, such as Customer, Vendor, or Item?

4. What table stores dimension combinations for documents, journals, or ledger entries?

5. How do you copy dimension information from a document line to the journal line during the posting process?

6. Which object in Microsoft Dynamics NAV 2013 contains most functionality related to managing dimensions?

Test Your Knowledge Solutions

Module Review and Takeaways

1. What is a dimension and what is its purpose in Microsoft Dynamics NAV 2013?

MODEL ANSWER:

A dimension is a value that is entered into a transaction to enable transactions with similar values to be grouped together for analysis.

2. Which of the following types is not a dimension type in Microsoft Dynamics NAV 2013?

() Global Dimension

() Shortcut Dimension

() Budget Dimension

(√) Department Dimension

3. What table stores information about dimensions users define on master records, such as Customer, Vendor, or Item?

MODEL ANSWER:

Default Dimension

4. What table stores dimension combinations for documents, journals, or ledger entries?

MODEL ANSWER:

Dimension Set Entry

5. How do you copy dimension information from a document line to the journal line during the posting process?

MODEL ANSWER:

You assign the same value of the Dimension Set ID field of the document line to the Dimension Set ID field of the journal line.

Module 8: Dimensions

6. Which object in Microsoft Dynamics NAV 2013 contains most functionality related to managing dimensions?

MODEL ANSWER:

Codeunit 408, DimensionManagement

MODULE 9: ROLE TAILORING

Module Overview

In Microsoft Dynamics NAV 2013, the RoleTailored user interface design gives users a quick overview of the information that is relevant to their job and enables them to focus on their own tasks. At the core of the RoleTailored user interface is the *Role Center*. This is the main entry point into the application for every user.

The primary goal of a Role Center is to increase user productivity. A Role Center maps to the user's role in the organization. It provides access to those functions and features that relate to the role, and hides any rarely used functions or features. Microsoft Dynamics NAV 2013 has more than 20 Role Centers. Each Role Center is represented by a page object.

When you develop custom application areas for Microsoft Dynamics NAV 2013, you must provide new Role Centers for any roles that are not represented in the default set. Or you must customize existing Role Centers to provide access to the new functions or features that you introduced.

Objectives

- Define the components of the RoleTailored user interface.
- Explain the structure, purpose, and functionality of a Role Center-type page.
- Create the **Seminar Manager Role Center** page.
- Describe the functionality of the **Departments** page and the MenuSuite object type.
- Integrate the Seminar Management department into the **Departments** page.

Prerequisite Knowledge

Microsoft Dynamics NAV 2013 supports more than 20 default roles with a default installation. Those roles combine the functionality of the application areas that are available in the standard application. When you create new application areas or customize existing ones, you may have to provide new Role Centers for any user roles that are not included in the default set. Or you may have to integrate any new pages into relevant existing Role Centers to give users access to frequently used features.

A Role Center is a composite page that consists of the following:

- Other pages, such as activities and lists
- System parts, such as notifications or Outlook
- Definitions of the actions in the ribbon
- Definition of the navigation pane and its contents

RoleTailored User Interface Overview

The RoleTailored user interface design gives users a quick overview of the information that is relevant to their job. This design paradigm enables users to do the following:

- Focus, prioritize, and apply their expertise quickly and efficiently.
- Visualize and understand key data.
- Reduce the time that is spent searching for functionality and data.

Research shows that the greatest challenges to users are frequent interruptions and balancing multiple concurrent tasks. Microsoft Dynamics NAV 2013 helps resolve this issue with a RoleTailored design that is combined with a familiar and easy-to-use interface. Instead of mapping to the structure of the database, the RoleTailored user interface maps to the tasks that a role must perform. This role-based navigation reduces complexity by promoting areas of the application that are relevant to each role. Role-based navigation reduces the information that users must understand. It does this by removing application areas that users rarely access.

Role Center

The Role Center links users to the processes in which they participate. By exposing the most frequently used functions in a single location, a Role Center serves as the entry point for every user. It enables users to focus on their tasks, without spending unnecessary time locating features in the user interface.

Module 9: Role Tailoring

The Order Processor Role Center is a typical example of a Role Center. It provides salespeople with quick access to different types of sales documents and allows them to easily access and manage important information, such as customers, items, or notifications.

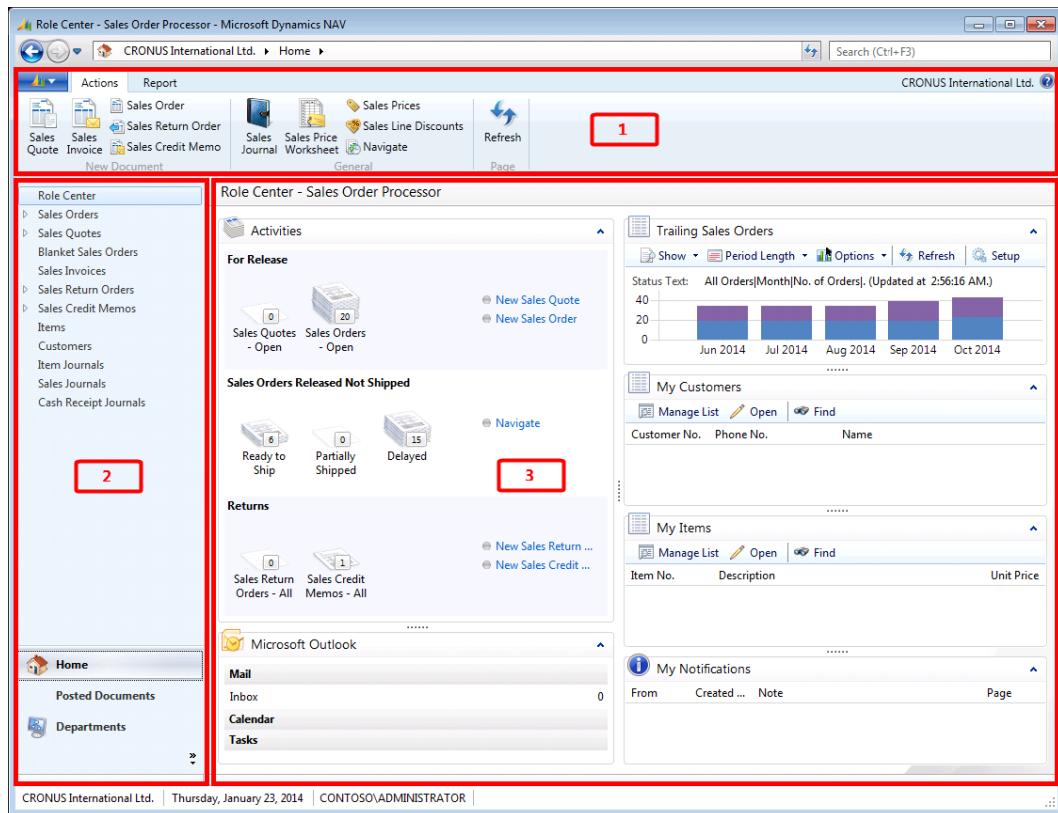


FIGURE 9.1: ORDER PROCESSOR ROLE CENTER

The components of a Role Center are as follows:

1. Ribbon
2. Navigation pane
3. Role center area

You define all these components in a page of type RoleCenter by using the Page Designer window. The role center area is the main area of a **Role Center** page, and replaces the content area that all other page types have.

The “Page 9006, Order Processor Role Center” image shows the components of the **Order Processor Role Center** page in Page Designer.

C/SIDE Solution Development in Microsoft Dynamics® NAV 2013

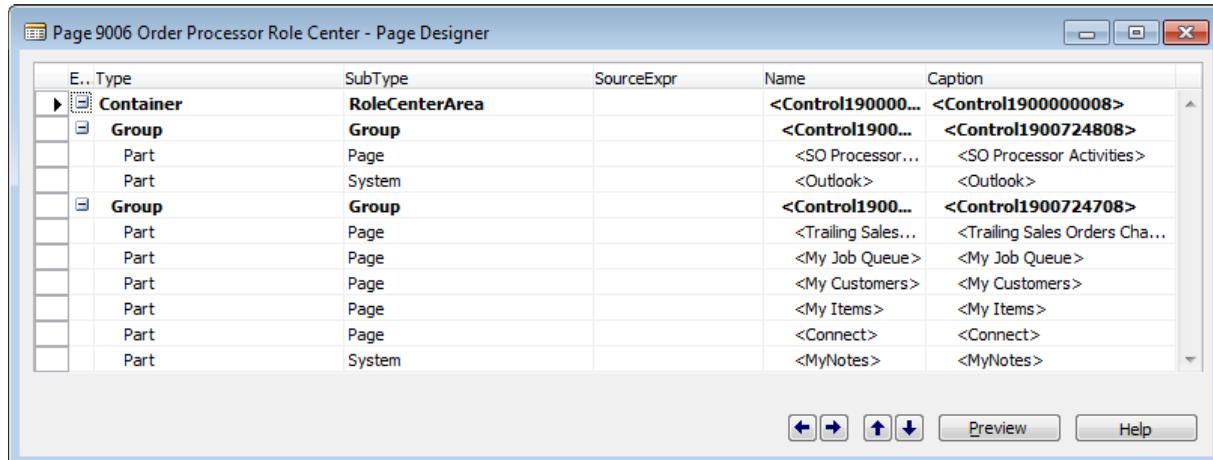


FIGURE 9.2: PAGE 9006, ORDER PROCESSOR ROLE CENTER

 **Note:** You cannot write any trigger code on pages of type **RoleCenter**. This applies not only to page triggers, such as **OnOpenPage** or **OnClosePage**, but also to action triggers. Any action that is defined in a **Role Center** page may only use the **RunObject** property. Any code in the triggers prevents you from compiling or saving the page.

Actions and Ribbon

Actions let users access features of an application, such as a List Place, or run a function, such as posting an invoice. The ribbon organizes action types into tabs and groups for quick and easy access by users. The ribbon is available from most page types.

The "Ribbon" image shows the ribbon for the **Customer List** page, and it shows the **Home**, **Actions**, **Navigate**, and **Report** tabs, and New, Manage, Process, Report, View, and Show Attached groups.



FIGURE 9.3: THE RIBBON

For each page, you define all actions in the Action Designer. This includes the contents of the ribbon. For pages of type **RoleCenter**, you also use Action Designer to define the contents of the navigation pane.

You define where an action is shown in the user interface by using different action container types. This is how different action container types map to elements of the RoleTailored user interface.

Action Container Type	RoleTailored Client Element
NewDocumentItems	Actions tab, New Document group
ActionItems	Actions tab
RelatedInformation	Navigate tab
Reports	Report tab
HomeItems	Home menu of the Navigation pane
ActivityButtons	Other menus of the Navigation pane, such as Posted Documents

 **Note:** If an action container of type ActionItems, RelatedInformation or Reports contains action groups, these groups appear as separate group in the **Actions**, **Navigate** and **Report** tabs. Any actions that do not belong to an action group and are defined at the action container level, appear in the **General** group of the appropriate tab.

When you define actions in the HomeItems or ActivityButtons action containers in the **Role Center** page, the following actions appear in the navigation pane for any user who belongs to a profile that uses this **Role Center** page:

- Actions in the HomeItems action container appear in the **Home** menu, and you cannot group these actions into action groups.
- Actions in the ActivityButtons action container are displayed as separate menus. You must group these actions into action groups. Every action group is listed as a separate menu in the navigation pane.

In addition to the **Home** menu, the navigation pane for the production planner profile includes **Journals**, **Worksheets**, **Product Design**, and **Capacities** menus. Each of these additional menus is defined as an action group in the ActivityButtons action container in page 9010, **Production Planner Role Center**.

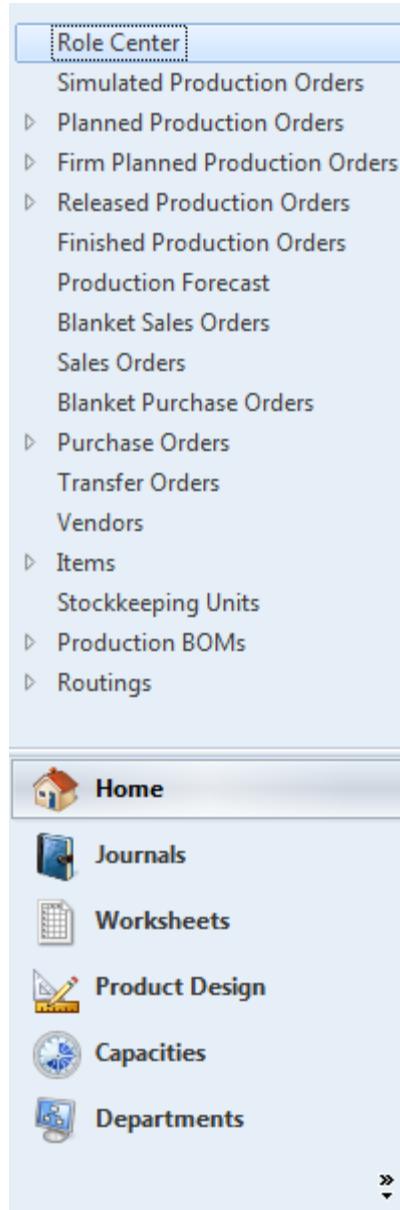


FIGURE 9.4: NAVIGATION PANE FOR THE PRODUCTION PLANNER PROFILE

Activities

Every Role Center includes the Activities part that contains stacks of documents. These are known as *cues*. A cue visually shows the number of documents of a specific type that exists in the application. Cues also let users quickly access the list of documents with an appropriate filter applied. For example, if a cue shows the number of released sales orders, and a user clicks the cue, the list of released sales orders is displayed.

Module 9: Role Tailoring

The Activities part is defined as a CardPart page, and uses groups of type CueGroup to show the cues. Every cue is defined as a field in the source table of the **Activities** page.

CueGroup is the only control type in Microsoft Dynamics NAV 2013 that lets you define control actions. Any control action that is defined on a CueGroup is displayed as a link on the right side of the cues.

Page 9060, **SO Processor Activities** is an example of an activities page. It includes three CueGroups that are named as follows: For Release, Sales Orders Released Not Shipped, and Returns. Each group contains actions. For example, the For Release group includes the New Sales Quote and New Sales Order actions.

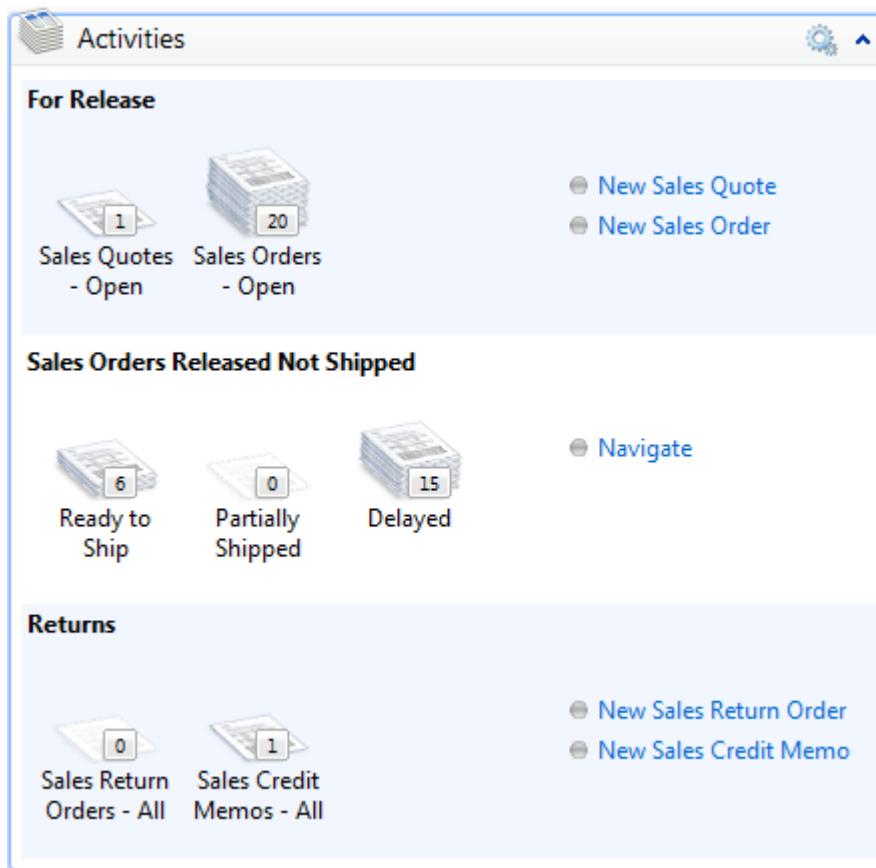


FIGURE 9.5: PAGE 9060, SO PROCESSOR ACTIVITIES

A field must be of type Integer to correctly show as a cue. If there is no record in the source table, the **Activities** page inserts one.

The source table of the **Activities** pages has the following characteristics:

- It is always named after the functional area, and ends with the word **Cue**. Examples are **Sales Cue**, **Finance Cue**, or **Purchase Cue**.
- It can only contain a single record.

- It has the primary key that has a single field that is named **Primary Key**.
- Every cue is defined as an integer field of FlowField class and it uses the Count method.

Lists

Role Centers may contain one or more lists, such as customers, vendors, or items. These lists are defined as page parts. Each page is defined as a ListPart page. The caption of each list is preceded by the word "my," so that the lists are displayed as **My Customers**, **My Vendors**, or **My Items**.

The **Order Processor Role Center** page includes the **My Customers** and **My Items** lists.

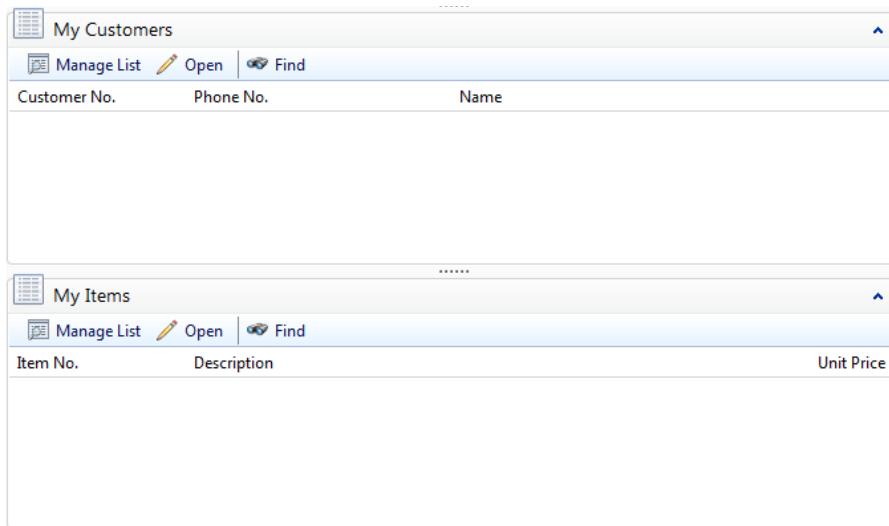


FIGURE 9.6: LISTS IN THE ORDER PROCESSOR ROLE CENTER

When a Role Center contains such a list, the system automatically shows the Manage List action. This lets users select the records to show in the list. Every user can decide which records, such as customers or vendors, to include in the list. The list is different for each user, and users manage their own lists independently.

To manage the lists for different users, every list page uses the source table that has the following characteristics:

- It is named after the record type of the list, preceded by the word "my." Examples of source tables for list pages are **My Customer**, **My Vendor**, and **My Item**. The name is always singular.

- It contains only two fields: **User ID** that specifies to which user the record belongs, and another field that relates to the specific record of the type that list page shows. For example, the **My Customer** table contains the **Customer No.** field.
- Its primary key always contains both fields.

When a list page is opened, the OnOpenPage trigger sets the range on the **User ID** field to the current user. This guarantees that the page only shows the records that belong to the current user. When a new record is inserted in the page, the **User ID** field is automatically populated with the ID of the current user.

Each list page must include an action named Open that opens the appropriate page for managing the record type that is shown in the list. For example, clicking Open in the **My Customer** list opens the **Customer Card** page.

Charts

Charts are graphical representations of business data combinations that you can view in list places, FactBoxes, and Role Centers. A Role Center may contain one or more charts that display information that is relevant to a profile.

The “SO per Location Chart” figure shows a chart that users can add to their Role Centers.

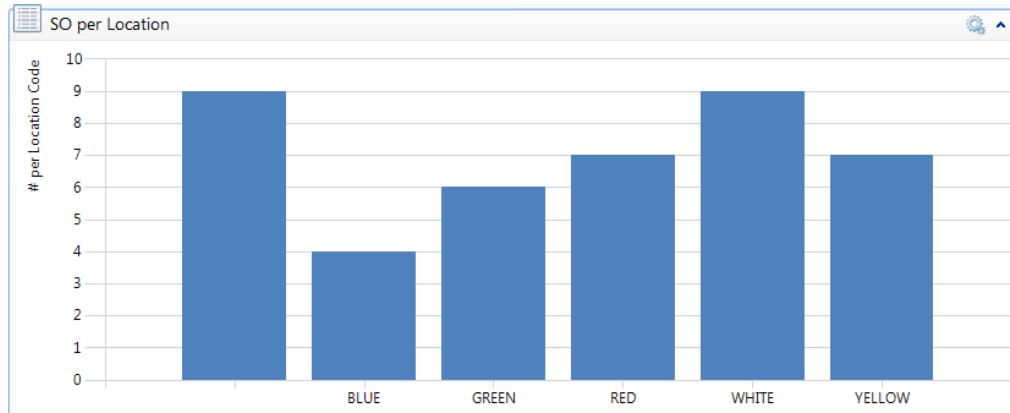


FIGURE 9.7: THE SO PER LOCATION CHART

Charts are not defined as objects in the development environment, but as records in the **Chart** table. To access the list of available charts, click **Departments > Administration > Application Setup > RoleTailored Client > Charts**. You create charts by defining data and chart properties in the **Generic Chart Setup** page.

The “Generic Card Setup Page” figure shows the SO per Location chart in the **Generic Chart Setup** page.

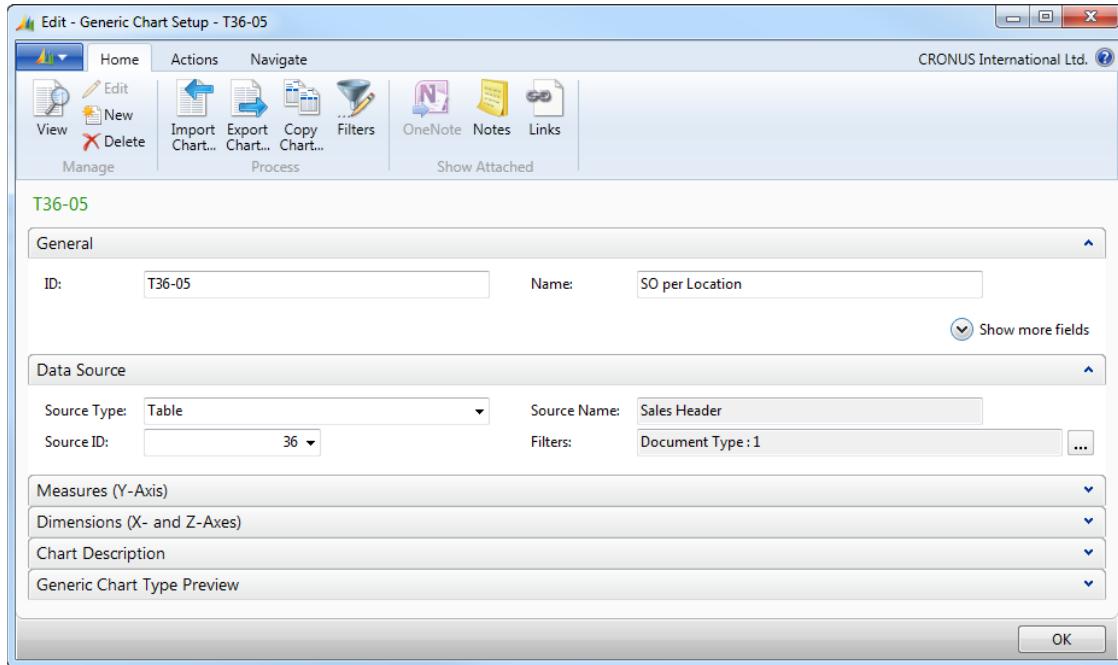


FIGURE 9.8: GENERIC CARD SETUP PAGE

Profiles

Profiles link Role Centers to users and align with the roles and responsibilities that users have in an organization.

To manage profiles, click **Departments > Administration > Application Setup > RoleTailored Client > Profiles**.

Use the **Profiles** page for an overview of all profiles.

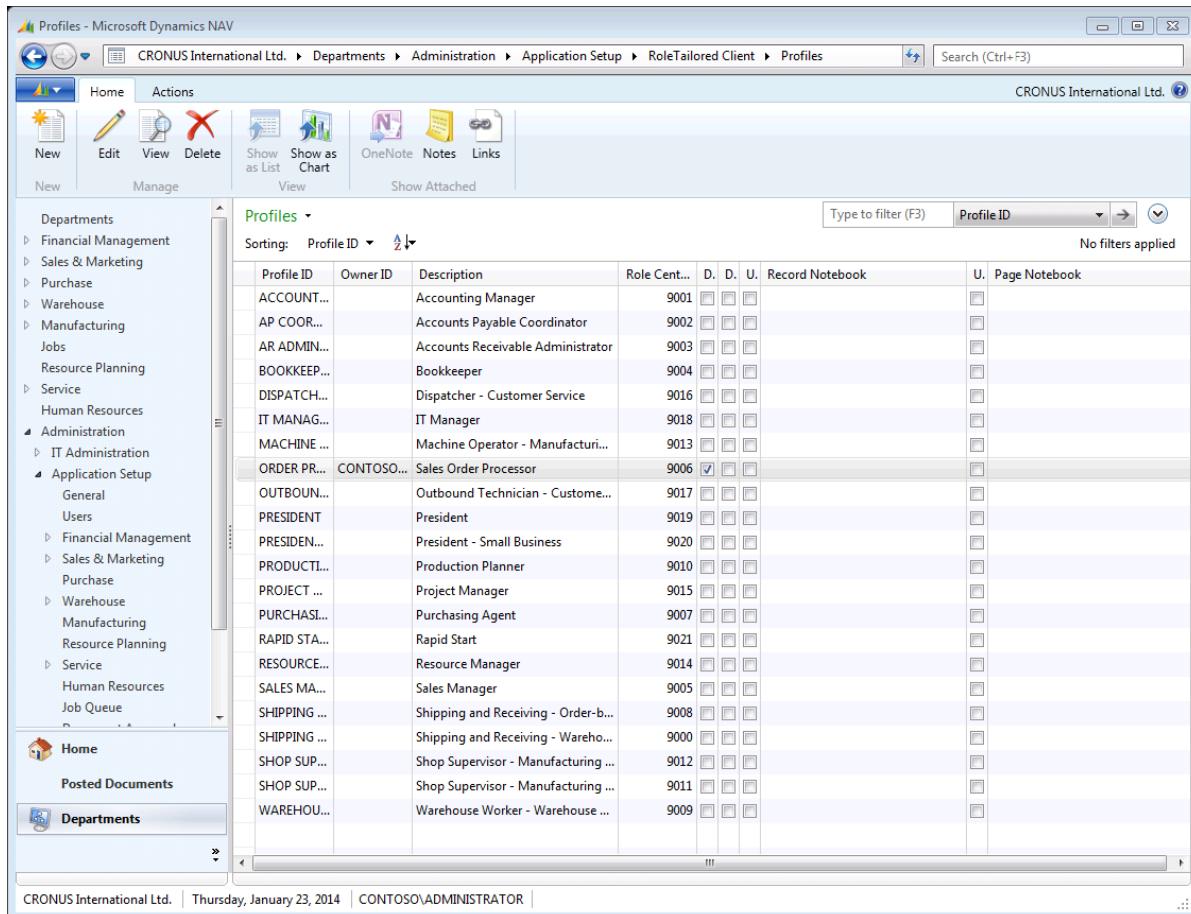


FIGURE 9.9: PROFILES LIST WINDOW

Use the **Profile Card** page to create new profiles or change existing ones.

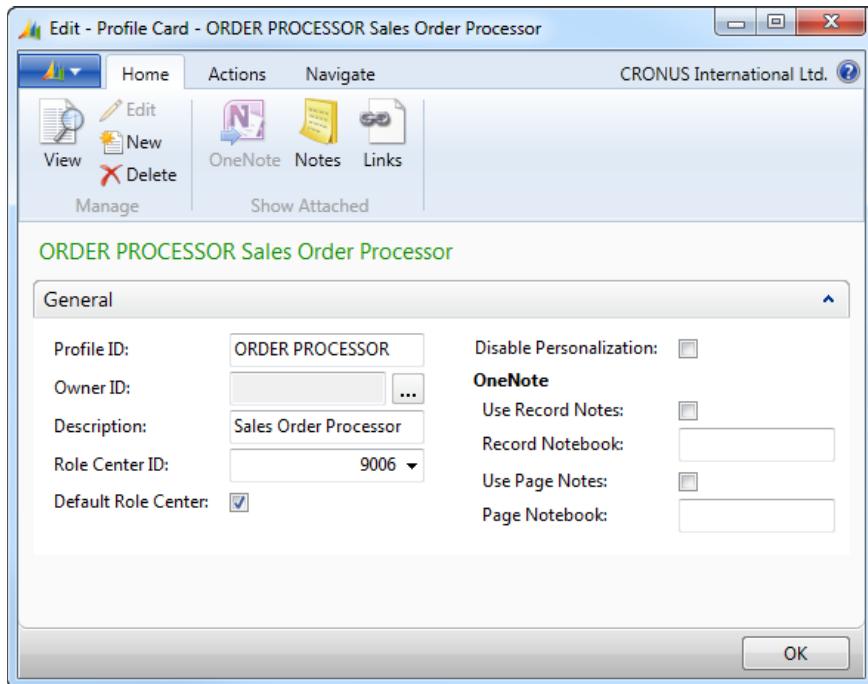


FIGURE 9.10: SALES ORDER PROFILE IN THE PROFILE CARD PAGE

Seminar Manager Role Center

Managing seminars and seminar registrations is a core process of CRONUS International Ltd. There are many employees in their organization who manage seminar information, organize new seminars, manage instructors and rooms, and perform other seminar-related activities.

To enable CRONUS employees to be as productive as possible, you must provide a Role Center that makes the most frequently used features available to users in as few clicks as possible, and hides any unnecessary information.

Solution Design

CRONUS International Ltd. functional requirements do not specifically mention Role Center functionality. However, there is a nonfunctional requirement that the solution must let users be as productive as possible by reducing the amount of searching and filtering.

Role Centers are one of the core productivity features of Microsoft Dynamics NAV 2013. It aligns with the following goals of the nonfunctional requirement:

- It helps users be as productive as possible.
- It significantly reduces time that is spent searching features.
- It significantly reduces time that is spent filtering lists.

Module 9: Role Tailoring

To provide a consistent user experience with other application areas, you must provide a **Seminar Role Center** page that includes all features that are typically found on Role Centers. These features let users quickly enter and process information.

At a minimum, the **Seminar Manager Role Center** page must contain the following features:

- Include the Activities part with cues that indicate the number of documents in various statuses.
- Include the **My Seminars** and **My Customers** lists.
- Include links in the **Home** menu to enable users to access master data (seminars, instructors, rooms, customers, and contacts) and documents (registrations).
- Include the **Posted Documents** menu to enable users to access posted information for seminars.
- Enable users to create new seminar registration and invoice documents directly from the Role Center.
- Enable users to run the invoice creation batch job.
- Enable users to run the **Navigate** page.
- Enable users to print the **Participant List** report.

Development

A Role Center is not a single page object. It depends on several other objects, including pages and tables, to provide users with familiar functionality.

Tables

To enable the Role Center functionality, you must create the following tables.

Table	Remarks
123456740 Seminar Cue	Contains the flow fields for the cues on the Activities page part of the Role Center page.
123456741 My Seminar	Contains the list of seminars that each user has included in the My Seminars list.

Pages

You must create the following pages:

Page	Remarks
123456740 Seminar Manager Role Center	The Role Center page.

Page	Remarks
123456741 Seminar Manager Activities	The Activities page part for the Role Center page.
123456742 My Seminars	A list part for the Role Center page that lets users manage their list of favorite seminars.

The “Seminar Manager Role Center” figure shows the Seminar Manager Role Center with all its components.

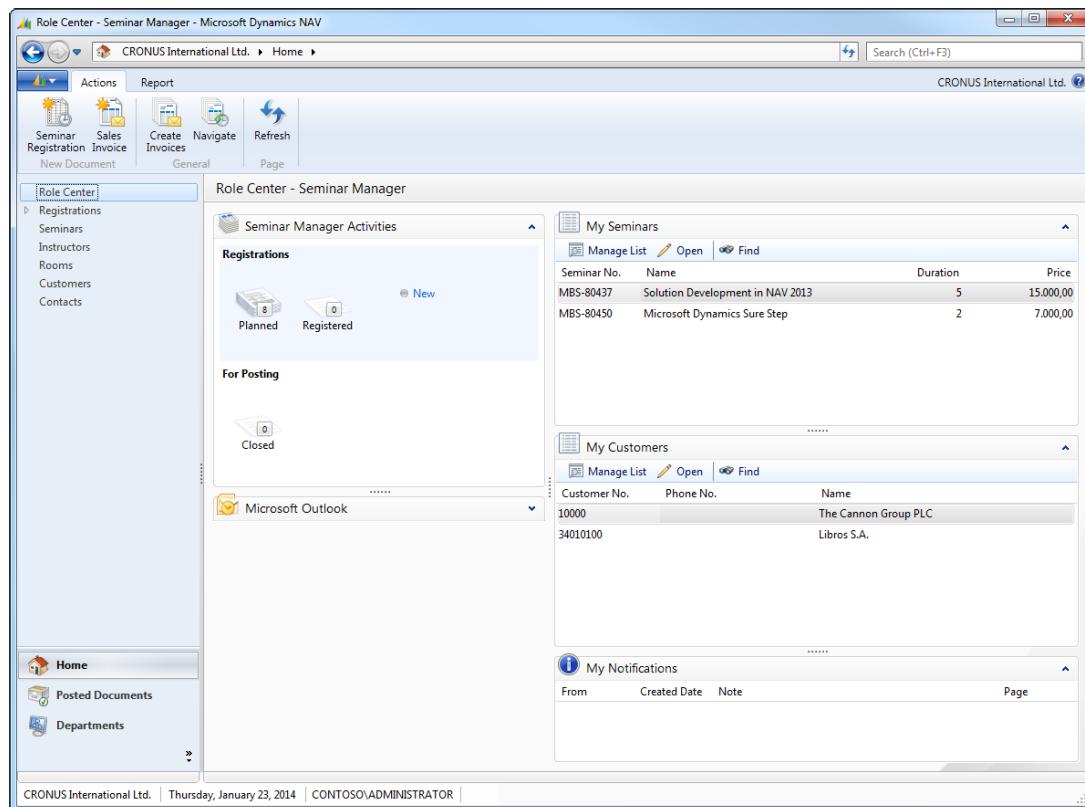


FIGURE 9.11: SEMINAR MANAGER ROLE CENTER

Lab 9.1: Create the Seminar Manager Role Center

Scenario

You are a developer at the company that is implementing Microsoft Dynamics NAV 2013 for CRONUS International Ltd. Your task is to develop the **Role Center** page for the Seminar Manager role. The **Role Center** page must contain the functionality that is normally found in a Microsoft Dynamics NAV 2013 **Role Center** page.

Exercise 1: Seminar Activity Page

Exercise Scenario

Start by creating the supporting objects. The first supporting object is the **Activities** page that is used as a page part on the **Seminar Manager Role Center** page. The **Activities** page uses a source table. Therefore, you must first develop the table, and then the page.

Task 1: Create the Cue Table

High Level Steps

1. Create the **Seminar Cue** table.
2. Set the flow field properties on the appropriate fields to indicate the count of planned, registered, and closed seminar registrations.

Detailed Steps

1. Create the **Seminar Cue** table.
 - a. Create a new table, and save it as 123456740, **Seminar Cue**.
 - b. Define the following fields in the table.

Field No.	Field Name	Data Type	Length
1	Primary Key	Code	10
2	Planned	Integer	
3	Registered	Integer	
4	Closed	Integer	

2. Set the flow field properties on the appropriate fields to indicate the count of planned, registered, and closed seminar registrations.
 - a. Set the properties on the **Planned** field to be the flow field, and to show the number of seminar registrations in status Planned.



Note: Count("Seminar Registration Header" WHERE
(Status=CONST(Planning)))

- b. Set the properties on the **Registered** field to be the flow field, and to show the number of seminar registrations in status Registration.
-



Note: Count("Seminar Registration Header" WHERE (Status=CONST(Registration)))

- c. Set the properties on the **Closed** field to be the flow field, and to show the number of seminar registrations in status Closed.
-



Note: Count("Seminar Registration Header" WHERE (Status=CONST(Closed)))

- d. Compile, save, and then close the table.

Task 2: Create the Activities Page

High Level Steps

1. Create the **Seminar Manager Activities** page.
2. Define two cue groups: **Registrations** and **For Posting**.
3. Add fields to cue groups.
4. Add an action to the Registrations cue group to enable users to create new seminar registrations.
5. Add code to the OnOpenPage trigger to make sure that there is always one record in the source table.

Detailed Steps

1. Create the **Seminar Manager Activities** page.
 - a. Create a new page for the **Seminar Cue** table, with the CardPart Page Wizard.
 - b. Finish the wizard without adding any fields.
 - c. Save the page as 123456741, **Seminar Manager Activities**.
2. Define two cue groups: **Registrations** and **For Posting**.
 - a. Under the ContentArea container, add the following controls.

Type	SubType	Caption
Group	CueGroup	Registrations
Group	CueGroup	For Posting

- b. Make sure that you indent both cue group controls at the same level, which is one level under the parent container.

3. Add fields to cue groups.
 - a. Add field controls for the **Planned** and **Registered** fields to the Registrations cue group.
 - b. Add field controls for the **Closed** field to the For Posting cue group.
 - c. Make sure that all controls are properly indented under their parent cue groups.

The “Page 123456741, Seminar Manager Activities in the Page Designer” figure shows the **Page Designer** window for the **Seminar Manager Activities** page, with all controls correctly aligned and indented.

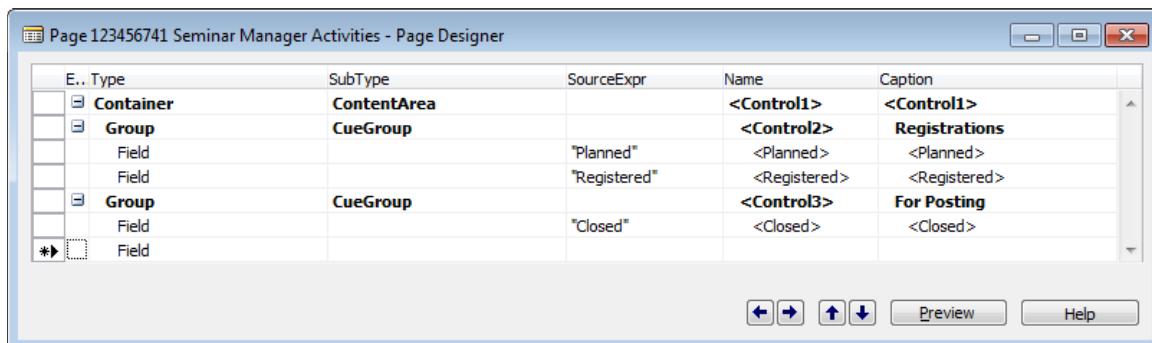


FIGURE 9.12: PAGE 123456741, SEMINAR MANAGER ACTIVITIES IN THE PAGE DESIGNER

- d. For each field control (Planned, Registered, and Closed), set the DrillDownPageID property to “Seminar Registration List”.
4. Add an action to the Registrations cue group to enable users to create new seminar registrations.
 - a. Select the Registrations cue group, and then click **View > Control Actions**.
 - b. In the **CueGroup - ActionDesigner** window on the first row, set **Type** to Action, and in the Caption column, enter “New”.
 - c. Set the following properties for the New action.

Property	Value
RunPageMode	Create
RunObject	Page Seminar Registration

5. Add code to the OnOpenPage trigger to make sure that there is always one record in the source table.
 - a. Open the C/AL Editor for the page.
 - b. In the OnOpenPage trigger, enter the following code.

```
RESET;  
  
IF NOT GET THEN BEGIN  
  
    INIT;  
  
    INSERT;  
  
END;
```

- c. Compile, save, and then close the page.

Exercise 2: My Seminars Page

Exercise Scenario

The next supporting Role Center object is the **My Seminars** page. This page also uses a source table. Therefore, before developing the page, you must first develop the underlying table. After you create the **My Seminar** table, you create the **My Seminars** page.

Task 1: Create the My Seminar Table

High Level Steps

1. Create the **My Seminar** table.
2. Define table relations on fields.
3. Define the appropriate primary key for the **My Seminar** table.

Detailed Steps

1. Create the **My Seminar** table.
 - a. Create a new table, and save it as 123456741, **My Seminar**.
 - b. Define the following fields in the table.

Field No.	Field Name	Data Type	Length
1	User ID	Code	50
2	Seminar No.	Code	20

 2. Define table relations on fields.
 - a. On the **User ID** field, set the TableRelation property to User."User Name".
 - b. On the **Seminar No.** field, set the TableRelation property to "Seminar".

3. Define the appropriate primary key for the **My Seminar** table.
 - a. Click **View > Keys**.
 - b. In the Key column, enter "User ID,Seminar No."
 - c. Compile, save, and then close the table.

Task 2: Create the My Seminars Page

High Level Steps

1. Create the **My Seminars** page. The page must show only records that belong to the current user.
2. Define a variable for the **Seminar** table, and add more fields from the **Seminar** table to the **My Seminars** page.
3. Enter code in appropriate triggers to make sure that the Seminar record always points to the record that is specified in the **Seminar No.** field of the current My Seminar record.
4. Add the **OpenSeminarCard** function to run the **Seminar Card** page for the current record.
5. Define an action to run the **OpenSeminarCard** function.

Detailed Steps

1. Create the **My Seminars** page. The page must show only records that belong to the current user.
 - a. Create a new page for the **My Seminar** table, with the ListPart Page Wizard.
 - b. Add the **Seminar No.** field by using the wizard, and finish the wizard.
 - c. In the OnOpenPage trigger, enter the following code.

```
SETRANGE("User ID",USERID);
```

- d. Save the page as 123456742, **My Seminars**.
2. Define a variable for the **Seminar** table, and add more fields from the **Seminar** table to the **My Seminars** page.
 - a. Define a new global variable of type Record for the **Seminar** table, and name it Seminar.
 - b. Under the Seminar No. field control, add the following controls.

Type	SourceExpr	Name	Caption
Field	Seminar.Name	<Control4>	Name
Field	Seminar."Seminar Duration"	<Control5>	Duration
Field	Seminar."Seminar Price"	<Control6>	Price

C/SIDE Solution Development in Microsoft Dynamics® NAV 2013

The “Page 123456742, My Seminars” figure shows the **Page Designer** window for the **My Seminars** page with all controls correctly aligned and indented.

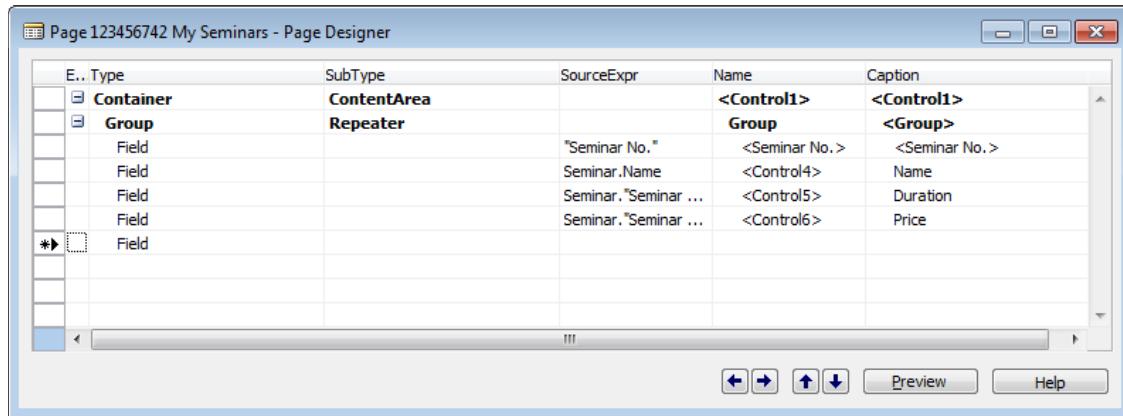


FIGURE 9.13: PAGE 123456742, MY SEMINARS

3. Enter code in appropriate triggers to make sure that the Seminar record always points to the record that is specified in the **Seminar No.** field of the current My Seminar record.
 - a. In the **C/AL Globals** window, define a new function, and name it **GetSeminar**. The function does not receive parameters.
 - b. In the function trigger for the **GetSeminar** function, enter the following code.

```
CLEAR(Seminar);  
  
IF Seminar.GET("Seminar No.") THEN;
```

- c. In the OnAfterGetRecord trigger, enter the following code.

```
GetSeminar;
```

- d. Enter the same code in the OnValidate trigger of the Seminar No. field control.
 - e. In the OnNewRecord trigger, enter the following code.

```
CLEAR(Seminar);
```

4. Add the **OpenSeminarCard** function to run the **Seminar Card** page for the current record.
 - a. In the **C/AL Globals** window, define a new function, and name it **OpenSeminarCard**. The function does not receive parameters.
 - b. In the function trigger for the **OpenSeminarCard** function, enter the following code.

```
IF Seminar.GET("Seminar No.") THEN
```

```
PAGE.RUN(PAGE::"Seminar Card",Seminar);
```

5. Define an action to run the **OpenSeminarCard** function.
 - a. Show the Actions Designer for page actions.
 - b. In the SubType column, select ActionItems.
 - c. On the new row, in the Caption column, enter "Open."

The "Action Designer for the My Seminars Page" figure shows how the **Action Designer** should look now.

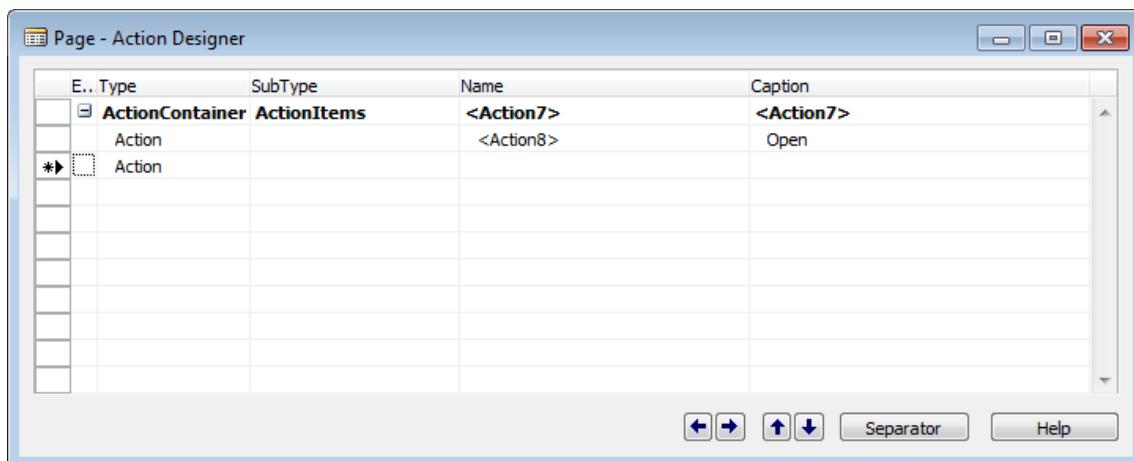


FIGURE 9.14: ACTION DESIGNER FOR THE MY SEMINARS PAGE

- d. On the Open action, define the following properties.

Property	Value
Image	Edit
ShortCutKey	Return

- e. In the OnAction trigger for the Open action, enter the following code.

```
OpenSeminarCard;
```

- f. Compile, save, and then close the page.

Exercise 3: The Role Center Page

Exercise Scenario

After you have completed development of all supporting tables and pages, you are ready to define the **Seminar Manager Role Center** page. For this page, you must design the page layout, and arrange various page parts in a manner that is consistent with standard Microsoft Dynamics NAV 2013 **Role Center** pages. Then you must define the navigation pane elements. This includes the Home items and any additional menus. Finally, you must define the ribbon actions for the **Role Center** page.

Task 1: Create the Seminar Manager Role Center Page

High Level Steps

1. Create the **Seminar Manager Role Center** page.
2. Add relevant controls to the **Seminar Manager Role Center** page.

Detailed Steps

1. Create the **Seminar Manager Role Center** page.
 - a. Create a new page of type RoleCenter.
 - b. Save the page as 123456740, **Seminar Manager Role Center**.
2. Add relevant controls to the **Seminar Manager Role Center** page.
 - a. Add two group controls, and then indent them one level under the RoleCenterArea control.
 - b. To the first group, add a page part and a system part control.
 - c. For the page part control, set properties to show the **Seminar Manager Activities** page.



Note: Set the *PagePartID* property.

- d. For the system part control, set properties to show the Outlook part.



Note: Set the *SystemPartID* property.

- e. To the second group, add the following part controls.

SubType	Part ID
Page	My Seminars
Page	My Customers
Page	Connect Online

SubType	Part ID
System	MyNotes

- f. For the **Connect Online** part, set the **Visible** property to FALSE.

The "Page 123456740 Seminar Manager Role Center in Page Designer" figure shows the **Seminar Manager Role Center** page in the **Page Designer** window.

