



Solution Development in Visual Studio  
Code

## **Copyright © 2018 Dynamics 365 Courseware ApS**

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

## **Warning and Disclaimer**

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the authors, nor the publisher, and its dealers and distributors will be held liable for any damages caused or alleged to be caused directly or indirectly by this book.

## **Trademark**

The publisher has endeavored to provide trademark information about all of the companies and products mentioned in this book by the appropriate use of capitals. However, the publisher cannot guarantee the accuracy of this information.

First edition published: March 2018

Published by:

Dynamics 365 Courseware ApS

Bolbrovej 121

2960 Rungsted Kyst

# TABLE OF CONTENT

Introduction.....	5
Module 1: Solution Overview.....	6
Objectives .....	6
CRONUS International Ltd.....	7
Functional Requirements .....	9
Solution Overview.....	13
Lab 1.1: Class discussion – Standard vs. new functionality .....	15
Module 2: Introduction to the Visual Studio Code editor .....	16
Development in AL.....	16
Functions in the Visual Studio Editor .....	18
Supported Objects .....	21
Snippet support.....	27
Differences in the Development Environments.....	30
C/Side and AL Side-by-side development .....	33
Getting Started with AL .....	35
The first test .....	43
Lab 2.1 Publish a test extension to Dynamics 365 BC.....	47
Module 3: Setting up a Git repository .....	49
Installing Git and initializing the repository.....	50
Cloning an existing repository .....	57
Other Git functionalities worth mentioning.....	59
Lab 3.1 Set up a Git Repository .....	64
Module 4: Data and Process Model .....	66
Module Overview.....	66
Developing solutions for Dynamics 365 BC.....	70
Objectives .....	71
Table Types and Characteristics .....	71
Standard Data Model.....	83
Lab 4.1 Prepare the workspace for the Seminar solution.....	97
Module Review .....	98
Module 5: Master Tables and Pages .....	99

Module Overview .....	99
Prerequisite Knowledge.....	100
Participants.....	104
Instructors and Rooms.....	107
Lab 5.1: Customize Resource Tables and Pages.....	110
Seminars.....	122
Development.....	128
Lab 5.2: Creating Seminar Tables and Pages.....	131
Lab 5.3: Creating Seminar Comment Line Table and Pages.....	160
Module Review.....	168
<b>Module 6: Documents</b> .....	169
Module Overview.....	169
Prerequisite Knowledge.....	170
Registrations .....	181
Lab 6.1: Importing, Reviewing and Completing Seminar Registration Tables.....	187
Demonstration: Reviewing the Seminar Registration Line Code.....	206
Demonstration: Reviewing the Seminar Charge Table .....	214
Lab 6-2: Reviewing the Seminar Comment Line Table and Pages.....	219
Lab 6.3: Create Seminar Registration Pages .....	223
Module Review .....	228
<b>Module 7: Posting</b> .....	229
Module Overview.....	229
Prerequisite Knowledge.....	230
Posting Seminar Registrations .....	246
Lab 7.1: Reviewing and Completing the Journal and Ledger Tables.....	254
Lab 7.2: Creating Codeunits and Pages for Seminar Journal Posting .....	267
Lab 7.3: Creating the Tables and Pages for Posted Registration Information.....	288
Lab 7.4: Modifying Tables, Pages, and Codeunits for Resource Posting .....	295
Lab 7.5: Creating the Codeunits for Document Posting .....	303
Module Review .....	325
<b>Module 8: Feature Integration</b> .....	326
Module Overview.....	326

Prerequisite Knowledge.....	326
Seminar Feature Integration.....	328
Lab 8.1: Integrating Seminar Features .....	333
Navigate Integration.....	338
Lab 8.2: Changing Objects to Integrate with Navigate .....	345
Module Review .....	353
Module 9: Reporting .....	354
Module Overview.....	354
Prerequisite Knowledge.....	354
Reporting Lab Overview.....	358
Participant List Reporting .....	358
Lab 9.1: Creating the Seminar Participant List.....	361
Task 1: A .....	361
Task 2: Create a codeunit to increment the No. Printed field just added to the Seminar Registration Header table.....	362
Task 3: Create the actual dataset in Report Dataset Designer .....	363
Lab 9.2: Review the Invoice Posting Batch Job.....	375
Task 1: Import the Create Invoices report object .....	375
Task 2: Review the Create Invoices report.....	375
Module Review .....	385
Module 10: Role Tailoring .....	386
Module Overview.....	386
Prerequisite Knowledge.....	386
Seminar Manager Role Center.....	396
Lab 10.1: Create the Seminar Manager Role Center.....	400
Module Review .....	429

## **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. 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.

### **Dynamics Courseware**

The 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.

### **Module 1: Solution Overview**

This training material simulates the development activities that are included in the scope of a fictional Dynamics 365 BC implementation project for CRONUS International Ltd.

Each module addresses a specific set of Dynamics 365 BC functions and concepts, and you then apply those concepts to the solution for CRONUS.

This module 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 the solution overview of the future solution

## **CRONUS International Ltd.**

CRONUS International Ltd. Has requested a Microsoft Certified Dynamics Partner for help with a project. 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 Dynamics granules under the parent company CRONUS International Ltd. For CRONUS to take advantage of its investment, and the existing functionality and flexibility of the Dynamics products, 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 itsseminars
- 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— this period, 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 Dynamics 365 BC 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 Dynamics 365 BC, he or she is also able to intuitively learn this solution.

## Functional Requirements

Based on the analysis of the processes and the requirements at CRONUS the functional requirements document has been created. 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 Dynamics 365 BC, are described in the following table.

<b>ID</b>	<b>Requirement</b>
<b>NF-01</b>	The solution must use the standard functionality of Dynamics 365 BC wherever possible, and it must not duplicate functions already present in the application or cause redundant functionality.
<b>NF-02</b>	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 Dynamics 365 BC is preferred.
<b>NF-03</b>	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 Dynamics 365 BC, and must seamlessly integrate into the standard application.
<b>NF-04</b>	The solution must help users be productive so that they do not have to spend much time on searching and filtering.

## Solution Development in Visual Studio Code

---

ID	Requirement
<b>FS-01</b>	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.
<b>FS-02</b>	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.
<b>FS-03</b>	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
<b>FA-01</b>	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.
<b>FP-01</b>	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.
<b>FP-02</b>	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.
<b>FI-01</b>	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.
<b>FI-02</b>	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
<b>FR-01</b>	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.
<b>FR-02</b>	A seminar cannot be scheduled to be held in a room that cannot hold at least the minimum number of participants for the seminar.
<b>FR-03</b>	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.
<b>FR-04</b>	Users must be able to assign additional expenses to a scheduled seminar, such as catering expenses or equipment rentals.
<b>FR-05</b>	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.
<b>FR-06</b>	For each scheduled seminar, users must be able to enter additional comments, for example, to note the necessary equipment, or other special requirements.
<b>FH-01</b>	When a seminar is completed, users must be able to move the seminar registration information into the transaction history and disable any further modifications of this information.

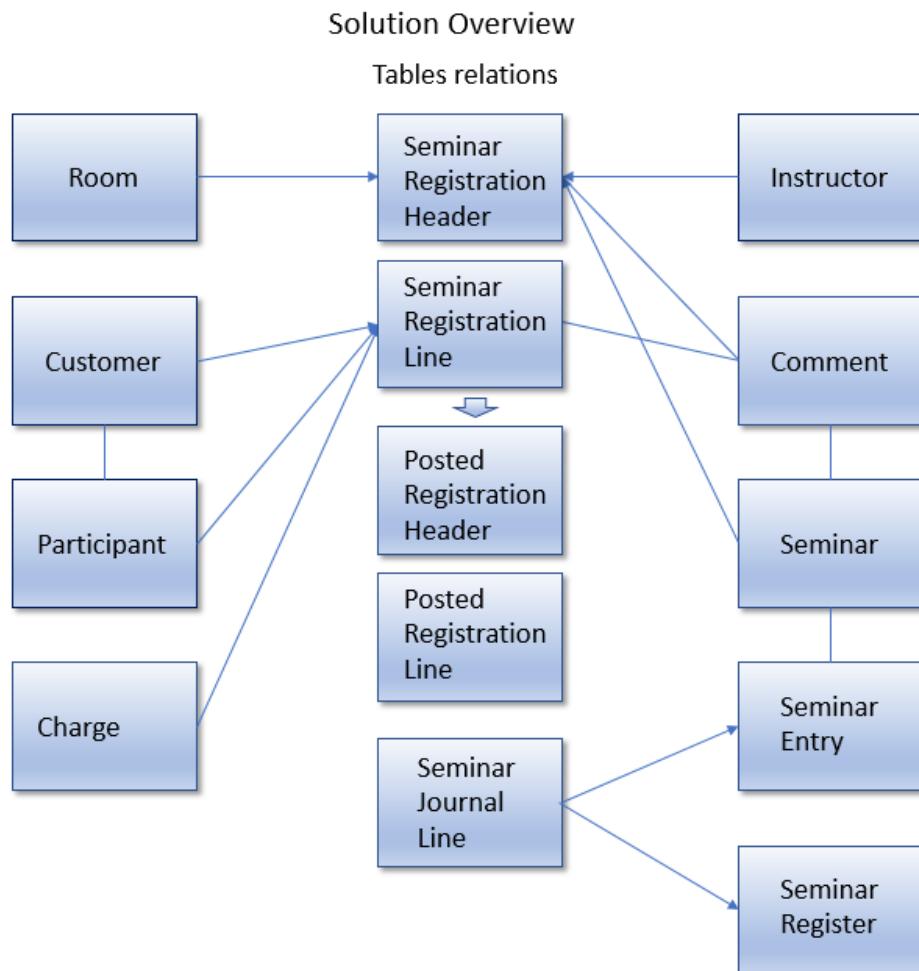
## Solution Development in Visual Studio Code

---

ID	Requirement
<b>FH-02</b>	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.
<b>FH-03</b>	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.
<b>FI-01</b>	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.
<b>FI-02</b>	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.
<b>FI-03</b>	The solution must let external applications connect to it, and read and write data by using industry-standard protocols.

## Solution Overview

The solution must be built as an add-on solution, interfering with as little standard functionality as possible. The overall solution Could look like this:



### Master Data tables

The Seminars are created in a new table

### Supplemental tables

Comment and charges are created in new tables

### Working tables

Seminar Registration Header and line are new working tables to hold the registrations

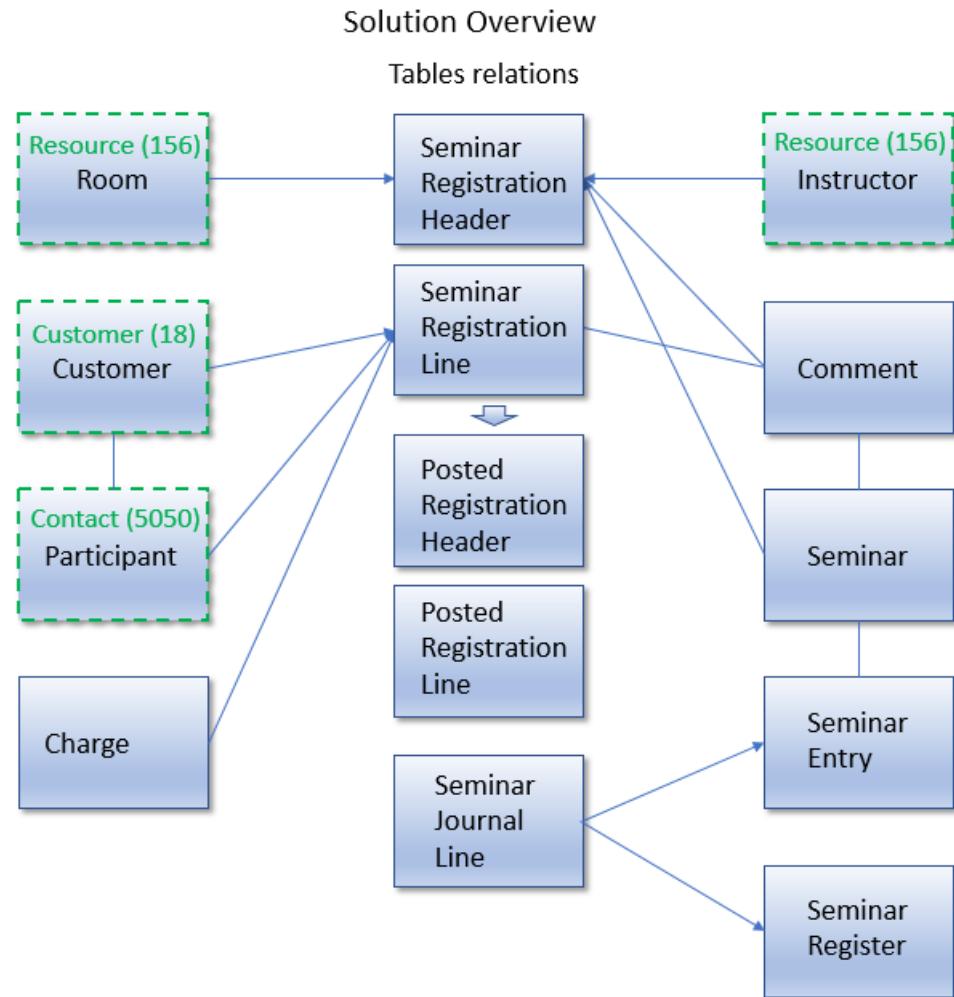
Seminar Journal Line is mandatory to create the Seminar entries if the solution should comply with the Design Patterns of Dynamics 365 BC.