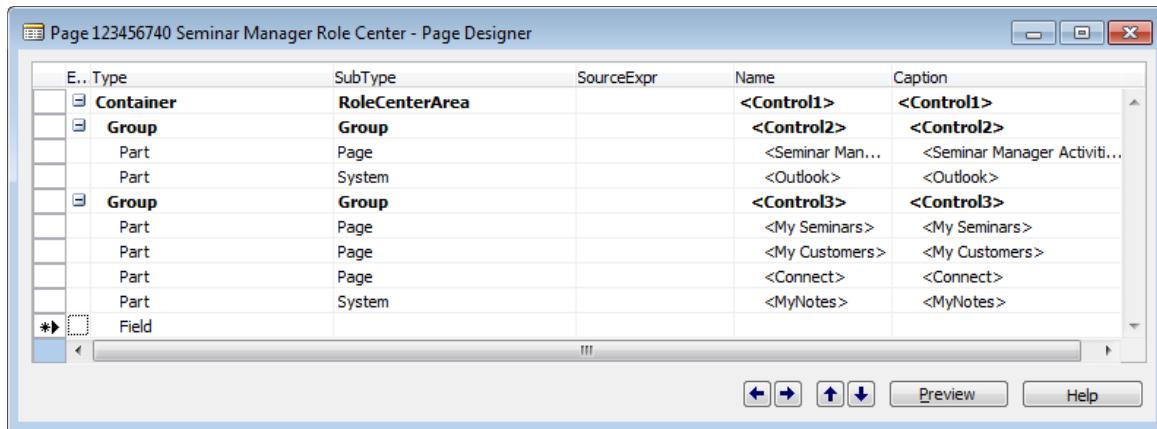


FIGURE 9.15: PAGE 123456740 SEMINAR MANAGER ROLE CENTER IN PAGE DESIGNER

- g. Save the page.

Task 2: Add Home Items

High Level Steps

- Define actions for home items to include list pages for relevant **Seminar Management** pages.

Detailed Steps

- Define actions for home items to include list pages for relevant **Seminar Management** pages.
 - Access the Action Designer for the **Seminar Manager Role Center** page actions.
 - Define a HomeItems container.
 - Add the following actions to the HomeItems container.

Caption	RunObject
Seminar Registrations	Page Seminar Registration List
Seminars	Page Seminar List
Instructors	Page Resource List
Rooms	Page Resource List
Customers	Page Customer List
Contacts	Page Contact List

- d. For the Instructors action, set the RunPageView property to "WHERE(Type=CONST(Person))".
- e. For the Rooms action, set the RunPageView property to "WHERE(Type=CONST(Machine))".
- f. Save the page.

Task 3: Add Activity Buttons

High Level Steps

1. Add the **Posted Documents** activity button.
2. Add page links for posted documents to the **Posted Documents** activity button.

Detailed Steps

1. Add the **Posted Documents** activity button.
 - a. In the **Action Designer** window for the **Seminar Manager Role Center** page, define a new action container of type **ActivityButtons**.
 - b. Under the **ActivityButtons** container, add an action group, and set its caption to "Posted Documents".
 - c. On the **Posted Documents** action group, set the **Image** property to "RegisteredDocs".
2. Add page links for posted documents to the **Posted Documents** activity button.
 - a. Under the **Posted Documents** action group, add the following actions.

Caption	RunObject
Posted Seminar Registrations	Page Posted Seminar Reg. List
Posted Sales Invoices	Page Posted Sales Invoices
Registers	Page Seminar Registers

Task 4: Add Ribbon Actions

High Level Steps

1. Define actions to appear in the New Document group of the ribbon on the **Seminar Manager Role Center** page.
2. Define actions to appear in the General group of the ribbon on the **Seminar Manager Role Center** page.
3. Define an action for the **Report** tab of the ribbon on the **Seminar Manager Role Center** page.

Detailed Steps

1. Define actions to appear in the New Document group of the ribbon on the **Seminar Manager Role Center** page.
 - a. In the **Action Designer** window for the **Seminar Manager Role Center** page, define a new action container of type NewDocumentItems.
 - b. Under the NewDocumentItems container, add the following actions.

Caption	RunObject	Image
Seminar Registration	Page Seminar Registration	NewTimesheet
Sales Invoice	Page Sales Invoice	NewInvoice

- c. For both actions, set the RunPageMode property to Create.
2. Define actions to appear in the General group of the ribbon on the **Seminar Manager Role Center** page.
 - a. In the Action Designer window for the **Seminar Manager Role Center** page, define a new action container of type ActionItems.
 - b. Under the ActionItems container, add the following actions.

Caption	RunObject	Image
Create Invoices	Report Create Seminar Invoices	CreateJobSalesInvoice
Navigate	Page Navigate	Navigate

3. Define an action for the **Report** tab of the ribbon on the **Seminar Manager Role Center** page.
 - a. In the Action Designer window for the **Seminar Manager Role Center** page, define a new action container of type Reports.
 - b. Under the Reports container, do the following:
 - Add an action.
 - Set its caption to "Participant List".
 - Set its RunObject property to "Report Seminar Reg.- Participant list".
 - Set its Image property to "Report".
 - c. Compile, save, and then close the page.

Task 5: Define a Profile for the Seminar Manager

High Level Steps

1. In the Microsoft Dynamics NAV 2013 client for Windows, define a new profile and configure it for use in the **Seminar Manager Role Center** page.

Detailed Steps

1. In the Microsoft Dynamics NAV 2013 client for Windows, define a new profile and configure it for use in the **Seminar Manager Role Center** page.
 - a. Start Microsoft Dynamics NAV 2013 client for Windows.
 - b. In the **Search** text box, enter “Profiles”.
 - c. In the search results, click **Profiles**.

 **Note:** Or, browse to **Departments > Administration > Application Setup > RoleTailored Client > Profiles**.

- d. Click **New**.
- e. In the **New – Profile Card** window, enter the following information.

Field	Value
Profile ID	SEMINAR MANAGER
Description	Seminar Manager
Role Center ID	123456740

- f. Select the **Default Role Center** check box.

The “Seminar Manager Profile Card” figure shows the **Profile Card** page for the Seminar Manager profile.

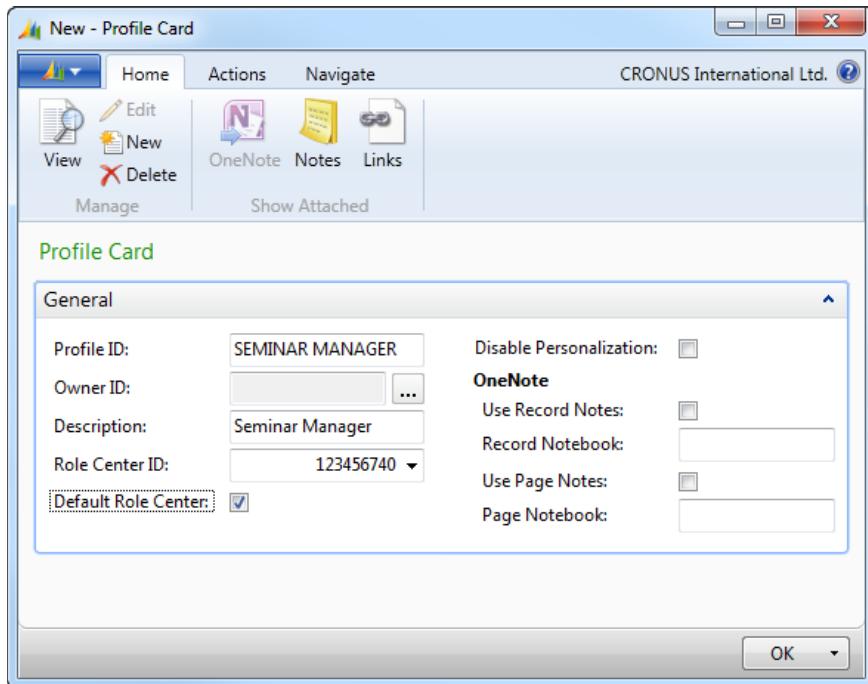


FIGURE 9.16: SEMINAR MANAGER PROFILE CARD

g. Restart the Microsoft Dynamics NAV 2013 client for Windows.

MenuSuite Object Type

In Microsoft Dynamics NAV 2013, you can create MenuSuite objects. These objects contain menu content that is displayed in department pages of the Microsoft Dynamics NAV 2013 client for Windows. The MenuSuite object that is provided in Microsoft Dynamics NAV is a basic MenuSuite object that you can change. You can also create new MenuSuite objects from the basic MenuSuite object. Each menu in the basic MenuSuite represents a department such as Financial Management, Jobs, and Manufacturing. Each department contains menu items that are specific to that department area.

Fundamentals

A *MenuSuite* is a set of menus, groups, and menu items. Each menu contains content for a specific application area, such as Financial Management, Jobs, or Manufacturing. You design MenuSuites in the Navigation Pane Designer in the Microsoft Dynamics NAV 2013 Development Environment.

A MenuSuite object has the following characteristics:

- It consists of a set of menus.
- A menu contains a collection of menu nodes, which are displayed in the **Departments** page.

- It is organized in a tree structure.
- A menu node can be either a group or a menu item.
- A menu node has a Globally Unique Identifier (GUID) and other properties.
- A group contains a collection of menu nodes.
- A menu item is the lowest level in the tree. When you click a menu item, its associated application object runs.

Design Levels

For each MenuSuite object that you create, you select a design level. Design levels are permissions that allow different roles to create MenuSuite objects for specific functions. The design level that you can work in depends on your active license. In other words, you can only design MenuSuite objects or create MenuSuites that your active license and permissions allow. For example, if your active license enables you to localize and you design an application that will be localized, your design level is at the Country or Region level. If you are a developer working at a Microsoft Certified Partner, you might design a MenuSuite object at the Partner level.

Unlike any other application objects, the ID and Name properties of a MenuSuite object are determined by the development environment depending on the design level. The following table shows the predefined MenuSuite IDs and their corresponding levels.

ID	Level	Remarks
1010	Dept – MBS	The base MenuSuite object that is defined by Microsoft. It defines the default Departments menu of standard Microsoft Dynamics NAV 2013, and it includes all W1 (worldwide) version objects. This object is the same for all local versions of Microsoft Dynamics NAV 2013. Any MenuSuite that you create inherits from this base design level or a design level that is inherited from this base design level.

Module 9: Role Tailoring

ID	Level	Remarks
1020	Dept – Region	Includes menus that are defined by Microsoft for any non-W1 functionality that is provided for a specific region. There is no strict definition of a region, and MenuSuite 1020 is not necessarily present in all localized versions. Regional functionality may include some specific functions or features that are the same for several countries. They otherwise have different local functionality.
1030	Dept – Country	Includes menus for any local functionality that is specific to a single country. This MenuSuite is typically defined by Microsoft or another party that is officially authorized to develop localizations.
1051..1060	Dept – Add-on	Includes menus for any ISV solutions or third-party add-on modules that your application uses. This MenuSuite is typically defined by the partner company that implements the add-on solution for its customer. There may be up to ten add-on MenuSuite objects.
1080	Dept – Partner	Includes menus for any customizations that are provided by a single partner that this partner reuses at multiple company deployments of Microsoft Dynamics NAV 2013.
1090	Dept – Company	Includes menus for any customizations that are provided for a single customer company, such as CRONUS International Ltd.

You determine the design level of a MenuSuite at the same time that you create it. When you save a MenuSuite, its design level is embedded in its ID and Name properties.

The **Design Level** window shows you the list of levels. You must select the level for the new MenuSuite.

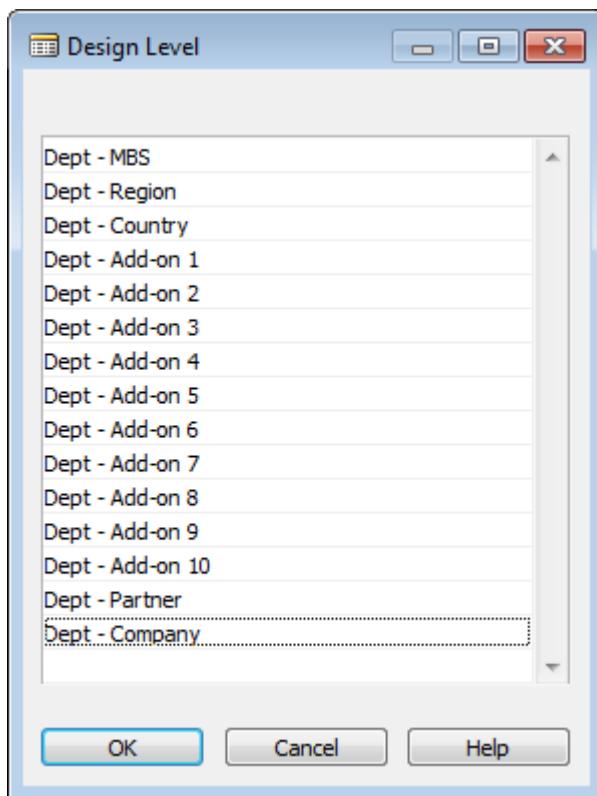


FIGURE 9.17: THE DESIGN LEVEL WINDOW

When you create a MenuSuite object, it inherits from the design level that is above the level that you selected. For example, if you design at the Region level, your MenuSuite object inherits all menus and menu items from the Dept - MBS level. Changes that you make to a MenuSuite object that you create are stored as the differences between the MenuSuite object at the Dept - MBS level and the Dept - Region level. The most basic design level is the Dept - MBS level. The highest design level is the Dept - Company level. You can create only one MenuSuite object for each level in an application. The Add-on level is the exception. It can have no more than ten instances of MenuSuite objects. At design time, the header area of Navigation Pane Designer displays the design level of the MenuSuite object.

Create and Design MenuSuites

Every new MenuSuite object that you create is designed at one of the design levels. If the Dept - MBS MenuSuite is the only MenuSuite in Object Designer, the MenuSuite object that you create inherits from the base 1010 Dept - MBS MenuSuite object. If there are other MenuSuite objects in Object Designer, the new MenuSuite inherits from one of the MenuSuite objects in Object Designer. The new MenuSuite object inherits from the MenuSuite object in the design level that is above the design level that you selected for the new MenuSuite object.

To create a new MenuSuite in the Object Designer, click **MenuSuite**, and then click **New**. To design an existing MenuSuite, select it, and then click **Design**.



Note: You can only have one Navigation Pane Designer open at the same time. Before you can create a new MenuSuite or edit another one, first make sure that the Navigation Pane Designer is closed.

Menus

Menus in the Navigation Pane Designer correspond to departments in the **Departments** menu. In the Navigation Pane Designer, each menu is displayed as a menu button with an icon and a caption. When you design MenuSuites, the Navigation Pane Designer header indicates the design level and the name of the menu that you are currently designing.

The “Navigation Pane Designer” image shows the Navigation Pane Designer for the **Financial Management** menu in the Company design level.

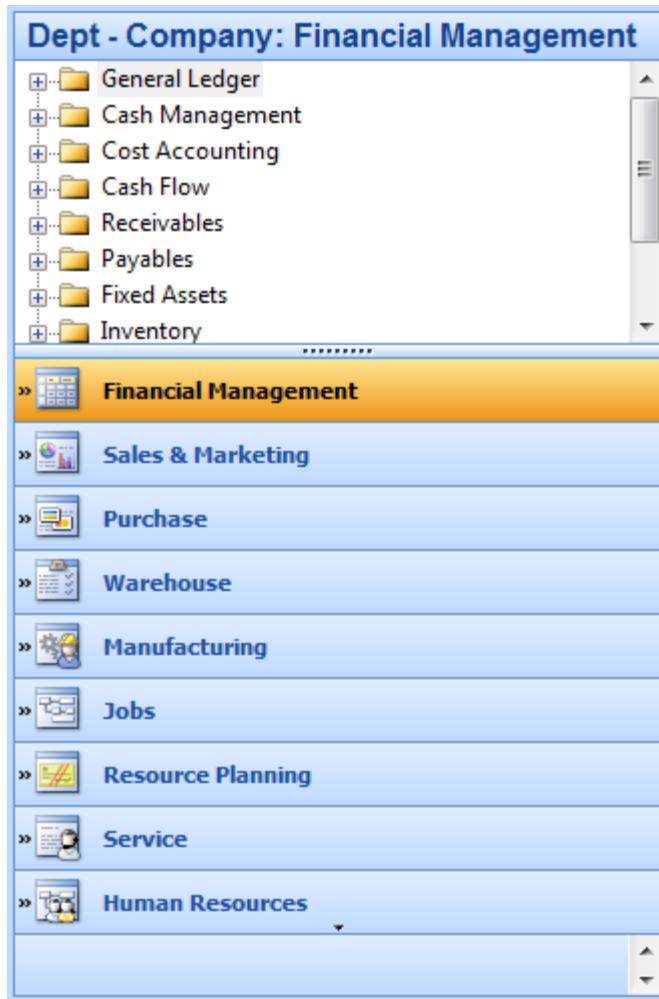


FIGURE 9.18: NAVIGATION PANE DESIGNER

Manage menus in the Navigation Pane Designer by right-clicking any of the existing menus, and then clicking any of the available options.

The “Managing Menus in the Navigation Pane Designer” image shows the shortcut menu in the Navigation Pane Designer after right-clicking any of the menus.

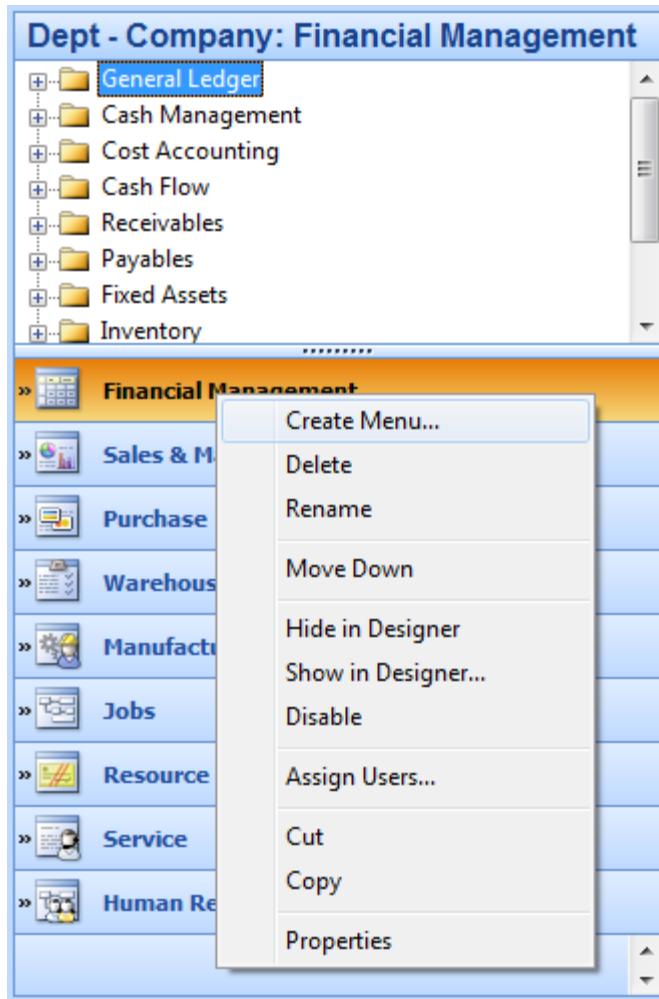


FIGURE 9.19: MANAGING MENUS IN THE NAVIGATION PANE DESIGNER

The following table explains the available options.

Option	Remarks
Create Menu...	Creates a new menu.
Delete	Deletes the selected menu.
Rename	Changes the caption of the selected menu.
Move Up / Down	Moves the selected menu up or down. Only the options that you can click are displayed.
Hide in Designer	Hides the menu from the Navigation Pane Designer. This does not hide the department from the Departments menu.
Show in Designer...	Shows the list of all hidden menus and lets you display the menus that you select.

Option	Remarks
Properties	Displays the properties for the menu. The properties include the Caption, CaptionML, and the Bitmap. The Bitmap property defines the icon that is shown next to the department in the Departments menu. Valid values for it are 0 to 15.

 **Note:** *The Disable and Assign Users options are obsolete in Microsoft Dynamics NAV 2013 and have no effect.*

In the list of menu buttons, all menus (departments) that are inherited have the » symbol on the menu button. This symbol comes in front of the name of the menu on the menu button. Any menus that you created at the current design level do not have this symbol.

The "Seminar Management Menu" image shows the Navigation Pane Designer with the new **Seminar Management** menu that was created at the Company design level. The **Seminar Management** menu does not show the » symbol on the menu button to the left of the icon, whereas all other menus do.



FIGURE 9.20: SEMINAR MANAGEMENT MENU

Groups and Items

You can create groups and items in a menu. *Groups* are categories of items and they define the navigation hierarchy. *Items* are actions that users can click. Items run different application objects.

To manage groups or items, right-click the Navigation Pane Designer anywhere in the design area below the header.

The “Managing Menus in the Navigation Pane Designer” image shows the shortcut menu in the Navigation Pane Designer after you right-click any groups or items, or if you right-click in the blank design area below the header.

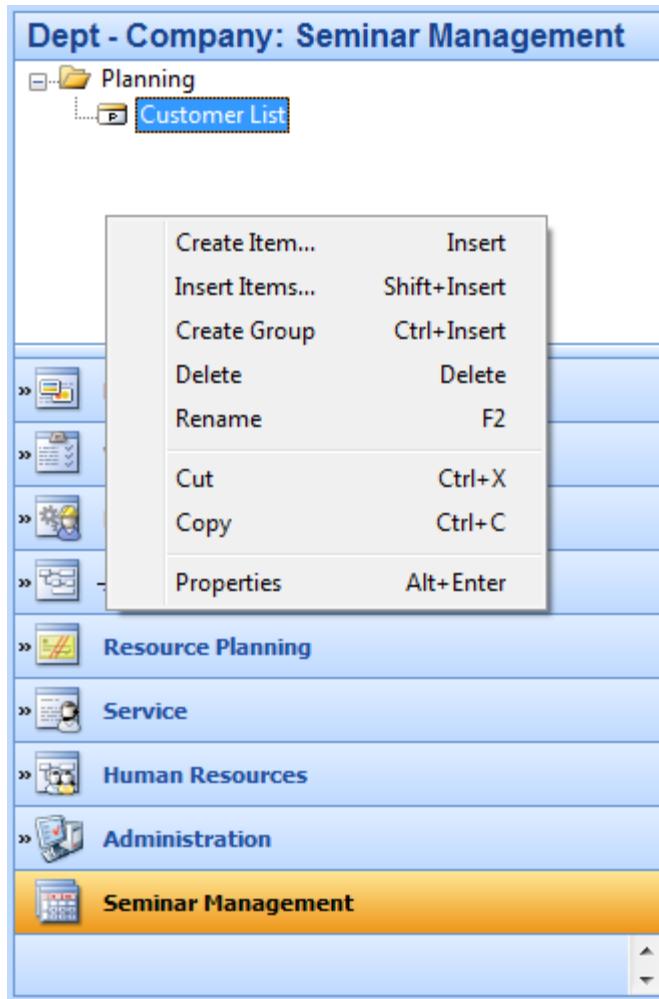


FIGURE 9.21: MANAGING MENUS IN THE NAVIGATION PANE DESIGNER

The following table explains the options.

Option	Remarks
Create Item...	Opens the Create Item window where you can define the item properties for a new item.
Insert Items...	Opens the Insert Items window where you can select from existing items in other menus. You can add items as a copy in the current group in the current menu.
Create Group	Creates a new group.
Delete	Deletes the currently highlighted item or group.

Option	Remarks
Properties	Shows the Item Properties window that lets you define item properties.

The **Create Item** and **Item Properties** windows let you define the same information for an item.

The "Item Properties" image shows the **Item Properties** window.

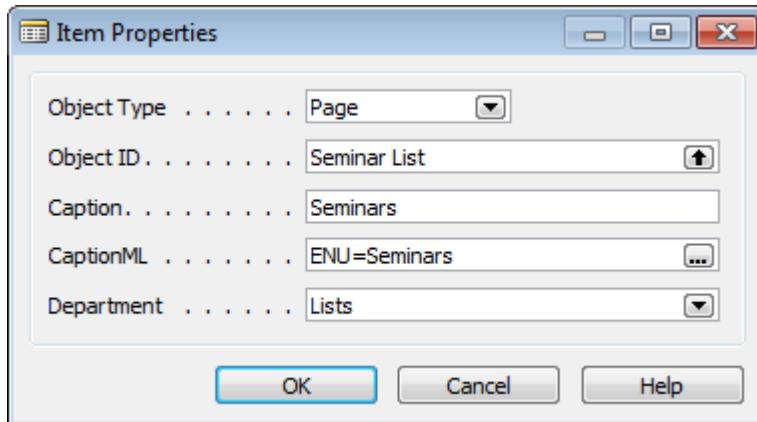


FIGURE 9.22: ITEM PROPERTIES WINDOW

You can define the following properties for each item.

Property	Remarks
Object Type	The type of the object that the item runs. It can be one of the following: <ul style="list-style-type: none"> • Report • XMLport • Codeunit • Page • Query
Object ID	The ID or name of the object that the item runs. You can look up the object or enter its ID or name manually.
Caption	The caption of the object in the currently active language. The caption is suggested automatically when you select the Object ID. However, you can later change it.

Property	Remarks
CaptionML	The multilanguage caption of the object. Similar to the Caption, it is suggested automatically. However, you can change it.
Department	The department category where the item appears in a Department page, and also the icon that is shown with the item in the search results when you use the Search field. It can have one of the following values: <ul style="list-style-type: none">• Lists• Tasks• Reports and Analysis• Documents• History• Administration

The “Departments” image shows the **Departments** page. There are menus on the first level. These menus map to the departments. The first-level groups of each department are shown together with the department. You can select by category after the list of the departments. Each department's categories are represented by the available items that are shown in the list.

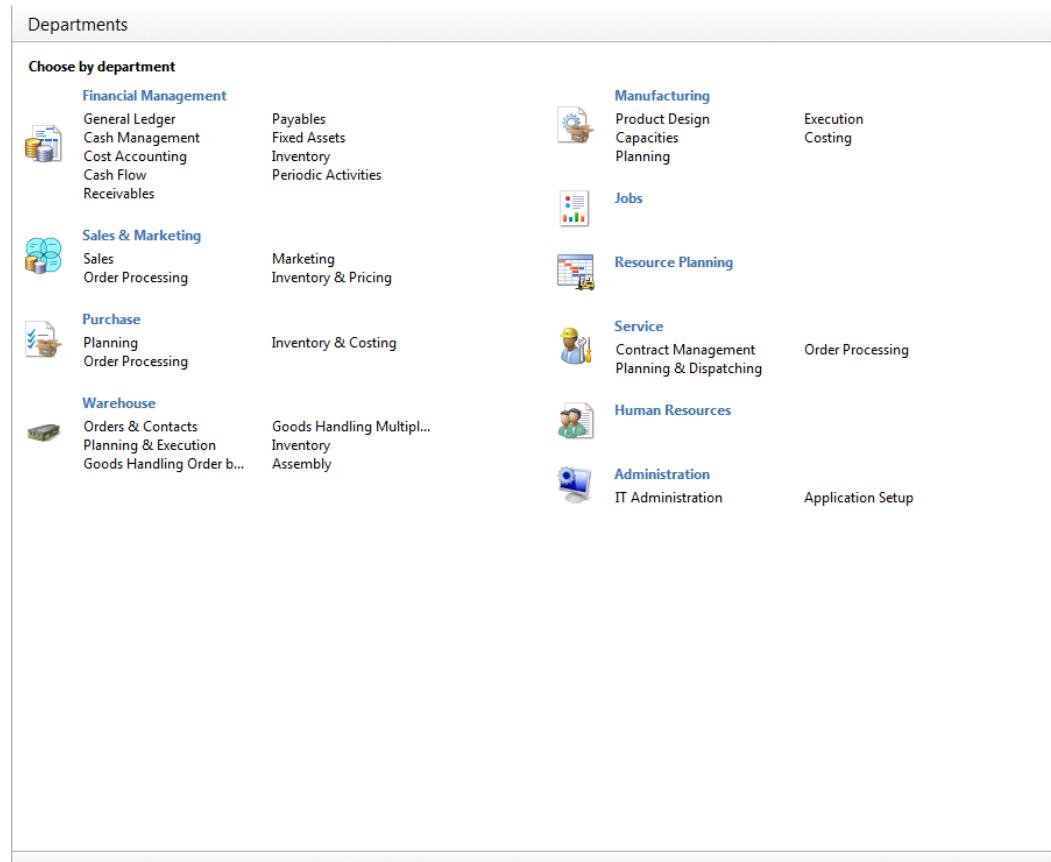


FIGURE 9.23: DEPARTMENTS WINDOW

Search

Microsoft Dynamics NAV 2013 client for Windows lets you search for application objects by using the **Search** field in the upper-right corner of the address bar. When you start to type characters in the **Search** field, a drop-down list shows object names that contain the characters that you enter. The drop-down list changes as you type more characters. Select the correct object from the list when it is displayed.

The second column in the drop-down list shows the navigation paths to the found objects. You can use the navigation to move to the **Departments** page where that page exists.

The Search feature can find only objects that can be accessed from the navigation pane. This includes any actions that are available in the **Home** menu, any of the additional menus (menu buttons) in the Role Center, and any actions in Departments.

Seminar Management Department Page

Provide a Role Center for users to work with a new application area as a way to increase user productivity. However, the Role Center alone is not the only task that is available to improve the user experience with the Microsoft Dynamics NAV 2013 client for Windows.

Many users do not belong to the seminar manager role, and use different Role Centers than the Seminar Manager Role Center. For example, accountants generally use the general ledger functionality, and accounts payable employees usually access the purchases and payables application area. However, these users may require occasional access to seminar-related information that their role centers do not provide.

To give all users access to the Seminar Management application area, you must define the **Seminar Management** department page. By providing this page, you make sure that every user can access the Seminar Management functionality in Departments, and that all objects that are related to Seminar Management can be accessed by using the **Search** field.

Solution Design

Similar to Role Center functionality, CRONUS International Ltd. functional requirements do not explicitly mention departments. However, the following nonfunctional requirement applies: The solution must be consistent, user-friendly, and easy to learn and to use. Any custom-built functionality must follow the standards, principles, and best practices of Microsoft Dynamics NAV 2013, and must seamlessly integrate into the standard application.

If you do not provide a department page for Seminar Management, your solution does not meet the goals of this requirement. Therefore, you must develop a new MenuSuite object that contains the menu definition for the **Seminar Management** department page.

Development

Your goal is to develop the **Seminar Management** department page that integrates in the **Departments** page in Microsoft Dynamics NAV 2013. To do this, you must develop the new **Seminar Management** menu in a new company-level MenuSuite.

The “Seminar Management menu in the Navigation Pane Designer” image shows the Navigation Pane Designer for the **Seminar Management** menu.

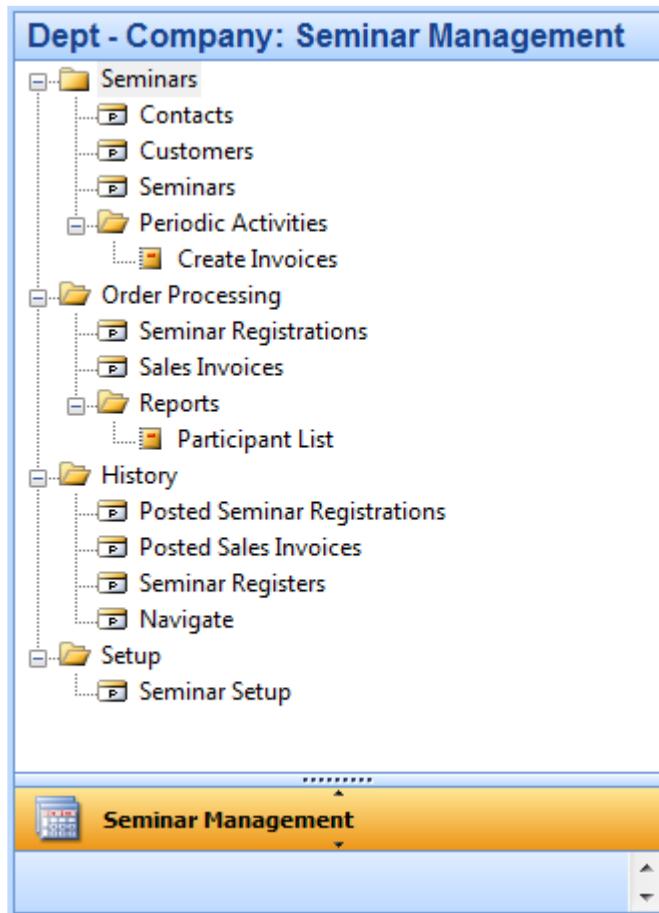


FIGURE 9.24: SEMINAR MANAGEMENT MENU IN THE NAVIGATION PANE DESIGNER

C/SIDE Solution Development in Microsoft Dynamics® NAV 2013

The "Seminar Management Department Page" image shows the **Seminar Management** Department page in the Microsoft Dynamics NAV 2013 client for Windows.

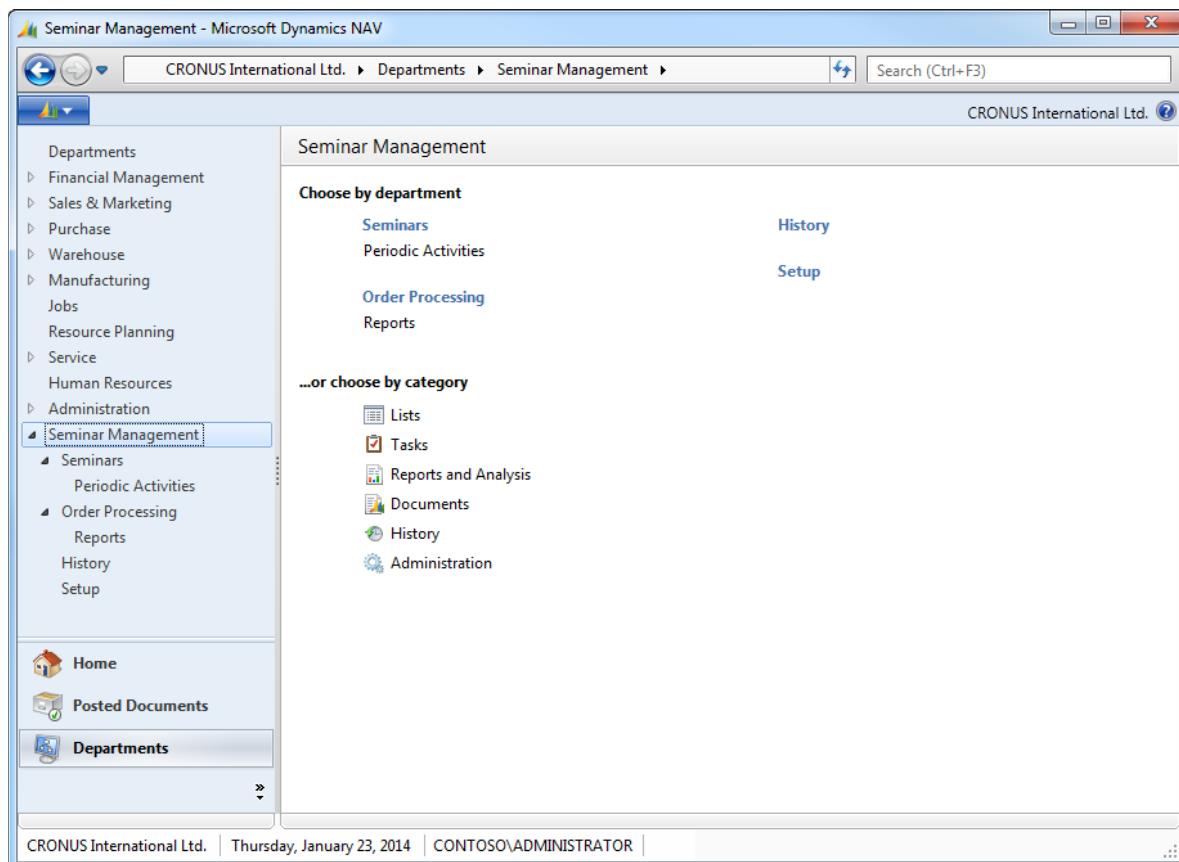


FIGURE 9.25: SEMINAR MANAGEMENT DEPARTMENT PAGE

Lab 9.2: Create Seminar Management Department Page

Scenario

You must develop the **Seminar Management** department page to enable users to access the Seminar Management application area functionality from the **Departments** page and to make Seminar Management objects available to search.

Exercise 1: Create and Design the MenuSuite

Exercise Scenario

To create the **Seminar Management** department page and make Seminar Management objects available to search, you must create a new MenuSuite, and then create a new menu with appropriate groups and items.

Task 1: Create the MenuSuite

High Level Steps

1. Create and save a new MenuSuite at the company level.
2. Add the **Seminar Management** menu.

Detailed Steps

1. Create and save a new MenuSuite at the company level.
 - a. In Object Designer, click **MenuSuite**, and then click **New**.
 - b. In the **Design Level** window, select the Dept – Company level, and then click **OK**.
 - c. Click any of the menu buttons in the Navigation Pane Designer.
 - d. On the **File** menu, click **Save**.
2. Add the **Seminar Management** menu.
 - a. Right-click any of the menu buttons.
 - b. In the shortcut menu, click **Create Menu**.
 - c. In the **Create Menu** window in the **Caption** field, enter "Seminar Management", in the **Bitmap** field, enter "13", and then click **OK**.

Task 2: Define the Group Hierarchy

High Level Steps

1. Define the groups as shown in Figure 9.24.

Detailed Steps

1. Define the groups as shown in Figure 9.24.
 - a. Right-click the **[Empty Menu]** item in the Navigation Pane Designer, and then click **Create Group**.
 - b. Right-click the **New Group**, click **Rename**, and then enter "Seminars".
 - c. Expand the Seminars group.
 - d. Right-click the **[Empty Group]** item and then click **Create Group**.
 - e. Rename the New Group to Periodic Activities.
 - f. Right-click the Seminars group, create a new group, and then rename it "Order Processing".
 - g. Expand the Order Processing group, right-click the **[Empty Group]** item, create a new group, and rename it "Reports".
 - h. Right-click the Seminars group, create a new group, and then rename it "History".
 - i. Right-click the Seminars group, create a new group, and then rename it "Setup".

Task 3: Add Items to Groups

High Level Steps

1. Add items to groups as shown in Figure 9.24.

Detailed Steps

1. Add items to groups as shown in Figure 9.24.
 - a. Under the Seminars group, right-click the Periodic Activities group, and then click **Create Item**.
 - b. In the **Create Item** window, enter the following information.

Field	Value
Object Type	Page
Object ID	Contact List
Caption	Contacts
Department	Lists

C.

Module 9: Role Tailoring

- d. Click **OK** to create the new item.
- e. Under Seminars group, create the following two items.

Object ID	Caption	Department
Page Customer List	Customers	Lists
Page Seminar List	Seminars	Lists

- f. Under the Periodic Activities group, right-click the **[Empty Group]** item, and then create the following item.

Object ID	Caption	Department
Report Create Seminar Invoices	Create Invoices	Tasks

- g. Under the Order Processing group, create the following items.

Object ID	Caption	Department
Page Seminar Registration List	Seminar Registrations	Documents
Page Sales Invoice List	Sales Invoices	Documents

- h. Under the Reports group, create the following item.

Object ID	Caption	Department
Report Seminar Reg.- Participant list	Participant List	Reports and Analysis

- i. Under the History group, create the following items.

Object ID	Caption	Department
Page Posted Seminar Reg. List	Seminar Registrations	History
Page Posted Sales Invoices	Posted Sales Invoices	History
Page Seminar Registers	Seminar Registers	History
Page Navigate	Navigate	History

- j. Under the Setup group, create the following items.

Object ID	Caption	Department
Page Seminar Setup	Seminar Setup	Administration

- k. Save the MenuSuite object.

Module Review

Module Review and Takeaways

Tailored clients for Microsoft Dynamics NAV 2013 give users Role Centers that increase user productivity. It enables users to focus on their most important tasks and locate information quickly and efficiently without spending unnecessary time on a complex user interface.

A Role Center typically includes the following:

- An Activities part
- Several lists that enable users to select their favorite records, such as customers or items
- Several system parts, such as Outlook or My Notifications.

The Activities part contains several groups of cues that are defined as flow fields in the **Activities** page source table.

The Role Center also defines the contents of the navigation pane that is visible and available on every list place or department page. It also defines the contents of the ribbon when users click the Role Center item in the **Home** menu.

Even though the Role Center is the most important feature that increases productivity based on user role, you must always provide a department page for any application area that you develop. This guarantees that the users can use the Departments page to manually browse to your custom application area functionality, and that all the pages that you create for a new application area can be accessed from the Search field.

Test Your Knowledge

Test your knowledge with the following questions.

1. Which page type do you use to define role centers for profiles in Microsoft Dynamics NAV 2013?

Module 9: Role Tailoring

2. Which control type do you use to show stacks of documents called cues?
What type of control must be the parent?

3. What kind of data is contained in fields shown by cues?

4. How do you add new menus to the Navigation pane of the RoleTailored client?

5. Role center pages are data bound.

True

False

6. An action in role center page can contain code in the OnAction trigger. This code executes when the user clicks the action.

True

False

Test Your Knowledge Solutions

Module Review and Takeaways

1. Which page type do you use to define role centers for profiles in Microsoft Dynamics NAV 2013?

MODEL ANSWER:

RoleCenter

2. Which control type do you use to show stacks of documents called cues?
What type of control must be the parent?

MODEL ANSWER:

Use Field controls within CueGroup controls to show cues.

3. What kind of data is contained in fields shown by cues?

MODEL ANSWER:

To show as cues, fields must be of integer type. Typically, they are FlowFields with Count calculation type.

4. How do you add new menus to the Navigation pane of the RoleTailored client?

MODEL ANSWER:

For each new menu, define the ActivityButtons action container.

5. Role center pages are data bound.

() True

() False

6. An action in role center page can contain code in the OnAction trigger. This code executes when the user clicks the action.

() True

() False

MODULE 10: INTERFACES

Module Overview

Microsoft Dynamics NAV® 2013 frequently exchanges information with external systems. For example, you may have to send electronic payment information to a bank, exchange purchase order information with your vendor's business management application, or receive item information from a bar code or RFID reader. In those situations, Microsoft Dynamics NAV 2013 performs only part of the transaction or the process. It interfaces with external software or hardware to perform other parts of the transaction or the process.

Microsoft Dynamics NAV 2013 includes several features that streamline interfacing to external applications or devices. This module covers the following features:

- Component Object Model (COM) Technologies
- File handing
- Microsoft .NET Interoperability

Objectives

- Explain how to use Automation and OCX to perform tasks with other applications.
- Describe file handling functions to import or export data.
- Design and implement email capability.

Prerequisite Knowledge

To interface Microsoft Dynamics NAV 2013 with other applications, use Component Object Model (COM) technologies in the Microsoft Dynamics NAV Development Environment to extend the functionality of Microsoft Dynamics NAV 2013. Also use external files to exchange information between Microsoft Dynamics NAV 2013 and third-party applications or devices.



Component Object Model (COM) Technologies

Use Component Object Model (COM) technologies in the Microsoft Dynamics NAV Development Environment to extend the functionality of Microsoft Dynamics NAV 2013. In the development environment, you can use two types of COM technologies: automation and custom controls (OCX).



Note: Automation and OCX objects are not supported under the Microsoft Dynamics NAV Web client and SharePoint applications that run Microsoft Dynamics NAV Portal Framework.

Automation

Automation is a client/server infrastructure that enables one application to access and communicate with another application. By using automation, an application, such as Microsoft Office Word, exposes its internal functions and routines as automation objects. Microsoft Dynamics NAV 2013 accesses these objects through an automation client that runs in the Microsoft Dynamics NAV 2013 client for Windows. The application that exposes the automation object, such as Word, acts as the automation server, and the automation client acts as the client.

Automation enables tasks that run manually to run automatically instead. For example, you can write a script to extract data from a database, put the data into a Microsoft Office Excel workbook, and then display the data graphically.



Note: Microsoft Dynamics NAV 2013 Server is a 64-bit application. However, not all COM components can run on 64-bit operating systems. Therefore, automation objects can run only on the client side. If you try to create or run any automation objects on the server side, a run-time error occurs. Instead of implementing server-side automation objects, use Microsoft .NET Framework Interoperability.

Custom Controls (OCX)

Custom controls are OLE Control Extensions (OCX) or ActiveX controls. These are specific types of Automation objects. OCX and ActiveX controls are generally small programs or application objects that you start from the Microsoft Dynamics NAV 2013 application to perform a specific function or task. Use custom controls for various types of tasks.

By default, there are several available custom controls in Microsoft Dynamics NAV 2013 on the **Tools** menu in the development environment. To develop custom controls, you can use tools such as Microsoft Visual C++ or Microsoft Visual Basic. Both products use wizards that make it easy to develop COM objects. You can also develop functional controls without understanding the complex details of COM.

Only nonvisual controls are supported. You cannot use a control to add graphical elements to a Microsoft Dynamics NAV object. For example, you cannot add a third-party control to a page. However, the control can display information and interact with the user in its own window.

 **Additional Reading:** If you must add visual, graphical, or user-interface elements to a page in the Microsoft Dynamics NAV 2013 client for Windows, use control add-ins. Control add-ins are delivered as Microsoft .NET Framework-based assemblies that you install on computers that run the Microsoft Dynamics NAV 2013 client for Windows. You register the control add-in in Microsoft Dynamics NAV Server. Then use the control add-in with pages. To read more about the control add-ins, refer to the RoleTailored Client Control Add-in Overview topic in Microsoft Dynamics NAV Developer and IT Pro Help.

Automation and OCX Variables

To use an automation object or ActiveX control, you must declare a variable of type Automation or OCX. Before you can use an automation object, you must create an instance of the object by calling the **CREATE** function.

The following is the syntax of the **CREATE** function..

[<i>Ok :=</i>] CREATE(<i>Automation</i> [, <i>NewServer</i>] [, <i>OnClient</i>])
--

C/SIDE Solution Development in Microsoft Dynamics® NAV 2013

The following are the parameters of the CREATE function.

Parameter	Type	Remarks
Automation	Automation	The Automation variable that you previously declared.
NewServer	Boolean	If NewServer is FALSE (this is the default value), then the CREATE function tries to reuse an already running instance of the automation server that is referenced by Automation before it creates a new instance. If NewServer is TRUE, then the CREATE function always creates a new instance of the automation server.
OnClient	Boolean	This parameter is used for backward compatibility. In Microsoft Dynamics NAV 2013 it must evaluate to TRUE. You can either set the parameter to TRUE, or define a Boolean type variable that evaluates to TRUE. If you omit this parameter or use the FALSE Boolean constant, a compile-time error occurs. If you use an expression or a variable that evaluates to FALSE at run time, then a run-time error occurs.

If you test for the return value, you receive TRUE if the automation object was successfully created, and FALSE if it could not be created. If you do not test for the return value, and the creation fails, then a run-time error occurs.

COM Considerations

If you want only to use existing automation objects or custom controls, then you do not require extensive COM knowledge. The functionality that is provided by a COM object is the same as any C/AL function.

However, following are several items to consider when you use automation:

- Exception handling is not supported. You cannot retrieve information about exceptions from a control or automation server.
- You must install the COM object or the whole application that exposes its functionality through automation on every computer that runs Microsoft Dynamics NAV 2012 client for Windows.
- C/AL data types do not map directly to COM data types. So, the mechanisms for calling methods in a control, passing parameters, and receiving return values are somewhat complex. You frequently must study the documentation for the automation server to learn about the actual data types that methods of COM objects accept or return. This is especially true if methods or properties use the IDispatch type or enumerations.

 **Additional Reading:** To learn more about how to use COM in Microsoft Dynamics NAV 2013, refer to the Extending Microsoft Dynamics NAV Using COM topic in the Microsoft Dynamics NAV Developer and IT Pro Help.

Demonstration: Use Automation to Create a Chart in Microsoft Excel

In this demonstration, you transfer data from the **G/L Entry** table to Microsoft Office Excel and create a chart. This example shows how to handle enumerations by creating a chart in Excel that shows the distribution of personnel expenses by departments.

Demonstration Steps

1. Create a codeunit and declare variables.
 - a. In Object Designer, click **Codeunit**, and then click **New** to create a new codeunit.
 - b. On the **View** menu, click **C/AL Globals**.
 - c. On the **Variables** tab, define the following variables.

Name	DataType	Subtype
GLEntry	Record	G/L Entry
xlApp	Automation	Application
xlBook	Automation	Workbook
xlSheet	Automation	Worksheet
xlChart	Automation	Chart
xlRange	Automation	Range

 **Note:** All of the automation objects must be from the Microsoft Excel 14.0 Object Library automation server. When creating these variables, in the Subtype property, click the **AssistEdit** button, then in the Automation Server text box, look up the Microsoft Excel 14.0 Object Library automation server. Then, select the appropriate class, as indicated in the Subtype column.

- d. Close the **C/AL Globals** window.
2. Add code to filter the **G/L Entry** table to only the relevant records.
 - a. In the OnRun trigger, enter the following C/AL code.

```
GLEntry.SETCURRENTKEY(  
    "G/L Account No.",  
    "Business Unit Code",  
    "Global Dimension 1 Code",  
    "Global Dimension 2 Code",  
    "Posting Date");  
  
GLEntry.SETFILTER("G/L Account No.",'8700..8790');
```

3. Add code to create an instance of Excel.
 - a. Append the following C/AL code to the end of the OnRun trigger.

```
CREATE(xlApp, FALSE, TRUE);  
  
xlApp.Visible := TRUE;
```

4. Add code to add a new workbook to Excel.
 - a. Append the following C/AL code to the end of the OnRun trigger.

```
xlBook := xlApp.Workbooks.Add(-4167);  
  
xlSheet:= xlApp.ActiveSheet;  
  
xlSheet.Name := 'Personnel Expenses';
```

 **Note:** In the code example that was mentioned earlier, value -4167 substitutes the xlWBATWorksheet enumerator. To find enumerator values of COM classes, you can use Visual Studio. For Excel, you can also use Visual Basic Editor. This is built into Excel.

Module 10: Interfaces

5. Add code to transfer data from Microsoft Dynamics NAV 2013 to Excel.
 - a. Append the following code to the end of the OnRun trigger.

```
GLEntry.SETRANGE("Global Dimension 1 Code",'ADM');

GLEntry.CALCSUMS(Amount);

xlSheet.Range('A2').Value := 'Administration';

xlSheet.Range('A3').Value := GLEntry.Amount;

GLEntry.SETRANGE("Global Dimension 1 Code",'PROD');

GLEntry.CALCSUMS(Amount);

xlSheet.Range('B2').Value := 'Production';

xlSheet.Range('B3').Value := GLEntry.Amount;

GLEntry.SETRANGE("Global Dimension 1 Code",'SALES');

GLEntry.CALCSUMS(Amount);

xlSheet.Range('C2').Value := 'Sales';

xlSheet.Range('C3').Value := GLEntry.Amount;
```

6. Add code to create the chart.
 - a. Append the following code to the end of the OnRun trigger.

```
xlRange := xlSheet.Range('A2:C3');

xlChart := xlBook.Charts.Add;

xlChart.Name := 'Personnel Expenses - Chart';

xlChart.ChartWizard(xlRange,-4102,7,1,1,0,0,'Personnel Expenses');
```

7. Test the code.
 - a. Save the codeunit as 90005, Create Excel Chart.
 - b. Close the **C/AL Editor** window.
 - c. In Object Designer, locate the codeunit 90005, Create Excel Chart.
 - d. Click **Run**.