### History Tables

Posted Seminar Registration Header and Line will together with the Seminar Entry and Seminar Register tables hold the history and audit trail

### Integration

In order to integrate with the standard solution with the least possible impact, some of the tables must be reused, either as-is or with minor modifications.



Rooms and Instructors can be created as resources in table 156 with the Type of Machine and Person and minor additional fields.

Customer can reuse the table 18 without any changes.

Participants can reuse the Contact table 5050 without any changes.

## Lab 1.1: Class discussion – Standard vs. new functionality

In the light of the previous solution overview, discuss the advantages or disadvantages of using the Job module for the seminars as opposed to the described solution, creating tables

for:

- Seminar
- Seminar Entry
- Seminar Journal Line
- Seminar Register

And creating posting routines:

- Seminar Check Journal Lines
- Seminar Post Journal Line

Also discuss the advantages or disadvantages of using standard functionality for the registrations:

- Seminar Registration Header -> Sales Header
- Seminar Registration Line -> Sales Line
- Posted Seminar Registration Header -> Posted Shipment Header
- Posted Seminar Registration Line -> Posted Shipment Line

And the equivalent posting routines:

- Seminar Post -> Sales-Post
- Seminar Post (Yes/No) -> Sales- Post (Yes/No)
- Seminar Post + Print -> Sales-Post + Print

# Module 2: Introduction to the Visual Studio Code editor Development in AL

The Visual Studio Code editor (VS Code) is a lightweight but powerful opensource editor. As described on the homepage, it runs on your desktop and is available for Windows, macOS and Linux. It comes with built-in support for JavaScript, TypeScript and Node.js and has a rich ecosystem of extensions for other languages (such as C++, C#, Java, Python, PHP, Go) and runtimes (such as .NET and Unity). Now Microsoft have released an extension for VS Code so that it supports the AL language, that is necessary to create extensions for Dynamics 365 "BC".

According to the Developer and IT-Pro Help for Dynamics NAV, the modern development environment is introduced with the following words:

Extensions are a programming model where functionality is defined as an addition to existing objects and defines how they are different or modify the behavior of the solution. This section explains how you can develop extensions using the development environment for Dynamics 365 "BC".

## Understanding objects in the development environment

All functionality in Dynamics 365 BC is coded in objects. The extension model is object-based; you create new objects, and extend existing objects depending on what you want your extension to do.

- Table objects define the table schema that holds data
- Page objects represent the pages seen in the user interface
- Codeunits contain code for logical calculations and for the application behavior
- Reports are objects to present data as print either as pdf or on paper
- Queries as extracts of data from one or more tables to be used either in code, in Generic Charts or as a oData web service
- XMLPorts will enable import and export of data to and from the Dynamics 365 BC database. XML Ports can also be used as parameter in Web Services

These objects are stored as code, known as AL code, and are saved in files with the .al file extension.



It is possible to store multiple objects in one .al file

There are two other special objects which are specifically used for building extensions.

- Table extension objects used to add fields to existing table objects
- page extension objects used to add fields, page parts and actions to existing page objects

For example, an extension for managing a business that sells organic food may define a table extension object for the Item table that contains two additional fields, Organic and Produced Locally. The Organic and Produced Locally fields aren't usually present in the Item table, but through the table extension these data fields will now be available to store data in and to access from code. You can then

use the page extension object to display the fields that you added to the table object.



Extension objects can have a name with a maximum length of 30 characters.

## Developing extensions in Visual Studio Code

Using the AL Language extension for Visual Studio Code, you'll get the benefits of a modern development environment along with seamless publishing and execution integration with your Dynamics 365 BC tenant. For more information on getting up and running, see [Getting Started with AL](#).

Visual Studio Code and the AL Language extension lets you do the following tasks:

- Create new files for your solution
- Get assistance with creating the appropriate configuration and setting files
- Use code snippets that provide templates for coding application objects
- Get compiler validation while coding
- Press F5 to publish your changes and see your code running

## Designer

The Designer works in the client itself allowing design of pages using a drag-and-drop interface. The Designer allows building extensions in the client itself by rearranging fields, adding fields, and previewing the page design.

## Compiling and deploying

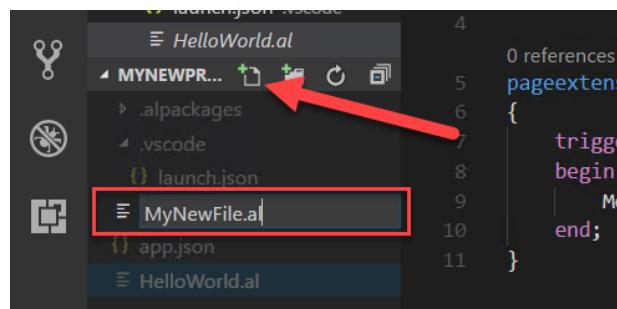
Extensions are compiled as **.app** package files. The **.app** package file can be deployed to the Dynamics 365 BC server.

An **.app** package contains the various artifacts that deliver the new functionality to the Dynamics 365 BC deployment as well as a manifest that specifies the name, publisher, version, and other attributes of the extension.