C/SIDE Solution Development in Microsoft Dynamics® NAV 2013

The “Excel Chart Created through Automation” figure shows the output of the **Create Excel Chart** codeunit.

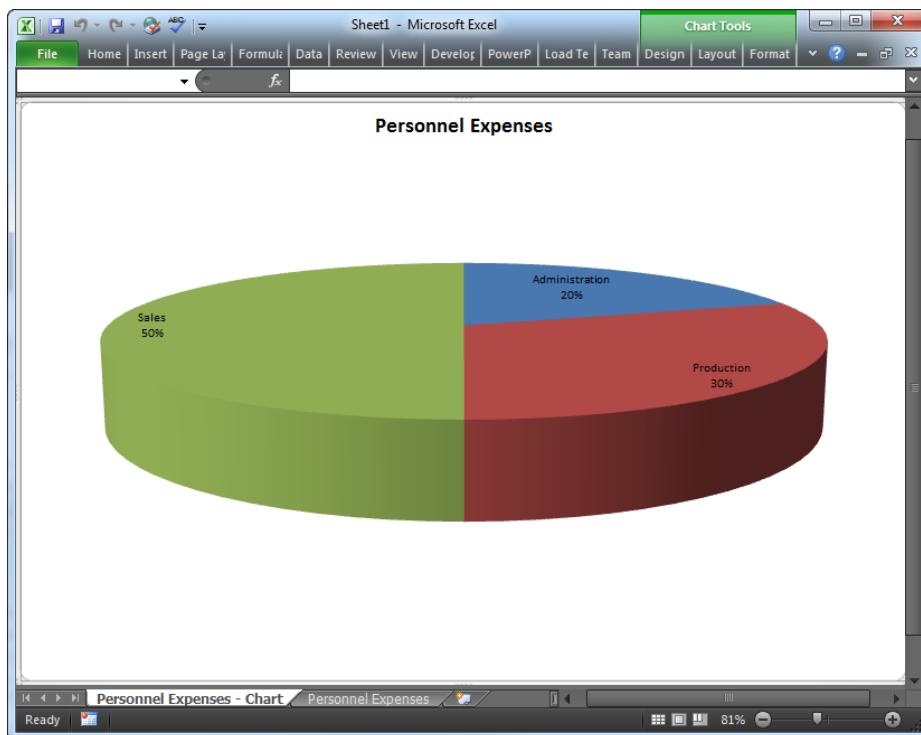


FIGURE 10.1: EXCEL CHART CREATED THROUGH AUTOMATION

File Handling

Use file-handling to import or export text-based data files that require parsing to interpret on input, or special formatting when output. For example, you can use file handling to import data that is not record-oriented and does not have either fixed-width or delimited structure. You can import data from or export data to any external file through file variables.

The File Data Type

To access an external file in C/SIDE, first declare a variable of type File. This is a complex data type. It has many functions that are used to open, read, write, and close external files.

Use each file variable to access one file at a time. If you want to access multiple files at the same time, you must declare one file variable for each file that you access.

The following table explains the most important file functions that are used to handle the files.

Function	Remarks
OPEN	Opens an ASCII or binary file. This function does not create the file if it does not exist. If you call OPEN on a file variable that refers to an open file, then a run-time error occurs. The OPEN function does not automatically close the existing file and open a new file.
CREATE	Creates and opens an ASCII or binary file. If the file already exists, it is truncated and then opened. Similar to OPEN , if you call CREATE on a file variable that refers to an open file, then a run-time error occurs.
CLOSE	Closes a file that was opened by OPEN or CREATE . If the file is not open, a run-time error occurs. You always must call explicitly CLOSE if you intend to reuse the same file variable for accessing more than one file.
WRITEMODE	Sets the read/write status of a file variable before you open the file. You cannot write to files that are not in write mode. You can also use this function to test whether the file is in write mode. You cannot switch between read and write modes. Therefore, make sure that you call WRITEMODE before you open a file.
TEXTMODE	Sets whether a file should be opened as an ASCII file or a binary file. Retrieves the current setting of this option for a file. You cannot switch between text and binary modes. Therefore, make sure that you call TEXTMODE before you open or create a file.
READ	Reads from an ASCII or binary file. If the file is in text mode, it reads a single line of text into a text variable. If the file is in binary mode, it reads a single variable of a simple C/AL data type. READ does this by reading as many bytes from the external file as the variable occupies in memory.
WRITE	Writes to an ASCII or binary file. If the file is in text mode, it writes the variable formatted as text followed by a new line character. If the file is in binary mode, it writes the variable exactly as it is stored in memory.

Function	Remarks
POS	Retrieves the current position of the file pointer in an ASCII or binary file.
LEN	Retrieves the length of an ASCII or a binary file. The length is always reported in bytes.

 **Additional Reading:** There are many more functions on the File data type. To learn more about these functions, refer to the Microsoft Dynamics NAV Developer and IT Pro Help.

ASCII and Binary Modes

There are two methods for reading or writing data in external files. You set the desired method by calling the **TEXTMODE** function. Following are two possible settings:

- ASCII (**TEXTMODE**=TRUE) – Each file access reads or writes a line of text. Use any type variable. It is converted to or from text during the reading or writing.
- BINARY (**TEXTMODE**=FALSE) – Each file access reads or writes a single variable. The variable is read or written in its binary format as it is stored in memory. The internal format can only be read if it is written in the same byte sequence that Microsoft Dynamics NAV 2013 uses to store variables. This is not a limitation when it reads binary files that are created by Microsoft Dynamics NAV 2013. However, you may experience issues when you handle files that were created by other applications or systems.

Demonstration: Write to and Read from Binary Files

In this demonstration, you write values of three variables into a binary file, clear the variables, and then read those variables back from the binary file.

Demonstration Steps

1. Create a codeunit and declare variables.
 - a. In Object Designer, click **Codeunit**, and then click **New** to create a new codeunit.
 - b. On the **View** menu, click **C/AL Globals**.
 - c. On the **Variables** tab, define the following variables.

Name	DataType	Length
Int	Integer	
Bool	Boolean	

Name	DataType	Length
String	Text	250
Bin	File	

2. Create two functions to write and read contents of the variables to and from a file.
- On the **Functions** tab, define the following two functions: **Save** and **Load**. The functions do not accept any parameters or return any values.
 - Close the **C/AL Globals** window.
 - In the function trigger of the **Save** function, enter the following C/AL code.

```
BinFile.TEXTMODE(FALSE);

BinFile.CREATE('C:\Temp\Variables.txt');

BinFile.WRITE(Int);

BinFile.WRITE(Bool);

BinFile.WRITE(String);

BinFile.CLOSE;
```

- In the function trigger of the **Load** function, enter the following C/AL code.

```
BinFile.TEXTMODE(FALSE);

BinFile.OPEN('C:\Temp\Variables.txt');

BinFile.READ(Int);

BinFile.READ(Bool);

BinFile.READ(String);

BinFile.CLOSE;
```

3. Add code that does the following:
- Sets values to variables.
 - Saves the variables.
 - Clears the variables.
 - Loads the variables.
 - Displays variable values to the user.

- a. In the OnRun trigger, enter the following code.

```
Int := 15;  
  
Bool := TRUE;  
  
String := 'Hello, World!';  
  
Save;  
  
CLEARALL;  
  
Load;  
  
MESSAGE('Int: %1\Bool: %2\String: %3',  
        Int,Bool,String);
```

4. Test the code.
 - a. Save the codeunit as 90004, Read and Write Variables.
 - b. Close the **C/AL Editor** window.
 - c. In Object Designer, locate the codeunit 90004, Read and Write Variables.
 - d. In Windows Explorer, create the C:\Temp folder.
 - e. Click **Run**.

Microsoft .NET Interoperability

You can extend Microsoft Dynamics NAV 2013 with functionality that is available in Microsoft .NET Framework assemblies. You can take advantage of the .NET Framework interoperability so that Microsoft Dynamics NAV objects can interact with .NET Framework objects. In the Microsoft Dynamics NAV objects, you can reference .NET Framework assemblies and call their members directly from C/AL code. Use assemblies from the .NET Framework class library. These are found in the global assembly cache, your own custom assemblies, or third-party assemblies.

.NET Framework interoperability offers an alternative to COM so that you can extend your solution. For example, you can use .NET Framework interoperability to do the following:

- Consume web services.
- Integrate with Microsoft Office products.
- Create .NET Framework applications that target the Microsoft Dynamics NAV 2013 client for Windows.



Note: .NET Framework objects can run both on the server side and client side. Client-side objects are not supported under the Microsoft Dynamics NAV Web client and SharePoint applications that run Microsoft Dynamics NAV Portal Framework.



Additional Reading: To learn more about Microsoft .NET Framework interoperability, refer to Microsoft .NET Framework Interoperability module of the course C/SIDE Introduction in Microsoft Dynamics NAV 2013, or to the Extending Microsoft Dynamics NAV Using Microsoft .NET Framework Interoperability topic in Microsoft Dynamics NAV 2013 Developer and IT Pro Help.

Email Confirmation

CRONUS International Ltd. wants to automate its communication with their customers as much as they can. To meet this requirement, you must design and develop a solution that sends email confirmations to all seminar participants.



Solution Design

The Functional Requirements Document for Microsoft Dynamics NAV 2013 implementation at CRONUS International Ltd. states the following:

- When a seminar is confirmed, seminar managers must be able to send an automatic email confirmation to all registered seminar participants.
- Seminar managers must be able to easily customize the template for the seminar confirmation notification email. The template must allow for the following:
 - Ability to enter free text
 - Placeholders for recipient name
 - Seminar name
 - Seminar starting date

Sending Email from Microsoft Dynamics NAV 2013

For the first requirement, you must provide a capability to send email messages from Microsoft Dynamics NAV 2013. Following are several capabilities that help you develop this functionality and meet this requirement:

- Microsoft Outlook automation library
- Microsoft Collaboration Data Objects (CDO) automation library
- Microsoft.Dynamics.Nav.SMTP .NET assembly that is provided with Microsoft Dynamics NAV 2013

- Codeunit 397, Mail
- Codeunit 400, SMTP Mail

Generally, when you select the optimal solution from a range of options, you should always opt for the simplest choice that meets the customer's requirements. It should provide as many of the following criteria as possible:

- Ease of development and maintenance
- Maximum forward compatibility
- Least amount of dependencies on outside applications
- No unnecessary dependencies on third-party products
- No redundancies with existing capabilities
- Fewest changes to customer infrastructure

When you compare these choices with the list of available technologies, you should discard automation, because it does not provide maximum forward compatibility. It runs on the client that requires the customer to install and maintain a compatible version of the automation server on each client computer.

Do not use the Microsoft.Dynamics.Nav.SMTP .NET assembly directly. It is not the simplest solution to develop and maintain, and it may not be sufficiently forward compatible. It also creates redundancy with the existing functionality of Mail and SMTP Mail codeunits.

The Mail and SMTP Mail codeunits already provide email capabilities. Therefore, you should select one of these two options instead of developing a completely custom email sending capability. The Mail codeunit depends on Outlook. Therefore, it may require configuration and maintenance on all client computers. The SMTP Mail codeunit depends on an external SMTP server. Because CRONUS International Ltd. uses a dedicated SMTP server, this requires less maintenance and no infrastructure changes to the CRONUS environment.

Therefore, your solution should use the SMTP Mail codeunit to manage sending email confirmations to seminar participants.

Customizing the Template

To enable seminar managers to customize the template, you can select among several options, such as the following:

- Importing a template into a BLOB field in a setup table
- Providing a table with text fields where each row represents a line in the confirmation email
- Using an external template file
- Using a file in a SharePoint library

The solution in this module selects the external template file. Therefore, you must provide a text field in the **Seminar Setup** table where users can configure the path of the text file that contains the confirmation email template.

 **Note:** *The suggested solution in this module does not imply that this is the best choice. There are very strong arguments for and against any of the given choices.*

To enable users to specify free text and use placeholders for the location of a recipient's name, seminar name, and seminar starting date, use the **STRSUBSTNO** function and replace the numbered specifiers (%1, %2, %3, and so on) with variables. Because CRONUS International Ltd. needs only three variables, you can define the following placeholders to be used in the confirmation template.

Placeholder	Meaning
%1	Name of the participant
%2	Name of the seminar for which the participant is registered
%3	Starting date of the seminar

Solution Development

To meet CRONUS International Ltd. requirements, you must create and customize several objects.

Tables

Customize the following table.

Table	Changes
123456701 Seminar Setup	Add a new field to enable users to configure the path of the confirmation email template file.

Pages

Customize the following pages.

Page	Changes
123456702 Seminar Setup	Add the new field from the Seminar Setup table.
123456710 Seminar Registration	Add an action to send seminar registration emails.

Page	Changes
123456713 Seminar Registration List	Add an action to send seminar registration emails.

Codeunits

Create the following codeunit.

Codeunit	Remarks
123456705 SeminarMailManagement	Contains functions to send seminar confirmation emails to registered participants of a seminar.

Lab 10.1: Create Email Confirmations

Scenario

You are a developer at a company that is implementing Microsoft Dynamics NAV 2013 at CRONUS International Ltd. Your goal is to develop the customizations to enable seminar managers to send automated email confirmations to seminar participants.

You first have to enable users to configure the email template that the solution uses to send confirmations to registered participants. Then you must make sure that any configuration settings in CRONUS International Ltd. database enable Microsoft Dynamics NAV 2013 to interface with the SMTP server that is installed at the localhost server, and send email confirmations to participants. Finally, you must develop the codeunit that automates sending confirmations, and add an action that calls the codeunit to the Seminar Registration pages.

Exercise 1: Import the Setup Table and Page

Exercise Scenario

Your colleague Isaac customized the **Seminar Setup** table and corresponding page to allow users to configure the path of the email template file.

Import the file that Isaac provides.

Task 1: Import the Object File

High Level Steps

1. Import the starter object.

Detailed Steps

1. Import the starter object.
 - a. Click **File > Import**.
 - b. Browse to the file Mod10\Labfiles\Lab 10.A - Starter.fob.
 - c. Click **Open**.
 - d. Click **OK** to open the Import Worksheet. The **Import Worksheet** window displays the following objects.

Type	No.	Name
Table	123456701	Seminar Setup
Page	123456702	Seminar Setup

- e. Click **Replace All**, and then click **OK** to complete the import.
- f. Click **OK** to close the **Import Objects** window.

Exercise 2: Verify the Configuration

Exercise Scenario

The next step is to make sure that Microsoft Dynamics NAV 2013 communicates with an SMTP server by configuring the **SMTP Mail Setup** page. Then you must make sure that there is an email address in the **Company Information** page. This email address is used in the **From** field for the outgoing email confirmations.

Task 1: Configure the SMTP Mail Setup Card

High Level Steps

1. Configure **SMTP Mail Setup** to use the local SMTP server.

Detailed Steps

1. Configure **SMTP Mail Setup** to use the local SMTP server.
 - a. In the **Search** field, enter "SMTP Mail Setup."
 - b. In the results, click the **SMTP Mail Setup** page.
 - c. In the **SMTP Server** field, enter "localhost."
 - d. Make sure that the **Authentication** field is set to Anonymous.



Note: Actual SMTP settings may be different for any real-life scenario.

- e. Close the **SMTP Mail Setup** page.

Task 2: Configure the Company Info Card

High Level Steps

1. Configure an email address in the Company Info card.

Detailed Steps

1. Configure an email address in the Company Info card.
 - a. In the **Search** field, enter "Company Information."
 - b. In the results, click the **Company Information** page.
 - c. On the **Communication** FastTab, in the **E-Mail** field, enter "cronus@cronus.com."
 - d. Close the **Company Information** page.

Exercise 3: Create the Codeunit

Exercise Scenario

Create a codeunit that uses the SMTP Mail codeunit to send out email messages to seminar registrants. Let the user specify the contents of the email and the recipient. Add the new email function to an action menu for easy access.

Task 1: Create the Codeunit

High Level Steps

1. Create a new codeunit to handle seminar email confirmations.
2. Declare global variables.
3. Declare global text constants.
4. Add a local function that initializes the confirmation sending process by reading relevant setup records. This guarantees that required fields contain information initializing global variables, and that the field reads the confirmation email template from the configured external file.
5. Add a local function that creates and sends an email to the recipient with the subject and body passed as parameters. The function must track the number of failures during the send process, and must return the success state of the send operation.
6. Add a function that initializes the send process. For each registration line in the registration header passed as a parameter the function sends the email confirmation to the email address of the contact that is specified in the line. If the send process succeeds, the function updates the **Confirmation Date** field in the registration line. At the conclusion, it must display a message that states that the operation succeeded, or show the number of failures with the last error message.

Detailed Steps

1. Create a new codeunit to handle seminar email confirmations.
 - a. In Object Designer, create a new codeunit.
 - b. Save the codeunit as 123456705, SeminarMailManagement.

2. Declare global variables.

- In the **C/AL Globals** window, create the following global variables.

Name	DataType	Subtype
SeminarSetup	Record	Seminar Setup
CompanyInfo	Record	Company Information
SMTP	Codeunit	SMTP Mail
Environment	DotNet	System.Environment
NoOfErrors	Integer	
ConfirmationBody	Text	



Note: You can find the System.Environment class in the mscorelib assembly.

3. Declare global text constants.

- In the **C/AL Globals** window, create the following text constants.

Name	ConstValue
Text001	Registration Confirmation for Seminar "%1"
Text002	Sending email confirmations was not successful. The last error message was:\%1
Text003	All email confirmations were sent successfully.

4. Add a local function that initializes the confirmation sending process by reading relevant setup records. This guarantees that required fields contain information initializing global variables, and that the field reads the confirmation email template from the configured external file.

- In the **C/AL Globals** window, create a new function and name it **Initialize**.
- Define the following local variables.

Name	DataType
Template	File
Line	Text

- In the **Initialize** function trigger, enter the following code.

```

SeminarSetup.GET;

SeminarSetup.TESTFIELD("Confirmation Template File");

CompanyInfo.GET;

CompanyInfo.TESTFIELD(Name);

CompanyInfo.TESTFIELD("E-Mail");

CLEAR(SMTP);

NoOfErrors := 0;

ConfirmationBody := '';

Template.TEXTMODE(TRUE);

Template.WRITEMODE(FALSE);

Template.OPEN(SeminarSetup."Confirmation Template File");

WHILE Template.POS < Template.LEN DO BEGIN

    Template.READ(Line);

    ConfirmationBody += Line + Environment.NewLine;

END;

Template.CLOSE;

```

5. Add a local function that creates and sends an email to the recipient with the subject and body passed as parameters. The function must track the number of failures during the send process, and must return the success state of the send operation.

- In the **C/AL Globals** window, create a new function and name it **SendMail**.
- Configure the function to receive the following parameters.

Var	Name	DataType
No	ToEmail	Text
No	Subject	Text
No	Body	Text

- Configure the function to return a Boolean value.
- Configure a local variable of name CompanyInfo, type Record, and subtype Company Information.

- e. In the **SendMail** function trigger, enter the following code.

```
SMTP.CreateMessage(  
    CompanyInfo.Name,  
    CompanyInfo."E-Mail",  
    ToEmail,  
    Subject,  
    Body,  
    FALSE);  
  
IF NOT SMTP.TrySend THEN BEGIN  
    NoOfErrors := NoOfErrors + 1;  
    EXIT(FALSE);  
  
END ELSE  
    EXIT(TRUE);
```

6. Add a function that initializes the send process. For each registration line in the registration header passed as a parameter the function sends the email confirmation to the email address of the contact that is specified in the line. If the send process succeeds, the function updates the **Confirmation Date** field in the registration line. At the conclusion, it must display a message that states that the operation succeeded, or show the number of failures with the last error message.

- In **C/AL Globals** window, create a new function and name it **SendConfirmations**.
- Configure the function to receive a parameter of name **SemRegHeader**, type Record, and subtype Seminar Registration Header.
- Define the following local variables for the function.

Name	DataType	Subtype
SemRegLine	Record	Seminar Registration Line
Contact	Record	Contact

- In the function trigger for the **SendConfirmations** function, enter the following code.

```

Initialize;

SemRegLine.SETRANGE("Document No.",SemRegHeader."No.");
IF SemRegLine.FINDSET(TRUE) THEN
  REPEAT
    Contact.GET(SemRegLine."Participant Contact No.");
    Contact.TESTFIELD("E-Mail");
    IF SendMail(
      Contact."E-Mail",
      STRSUBSTNO(
        Text001,
        SemRegHeader."Seminar Name"),
      STRSUBSTNO(
        ConfirmationBody,
        Contact.Name,
        SemRegHeader."Seminar Name",
        SemRegHeader."Starting Date"))
    THEN BEGIN
      SemRegLine."Confirmation Date" := TODAY;
      SemRegLine.MODIFY;
    END;
    UNTIL SemRegLine.NEXT = 0;
    IF NoOfErrors > 0 THEN
      ERROR(Text002,SMTP.GetLastSendMailErrorText);
    MESSAGE(Text003);
  
```

- e. Compile, save, and then close the codeunit.

Task 2: Customize the Pages

High Level Steps

1. Add a function to the **Seminar Registration** page to send confirmations to all seminar participants.
2. Add a function to the **Seminar Registration List** page to send confirmations to all participants of the seminar.

Detailed Steps

1. Add a function to the **Seminar Registration** page to send confirmations to all seminar participants.
 - a. Design page 123456710, **Seminar Registration**.
 - b. Add a global variable named SemMailMgt for the codeunit 123456705, SeminarMailManagement.
 - c. Add an action to the ActionItems container, and configure the following properties.

Property	Value
Caption	Send Registration Co&nfirmations
Image	SendConfirmation
Promoted	Yes
PromotedCategory	Process

- d. In the OnAction trigger for the Send Registration Confirmations action, enter the following code.

```
SemMailMgt.SendConfirmations(Rec);
```

2. Add a function to the **Seminar Registration List** page to send confirmations to all participants of the seminar.
 - a. Design page 123456713, **Seminar Registration List**.
 - b. Repeat the same customizations that you made for the page 123456710, **Seminar Registration**.

Module Review

Module Review and Takeaways

This module demonstrates how to use the built-in capabilities of Microsoft Dynamics NAV 2013 to interface with external applications. It listed the most common interfacing capabilities. This includes COM technologies, Microsoft .NET Interoperability, and data exchange through files.

Microsoft Dynamics NAV 2013 supports automation and OCX controls. However the support is limited to client-side objects and those objects that do not provide any user interface. Server-side COM is not supported, because not all COM components support 64-bit architecture.

A better alternative to COM is Microsoft .NET Interoperability which lets you run .NET assemblies on the server side. You can also use Microsoft .NET Framework to extend the capabilities of Microsoft Dynamics NAV 2013 client for Windows.

Microsoft Dynamics NAV 2013 does not support COM or .NET Interoperability under the web client or Microsoft Dynamics NAV Portal Framework.

Test Your Knowledge

Test your knowledge with the following questions.

1. Which data types allow you to access COM objects from C/AL?

2. COM objects can only be instantiated and accessed on the server side.

True

False

3. What C/AL function is used to create an instance of an automation server class?

C/SIDE Solution Development in Microsoft Dynamics® NAV 2013

4. Microsoft Dynamics NAV 2013 can read and write data in text or binary files.

True

False

5. Can you use the same File variable to access data in multiple files?

Test Your Knowledge Solutions

Module Review and Takeaways

1. Which two data types allow you to access COM objects from C/AL?

MODEL ANSWER:

OCX and Automation.

2. COM objects can only be instantiated and accessed on the server side.

() True

() False

3. What C/AL function is used to create an instance of an automation server class?

MODEL ANSWER:

CREATE

4. Microsoft Dynamics NAV 2013 can read and write data in text or binary files.

() True

() False

5. Can you use the same File variable to access data in multiple files?

MODEL ANSWER:

Yes, but you can only access one file at a time. You must call CLOSE on the first file, before you OPEN or CREATE the second file, and so on.

MODULE 11: WEB SERVICES

Module Overview

Web services are a lightweight, industry-standard way to make application functionality available to many different external systems and users. Web services architecture enables programs that may be written in different languages or even on different platforms, to communicate with one another in a language- and platform-independent manner. Applications access Web services through widespread Web protocols and data formats such as HTTP, XML, SOAP, and OData. This eliminates concern about how each Web service is implemented.

Microsoft Dynamics NAV® 2013 supports the creation and publishing of Microsoft Dynamics NAV 2013 data and functionality as web services.

Objectives

- Describe Microsoft Dynamics NAV 2013 Web services architecture.
- Explain the protocols Microsoft Dynamics NAV 2013 uses for Web services.
- Evaluate the benefits of Web services over other integration options in Microsoft Dynamics NAV.
- Explain how to expose codeunit, page, and query objects as Web services.
- Consume Web services from external applications.

Prerequisite Knowledge

Microsoft Dynamics NAV 2013 supports creating and publishing Microsoft Dynamics NAV functionality as web services. You can expose pages, codeunits, or queries as web services, and even improve a page web service that uses an extension codeunit. When you publish Microsoft Dynamics NAV objects as web services, they are immediately available on the network.

Microsoft Dynamics NAV 2013 web services support the following protocols:

- SOAP (Simple Object Access Protocol) – Publishes pages for create, read, update, and delete access, and codeunits for function call access.
- OData (Open Data Protocol) – Publishes pages and queries for read access.

Web Services Architecture

Web services are a built-in feature of Microsoft Dynamics NAV 2013 Server. They are contained in the same executable file, and run as part of the same service process in Microsoft Windows. By default, when you install Microsoft Dynamics NAV 2013, it is configured to run both SOAP and OData web services. By using the Microsoft Dynamics NAV Server Administration Tool, you may turn either of the web services on or off.

 **Additional Reading:** For more information about Microsoft Dynamics NAV Server Administration Tool, refer to the Microsoft Dynamics NAV Server Administration Tool topic in the Microsoft Dynamics NAV 2013 Developer and IT Pro Help.

Transactions and State

When you make a web service call, no server state is preserved between the calls. This means that no variables or single-instance codeunits are remembered and no session is maintained. Therefore, each separate web service call is always a single independent transaction.

Security

Every web service consumer must authenticate with Microsoft Dynamics NAV 2013. Web services uses all authentication types that are available in Microsoft Dynamics NAV 2013. You can also configure the web services to use Secure Sockets Layer (SSL) encryption. SSL is a cryptographic protocol that provides security and data integrity for data communications over a network. By encrypting the Microsoft Dynamics NAV web services that use SSL, you make your data and the network secure and reliable.

Publishing Web Services

Web services automatically publish objects from Microsoft Dynamics NAV 2013 as SOAP and OData web services. You publish the objects as web services on the **Web Services** page in the Microsoft Dynamics NAV 2013 client for Windows.

To publish an object as a web service, follow these steps:

1. Start Microsoft Dynamics NAV 2013 client for Windows.
2. In the **Search** field, enter “Web Services”, and then click the **Web Services** page in the results.
3. Click **New**.

In the **New – Web Services** page, enter appropriate values in the **Object Type**, **Object ID**, **Service Name**, and **Published** fields.

Following definitions of fields that are displayed in the Web Services page.

Field	Remarks
Object Type	Specifies the type of the object to expose as a web service. You can select from Codeunit, Page, and Query options.
Object ID	Specifies the ID of the object to expose. You can enter the ID manually, or select it from the drop-down list.
Service Name	Specifies the name of the web service that is created for the specified object. The service name is visible to consumers of the web service and is used for identifying and distinguishing web services. For naming guidance, see the Web Services Best Practices topic in the Microsoft Dynamics NAV 2013 Developer and IT Pro Help.

Field	Remarks
Published	Specifies whether the web service is available to serve the requests. The service is available immediately when you select this check box.

When you publish a web service for an object, depending on the object type, it is available as either of the following:

- SOAP for codeunits and pages
- OData for pages and queries

SOAP Web Services

SOAP web services enable full flexibility for building operation-centric services. They provide industry standard interoperability. You can use SOAP to interact with page or codeunit web services in Microsoft Dynamics NAV 2013. SOAP Web services use the Web Service Description Language (WSDL) to describe their data structure and methods.

Web service users can discover published web services by pointing a browser or a tool, such as the Web Services Discovery Tool, at the computer that is running Microsoft Dynamics NAV Server and retrieve a list of available services. For SOAP web services, you typically enter a URI in a browser to view the discovery document, which lists published web services, or to view the WSDL document for a particular web service.

To display all published SOAP web services that are available at a Microsoft Dynamics NAV Server instance, use a URI of the following type.

```
http://<Server>:<Port>/<ServerInstance>/WS/<CompanyName>/services
```



Note: Specifying the *CompanyName* parameter here is optional.

For example, to display all published SOAP web services that are exposed by the CRONUS International Ltd. demo database, point the browser to the following URI.

```
http://localhost:7047/DynamicsNAV70/WS/services
```

To access the WSDL definition of a particular web service, you use the following syntax.

```
http://<Server>:<Port>/<ServerInstance>/WS/<CompanyName>/<ObjectType>/<ServiceName>
```

Module 11: Web Services

The following table explains the components of the SOAP web service URI.

Component	Remarks
Server	The network address of the Microsoft Dynamics NAV 2013 Server. For example, localhost, nav.cronus.com, and 192.168.210.6 are all examples of valid network addresses.
Port	The port on which the SOAP web services are running. The default port is 7047. However, you can configure a different value.
ServerInstance	The name of the Microsoft Dynamics NAV 2013 Server instance. You may run multiple instances of Microsoft Dynamics NAV 2013 on the same server. The default value is DynamicsNAV70. However, you can configure a different value.
CompanyName	The name of the company that you want to access. This part of the URI is case-sensitive. Therefore, you must make sure that you enter it exactly as the company is named in Microsoft Dynamics NAV 2013. Otherwise, the server is unable to locate the web service. When you access the WSDL document, the CompanyName part is optional. When it is consuming the web service, it is mandatory.
ObjectType	The type of the object. Possible values are Page and Codeunit.
ServiceName	The name of the service, as configured in the Web Services page in Microsoft Dynamics NAV 2013. The name is arbitrary, but every web service of the same object type must have a unique name.

The following example displays the WSDL description for the Customer service.

```
http://localhost:7047/DynamicsNAV70/WS/Page/Customer
```

System Service

You can use the SystemService service in a SOAP web service application to retrieve a list of companies that are available in a specific database. A company name is typically part of the URI when you consume a Microsoft Dynamics NAV 2013 web service. The system service lets you retrieve names of available companies.

The SystemService service is available at the following URI.

```
http://<Server>:<Port>/<ServerInstance>/WS/SystemService
```

For example, for a default Microsoft Dynamics NAV 2013 demo installation, the SystemService service is available at the following URI.

```
http://localhost:7047/DynamicsNAV70/WS/SystemService
```

Page Web Services

When you expose a page as a SOAP web service, you expose a default set of operations that you can use to manage common operations such as Create, Read, Update, and Delete. Page-based web services offer built-in optimistic concurrency management. Each operation call in a page-based web service is managed as a single transaction.

Codeunit Web Services

When you expose a codeunit as a web service, all functions that are defined in the codeunit are exposed as operations. This lets you run business logic that is contained in C/AL code from external applications. For example, external applications can create new sales orders through a page web service, and also can call the release or post functions.

Following are several limitations of codeunit web services:

- Only global functions are visible as methods. You cannot call local functions directly from web services.
- Each exposed method may only have parameters of a simple type that map directly to XSD data types. You cannot use complex data types, such as record, page, or codeunit. The only exception is the XMLport data type, which is translated to a complex XSD definition. The XMLport data type enables client applications to map complex Microsoft Dynamics NAV 2013 data structures to object hierarchies.
- Single-instance and ordinary codeunits are the same. A single-instance codeunit is instantiated one time for each web service call.

Page Operations with SOAP Web Services

When you publish a page as a SOAP web service, it has a set of default operations that are exposed to consumers of the web service.

These operations match the corresponding actions that a user can perform by interacting with a page in the RoleTailored client. All page and table triggers that run when a user performs the same action from the RoleTailored client run when you call a web service operation.

The following table lists the operations and provides links to reference pages.

Operation	Description
Create	Creates a single record in the underlying table of a page.
CreateMultiple	Creates multiple records of the same type.
Delete	Deletes a single record.
Delete_<part>	Deletes a single record from a subpage on a page. For example, deletes a line in the Lines subpage of the Sales Order page.
GetRecIdFromKey	Converts a key, which uniquely represents a record in the database when it interacts with SOAP web services, into a string that matches the Record ID format that is used internally by Microsoft Dynamics NAV 2013.
IsUpdated	Verifies whether another user has updated a record after it was last read from the database.
Read	Reads a single record.
ReadByRecId	Reads a record by Record ID.
ReadMultiple	Reads a filtered set of records. This method lets you page through records, specify the page size and the key of the last record read.
Update	Updates a single record.
UpdateMultiple	Updates multiple records of the same type.

All basic page operations are atomic. This means that either all relevant records are affected or no records are affected, even if there was a faulty condition in only one record.

 **Note:** Only those methods that match the operations a user can do in the RoleTailored client are actually present on each page web service. For example, if page has the ModifyAllowed property set to **No**, then the Update and UpdateMultiple operations are not available in the web service.

Extension Codeunits

For SOAP services, you can also use extension codeunits to extend the default set of operations that are available on a page. Adding an extension codeunit to a page is useful if you want to perform operations other than the standard Create, Read, Update, and Delete operations. The benefit of adding an extension codeunit to a page is that you can make the web service complete by adding operations that are logical to that service. Those operations can use the same object identification principle as the basic page operation.

When extending a page web service by using an extension codeunit, you must create a new web service record that has the same name as the name of the page web service that you are extending. Do not select the Published check box.

When you configure an extension codeunit for a page web service, add only those functions as operations to the web service that have the first parameter set to the Record data type, and the subtype equal to the source table of the page.

OData Web Services

The OData standard is good for web service applications that require a uniform, flexible, general purpose interface for exposing retrieve operations on a tabular data model to clients. OData supports Representational State Transfer (REST) based data services, which enable resources that are identified by using Uniform Resource Identifiers (URIs). They are defined in an abstract data model (EDM), and published within corporate networks and across the Internet by using simple Hypertext Transfer Protocol (HTTP) messages. OData services are lightweight with functionality that is frequently referenced directly in the URI.

Microsoft .NET Framework 4.0 was improved with WCF Data Services, which implement all the non-NAV specific parts of the OData stack. Visual Studio service references understand OData services and can generate EDM-based proxies. This enables the developer to use LINQ to write the data access logic.

OData is supported in PowerPivot, a data-analysis add-in to Microsoft Excel 2010 that provides improved Business Intelligence capabilities. PowerPivot supports sharing and collaboration on user-generated Business Intelligence solutions in a Microsoft SharePoint Server 2010 environment.

 **Additional Reading:** For more information about PowerPivot, see <http://www.powerpivot.com/>.

In addition to the AtomPub format, the OData implementation in Microsoft Dynamics NAV 2013 also supports the JSON format. This is a somewhat less verbose format that may perform better in low-bandwidth environments.

Whereas SOAP web services expose a WSDL document, OData web services expose an EDMX document that contains metadata for all published web services. The Entity Data Model (EDM) is a specification for defining the data that is used by applications that are built on the Entity Framework. EDMX is an XML-based file format that is the packaging format for the service metadata of a data service.

Obtaining the Service Metadata Document

To obtain the service metadata (EDMX) document for the OData web services that are published by a Microsoft Dynamics NAV 2013 server instance, use the following URI.

http://<Server>:<Port>/<ServerInstance>/OData/\$metadata

The following table explains the components of the OData service metadata URI.

Component	Remarks
Server	The same as that used with SOAP web services.
Port	The port on which the OData web services run. The default port is 7048. However, you can configure a different value.
ServerInstance	The same as that used with SOAP web services.

 **Note:** By default, the Company entity type is always present in the EDMX document, and Microsoft Dynamics NAV 2013 always publishes it.

Obtaining AtomPub Documents and Feeds

When you publish a page or query web service, you expose an OData service that can be accessed from a uniform resource identifier (URI) by using a web browser or any other HTTP client. OData clients can use Atom Publishing Protocol (AtomPub) documents to interact with Microsoft Dynamics NAV data. AtomPub documents and feeds are XML.

To obtain a list of companies that use an AtomPub document, use the following URI.

```
http://localhost:7048/DynamicsNAV70/OData/Company
```

 **Note:** You must modify your Internet Explorer settings to display the actual XML for a feed instead of the feed content that has changed. In Internet Explorer, click Tools > Internet Options. On the Content tab, in the Feeds and Web Slices group, click Settings, and then click to clear the Turn on feed reading view check box. Restart Internet Explorer to enable the new setting.

 **Additional Reading:** To learn more about AtomPub URI syntax and possibilities, refer to the How to: Use OData to Return/Obtain an AtomPub document topic in the Microsoft Dynamics NAV 2013 Developer and IT Pro Help.

Handling UI Interaction

In Microsoft Dynamics NAV 2013, web services are stateless. Each call must either fully succeed or fail. Microsoft Dynamics NAV 2013 cannot interact with a web services user in the same manner that the RoleTailored client can. For example, if the C/AL code in a page opens a confirmation dialog box, a user in the RoleTailored client can respond to that confirmation. However, the web service call fails.

When you publish a web service, you must make sure that the code that you are publishing does not assume the ability to interact with a user through the UI. You can use the **GUIALLOWED** function to suppress the UI. For example, you can use this function to determine whether a codeunit is called from the RoleTailored client or from a web service. You must make sure that you suppress any user interaction, such as CONFIRM, STRMENU, or the running of pages or reports, when a codeunit is called from a web service client.

When you implement a conditional code check in Microsoft Dynamics NAV, you should implement the check only around code that could cause an error. You should not encapsulate the whole business logic.

Variables of the Dialog data type or any of the functions that are listed as dialog functions can cause callback not allowed exceptions when they are called from a web service application. The **MESSAGE** function is the only function in this category that does not cause an exception, and it is suppressed.

The following are other keywords that you should not use:

- FORM.RUN
- FORM.RUNMODAL
- ACTIVATE
- REPORT.RUN
- REPORT.RUNMODAL
- HYPERLINK
- FILE.UPLOAD
- FILE.DOWNLOAD

Also, if you access any client-side Automation or .NET Framework interoperability objects from the C/AL code during a web-services call, a run-time error occurs.

Registration Web Service

CRONUS International Ltd. wants to automate the seminar registration process as much as it can. It plans to deploy a website that lets web users see the list of available seminars and register for scheduled seminars.

Solution Design

The functional requirements document for the CRONUS International Ltd. implementation includes the following requirements:

- External applications must be able to retrieve the list of scheduled seminars. The information that is available to the external applications must include at least the following:
 - Name of the seminar
 - Starting date of the seminar
 - Duration of the seminar
 - Status of the seminar
 - Maximum participants
 - Number of registered participants

- External applications must be unable to delete or modify information about scheduled seminars.
- External applications must be able to register new users for a seminar if the maximum number of participants has not been reached.

Your obvious choice is to use the SOAP web services and expose the necessary functionality.

Retrieving List of Scheduled Seminars

External applications must be able to retrieve the list of scheduled seminars. Reading data is best achieved through a page web service, where you can use the ReadMultiple method to retrieve multiple records from a page. Because the data must only be read, your simplest choice is to expose the **Seminar Registration List** page. This page already includes most of the information that is specified in the requirement. The only information that CRONUS requires, and that the page does not expose, is the number of registered participants. You can solve this by adding a new flow field to the table, and to the page.

You should try to expose only necessary information to external applications. Make sure that no sensitive data is available to anyone who does not absolutely require such data. Because the Seminar Registration List page includes more information than is specified in the requirement, consider creating a completely new page specifically for integrating with web services.

 **Note:** It is a good practice to always create a new page specifically for the web services integration, unless you explicitly want to expose the identical functionality of the page itself. The most important reason to do this is data sensitivity and to avoid exposing more information than needed. Another important reason is to avoid regression issues. If you delete or rename fields, or modify page or field properties, such as Editable, you may easily cause all external applications to stop working. By providing a separate page for web services, you completely avoid these issues.

 **Note:** In the labs in this chapter you will not create a new page, but will reuse the existing **Seminar Registration List** page.

For CRONUS, you decided that the risk of exposing the existing page is low. Because no sensitive data will be made available, you decide not to develop a new page. Instead, you decide to expose the **Seminar Registration List** page as a web service.

Preventing Users from Modifying or Deleting Records

Another functional requirement requires you to prevent external users from modifying or deleting information about scheduled seminars. Because the **Seminar Registration List** page is noneditable, you do not have to take any steps specifically to support this requirement.

Registering through Web Services

When registering students for seminars, seminar managers create new Seminar Registration Line records through the **Seminar Registration** page. Therefore, your first design choice might be to use the **Seminar Registration** page, and use its **Lines** subpage to create new Seminar Registration Line records. However, this design choice is not a good one, because it violates the previous requirement. If you expose the Seminar Registration page as a web service, you automatically enable external applications to perform any operations that seminar managers can do by using the user interface.

Following are several other alternatives to provide registration functionality:

- Expose the subpage directly. This is not a good choice because you have to add the **Document No.** and **Line No.** fields to the **Seminar Registration Subform** page. In standard Microsoft Dynamics NAV 2013, these fields are never exposed in the document pages because they easily could enable users to cause data-related errors or inconsistencies.
- Provide a separate page specifically for web services use. Although it is a slightly better choice, it opens the door to a poorly written (or malicious) external application. This could cause the same data-related errors or inconsistencies for users as the first option.
- Provide an extension codeunit that performs the task through C/AL logic. We recommend this option as the cleanest one. Therefore, you will provide a new codeunit, and expose this codeunit as an extension codeunit for the web service that publishes the **Seminar Registration List** page.

The codeunit must contain a function that receives the Customer No. and Contact No., and inserts a new Seminar Registration Line record. The function must make sure not to enable the registration of more participants in a seminar than is specified in the **Maximum Participants** field.

Solution Development

To meet the requirements that are defined in the functional requirements document, you must create and customize several objects.

Tables

You must customize the following table.

Table	Changes
123456710 Seminar Registration Header	Add a new flow field that is named Registered Participants, and configure it to show the count of registration lines.

Pages

You must customize the following page.

Page	Changes
123456710 Seminar Registration	Add the Registered Participants field.

Codeunits

You must create the following codeunit.

Codeunit	Remarks
123456710 Seminar Web Services Ext.	Contains a function that registers a participant for a seminar. This codeunit will be used as an extension codeunit for the Seminar Registration List page.

Lab 11.1: Creating a Web Service

Scenario

You are a developer working on a project to implement Microsoft Dynamics NAV 2013 for CRONUS International Ltd. Your goal is to develop the necessary changes to enable external applications to perform select functions of the Seminar Management module. After you develop changes, you must publish the web services and verify that they were correctly exposed.

Exercise 1: Customize the Objects

Exercise Scenario

Customize the tables and pages to support CRONUS International Ltd. requirements.

Task 1: Customize the Table

High Level Steps

1. Add a flow field to the table 123456710, **Seminar Registration Header** that shows the count of corresponding Seminar Registration Line records.

Detailed Steps

1. Add a flow field to the table 123456710, **Seminar Registration Header** that shows the count of corresponding Seminar Registration Line records.
 - a. Design table 123456710, **Seminar Registration Header**.
 - b. Add the field 61, **Registered Participants** of type Integer.
 - c. Set the Editable property to **No**.
 - d. Set the FieldClass property to FlowField.
 - e. Set the CalcFormula property so that it applies the Count method to the **Seminar Registration Line** table, and filters the table to only include those records with the same Document No. as the **No.** field of the header.

The **CalcFormula** field should contain the following expression.

```
Count("Seminar Registration Line" WHERE (Document No.=FIELD(No.)))
```

- f. Compile, save, and then close the table.

Task 2: Customize the Page

High Level Steps

1. Add the **Registered Participants** field to the **Seminar Registration List** page.

Detailed Steps

1. Add the **Registered Participants** field to the **Seminar Registration List** page.
 - a. Design the page 123456713, **Seminar Registration List**.
 - b. Under the repeater control, add a new field control for the **Registered Participants** field.
 - c. Compile, save, and then close the page.

Exercise 2: Configure and Test the Web Service

Exercise Scenario

After you customize the table and the page, you are now ready to configure the web service for the **Seminar Registration List** page.

Task 1: Publishing Seminar Registration List

High Level Steps

1. Publish page 123456710, **Seminar Registration** as a web service of name ScheduledSeminar.

Detailed Steps

1. Publish page 123456710, **Seminar Registration** as a web service of name ScheduledSeminar.
 - a. Start Microsoft Dynamics NAV 2013 client for Windows.
 - b. In the **Search** field, enter “Web Services”.
 - c. Click the **Web Services** page in the results.
 - d. Click **New**.

In the **New – Web Services** page, enter the following information.

Field	Value
Object Type	Page
Object ID	123456710
Service Name	ScheduledSeminar
Published	(Selected)

- e. Close the **New – Web Services** page.

Task 2: Test the Web Service

High Level Steps

1. Use Internet Explorer to verify that the ScheduledSeminar SOAP web service is available in the DynamicsNAV70 instance on the localhost computer.

Detailed Steps

1. Use Internet Explorer to verify that the ScheduledSeminar SOAP web service is available in the DynamicsNAV70 instance on the localhost computer.
 - a. Start Internet Explorer.
 - b. Browse to the following URL.

```
http://localhost:7047/DynamicsNAV70/WS/CRONUS%20International%20Ltd/Page/ScheduledSeminar
```

- c. Verify that you receive a valid XML response.

Exercise 3: Extend the ScheduledSeminar Web Service with an Extension Codeunit

Exercise Scenario

Your next task is to create a new codeunit with a function that creates a new **Seminar Registration Line** record for the specified Seminar Registration Header, customer, and contact. Then you must expose the codeunit as the extension codeunit for the ScheduledSeminar web service.

Task 1: Create the Codeunit

High Level Steps

1. Create the codeunit 123456710, Seminar Web Services Ext.
2. Add the **RegisterParticipant** function that creates a new **Seminar Registration Line** for the specified Seminar Registration Header, customer and contact.

Detailed Steps

1. Create the codeunit 123456710, Seminar Web Services Ext.
 - a. Create a new codeunit.
 - b. Save the codeunit as 123456710, Seminar Web Service Ext.

2. Add the **RegisterParticipant** function that creates a new **Seminar Registration Line** for the specified Seminar Registration Header, customer and contact.
 - a. Create a new global function and name it "RegisterParticipant".
 - b. Define the following parameters for the function.

Name	DataType	Subtype	Length
SemRegHeader	Record	Seminar Registration Header	
CustNo	Code		20
ContNo	Code		20

- c. In the **RegisterParticipant** function, create the following local variables.

Name	DataType	Subtype
SemRegLine	Record	Seminar Registration Line
LineNo	Integer	

- d. Define a global text constant Text001, with the following value:
"You cannot register for this seminar. Maximum number of participants has already been registered."
- e. In the function trigger for the **RegisterParticipant** function, enter the following C/AL code.

```

LineNo := 10000;

SemRegLine.LOCKTABLE;

SemRegLine.SETRANGE("Document No.",SemRegHeader."No.");

IF SemRegLine.FINDLAST THEN

  LineNo := LineNo + SemRegLine."Line No.';

  SemRegHeader.CALCFIELDS("Registered Participants");

  IF SemRegHeader."Registered Participants" >= SemRegHeader."Maximum Participants" THEN

    ERROR(Text001);

  SemRegLine.RESET;

  SemRegLine.INIT;

  SemRegLine.VALIDATE("Document No.",SemRegHeader."No.");

```

```
SemRegLine."Line No." := LineNo;  
  
SemRegLine.VALIDATE("Bill-to Customer No.",CustNo);  
  
SemRegLine.VALIDATE("Participant Contact No.",ContNo);  
  
SemRegLine."Registration Date" := TODAY;  
  
SemRegLine.INSERT(TRUE);
```

- f. Compile, save, and then close the codeunit.

Task 2: Configure the Extension Codeunit

High Level Steps

1. Configure the Seminar Web Service Ext. codeunit as the extension codeunit for the ScheduledSeminar web service.

Detailed Steps

1. Configure the Seminar Web Service Ext. codeunit as the extension codeunit for the ScheduledSeminar web service.
 - a. In the **Web Services** page in the Microsoft Dynamics NAV 2013 client for Windows, click **New**.
 - b. Enter the following information for the new web service.

Property	Value
Object Type	Codeunit
Object ID	123456710
Service Name	ScheduledSeminar
Published	(not selected)

- c. Close the **New – Web Service** page.

Task 3: Verify the Web Service

High Level Steps

1. Use Internet Explorer to verify that the ScheduledSeminar web service now includes the RegisterParticipant method.

Detailed Steps

1. Use Internet Explorer to verify that the ScheduledSeminar web service now includes the RegisterParticipant method.
 - a. Start Internet Explorer.
 - b. Browse to the following URL.

```
http://localhost:7047/DynamicsNAV70/WS/CRONUS%20International%20Ltd/Page/ScheduledSeminar
```

- c. Verify that the following node exists.

```
<message name="RegisterParticipant">
```

- d. Close Internet Explorer.

Lab 11.2: Create a Windows Forms Application to Test the Web Service

Scenario

You must create a simple application to test the newly published web services. You will create a new C# Windows Forms Application by using Microsoft Visual Studio. Connect it to the ScheduledSeminar web service. Enable it to display the list of scheduled seminars, and let users create new registrations.

Objectives

Now that Web service functions are published (exposed), the next step is to create an application to use (consume) that published service in an external application.

Exercise 1: Create a new Windows Forms Application

Exercise Scenario

Create a new Windows Forms Application and connect it to the ScheduledSeminar web service to display the list of scheduled seminars. Let users register for seminars. At the end of the registration process, display a message that states whether the registration succeeded or failed.

Task 1: Create the New Application

High Level Steps

1. In Microsoft Visual Studio, create a new C# Windows Forms Application.
2. Add a data grid control to the Form1 form.
3. Add the web service reference for the ScheduledSeminar web service.
4. When the application starts, bind the ReadMultiple results of the ScheduledSeminar function to the dataGridView1 grid control.
5. Add two text box controls and a button to the form.
6. Add the code to run the RegisterParticipant method on the ScheduledSeminar web service when a user clicks **Register**.
7. Test the web service.

Detailed Steps

1. In Microsoft Visual Studio, create a new C# Windows Forms Application.
 - a. Start Microsoft Visual Studio.
 - b. Click **File > New > Project**.
 - c. Under Visual C#, select Windows.
 - d. In the list of project types, select Windows Forms Application.
 - e. In the **Name** field, enter "TestSchedSemWS".
 - f. Click **OK**.
2. Add a data grid control to the Form1 form.
 - a. In the **Toolbox**, under the Data group, select the DataGridView control.
 - b. Drag the DataGridView control to the Form1 form.
3. Add the web service reference for the ScheduledSeminar web service.
 - a. In **Solution Explorer**, right-click **References**, and then click Add Service Reference.
 - b. In the **Add Service Reference** window, click **Advanced**.
 - c. In the **Service Reference Settings** window, click **Add Web Reference**.
 - d. In the **Add Web Reference** window, in the **URL** field, enter the following:
<http://localhost:7047/DynamicsNAV70/WS/CRONUS%20International%20Ltd/Page/ScheduledSeminar>
 - e. In the **Web reference name** field, enter "ScheduledSeminar".
 - f. Click **Add Reference**.
4. When the application starts, bind the ReadMultiple results of the ScheduledSeminar function to the dataGridView1 grid control.
 - a. Double-click **Form1** to open the Form1.cs editor.
 - b. At the end of the **using block**, enter the following C# code.

```
using TestSchedSemWS.ScheduledSeminar;
```

- c. In the **Form1_Load** function, enter the following C# code.

```
ScheduledSeminar_Service svc =  
  
    new ScheduledSeminar_Service  
  
    {  
  
        UseDefaultCredentials = true  
  
    };  
  
    dataGridView1.DataSource =  
  
    svc.ReadMultiple(  
  
        new ScheduledSeminar_Filter[] { },  
  
        null,  
  
        0);
```

- d. Press CTRL+SHIFT+S to save all changes.
5. Add two text box controls and a button to the form.
 - a. From the **Toolbox**, drag the TextBox control to the Form1 form two times.
 - b. From the **Toolbox**, drag the Button control to the Form1 form.
 - c. Set the Text property on the button1 control to "Register".
6. Add the code to run the RegisterParticipant method on the ScheduledSeminar web service when a user clicks **Register**.
 - a. Double-click the button1 control.
 - b. In the **button1_Click** function, enter the following C# code.

```
if (dataGridView1.SelectedRows.Count > 0)  
  
{  
  
    ScheduledSeminar_Service svc =  
  
        new ScheduledSeminar_Service  
  
    {  
  
        UseDefaultCredentials = true
```

```
};

try

{

    svc.RegisterParticipant(
        dataGridView1.SelectedRows[0].Cells["Key"].Value.ToString(),
        textBox1.Text,
        textBox2.Text);

    MessageBox.Show("Registration completed.");
}

catch (Exception ex)

{

    MessageBox.Show(
        "Web service call has failed , and you receive the following error
message:\n" +
        ex.Message);

}

}

else

{

    MessageBox.Show("Please, select the scheduled seminar first.");
}
```

- c. Press CTRL+SHIFT+S to save.

Module 11: Web Services

7. Test the web service.
 - a. Press F5 to run the application.
 - b. Verify that the application displays the list of available Seminar Registrations.
 - c. Enter "10000" and "CT100140" into the two text fields.
 - d. Click **Register**.
 - e. In the Microsoft Dynamics NAV 2013 client for Windows, verify that the participant was registered for the seminar.



Note: These steps only work if you already created seminar registrations. If the list is empty, then you must use the Seminar Registration page to create a seminar registration. Follow the steps in Appendix to create test records.

Module Review

Module Review and Takeaways

Web services are an industry-standard concept that enables applications to interact regardless of their platform, programming language, or technology. Microsoft Dynamics NAV 2013 supports SOAP and OData web services, and lets you publish pages, codeunits, and queries for consumption from SOAP or OData clients.

When you expose a page as a web service, you can access it through SOAP to create, read, update, and delete records in the page. This guarantees that all business logic that is contained in page and table triggers execute in the same manner as if a user performed the operation in the RoleTailored client.

You can expose codeunits as web services and call their global methods. You can also use codeunits to extend the functionality of page web services by using new methods, such as registering participants for scheduled seminars.

Finally, you can use pages and queries as sources for the OData web services, and then use the OData web services from an OData client. One of the best available options for OData web services consumption is the PowerPivot add-on for Microsoft Office Excel.

Test Your Knowledge

Test your knowledge with the following questions.

1. Which two types of web services can you publish from Microsoft Dynamics NAV 2013?

2. Which types of objects can you publish as OData web services?

Module 11: Web Services

3. What happens when the code execution in web services encounters a CONFIRM function?

4. Which of the following functions is not a valid operation for a page SOAP web service?

- () CreateMultiple
- () DeleteMultiple
- () UpdateMultiple
- () ReadMultiple

5. What are extension codeunits?

6. How do you properly expose an extension codeunit?

- () Create a Web Service entry for a codeunit. Set Object ID field to the ID of the page you want to extend. Set Service Name field to the name of the codeunit. Leave the Published field unselected.
- () Create a Web Service entry for a codeunit. Set Object ID field to the ID of the codeunit. Set Service Name field to the name of the page web service you want to extend. Leave the Published field unselected.
- () Create a Web Service entry for a codeunit. Set Object ID field to the ID of the page you want to extend. Set Service Name field to the name of the codeunit. Select the Published field.
- () Create a Web Service entry for a codeunit. Set Object ID field to the ID of the codeunit. Set Service Name field to the name of the page web service you want to extend. Select the Published field.

Test Your Knowledge Solutions

Module Review and Takeaways

1. Which two types of web services can you publish from Microsoft Dynamics NAV 2013?

MODEL ANSWER:

SOAP and OData

2. Which types of objects can you publish as OData web services?

MODEL ANSWER:

Pages and Queries

3. What happens when the code execution in web services encounters a CONFIRM function?

MODEL ANSWER:

A run time error occurs.

4. Which of the following functions is not a valid operation for a page SOAP web service?

() CreateMultiple

() DeleteMultiple

() UpdateMultiple

() ReadMultiple

5. What are extension codeunits?

MODEL ANSWER:

Extension codeunits allow adding operations to page SOAP web services.

Module 11: Web Services

6. How do you properly expose an extension codeunit?
 - () Create a Web Service entry for a codeunit. Set Object ID field to the ID of the page you want to extend. Set Service Name field to the name of the codeunit. Leave the Published field unselected.
 - (✓) Create a Web Service entry for a codeunit. Set Object ID field to the ID of the codeunit. Set Service Name field to the name of the page web service you want to extend. Leave the Published field unselected.
 - () Create a Web Service entry for a codeunit. Set Object ID field to the ID of the page you want to extend. Set Service Name field to the name of the codeunit. Select the Published field.
 - () Create a Web Service entry for a codeunit. Set Object ID field to the ID of the codeunit. Set Service Name field to the name of the page web service you want to extend. Select the Published field.

MODULE 12: TESTING AND DEBUGGING

Module Overview

Software does not always behave as developers expect, and various types of errors and bugs frequently occur during development and production of every application. Microsoft Dynamics NAV 2013 includes a comprehensive set of tools and features that you can use to guarantee the highest quality of the customizations that you ship.

You can use the testing features to develop fully automated unit tests. These tests guarantee that your code always runs as designed, and that any bug or error that is introduced by a later change is detected immediately by the developer, instead of the end-users.

When there are bugs in the code, you can use the **Debugger** to step through code execution line by line and inspect the values of variables, parameters, or text constants.

Objectives

- Demonstrate the unit testing features of Microsoft Dynamics NAV 2013.
- Explain the test codeunits, test functions, and handler functions.
- Describe how to automate user interface testing.
- Explain the functionality and purpose of test runner codeunits.
- Develop a unit testing framework for the Seminar Management solution.
- Describe the Debugger functionality and features.
- Demonstrate the debugging process.

Prerequisite Knowledge

Test-driven Development Fundamentals

Test-driven development (TDD) is an advanced technique that uses automated unit tests to drive the design of software and force decoupling of dependencies. The technique results in a comprehensive suite of unit tests that you run at any time to provide feedback that the software is still working. Developers who use the agile development methodology favor this technique.

The motto of test-driven development is "Red, Green, Refactor."

- Red: Create a test and make it fail.
- Green: Make the test pass by any means necessary.
- Refactor: Change the code to remove duplications in your project and to improve the design while making sure that all tests still pass.

These steps explain the example TDD process:

1. Understand the requirements of the story, work item, or feature that you are working on.
2. **Red:** Create a test and make it fail.
 - a. Imagine how the new code should be called and write the test as if the code already existed.
 - b. Create the new production code stub. Write just enough code so that it compiles.
 - c. Run the test. It should fail. This is a calibration measure to make sure that your test is calling the correct code and that the code is not working by accident. This is a meaningful failure, and you expect it to fail.
3. **Green:** Make the test pass by any means necessary.
 - a. Write the production code to make the test pass. Keep it simple.
 - b. Some developers advocate hard-coding of the expected return value first to verify that the test correctly detects success. This varies from practitioner to practitioner.
 - c. If you have written the code so that the test passes as intended, you are finished. You do not have to write more code. If new functionality is still needed, then another test is needed. Make this one test pass, and then continue.
 - d. When the test passes, you might want to run all tests to this point to guarantee that everything else is still working.

Module 12: Testing and Debugging

4. **Refactor:** Change the code to remove duplication in your project and to improve the design while ensuring that all tests still pass.
 - a. Remove duplication that is caused by the addition of the new functionality.
 - b. Make design changes to improve the overall solution.
 - c. After each refactoring, rerun all the tests to make sure that they all still pass.
5. Repeat the cycle. Each cycle should be very short, and a typical hour should contain many Red/Green/Refactor cycles.

Characteristics of a Good Unit Test

A good unit test has the following characteristics:

- Runs fast. If the tests are slow, they will not be run frequently.
- Is very limited in scope. If the test fails, the source of the problem should be obvious. It is important to only test one thing in a single test.
- Runs and passes in isolation. If the tests require special environmental setup or fail unexpectedly, then they are not good unit tests. Change them for simplicity and reliability. Tests should run and pass on any computer. The "it works on my box" excuse does not work.
- Clearly reveals its intention. Another developer should be able to view the test and understand what is expected of the production code.

Benefits of Test-Driven Development

Even though the Test-Driven Development approach seems to add overhead and increase the total amount of work that is needed to complete a task, it has the following benefits:

- The suite of unit tests provides constant feedback that each component is still working.
- The unit tests act as documentation that cannot go out-of-date, unlike separate documentation that frequently is outdated.
- When the test passes and the production code is refactored to remove duplication, it is clear that the code is finished. The developer can move on to a new test.
- Test-driven development forces critical analysis and design because you cannot create production code without truly understanding the result that you want and how to test it.

- The software is better designed, that is, loosely coupled and easily maintainable. The developer is free to make design decisions and refactor at any time with the confidence that the software is still working. This confidence is gained by running the tests. The need for a design pattern may emerge, and the code can be changed at that time.
- The test suite acts as a regression safety net on bugs: If a bug is found, the developer should create a test to reveal the bug and then modify the production code so that the bug is removed and all the other tests still pass. On each successive test run, all previous bug fixes are verified.
- Debugging time is reduced.



Note: *Test-Driven Development practices may be enabled to varying extents in different development environments. Practices that work for some development tools may not work in others. Microsoft Dynamics NAV 2013 enables you to write automated unit tests that you can run easily and frequently. Therefore it enables and promotes the most important TDD practices.*

Test Features

Microsoft Dynamics NAV 2013 includes the following features to help you test your application:

- Test codeunits
- Test runner codeunits
- Test pages
- UI handlers
- ASSERTERROR statement

Test Codeunits

Test codeunits are a special type of codeunit that enable you to write and run code that automatically tests application functionality. Test codeunits can contain the following types of functions:

- Test functions
- Handler functions
- Normal functions

Module 12: Testing and Debugging

When a test codeunit runs, it does the following:

- Runs the OnRun trigger.
- Runs each test function in the test codeunit.
- Records the result in a log.
- Displays the resulting log on message window.

The result of a test function is either SUCCESS or FAILURE. If any error is raised by either the code that is tested or the test code itself, then the result is FAILURE, and the error text is included in the results log. The codeunit continues to run the remaining test functions in the codeunit whether a function results in SUCCESS or FAILURE.

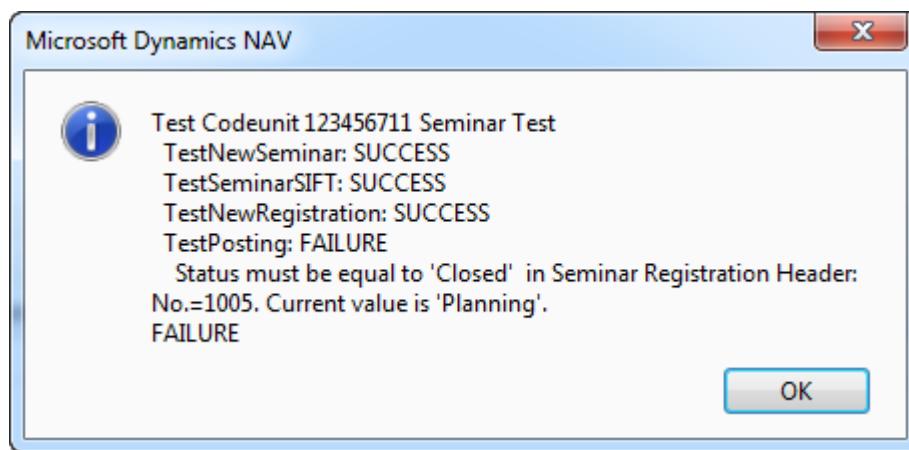


FIGURE 12.1: EXAMPLE

You create a test function by setting the Subtype codeunit property to Test:

1. Design a codeunit.
2. Click **View > Properties**.
3. Set **Subtype** to Test.

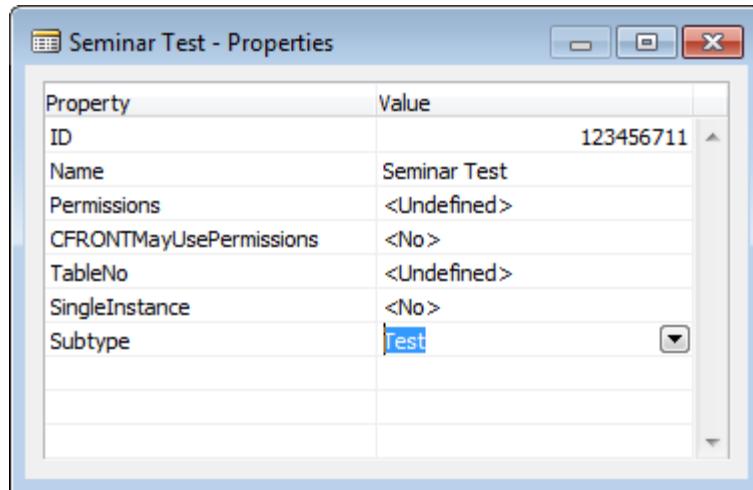


FIGURE 12.2: SUBTYPE PROPERTY FOR TEST CODEUNITS

Test Functions

A *test function* is a special type of a function that you use to perform a test on an area of the application. Test functions do not accept any parameters, and do not return a value. You can only define test functions in a test codeunit.

You can check or set the type of a function in a test codeunit by inspecting its FunctionType property by doing the following:

1. In a test codeunit, click **View > C/AL Globals**.
2. Click the **Functions** tab.
3. Select a function.
4. Click **View > Properties**.
5. Check or set the FunctionType property.

The test functions have their FunctionType property set to Test.

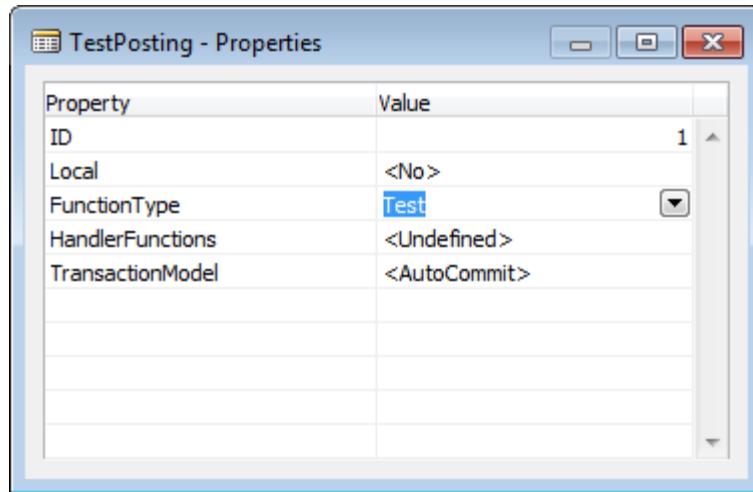


FIGURE 12.3: THE FUNCTIONTYPE PROPERTY FOR TEST FUNCTIONS

 **Note:** You do not have to explicitly set the *FunctionType* property for test functions. This property is automatically set to *Test* for all new functions that you create in a test codeunit.

Transaction Model for Test Functions

Most of the tests in Microsoft Dynamics NAV 2013 are data related, and most of data is very volatile. Writing tests that are completely independent of data can be difficult, and failed tests could easily cause data chaos. Therefore, set the transaction behavior for each test function by setting the *TransactionModel* property.

The *TransactionModel* property has following values.

TransactionModel	Remarks
AutoCommit	A commit is issued at the end of the test function. If an error occurs during the test function, then the transaction is rolled back. If an error occurs and you catch it with an ASERTERROR statement, then the transaction is rolled back. If the code that is being tested calls the COMMIT function before an error occurs, then the transaction is rolled back only to the point at which the COMMIT was called. AutoCommit is the default value.

TransactionModel	Remarks
AutoRollback	The transaction is rolled back after test execution. Calls to the COMMIT function fail with an error during a test that is set to AutoRollback.
None	<p>The None transaction model enables a TestPage to behave exactly like an actual page. The test function does not have an open write transaction. Therefore, it cannot write directly to the database. Each interaction with the database occurs through TestPage triggers. They open their own write transactions. At the end of each transaction, if no errors occurred, then all changes are committed to the database. If an error occurred, then changes are rolled back to the end of the transaction.</p> <p>This differs from the AutoCommit and AutoRollback transaction models. With AutoCommit and AutoRollback, the test function starts a write transaction. Triggers that are invoked by the test code inherit this open transaction instead of running in their own separate transactions. With the AutoCommit and AutoRollback settings, if several page interactions are invoked from test code, then they share the same transaction. With the None setting, each page interaction runs in a separate transaction.</p> <p>You should use the None transaction model for tests that do not write to the database, such as tests that validate calculation formulas or tests that only read from the database.</p>



Note: The **ERROR** function always causes the transaction to be rolled back, regardless of the TransactionModel setting.

 **Note:** If you select the AutoRollback transaction model, then the code that the function tests must not call the **COMMIT** function. If the code calls it, an error occurs and the test function reports FAILURE.

Demonstration: Creating and Running a Test Codeunit

This demonstration shows how you can create a test codeunit, add a test function, and run the test codeunit directly to view the resulting log.

Demonstration Steps

1. Create a new codeunit and save it as 90001, Test Sales.
 - a. In Object Designer, click Codeunit.
 - b. Click **New**.
 - c. Click **File > Save**.
 - d. In the **Save As** dialog box, in the **ID** field, type "90001".
 - e. In the **Name** field, type "Test Sales".
 - f. Make sure that the **Compiled** check box is selected, and then click **OK**.
2. Set the properties to make the new codeunit a test codeunit.
 - a. Click **View > Properties**, or press SHIFT+F4.
 - b. Set Subtype to Test.
 - c. Close the **Properties** window.
3. Create a typical function that creates a sales order and returns a reference to its header.
 - a. Click **View > C/AL Globals**.
 - b. On the **Functions** tab, in the first empty line, type "CreateSalesOrder".
 - c. Select the **CreateSalesOrder** function, and then press SHIFT+F4.
 - d. Set the FunctionType property to Normal.
 - e. Close the **Properties** window.
 - f. Click **Locals**.
 - g. On the **Parameters** tab, enter the following information.

Var	Name	DataType	Subtype	Length
Yes	SalesHeader	Record	Sales Header	
	CustNo	Code		20
	ItemNo	Code		20
	Qty	Decimal		

C/SIDE Solution Development In Microsoft Dynamics® NAV 2013

- h. On the **Variables** tab, enter the following information.

Name	DataType	Subtype
SalesLine	Record	Sales Line

- i. Close the **C/AL Locals** and **C/AL Globals** windows.
j. In the function trigger for the **CreateSalesOrder** function, enter the following C/AL code.

```
SalesHeader.INIT;  
  
SalesHeader."Document Type" := SalesHeader."Document Type"::Order;  
  
SalesHeader."No." := '';  
  
SalesHeader.INSERT(TRUE);  
  
SalesHeader.VALIDATE("Sell-to Customer No.",CustNo);  
  
SalesHeader.MODIFY(TRUE);  
  
SalesLine.INIT;  
  
SalesLine."Document Type" := SalesHeader."Document Type";  
  
SalesLine."Document No." := SalesHeader."No.";  
  
SalesLine."Line No." := 10000;  
  
SalesLine.INSERT(TRUE);  
  
SalesLine.VALIDATE(Type,SalesLine.Type::Item);  
  
SalesLine.VALIDATE("No.",ItemNo);  
  
SalesLine.VALIDATE(Quantity,Qty);  
  
SalesLine.MODIFY(TRUE);
```

4. Add a new function to test creating a sales order, and name it **TestReleaseSalesOrder**. Make sure that it automatically rolls back any data changes after it finishes.
- Click **View > C/AL Globals**.
 - On the **Functions** tab, in the first empty row, type "TestReleaseSalesOrder".
 - Select the **TestReleaseSalesOrder** function, and then press SHIFT+F4.
 - Set the TransactionModel property to AutoRollback.

Module 12: Testing and Debugging

- e. Close the **Properties** window.
- f. Click **Locals**.
- g. On the **Variables** tab, create the following local variable.

Name	DataType	Subtype
SalesHeader	Record	Sales Header

- h. Close the **C/AL Locals** and **C/AL Globals** windows.
- i. In the TestReleaseSalesOrder function trigger, enter the following C/AL code.

```
CreateSalesOrder(SalesHeader,'10000','1000',10);  
  
SalesHeader.TESTFIELD(Status,SalesHeader.Status::Open);  
  
CODEUNIT.RUN(CODEUNIT::"Release Sales Document",SalesHeader);  
  
SalesHeader.TESTFIELD(Status,SalesHeader.Status::Released);
```

5. Save the codeunit, run it, and then view the results.
 - a. Click **File > Save**. Click OK in the **Save** dialog box.
 - b. Close the codeunit.
 - c. In Object Designer, select the codeunit 90001, TestSales.
 - d. Click **Run** to run the test.
 - e. After the test finishes, view the log message.

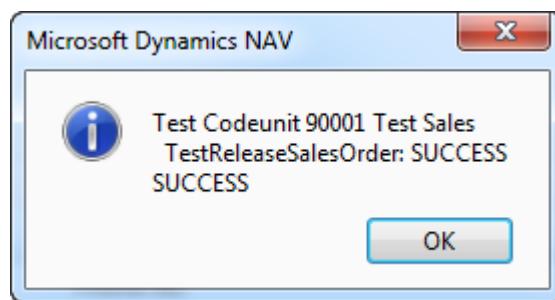


FIGURE 12.4: TESTSALES INITIAL TEST RESULTS

ASERTERROR Statement

When Microsoft Dynamics NAV 2013 runs test codeunits, a run-time error causes the currently running test function to return FAILURE. However, you sometimes have to test whether a specific error occurs. From the perspective of a test function, the error is expected. For example, if you want to test that you cannot change a value of a field on a released sales header, you actually expect an error. If the error occurs, the test must return SUCCESS; on the other hand, if the error does not occur, the test must return FAILURE, because the application behavior is not as you expected.

You use the ASERTERROR statement in test functions to test how your application behaves under failing conditions. The ASERTERROR keyword specifies that an error is expected at run time in the statement that follows the ASERTERROR keyword.

If a simple or compound statement that follows the ASERTERROR keyword causes an error, then the execution successfully continues to the next statement in the test function. You can receive the error text of the statement by using the **GETLASTERRORTEXT** Function.

If a statement that follows the ASERTERROR keyword does not cause an error, then the ASERTERROR statement causes the error. The test function that is running produces a FAILURE result. The error text states: "An error was expected inside an ASERTERROR statement."

 **Note:** When a run-time error occurs in Microsoft Dynamics NAV 2013, the transaction is rolled back. This applies even to the expected error when you use ASERTERROR. However, the execution continues, the transaction is rolled back, and any data that was written to the database before ASERTERROR does not exist in the database after ASERTERROR. Therefore, you should write code that tests only whether the appropriate error has occurred after the ASERTERROR statement.

 **Note:** If you must have two successive ASERTERROR statements in a single test function, then you should write another test function, and move the second ASERTERROR there. Every test function should be as limited in scope as possible, and should always test a single condition or case.

Demonstration: Using ASSERTERROR in Test Functions

This demonstration shows how you can use the ASSERTERROR statement to test failing conditions.

Demonstration Steps

1. Add a new test function to the Test Sales codeunit to verify that you cannot modify header fields for a released order.
 - a. Design codeunit 90001, Test Sales.
 - b. Click **View > C/AL Globals**.
 - c. On the **Functions** tab, select the row for the **TestReleaseSalesOrder** function.

 **Note:** Make sure that you select the whole row, not just the name text for the function.

- d. Press the following keyboard shortcut: CTRL+C, DOWN, F3, CTRL+V.

 **Note:** This creates a copy of the **TestReleaseSalesOrder** function.

- e. Overwrite the name of the copy of the **TestReleaseSalesOrder** function by using the following text:
"TestChangeReleasedSalesOrder".
- f. At the end of the **TestChangeReleasedSalesOrder** function trigger, enter the following code line.

```
ASSERTERROR SalesHeader.VALIDATE("Location Code",'SILVER');
```

This is the complete function trigger code for **TestChangeReleasedSalesOrder**:

TestChangeReleasedSalesOrder Function

```
CreateSalesOrder(SalesHeader,'10000','1000',10);  
  
SalesHeader.TESTFIELD(Status,SalesHeader.Status::Open);  
  
CODEUNIT.RUN(CODEUNIT::"Release Sales Document",SalesHeader);  
  
SalesHeader.TESTFIELD(Status,SalesHeader.Status::Released);  
  
ASSERTERROR SalesHeader.VALIDATE("Location Code",'SILVER');
```

2. Save the codeunit, run it, and then view the results.
 - a. Click **File > Save**. Click **OK** in the **Save** dialog box.
 - b. Close the codeunit.
 - c. In Object Designer, select the codeunit 90001, TestSales.
 - d. Click **Run** to run the test.
 - e. After the test is complete, view the log message.

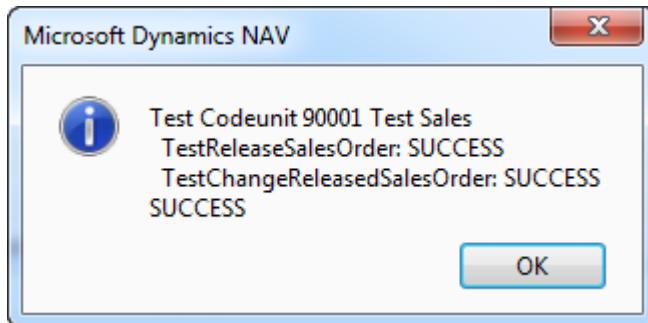


FIGURE 12.5: TESTSALES SECOND RUN RESULTS

 **Note:** *The test fails because instead of calling the Sales-Post codeunit, you should call the Sales-Post (Yes/No). This is the codeunit that runs when users start the posting process.*

Handler Functions

When writing test codeunits, it is a best practice to fully automate tests that do not require any user input. In the event that any user interaction is required, such as making a choice, or confirming a condition, then the code must make those selections on behalf of the user. Moreover, to fully test the functionality of the solution, the test code must test all possible code branches that result from different user choices.

A handler function lets you automate tests by handling instances when user interaction is required by the code that is being tested. In these instances, the test function calls the handler function. This runs instead of the user interface. You specify that a function is a handler function by setting its FunctionType property.

Module 12: Testing and Debugging

Following are several types of handler functions:

- MessageHandler
- ConfirmHandler
- StrMenuHandler
- PageHandler
- ModalPageHandler
- ReportHandler
- RequestPageHandler

Each handler function replaces a specific type of user interaction that may occur in the code. Because there are several types of interactions, different handler functions take different parameters. The following table shows the function signatures for handler functions.

Handler Type	Function Signature
MessageHandler	<Function name>(<Msg> : Text[1024])
ConfirmHandler	<Function name>(<Question> : Text[1024]; VAR <Reply> : Boolean)
StrMenuHandler	<Function name>(<Options : Test[1024]>; VAR <Choice> : Integer; <Instruction> : Text[1024])
PageHandler	<Function name>(VAR <variable name> : Page <page id>) <Function name>(VAR <variable name> : TestPage <testpage id>)
ModalPageHandler	<Function name>(VAR <variable name> : Page <page id>; VAR <Response> : Action) <Function name>(VAR <variable name> : Page <testpage id>)
ReportHandler	<Function name>(VAR <report name> : Report <report id>)
RequestPageHandler	<Function name>(VAR <TestRequestPage> : TestRequestPage)

 **Note:** When you create handler functions, you must define the appropriate parameters that are defined by the signature for the handler type. If the signature of your handler function does not match the expected signature, you cannot compile the codeunit.

HandlerFunctions Property

Defining a handler function does not cause the handler to automatically replace the interaction. You must manually attach a handler function to a test function where the user interaction occurs.

To attach a handler function to a test function, you specify the handler function name in the HandlerFunctions property of the test function as follows:

1. On the **Functions** tab of the **C/AL Globals** window, select the test function.
2. Click **View > Properties**.
3. Type the handler function name in the **HandlerFunctions** property.

 **Note:** If the test function uses more than one handler function, separate the handler function names by a comma.

Every handler function that you enter in the **HandlerFunctions** property must be called at least one time in the test function. If you run a test function that has a handler function listed, and that handler function is not called, then the test fails.

Demonstration: Using Handler Functions to Automate User Interaction

This demonstration shows how you can use the **ConfirmHandler** and **StrMenuHandler** functions to handle different types of user interaction.

Demonstration Steps

1. Add a new test function to the Test Sales codeunit to test changing the **Sell-to Customer No.** field for an existing sales order.
 - a. Design codeunit 90001, Test Sales.
 - b. Click **View > C/AL Globals**.
 - c. Create a new test function, and name it "TestChangeCustomerOnSalesOrder".
 - d. For the **TestChangeCustomerOnSalesOrder** function, define a new local variable for the **Sales Header** table, and call it SalesHeader.
 - e. In the TestChangeCustomerOnSalesOrder function trigger, enter the following code.

```
CreateSalesOrder(SalesHeader,'10000','1000',10);
```

```
SalesHeader.VALIDATE("Sell-to Customer No.",'20000');
```

```
SalesHeader.MODIFY(TRUE);
```

2. Save the codeunit, run it, and then view the results.
 - a. Click **File > Save**. Click **OK** to the **Save** dialog box.
 - b. Close the codeunit.
 - c. In Object Designer, select the codeunit 90001, TestSales.
 - d. Click **Run** to run the test.
 - e. The test codeunit requests confirmation:

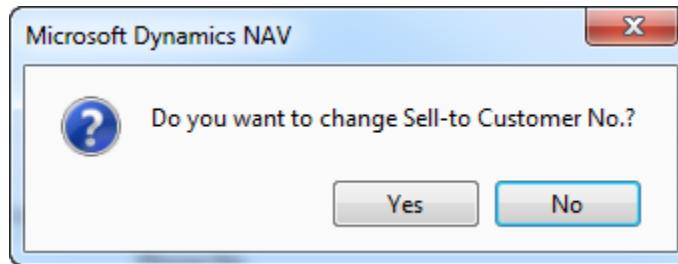


FIGURE 12.6: TESTSALES ASKING FOR CONFIRMATION



Note: Tests should always be automated and never require user interaction.

3. Create a handler function to handle the confirmation dialog box. Attach it to the **TestChangeCustomerOnSalesOrder** function.
 - a. Design the Test Sales codeunit.
 - b. Create a new function and name it "HandleChangeCustomerConfirm".
 - c. Select the **HandleChangeCustomerConfirm** function, and then press SHIFT+F4.
 - d. Set **FunctionType** to **ConfirmHandler**.
 - e. Close the **Properties** window.
 - f. Click **Locals**.
 - g. On the **Parameters** tab, enter the following information.

Var	Name	DataType	Length
	Question	Text	1024
Yes	Reply	Boolean	

- h. Close the **C/AL Locals** window.
- i. Select the **TestChangeCustomerOnSalesOrder** function, and then press SHIFT+F4.
- j. In the Value column for the HandlerFunctions property, type "HandleChangeCustomerConfirm".
- k. Close the **Properties** and **C/AL Globals** windows.

C/SIDE Solution Development In Microsoft Dynamics® NAV 2013

- I. In the HandleChangeCustomerConfirm function trigger, enter the following code:

```
Reply := TRUE;
```

4. Save the codeunit, run it, and then view the results.
 - a. Click **File > Save**. Confirm the **Save** dialog box.
 - b. Close the codeunit.
 - c. In Object Designer, select the codeunit 90001, TestSales.
 - d. Click **Run** to run the test.
 - e. After the test is complete, view the log message.

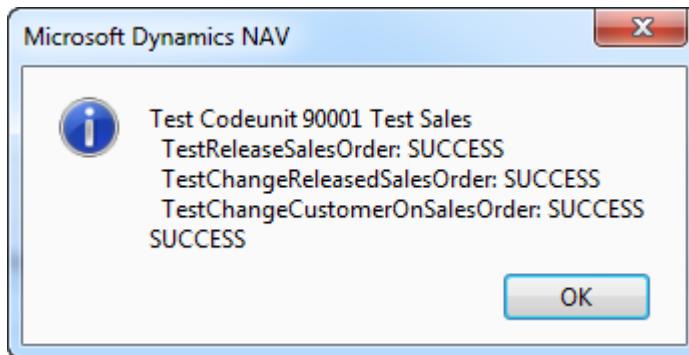


FIGURE 12.7: TEST SALES RESULTS AFTER CONFIRMHANDLER IS IMPLEMENTED

5. Add a handler function to handle the **StrMenu** function for sales order posting.
 - a. Design the **Test Sales** codeunit.
 - b. Create a new function and name it "HandleSalesPostStrMenu".
 - c. Select the **HandleSalesPostStrMenu** function, and then press SHIFT+F4.
 - d. Set **FunctionType** to StrMenuHandler.
 - e. Close the **Properties** window.
 - f. Click **Locals**.
 - g. On the **Parameters** tab, enter the following information.

Var	Name	DataType	Length
No	Options	Text	1024
Yes	Choice	Integer	
No	Instruction	Text	1024

- h. Close the **C/AL Locals** window.
- i. On the **Variables** tab of the **C/AL Globals** window, declare an Option variable and name it PostingType.
- j. Select the PostingType variable and press SHIFT+F4.

Module 12: Testing and Debugging

- k. Set the OptionString property to "Ship,Invoice,All" (make sure not to enter any spaces).
- l. Close the **Properties** and **C/AL Globals** windows.
- m. In the HandleSalesPostStrMenu function trigger, enter the following code.

```
CASE PostingType OF  
    PostingType::Ship: Choice := 1;  
  
    PostingType::Invoice: Choice := 2;  
  
    PostingType::All: Choice := 3;  
  
END;
```

6. Add a function to test posting of an invoice from a sales order. This process must fail. Therefore, make sure that the function tests for the correct failing conditions.
 - a. Create a new function and name it "TestPostInvoiceFromSalesOrder".
 - b. For the **TestPostInvoiceFromSalesOrder** function, define a new local variable for the **Sales Header** table, and call it SalesHeader.
 - c. Select the **TestPostInvoiceFromSalesOrder** function, and press SHIFT+F4.
 - d. In the Value column for the HandlerFunctions property, type "HandleSalesPostStrMenu".
 - e. Close the **Properties** window.
 - f. On the **Text Constants** tab, enter the following information.

Name	ConstValue
Text001	There is nothing to post.
Text002	Expected error:\%1\Actual error:\%2

- g. Close the **C/AL Globals** window.
- h. In the TestPostInvoiceFromSalesOrder function trigger, enter the following code.

```
CreateSalesOrder(SalesHeader,'10000','1000',10);  
  
PostingType := PostingType::Invoice;  
  
ASSERTERROR CODEUNIT.RUN(CODEUNIT::"Sales-Post (Yes/No)",SalesHeader);  
  
IF GETLASTERRORTEXT <> Text001 THEN  
  
    ERROR(Text002,Text001,GETLASTERRORTEXT);
```

 **Note:** When this function runs, the Sales-Post (Yes/No) codeunit runs the **STRMENU** function, which asks users whether they want to ship, invoice, or both. This **STRMENU** is substituted with the code in the **HandleSalesPostStrMenu** function, which returns the option for Invoice. Then the posting fails because the invoice cannot be posted before shipment. This error is trapped by the **ASSERTERROR**. Finally, the code verifies whether the error message has the expected value of "There is nothing to post." and raises an error if it does not.

7. Add a function which posts a shipment and then an invoice from a sales order.
 - a. Create a new function and name it "TestPostSalesOrder".
 - b. For the **TestPostSalesOrder** function, define a new local variable for the **Sales Header** table, and call it SalesHeader.
 - c. Select the **TestPostSalesOrder** function, and press SHIFT+F4.
 - d. In the Value column for the HandlerFunctions property, type "HandleSalesPostStrMenu".
 - e. Close the **Properties** and the **C/AL Globals** windows.
 - f. In the TestPostSalesOrder function trigger, enter the following code.

```
CreateSalesOrder(SalesHeader,'10000','1000',10);  
  
PostingType := PostingType::Ship;  
  
CODEUNIT.RUN(CODEUNIT::"Sales-Post (Yes/No)",SalesHeader);  
  
PostingType := PostingType::Invoice;  
  
CODEUNIT.RUN(CODEUNIT::"Sales-Post (Yes/No)",SalesHeader);
```

8. Save the codeunit, run it, and then view the results.
 - a. Click **File > Save**. Click **OK** the **Save** dialog box.
 - b. Close the codeunit.
 - c. In Object Designer, select the codeunit 90001, TestSales.
 - d. Click **Run** to run the test.
 - e. After the test finishes, view the log message.

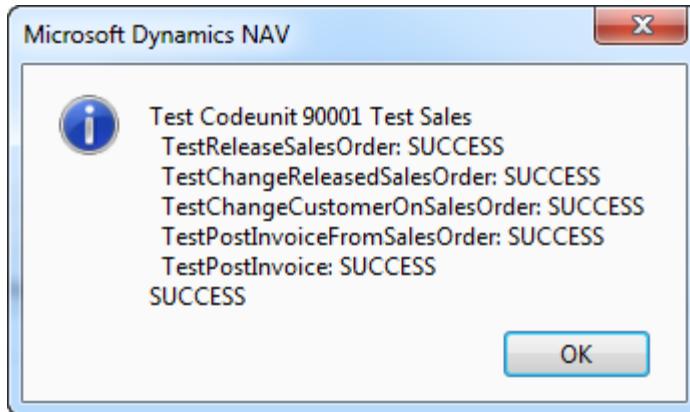


FIGURE 12.8: TEST SALES FINAL RESULTS

Testing Pages

Microsoft Dynamics NAV 2013 enables you to use variables of type TestPage to automate user interaction testing. With test pages, you can do the following:

- View or change the value of a field on a test page.
- View the data on page parts.
- View or change the value of a field on a subpage.
- Filter the data on a test page.
- Perform any actions that are available on the page.
- Browse to different records.

 **Note:** Test functions and code on test pages run on the Microsoft Dynamics NAV 2013 Server instance, even though they simulate client interactions.

Test Pages and the TransactionModel Property

To create meaningful tests, you first must understand how transactions run on pages. In a typical user scenario, a user who is logged on to a client enters data into one field of a page. Then the user enters data in another field on the page. The user also checks the value of a third field. Finally, the user saves and closes the page. Every time that a user enters data into a field, C/AL code may be triggered and a new transaction is automatically started. The trigger code runs within this new transaction. Field data is sent to the server where it is processed and frequently updated in the database. When the C/AL code in the trigger is finished, the transaction is automatically committed to the database and the page is refreshed with updated data.

C/SIDE Solution Development In Microsoft Dynamics® NAV 2013

When you create test functions that exercise pages that interact with the database, you have the following options for simulating user scenarios, and then returning the database to its initial, familiar state:

- Set the TransactionModel property on the test function to AutoRollback. This assumes that the code that you test does not include calls to the **COMMIT** function. Any calls to the **COMMIT** function result in an error. Most business logic does not call the **COMMIT** function, but relies on implicit commits at the end of the outermost C/AL trigger. The test continues as follows:
 - a. The test function starts a transaction.
 - b. The test function initializes data in the database. Database changes are made in the transaction that was started by the test function.
 - c. Fields on the test page are set or updated. Database changes are made in the transaction that was started by the test function.
 - d. The test function reads the values of fields on the test page or reads from the database to validate the test.
 - e. After the test function finishes, the transaction is rolled back and the database is returned to its initial state.
- If the code that you test includes calls to the **COMMIT** function, then set the TransactionModel property on the test function to AutoCommit. The test continues as follows:
 - a. The test function starts a transaction.
 - b. The test function initializes data in the database. Database changes are made in the transaction that was started by the test function.
 - c. Fields on the test page are set or updated. Database changes are made in the transaction that was started by the test function.
 - d. When the **COMMIT** function is called, changes are committed to the database.
 - e. The test function reads the values of fields on the test page, or reads from the database to validate the test.
 - f. After the test function finishes, changes are committed to the database. To return the database to its initial state, you must manually revert the changes by deleting, updating, or inserting records, or you must use the TestIsolation property on the test runner codeunit to roll back changes.

Module 12: Testing and Debugging

- Set the TransactionModel property on the test function to None to simulate the behavior of an actual user. The test function does not start a transaction and cannot write directly to the database. However, a new transaction is started every time that a field on the page is updated and C/AL code is triggered. At the end of each transaction, changes are automatically committed to the database. Use this option if your test does not write to the database. You do not have to initialize data in the database before the test starts. For example, use this option for tests that validate calculation formulas or tests that read from the database. The test continues as follows:
 - If a field on the test page is set or updated, then the test page starts a transaction. At the end of the transaction, changes are committed to the database.
 - The test function runs the test code.
 - After the test finishes, no transactions are rolled back. To return the database to its initial state, you must manually revert the changes by deleting, updating, or inserting records, or you must use the TestIsolation property on the test runner codeunit to roll back changes.

Accessing Fields on Test Pages

You access the fields on a test page by using the dot notation. For example, if you have a test page variable named CustomerCard that represents the **Customer Card page**, then to access the **Name** field on the test page, you write CustomerCard.Name in your code.

To retrieve the value of a field or to write a value in a field, use the Value property. For example, if you have a test page variable named CustomerCard that represents the **Customer Card** page, then you can read the value from a field or assign a value to a field, by writing code that resembles the following.

```
CustNo := CustomerCard."No.".Value;  
CustomerCard.Address.Value := ' 612 South Sunset Drive';
```

Accessing Page Parts and Subpages

You access page parts and subpages on a test page by using the dot notation. For example, to compare the value of the **No.** field on a page to the value of the **No.** field on a FactBox on the page, write the following code.

```
If CustomerCard."No.".Value <> CustomerCard."Sales Hist. Sell-to  
FactBox"."No.".Value THEN  
  
    ERROR('Page part data is not updated. Expected Customer No. is %1, actual  
Customer No. is %2.',CustomerCard."No.".Value, CustomerCard."Sales Hist. Sell-to  
FactBox"."No.".Value);
```

You can use the **Symbol Menu** to view page parts and subpages and to access the functions, properties, fields, and actions on test page parts and subpages.

Filtering Data on Test pages

To filter the data that can be accessed on a test page, you use the FILTER property and filter functions. For example, to filter the customers on the **Customer List** page based on a range of values in the **No.** field, write the following code.

```
CustomerList.FILTER.SETFILTER("No.", '20000..30000');
```

Invoking Actions on Test Pages

Any action that is available on a page is also available on the test page that mimics that page. You access page actions by using the dot notation and the **INVOKE** function. Use the **Symbol Menu** to view the actions that are available on a test page. To view actions that you designed by using **Page Designer**, select the test page variable in the first column of the symbols, and then select **Actions** in the second column.

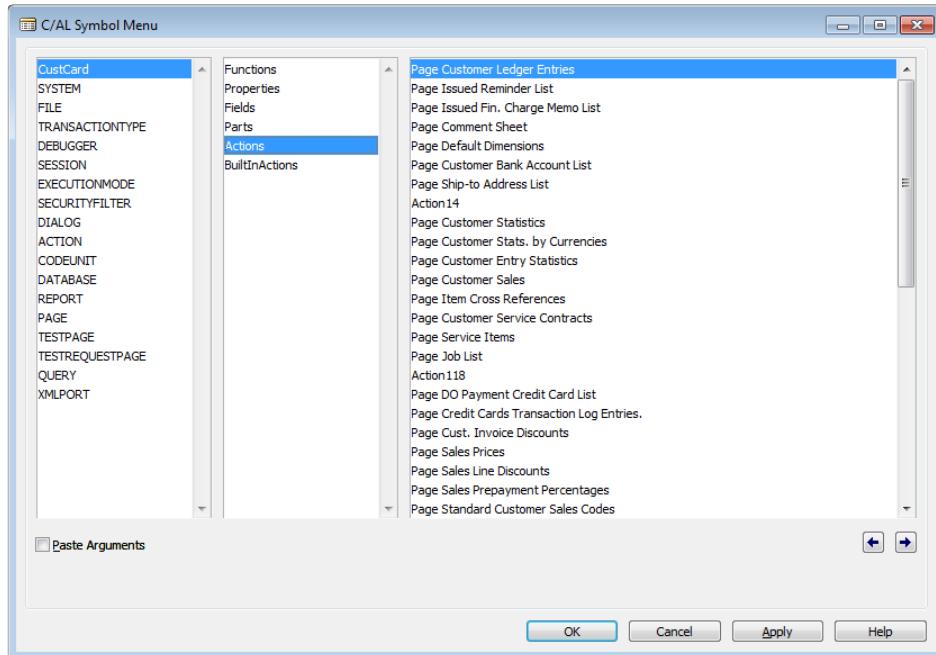


FIGURE 12.9: ACTIONS FOR A TESTPAGE VARIABLE FOR THE CUSTOMER CARD PAGE

For example, to simulate clicking the Sales Prices action on the **Customer Card** page, write the following code.

```
CustCard.OPENVIEW;  
  
CustCard."Page Sales Prices".INVOKE;
```

To view built-in actions, such as **Yes**, **No**, **OK**, or **Cancel**, select the test page variable in the first column of the symbols, and then select **BuiltInActions** in the second column.

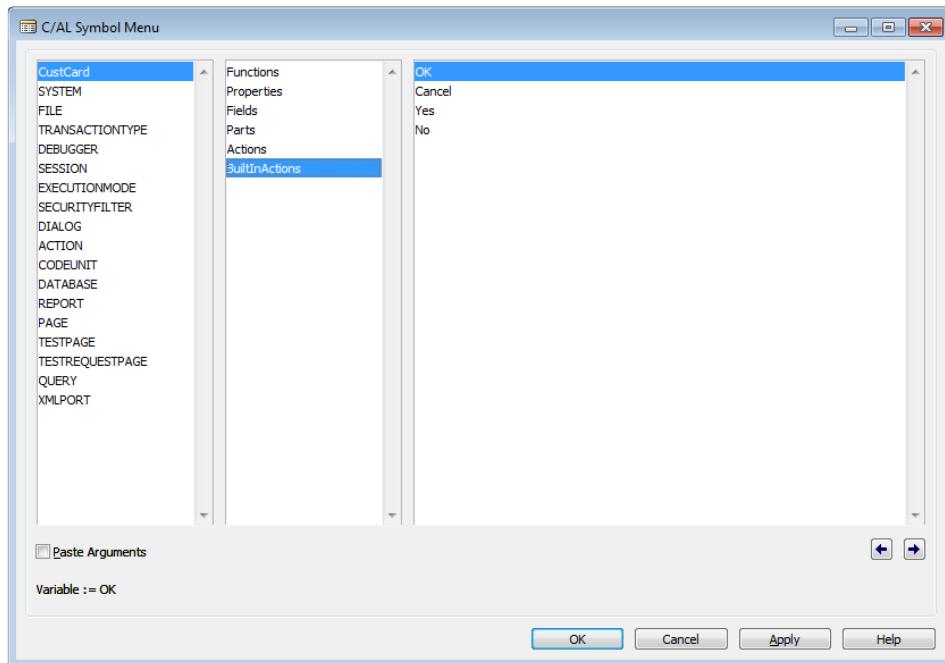


FIGURE 12.10: BUILT-IN ACTIONS FOR A TESTPAGE VARIABLE FOR THE CUSTOMER CARD PAGE

Note: The **Symbol Menu** may include built-in actions that are not available on the page. If you call a built-in action that is not available on the page, then the test fails.

For example, to simulate clicking **OK** after you create a new customer on the **Customer Card** page, you can write the following code.

```
CustomerCard.OPENNEW;  
  
CustomerCard.Name.Value := 'Adventure Works';  
  
CustomerCard.OK.INVOKE;
```

Navigating Among Records

To simulate moving to different items on a list page or moving to different records on a card page, you use one of the following navigation functions:

1. NEXT
2. PREVIOUS
3. FIRST
4. LAST
5. GOTORECORD
6. GOTOKEY

Module 12: Testing and Debugging

7. FINDFIRSTFIELD
8. FINDNEXTFIELD
9. FINDPREVIOUSFIELD

For example, to simulate showing a specific customer on a TestPage variable for the **Customer Card** page, write the following code.

```
CustCard.OPENVIEW;  
  
CustCard.GOTOKEY('30000');
```

Demonstration: Using a TestPage Variable in a Test Codeunit

The following demonstration shows how to automate testing a page by using a TestPage variable.