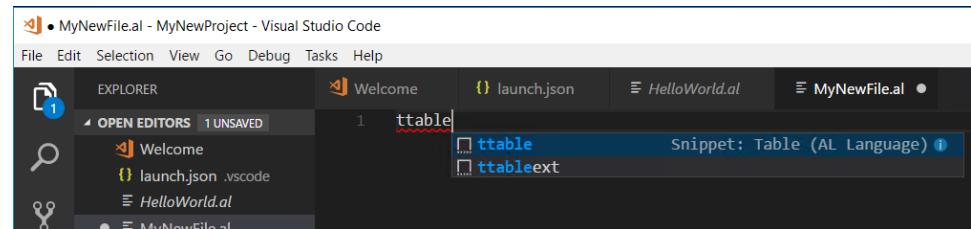
# Functions in the Visual Studio Editor

## Creating files and using snippets

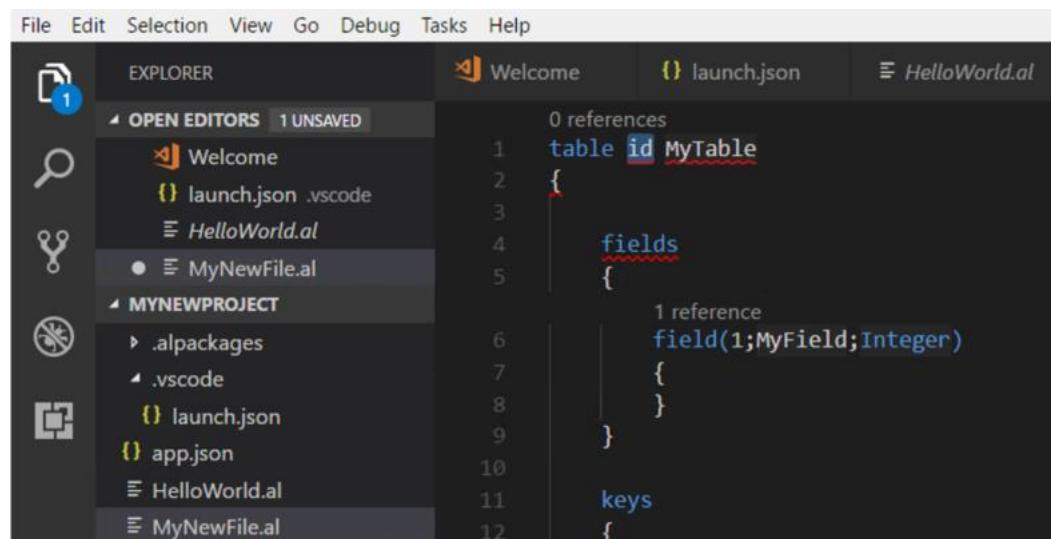
Creating a new file for testing. Remember to give it the .al extension, or the snippets will not work.



On an empty line, enter a snippet name:



That will create a template of the object:



## Defining variables

In any object, it is possible to create functions, global and local variables and text constants. Text constants are defined as labels and all multi language is transferred to external files:

```
codeunit 123456799 "CSD TestCodeunit"
{
    trigger OnRun()
    begin
        //Global Stuff
    end;

    local procedure MyProcedure()
    var
        MyLocalCodeVar: Code[20];
        MyLocalTextConst : label 'My new local text';
        MyLocalCustRec : Record Customer;

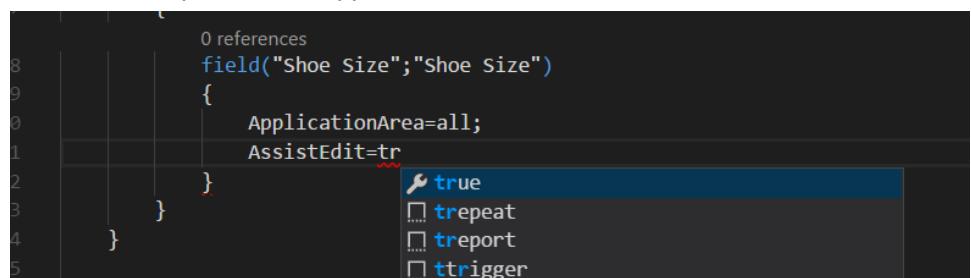
    begin
        //Local Stuff
    end;
}
```

```
var
    MyGlobalCodeVar: Code[20];
    MyGlobalTextConst : label 'My new global text';
    MyGlobalCustRec : Record Customer;
    MyGlobalArray : array[10] of code[20];
    TempCust : Record Customer temporary;

    [InDataSet]
    MyGlobalIncludedInDataset : Boolean;
}
```

## Intellisense

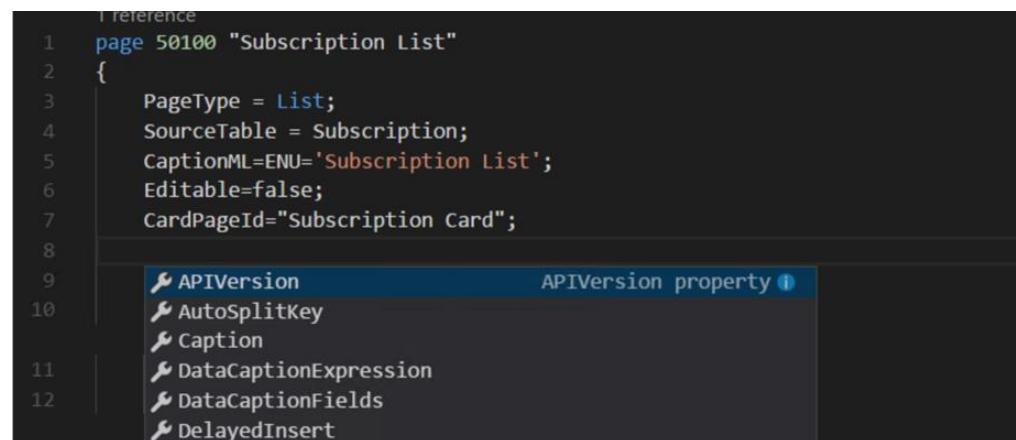
In any given place of the code window, Intellisense will be available. Start typing and the list of possibilities appear:



If the Intellisense does not work, then there is something wrong earlier in the code. Look for the error as a red underline.

### Property list

In addition to that, it is possible to press **Ctrl+Space** on any given place in the code window to get an overview of all available properties that can be used in this place of the .al file. When a property has been used, it will disappear from the list.