Demonstration Steps

1. Add a function to create and post a sales order through a TestPage variable.
 - a. Design the Test Sales codeunit.
 - b. Create a new function, name it "TestSalesOrderPagePost", and set its Handlers property to "HandleSalesPostStrMenu".
 - c. Define the following local variables for the **TestSalesOrderPagePost** function.

Name	DataType	Subtype
SalesHeader	Record	Sales Header
SalesOrder	TestPage	Sales Order

- d. In the **TestSalesOrderPagePost** function trigger, write the following code.

```
CreateSalesOrder(SalesHeader,'10000','1000',10);  
  
SalesOrder.OPENVIEW;  
  
SalesOrder.GOTOKEY(SalesHeader."Document Type",SalesHeader."No.");  
  
PostingType := PostingType::Ship;  
  
SalesOrder.Post.INVOKE;  
  
PostingType := PostingType::Invoice;  
  
SalesOrder.Post.INVOKE;
```

2. Save the codeunit, run it, and then view the results.
 - a. Click **File > Save**. Click **OK** in the **Save** dialog box.
 - b. Close the codeunit.
 - c. In Object Designer, select the codeunit 90001, TestSales.
 - d. Click **Run** to run the test.
 - e. After the test finishes, view the log message.

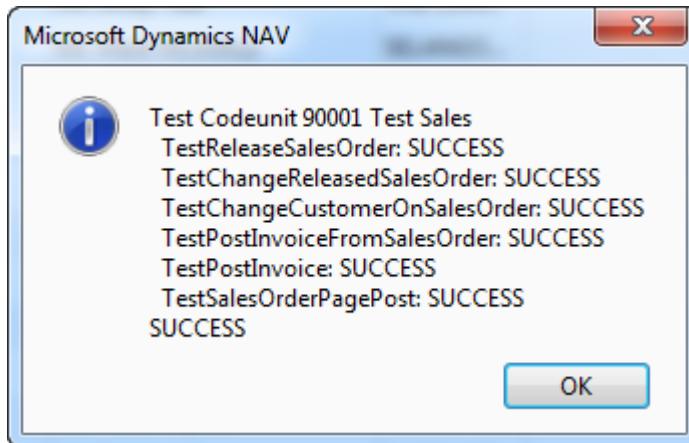


FIGURE 12.11: TEST SALES RESULTS AFTER A TESTPAGE TEST

Test Runner Codeunits

You use test runner codeunits to manage the execution of test codeunits and to integrate with test management or test reporting frameworks. A test runner codeunit manages the execution of test codeunits that are run from its **OnRun** trigger.

1. With test runner codeunits you can do the following:
 2. Run multiple test codeunits.
 3. Intercept each test codeunit or test function before they run.
 4. Control which codeunits or test functions to run or to skip.
 5. Retrieve the test results of a test function and the whole test codeunit after they run.

To create a test runner codeunit, you set the **Subtype** property to **TestRunner**.

Test runner codeunits support two built-in triggers:

6. **OnBeforeTestRun**
7. **OnAfterTestRun**

These two triggers are not automatically created when you create a test runner codeunit. You must manually create them if you want to use them. If you do not need either or both of these triggers, you do not have to create them.

Module 12: Testing and Debugging

When you create these triggers, you must match the following signatures.

Trigger	Signature
OnBeforeTestRun	OnBeforeTestRun(CodeUnitId: Integer;CodeUnitName: Text[30];FunctionName: Text[128]): Boolean
OnAfterTestRun	OnAfterTestRun(CodeUnitId: Integer;CodeUnitName: Text[30];FunctionName: Text[128];Success: Boolean)

OnBeforeTestRun Trigger

The OnBeforeTestRun trigger runs before each test codeunit and test function and enables you to skip running a test codeunit or a test function. The OnBeforeTestRun trigger parameters contain the information about the test codeunit and test function. You control whether to run the current test codeunit or test function through the return value. If you return TRUE, the test codeunit or test function runs. Otherwise, it is skipped.

If the FunctionName parameter is blank, you can control whether to run or skip the whole test codeunit. In this case, returning TRUE runs all test functions, and returning FALSE skips all test functions in the test codeunit.

OnBeforeTestRun always runs in its own database transaction.

You can use the OnBeforeTestRun triggers to perform preprocessing, such as general initialization and logging, or to automate tests by integrating the test runner codeunit with a test management framework.

OnAfterTestRun Trigger

When it is implemented, the OnAfterTestRun trigger is called after each test function runs and after the whole test codeunit runs. The OnAfterTestRun trigger also tells you whether the test codeunit or test function failed or succeeded. If you implement the OnAfterTestRun trigger, then the result logs of the test codeunits that run from the test runner codeunit are not shown.

You can use the OnAfterTestRun trigger to perform post-processing, such as logging, or to automate tests by integrating the test runner codeunit with a test management framework. The OnAfterTestRun trigger is run in its own database transaction.

Demonstration: Creating and Running a Test Runner Codeunit

This demonstration shows how to create a test runner codeunit to control which test codeunits or test functions run during unit testing.

Demonstration Steps

1. Create a new test runner codeunit, and save it as 90002, Test Runner.
 - a. In Object Designer, click **Codeunit**, and then click **New**.
 - b. Press SHIFT+F4.
 - c. Set the Subtype property to TestRunner.
 - d. Close the **Properties** window.
 - e. Click **File > Save**.
 - f. In the **Save As** dialog box, in the **ID** field, type "90002".
 - g. In the Name field, type "Test Runner".
 - h. Make sure that the Compiled check box is selected, and then click **OK**.
2. Add the OnBeforeTestRun and OnAfterTestRun triggers to the Test Runner codeunit.
 - a. Click **View > C/AL Globals**.
 - b. On the **Functions** tab, write "OnBeforeTestRun" and "OnAfterTestRun" on two separate lines.
 - c. Select the **OnBeforeTestRun** function, and then click **Locals**.
 - d. On the **Parameters** tab, enter the following information.

Name	DataType	Length
CodeUnitId	Integer	
CodeUnitName	Text	30
FunctionName	Text	128

- e. On the **Return Value** tab, set Return Type to Boolean.
- f. Close the **C/AL Locals** window.
- g. Select the **OnAfterTestRun** function, and then click **Locals**.
- h. On the **Parameters** tab, enter the following information.

Name	DataType	Length
CodeUnitId	Integer	
CodeUnitName	Text	30
FunctionName	Text	128
Success	Boolean	

Module 12: Testing and Debugging

- i. Close the **C/AL Locals** window.
3. Define global variables to keep track of succeeded, failed, and skipped tests, and a text constant to show the results.
 - a. On the **Variables** tab of the **C/AL Globals** window, enter three Integer variables, and name them Succeeded, Failed, and Skipped.
 - b. On the **Text Constants** tab, create the Text001 constant with the following value: "Testing summary:\Succeeded: %1\Failed: %2\Skipped: %3".
 - c. Close the **C/AL Globals** window.
4. Write code to run the Test Sales codeunit, and to show the results.
 - a. In the OnRun trigger, write the following code.

```
CODEUNIT.RUN(CODEUNIT::"Test Sales");

MESSAGE(Text001,Succeeded,Failed,Skipped);
```

5. Write code to skip any test functions where name includes the word "Invoice", and to run everything else.
 - a. In the **OnBeforeTestRun** function trigger, write the following code.

```
IF (FunctionName <> "") AND (STRPOS(FunctionName,'Invoice') <> 0)

THEN BEGIN

    Skipped := Skipped + 1;

    EXIT(FALSE);

END;

EXIT(TRUE);
```

6. Write code to count the number of succeeded and failed test function runs.
 - a. In the **OnAfterTestRun** function trigger, write the following code.

```
IF FunctionName = "" THEN

    EXIT;

IF Success THEN

    Succeeded := Succeeded + 1
```

```
ELSE
```

```
Failed := Failed + 1;
```

7. Save, close, and run the codeunit, and then view the results.
 - a. Click **File > Save**.
 - b. Confirm the **Save** dialog box.
 - c. Close the codeunit.
 - d. In Object Designer, select the Test Runner codeunit.
 - e. Click **Run**.
 - f. The message box shows the summary of the results.

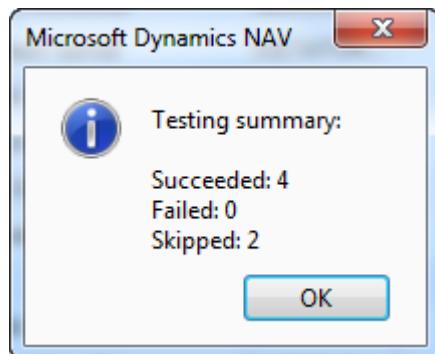


FIGURE 12.12: TEST RUNNER RESULTS



Note: The default log message box that you view when you directly run a test codeunit is suppressed because you implemented the *OnAfterTestRun* trigger.

Testing Seminar Management

An important part of every development process is development of unit test scripts. A comprehensive set of features in Microsoft Dynamics NAV 2013 enables comprehensive testing automation of almost every aspect of the application. This includes all C/AL code and most of the user interface.

Whether you follow a Test-Driven Development approach and follow the red-green-refactor paradigm, or you first develop your application and then write test scripts, your solution should always include a unit test framework that consists of test and test runner codeunits.

Solution Design

Most customers do not explicitly specify the kinds of unit tests that you have to run to guarantee highest code quality and lowest bug rates. However, all customers require a high-quality solution. This maintains the integrity of the standard application and does not disrupt daily use due to frequent bugs or errors.

CRONUS International Ltd. did not specify any testing requirements, but you must still make sure that the solution that you deliver is as bug-free as possible. You also must avoid regression issues in the future, and provide automated means to pinpoint any bugs that result from rework or an upgrade to a future version of Microsoft Dynamics NAV 2013.

Your design decision is to implement a test framework that provides the following functionality:

1. Developers can easily configure which test codeunits to add to or remove from the framework.
2. Developers can easily select which test codeunits and test functions to run or skip.
3. During each test run, only the selected test codeunits and test functions must run.
4. For each test run, the framework must keep a history that contains the following information:
 - a. Date and time of the test run
 - b. ID of the user who ran the test
5. For each test function, the framework must keep a history with the following information:
 - a. Date and time of the test run
 - b. Indicator of test success or failure
 - c. Error message in the event of failure
6. For each test codeunit and test function, the framework must provide statistics with the following information broken down by last run, previous run, and all previous runs:
 - a. Count of successful tests
 - b. Count of failed tests
 - c. Total count of tests
 - d. Success ratio

C/SIDE Solution Development In Microsoft Dynamics® NAV 2013

All developers who are working on the implementation project for CRONUS International Ltd. must provide unit tests for all functionality that they developed. In the future, developers must follow the Test-Driven Development practices to first develop the unit tests and add them to the framework, and then to develop the functionality. For easier maintenance and management of test runs, there must be one test codeunit per functional area.

The test framework will use the test runner codeunit functionality of Microsoft Dynamics NAV 2013. The test runner codeunit will read the list of test codeunits from the setup table and run each test codeunit that is selected for testing.

Through the OnBeforeTestRun trigger, the test runner codeunit will decide whether to run a specific test function based on the information in the setup table.

The OnAfterTestRun trigger will log the success or failure information about test runs into the history tables.

Solution Development

To meet the design goals, you first have to develop the test framework, and then develop the test codeunits for application functionality.

Test Framework

The test framework consists of the following objects.

Object Type	Object ID	Object Name	Remarks
Table	123456751	Seminar Unit Test Setup	<p>Contains the list of test codeunits, test functions, and a check box to specify whether a specific test codeunit or test function is included in test runs.</p> <p>When a user inserts a new row by specifying the Codeunit ID, the list of functions from that test codeunit must automatically be inserted into the Seminar Unit Test Setup table.</p> <p>Users can delete rows for test codeunits only, but not test functions. These are the rows where Function Name is empty.</p> <p>When a user deletes a row, all test function rows for that test codeunit must be deleted automatically.</p>
Table	123456752	Seminar Unit Test Register	<p>Contains the history of test runs. A <i>test run</i> is the event when a user starts all selected tests that are configured in the Seminar Unit Test Setup table. Each line in the Seminar Unit Test Register table corresponds to one test run.</p>
Table	123456753	Seminar Unit Test Entry	<p>Contains the success or failure history of test functions. Each line in the Seminar Unit Test Entry corresponds to one test run of a test function.</p>

C/SIDE Solution Development In Microsoft Dynamics® NAV 2013

Object Type	Object ID	Object Name	Remarks
Page	123456751	Seminar Unit Test Setup	Editable list page over the Seminar Unit Test Setup table that lets users configure which test codeunits and test functions are included in test runs. It also lets users run tests and access the test history and statistics of a specific test codeunit or test function. The only editable fields are Codeunit ID and Run .
Page	123456752	Seminar Unit Test Register	Noneditable list page over the Seminar Unit Test Register table. It lets users access the details of a specific test run.
Page	123456753	Seminar Unit Test Entries	Noneditable list page over the Seminar Unit Test Entry table.
Page	123456754	Seminar Unit Test Statistics	Noneditable card page that follows all design rules of statistics pages.

Module 12: Testing and Debugging

Object Type	Object ID	Object Name	Remarks
Codeunit	123456751	Seminar Unit Test Runner	<p>Test runner codeunit that reads information from the Seminar Unit Test Setup table, runs the selected test codeunits and test functions, and logs the test history into the Seminar Unit Test Register and Seminar Unit Test Entry tables.</p> <p>The codeunit must include a setup mode in which it runs a single test codeunit. Instead of running the tests, the codeunit skips all test functions and inserts a row into the Seminar Unit Test Setup table for each test function that is present in the test codeunit. This test mode is run from the OnInsert trigger of the Seminar Unit Test Setup table.</p>

Specific Test Codeunits

For each functional area of the Seminar Management application area, you must provide one test codeunit. This means that there should be the following test codeunits.

Test Codeunit	Unit Test Examples
123456752 Seminar Master Data Tests	<ul style="list-style-type: none"> • Test the Seminar Setup Card page to make sure that users can look up number series and cannot enter invalid number series. • Test the Seminar Card page to make sure that number series functionality is integrated and works as expected by Microsoft Dynamics NAV 2013 standards. • Test the Seminar Card page to make sure that the VAT Prod. Posting Group field is updated when Gen. Prod. Posting Group field is modified. • Test the Seminar List and Seminar Card pages to make sure that the Ledger Entries action shows the correct ledger entries for the selected seminar.
123456753 Seminar Registration Tests	<ul style="list-style-type: none"> • Test that creating a new seminar registration from the Seminar List creates a new seminar registration for the selected seminar, and that seminar related fields are copied from the Seminar record into the Seminar Registration page. • Test that when a room is selected, the room-related fields are copied from the Resource record into the Seminar Registration page. • Test to specify that when a room allows fewer participants than is specified in the Seminar Registration Header record, the user has to confirm that room. • Test that only canceled seminar registrations can be deleted.

Module 12: Testing and Debugging

Test Codeunit	Unit Test Examples
123456754 Seminar Posting Tests	<ul style="list-style-type: none">• Test that posting succeeds only for closed seminar registrations.• Test that posting succeeds only for seminars that have registered participants.• Test that posting generates a register record with correct information for the user, date, and source code.
123456756 Seminar Reporting Tests	<ul style="list-style-type: none">• Test that clicking Print on the Seminar Registration page runs the Seminar Reg.-Participant List report.
123456757 Seminar Statistics Tests	<ul style="list-style-type: none">• Test that clicking Statistics on the Seminar Card and Seminar List pages shows the Seminar Statistics page that has the correct seminar selected.
123456758 Seminar Dimensions Tests	<ul style="list-style-type: none">• Test that global dimensions that are set to the Seminar record are available in the Default Dimensions page that is invoked from the Dimensions action on the Seminar Card page.• Test that dimensions that are set to Seminar Registration through the Dimensions action update the Shortcut Dimension 1 Code and Shortcut Dimension 2 Code fields.



Note: The list of the unit tests for each test codeunit is not comprehensive, and you can add more tests. In a real-world project, there would be many more tests for each functional area.

Lab 12.1: Create Seminar Management Unit Tests

Scenario

Because of how much work is related to the development of the Seminar Management unit test framework and unit tests, Simon, the development manager on the implementation project for CRONUS International Ltd., has split the work tasks. Simon will develop the test framework, and you will develop the individual unit tests.

Exercise 1: Import the Testing Framework

Exercise Scenario

Simon developed the test framework and gave you the objects. You now import the Seminar Management test framework from the.fob file that was provided.

Task 1: Import the Objects

High Level Steps

1. Import the Lab 12.1 - Starter.fob object file into the Microsoft Dynamics NAV 2013 Development Environment.

Detailed Steps

1. Import the Lab 12.1 - Starter.fob object file into the Microsoft Dynamics NAV 2013 Development Environment.
 - a. In Object Designer, click **File > Import**.
 - b. Browse to the Lab 12.A - Starter.fob file, and then click **Open**.
 - c. In the dialog box stating that there were no conflicts, click **Yes** to complete the import.

Exercise 2: Create the Unit Tests

Exercise Scenario

You develop a test codeunit to contain all unit tests for Seminar Management master data.

Task 1: Master Data Unit Tests

High Level Steps

1. Create the Seminar Master Data Tests codeunit.
2. Create a test function to test that the **AssistEdit(...)** button on the **No.** field on the **Seminar Card** page runs the standard **No. Series** functionality.

Module 12: Testing and Debugging

3. Create a modal page handler function for the **No. Series List** page that simulates clicking **OK**, and then attach this handler to the **TestSeminarCardNoSeries** function.
4. Include the Seminar Master Data Tests codeunit in the test framework configuration.

Detailed Steps

1. Create the Seminar Master Data Tests codeunit.
 - a. In Object Designer, click **Codeunit**, and then click **New**.
 - b. Set the properties on the codeunit to make it a test codeunit.
 - c. Save the codeunit as 123456752, Seminar Master Data Tests.
2. Create a test function to test that the **AssistEdit(...)** button on the **No.** field on the **Seminar Card** page runs the standard **No. Series** functionality.
 - a. Create a test function and name it "TestSeminarCardNoSeries".
 - b. Declare the following local variables for the **TestSeminarCardNoSeries** function.

Name	DataType	Subtype
SeminarCard	TestPage	Seminar Card

- c. In the TestSeminarCardNoSeries function trigger, write the following code.

```
SeminarCard.OPENNEW;
```

```
SeminarCard."No.".ASSISTEDIT;
```

3. Create a modal page handler function for the **No. Series List** page that simulates clicking **OK**, and then attach this handler to the **TestSeminarCardNoSeries** function.
 - a. Create a new function, and name it "NoSeriesListHandler".
 - b. Set the properties on the function to make it a modal page handler.
 - c. Define the following parameter for the **NoSeriesListHandler** function.

Var	Name	DataType	Subtype
Yes	NoSeriesList	TestPage	No. Series List

- d. In the NoSeriesListHandler function trigger, write the following code.

```
NoSeriesList.OK.INVOKE;
```

- e. Set the HandlerFunctions property for the **TestSeminarCardNoSeries** function to "NoSeriesListHandler".
 - f. Save, and then close the codeunit.
4. Include the Seminar Master Data Tests codeunit in the test framework configuration.
 - a. Run page 123456751, **Seminar Unit Test Setup**.
 - b. Click **New**.
 - c. In the **Codeunit ID** field, type "123456752". The **TestSeminarCardNoSeries** function is added to the list automatically.
 - d. Click **Close**.

Task 2: Seminar Registration Unit Tests

High Level Steps

1. Create the Seminar Registration Tests codeunit.
2. Create a test function to test the confirmation of the change of maximum number of participants, when the selected room accommodates less than was defined for the seminar.
3. Create a confirm handler that confirms the question, and then attach it to the **TestSeminarRegistrationRoomMaxParticipants** function.
4. Include the Seminar Registration Tests codeunit in the test framework configuration.

Detailed Steps

1. Create the Seminar Registration Tests codeunit.
 - a. In Object Designer, click **Codeunit**, and then click **New**.
 - b. Set the properties on the codeunit to make it a test codeunit.
 - c. Save the codeunit as 123456753, Seminar Registration Tests.
2. Create a test function to test the confirmation of the change of maximum number of participants, when the selected room accommodates less than was defined for the seminar.
 - a. Create a new test function and name it "TestSeminarRegistrationRoomMaxParticipants".
 - b. Define the following local variables for the **TestSeminarRegistrationRoomMaxParticipants** function.

Name	DataType	Subtype
Seminar	Record	Seminar
Resource	Record	Resource
SeminarRegistration	TestPage	Seminar Registration

Module 12: Testing and Debugging

- c. In the function trigger for the **TestSeminarRegistrationRoomMaxParticipants** function, write the following code.

```
Seminar.FINDFIRST;  
  
SeminarRegistration.OPENNEW;  
  
SeminarRegistration."Seminar No.".SETVALUE(Seminar."No.");  
  
Resource.SETRANGE(Type,Resource.Type::Machine);  
  
Resource.SETFILTER("Maximum Participants",'<>0');  
  
Resource.FINDFIRST;  
  
SeminarRegistration."Maximum Participants".SETVALUE(Resource."Maximum  
Participants" + 1);  
  
SeminarRegistration."Room Resource No.".SETVALUE(Resource."No.");  
  
SeminarRegistration."Maximum Participants".ASERTEQUALS(Resource."Maximum  
Participants");  
  
SeminarRegistration.OK.INVOKE;
```

3. Create a confirm handler that confirms the question, and then attach it to the **TestSeminarRegistrationRoomMaxParticipants** function.
- Create a new function, and name it "ConfirmMaxParticipants".
 - Set the properties for the **ConfirmMaxParticipants** function to make it a confirm handler.
 - Define the following parameters for the **ConfirmMaxParticipants** function.

Var	Name	DataType	Length
No	Question	Text	1024
Yes	Reply	Boolean	

- In the ConfirmMaxParticipants function trigger, write the following code:

```
Reply := TRUE;
```

- Set the HandlerFunctions property of the **TestSeminarRegistrationRoomMaxParticipants** function to "ConfirmMaxParticipants".
- Save and close the codeunit.

4. Include the Seminar Registration Tests codeunit in the test framework configuration.
 - a. Run page 123456751, **Seminar Unit Test Setup**.
 - b. Click **New**.
 - c. In the **Codeunit ID** field, type "123456753".
 - d. Click **Close**.

Task 3: Dimension Integration Unit Tests

High Level Steps

1. Create the Seminar Dimensions Tests codeunit.
2. Create a function to test that global dimensions set through the Dimensions action of the **Seminar Registration** page are correctly copied to the **Shortcut Dimension 1 Code** and **Shortcut Dimension 2 Code** fields.
3. Create a new modal page handler function for the **Edit Dimension Set Entries** page to simulate entering values for two global dimensions for a seminar registration.
4. Include the Seminar Dimensions Tests codeunit in the test framework configuration.

Detailed Steps

1. Create the Seminar Dimensions Tests codeunit.
 - a. In Object Designer, click **Codeunit**, and then click **New**.
 - b. Set the properties on the codeunit to make it a test codeunit.
 - c. Define the following global variables for the function.

Name	DataType	Subtype
GLSetup	Record	General Ledger Setup
DimVal1	Record	Dimension Value
DimVal2	Record	Dimension Value
 - d. Save the codeunit as 123456758, Seminar Dimensions Tests.
2. Create a function to test that global dimensions set through the Dimensions action of the **Seminar Registration** page are correctly copied to the **Shortcut Dimension 1 Code** and **Shortcut Dimension 2 Code** fields.
 - a. Create a new function and name it "TestShortcutDimensionsRegistration".

Module 12: Testing and Debugging

- b. Define the following local variables for the **TestShortcutDimensionsRegistration** function.

Name	DataType	Subtype
SeminarRegistration	TestPage	Seminar Registration
Seminar	Record	Seminar
SemReg	Record	Seminar Registration Header

- c. In the **TestShortcutDimensionsRegistration** function trigger, write the following code.

```
Seminar.FINDFIRST;  
  
SeminarRegistration.OPENNEW;  
  
SeminarRegistration."Seminar No.".SETVALUE(Seminar."No.");  
  
GLSetup.GET;  
  
GLSetup.TESTFIELD("Global Dimension 1 Code");  
  
GLSetup.TESTFIELD("Global Dimension 2 Code");  
  
SeminarRegistration.Action43.INVOKE;  
  
SemReg.GET(SeminarRegistration."No.".VALUE);  
  
SemReg.TESTFIELD("Shortcut Dimension 1 Code",DimVal1.Code);  
  
SemReg.TESTFIELD("Shortcut Dimension 2 Code",DimVal2.Code);
```

 **Note:** Action43 in this code example refers to the Dimension action on the **Seminar Registration** page. Its ID may differ, depending on the exact steps you made during developing the page. If this code example does not compile because Action43 is unknown, then design the **Seminar Registration** page, find the **Dimensions** action, and then note its ID. Then use that ID in this code to refer to that action.

3. Create a new modal page handler function for the **Edit Dimension Set Entries** page to simulate entering values for two global dimensions for a seminar registration.
- Create a new function and name it "EditDimSetHandler".
 - Set properties on the **EditDimSetHandler** function to make it a modal page handler.
 - Define the following parameters for the **EditDimSetHandler** function.

Var	Name	DataType	Subtype
Yes	EditDimSet	TestPage	Edit Dimension Set Entries

- d. In the EditDimSetHandler function trigger, write the following code.

```
DimVal1.SETRANGE("Dimension Code",GLSetup."Global Dimension 1 Code");

DimVal1.FINDFIRST;

EditDimSet.NEW;

EditDimSet."Dimension Code".SETVALUE(GLSetup."Global Dimension 1 Code");

EditDimSet.DimensionValueCode.SETVALUE(DimVal1.Code);

DimVal2.SETRANGE("Dimension Code",GLSetup."Global Dimension 2 Code");

DimVal2.FINDFIRST;

EditDimSet.NEW;

EditDimSet."Dimension Code".SETVALUE(GLSetup."Global Dimension 2 Code");

EditDimSet.DimensionValueCode.SETVALUE(DimVal2.Code);

EditDimSet.OK.INVOKE;
```

- e. Set the HandlerFunctions property for the **TestShortcutDimensionsRegistration** function to "EditDimSetHandler".
- f. Save, and then close the codeunit.
4. Include the Seminar Dimensions Tests codeunit in the test framework configuration.
- Run page 123456751, **Seminar Unit Test Setup**.
 - Click **New**.
 - In the **Codeunit ID** field, type "123456758".
 - Click **Close**.

Exercise 3: Run Unit Tests

Exercise Scenario

You now run the unit tests through the Seminar Management unit test framework, and then check the unit test history to verify that the tests have run successfully.

Task 1: Run the Tests

High Level Steps

1. Use the test framework to run unit tests.

Detailed Steps

1. Use the test framework to run unit tests.
 - a. Run page 123456751, **Seminar Unit Test Setup**.
 - b. Click **Run Tests**. The tests run automatically and the progress is shown in a dialog box.
 - c. After the progress dialog box closes, click **Test Entries** to verify that tests have generated results.
 - d. Close the **Test Entries** page.
 - e. Click **Run Tests** two more times to generate more test results.

Task 2: Check Test Statistics

High Level Steps

1. Show statistics for a test codeunit.

Detailed Steps

1. Show statistics for a test codeunit.
 - a. In the **Seminar Unit Test Setup** page, select the row for the Seminar Master Data Tests codeunit.
 - b. Click **Statistics**.
 - c. Verify that the statistics show the number of succeeded and failed tests over all previous tests.
 - d. Close **Statistics**.

Task 3: Check the Test Register

High Level Steps

1. Show the unit test register.
2. Show test entries for a test run.

Detailed Steps

1. Show the unit test register.
 - a. Run the page 123456752, **Seminar Unit Test Register**.
 - b. Verify that there are three register lines that contain date, time, and user ID for each test run.
2. Show test entries for a test run.
 - a. Select the first row in the **Seminar Unit Test Register** page.
 - b. Click **Test Entries** to show the **Seminar Unit Test Entries** page.
 - c. Verify that the entries shown match the entries specified in the **From Entry No.** and **To Entry No.** fields of the **Seminar Unit Test Register** page.
 - d. Close the **Seminar Unit Test Entries** and **Seminar Unit Test Register** pages.



Debugging

The process of finding and correcting errors is called *debugging*. Microsoft Dynamics NAV 2013 provides an integrated debugger to help you inspect your code to verify that your application runs as expected. The debugger user interface (UI) runs in the Microsoft Dynamics NAV 2013 client for Windows. The debugger services run in the Microsoft Dynamics NAV Server.

Activating the Debugger

When you *activate* the debugger, you start it. When you start the debugger, it can be in one of the following states:

- Attached to a session.
- Waiting to attach to a session.

To start the debugger in the Microsoft Dynamics NAV 2013 Development Environment, in the **Tools** menu, click **Debugger > Debug Session**. This opens the **Session List** window that shows all debuggable sessions that currently run on the same Microsoft Dynamics NAV 2013 server instances on the local machine instance as the debugger . This is the same instance that is targeted when you use the Run action on Application objects in the Object Designer window.

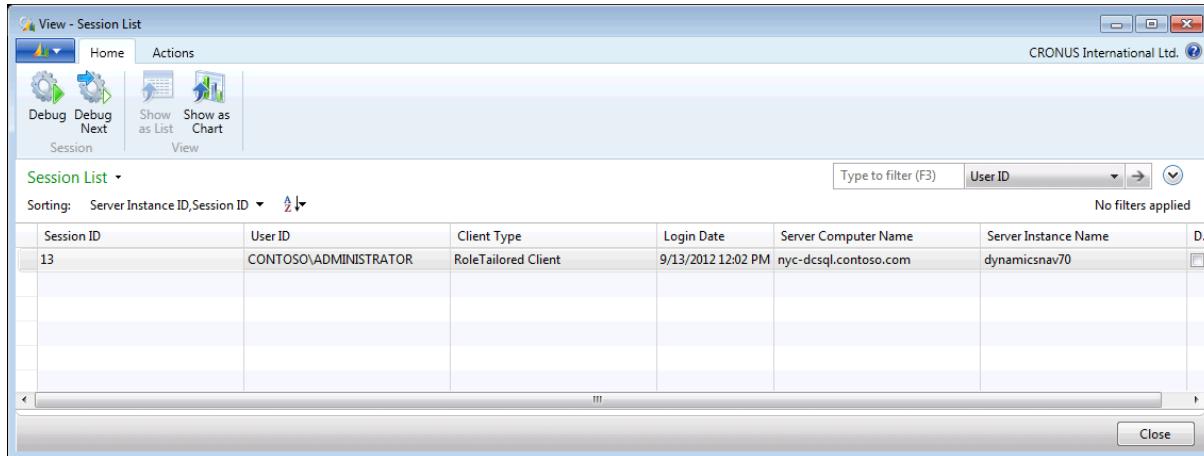


FIGURE 12.13: SESSION LIST

In the **Session List** page, you attach the debugger to a session by doing one of the following:

- Select a specific session, and then click **Debug**.
- Click **Debug Next**, and then start a new session.

 **Note:** Selecting **Debug Next Session** is useful if you want to debug web services. A web service call exists as a session only during the web service call. This typically is not long enough for you to select the specific session in the **Session List** page.

Debugger Page

After you start a debugger, the **Debugger** page opens. You use the **Debugger** page to manage the debug process as follows:

- Step through the code.
- Manage the code execution.
- Manage the breakpoints.
- View the variables in scope of the current line.
- View the last error message.
- Manage watches.
- View the call stack.

C/SIDE Solution Development In Microsoft Dynamics® NAV 2013

The “Debugger” figure shows the **Debugger** page.

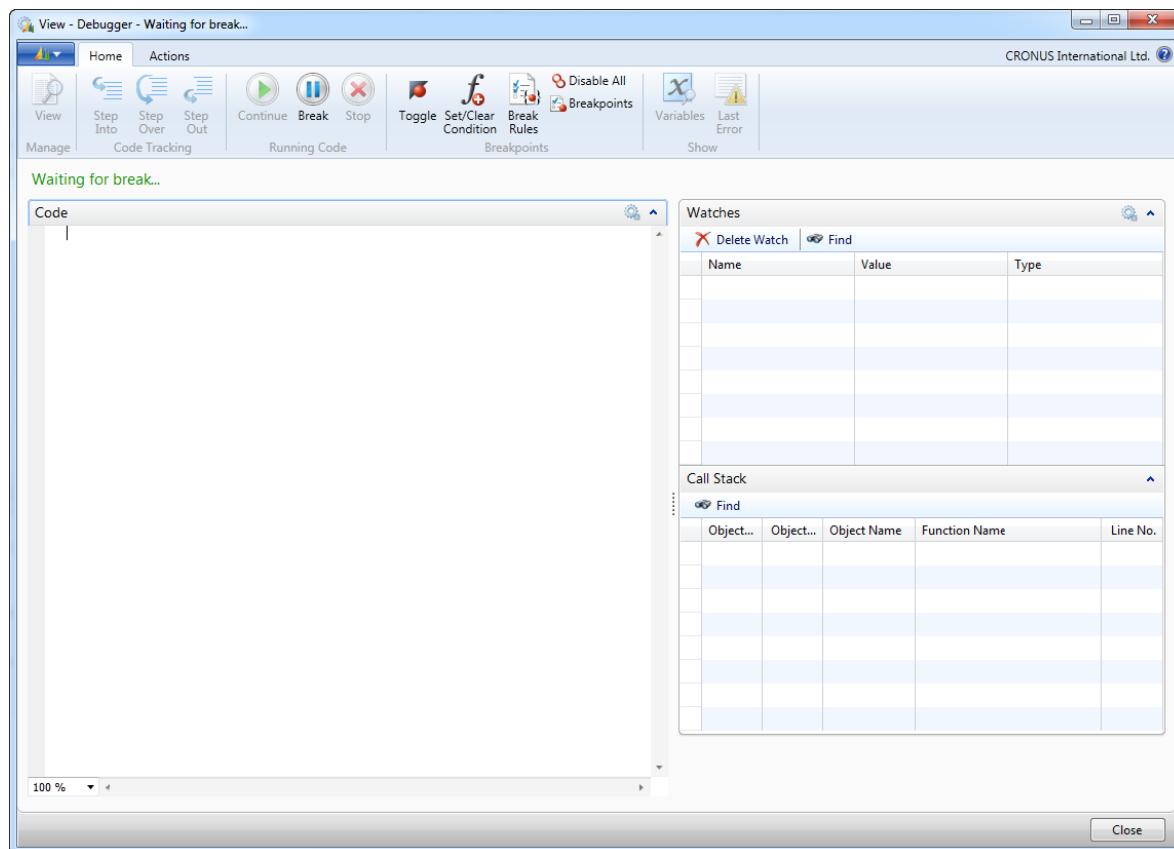


FIGURE 12.14: DEBUGGER

Breakpoints

You can break code execution of the session that you are debugging by doing the following:

- Setting a breakpoint on a line of code.
- Specifying a break on the next statement.
- Specifying a break on errors.
- Specifying a break on record changes.

You can set breakpoints before you start a debugging session or when you are debugging. Breakpoints and break rules are applied immediately in the session to which the debugger is attached.

Break Rules

The debugger usually stops on breakpoints. However, you can enable other break rules that enable the debugger to do the following:

- Stop on error.
- Break on record changes.
- Skip any breaks in Codeunit 1.

To define these additional rules in the **Debugger** page, click **Break Rules**. It opens the **Debugger Break Rules** dialog box.

The “Debugger Break Rules” figure shows the Debugger Break Rules dialog box.

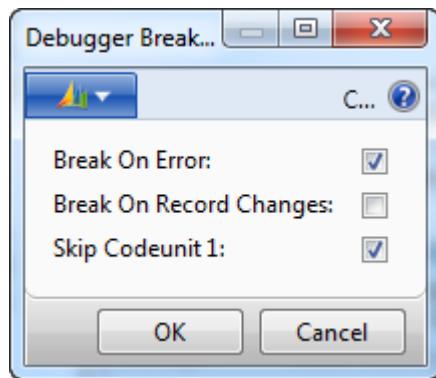


FIGURE 12.15: DEBUGGER BREAK RULES

Breakpoints in Code

If you set a breakpoint on a line of C/AL code, then execution breaks before the first statement on the line executes. If you set a breakpoint on a line of code that does not have a C/AL statement, then the breakpoint is automatically set on the next statement.

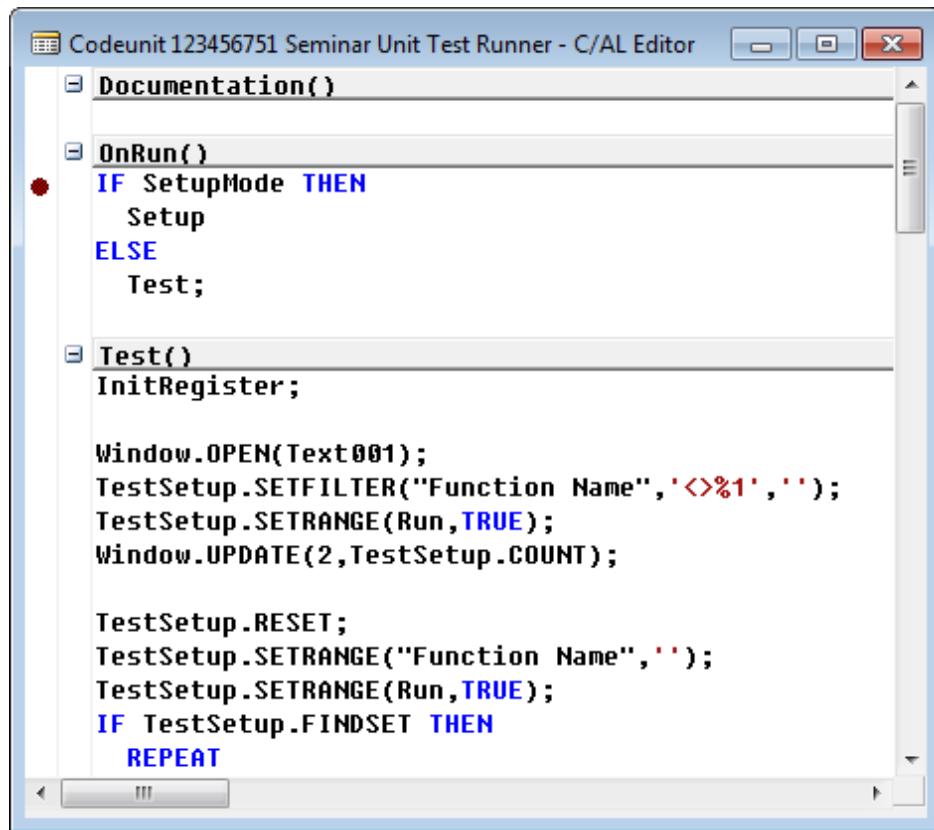
You can set a breakpoint on a code line in the **C/AL Editor** window or the **Debugger** page by positioning the cursor in the line where you want to set the breakpoint, and then pressing F9.

 **Note:** As an alternative, in the **Tools** menu, click **Debugger > Toggle Breakpoint** in the **C/AL Editor** window, or click **Toggle** in the **Debugger** page.

You can enable or disable a breakpoint. The debugger only stops on enabled breakpoints.

C/SIDE Solution Development In Microsoft Dynamics® NAV 2013

The “Enabled Breakpoint” figure shows how the enabled breakpoint looks in the **C/AL Editor** window.



The screenshot shows the Microsoft Dynamics NAV 2013 C/AL Editor window titled "Codeunit 123456751 Seminar Unit Test Runner - C/AL Editor". The code editor displays the following AL code:

```
Codeunit 123456751 Seminar Unit Test Runner - C/AL Editor
Documentation()
OnRun()
    IF SetupMode THEN
        Setup
    ELSE
        Test;

Test()
InitRegister;

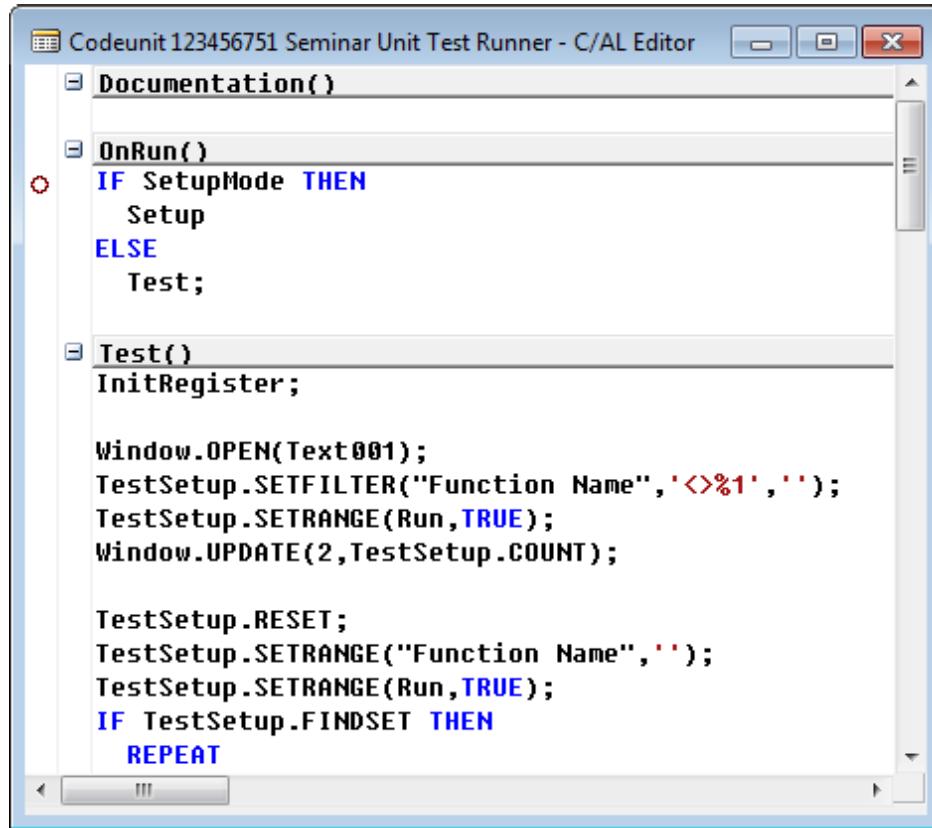
Window.OPEN(Text001);
TestSetup.SETFILTER("Function Name", '<>%1', '');
TestSetup.SETRANGE(Run, TRUE);
Window.UPDATE(2, TestSetup.COUNT);

TestSetup.RESET;
TestSetup.SETRANGE("Function Name", '');
TestSetup.SETRANGE(Run, TRUE);
IF TestSetup.FINDSET THEN
    REPEAT
```

A red dot, indicating an enabled breakpoint, is positioned next to the opening brace of the `OnRun()` block.

FIGURE 12.16: ENABLED BREAKPOINT

The “Disabled Breakpoint” figure shows the disabled breakpoint in the **C/AL Editor** window.



The screenshot shows the C/AL Editor window titled "Codeunit 123456751 Seminar Unit Test Runner - C/AL Editor". The code editor displays the following C/AL code:

```

Codeunit 123456751 Seminar Unit Test Runner - C/AL Editor

Documentation()

OnRun()
IF SetupMode THEN
    Setup
ELSE
    Test;

Test()
InitRegister;

Window.OPEN(Text001);
TestSetup.SETFILTER("Function Name", '<>%1', '');
TestSetup.SETRANGE(Run, TRUE);
Window.UPDATE(2, TestSetup.COUNT);

TestSetup.RESET;
TestSetup.SETRANGE("Function Name", '');
TestSetup.SETRANGE(Run, TRUE);
IF TestSetup.FINDSET THEN
    REPEAT

```

A red circle with a slash is placed next to the first line of the `OnRun()` code, indicating it is a disabled breakpoint.

FIGURE 12.17: DISABLED BREAKPOINT

 **Note:** You disable breakpoints when you do not want the debugger to stop at specific points in the code, but may later want to debug those points. Then instead of searching through the code and creating a new breakpoint, you can enable a disabled breakpoint from the list of all breakpoints.

Breakpoints Overview

You manage all breakpoints from the **Debugger Breakpoint List** page. To access it, in the **Debugger** page, click **Breakpoints**. The **Debugger Breakpoint List** page enables you to do the following:

- Delete a specific breakpoint.
- Delete all breakpoints.
- Enable or disable a specific breakpoint.
- Enable or disable all breakpoints.
- Create a new breakpoint by specifying the object type, ID, and line number manually.

C/SIDE Solution Development In Microsoft Dynamics® NAV 2013

The “Debugger Breakpoint List” image shows the **Debugger Breakpoint List** page.

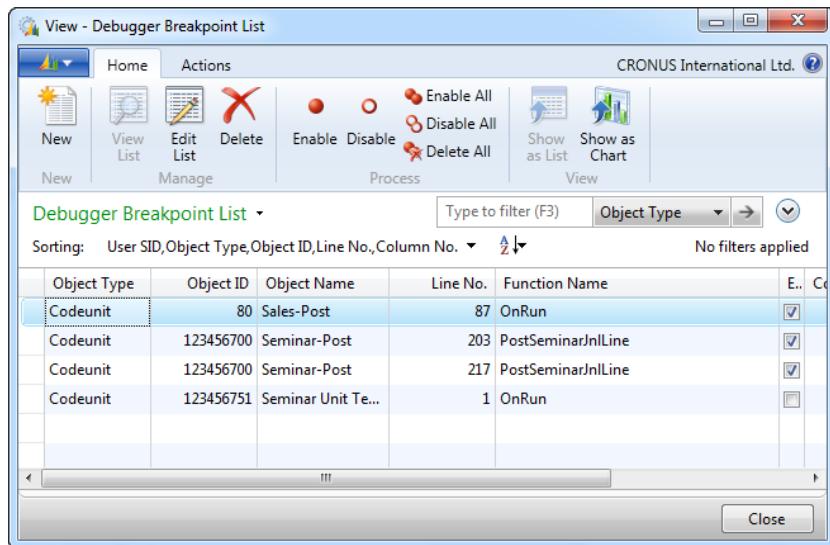


FIGURE 12.18: DEBUGGER BREAKPOINT LIST

Conditional Breakpoints

Most breakpoints break the code execution unconditionally. Sometimes you want to break the execution only if certain conditions are met, but execute the code without breaking. To set a condition on a breakpoint, when the execution stops on the breakpoint, in the **Debugger** page click **Set/Clear Condition**. This shows the **Debugger Breakpoint Condition** dialog box where you can enter a Boolean expression. This may include any of the variables in scope of the breakpoint line. When the code execution reaches the breakpoint, the debugger first evaluates the condition expression. Then, if it evaluates to TRUE, the debugger breaks. If it evaluates to FALSE, it continues execution without breaking.

 **Note:** You can disable a conditional breakpoint. When you do this, the condition remains defined on the breakpoint. It will apply after you enable the breakpoint again.

Module 12: Testing and Debugging

An enabled conditional breakpoint is shown with a white plus (+) sign in the red breakpoint circle. A disabled conditional breakpoint is shown with a red plus (+) sign in the white breakpoint circle.

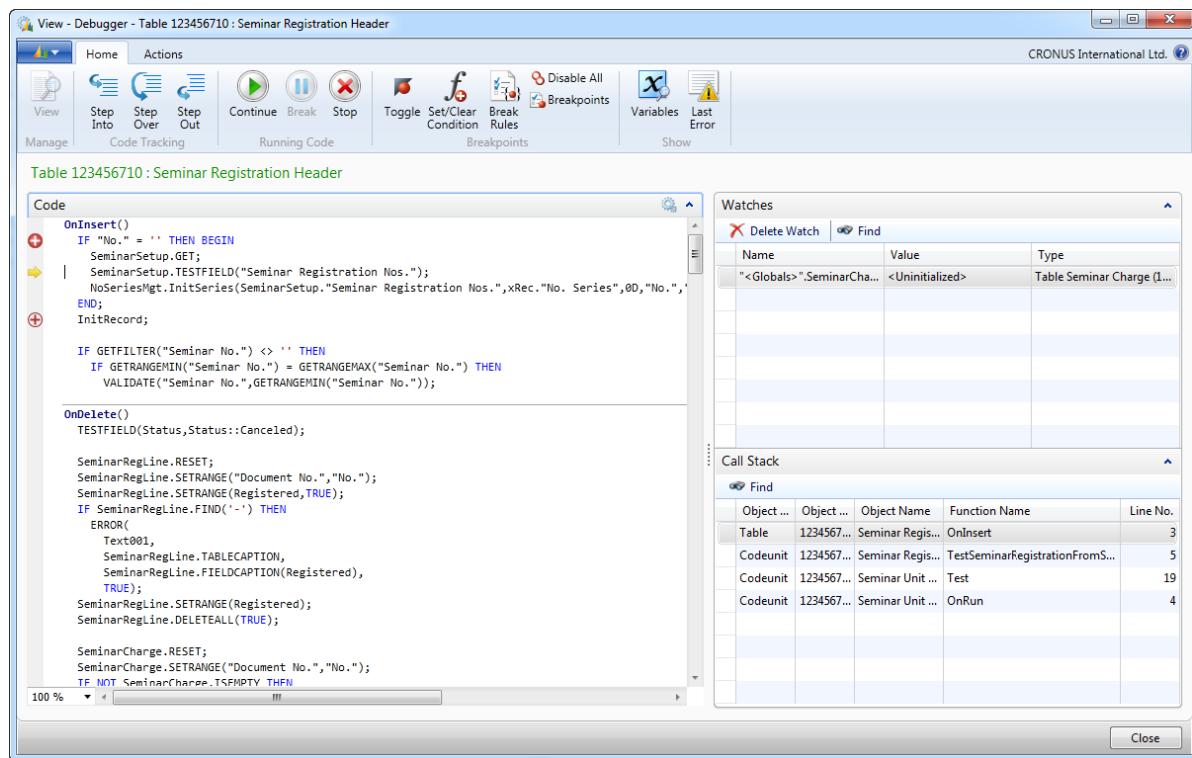


FIGURE 12.19: CONDITIONAL BREAKPOINTS

Note: Conditional breakpoints are only clearly distinct in the **Debugger** page. In the **C/AL Editor** window where they are displayed as regular breakpoints, you cannot see the difference between unconditional and conditional breakpoints.

Code Tracking

After a breakpoint is reached, you use the debugger to execute C/AL code one line at a time. This procedure is called *stepping*. The Code Tracking group on the **Home** tab provides the following three actions for stepping:

- Step Into
- Step Over
- Step Out

C/SIDE Solution Development In Microsoft Dynamics® NAV 2013

Step Into and Step Over differ in how they handle function calls. Either command instructs the debugger to execute the next line of code. If the line contains a function call, Step Into executes only the call itself and then stops at the first line of code inside the function. Step Over executes the function and then stops at the first line outside the function. Use Step Into if you want to look inside the function call. Use Step Over if you want to avoid stepping into functions.