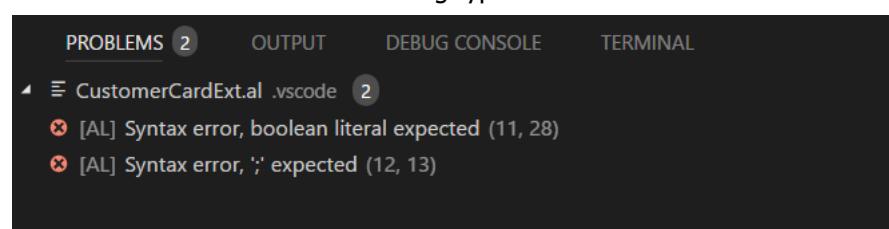


A screenshot of the Visual Studio Code interface. The code editor shows a snippet of AL code for a page, specifically a subscription list. A property list overlay is displayed, listing various properties like APIVersion, AutoSplitKey, Caption, and DataCaptionExpression. The 'APIVersion' property is highlighted in blue, indicating it is currently selected or being used.

```
1  page 50100 "Subscription List"
2  {
3      PageType = List;
4      SourceTable = Subscription;
5      CaptionML=ENU='Subscription List';
6      Editable=false;
7      CardPageId="Subscription Card";
8
9      APIVersion
10     AutoSplitKey
11     Caption
12     DataCaptionExpression
13     DataCaptionFields
14     DelayedInsert
```

### Errors

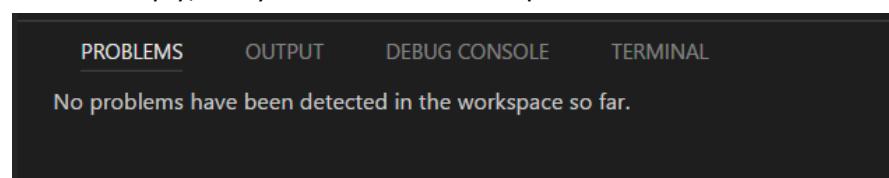
The Output Window will continuously show the status of the code and highlight all errors even as the command is being typed.



A screenshot of the Visual Studio Code Output window. The 'PROBLEMS' tab is selected, showing two errors for a file named 'CustomerCardExt.al'. The errors are: '[AL] Syntax error, boolean literal expected (11, 28)' and '[AL] Syntax error, ';' expected (12, 13)'.

PROBLEMS	OUTPUT	DEBUG CONSOLE	TERMINAL
CustomerCardExt.al .vscode	2		
✖ [AL] Syntax error, boolean literal expected (11, 28)			
✖ [AL] Syntax error, ';' expected (12, 13)			

It might not be a problem whenever the line is completed, but the output window must be empty, every time a section is completed.



A screenshot of the Visual Studio Code Output window. The 'PROBLEMS' tab is selected, and the message 'No problems have been detected in the workspace so far.' is displayed.

PROBLEMS	OUTPUT	DEBUG CONSOLE	TERMINAL
No problems have been detected in the workspace so far.			

# Supported Objects

Extensions made in VS Code supports the following objects:

## New tables

New tables can be created with no limitations.

The numbering of both object id and field numbers follow the same rules that apply to a new table made in the C/Side development Environment.

Code can be applied to both object triggers and field triggers and properties can be added without limitations. It is recommended that the object id is placed in a remote area of the 50.000-99.999 area or even better, that an add-on number series Is used, to avoid conflicts with other extensions.

## Table Extensions

### Table properties

As a main rule, it is not possible to change table properties in a table extension. However, there are a number of exceptions:

- Caption
- DataCaptionFields
- Description

Those are the only ones that can be changed in a table extension.

### Table Triggers

It is not possible to change to delete code in existing triggers. However, it is possible to add code in triggers before and after the existing trigger. The available triggers are:

- OnBeforeInsert
- OnAfterInsert
- OnBeforeModify
- OnAfterModify
- OnBeforeDelete
- OnAfterDelete
- OnBeforeRename
- OnAfterRename

### Fields

Existing tables can be extended with new fields. Also here, It is recommended that the field number is placed in a remote area of the 50.000-99.999 area or even better, that an add-on number series Is used, to avoid conflicts with other extensions.

### Field Properties

Field properties can be added without limitations on new fields. The properties on existing fields that can be changes are:

- Caption

- DataCaptionFields
- Description
- CaptionClass
- OptionCaption
- ClosingDates
- Width

### Field Triggers

It is not possible to add code to the existing triggers on a field, but it is possible to add code to triggers before and after the OnValidate trigger:

- OnBeforeValidate
- OnAfterValidate

On new fields, it is possible to add code to the usual triggers:

- OnValidate
- OnLookup

### New pages

New pages can be created with no limitations.

The numbering of both object id and field numbers follow the same rules that apply to a new table made in the C/Side development Environment.

Code can be applied to both object triggers and field triggers and properties can be added without limitations.

It is recommended that the object id is placed in a remote area of the 50.000-99.999 area or even better, that an add-on number series Is used, to avoid conflicts with other extensions.

### Page Extensions

#### Page properties

As a main rule, it is not possible to change page properties in a page extension. However, there are a number of exceptions:

- Caption
- Description
- InstructionalText
- DataCaptionExpression
- PromotedActionCategories

Those are the only page properties that can be changed in a page extension.

#### Page Triggers

It is not possible to change to delete code in existing triggers. However, it is possible to add code in separate triggers that will be executed after the existing trigger. The available triggers are:

- OnAfterGetCurrRecord

- OnAfterGetRecord
- OnClosePage
- OnDeleteRecord
- OnInsertRecord
- OnModifyRecord
- OnNewRecord
- OnOpenPage
- OnQueryClosePage

### Fields

Existing tables can be extended with new fields. Also here, It is recommended that the field number is placed in a remote area of the 50.000-99.999 area or even better, that an add-on number series Is used, to avoid conflicts with other extensions.

### Field Properties

Field properties can be added without limitations on new fields. The properties on existing fields that can be changes are:

- Caption
- DataCaptionFields
- Description
- CaptionClass
- OptionCaption
- ClosingDates
- Width
- Style
- ApplicationArea
- Visible
- StyleExpr
- ToolTip
- Enabled
- HideValue
- ShowCaption
- Importance
- QuickEntry
- ODataEDMType

### Field Triggers

It is possible to add code to the existing triggers on a field:

- OnBeforeValidate
- OnAfterValidate

On new fields, it is possible to add code to the usual triggers:

- OnValidate
- OnLookup

Besides these, it is possible to change the following control properties:

### **Page field group**

- Visible
- Enabled
- InstructionalText
- FreezeColumn

### **Page part**

- Visible
- Enabled
- ToolTip

### **Page action group**

- Caption
- Description
- ToolTip
- Visible
- Enabled

### **Page action**

- Caption
- Description
- ToolTip
- Visible
- Enabled
- ApplicationArea
- Promoted
- PromotedIsBig
- PromotedOnly
- PromotedCategory
- ShortcutKey
- InFooterBar

### Page action triggers

Page action triggers cannot be changed or extended. Instead, an Eventsubscription must be made.

### Reports

Reports cannot be extended nor as an extension or through event subscriptions to triggers in the reports.

Only option to interact with a report is if a **Business Event** or **Integration Event** has been added to the report

### Codeunits

New codeunits can be created without limitations. Existing codeunits cannot be extended. Only option to interact with a codeunit is if a **Business Event** or **Integration Event** has been added to the codeunit.

#### Install Codeunits

Install codeunits are codeunits with the **SubType=Install**.

An Install codeunit have two triggers built in:

- OnInstallAppPerCompany  
All changes that must be made to tables with **DataPerCompany=true**.
- OnInstallAppPerDatabase  
All changes that must be made to tables with **DataPerCompany=false**.

The triggers will be executed on installing the app. All other procedures will be ignored unless called from the two triggers. If an error occur in the triggers, the app will not be installed.

#### Upgrade Codeunits

Install codeunits are codeunits with the **SubType=Upgrade**.

An Install codeunit have these triggers built in:

Used to check that certain requirements are met in order to run the upgrade.

- OnCheckPreconditionsPerCompany  
Check preconditions for tables with **DataPerCompany=true**.
- OnCheckPreconditionsPerCompany  
Check preconditions for tables with **DataPerCompany=false**.

Used to perform the actual upgrade

- OnUpgradePerCompany
- OnUpgradePerDatabase

Used to check that the upgrade was successful.

- OnValidateUpgradePerCompany

- OnValidateUpgradePerDatabase

The triggers will be executed on upgrading the app. All other procedures will be ignored unless called from the two triggers. If an error occur in the triggers, the app will not be installed.

### Queries

New queries can be created without limitations. Existing queries cannot be extended. Only option to interact with a query is if a **Business Event** or **Integration Event** has been added to the query.

### XMLports

XMLports cannot be extended nor as an extension or through event subscriptions to triggers in the XMLport.

Only option to interact with a XMLport is if a **Business Event** or **Integration Event** has been added to the XMLport

### MenuSuites

Changes to MenuSuites are not supported in extensions. However, a menu item will automatic be created with the name of the extension.

It is possible to add menu items can added by adding a UsageCategory to the following objects:

- Pages
- Reports

The UsageCategory can be set to:

- Administration
- Documents
- History
- Lists
- None
- ReportsAndAnalysis
- Tasks

### Snippet support

Snippets will help you create the basic layout for an object when using the AL Extension in Visual Studio Code. Snippets can be used for whole objects and for individual parts like fields, key, actions and others.

The main snippets are the object snippets:

Table, Page, Codeunit, Report, XMLport, Query

Besides those are the extension objects which also supports snippets:

Table extension, Page extension

The snippets include all the necessary section to build the object. An example is the table snippet. The ttable snippet is divided into four sections. The first block contains metadata for the overall table; the table type. The fields section describes the data elements that make up the table; their name and the type of data they can store. The keys section contains the definitions of the keys that the table needs to support. The final section details the triggers and code that can run on the table.

Metadata:

```
5 references
table 50100 Subscription
{
    CaptionML = ENU = 'Subscription';
```

Fields:

```
fields
{
    3 references
    field(1; Code; Code[20])
    {
        CaptionML = ENU = 'Code';
```

Keys:

```
keys
{
    - reference
    key(PK; Code)
    {
        Clustered = true;
```

Triggers:

```
trigger OnInsert();
var
    myInt : Integer;
begin

end;
```

## Solution Development in Visual Studio Code

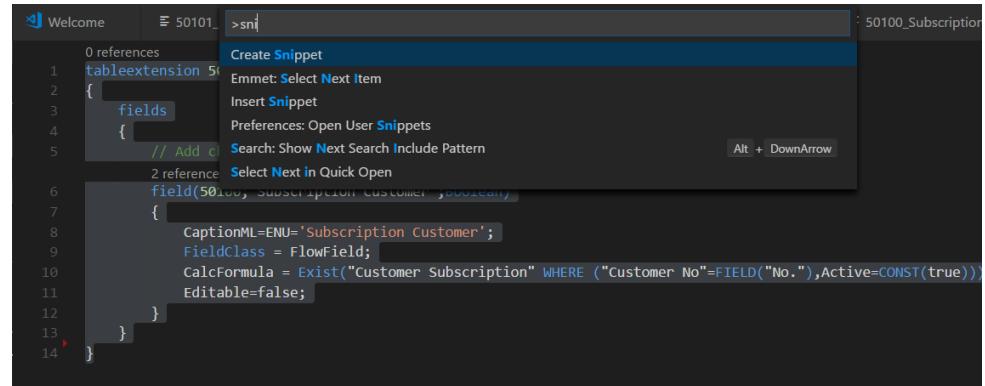
---

Below is a list of some of the snippets and their use:

<b>Objects</b>	<b>Description</b>
ttable	Create new table
ttableext	Create extension for an existing table
tpage	Create new page
tpageext	Create extension for an existing page
treport	Create new report
txmlport	Create new XMLport
tquery	Create new query
tcodeunit	Create new codeunit
<b>In objects</b>	
ttrigger	Create example of a new trigger
tprocedure	Create example of a new procedure (function)
taction	Create example of a new action
<b>In tables</b>	
tfield	Create example of a new field
tfieldcode	Create example of a new field of type code, length in [20]
tfielddate	Create example of a new field of type date
tfielddecimal	Create example of a new field of type decimal
	...
<b>Events</b>	
teventbus	Create example of a new business event
teventint	Create example of a new integration event
teventsbus	Create example of a new event subscription
<b>Repetitive commands</b>	
tfor	Create example of a for - to loop
tforeach	Create example of a foreach loop
twhile	Create example of a while loop
trepeat	Create example of a repeat - until loop
<b>Statements</b>	
tif	Create example of an if - then statement
tcaseelse	Create example of an if - then - else statement
tcaseof	Create example of an case statement