Use Step Out when you are inside a function call and want to return to the calling function. Step Out resumes execution of your code until the function returns, and then breaks at the return point in the calling function.

The Running Code group on the **Home** tab provides the Continue action. The Continue action executes code until the next breakpoint or until execution ends.

The **Debugger** page indicates the current line of code by a yellow arrow to the left of the line that is being debugged. The arrow is positioned on the line that executes next.

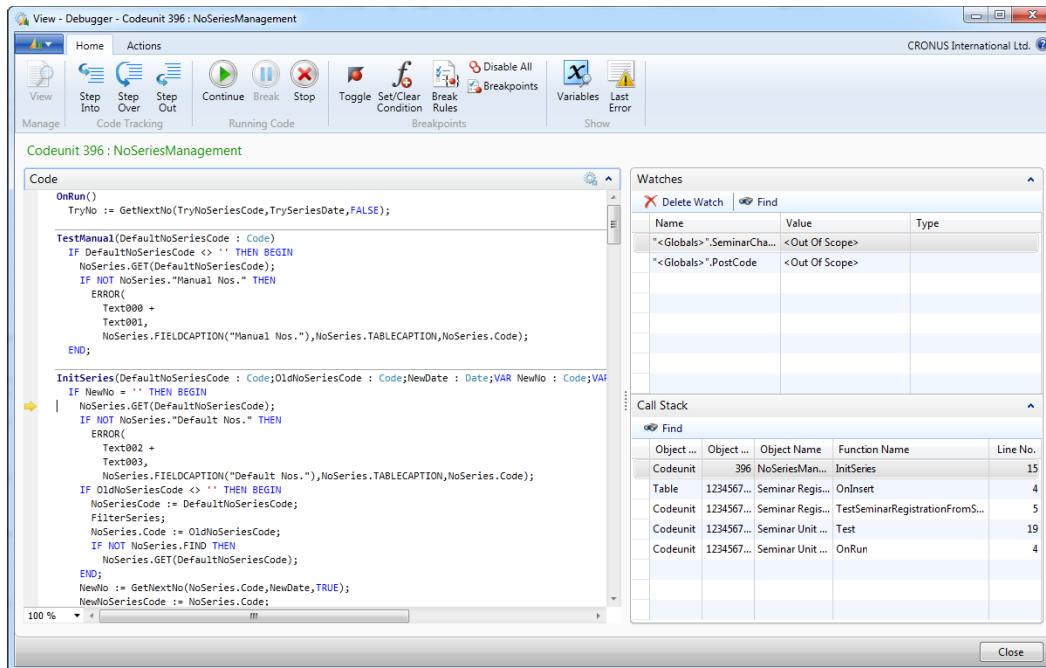


FIGURE 12.20: CURRENT LINE IN THE DEBUGGER PAGE

Variables and Watches

Bugs are frequently caused because variables contain values that differ from what you expected when you designed the application. During debugging, you frequently have to inspect the contents of variables, parameters, or text constants at each point.

Module 12: Testing and Debugging

The **Debugger Variable List** page provides an overview of all variables in scope of the current line of code. For each variable, you view the name, value, and data type.

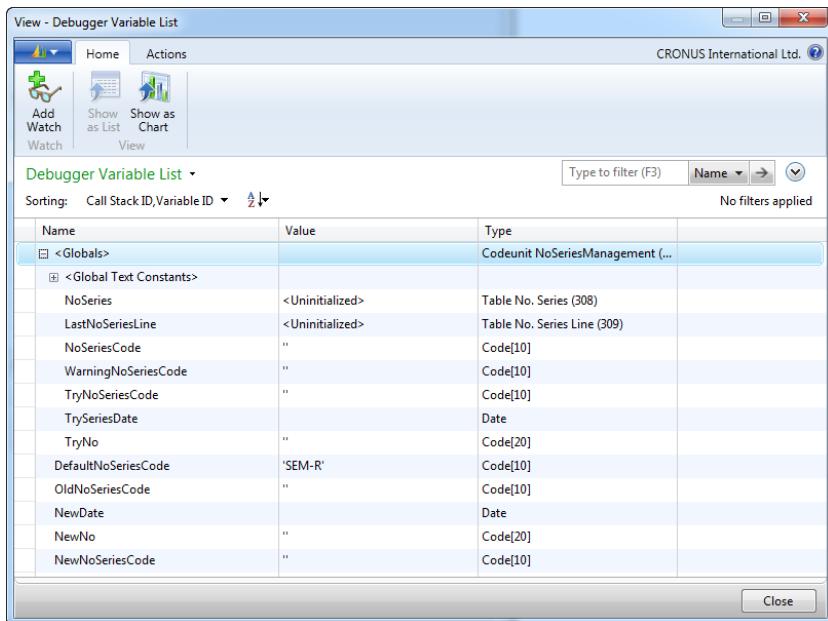


FIGURE 12.21: DEBUGGER VARIABLES LIST

You can use the **Watches** FactBox to view the values of variables. Variables that you add to the **Watches** FactBox are displayed until you delete them, even if they go out of scope in the currently executing code. This differs from the **Debugger Variable List** page, which displays only the variables that are currently in scope. If a variable is out of scope, then <Out of Scope> is displayed in the Value column of the **Watches** FactBox.

To add a variable to the Watches list, follow this procedure:

1. In the **Debugger** window, in the code viewer, rest the pointer over the variable that you want to add to **Watches**. A DataTip appears.
2. In the DataTip, click the **Watch** icon to the left of the variable name.

Alternatively:

1. In the **Debugger** window, click **Variables**.
2. In the **Debugger Variable List** window, select the variable that you want to add to **Watches**, and then click **Add Watch**.

Watches		
		X Delete Watch Find
Name	Value	Type
"<Globals>".SeminarCha...	<Uninitialized>	Table Seminar Charge (1...)
"<Globals>".PostCode	<Uninitialized>	Table Post Code (225)

FIGURE 12.22: WATCHES FACTBOX



Note: Variables that you add to the **Watches** FactBox are persisted between debugging sessions.

Call Stack

In the **Call Stack** FactBox, you can view the triggers or function calls that led to the current line of code. Each line shows a single function or a trigger, and shows information about the object type, object ID, object name, function name, and line number of the line where another function was called or a breakpoint was hit. The trigger or function that started the current transaction is at the bottom of the call stack. The current function or trigger is at the top of the call stack.

You can click any row lower in the **Call Stack** FactBox to view the code for the object that is indicated in the Call Stack line with a green arrow. The arrow indicates the line that called a function or ran a trigger higher in the call stack.

Call Stack					
 Find					
Object ...	Object ...	Object Name	Function Name	Line No.	
Codeunit	396	NoSeriesMan...	InitSeries	15	
Table	1234567...	Seminar Regis...	OnInsert	4	
Codeunit	1234567...	Seminar Regis...	TestSeminarRegistrationFromS...	5	
Codeunit	1234567...	Seminar Unit ...	Test	19	
Codeunit	1234567...	Seminar Unit ...	OnRun	4	

FIGURE 12.23: CALL STACK

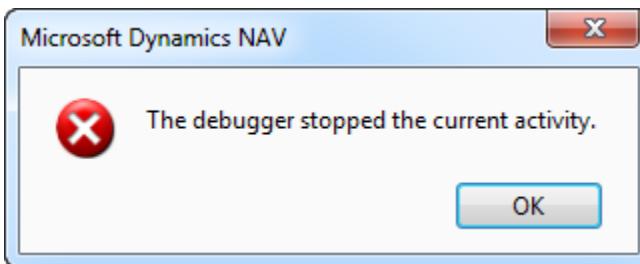
Note: When you click any line lower in the call stack, you can access the variables in scope at that point of execution. The **Watches** FactBox also updates to show the values of watched variables at that point.

Running Code

You can control how debugger runs the code by clicking actions in the Running Code group of the **Home** tab in the **Debugger** page.

To continue the execution of the code without stepping through lines, click **Continue**, or press F5. This runs the code until the next breakpoint is reached, or until all C/AL code in the current call is executed.

To stop current debugging activity, click **Stop**. This ends the current transaction, stops executing any remaining code on its execution path, and shows the following error message.

**FIGURE 12.24: RUN-TIME ERROR AFTER STOPPING THE DEBUGGER**

 **Note:** When you debug test functions that run from a test runner codeunit, each test function call is a separate transaction. When you click **Stop** in a test function, the current transaction ends and returns FAILURE as the result of the current test function. Then the test runner continues to run the remaining test codeunits. Test functions are included in the test run.

To break the debugger at the next C/AL statement, click **Break**. This action is enabled only when the debugger is not active, and when its caption includes the "Waiting for break" text. If you click **Break**, and then start an activity, such as clicking an action, starting data entry, or running an object, the debugger will break at the first C/AL statement that executes in the current session, regardless of whether a breakpoint is set on it.

Demonstration: Using the Debugger

The following demonstration shows how to use the Debugger feature of Microsoft Dynamics NAV 2013.

Demonstration Steps

1. Set a breakpoint to break when a new record is inserted into the **Seminar** table.
 - a. In Object Designer, design table 123456700, **Seminar**.
 - b. Click **View > C/AL Code**.
 - c. Select the first line of code in the OnInsert trigger, and press F9. An enabled breakpoint indicator is displayed to the left of the line.
 - d. Close the **Table Designer**.
 - e. Start the Microsoft Dynamics NAV 2013 client for Windows. Make sure that there is only one instance of it running.
2. Start the **Debugger**.
 - a. Click **Tools > Debugger > Debug Session**.
 - b. In the **Session List** page, select the session where **Client Type** is RoleTailored client, and then click **Debug**. The **Debugger** page opens.
3. Trigger the breakpoint.
 - a. In the Microsoft Dynamics NAV 2013 client for Windows, in the **Search** field, type "Seminars", then run the **Seminars** page.
 - b. In the **Seminars** page, click **New**.

Module 12: Testing and Debugging

- c. In the **Seminar Card** page, click the **Name** field. This causes the page to insert a new record into the **Seminar** table, and runs the OnInsert trigger. The **Debugger** page starts, and the code execution stops at the first line of the OnInsert trigger.
4. Add a field to the **Watches** list.
 - a. Position the pointer over the "No." text in the first line of code in the OnInsert trigger until the tooltip appears.
 - b. Click the **Add Watch** icon next to Rec.Fields."No." A watch with name <Globals>.Rec.Fields."No." appears in the **Watches** FactBox.
 - c. Check the watch to verify that the value for the **No.** field is blank.
5. Step through the code.
 - a. Click **Step Over** (or press F10) until you reach the NoSeriesMgt.InitSeries function call.



Note: Depending on how you tested labs in earlier modules, at this point you may get the following error: "The Seminar Setup does not exist." If you get this error, run page **123456702, Seminar Setup** and then repeat this step.

- b. Click **Step Into** (or press F11) to start debugging the **InitSeries** function of the NoSeriesManagement codeunit. View the **Call Stack** FactBox, where a new line is added to the top. This indicates the position of the C/AL code in the current call.
6. Add another breakpoint and continue to run the code.
 - a. Select the following line of code.

NewNo := GetNextNo(NoSeries.Code, NewDate, TRUE);

- b. Press F9 to add a breakpoint.
- c. Click **Continue** (or press F5). This continues the execution until the next breakpoint is reached.
7. Step into and out of a function and view **Watches**.
 - a. Press F11 to step into the **GetNextNo** function. A new line is added to the top of the **Call Stack** FactBox. It indicates the first line in the **GetNextNo** function of the NoSeriesManagement codeunit.
 - b. Press F10 two times.
 - c. Position the pointer over the SeriesDate variable in the first line in the **GetNextNo** function until the tooltip appears.

- d. In the tooltip, click the **AddWatch** icon to add a watch for the SeriesDate variable.
 - e. Check the **Watches** FactBox. Verify that the <Globals>.Rec.Fields."No." line indicates an <Out Of Scope> value. Verify that the SeriesDate line contains a value of 01/23/14.
 - f. Click **Step Out**. Execution returns to the last line that is reached in the **InitSeries** function.
8. Use the **Call Stack** and **Watches** FactBoxes.
- a. Add a watch for the NewNo parameter.
 - b. Verify that the value of the NewNo parameter is blank.
 - c. Press F10.
 - d. Verify that the NewNo parameter in the **Watches** FactBox contains the value that is assigned from a number series.
 - e. Verify that the SeriesDate variable is out of scope.
 - f. In the **Call Stack** FactBox, click the line for the OnInsert trigger in the **Seminar** table. The **Code** FastTab updates and shows the C/AL code in the **Seminar** table with a green arrow that indicates the line where the **InitSeries** function of the NoSeriesManagement codeunit was called.
 - g. Verify that the <Globals>.Rec.Fields."No." line shows the value for the **No.** field that was assigned from the number series.

 **Note:** The **No.** field is in the scope of the OnInsert trigger of the **Seminar** table. This is lower in the Call Stack FactBox. It contains the same value as the NewNo parameter in scope of the **InitSeries** function, because the **No.** field was passed by reference to the NewNo parameter of the **InitSeries** function.

- h. Verify that the NewNo line shows an <Out Of Scope> value.

 **Note:** The NewNo parameter is in scope for the **InitSeries** function and is not known in the scope of the OnInsert trigger that is currently shown in the **Debugger** page.

9. End debugging.
 - a. Click F5 to continue running the code. The **Debugger** switches to the "Waiting for break" mode.
 - b. Focus the **Seminar Card** page. The **No.** field on the **Seminar Card** page shows the value that was assigned from the number series.
 - c. Close the **Debugger** page.
 - d. Close the **Session List** page.

Module Review

Module Review and Takeaways

Unit testing functionality of Microsoft Dynamics NAV 2013 includes many features to fully automate testing of code and avoid regression issues. Test codeunits contain test functions. Test functions contain code that simulates transactions or user activities, and validates the application functionality against the design goals. When they are executed, test codeunits report SUCCESS or FAILURE. This indicates whether the tested functionality behaves as expected. Handler functions replace user interactions, such as confirmation dialog boxes, or modal pages. Test pages can simulate the whole user interface and most types of interaction with the user, such as calling an action, drilling down on a field in a page, or inserting a new row in a subpage on a document. Test runner codeunits automate the running of test codeunits and enable you to control which tests are executed, and which tests collected test results for logging or integrating with test management solutions.

Debugger provides the functionality to analyze the code execution, follow the code line by line as it runs, and inspect the variables to determine the causes of bugs, errors, or other types of issues.

Test Your Knowledge

Test your knowledge with the following questions.

1. Which type is not a valid codeunit subtype in Microsoft Dynamics NAV 2013?

- Normal
- Test
- TestRunner
- UnitTest

2. What kind of functions can test codeunits contain?

C/SIDE Solution Development In Microsoft Dynamics® NAV 2013

3. You can define ConfirmHandler functions only in test runner codeunits.
 True
 False
4. The OnBeforeTestRun and OnAfterTestRun triggers are defined automatically for every test runner codeunit when you set the Subtype property to TestRunner.
 True
 False
5. Which C/AL statement can you use in test code to make sure that the following statement fails?

6. When you set the TransactionModel of a test function to AutoRollback, what happens if the test code run from that function encounters a COMMIT function call?

7. Which C/AL data type do you use to simulate user interaction with a page in test code?

Module 12: Testing and Debugging

8. You can debug web services sessions in Microsoft Dynamics NAV 2013.

True

False

9. What is a conditional breakpoint and how do you define it?

Test Your Knowledge Solutions

Module Review and Takeaways

1. Which type is not a valid codeunit subtype in Microsoft Dynamics NAV 2013?

() Normal
() Test
() TestRunner
() UnitTest

2. What kind of functions can test codeunits contain?

MODEL ANSWER:

Test codeunits can contain normal, test, and handler functions.

3. You can define ConfirmHandler functions only in test runner codeunits.

() True
() False

4. The OnBeforeTestRun and OnAfterTestRun triggers are defined automatically for every test runner codeunit when you set the Subtype property to TestRunner.

() True
() False

5. Which C/AL statement can you use in test code to make sure that the following statement fails?

MODEL ANSWER:

ASSERTERROR

6. When you set the TransactionModel of a test function to AutoRollback, what happens if the test code run from that function encounters a COMMIT function call?

MODEL ANSWER:

A run time error occurs.

Module 12: Testing and Debugging

7. Which C/AL data type do you use to simulate user interaction with a page in test code?

MODEL ANSWER:

TestPage

8. You can debug web services sessions in Microsoft Dynamics NAV 2013.

() True

() False

9. What is a conditional breakpoint and how do you define it?

MODEL ANSWER:

A conditional breakpoint breaks the execution only if the Boolean expression defined on it evaluates to TRUE. You define a condition for a breakpoint by clicking the **Set/Clear Condition** action in the Debugger.

MODULE 13: SQL SERVER OPTIMIZATION

Module Overview

Microsoft Dynamics NAV 2013 runs on Microsoft SQL Server. This module covers the integration between Microsoft Dynamics NAV with Microsoft SQL Server in more detail.

Objectives

The objectives are as follows:

- Explain the advantages of SQL Server for Microsoft Dynamics NAV 2013.
- Work with and store tables and indexes.
- Use collation options and descriptions.
- Introduce SQL Server Query Optimizer.
- Explain the areas within Microsoft Dynamics NAV that are to be optimized.
- Demonstrate how the Microsoft Dynamics® NAV database driver allows the Microsoft Dynamics NAV clients to communicate with SQL Server.
- Introduce the value of optimizing indexes to maximize performance.
- Describe the performance effect of locking, blocking, and deadlocks.
- Present how SIFT data is stored in SQL Server.

SQL Server for Microsoft Dynamics NAV

SQL Server is a comprehensive database platform that provides enterprise-class data management with integrated business intelligence (BI) tools. SQL Server can be characterized as a set-based engine. This means that SQL Server is very efficient when it retrieves a set of records from a table, but less so when records are accessed one at a time.

Access to SQL Server from Microsoft Dynamics NAV is performed with the Microsoft Dynamics database driver that is discussed later in this module. The SQL Server interface from Microsoft Dynamics NAV Server was rewritten for Microsoft Dynamics NAV 2013 to use ADO.NET instead of ODBC. The advantages of the new access layer are described later in this module.

When SQL Server receives a query (in the form of a Transact-SQL statement), it uses the SQL Query Optimizer to create and execute the query. The Query Optimizer evaluates the query and makes decisions about how to execute the query in the execution plan. For example, the Query Optimizer decides which index to use, whether to use parallel execution, and so on.

Query Optimizer assumes that the client generates queries according to its own logic, and that these queries are not optimized for SQL Server. The primary criteria that Query Optimizer uses to decide which execution plan to use is the performance cost of executing the query.

SQL Server stores data in B+ tree structures. One index is used to store the data physically on a disk. Other indexes are used to find a range and point to the data in the main index. This main index is called the *Clustered Index*. On SQL Server, you can define any index as the main (clustered) index. However, on the Microsoft Dynamics NAV Development Environment it is the default primary key of the table.

 **Note:** By default, the primary key of a table in Microsoft Dynamics NAV becomes the clustered index. You can use the key property Clustered to define another key to become the clustered index on SQL Server. We recommend that the primary key and the clustered index be the same.

Microsoft Dynamics NAV 2013 no longer uses server cursors to retrieve records. Instead, records are retrieved by using multiple active result sets (MARS).

Representation of Microsoft Dynamics NAV Tables and Indexes in SQL Server

By default, Microsoft Dynamics NAV provides unique data for each company in its database. On SQL Server, each company in the development environment has its own copy of each table.

Representation of Microsoft Dynamics NAV Tables and Indexes in SQL Server

Each table in the Development Environment has a corresponding table in SQL Server for every company in the database, with a name in the following format:

Table Name Format	Example
<Company Name>\$< Table Name>	CRONUS International Ltd_\$G_L Entry

However, you can share data across companies by setting the DataPerCompany table property to FALSE. In Microsoft Dynamics NAV terms, this is known as *data common to all companies*. When the DataPerCompany property is turned off, there is just one table in SQL Server that is accessed from every company in the database. The naming convention for these common tables on SQL Server is the same, but without the <Company Name>\$ portion.

The Microsoft Dynamics NAV Development Environment uses naming conventions that comply with SQL Server, such as not using special characters. Some special characters are available in the table designer, and they are translated to comply with the character set that is used on SQL Server.

The table has several indexes that represent the keys that are designed and enabled in the table designer. The indexes have generic names in the following format.

Index name format	Example
\$<Index Number>	\$1, \$2, and so on

However, the primary key index uses the following name format.

Primary key name format	Example
<Company Name>\$< Table Name>\$0	CRONUS International Ltd_\$G_L Entry\$0

C/SIDE Solution Development in Microsoft Dynamics® NAV 2013

By default, Microsoft Dynamics NAV clusters the primary key. Also by default, Microsoft Dynamics NAV adds the rest of the primary key to every secondary index. This makes the indexes unique and complies with the best practices that are defined for SQL Server. Developers can make additional changes to the way indexes are defined on SQL Server by using the MaintainSQLIndex, SQLIndex, and Clustered properties on the keys that are defined in the table designer.

To obtain a list of indexes and their definition in SQL Server, run the sp_helpindex stored procedure in a query window, as follows.

Code Example

```
sp_helpindex "CRONUS International Ltd_$G_L Entry"  
GO
```

The query outputs the index name if the index is clustered or unique, if there is a primary key constraint, and also the index keys that are defined in the index.

There are some differences between the Dynamics NAV and SQL Server terminology, as the following list describes.

SQL Server Terminology	Dynamics NAV Terminology
Primary key constraint	Primary key
Clustered index	No equivalent
Nonclustered index	Secondary key
Index key	Field in a key definition

In SQL Server, a table does not have to have a clustered index. This is known as a *heap* and can be used for archiving, because data is stored as it arrives. However, heaps are not ideal for tables that are read, because reading from an unstructured source is too slow.

Collation Options

SQL Server supports several collations. A *collation* encodes the rules that govern the correct use of characters for either a language, such as Macedonian or Polish, or an alphabet, such as Latin1_General (the Latin alphabet that is used by Western European languages). Microsoft Dynamics NAV 2013 only supports the latest Windows collations. Any database that is upgraded by Microsoft Dynamics NAV 2013 is converted to the most recent corresponding Windows collation. Each SQL Server collation specifies the following three properties:

Module 13: SQL Server Optimization

- The sort order to use for Unicode data types (nchar, nvarchar, and ntext). A *sort order* defines the sequence in which characters are sorted, and the way that characters are evaluated in comparison operations.
- The sort order to use for non-Unicode character data types (char, varchar, and text).
- The code page that is used to store non-Unicode character data.

You can specify SQL Server collations at any level. Each instance of SQL Server has a defined default collation. This is the default collation for all objects in that instance of SQL Server, unless otherwise specified. Each database can have its own collation. This can differ from the default collation. You can specify separate collations for each column, variable, or parameter. Microsoft Dynamics NAV sets the database default collation for reference only. All columns that are created by Microsoft Dynamics NAV explicitly has the collation set.

It is a good practice to set the collation as generic as possible to the language that is most common to users. If all users speak the same language, set up SQL Server with a collation that supports that language. For example, if all users speak French, define a French collation on SQL Server. If users speak multiple languages, define a collation that best supports the requirements of the various languages. For example, if the majority of users speak western European languages, then Latin1_General collation is the best option.

Collation settings are defined in the Microsoft Dynamics NAV Development Environment when a database is created. You can change it afterward with some limitations.

Demonstration: Open the Collation window

Ask Mort to verify the collation settings of the Dynamics NAV Database to update the documentation.

Demonstration Steps

1. To open the collation window follow these steps.
 - a. Start the Microsoft Dynamics NAV Development Environment.
 - b. In the **File** menu, click **Database** and then **Alter**. The **Alter Database** window opens.
 - c. In the **Alter Database** window, click the **Collation** tab. See Collation Tab of the Alter Database Window.

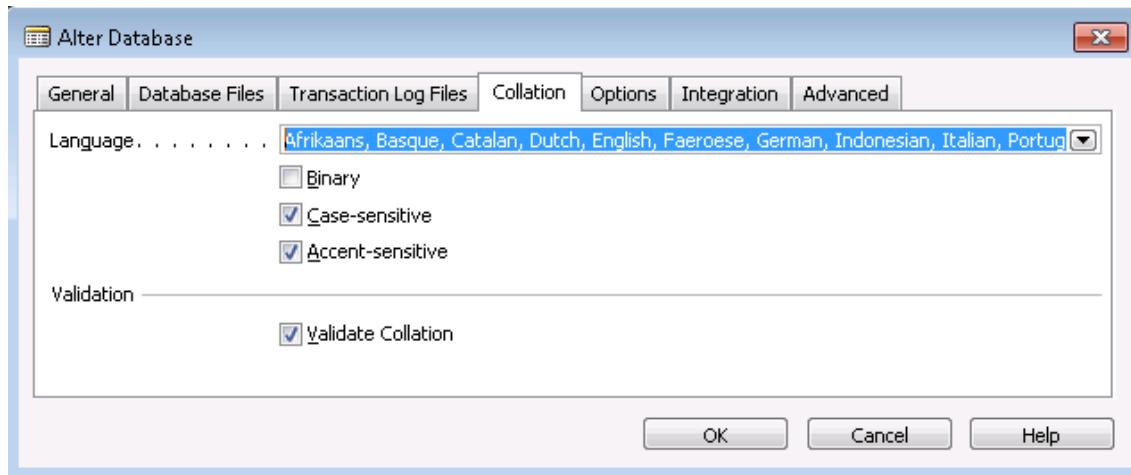


FIGURE 13.1: THE COLLATION TAB OF THE ALTER DATABASE WINDOW

Collation Description

Select the name of a specific Windows collation from the drop-down list. Note that when the **Validate Code Page** field is checked, only valid subsets of collations are available in the list based on the Windows Locale. For example, the following are subsets for the Latin1_General (1252) locale consecutively:

- Afrikaans, Basque, Catalan, Dutch, English, Faeroese, German, Indonesian, Italian, Portuguese
- Danish, Norwegian

Sort Order

Select Sort Order options to use with the collation that you selected. These options are Binary, Case-sensitive, and Accent-sensitive. Binary is the fastest sorting order, and is case-sensitive and accent-sensitive. If you select Binary, the Case-sensitive and Accent-sensitive options are not available.

SQL Server Query Optimizer

Query Optimizer is the brain of SQL Server when it decides how to execute a query. SQL Server collects statistics about individual columns (single-column statistics) or sets of columns (multicolumn statistics). Query Optimizer uses statistics to estimate the selectivity of expressions, and therefore, the size of intermediate and final query results. Good statistics let the optimizer accurately assess the cost of different query plans and select a high-quality plan. All information about a single statistics object is stored in several columns of a single row in the sysindexes table, and in a statistics binary large object (statblob) that is kept in an internal-only table.

SQL Server Statistics

SQL Server maintains some information at the table level. Tables are not part of a statistics object, but SQL Server uses them occasionally during query cost estimation. The following data is stored at the table level:

- Number of rows in the table or index (rows column in sys.sysindexes)
- Number of pages that are occupied by the table or index (dpages column in sys.sysindexes)

SQL Server collects the following statistics about table columns and stores them in a statistics object (statblob):

- Time that the statistics are collected
- Number of rows that are used to produce the histogram and density information (described later)
- Average key length
- Single-column histogram that includes the number of steps

A *histogram* is a set of up to 200 values of a given column. All or a sample of the values in a given column are sorted. The ordered sequence is divided into up to 199 intervals so that the most statistically significant information is captured. Generally, these intervals are not of equal size.

 **Note:** The way that Microsoft Dynamics NAV executes queries disables the use of histograms. You do this to enable performant reuse of query plans without regard to actual parameter values. Microsoft Dynamics NAV uses the OPTIMIZE FOR UNKNOWN option to guarantee this type of computation of the query execution plan.

Users can view the statistical information when they run the DBCC SHOW_STATISTICS command. For example, they can run it for index \$6 in the **Cust. Ledger Entry** table, as follows.

Code Example

```
DBCC SHOW_STATISTICS  
("CRONUS International Ltd$_Cust_ Ledger Entry","$6")  
GO
```

C/SIDE Solution Development in Microsoft Dynamics® NAV 2013

The result set has three sections, similar to the following.

The screenshot shows the SQL Query Results window in SSMS. The query executed was:

```
DBCC SHOW_STATISTICS
L ("CRONUS International Ltd_$Cust_Ledger Entry", "$6")
GO
```

The results are displayed in three tabs: Results, Messages, and Client Statistics.

Results Tab:

Name	Updated	Rows	Rows Sampled	Steps	Density	Average key length	String Index	Filter Expression	Unfiltered Rows
\$6	Aug 10 2012 11:02AM	58	58	4	0	30.34483	NO	NULL	58

Client Statistics Tab:

All density	Average Length	Columns
1	0.25	Document Type
2	0.04166667	Document Type, Customer No_
3	0.025	Document Type, Customer No_ Posting Date
4	0.025	Document Type, Customer No_ Posting Date, Currency Code
5	0.01724138	Document Type, Customer No_ Posting Date, Currency Code, Entry No_

Statistics Distribution Tab:

RANGE_HI_KEY	RANGE_ROWS	EQ_ROWS	DISTINCT_RANGE_ROWS	AVG_RANGE_ROWS
1	0	8	0	1
2	0	43	0	1
3	0	6	0	1
4	0	1	0	1

FIGURE 13.2: DBCC SHOW_STATISTICS RESULTSET

Suppose that a user filters on the Document Type column in the **Cust. Ledger Entry** table, for example, to look for all Credit Memo type entries. The Credit Memo type entries have a value of Document Type that is equal to three. Microsoft Dynamics NAV issues a query similar to the following.

Code Example

```
SELECT * FROM
"CRONUS International Ltd_$Cust_Ledger Entry"
WHERE
"Document Type" = 3
GO
```

The Query Optimizer analyzes the usefulness of every index in the table so that the query is executed at minimal cost. An index minimizes first the cost of data retrieval, followed by costs of sorting, and so on. When you analyze this particular index (index \$6) from a data retrieval perspective, the Query Optimizer makes most of its decisions based on the statistics in the following way.

Document Type is filtered because there are only three distinct values in the index (refer to the All Density column in the previous table). This indicates that 0.25 of the table is within the filtered set.

Module 13: SQL Server Optimization

Based on this, the Query Optimizer decides that there is no point in using this index to do this operation. This is because it has to load the index, scan the range, and look up the data in the clustered indexes to return the results. Because there is no better way to read the data, Query Optimizer decides to do a Clustered Index Scan instead.

As a rule, if the selectivity is close and better than one percent, the index is considered good. Be careful with this simple rule, because operations such as SELECT TOP 1 (asking for the first record in a set) escalate the index benefit, and the index will probably be used.

To continue with this example, filtering on a unique value makes the index help with selectivity, such as the following.

Code Example

```
SELECT * FROM  
"CRONUS International Ltd_$Cust_ Ledger Entry"  
  
WHERE  
"Document Type" = 3 AND  
"Customer No_" = '10000'  
  
GO
```

The combined selectivity is used, and a plan is calculated. This may result in a decision to use this index.

However, if users do not filter on an index key or use the <> (not equal operator) or use the OR operator, SQL Server cannot combine the subqueries. If the <> operator is used, then the optimizer will assume 1-selectivity * number of rows. If OR is used, the optimizer combines the selectivity. This may result in bad behavior, such as the following.

Code Example

```
SELECT * FROM  
"CRONUS International Ltd_$Cust_ Ledger Entry"  
  
WHERE  
"Customer No_" = '10000' AND
```

```
(("Document Type" = 3) OR ("Document Type" = 4))
```

```
GO
```

In this example, the Query Optimizer decides to use the index, doing a non-clustered index seek, but it has to traverse the area of Document Type = 3. Because the set is read from start to finish, it has the same effect as if the user did a table scan.

Similarly, if the user leaves one of the index keys unfiltered, all the subtree index entries must be scanned. There is one simple rule about performance: Scan is bad. Seek is good. Developers must avoid scans as much as they can, ensure that indexes are of a good selectivity, and that the queries do not have scan-like behavior.

On the other side of the spectrum is an index that is too complex and is designed to fully match the whole query, such as with eight index keys. If the index has only four index keys, SQL Server scans a slightly larger set to provide the required records, but at the extreme cost of having to maintain the composite index. This delays every modification in the table, because SQL Server has to update the index accordingly. In most cases, users have to optimize the transaction speed. If you over-index the tables, then users pay a price in performance. If a specific report is slow when fewer composite indexes are used, it might be worth it because processing (such as posting inventory) is quicker.

Additionally, if a fairly small set is ordered by a different column, it is not necessary for the index to fully support the sorting. SQL Server can efficiently sort small result sets quickly.

The previous situation demonstrates that the way that indexes are designed and used can severely affect SQL Server performance. Use the following principles:

- Reduce the number of indexes for faster table updates.
- Design indexes with index keys of good selectivity.
- Put index keys that are more likely to be filtered toward the beginning of an index.
- If the filtered index keys point to a low number of records, you do not have to add additional index keys to support index selectivity or sorting. SQL Server returns the set sorted as you want.
- There is no point in indexing empty (unused) columns, or columns that have the same value for all rows. This creates an additional overhead with no benefit.
- Make sure that users filter on unique values in indexes; otherwise, SQL Server performs in a manner that is similar to table scans.

- Do not make over selective index keys. If users index on **DateTime** fields, for example, they force creation of a unique index leaf in the index for each record in the table.
- Put date fields toward the end of the index, since the index is not always filtered. If an index is always filtered on a unique value, then it is a good index.

To determine whether an index is good, imagine a telephone book or list of personal details that are designed to find people by name and surname, date of birth, or Social Security number. Compare this to a telephone book that is indexed by gender. When you design indexes, select those that support a high level of selectivity. Common sense applies.

Optimize a Microsoft Dynamics NAV Application

There are several areas where users must focus when they optimize Microsoft Dynamics NAV applications. These areas follow, in order of importance (based on the processing costs):

- SIFT
- Indexes
- Locks
- Suboptimum code
- GUI

Optimizing SIFT Tables

Use SIFT tables in Microsoft Dynamics NAV version 5.0 and older, to implement SIFT on the SQL Server, and store aggregate values for SumIndexFields for keys in the source tables. Starting with version 5.0 Service Pack 1, these SIFT tables are replaced by indexed views. Separate SIFT tables are no longer part of Microsoft Dynamics NAV on SQL Server. This discussion section is included because Microsoft Dynamics NAV developers can encounter issues about SIFT tables in implementations of older versions of Microsoft Dynamics NAV. There is similar overhead with indexed views.

The overhead of the separate SIFT indexes is very large and should be carefully considered for activation. By default, Microsoft Dynamics NAV starts the SIFT indexes when users create a new index with SumIndexFields. Users should review all existing SIFT indexes and decide whether they really have to keep them started.

Users can deactivate the creation and maintenance of a SIFT index by using the MaintainSIFTIndex property in the Microsoft Dynamics NAV key designer. If they make the property FALSE, and there is no other maintained SIFT index that supports the retrieval of the cumulative sum, Microsoft Dynamics NAV asks SQL Server to calculate the total.

For example, if users have a **Sales Line** table and put Amount in the SumIndexFields for the primary key (Document Type, Document No., Line No.), a new SIFT table named **CRONUS International Ltd_\$37\$0** is created and maintained. When you use a **CALCSUM** function to display a FlowField in Microsoft Dynamics NAV that displays the total of all Sales Lines for a specific Sales Header (Order ORD-980001), the resulting query looks exactly like the following.

Code Example

```
SELECT SUM(SUM$Amount) FROM  
"CRONUS International Ltd_$Sales Line$VSIFT$0"  
  
WHERE  
  
"Document Type" = 1 AND  
  
"Document No_" = 'ORD-980001'
```

If users make the SIFT table unavailable by clearing the **MaintainSIFTIndex** check box, Microsoft Dynamics NAV still works, and the resulting query resembles the following.

Code Example

```
SELECT SUM("Amount") FROM  
"CRONUS International Ltd_$Sales Line"  
  
WHERE  
  
"Document Type" = 1 AND  
  
"Document No_" = 'ORD-980001'
```

This is a very light load on CPU overhead compared to the large cost of maintaining the SIFT indexes.

SIFT tables are very useful when users have to total a larger number of records. Considering this information, users can check existing SIFT indexes and see whether they need some level of detail. There is no need, for example, to store a cumulative total of just a few records.

Optimize Indexes

The second largest Microsoft Dynamics NAV overhead is the processing load to maintain indexes. The Microsoft Dynamics NAV database is over-indexed, because customers require certain reports and pages to be ordered in different ways. Therefore, many keys are made available for each sequence. However, SQL Server can sort data directly and quickly if the set is small. Therefore, you do not have to keep indexes for sorting purposes only. For example, in the **Warehouse Activity Line** table, there are several keys that begin with **Activity Type** and **No.** fields, such as the following.

Code Example

```
"Activity Type,No.,Sorting Sequence No."  
"Activity Type,No.,Shelf No."  
"Activity Type,No.,Action Type,Bin Code"  
etc.
```

These indexes are not needed on SQL Server, because the Microsoft Dynamics NAV code always filters on **Activity Type** and **No.** fields when it uses these keys. In SQL Server, the Query Optimizer looks at the filter and realizes that the clustered index is Activity Type No_ and Line No_. It also determines that the set is small, and that it does not have to use an index to retrieve the set and return it in that specific order. It uses only the clustered index for these operations.

Since the whole functionality is not used by customers, if they never select the stock by **Sorting Sequence No.**, then they do not have to maintain the index.

Developers must analyze the existing indexes and focus on use and benefits compared to the processing overhead, and then determine the appropriate action. Decide between disabling the index completely by using the key property **Enable**, the **KeyGroups** property, or the **MaintainSQLIndex** property. Indexes that remain active can change structure by using the **SQLIndex** property. Developers can also make the table clustered by a different index.

Enabled Property The **Enabled** property turns a specific key on and off. If you are not using the key or if you rarely use the key, you may want to mark it as disabled for performance reasons.

KeyGroups Property

Make one or more keys a member of a predefined key group. This allows for the key to be defined, but enabled only when it is used. Use the **KeyGroups** property to select the predefined key groups. Select the **KeyGroups** option on the **Database Information** window (select **File > Database > Information > Tables**). There are key groups already defined, such as **Acc(Dim)**, **Item(MFG)**, but more can be created and assigned to keys.

The purpose of key groups set up a group of special keys that are infrequently used (such as for a special report that is run one time every year). Adding many keys to tables eventually decreases performance. When you use key groups it makes it possible to have the necessary keys defined, but only active when it is necessary.

MaintainSQLIndex Property

This property determines whether an SQL Server index that corresponds to the Microsoft Dynamics NAV key should be created (when set to **Yes**) or dropped (when set to **No**). A Microsoft Dynamics NAV key is created to sort and search for data in a table by the required key fields. However, SQL Server can sort data without an index on the fields to be sorted. If an index exists, sorting by the fields matching the index is faster, but modifications to the table will be slower. The more indexes there are on a table, the slower the modifications become. If a key must be created to allow for only occasional sorting (for example, when it is running infrequent reports), users can disable this property to prevent slow modifications to the table. Additionally, if there are many keys in a table, SQL Server does not use all the corresponding indexes. To eliminate the overhead of the indexes we recommend not maintaining them by setting MaintainSQLIndex to **No**. Then, if there is a reference in the C/AL code to the key, it does not cause an error message because the key still exists.

SQLIndex Property

This property lets users define the fields that are used in the SQL index. The fields in the SQL index can be any of the following:

- Different from the fields that are defined in the key in Microsoft Dynamics NAV. There can be fewer fields or more fields.
- Arranged in a different order.

If the key in question is not the primary key and you use the SQLIndex property to define the index on SQL Server, the index that is created contains exactly the fields that users specify. It is not necessarily a unique index. It will only be a unique index if it contains all fields from the primary key.

If you define the SQL index for the primary key, it must include all the fields that are defined in the Microsoft Dynamics NAV primary key. You can add additional fields that can be rearranged to suit individual needs.

Clustered Property

Use this property to determine which index is clustered. By default, the index that corresponds to the Microsoft Dynamics NAV primary key is clustered.

The Index Usage Query

The Index Usage Query was released on the Microsoft Dynamics NAV Team Blog and can be found in the following location.

 *Index Usage Query on Dynamics NAV Team Blog*

<http://go.microsoft.com/fwlink/?LinkId=269777>

This query shows a list of all tables and indexes in an SQL database to help identify tables where the most blocks occur, and to give a starting point for index tuning.

Use the Index Information Query to see the number of records in each table. By changing ORDER BY, you also can use it to see which index causes the most blocking, wait time, updates, or locks. Use the Index Information Query to compare Index Updates with Index Reads for an idea of cost versus benefit for each index. Then you can use this information to decide whether to maintain certain indexes.

To summarize, the Index Information Query provides information about the following:

- Index and Table Information
- Index usage (benefits and costs information for each index)
- Index locks, blocks, wait time, and updates per read (cost versus benefit)

You must execute the query in the Microsoft Dynamics NAV database on SQL Server. It creates a table named **z_IUQ2_Temp_Index_Keys** and uses various Microsoft Dynamic Management Views to collect information for each index for this table. When you execute the complete query, it can take several minutes to execute for each company in the database. If you just want to change sorting or display the results again later, you only have to run the last part of the query as follows.

Code Example

```
-- Select results

SELECT

[F_Table_Name] TableName,
[F_Row_Count] No_Of_Records,
[F_Data] Data_Size,
```

```
[F_Index_Size] Index_Size,  
[F_Index_Name] Index_Name,  
[F_User_Updates] Index_Updates,  
[F_User_Reads] Index_Reads,  
CASE WHEN  
    F_User_Reads = 0 THEN F_User_Updates  
ELSE  
    F_User_Updates / F_User_Reads  
END AS Updates_Per_Read,  
[F_Locks] Locks,  
[F_Blocks] Blocks,  
[F_Block_Wait_Time] Block_Wait_Time,  
[F_Last_Used] Index_Last_Used,  
[F_Index_Type] Index_Type,  
[Index_Key_List] Index_Fields  
FROM z_IUQ2_Temp_Index_Keys  
--order by F_Row_Count desc, F_Table_Name, [F_Index_ID]  
--order by F_User_Updates desc  
--order by Blocks desc  
--order by Block_Wait_Time desc  
--order by Updates_Per_Read desc  
ORDER BY F_Table_Name
```

The last lines suggest various other ORDER BY clauses that you can use to replace the default ORDER BY clause that is by Table Name. There are several other columns that are available. You can easily change the query, for example ORDER BY user updates, to see the indexes that are causing the largest overheads, and then check the actual usage of these indexes.

Code Example

```
--order by F_Row_Count desc, F_Table_Name, [F_Index_ID]
--order by F_User_Updates desc
--order by Blocks desc
--order by Block_Wait_Time desc
--order by Updates_Per_Read desc
ORDER BY F_Table_Name
```

To analyze query results, it is important to understand that the Index Information Query uses SQL Server Dynamic Management Views to collect index information. Some of these Dynamic Management Views access their information from the SQL Server cache. The cache is located in the RAM memory of the server. If you recently restarted SQL Server, then enough time may not have elapsed to warm up the cache. This means that the cache is not representing the actual workload on the server. Therefore, the results of the Index Information Query are misleading. To avoid this scenario, make sure that before you run the Index Information Query, enough time has elapsed since the last restart of SQL Server.

When the query results contain NULL values for most of the indexes in the database, it means that there is not enough information in the cache about the index usage. To obtain solid results, wait until a representative workload processes in Microsoft Dynamics NAV.

The following is an example of the query results on a Microsoft Dynamics NAV Demo Database.

	TableName	No_Of_Records	Data_Size	Index_Size	Index_Name	Index_Updates	Index_Reads	Updates_Per_Read
25	Object Tracking	4316	208	152	\$1	43	9443	0
26	Object Metadata	3862	37048	16	Object Metadata\$0	28	178	0
27	CRONUS International Ltd.\$Sales Line	103	224	176	\$8	0	2	0
28	CRONUS International Ltd.\$Detailed Cust...	78	32	144	CRONUS International Ltd.\$Detailed...	0	56	0
29	CRONUS International Ltd.\$Value Entry	377	256	448	CRONUS International Ltd.\$Value E...	0	53	0
30	CRONUS International Ltd.\$Detailed Cust...	43	8	8	VSIFTIDX	0	352	0
31	CRONUS International Ltd.\$Sales Line	103	224	176	CRONUS International Ltd.\$Sales Li...	0	44	0
32	CRONUS International Ltd.\$Sales Header	52	136	128	\$2	0	252	0
33	CRONUS International Ltd.\$Sales Header	52	136	128	\$3	0	252	0
34	CRONUS International Ltd.\$Sales Header	52	136	128	\$4	0	0	0
35	CRONUS International Ltd.\$Detailed Cust ...	52	8	8	VSIFTIDX	0	478	0

Query executed successfully. NAV7DEMO|SQL2008R2EXPRESS (...) NAV7DEMO\Administrator... NAV7_DEV1 00:00:00 2169 rows

FIGURE 13.3: RESULTS OF THE INDEX USAGE QUERY

C/SIDE Solution Development in Microsoft Dynamics® NAV 2013

The column on the left side shows data for the table (No. of records, data and index size) where you can view the effect of indexes on the table.

The columns on the right side show data for each index. This includes Updates (costs) and Reads (benefits), and when it was last used since the last time SQL Server was restarted.

The following table describes the different columns of the Index Usage Query.

Field Name	Description
TableName	The name of the table.
No_Of_Records	Number of records in the table.
Data_Size	The current size of table data.
Index_Size	The current size of the index.
Index_Name	The name of the index.
Index_Updates	The number of times the index was updated by SQL Server.
Index_Reads	The number of times the index was read by SQL Server.
Updates_Per_Read	The number of Updates divided by the number of reads.
Locks	The number of times the index was involved in a lock.
Blocks	The number of times the index was involved in a block.
Block_Wait_Time	The time that the index was blocked (in ms).
Index_Last_Used	The datetime the index was last used.
Index_Type	The type of index (Clustered, NonClustered.)
Index_Fields	The fields of the index.

Define Keys to Improve Performance

When you write C/AL code that searches through a subset of the records in a table, you must consider the keys that are defined for the table and then write code that optimizes for the keys. For example, the entries for a specific customer are usually a small subset of a table that contains entries for all customers.

Module 13: SQL Server Optimization

The time that is required to complete a loop through a subset of records depends on the size of the subset. If a subset cannot be located and read efficiently, then performance deteriorates.

To maximize performance, you must define the keys in the table that support the code that you run. Then you must specify these keys correctly in your code.

For example, to retrieve the entries for a specific customer, you apply a filter to the **Customer No.** field in the **Cust. Ledger Entry** table. To run the code efficiently on Microsoft SQL Server, you must define a key in the table that has **Customer No.** as the first field.

The table could have the following keys:

- Entry No.
- Customer No.
- Posting Date

The following is an example of code that finds a subset of records.

Code Example

```
SETRANGE("Customer No.",'1000');

IF FIND('-') THEN

REPEAT

UNTIL NEXT = 0;
```

SQL Server automatically selects the index to use to retrieve data in the most efficient way. SQL Server calculates the cost of retrieving data by using different indexes. Then it selects the path that has the smallest cost. For Microsoft Dynamics NAV, that calculation is based only on the statistical distribution of values in a column.

For example, if a table contains 1000 rows and a column in the table contains either the value 0 or the value 1, then that column is said to have a *low selectivity*. If a column contains the values ranging from 1 to 500, then the column is said to have a *high selectivity*. In the following code example, SQL Server selects an index that contains the HighSelectivityColumn. Then it sorts the rows by the LowSelectivityColumn.

Code Example

```
SETCURRENTKEY("LowSelectivityColumn");
SETFILTER("LowSelectivityColumn",'1');
SETFILTER("HighSelectivityColumn",'777');
FIND('-')
```

Implicit/Explicit Locking

There are additional considerations to make when you work with Microsoft Dynamics NAV on SQL Server. Microsoft Dynamics NAV is designed to read without locks, and it locks only if it is necessary. If records will be changed, indicate that intent (use explicit locking) so that the data is read correctly.

Implicit Locking

The following table demonstrates implicit locking. The C/AL pseudo-code on the left is mapped to the equivalent action on SQL Server.

Sample code	Result
TableX.FIND('-');	SELECT * FROM TableX WITH (READUNCOMMITTED) (the retrieved record time stamp = TS1)
TableX.Field1 := Value;	
TableX.MODIFY;	performs the update UPDATE TableX SET Field1 = Value WHERE TimeStamp <= TS1

The call to TableX.MODIFY will implicitly lock the table from that point forward. This means that any call to FIND made on the table after this point will have the WITH(UPDLOCK) applied.

Because the record was read without locking the record, NAV guarantees data consistency by automatically adding a check in the where clause on the timestamp. This means that other users' changes are not overwritten.

Explicit Locking

If users indicate that they plan to modify the record by using explicit locking, then the SQL sent is slightly different, as shown by the following pseudo code.

Sample code	Result
TableX.LOCKTABLE;	Indicates to explicit lock.
TableX.FIND('');	SELECT * FROM TableX WITH (UPDLOCK) (the retrieved record timestamp = TS1)
TableX.Field1 := Value;	
TableX.MODIFY;	UPDATE TableX SET Field1 = Value (The retrieved record timestamp is guaranteed to be TS1.)

Problems with NEXT

Sometimes, the NEXT command causes performance problems in Microsoft Dynamics NAV. Users should pay particular attention and avoid these situations. The problem is if users change the content or the definition of the result-set, then the set must be retrieved again. Microsoft Dynamics NAV guarantees that all changes that are made by the current user are included in a result-set. This is known as a *dynamic result-set*.

Be aware that if during traversal of a result-set any of following actions are performed, then retrieval of the result-set will be reset (Issue a new SQL statement):

- Changed filter
- Changed sorting
- Modified key value.
- Inserted a record.
- Modified, deleted, or inserted a record by using another instance of a record.
- Changed transaction type.

C/SIDE Solution Development in Microsoft Dynamics® NAV 2013

The following code examples demonstrate this problem.

Code	Result
SETCURRENTKEY(FieldA);	
SETRANGE(FieldA,Value);	
FIND(' -')	Set or part of the set is retrieved, first record loaded.
REPEAT	
FieldA := NewValue;	Record position is now outside the set.
MODIFY;	Record is put outside the set.
UNTIL NEXT = 0;	The system is asked to go to NEXT from position, but from outside the set.

"Jumping" through data - NEXT "from current record content

Code	Result
SETCURRENTKEY(FieldA);	
SETRANGE(FieldA,Value);	
FIND(' -');	Set based on filter on FieldA.
REPEAT	
...	
...	
FieldA := <SomeValue>	The current key value was changed. So, the call to next has to retrieve the record conforming with being greater than <SomeValue>.
...	
UNTIL NEXT = 0	The system is asked go to NEXT from undefined position.

FindFirst(only) followed by call to next

Code	Result
SETRANGE(FieldA,Value);	
FINDFIRST;	This call is optimized for retrieving the first record only. Therefore, calling next after this issued a new SQL query.
REPEAT	
...	
UNTIL NEXT = 0;	The system is asked to go to NEXT on non-existing set.
SETRANGE(FieldA,Value);	

Solutions

To eliminate performance problems with NEXT, consider the following solutions:

- Use a separate looping variable.
- Read records to temporary tables, modify within, and write back afterwards.

Suboptimum Coding and Other Performance Penalties

Taking performance into consideration frequently influences programming decisions. For example, if users do not use explicit locking, or if the program loops and provokes problems with NEXT, they frequently pay a big price in performance. Users also have to review their code and see how many times the code reads the same table, or use ISEMPTY or COUNT for checking if there is a record (IF COUNT = 0, IF COUNT = 1. Additionally, there are features in the application that must be avoided or minimized. Users should review the application setup for the performance aspect and take corrective actions if they can.

Data Access Redesign

The SQL Server interface from Microsoft Dynamics NAV Server was rewritten for Microsoft Dynamics NAV 2013 to use ADO.NET instead of ODBC.

Simplified Deployment

The new ADO.NET interface is a managed data access layer that supports SQL Server connection pooling. This can significantly decrease memory consumption by Microsoft Dynamics NAV Server. SQL Server connection pooling also simplifies deployment of the Microsoft Dynamics NAV three-tier architecture for deployments where the three tiers are installed on separate computers. Specifically, administrators are no longer required to manually create SPNs or to set up delegation when the client, Microsoft Dynamics NAV Server, and SQL Server are on separate computers.

Decreased Resource Consumption

There is no longer a one-to-one correlation between the number of client connections and the number of SQL Server connections. In earlier versions of Microsoft Dynamics NAV, each SQL Server connection could consume up to 40 MB of memory. Memory allocation is now in managed memory. This is generally more efficient than unmanaged memory.

In Microsoft Dynamics NAV 2013, all users who are connected to the same Microsoft Dynamics NAV Server instance share the data cache. This means that after one user has read a record, a second user who reads the same record retrieves it from the cache. In earlier versions of Microsoft Dynamics NAV, the data cache was isolated for each user.

Caching

Microsoft Dynamics NAV 2013 uses an improved cache system. The following functions use the cache system:

- GET
- FIND
- FINDFIRST
- FINDLAST
- FINDSET
- COUNT
- ISEMPTY
- CALCFIELDS
- CALCSUMS

Module 13: SQL Server Optimization

There are two types of caches:

- Global cache – For all users who are connected to a Microsoft Dynamics NAV Server instance.
- Private cache – For each user, for each company, in a transactional scope. Data in a private cache for a given table and company are flushed when a transaction ends.

The lock state of a table determines the cache to use. If a table is not locked, then the global cache is queried for data. Otherwise, the private cache is queried.

Results from query objects are not cached.

For a call to any of the FIND functions, 1024 rows are cached. You can set the size of the cache by using the Data Cache Size setting in the Microsoft Dynamics NAV Server configuration file. The default size is 9. This approximates a cache size of 500 MB. If you increase this number by one, then the cache size doubles.

You can bypass the cache by using the **SELECTLATESTVERSION** Function.

Microsoft Dynamics NAV 2013 synchronizes caching between Microsoft Dynamics NAV Server instances that are connected to the same database. By default, synchronization occurs every 30 seconds.

You can set the cache synchronization interval by using the CacheSynchronizationPeriod parameter in the CustomSettings.config file.

Improved Performance

Microsoft Dynamics NAV 2013 no longer uses server cursors to retrieve records. Instead, you retrieve records by using multiple active result sets (MARS). Functions such as **Next**, **Find('')**, **Find('+')**, **Find('>')**, and **Find('<')** are generally faster with MARS.

 **Note:** Because Microsoft Dynamics NAV 2013 no longer uses server cursors to retrieve records, the Record Set property under Caching on the **Advanced** tab of the **Alter Database** page was no longer needed and was removed.

SIFT indexes also are improved. For example, COUNT and AVERAGE formulas can now use SIFT indexes. MIN and MAX formulas now use SQL Server MIN and MAX functions exclusively.

RecordId's and SQL Variant columns in a table no longer prevent use of BULK insert inserts.

In most cases, filtering on FlowFields issues a single SQL statement. In earlier versions of Microsoft Dynamics NAV, filtering on FlowFields issued an SQL statement for each filtered FlowField and for each record in the table to calculate the filtered FlowFields. The following exceptions are in Microsoft Dynamics NAV 2013 in which filtering on FlowFields does not issue a single SQL statement:

- You use the ValuesFilter option on a field and the field has a value.
- A second predicate is specified on a source field and the field that is used for the second predicate has a value. For example, when you specify the CalcFormula Property for a FlowField, you can specify table filters in the **Calculation Formula** window. If you specify two or more filters on the same source field, then filtering does not issue a single SQL statement.
- You specify Validated for the SecurityFiltering Property on a record. This value for the SecurityFiltering property means that each record that is part of the calculation must be verified for inclusion in the security filter.

In most cases, calling the **FIND** or **NEXT** functions after you set the view to include only marked records issues a single SQL statement. In earlier versions of Microsoft Dynamics NAV, calling **FIND** or **NEXT** functions that have marked records issued an SQL statement for each mark. There are some exceptions if many records are marked.

C/AL Database Functions and Performance on SQL Server

GET, FIND, and NEXT

The C/AL language offers several methods to retrieve record data. Each of the following functions is optimized for a specific purpose. To achieve optimal performance you must use the method that is best suited for a given purpose.

- **Record.GET** – This function is optimized for retrieving a single record based on primary key values.
- **Record.FIND** – The **FIND** function is optimized for retrieving a single record based on the primary keys in the record and any filter or range that was set.
- **Record.FIND('-')** and **Record.FIND('+')** – These functions are optimized for reading an open-end dataset when the application might not read all records.
- **Record.FINDSET(ForUpdate, UpdateKey)** – The **FINDSET** function is optimized for reading the whole set of records within the specified filter and range. The **UpdateKey** parameter does not influence the efficiency of this method in Microsoft Dynamics NAV 2013.

- **Record.FINDFIRST and Record.FINDLAST** – The **FINDFIRST** and **FINDLAST** functions are optimized for finding the single first or last record within the specified filter and range. Use these functions when you only require one record.
- **Record.FINDFIRST and Record.FINDLAST** – The **FINDFIRST** and **FINDLAST** functions are optimized for finding the single first or last record within the specified filter and range. Use these functions when you only require one record.
- **Record.FINDFIRST and Record.FINDLAST** – The **FINDFIRST** and **FINDLAST** functions are optimized for finding the single first or last record within the specified filter and range. Use these functions when you only require one record.
- **Record.FINDFIRST and Record.FINDLAST** – The **FINDFIRST** and **FINDLAST** functions are optimized for finding the single first or last record within the specified filter and range. Use these functions when you only require one record.
- **Record.NEXT** – The **NEXT** function can be called at any time. However, if **Record.NEXT** is not called as part of retrieving a continuous result set, then Microsoft Dynamics NAV calls a separate SQL statement in order to find the next record.

Dynamic Result Sets

In Microsoft Dynamics NAV any result set that is returned from a call to the find methods discussed in the previous section is dynamic. This means that the result set is guaranteed to contain any changes that you make later in the result set. However this feature comes at a cost. If any modifications are made to a table that is being traversed, then Microsoft Dynamics NAV might have to issue an additional SQL statement to guarantee that the result set is dynamic.

The following code shows how records are most efficiently retrieved. FINDSET is the most efficient method to use because it reads all records.

Code Example

```
IF FINDSET THEN  
  
REPEAT  
  
// Insert statements to repeat.  
  
UNTIL NEXT = 0;
```

CALCFIELDS, CALCSUMS, and COUNT Functions

Each call to **CALCFIELDS**, **CALCFIELD**, **CALCSUMS**, or **CALCSUM** functions that calculates a sum requires a separate SQL statement. This is true unless the client calculated the same sum or another sum that uses the same SumIndexFields or filters in a recent operation.

Each **CALCFIELDS** or **CALCSUMS** function request should be confined to use only one SIFT index. You can use the SIFT index only as follows:

- All requested sum-fields are contained in the same SIFT index.
- The filtered fields are part of the key fields that are specified in the SIFT index that contains all the sum fields.

If neither of these requirements is fulfilled, then the sum is calculated directly from the base table.

In Microsoft Dynamics NAV 2013, use SIFT indexes to count records in a filter, as long as a SIFT index exists that contains all filtered fields in the key fields that are defined for the SIFT index.

SETAUTOCALCFIELDS

It is a common task to retrieve data and request calculation of associated FlowFields. The following example traverses customer records, calculates the balance, and marks the customer as blocked if the customer exceeds the maximum credit limit.

Code Example

```
IF Customer.FINDSET() THEN REPEAT  
  
    Customer.CALCFIELDSS(Customer.Balance)  
  
    IF (Customer.Balance > MaxCreditLimit) THEN BEGIN  
  
        Customer(Blocked) = True;  
  
        Customer.MODIFY();  
  
    END  
  
    ELSE IF (Customer.Balance > LargeCredit) THEN BEGIN  
  
        Customer(Caution) = True;  
  
        Customer.MODIFY();  
  
    END  
END
```

```
END;  
UNTIL Customer.NEXT = 0;
```

In Microsoft Dynamics NAV 2013, you can do this much faster. First, set a filter on the customer. You can also do this in Microsoft Dynamics NAV 2009, but behind the scenes the same code is executed. In Microsoft Dynamics NAV 2013, setting a filter on a record translates into a single SQL statement.

Code Example

```
Customer.SETFILTER(Customer.Balance,'>%1', LargeCredit);  
  
IF Customer.FINDSET() THEN REPEAT  
  
    Customer.CALCFIELDS(Customer.Balance)  
  
    IF (Customer.Balance > MaxCreditLimit) THEN BEGIN  
  
        Customer.Blocked = True;  
  
        Customer.MODIFY();  
  
    END  
  
    ELSE IF (Customer.Balance > LargeCredit) THEN BEGIN  
  
        Customer.Caution = True;  
  
        Customer.MODIFY();  
  
    END;  
  
UNTIL Customer.NEXT = 0;
```

In the previous example, an additional call to the CALCFIELDS function still must be issued for the code to check the value of Customer.Balance. You can optimize this more by using the new **SETAUTOCALCFIELDS** function.

Code Example

```
Customer.SETFILTER(Customer.Balance,'>%1', LargeCredit);  
  
Customer.SETAUTOCALCFIELDS(Customer.Balance)  
  
IF Customer.FINDSET() THEN REPEAT  
  
    IF (Customer.Balance > MaxCreditLimit) THEN BEGIN
```

```
Customer.Blocked = True;  
  
Customer.MODIFY();  
  
END  
  
ELSE IF (Customer.Balance > LargeCredit) THEN BEGIN  
  
Customer.Caution = True;  
  
Customer.MODIFY();  
  
END;  
  
UNTIL Customer.NEXT = 0;
```

INSERT, MODIFY, DELETE, and LOCKTABLE

Each call to **INSERT**, **MODIFY**, or **DELETE** functions requires a separate SQL statement. If the table that you modify contains SumIndexes, then the operations are significantly slower. As a test, select a table that contains SumIndexes and execute one hundred **INSERT**, **MODIFY**, or **DELETE** operations to measure how long it takes to maintain the table and all its SumIndexes.

The **LOCKTABLE** function does not require any separate SQL statements. It only causes any successive reading from the table to lock the table or parts of it.

Bulk Inserts

Microsoft Dynamics NAV automatically buffers inserts to send them to Microsoft SQL Server at one time.

By using bulk inserts, the number of server calls is reduced. This improves performance.

Bulk inserts also improve scalability by delaying the actual insert until the last possible moment in the transaction. This reduces the time that tables are locked, especially tables that contain SIFT indexes.

Software developers who want to write high performance code that uses this feature should understand the following bulk insert constraints.

Bulk Insert Constraints

If you want to write code that uses the bulk insert functionality, you must be aware of the following constraints.

Module 13: SQL Server Optimization

Records are sent to SQL Server when the following occurs:

- You call COMMIT to commit the transaction.
- You call MODIFY or DELETE on the table.
- You call any FIND or CALC methods on the table.
- Records are not buffered if one of the following conditions is TRUE:
 - The application is using the return value from an INSERT call, for example, "IF (GLEntry.INSERT) THEN".
 - The table that you insert the records into contains any of the following:
 - BLOB fields
 - Fields that have the AutoIncrement property set to Yes

The following code example cannot use buffered inserts because it contains a FIND call on the **GL/Entry** table within the loop.

Code Example

```
IF (JnlLine.FIND('-')) THEN BEGIN  
  
    GLEntry.LOCKTABLE;  
  
    REPEAT  
  
        IF (GLEntry.FINDLAST) THEN  
  
            GLEntry.NEXT := GLEntry."Entry No." + 1  
  
        ELSE  
  
            GLEntry.NEXT := 1;  
  
        // The FIND call will flush the buffered records.  
  
        GLEntry."Entry No." := GLEntry.NEXT ;  
  
        GLEntry.INSERT;  
  
    UNTIL (JnlLine.FIND('>') = 0)  
  
END;  
  
COMMIT;
```

If you rewrite the code as shown in the following example, you can use buffered inserts.

Code Example

```
IF (JnlLine.FIND('-')) THEN BEGIN  
  
    GLEntry.LOCKTABLE;  
  
    IF (GLEntry.FINDLAST) THEN  
  
        GLEntry.Next := GLEntry."Entry No." + 1  
  
    ELSE  
  
        GLEntry.Next := 1;  
  
    REPEAT  
  
        GLEntry."Entry No.":= GLEntry.Next;  
  
        GLEntry.Next := GLEntry."Entry No." + 1;  
  
        GLEntry.INSERT;  
  
        UNTIL (JnlLine.FIND('>') = 0)  
  
    END;  
  
    COMMIT;  
  
// The inserts are performed here.
```

Locking, Blocking, and Deadlocks

When data is read from the database, Microsoft Dynamics NAV uses the READUNCOMMITTED isolation level. This means that any user can modify the records that are currently being read, until the table is either changed by a write operation, or locked with the **Record.LOCKTABLE** function. From this point until the end of the transaction, all read operations on the table are performed with both REPEATABLE READ and UPDLOCK locking. This is frequently known as *pessimistic concurrency*.

Locking

Records can be read with different types of locks such as UPDLOCK. At this level, records that are read are locked. This means that no other user can modify the record.

Module 13: SQL Server Optimization

An example of a lock of a customer record can be demonstrated by the following code.

Code Example

```
Customer.LOCKTABLE;  
  
Customer.GET('10000');      // Customer 10000 is locked  
  
Customer.Blocked := Customer.Blocked::All;  
  
Customer.MODIFY;  
  
COMMIT;                   // Lock is removed
```

If the record is not locked, the following situation can occur:

User A	User B	Comment
Customer.GET('10000') ;		User A reads record without any lock.
	Customer.GET('10000') ;	User B reads same record without any lock.
...	Customer.Blocked := Customer.Blocked::All ; Customer.MODIFY; COMMIT;	User B modifies record.
Customer.Blocked := Customer.Blocked::""; Customer.MODIFY;		User A gets an error: "Another user has modified the record...".
ERROR	SUCCESS	

Blocking

When other users try to lock data that is currently locked, they are blocked and have to wait. If they wait longer than the defined time out, they receive the following Microsoft Dynamics NAV error: "The XYZ table cannot be locked or changed because it is already locked by the user."

If it is necessary, change the default time-out by selecting **File > Database > Alter**. On the **Advanced** tab, select **Lock Timeout** and **Timeout duration (sec)** value.

Refer to the previous example, where two users try to modify the same record. The data that will be changed can be locked. This prevents other users from doing the same. Following is an example.

User A	User B	Comment
Customer.LOCKTABLE; Customer.GET('10000') ;		User A reads record with lock.
	Customer.LOCKTABLE; Customer.GET('10000');	User B tries to read same record with a lock.
...	... blocked, waiting ...	User B waits and is blocked, because the record is locked by user A.
Customer.Blocked := Customer.Blocked::All ; Customer.MODIFY;	... blocked, waiting ...	User A successfully modifies record.
COMMIT;		Lock is released.
	...	Data is sent to user B.
	Customer.Blocked := Customer.Blocked::""; Customer.MODIFY;	User B successfully modifies record.
	COMMIT;	Lock is released.
SUCCESS	SUCCESS	

Deadlocks

There is a potential situation when blocking cannot be resolved by SQL server. The situation arises when two or more users manage to lock data. Then it is blocked when they try to lock data that is already locked by one of the other users. SQL server resolves the issue by ending the transaction that has done the least amount of work.

For example, consider a case in which two users are working at the same time and try to retrieve one another's blocked records, as shown in the following pseudo code.

Module 13: SQL Server Optimization

User A	User B	Comment
TableX.LOCKTABLE; TableY.LOCKTABLE;	TableX.LOCKTABLE; TableY.LOCKTABLE;	Indicates that the next read will use UPDLOCK.
TableX.FINDFIRST;	TableY.FINDFIRST;	A blocks Record1 from TableX. B blocks Record 1 from tableY.
...	...	

User A	User B	Comment
TableY.FINDFIRST;	TableX.FINDFIRST;	A wants B's record, whereas B wants A's record. A conflict occurs.
"Your activity was deadlocked with another user"		SQL Server detects deadlock and arbitrarily selects one over the other. Therefore, one will receive an error.
ERROR	SUCCESS	

SQL Server supports record level locking. So, there may be a situation where these two activities bypass one another without any problem, such as with this pseudo code. Be aware that User A is retrieving the last record compared to the situation that was discussed earlier.

User A	User B	Comment
TableX.LOCKTABLE; TableY.LOCKTABLE;	TableX.LOCKTABLE; TableY.LOCKTABLE;	Indicates that the next read will use UPDLOCK.
TableX.FINDFIRST;	TableY.FINDFIRST;	A blocks Record1 from TableX. B blocks Record 1 from tableY.
...	...	
TableY.FINDLAST;	TableX.FINDLAST;	No conflict, as no records are in contention.
SUCCESS	SUCCESS	

Be aware that there would be a deadlock if one of the tables was empty, or contained one record only. To add to this complexity, there may be a situation where two processes read the same table from opposite directions and meet in the middle, such as with the following pseudo code.

User A	User B	Comment
TableX.LOCKTABLE; TableY.LOCKTABLE;	TableX.LOCKTABLE; TableY.LOCKTABLE;	Indicates that the next read will use UPDLOCK.
TableX.FIND('>');	TableY.FIND('+'');	A reads from top of TableX. B reads from bottom of TableX.

User A	User B	Comment
REPEAT	REPEAT	
...	...	
UNTIL NEXT = 0;	UNTIL NEXT(-1) = 0;	...after some time... A wants B's record, whereas B wants A's record. A conflict occurs.
	"Your activity was deadlocked with another user"	SQL Server detects deadlock and selects one of the users for failure.
SUCCESS	ERROR	

There are also situations where a block on index update may produce the conflict, and situations where updating SIFT tables can cause a deadlock. These situations can be complex and difficult to avoid. However, the transaction that is selected to fail is rolled back to the beginning, so there should be no major issue. However, if the process is written with several partial commits, then there might be dirty data in the database as a side-product of those deadlocks. That can become a major issue for the customer.

Avoid Deadlocks

Many deadlocks could lead to major customer dissatisfaction, but deadlocks cannot be avoided completely. To reduce the number of deadlocks, do the following:

- Process tables in the same sequence.
- Process records in the same order.
- Keep the transaction length to a minimum.

Module 13: SQL Server Optimization

If this is not possible because of the complexity of the processes, revert to serializing the code by making sure that conflicting processes cannot execute in parallel. The following code demonstrates how you can do this.

User A	User B	Comment
TableX.LOCKTABLE; TableY.LOCKTABLE; TableA.LOCKTABLE;	TableX.LOCKTABLE; TableY.LOCKTABLE; TableA.LOCKTABLE;	Indicates that the next read will use UPDLOCK.
TableA.FINDFIRST;	...	User A locks Record1 from TableX.

User A	User B	Comment
	TableA.FINDFIRST;	User B tries to lock Record1 from TableX.
...	Blocked	User B is blocked.
TableY.FINDFIRST; TableX.FINDFIRST;	Blocked	User A processes tables in opposite order.
COMMIT;		Block is released.
	OK on read table A	
	TableX.FINDFIRST; TableY.FINDFIRST;	User B processes tables in opposite order.
	COMMIT;	
SUCCESS	SUCCESS	

By serializing the transactions, you may experience a greater probability of timeouts. Therefore, keeping the length of transactions short becomes even more important. This also demonstrates that you can combine the various principles and methods, depending on the situation and complexity; one method may work for one customer while the other method works for another customer.

We also recommend that you consider following these *golden rules*:

- Test conditions of data validity before the start of locking.
- Allow for some time gap between heavy processes so that other users can process.
- Never allow for user input during an opened transaction.

If the transaction is too complex or if there is limited time, consider discussing the possibility of over-night processing of heavy jobs with the customer. This avoids the daily concurrency complexity and avoids the high costs of rewriting the code.

 **Best Practice:** *In order to minimize the possibility of deadlocks (or blocks) occurring, focus first on increasing the performance of the transactions. The longer it takes for a transaction to complete, the greater the possibility for a deadlock to occur. By increasing the performance, you reduce the possibility of deadlocks occurring. Increasing performance is achieved by analyzing index usage. Make sure that indexes that are not used are not maintained. Make sure that indexes that improve the performance are available for SQL Server to use and make sure that the available indexes are optimized.*

SIFT Data Storage in SQL Server

Use SIFT tables in Microsoft Dynamics NAV version 5.0 and older, to implement SIFT on SQL Server, and to store aggregate values for SumIndexFields for keys in the source tables. Starting with version 5.0 Service Pack 1, indexed views replace these SIFT tables. SIFT tables are no longer part of Microsoft Dynamics NAV. This section is preserved because developers who work with Microsoft Dynamics NAV are likely to encounter issues about SIFT tables in implementations of older versions of Microsoft Dynamics NAV. You must have a good understanding of how they work.

A SumIndexField is always associated with a key, and each key can have no more than 20 SumIndexFields associated with it. When the MaintainSIFTIndex property of a key is set to **Yes**, Microsoft Dynamics NAV regards this key as a SIFT key and creates the SIFT structures that are needed to support it.

You can associate any field of the Decimal data type with a key as a SumIndexField. Microsoft Dynamics NAV then creates and maintains a structure that stores the calculated totals that are required for the fast calculation of aggregated totals.

In the SQL Server Option for Microsoft Dynamics NAV, this maintained structure is a typical table, but is called a SIFT table. These SIFT tables exist on SQL Server, but are not visible in the table designer in C/SIDE. As soon as you create the first SIFT table for a base table, a dedicated SQL Server trigger is also created and then is maintained automatically by Microsoft Dynamics NAV. This is known as a SIFT trigger. A base table is also a standard Microsoft Dynamics NAV table, instead of an additional SQL Server table that is created to support Microsoft Dynamics NAV functionality.

Module 13: SQL Server Optimization

One SIFT trigger is created for each base table that contains SumIndexFields. This dedicated SQL Server trigger supports all the SIFT tables that you create to support this base table. The SIFT trigger implements all modifications that are made on the base table when a SIFT table is affected. This means that the SIFT trigger automatically updates the information in all existing SIFT tables after every modification of the records in the base table.

The name of the SIFT trigger has the following format: <base Table Name>_TG. For example, the SIFT trigger for table 17, G/L Entry is named CRONUS International Ltd_\$G/L Entry_TG. Regardless of the number of SIFT keys that are defined for a base table, only one SIFT trigger is created.

You create a SIFT table for every base table key that has at least one SumIndexField associated with it. Regardless of how many SumIndexFields are associated with a key, you can create only one SIFT table for that key.

The name of the SIFT table has the following format: <Company Name>\$<base Table ID>\$<Key Index>. For example, one of the SIFT tables that were created for table 17, G/L Entry is named **CRONUS International Ltd_\$17\$0**.

The column layout of the SIFT tables is based on the layout of the SIFT key together with the SumIndexFields that are associated with this SIFT key. But the first column in every SIFT table is always named *bucket*, and it contains the value of the bucket or the SIFT level for the precalculated sums that are stored in the table. To view the structure, examine the SIFTLevels property for a key in Microsoft Dynamics NAV.

After the bucket column is a set of columns with names that begin with the letter *f*. These are also known as f- or key-columns. Each of these columns represents one field of the SIFT key.

The name of these columns has the format, f<Field No.>, where Field No. is the integer value of the Field No. property of the represented SIFT key field. For example, column f3 in CRONUS International Ltd_\$17\$0 represents the **G/L Account No.** field (it is field number three in the base table G/L Entry).

Finally, there is a group of columns with names that begin with the letter *s* followed by numbers. These are known as s-columns. These columns represent every SumIndexField that is associated with the SIFT key.

The name of these columns has the format, s<Field No.>. Field No. is the integer value of the Field No. property of the represented SumIndexField. The precalculated totals of values for the corresponding SumIndexFields are stored in these fields of the SIFT table.

SIFT tables are one of the biggest Microsoft Dynamics NAV performance problems on SQL Server. One record update in the base table produces a potentially large stream of Input/Output (I/O) requests with updates to the records in the SIFT tables. This could possibly block other users during that time.

SQL Server Profiler

Microsoft SQL Profiler is a graphical user interface to SQL Trace for monitoring an instance of the Database Engine. You can capture and save data about each event to a file or table to analyze later. For example, you can monitor a production environment to see which queries are affecting performance by executing too slowly.

SQL Server Profiler

SQL Server Profiler can be used to monitor events that occur on SQL Server. It can be used to do the following tasks:

- Create a trace that is based on a reusable template.
- Watch the trace results as the trace runs.
- Store the trace results in a table.
- Start, stop, pause, and modify the trace results as necessary.
- Replay the trace results.

SQL Profiler then can analyze or use the trace file to troubleshoot logic or performance problems. You can use this utility to monitor several areas of server activity, such as the following:

- Analyzing and debugging SQL statements and stored procedures
- Monitoring slow performance
- Stress analysis
- General debugging and troubleshooting
- Fine-tuning indexes
- Auditing and reviewing security activity

To summarize, you create a template or use an existing template that defines the data that you want to collect. Then you collect the data by running a trace on the events that you defined in your template. During the run, Profiler displays the event classes and data columns that describe the event data that is being collected.

SQL Server Profiler Terminology

Template

A template defines the default configuration for a trace. Templates can be saved, imported, and exported between SQL Server instances. Templates from one SQL Server version cannot be imported to a different SQL Server version. SQL Server includes the following ten predefined templates:

- **Event** – An event is an action that is generated by the SQL Server engine, such as a logon connection or the execution of a Transact-SQL statement. Events are grouped by event categories. All the data that is generated by an event is displayed in the trace. This contains columns of data that describe the event in detail.
- **Trace** – The trace does the actual data capture, based on the events that you defined in the template.
- **Event Class** – An event class can be defined as a type of event that can be traced. Examples of event classes are SP:Starting and RPC:Completed.
- **Event Category** – Groups of events are called an *event category*. Examples of event categories are Stored Procedure and Locks. There can be multiple event categories that can be selected for a single trace.
- **Data Column** – Data column is an attribute of an event class that is captured in the trace. A data column contains values of an event class.
- **Filter** – Filters are used to create selectivity in data that are collected in trace. By default, SQL Profiler monitors all events on SQL Server. You can apply a filter to only monitor events in the Microsoft Dynamics NAV database.

Use SQL Server Profiler

SQL Profiler is a component of client tools that can be installed independently from the SQL Server Database Engine. It is mandatory for the user to have system admin rights to start the profiler. You can start SQL Profiler by using the following methods:

- **Start SQL Profiler** – SQL Profiler is available from the Microsoft Windows Start menu or SQL Server Enterprise Manager. Use either of the following methods to start Profiler:
 - Click Start, locate Microsoft SQL Server among your available programs, and then click Profiler on the Performance Tools group.

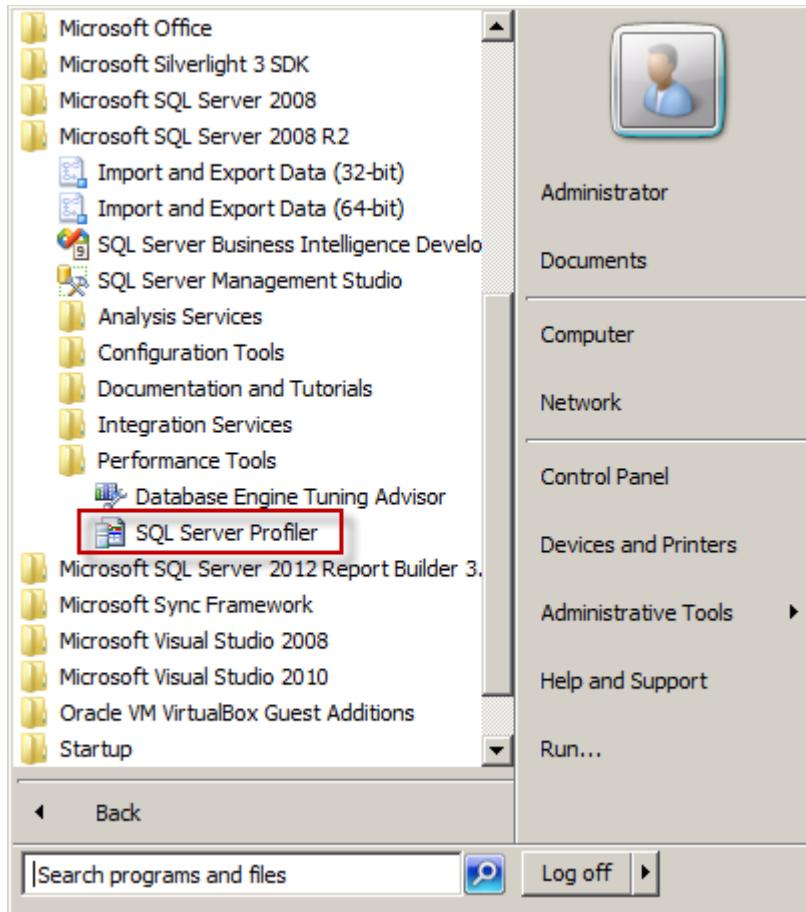


FIGURE 13.4: START SQL SERVER PROFILER FROM START MENU

- In Enterprise Manager, select **SQL Profiler** on the **Tools** menu.

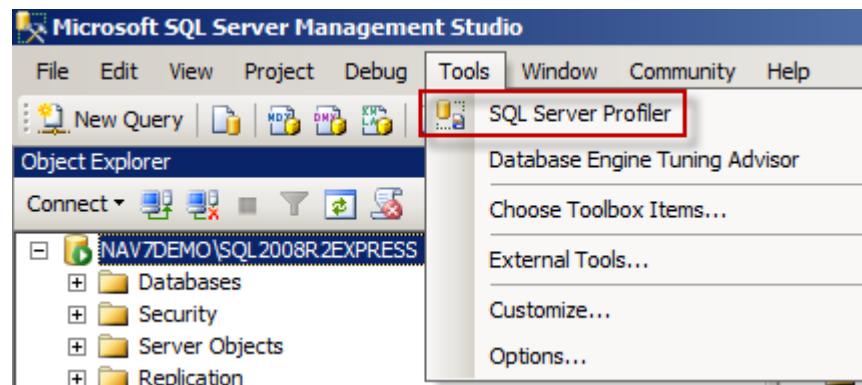


FIGURE 13.5: START SQL SERVER PROFILER FROM SQL SERVER MANAGEMENT STUDIO

- **Collect Data** – Select **Menu > File > New Trace** to create a new trace. A window opens to connect it to a database. The database can be a local database or a database that is available on the network, such as a production server. In the following example a connection is made to a SQL Instance that is named **SQL2008R2EXPRESS** on the server **NAV7DEMO**.



FIGURE 13.6: CONNECT TO SERVER WINDOW



Note: NoteBe aware that you must be a member of the SYSADMIN fixed server role to be able to setup traces with Profiler.

The **Trace Properties** window opens after the connection to SQL Server is made. Here you can enter the trace name. Notice that trace provider name, type, and versions are prepopulated and you cannot alter them. These are set based on the instance of SQL Server with which you are connected.

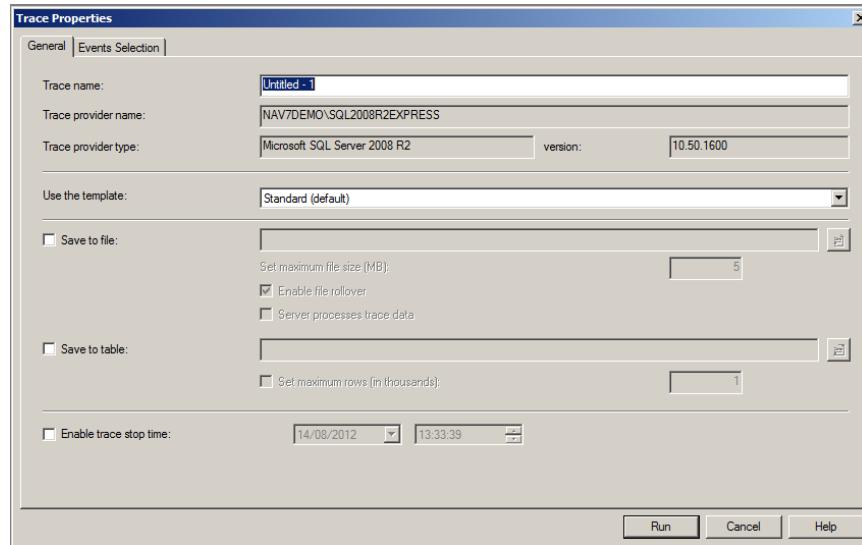


FIGURE 13.7: TRACE PROPERTIES GENERAL WINDOW

The **Trace Properties** window contains a drop-down for template selection. This is named **Use the template**. The selected default template is standard. To view the events that are included in the **Standard** template, select the **Events Selection** tab.

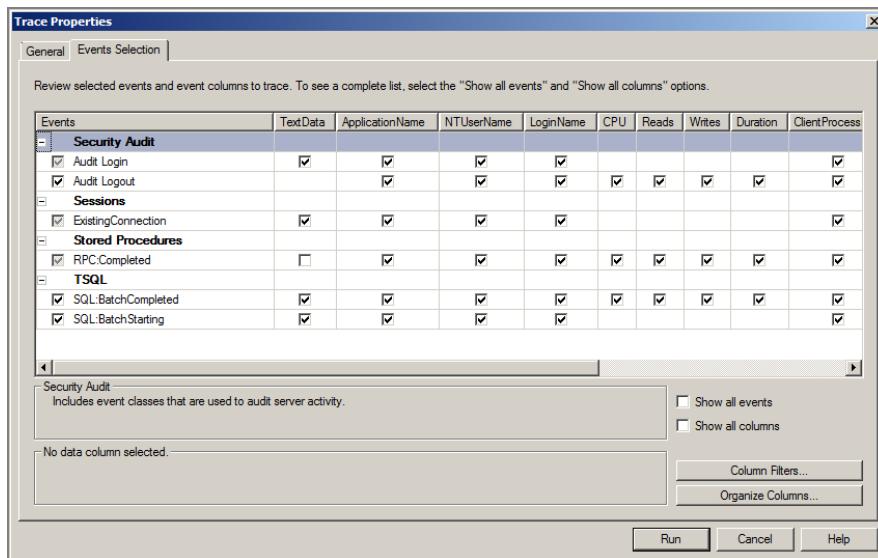


FIGURE 13.8: TRACE PROPERTIES EVENTS SELECTION

The **Event selection** window contains check boxes that can be configured, depending on the requirements of the data that you want to collect. There is also a **Column Filters** button that you can click to further filter trace data.

The **Trace Properties** window contains a **Run** button to start the trace. The trace can be paused or stopped as required by using the Play menu.

Module 13: SQL Server Optimization

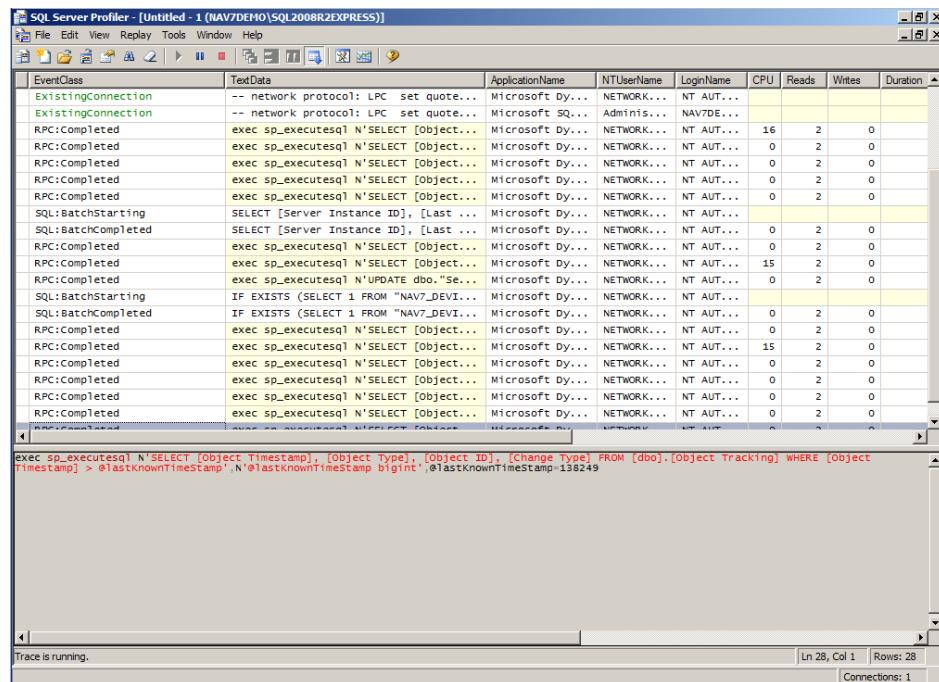


FIGURE 13.9: A RUNNING TRACE



Additional Reading: More information about SQL Profiler is available on the MSDN website that is located at: <http://go.microsoft.com/fwlink/?LinkId=269809>.

Lab 13.1: Analyze Index Usage

Scenario

Mort is asked to analyze the indexes of the Cronus database. To optimize the performance of certain processes, he must make sure that indexes that are not used by SQL Server are disabled to minimize overhead. To perform this analysis, Mort executes a query. This query displays a list of all tables and indexes in an SQL database to help identify the tables in which most blocks occur. This gives a starting point for index tuning.

Objectives

Identify indexes that might cause overhead on SQL Server.

Exercise 1: Use the Index Information Query to identify and disable unused indexes.

Task 1: Execute the Index Information Query

High Level Steps

1. Open SQL Server Management Studio.
2. Execute the Query.
3. Run the Index Information Query.
4. Analyze the Query Results.

Detailed Steps

1. Open SQL Server Management Studio.
 - a. To open Microsoft SQL Server Management Studio select: **Start > All Programs > Microsoft SQL Server 2012 > SQL Server Management Studio** on the main Toolbar. The **Connect to SQL Server** window opens.
 - b. Select **Connect** to connect to SQL Server.
 - c. To open a new Query window, select the **New Query** button (or **Ctrl+N**). It can be found at the top of the screen, below the menu.
2. Execute the Query.
 - a. Type the following Query in the Query window.

Code Example

```
use [Demo Database NAV (7-0)]  
  
IF OBJECT_ID ('z_IUQ2_Temp_Index_Keys', 'U') IS NOT NULL
```

```
DROP TABLE z_IUQ2_Temp_Index_Keys;

-- Generate list of indexes with key list

create table z_IUQ2_Temp_Index_Keys

([I1] [bigint] NOT NULL,

[F_Obj_ID] [bigint] NOT NULL,

[F_Schema_Name] [nvarchar] (128) NULL,

[F_Table_Name] [nvarchar] (128) NOT NULL,

[F_Row_Count] [bigint] NULL,

[F_Reserved] [bigint] NULL,

[F_Data] [bigint] NULL,

[F_Index_Size] [bigint] NULL,

[F_UnUsed] [bigint] NULL,

[F_Index_Name] [nvarchar] (128) NULL,

[F_Index_ID] [bigint] NOT NULL,

[F_Column_Name] [nvarchar] (128) NULL,

[F_User_Updates] [bigint] NULL,

[F_User_Reads] [bigint] NULL,

[F_Locks] [bigint] NULL,

[F_Blocks] [bigint] NULL,

[F_Block_Wait_Time] [bigint] NULL,

[F_Last_Used] [datetime] NULL,

[F_Index_Type] [nvarchar] (128) NOT NULL,

[F_Index_Column_ID] [bigint] NOT NULL,

[F_Last_Seek] [datetime] NULL,

[F_Last_Scan] [datetime] NULL,
```

```
[F_Last_Lookup] [datetime] NULL,  
[Index_Key_List] [nvarchar] (MAX) NULL  
)  
GO  
CREATE NONCLUSTERED INDEX [Object_ID_Index] ON [dbo].  
[z_IUQ2_Temp_Index_Keys]  
(  
[F_Obj_ID]  
ASC  
)  
GO  
CREATE NONCLUSTERED INDEX [Index_ID_Index] ON [dbo].  
[z_IUQ2_Temp_Index_Keys]  
(  
[F_Index_ID]  
ASC  
)  
GO  
CREATE NONCLUSTERED INDEX [RowCount_ID_Index] ON [dbo].  
[z_IUQ2_Temp_Index_Keys]  
(  
[F_Row_Count]  
ASC  
)  
GO
```

```
INSERT INTO z_IUQ2_Temp_Index_Keys

SELECT

(
    row_number() over(order by a3.name, a2.name))%2 as l1,
    a1.object_id,
    a3.name AS [schemaname],
    a2.name AS [tablename],
    a1.rows as row_count,
    (
        a1.reserved + ISNULL(a4.reserved,0))* 8 AS reserved,
        a1.data * 8 AS data,
        (
            CASE WHEN (a1.used + ISNULL(a4.used,0)) > a1.data THEN (a1.used +
            ISNULL(a4.used,0)) - a1.data ELSE 0 END) * 8 AS index_size,
        (
            CASE WHEN (a1.reserved + ISNULL(a4.reserved,0)) > a1.used THEN (a1.reserved +
            ISNULL(a4.reserved,0)) - a1.used ELSE 0 END) * 8 AS unused,
        -- Index Description
        SI.name,
        SI.Index_ID,
        index_col(object_name(SIC.object_id),SIC.index_id,SIC.Index_Column_ID),
        -- Index Stats
        US.user_updates,
        US.user_seeks + US.user_scans + US.user_lookups User_Reads,
        -- Index blocks
        IStats.row_lock_count + IStats.page_lock_count,
```

```
IStats.row_lock_wait_count + IStats.page_lock_wait_count,  
  
IStats.row_lock_wait_in_ms + IStats.page_lock_wait_in_ms,  
  
-- Dates  
  
CASE  
  
WHEN  
  
(ISNULL(US.last_user_seek,'00:00:00.000') >=  
ISNULL(US.last_user_scan,'00:00:00.000')) and  
  
(ISNULL(US.last_user_seek,'00:00:00.000') >=  
ISNULL(US.last_user_lookup,'00:00:00.000'))  
  
THEN  
  
US.last_user_seek  
  
WHEN  
  
(ISNULL(US.last_user_scan,'00:00:00.000') >=  
ISNULL(US.last_user_seek,'00:00:00.000')) and  
  
(ISNULL(US.last_user_scan,'00:00:00.000') >=  
ISNULL(US.last_user_lookup,'00:00:00.000'))  
  
THEN  
  
US.last_user_scan  
  
ELSE  
  
US.last_user_lookup  
  
END AS Last_Used_For_Reads,  
  
SI.type_desc,  
  
SIC.index_column_id,  
  
US.last_user_seek,  
  
US.last_user_scan,  
  
US.last_user_lookup,  
  
"
```

```
FROM
(
SELECT
ps.object_id,
SUM
(
CASE
WHEN
(ps.index_id < 2)
THEN
row_count
ELSE
0
END
)
AS [rows],
SUM(ps.reserved_page_count) AS reserved,
SUM
(
CASE
WHEN
(ps.index_id < 2)
THEN
(ps.in_row_data_page_count + ps.lob_used_page_count +
ps.row_overflow_used_page_count)
ELSE
```

```
(ps.lob_used_page_count + ps.row_overflow_used_page_count)

END

)

AS data,

SUM (ps.used_page_count) AS used

FROM

sys.dm_db_partition_stats ps

GROUP BY ps.object_id) AS a1

LEFT OUTER JOIN

(

SELECT

it.parent_id,

SUM (ps.reserved_page_count) AS reserved,

SUM (ps.used_page_count) AS used

FROM sys.dm_db_partition_stats ps

INNER JOIN sys.internal_tables it ON (it.object_id = ps.object_id)

WHERE it.internal_type IN (202,204)

GROUP BY it.parent_id) AS a4 ON (a4.parent_id = a1.object_id)

INNER JOIN sys.all_objects a2 ON ( a1.object_id = a2.object_id )

INNER JOIN sys.schemas a3 ON (a2.schema_id = a3.schema_id)

INNER JOIN sys.indexes SI ON (SI.object_id = a1."object_id")

INNER JOIN sys.index_columns SIC ON (SIC.object_id = SI.object_id and

SIC.index_id = SI.index_id)

LEFT OUTER JOIN sys.dm_db_index_usage_stats US ON (US.object_id = SI.object_id and

US.index_id = SI.index_id and US.database_id = db_id())

LEFT OUTER JOIN sys.dm_db_index_operational_stats(NULL,NULL,NULL,NULL)

IStats ON (IStats.object_id = SI.object_id and IStats.index_id = SI.index_id and
```

```
IStats.database_id = db_id()

WHERE a2.type <> N'S' and a2.type <> N'IT'

ORDER BY row_count DESC

GO

-- Populate key string

DECLARE IndexCursor CURSOR FOR SELECT

F_Obj_ID,

F_Index_ID

FROM

z_IUQ2_Temp_Index_Keys

FOR UPDATE OF

Index_Key_List

DECLARE @objID int

DECLARE @IndID int

DECLARE @KeyString VARCHAR(MAX)

SET @KeyString = NULL

OPEN IndexCursor

SET NOCOUNT ON

FETCH NEXT FROM IndexCursor INTO @ObjID, @IndID

WHILE @@fetch_status = 0 BEGIN

SET @KeyString = ""

SELECT @KeyString = COALESCE(@KeyString,"") + F_Column_Name + ','

FROM z_IUQ2_Temp_Index_Keys

WHERE F_Obj_ID = @ObjID and F_Index_ID = @IndID

ORDER BY F_Index_ID, F_Index_Column_ID
```

```
SET @KeyString = LEFT(@KeyString,LEN(@KeyString) -2)

UPDATE z_IUQ2_Temp_Index_Keys

SET Index_Key_List = @KeyString

WHERE CURRENT OF IndexCursor

FETCH NEXT FROM IndexCursor INTO @ObjID, @IndID

END;

CLOSE IndexCursor

DEALLOCATE IndexCursor

GO

-- clean up table to one line per index

DELETE FROM z_IUQ2_Temp_Index_Keys

WHERE [F_Index_Column_ID] > 1

GO

-- Select results

SELECT

[F_Table_Name] TableName,

[F_Row_Count] No_Of_Records,

[F_Data] Data_Size,

[F_Index_Size] Index_Size,

[F_Index_Name] Index_Name,

[F_User_Updates] Index_Updates,

[F_User_Reads] Index_Reads,

CASE WHEN

F_User_Reads = 0 THEN F_User_Updates

ELSE
```

```
F_User_Updates / F_User_Reads  
  
END AS Updates_Per_Read,  
  
[F_Locks] Locks,  
  
[F_Blocks] Blocks,  
  
[F_Block_Wait_Time] Block_Wait_Time,  
  
[F_Last_Used] Index_Last_Used,  
  
[F_Index_Type] Index_Type,  
  
[Index_Key_List] Index_Fields  
  
FROM z_IUQ2_Temp_Index_Keys  
  
--order by F_Row_Count desc, F_Table_Name, [F_Index_ID]  
  
--order by F_User_Updates desc  
  
--order by Blocks desc  
  
--order by Block_Wait_Time desc  
  
order by Updates_Per_Read desc  
  
--ORDER BY F_Table_Name
```

- b. As an alternative you can copy/paste the query into the Query Designer from the file IndexUsageQuery.sql.
3. Run the Index Information Query.
 - a. Select **Query**, and then **Execute** to run the query.
4. Analyze the Query Results.

In this exercise we presume the results are valid and representative for the workload in Microsoft Dynamics NAV.

- a. In the Query Results the following might be an example of an index that is unused, and that is causing overhead during write statements:
 - i. Table Name: **CRONUS International Ltd_\$Sales Header**
 - ii. Index Name: \$4
 - iii. Index Updates: 1246

 **Note:** This number might be different, because it depends on how the database was used. It will be different almost every time you run this query. The number does not need to be the same when you test it.

- iv. Index Reads: 0
- v. Index Fields: Document Type, Combine Shipments, Bill-to Customer No_, Currency Code, EU 3-Party Trade, No

According to the Index Information Query this index has never been used by SQL Server in a READ operation. However, it has already been updated multiple times. To disable the overhead of updating this index when there is a write operation in the Sales Header table, we will no longer maintain this index on SQL Server.

Task 2: Disable an unused Index

High Level Steps

1. Determine the key that correspond to index \$4 in the Sales Header table.
2. Disable the index on SQL Server.

Detailed Steps

1. Determine the key that correspond to index \$4 in the Sales Header table.
 - a. Indexes in SQL Server correspond to Keys in Microsoft Dynamics NAV. Index **\$4** in SQL Server corresponds to the fifth key in the Dynamics NAV Table designer for table **Sales Header**.
 - b. In the Microsoft Dynamics NAV Development Environment, open the Object Designer (if not already open).
 - c. Select Tables (on the left) and scroll to table 36 "Sales Header".
 - d. Click on the Design button.
 - e. Select **View, Keys**.
 - f. Select the fifth row and select **View, Properties**.
 - g. In SQL Server Management Studio, in the Object Explorer (on the left), expand the Databases folder.
 - h. In the Databases folder, expand "Demo Database NAV (7-0)" and then expand Tables.
 - i. In the Tables folder, scroll down to the line: "dbo.CRONUS International Ltd_\$Sales Header" and expand it.
 - j. Expand the Indexes folder.
 - k. Select the line that contains "\$4 (Unique, Non-Clustered)" then Right-Click and select **Properties**.