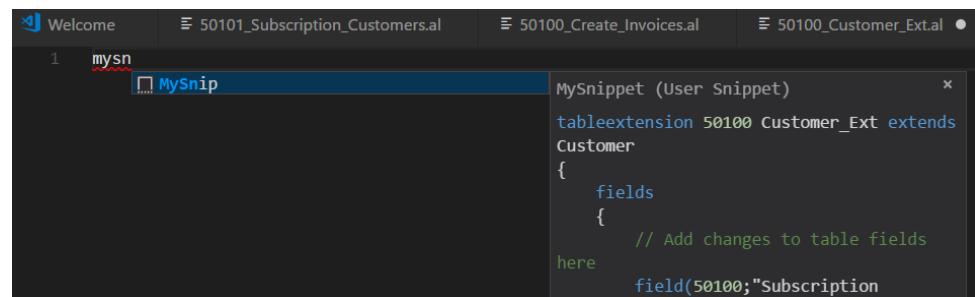
### Creating own snippets

Creating snippets is possible, but not as simple as it could be, unless the **Snippet-Creator extension by Nikita Kunevich** is installed in VS Code. Then all there is to do is to mark the code to include, press **Ctrl+Shift+P** and select **Create Snippet**:



Give information about language (al), Name, Shortcut and description.

Then create a new file and test the snippet:



Now the new snippet appears in the File/Preferences/User Snippets/AL:



# Differences in the Development Environments

Coming from the Dynamics NAV Development Environment and C/SIDE, there are some differences and optimizations that you should familiarize yourself with. The following sections go through these changes.

## Data types

C/SIDE	AL Language Development Environment
Dates are parsed based on culture settings.	Locale independent and supports only: yyyy-mm-ddD.
Boolean values could be expressed as <b>yes/no</b> .	Boolean values are expressed as <b>true/false</b> .
For tables, integers could allow decimal values. For example, 5.0 converts to an integer, 5.4 throws an error at runtime.	For tables, Min, Max, InitValue numbers with a fraction are expressed as decimal, thus they are not a valid integer data type.
The largest constant integer could be 99999999999999.	Transforms to 999'999'999'999'999.0, a decimal value. In AL, this can be expressed as 99999999999999.0 or 99999999999999L.

## Syntax updates

C/SIDE	AL Language Development Environment
The token for multilanguage comment is @@@.	A multilanguage comment is marked with Comment.
Supports TryFunction on code developed in C/SIDE.	Supports calling referenced TryFunctions from W1.
Reserved commands are always uppercase	Reserved commands stays in the same as it was entered

**Several properties have been renamed, to mention some:**

C/SIDE	AL Language Development Environment
AutoFormatExpr	AutoFormatExpression
DataCaptionExpr	DataCaptionExpression
Layout	GridLayout
ProviderID	Provider

## Multilanguage properties

With the introduction of .xliff files, the ML properties, such as CaptionML and TooltipML will be deprecated in a later version. Use the equivalent properties instead, such as Caption and Tooltip, then make sure the manifest is set up to generate the /Translations folder and use the generated .xliff files for translations of the extension.

## Pages

The ActionContainer elements in AL have been renamed; the following table lists the renamed elements:

C/SIDE	AL Language Development Environment
ActionItems	Processing
ActivityButtons	Sections
HomeItems	Embedding
NewDocumentItems	Creation
RelatedInformation	Navigation
Reports	Reporting

For instance, area(Sections) can be defined inside the actions section of the page. Likewise, Container and ContainerType elements in C/SIDE have been renamed to

area(Content|FactBoxes|RoleCenter) and can be defined inside the layout section of the page.

### Naming

Controls, actions, and methods names must be unique on pages. In C/SIDE, you could create a Part control with the same name as a method, which would give you an error at runtime. This is now prevented, by disallowing duplicates. Similarly, trigger and trigger event names are disallowed on matching application object types. Likewise, actions and fields could have same names before, but that would have prevented page testability access, and will now throw a compilation error.



AI extension names and field names in extensions must be prefixed with a code that is at least three characters long. This is to separate the name from similar extensions to the same table. In solutions published through App Source, a prefix will be assigned to the extension.

### Property dependencies

Some properties require that you set another property. An example is PromotedCategory, which requires that you have enabled the property Promoted. The following table lists some of the properties that have this dependency.

Property	Depends on the property...
PromotedCategory	Promoted
PromotedIsBig	Promoted
ValidateTableRelation	TableRelation
SourceTableTemporary	SourceTable
RunPageMode	RunObject

### Limited functionality

The InitValue property of type Duration is not allowed in new development environment.

The InitValue of type DateTime only allows for the value 0DT.

### Discontinued functionality

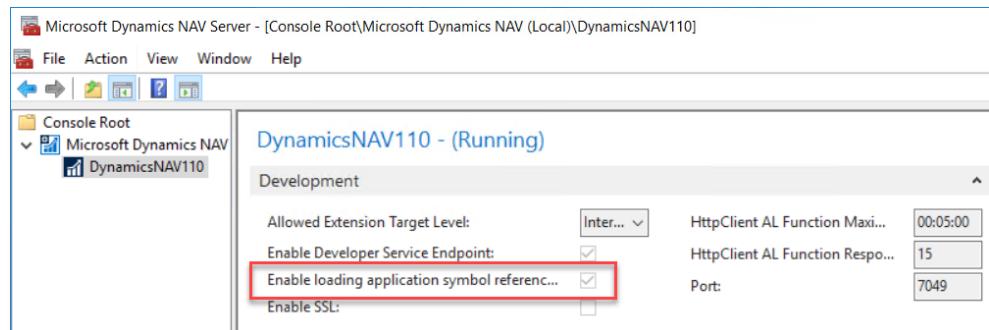
All variables of the type DotNet is not allowed in new development environment.

## C/Side and AL Side-by-side development

Dynamics NAV 2018 supports development using both C/SIDE and AL, as well as Designer side-by-side. When new objects are added or changed in C/SIDE these changes must be reflected in the symbol download in Visual Studio Code using the AL Language extension. To enable this reflection, a new command and argument has been added to finsql.exe called **generatesymbolreference**.