Module 13: SQL Server Optimization

- I. The Index Properties window opens for index \$4.
- m. Verify that you can open the index properties and compare the fields in the index with the fields in the key.

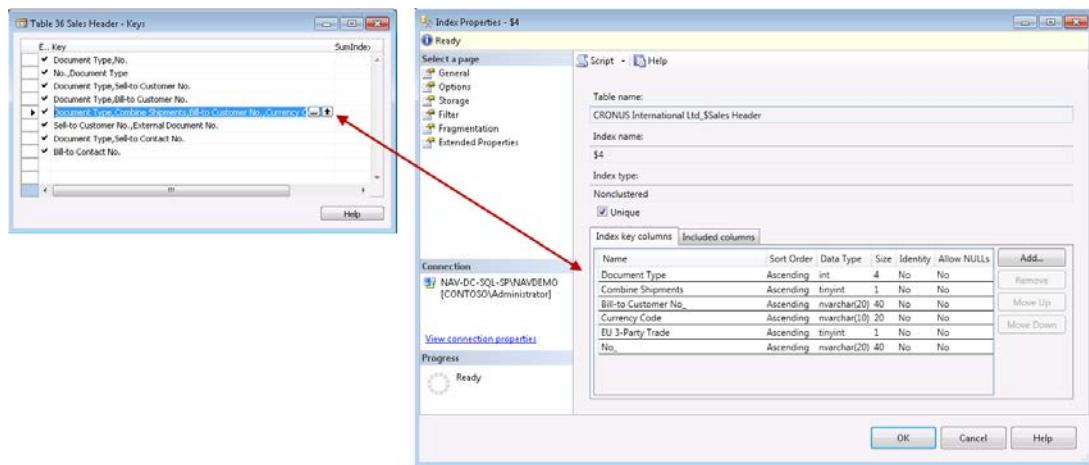


FIGURE 13.10: COMPARE KEY AND INDEX

2. Disable the index on SQL Server.
- a. So that you do not maintain the index in SQL Server, the property MaintainSQLIndex for the key has to be set to **No** as follows:

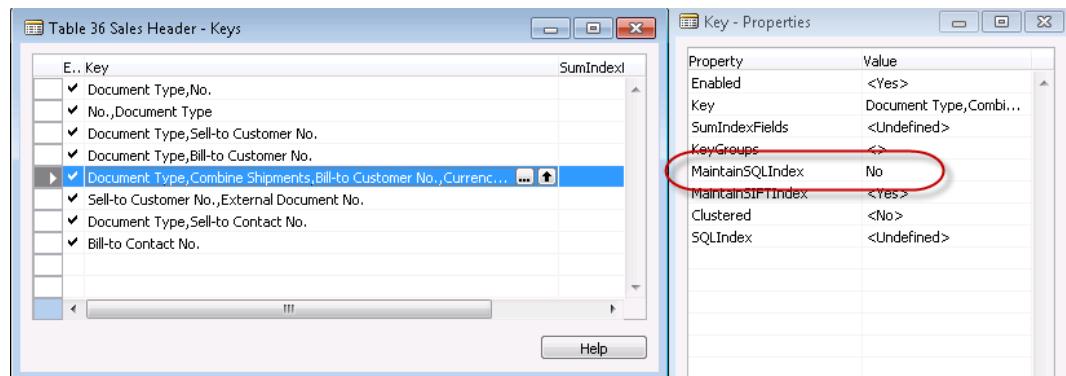


FIGURE 13.11: MAINTAINSQINDEX PROPERTY

- b. Close, and then save the table.

Lab 13.2: Optimize C/AL Code

Scenario

Mort is asked to improve the performance of a code unit.

Exercise 1: Analyze and improve the C/AL code and corresponding SQL statements

Task 1: Analyze the generated SQL Statements

High Level Steps

1. Import Codeunit 123456780.
2. Design CodeUnit 123456780.
3. Analyze the C/AL code of the Batch Job and make improvements where applicable.
4. Start SQL Profiler.
5. Start a new trace.
6. Execute the Batch Job.
7. Stop, and then save the trace.
8. Analyze the Trace.

Detailed Steps

1. Import Codeunit 123456780.
 - a. In the Microsoft Dynamics NAV Development Environment, open the Object Designer by selecting **Tools**, and then click **Object Designer**.
 - b. Select **File**, and then click **Import**.
 - c. Select the file CodeUnit 123456780.fob.
 - d. Click **Open**.
2. Design CodeUnit 123456780.
 - a. In the Object Designer, click **CodeUnit**.
 - b. Select **CodeUnit 123456780** in the list.
 - c. Click **Design** to open the C/AL Editor.
3. Analyze the C/AL code of the Batch Job and make improvements where applicable.
 - a. The following code contains the batch job that updates the **Name 2** field of the **Customer** table.

Code Example

```
Customer.SETRANGE(City,'London');

Customer.FINDFIRST;

REPEAT

    Customer."Name 2" := 'Updated' + FORMAT(CURRENTDATETIME);

    Customer.MODIFY;

UNTIL Customer.NEXT = 0;

MESSAGE('Ready!');
```

4. Start SQL Profiler.
 - a. Start the SQL Profiler by clicking **Start > All Programs > Microsoft SQL Server 2012 > Performance Tools > SQL Server Profiler.**
5. Start a new trace.
 - a. In the **SQL Server Profiler** window, start a new trace. In the **Tools** menu, click **File**, and then **New Trace**.
 - b. In the **Connect to Server** window, select **Connect**. The **Trace Properties** window opens.
 - c. In the **Use the template** field, select **Tuning**.
 - d. Click the **Events Selection** tab.
 - e. Select **Column Filters**. The **Edit Filter** window opens.
 - f. Select the field **DatabaseName**.
 - g. In the filter window, enter the Dynamics NAV database name in the **Like** option.
 - h. Select **OK**.
 - i. Click **Run** to start the trace.
6. Execute the Batch Job.
 - a. In the Object Designer, select the CodeUnit 123456780, and then click **Run**.
7. Stop, and then save the trace.
 - a. In the SQL Profiler window, click **File**, and then **Stop Trace**.
 - b. To save the trace file to the database for further analysis click: **File > Save as >Trace Table**.

- c. In the **Connect to Server** window, click **Connect** to connect to SQL Server.
 - d. In the **Destination Table** window, in the **Database** field select the Dynamics NAV database: "Demo Database NAV (7-0)".
 - e. In the **Destination Table** window, in the **Table** field enter "SQLProfilerTraceResultBefore".
 - f. Click Ok.
8. Analyze the Trace.
- a. To open Microsoft SQL Server Management Studio, click **Start > All Programs > Microsoft SQL Server 2012 > SQL Server Management Studio**. The **Connect to SQL Server** window opens.
 - b. Click **Connect** to connect to SQL Server.
 - c. To open a new Query window click **New Query**.
 - d. Enter the following Query to display the trace results.

Code Example

```
use [Demo Database NAV (7-0)]  
  
SELECT * FROM SQLProfilerTraceResultBefore  
  
WHERE TextData LIKE '%Customer%'
```

- e. Click **Query**, and then **Execute** to run the query.
- f. In the query result, you see multiple lines lines. There are several SELECT and multiple UPDATE statements.
 - i. Please ignore statements that don't begin with SELECT or UPDATE.
- g. In the beginning there's a SELECT statement that retrieves one customer record with a WITH(READUNCOMMITTED) at the end.
- h. The next SELECT statement on the Custmer table, retrieves the customer record(s) to be modified with a TOP X and uses a WITH(UPLOCK) statement.
- i. Optimize the C/AL Code so that only one SELECT statement is generated and executed.

Task 2: Optimize the C/AL Code

High Level Steps

1. In the C/AL code of the imported CodeUnit (123456780) add a **SETCURRENTKEY** function.
2. Change **FIND('')** to **FINDSET**.
3. Save, and then compile the changes to the CodeUnit.

Detailed Steps

1. In the C/AL code of the imported CodeUnit (123456780) add a **SETCURRENTKEY** function.
 - a. Add a **SETCURRENTKEY(City)** statement before executing the **FIND** function.
2. Change **FIND('')** to **FINDSET**.
 - a. Because the code is looping over **Customer** records, and inside the loop it is modifying the records, you should use a **FINDSET(TRUE,FALSE)** statement. This applies the correct isolation level to the records. This avoids creating a second **SELECT** statement to lock the records.
 - b. Also, use an IF statement to only run the update when there are customer records.
 - c. The code now resembles this:

```
Customer.SETCURRENTKEY(City);

Customer.SETRANGE(City,'London');

IF Customer.FINDSET(TRUE,FALSE) THEN

REPEAT

    Customer."Name 2" := 'Updated' + FORMAT(CURRENTDATETIME);

    Customer.MODIFY;

UNTIL Customer.NEXT = 0;

MESSAGE('Ready!');
```

3. Save, and then compile the changes to the CodeUnit.
 - a. Save the changes to the CodeUnit by selecting **File**, and then **Save**.
 - b. Close the CodeUnit.

Task 3: Analyze the generated SQL Statements after Optimization

High Level Steps

1. Start a new SQL Profiler Trace.
2. Execute the optimized CodeUnit.
3. Export the Trace results.
4. Analyze the Trace.

Detailed Steps

1. Start a new SQL Profiler Trace.
 - a. Start a new trace in SQL Profiler by using the same Trace Template and DatabaseName filter.
2. Execute the optimized CodeUnit.
 - a. In the Object Designer, select the CodeUnit 123456780 and click **Run**.
3. Export the Trace results.
 - a. Stop the Trace in SQL Profiler.
 - b. Export the Trace Results to a new table named **SQLProfilerTraceResultAfter**.
4. Analyze the Trace.
 - a. Open a new Query window by click **New Query**).
 - b. Enter the following query to display the trace results.

Code Example

```
use [Demo Database NAV (7-0)]  
  
SELECT * FROM SQLProfilerTraceResultAfter  
  
WHERE TextData LIKE '%Customer%'
```

- c. Click **Query** and then **Execute** to run the query.
- d. In the query result, you see multiple lines. There are one SELECT and four UPDATE statements.
 - i. Please ignore statements that don't begin with SELECT or UPDATE.
- e. The SELECT statement retrieves the customer records with a WITH(UPDLOCK) at the end.
- f. Notice that by optimizing the code the same result is obtained with less SQL statements and more optimal locking of tables.

Module Review

Module Review and Takeaways

This module covered the major points of insuring optimal performance in Microsoft Dynamics NAV applications, especially when you use SQL Server.

The specific ways in which SQL Server is implemented were discussed in detail to reveal why some operations are expensive as measured by performance, and other operations are not as expensive and just as effective.

The labs demonstrated how to use the SQL Profiler to analyze what happens on SQL Server. This tool should be used sparingly in a production environment, since the tool itself consumes a large number of system resources.

Test Your Knowledge

Test your knowledge with the following questions.

1. What are two important proprietary Dynamics NAV features that are simulated on SQL Server? Explain how these features are simulated.

2. What is the purpose of collation?

3. How can a SQL Server index be disabled from the table designer in C/SIDE without disabling the key?

C/SIDE Solution Development in Microsoft Dynamics® NAV 2013

4. What can be done to help avoid deadlocks?

5. Explain the difference between the clustered index and the primary key.

6. How is SIFT stored on SQL Server?

7. What tools in SQL Server can be used to troubleshoot performance issues?

Test Your Knowledge Solutions

Module Review and Takeaways

1. What are two important proprietary Dynamics NAV features that are simulated on SQL Server? Explain how these features are simulated.

MODEL ANSWER:

SIFT, which is simulated on SQL Server by indexed views, and before version 5.0 SP1 by additional SIFT tables.

Data Versioning, which is simulated on SQL Server by including a datetime value for each record in the database.

2. What is the purpose of collation?

MODEL ANSWER:

The collation of a database determines which character set is used to store the values in the database. It determines the way that data is sorted, and can affect the way that data is retrieved from the database.

3. How can a SQL Server index be disabled from the table designer in C/SIDE without disabling the key?

MODEL ANSWER:

By turning off the MaintainSQLIndex property of the key.

4. What can be done to help avoid deadlocks?

MODEL ANSWER:

To help avoid deadlocks, you can do the following:

- Lock tables in the same order for different types of transactions.
- Process records in the same order for different types of transactions.
- Keep transaction length to a minimum.
- Serialize the transaction, by locking a general table at the start of every transaction.

5. Explain the difference between the clustered index and the primary key.

MODEL ANSWER:

The clustered index is the index that is used by SQL Server to physically store the data. If the clustered index is set to any particular field, then SQL Server physically stores the records in the table in the order of that field.

The primary key is the key that defines the uniqueness of a record. Primary key field values uniquely identify a record in the table.

You can set an index other than the primary key index as a table's clustered index.

6. How is SIFT stored on SQL Server?

MODEL ANSWER:

Before version 5.0 SP1, SIFT was stored on SQL Server in separate SIFT tables, and SIFT totals were updated by table triggers on SQL Server. From version 5.0 SP1 forward, SIFT is stored on SQL Server by indexed views.

7. What tools in SQL Server can be used to troubleshoot performance issues?

MODEL ANSWER:

The SQL Profiler.

MODULE 14: APPENDIX

Module Overview

This training material simulates the development activities that are included in the scope of a fictional Microsoft Dynamics NAV® 2013 implementation project for CRONUS International Ltd.

Each module addresses a specific set of Microsoft Dynamics NAV 2013 functions and concepts, and you then apply those concepts to the solution for CRONUS.

This Appendix provides the case study for the implementation project at CRONUS, and it helps you test the solution that you develop.

Objectives

- Present the case study for the CRONUS International Ltd. implementation project.
- Provide test scripts for the function testing of all customized functionality.

CRONUS International Ltd.

You are a certified Microsoft Dynamics NAV 2013 developer, and you work for a Microsoft Certified Dynamics Partner. The project is for the CRONUS International Ltd., software training center. Because of significant growth, CRONUS requires a new computer-based system so that it can store and integrate all its seminar, instructor, customer, and financial information in one solution.

The client currently uses a full suite of Microsoft Dynamics NAV granules under the parent company CRONUS International Ltd. For CRONUS to take advantage of its investment, and the existing functionality and flexibility of Microsoft Dynamics NAV, it decided to add a customized Seminar Management module to its current solution.

With this new module, CRONUS must be able to do the following:

- Track its master data.
- Register participants in its seminars.
- Create invoices for customers.
- Have an overview of its statistics.

The preliminary analysis of the processes is complete and the functional requirements document is prepared.

Seminars

CRONUS holds several seminars. Each seminar has a fixed duration, and a minimum and maximum number of participants. In some cases, seminars can be overbooked, depending on the capacity of the assigned room. If there are not a sufficient number of participants, each seminar can be canceled. The price of each seminar is fixed. However, the solution must also be able to assign additional expenses to an instance of a seminar, such as catering expenses or equipment rentals.

Also, additional comments for each seminar can be entered for required equipment or other particular requirements.

Each seminar is held in a seminar room. Some seminars are held in-house and some are held off-site. If a seminar is held in-house, a room must be assigned. For off-site rooms, the rental rate must also be tracked.

Instructors

Each seminar is taught by an instructor, who is a CRONUS employee or a subcontractor. It is important to keep track of instructor availability and capacity, to avoid any scheduling conflicts.

Participants

Seminar participants must be associated with customers. The application must also maintain additional participant information, such as contact information, and the history of previously attended seminars.

Registration

A customer can register one or more participants for a seminar. The registration information must include how the seminar is invoiced, for example, whether to include expenses or catering.

Invoicing

When each seminar is finished, the customers who have participants are automatically invoiced.

Reporting and Statistics

Reports and statistics for seminar registrations must be available. These include the following:

- List of the participants registered for a seminar
- Total costs for each seminar, distributed according to what can or cannot be charged to the customer
- Statistics for different time periods, for example—one month, last year, the current year and cumulatively.

Interfaces

The solution must be able to use email notification that can be sent to the participants in several situations, such as registration confirmation. The solution must also be able to use simple web services integration so that external systems, such as websites, can easily connect to Microsoft Dynamics NAV 2013 at CRONUS, and they can also access the scheduled seminar list, or submit seminar registrations electronically.

Other Requirements

To make the solution user-friendly for CRONUS, the solution must include the following:

- **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 areas of Microsoft Dynamics NAV 2013, he or she is also able to intuitively learn this solution.

- **Efficient** - Experienced users must be able to work with the program efficiently. This means that the mouse and the functions that are used most frequently can be accessed from the keyboard.
- **Clarity** - The user interface must be intuitive so that the least-experienced user can easily understand how the program functions.
- **Good error reporting** - 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 about how the user can correct the error.
- **Design consistency** - Because the solution for the Seminar Management module can be sold as an add-on to other customers, it must follow the guidelines that are included in the Microsoft Dynamics NAV 2013 Developer and IT Pro Help.

Functional Requirements

Based on the analysis of the processes and the requirements at CRONUS, Simon, a business consultant at your company, has created the functional requirements document. It describes the requirements in a way that is easy to understand and verify.

The important functional and nonfunctional requirements for which you must design and develop a solution, in Microsoft Dynamics NAV 2013, are described in the following table.

ID	Requirement
NF-01	The solution must use the standard functionality of Microsoft Dynamics NAV 2013 wherever possible, and it must not duplicate functions already present in the application or cause redundant functionality.
NF-02	As long as there are no other requirements that contradict the proposed solution, any solution that proposes a standard functionality or the best practices of Microsoft Dynamics NAV 2013 is preferred.
NF-03	The solution must be consistent, user-friendly, and easy to learn and use. Any custom-built functionality must follow the standards, principles and best practices of Microsoft Dynamics NAV 2013, and must seamlessly integrate into the standard application.
NF-04	The solution must help users be productive so that they do not have to spend much time on searching and filtering.

Module 14: Appendix

ID	Requirement
FS-01	There are many seminars. Each seminar has a fixed duration, and a minimum and maximum number of participants. In some cases, seminars can be overbooked, depending on the capacity of the assigned room. If there are not a sufficient number of participants, each seminar can be canceled. The price of each seminar is fixed.
FS-02	Each seminar is held in a seminar room. Some seminars are held in-house and some are held off-site. If a seminar is held in-house, a room must be assigned. For off-site rooms, the rental rate must also be tracked. For all types of rooms, the solution must track the maximum number of participants the room can hold.
FS-03	Each seminar is taught by an instructor, who is either a CRONUS International Ltd. employee or a subcontractor. For subcontractors, the subcontracting rate must also be tracked.
FA-01	The solution must be aware of and provide the tools to manage and plan for the availability and the capacity of both rooms and instructors.
FP-01	Participants are persons who participate in seminars. For each participant, the solution must track the name, address and contact details including, at minimum a telephone number and an email address.
FP-02	If multiple participants from the same company attend the same seminar, then only one invoice must be sent to that company. The invoice must include all the names of the participants.
FI-01	If a participant pays for the seminar, then he or she will receive the invoice. However, if the company pays for the seminar, then the company will receive the invoice.
FI-02	The solution must automatically generate invoices for participation in a seminar, based on the transaction history and the information that is available on the completed seminars. Generating invoices does not have to start automatically. It is acceptable for a user to start the process manually.

ID	Requirement
FR-01	Users must be able to schedule seminars. Each seminar has a starting date, an allocated seminar room, the assigned instructor, the minimum and the maximum number of participants, and the price. The minimum number of participants and the price information are obtained from the seminar master record. The maximum number of participants is obtained as the lower number of maximum participants of the seminar and the maximum participants of the room.
FR-02	A seminar cannot be scheduled to be held in a room that cannot hold at least the minimum number of participants for the seminar.
FR-03	If the maximum room number exceeds the maximum number of participants that are scheduled for the seminar, then the user who is maintaining the registrations can determine whether to register more participants. This number can be the room's maximum capacity. However, the user must be warned if he or she is registering more participants than the maximum number of participants that can be registered for the seminar.
FR-04	Users must be able to assign additional expenses to a scheduled seminar, such as catering expenses or equipment rentals.
FR-05	Users must be able to register one or more participants for scheduled seminars. For each registered participant, the user must be able to specify if additional expenses must be invoiced for this registration. The default is Yes.
FR-06	For each scheduled seminar, users must be able to enter additional comments, for example, to note the necessary equipment, or other special requirements.
FH-01	When a seminar is completed, users must be able to move the seminar registration information into the transaction history, and disable any further modification of this information.

ID	Requirement
FH-02	Transaction history for the seminars must include the details for the participants, instructors, the rooms that are involved with the seminars, and the information about additional charges. This information will be the basis for seminar cost analytical and statistical reporting.
FH-03	Seminar registration information must integrate with the availability planning functionality for instructors and rooms, and it must provide the basis for automatically invoicing the customers.
FI-01	After a seminar is confirmed, seminar managers must be able to send an automatic email confirmation to all registered seminar participants. The solution must use the existing CRONUS's SMTP server to send any email messages.
FI-02	Seminar managers must be able to easily customize the template for the seminar confirmation notification email. Additionally, they must be able to enter free text, and placeholders for the recipient's name, the seminar name, and the seminar starting date.
FI-03	The solution must let external applications connect to it, and read and write data by using industry-standard protocols.

Content Structure

Each module in this training material introduces a major feature or concept in Microsoft Dynamics NAV 2013, that you can use after you complete this course.

The modules in this training material are structured as follows:

- **Prerequisite Knowledge** – This section is at the beginning of each module. It shows standard Microsoft Dynamics NAV 2013 concepts that are relevant to the development tasks and the activities that you will perform in the labs.
- **Solution Design** – This section analyzes the requirements and defines the high-level design points. Additionally, it will help you map the customer requirements to the standard Microsoft Dynamics NAV 2013 functionality. It also shows the best practices, and helps you consider the different design choices and select the optimal one.

- **Solution Development** – This section translates the design points into the summary of the development tasks and the activities that you perform in the labs. It also shows the user interface of the solution that you develop so that you can visualize what the solution should resemble when you develop it.
- **Labs:** The labs guide you step-by-step through the development tasks that are summarized in the Solution Development section.

Lab 14.1: Function Testing

Scenario

The labs in this training material focus mostly on the steps that are required to develop the solution and customize the standard Microsoft Dynamics NAV 2013 application. However, it is important for developers to make sure that the code and functionality that they create works, and complies with the requirements without having any issues. Developers do this by using the Function Testing process.

The labs in each module provide only the most important testing steps. However, you should always test the solution. One development task in the implementation project is to prepare the test scripts that developers, testers, or key users can follow to verify the functionality of the solution.

The test scripts presented here are just a suggestion, and are intended to help you test the solution as you develop it.

Exercise 1: Function Testing: Master Tables and Pages

Exercise Scenario

After you complete the labs that are in the Master Tables and Pages module, to prepare the necessary testing data subset, and to test the functionality of the master tables and pages, follow these steps.

Task 1: Creating No. Series

High Level Steps

1. Open the No. Series List page.
2. Create a number series for seminars.
3. Create a number series for seminar registrations.
4. Create a number series for posted seminar registrations.

Detailed Steps

1. Open the No. Series List page.
 - a. Start Microsoft Dynamics NAV 2013 client for Windows.
 - b. Click **Departments > Administration > Application Setup > General > No. Series**.
2. Create a number series for seminars.
 - a. Click **New**.
 - b. In the **Code** field, type "SEM".
 - c. In the **Description** field, type "Seminars".
 - d. Select the **Default Nos.** and **Manual Nos.** check boxes.

- e. Click **Navigate > Lines** to open the **No. Series Lines** page.
 - f. In the **Starting No. field**, type "S0001".
 - g. Click **OK** to close the **No. Series Lines** page.
3. Create a number series for seminar registrations.
 - a. Click **New**.
 - b. In the **Code** field, type "SEM-R".
 - c. In the **Description** field, type "Seminar Registrations".
 - d. Select the **Default Nos.** and **Manual Nos.** check boxes.
 - e. Click **Navigate > Lines** to open the **No. Series Lines** page.
 - f. In the **Starting No. field**, type "3001".
 - g. Click **OK** to close the **No. Series Lines** page.
 4. Create a number series for posted seminar registrations.
 - a. Click **New**.
 - b. In the **Code** field, type "SEM-R+".
 - c. In the **Description** field, type "Posted Seminar Registrations".
 - d. Select the **Default Nos.** and **Manual Nos.** check boxes.
 - e. Click **Navigate > Lines** to open the **No. Series Lines** page.
 - f. In the **Starting No. field**, type "301001".
 - g. Click **OK** to close the **No. Series Lines** page.
 - h. Click **OK** to close the **No. Series** page.

Task 2: Testing Seminar Setup

High Level Steps

1. Run the **Seminar Setup** page and switch to the Edit mode.
2. Configure the number series for seminars, seminar registrations, and posted seminar registrations.

Detailed Steps

1. Run the **Seminar Setup** page and switch to the Edit mode.
 - a. In Object Designer, select the **Seminar Setup** page.
 - b. Click **Run**.
 - c. In the **Seminar Setup** page, click **Edit**.
2. Configure the number series for seminars, seminar registrations, and posted seminar registrations.
 - a. In the **Seminar Nos.** field, type "SEM".
 - b. In the **Seminar Registration Nos.** field, type "SEM-R".
 - c. In the **Posted Seminar Registration Nos.** field, type "SEM-R+".

Task 3: Testing the Seminar List and the Seminar Card

High Level Steps

1. Run the **Seminar List** page.
2. Create a new Seminar.
3. Create another seminar.
4. Create one more seminar.
5. Enter a comment line about each seminar.

Detailed Steps

1. Run the **Seminar List** page.
 - a. In Object Designer, select the **Seminar List** page.
 - b. Click **Run**.
2. Create a new Seminar.
 - a. Click **New**.

 **Note:** This must open the **New – Seminar Card** page. If the **New – Seminar Card** page does not open, then the **CardPageID** property is not defined correctly in the **Seminar List** page.

- b. Click the **Name** field.

 **Note:** After you leave the **No.** field, the application must assign the number automatically. This verifies that the number series functionality works as expected.

- c. Enter the following information in the card.

Field	Value
Name	80436 C/SIDE Introduction
Seminar Duration	5
Minimum Participants	4
Maximum Participants	12
Gen. Prod. Posting Group	MISC
Seminar Price	1500

 **Note:** After you enter these values, the Search Name, VAT Prod. Posting Group, and the Last Date Modified fields are automatically updated.

- d. Do not close the **Seminar Card** page.

C/SIDE Solution Development in Microsoft Dynamics® NAV 2013

3. Create another seminar.
 - a. Click **New**.
 - b. Click the **Name** field.
 - c. Enter the following information in the card.

Field	Value
Name	80437 C/SIDE Solution Development
Seminar Duration	5
Minimum Participants	4
Maximum Participants	10
Gen. Prod. Posting Group	MISC
Seminar Price	2000

- d. Do not close the **Seminar Card** page.
4. Create one more seminar.
 - a. Click **New**.
 - b. Click the **Name** field.
 - c. Enter the following information in the card.
- | Field | Value |
|---------------------------------|------------------------------------|
| Name | 80450 Microsoft Dynamics Sure Step |
| Seminar Duration | 3 |
| Minimum Participants | 6 |
| Maximum Participants | 12 |
| Gen. Prod. Posting Group | MISC |
| Seminar Price | 999 |

 - d. Click **OK** to close the **Seminar Card** page.
 - e. Do not close the **Seminar List** page.
5. Enter a comment line about each seminar.
 - a. Select seminar 80436, and then click **Navigate > Comments**.
 - b. In the **Comment Sheet** page, in the **Comment** field, type "Students must be familiar with programming".
 - c. Click **OK** to close the **Comments Sheet** page.
 - d. Select seminar 80437, and then click **Navigate > Comments**.
 - e. In the **Comment Sheet** page, in the **Comment** field, type "Students must have completed 80436".
 - f. Click **OK** to close the **Comments Sheet** page.
 - g. Select seminar 80450, and then click **Navigate > Comments**.

- h. In the **Comment Sheet** page, in the **Comment** field, type "Project management knowledge required".
- i. Click **OK** to close the **Comments Sheet** page.
- j. Close the **Seminar List** page.

Task 4: Testing the Resource Card

High Level Steps

1. Run the **Resource Card** page.
2. Create a resource.
3. Create another resource.
4. Set the **Quantity Per Day** to eight for the LINDA and TIMOTHY resources.

Detailed Steps

1. Run the **Resource Card** page.
 - a. In Object Designer, select the **Resource Card** page.
 - b. Click **Run**.
2. Create a resource.
 - a. In the **Resource Card** page, click **New**.
 - b. In the **New – Resource Card** page, enter the following information.

Field	Value
No.	VENICE
Name	Room "Venice"
Type	Machine
Quantity Per Day	8
Gen. Prod. Posting Group	SERVICES
Unit Price	60
Maximum Participants	15

- c. In the **Base Unit of Measure** field, show the drop-down list box, and then click the **New** link in the drop-down list box (do not click **New** in the ribbon).
- d. In the **Select – Resource Unit of Measure** page, in the **Code** field, type "HOUR".
- e. Click **OK** to accept the change and close the **Select – Resource Unit of Measure** page.

C/SIDE Solution Development in Microsoft Dynamics® NAV 2013

3. Create another resource.
 - a. Create a new resource by following the same steps that you used for the VENICE room. Enter the same information in the card, except for the following fields.

Field	Value
No.	DUBROVNIK
Name	Room "Dubrovnik"
Type	Machine
Maximum Participants	10

- b. Press **CTRL+SHIFT+D**.
- c. In the **Edit Dimension Set Entries** page, enter the following dimension.

Field	Value
Dimension Code	BUSINESSGROUP
Dimension Value Code	INDUSTRIAL

- d. Close the Edit Dimension Set Entries and Resource Card pages.

4. Set the **Quantity Per Day** to eight for the LINDA and TIMOTHY resources.
 - a. In Object Designer, run the **Resource List** page.
 - b. Select the LINDA resource.
 - c. Click **Edit**.
 - d. In the **Quantity Per Day** field, type "8".
 - e. Press CTRL+SHIFT+D.
 - f. In the **Edit Dimension Set Entries** page, enter the following dimension.

Field	Value
Dimension Code	SALESPERSON
Dimension Value Code	LM

- g. Close the **Edit Dimension Set Entries** page.
- h. Click **OK** to close the **Resource Card** page.
- i. Repeat these steps for the TIMOTHY resource.

Exercise 2: Function Testing: Documents

Exercise Scenario

After you complete the labs in the Documents module, to test the functionality of the documents, follow these steps.

Task 1: Testing the Seminar Registration

High Level Steps

1. Run the **Seminar Registration List** page.
2. Create a new seminar registration.
3. Create one more seminar registration.

Detailed Steps

1. Run the **Seminar Registration List** page.
 - a. In Object Designer, select the **Seminar Registration List** page.
 - b. Click **Run** to open the **New – Seminar Registration** page.
2. Create a new seminar registration.
 - a. Click **New**.
 - b. Enter the following information in the **New – Seminar Registration** page.

Field	Value
Starting Date	01/07/13
Seminar No.	S0001

 **Note:** After you select the **Seminar No.**, the default values for the seminar are copied into the fields in the **Seminar Registration** page.

- c. In the **Room Resource No.** type "DUBROVNIK". A confirmation dialog box that asks about the maximum number of participants is shown.
- d. Click **Yes** to confirm the number of participants. The **Maximum Participants** field is set to 10.
- e. In the **Instructor Resource No.** field type "LINDA".
- f. In the **Lines** subpage, enter the following information.

Bill-to Customer No.	Participant Contact No.
10000	CT100140
10000	CT100156
10000	CT200136
01445544	CT100218

 **Note:** Verify that the content of the **Customer Details** FactBox updates, depending on the value of the **Bill-to Customer No.** field.

- g. Click **Navigate > Comments**.

C/SIDE Solution Development in Microsoft Dynamics® NAV 2013

- h. In the **Edit – Seminar Comment Sheet** page, enter the following comment: "This is the first time this instructor teaches this seminar".
 - i. Close the **Seminar Comment Sheet** page.
3. Create one more seminar registration.
- a. Click **Home > New**.
 - b. Enter the following information.

Field	Value
Starting Date	01/07/13
Seminar No.	S0003
Instructor Resource No.	TIMOTHY
Room Resource No.	VENICE

- c. In the **Lines** subpage, enter the following information.

Bill-to Customer No.	Participant Contact No.
01121212	CT100132
01454545	CT100152
34010100	CT100230

- d. Click **Navigate > Charges**.
- e. In the **Edit – Seminar Charges** page, enter the following information.

Field	Value
No.	MARK
Quantity	4

- f. Close the **Edit – Seminar Charges**, the **Resource Registration**, and the **Resource List** pages.

Exercise 3: Function Testing: Posting

Exercise Scenario

To test the functionality that you developed in the Posting module, follow these steps.

Task 1: Testing the Source Code Setup Page

High Level Steps

1. Run the **Source Code Setup** page.

Detailed Steps

1. Run the **Source Code Setup** page.
 - a. In Microsoft Dynamics NAV 2013 client for Windows, click **Departments > Administration > Application Setup > Financial Management > Trail Codes > Source Code Setup**.
 - b. Expand the **Seminar Management** FastTab.
 - c. In the **Seminar** field show the drop-down list box.
 - d. Click **New**.
 - e. In the **Select – Source Codes** page, enter the following information.

Field	Value
Code	SEM
Description	Seminar

- f. Click **OK** to accept the selection.
- g. Make sure that the **Seminar** field contains "SEM".
- h. Close the **Source Code Setup** page.

Task 2: Testing the Seminar Posting Codeunits

High Level Steps

1. Run the **Seminar Registration List** page.
2. Try posting a seminar registration.
3. Correct the error and retry posting.
4. Edit and try posting another seminar registration.
5. Correct the error and retry posting the seminar registration.

Detailed Steps

1. Run the **Seminar Registration List** page.
 - a. In Object Designer, select the **Seminar Registration List** page.
 - b. In the **Seminar Registration List** page, select the seminar registration for the 80436 seminar.

2. Try posting a seminar registration.
 - a. Press **F9** to start the posting process.
 - b. In the confirmation dialog box, click **Yes**.
 - A run-time error occurs stating that the Status must be equal to "Closed".
 - c. Click **OK** to close the error message.
3. Correct the error and retry posting.
 - a. Click **Edit**.
 - b. In the **Seminar Registration** page, change the Status to Closed.
 - c. Again, in the **Seminar Registration** page, click **F9** to start posting.
 - d. In the confirmation dialog box, click **Yes**.
 - e. In case there is an error message saying: "The combination of dimensions used in 3001, line no. 40000 is blocked. Dimension combinations BUSINESSGROUP - INDUSTRIAL and SALESCAMPAIGN - WINTER can't be used concurrently." Follow these steps:
 - i. Select the fourth line in the Lines and select **CTRL+SHIFT+D** to open the "Edit Dimensions Set Entries" window.
 - ii. In the "Edit Dimensions Set Entries" window select the row containing "SALESCAMPAGN" and select Delete.
 - iii. Answer **Yes** to "Delete Dimension Set Entry?"
 - iv. Close the "Edit Dimension Set Entries" window.
 - v. Again, in the **Seminar Registration** page, click **F9** to start posting.
 - f. In case there's an error message saying "The length of the string is 21, but it must be less than or equal to 20 characters. Value: CONTOSO\ADMINISTRATOR" follow these steps:
 - i. In the Development Environment, in the Object Designer, select table **Posted Seminar Reg. Header** and click Design.
 - ii. In the Table Designer, in the field **User ID** set **Length** to **50**.
 - iii. Close and Save the Table Designer.
 - iv. Repeat these steps for tables: **Seminar Ledger Entry** and **Seminar Register**.
 - v. Again, in the **Seminar Registration** page, click **F9** to start posting.

- g. A progress dialog box appears which shows the posting process. After posting is completed, the dialog box disappears.
 - h. After posting is completed, the **Seminar Registration** page closes automatically.
 - i. The **Seminar Registration** that you posted disappears from the **Seminar Registration List** page.
4. Edit and try posting another seminar registration.
- a. In the **Seminar Registration List** page, select the seminar registration for the 80450 seminar.
 - b. Click **Edit**.
 - c. In the **Seminar Registration** page, change the Status to Closed.
 - d. Click **OK** to close the **Seminar Registration** page.
 - e. In the **Seminar Registration List** page, click **F9** to start posting.
 - f. In the confirmation dialog box, click **Yes**.
 - g. A run time error occurs stating that the **Bill-to Customer No.** field must have a value in the Seminar Journal Line. This occurs because you are posting a seminar charge where the **Bill-to Customer No.** field is empty.
 - h. Click **OK** to close the error message.
5. Correct the error and retry posting the seminar registration.
- a. In the **Seminar Registration List**, click **Navigate > Charges**.
 - b. In the **Edit – Seminar Charges** page, in the **Bill-to Customer No.** field, type "01121212".
 - c. Click **OK** to close the page.
 - d. In the **Seminar Registration List** page, click **F9** to start posting.
 - e. In the confirmation dialog box, click **Yes**.
 - f. After posting is completed, the seminar registration disappears from the list.
 - g. Close the **Seminar Registration List**.

Task 3: Testing the Posted Pages

High Level Steps

1. Test the **Posted Seminar Reg. List** page.
2. Test the **Seminar Registers** page.
3. Test the **Resource Ledger Entries** page.

Detailed Steps

1. Test the **Posted Seminar Reg. List** page.
 - a. In Object Designer, select the **Posted Seminar Reg. List** page.
 - b. Click **Run**.
 - c. Verify that there are two posted seminar registrations in the list.
 - d. Select the first posted registration. It must be for the 80436 seminar.
 - e. Click **Navigate > Comments**.
 - f. Verify that a comment line is present.
 - g. Click **Navigate > Charges**.
 - h. Verify that there are no lines.
 - i. Double-click the second posted registration, for the 80450 seminar to open the **Posted Seminar Registration** page.
 - j. Verify that there are three seminar registration lines.
 - k. Click **Navigate > Charges**.
 - l. Verify that there is one line for the MARK resource.
 - m. Close the **View – Seminar Charges, Posted Seminar Registration**, and **Posted Seminar Reg. List** pages.
2. Test the **Seminar Registers** page.
 - a. In Object Designer, select the **Seminar Registers** page.
 - b. Click **Run**.
 - c. Verify that there are two lines, and that they have today's date, your User ID, and the source code SEM.
 - d. Verify that **From Entry No.** and **To Entry No.** are 1 through 6, and 7 through 12.
 - e. Select the second line.
 - f. Click **Seminar Ledger**.
 - g. The **View – Seminar Ledger Entries** page opens. It shows six entries—three entries for participants, and one for the charge, instructor, and each room.
 - h. Close the **Seminar Ledger Entries** page.
3. Test the **Resource Ledger Entries** page.
 - a. In Object Designer, select the **Resource Registers** page.
 - b. Click **Run**.
 - c. In the **Resource Registers** page, scroll down to the last line.
 - d. Verify that the last two lines have the source code SEM.
 - e. Select the last line, and then click **Resource Ledger**.
 - f. Verify that two lines are shown, one for TIMOTHY, and one for VENICE.

- g. Verify that both lines have a value in the **Seminar No.** and **Seminar Registration No.** fields.
- h. Close the **Resource Ledger Entries** and **Resource Registers** pages.

Exercise 4: Function Testing: Feature Integration

Exercise Scenario

To test how different features of the Seminar Management solution integrate together and with the standard Microsoft Dynamics NAV 2013 application, follow these steps.

Task 1: Testing the Seminar List and the Seminar Card

High Level Steps

1. Test how to create a new Seminar Registration from the **Seminar List**.
2. Test how to run the **Seminar Registration List** page for a specific seminar.
3. Test how to run the **Seminar Ledger Entries** page for a specific seminar.

Detailed Steps

1. Test how to create a new Seminar Registration from the **Seminar List**.
 - a. In Object Designer, select the **Seminar List** page.
 - b. Click **Run**.
 - c. In the **Seminar List** page, select the 80437 seminar.
 - d. Click **Actions > Seminar Registration**. The **New – Seminar Registration** page opens.
 - e. Click the **Starting Date** field. The **Seminar No.** field is automatically set to the **No.** of the **Seminar** record for which you created a new registration. Also, all other default fields related to the **Seminar No.** field are populated automatically.
 - f. Close the **Seminar Registration** page.
 - g. In the **Seminar List** page, select the 80450 seminar.
 - h. Click **Actions > Seminar Registration**.
 - i. Click the **Starting Date** field.
 - j. Close the **Seminar Registration** page.
2. Test how to run the **Seminar Registration List** page for a specific seminar.
 - a. In the **Seminar List** page, select the 80437 seminar.
 - b. Click **Navigate > Registrations**.
 - c. Verify that only one Seminar Registration is shown.

- d. Do not close the **Seminar List** page.
3. Test how to run the **Seminar Ledger Entries** page for a specific seminar.
 - a. In the **Seminar List** page, select the 80436 seminar.
 - b. Press **CTRL+F7**.
 - c. Verify that six Seminar Ledger Entry records are shown.
 - d. Do not close the **Seminar Ledger Entry** page.

Task 2: Testing the Navigate Feature

High Level Steps

1. Navigate to all records that are created for a **Seminar Ledger Entry**.

Detailed Steps

1. Navigate to all records that are created for a **Seminar Ledger Entry**.
 - a. In the **Seminar Ledger Entry** page, click **Navigate**.
 - b. A dialog box appears and it shows the counting progress.
 - c. After the progress dialog box closes, the **Navigate** page opens.
 - d. Verify that three lines are shown—one for each **Res. Ledger Entry**, **Posted Seminar Reg. Header**, and **Seminar Ledger Entry** tables.
 - e. Select the **Posted Seminar Reg. Header** line, and then click **Show**. The **Posted Seminar Registration** page opens.
 - f. Click **Close**.
 - g. Select the **Seminar Ledger Entry** line, and then click **Show**. The **Seminar Ledger Entries** page opens.
 - h. Click **Close**.
 - i. Close the **Seminar Ledger Entries**, and the **Seminar List** pages.

Exercise 5: Function Testing: Dimensions

Exercise Scenario

To test the dimensions functionality and to verify that the Seminar Management solution integrates with the standard dimension management features, follow these steps.

Task 1: Testing Default Dimensions

High Level Steps

1. Test how to assign default dimensions to multiple **Seminar** records.
2. Test how to assign default dimensions to a single **Seminar** record.

3. Test how to copy the default dimensions from the **Seminar** and the **Resource** records to the **Seminar Registration** records.
4. Test how to post with dimensions.

Detailed Steps

1. Test how to assign default dimensions to multiple **Seminar** records.
 - a. In Object Designer, select the **Seminar List** page.
 - b. Click **Run**.
 - c. In the **Seminar List** page, select all seminars.
 - d. Click **Navigate > Dimensions > Dimensions-Multiple**.
 - e. In the **Default Dimensions-Multiple** page, click **New**.
 - f. Enter the following default dimensions.

Dimension Code	Dimension Value Code
DEPARTMENT	SALES
BUSINESSGROUP	OFFICE

- g. Close the **Default Dimensions-Multiple** page.
2. Test how to assign default dimensions to a single **Seminar** record.
 - a. Double-click the 80437 seminar.
 - b. In the **Seminar Card** page, press **CTRL+SHIFT+D**.
 - c. Delete the BUSINESSGROUP dimension.
 - d. For the DEPARTMENT dimension, clear the **Dimension Value** field, and set **Value Posting** to the code Mandatory.
 - e. Close the **Default Dimensions** and the **Seminar Card** pages.
3. Test how to copy the default dimensions from the **Seminar** and the **Resource** records to the **Seminar Registration** records.
 - a. In the **Seminar List** page, select the 80436 seminar.
 - b. Click **Actions > Seminar Registration**.
 - c. In the **New – Seminar Registration** page, click the **Starting Date field**.
 - d. Press **CTRL+SHIFT+D**.
 - e. Verify that the **Edit Dimension Set Entries** page contains the BUSINESSGROUP and DEPARTMENT dimensions.
 - f. Close the **Edit Dimension Set Entries** and the **New – Seminar Registration** pages.
 - g. In the **Seminar List** page, select the 80437 seminar.
 - h. Click **Actions > Seminar Registration**.
 - i. Click the **Starting Date** field.
 - j. Press **CTRL+SHIFT+D**.

- k. Verify that the **Edit Dimension Set Entries** page is empty, and then close it.
 - l. In the **Instructor Resource No.** field, type "LINDA".
 - m. In the **Room Resource No.** field, type "DUBROVNIK".
 - n. Press **CTRL+SHIFT+D**.
 - o. Verify that the **Edit Dimension Set Entries** page contains the values for the BUSINESSGROUP and the SALESPERSON dimensions.
 - p. In the **Lines** subpage, in the **Bill-to Customer No.** field, type "10000", and in the **Participant Contact No.** field, type "CT100140".
 - q. Set the **Starting Date** to 01/21/13.
 - r. Set the Status to Closed.
4. Test how to post with dimensions.
 - a. Click **Post**.
 - b. In the confirmation dialog box, click **Yes**.
 - A run-time error occurs, stating that you must specify the **Dimension Value Code** for the DEPARTMENT dimension. This occurs because you specified that the DEPARTMENT dimension has the Mandatory code setting in the default dimensions configuration for the 80437 seminar.
 - c. Click **OK** to close the error message.
 - d. Press **CTRL+SHIFT+D**.
 - e. To the dimension set, add the DEPARTMENT dimension, with the value of SALES. Close the **Edit Dimension Set Entries** page.
 - f. A confirmation dialog box asks whether you want to update the dimensions for the lines.
 - g. Click **Yes** to continue.
 - h. In the **Lines** subpage, click **Line > Dimensions**.
 - i. Verify that there are AREA, BUSINESSGROUP, CUSTOMERGROUP, DEPARTMENT, and SALESPERSON dimensions. The AREA and CUSTOMERGROUP dimensions come from customer 10000.
 - j. Close the **Edit Dimension Set Entries** page.
 - k. Press **F9**.
 - l. Click **Yes** to continue.
 - m. Close all open pages.

Task 2: Testing Posted Dimensions

High Level Steps

1. Test posted document dimensions.
2. Test the seminar ledger entries dimensions.

Detailed Steps

1. Test posted document dimensions.
 - a. In Object Designer, select the **Posted Seminar Reg. List** page.
 - b. Click **Run**.
 - c. Select the last posted seminar in the list.
 - d. Press **CTRL+SHIFT+D**.
 - e. Verify that the BUSINESSGROUP, DEPARTMENT, and the SALESPERSON dimensions are present. Close the **Dimension Set Entries** page.
 - f. Click **View**.
 - g. In the **Posted Seminar Registration** page, in the **Lines** subpage, click **Line > Dimensions**.
 - h. Verify that the AREA, BUSINESSGROUP, CUSTOMERGROUP, DEPARTMENT, and the SALESPERSON dimensions are present.
 - i. Close the **Dimension Set Entries** page.
2. Test the seminar ledger entries dimensions.
 - a. Click **Navigate**.
 - b. In the **Navigate** page, select the **Seminar Ledger Entry** line, and then click **Show**.
 - The **Seminar Ledger Entries** page opens, with three lines.
 - c. Select the first line.
 - d. Press **CTRL+SHIFT+D**, and verify that there are five dimension lines present.
 - e. Close the **Dimension Set Entries** page.
 - f. Select the second line.
 - g. Press **CTRL+SHIFT+D**, and verify that there are three dimension lines present.
 - h. Close the **Dimension Set Entries**, the **Seminar Ledger Entries**, and the **Navigate** pages.

Exercise 6: Function Testing: RoleTailoring

Exercise Scenario

After you develop the Role Center and the Department pages, to verify the functionality, follow these steps.

Task 1: Testing the Role Center

High Level Steps

1. Set page **123456740 Seminar Manager Role Center** as the default Role Center.
2. Test activities.
3. Test My Seminars.

Detailed Steps

1. Set page **123456740 Seminar Manager Role Center** as the default Role Center.
 - a. Start Microsoft Dynamics NAV 2013 client for Windows, if it's not already open.
 - b. In the menu select **Departments, Administration, Application Setup, RoleTailored Client, Profiles**.
 - c. Click **New** to create a new profile.
 - d. In the New – Profile Card type **SEMINAR MANAGER** in the **Profile ID** and type **1234567840** in the **Role Center ID**.
 - e. Click **OK** to close.
 - f. In the menu select **Departments, Administration, Application Setup, RoleTailored Client/User Personalization**.
 - g. Select **CONTOSO\ADMINISTRATOR** in the list and then click **Edit**.
 - h. Set the **Profile ID** to **SEMINAR MANAGER**.
 - i. Click **OK** to close the page.
 - j. Restart Microsoft Dynamics NAV 2013 client for Windows.
2. Test activities.
 - a. In the **Seminar Manager Activities** part, click the **Planned** cue.
 - b. Verify that the **Registrations, Planned** list place is shown.
 - c. Click **Role Center**.
 - d. In the **Seminar Manager Activities**, click **New**.
 - e. Verify that the **New – Seminar Registration** page is shown, and then close it.

3. Test My Seminars.
 - a. In the **My Seminars** list part, click **Manage List**.
 - b. In the **Edit – My Seminars** page, select the 80436 seminar, and then click **OK**.
 - c. Verify that in the **My Seminars** list part there is a line for the 80436 seminar.
 - d. In the **My Seminars** list part, click **Open**.
 - e. Verify that the **Edit – Seminar Card** page opens for the 80436 seminar.
 - f. Close the **Edit – Seminar Card** page.

Task 2: Testing the Search Field

High Level Steps

1. Search for the **Seminar Registers** page.

Detailed Steps

1. Search for the **Seminar Registers** page.
 - a. In the **Search** field, type "Seminar Registers".
 - b. Click the **Seminar Registers** page.
 - c. Verify that the **Seminar Registers** list place shows.

Module Review

Module Review and Takeaways

CRONUS International Ltd. is a fictional company that is in the process of implementing Microsoft Dynamics NAV 2013. This module presented a summary of the company's functional requirements.

When testing functionality, you frequently must enter test information to verify that customized behavior matches the customer's requirement. This module showed sample steps you can follow to test the application as you progress through the course.