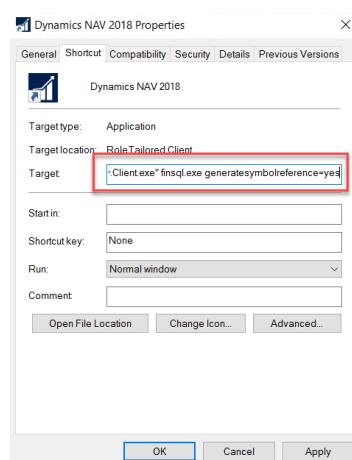
### Dynamics NAV Server setting

To allow any symbol generation, you must also enable the *EnableSymbolLoadingAtServerStartup* setting in the Dynamics NAV server settings. If the setting is not enabled, the **generatesymbolreference** setting does not have any effect.



### Continuously generate symbols each time you compile objects in C/SIDE

Use **generatesymbolreference** set to *yes* as a command line argument each time you start finsql.exe to have all compilations add a symbol reference to the Object Metadata table. The default setting of the argument is *no*.



### Get started generating symbols and compiling all objects

Use the **generatesymbolreference** command specified with the database and server name. This command will add symbol references to the Object Metadata table for the specified database.

#### Syntax example

 Copy

```
finsql.exe Command=generatesymbolreference, Database="Demo Database NAV (11-0)", ServerName=.\\NAVDEMO
```

This is a lengthy operation and will block the objects in the database for other modification.

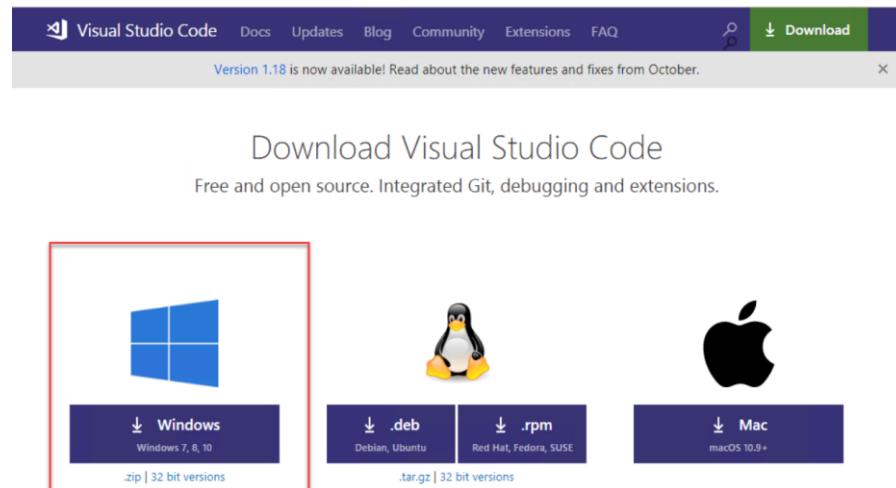


Alternatively, all objects can be compiled after opening the Development Environment with the **generatesymbolreference=yes** parameter

# Getting Started with AL

Download and install VS Code from this website:

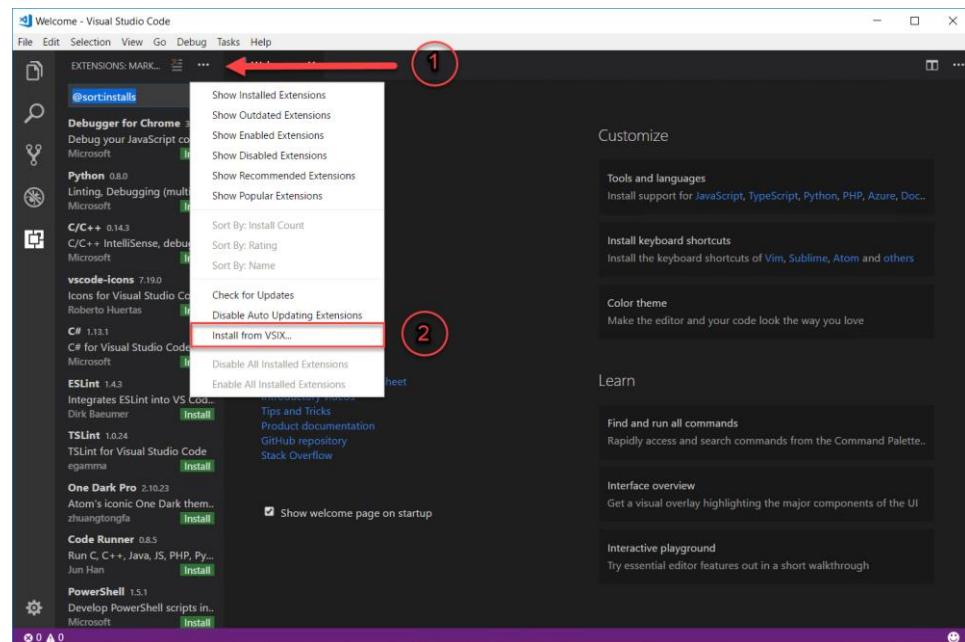
<https://code.visualstudio.com/download>



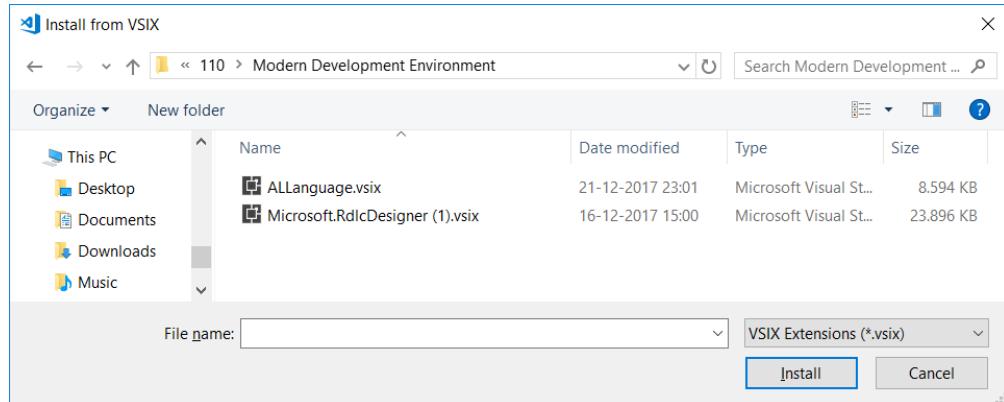
After installing VS Code, it is necessary to install the AL Language extension. The extension can be installed using the downloaded version from the cloud, but the only way to ensure the correct version, is to install the AL Language from the installed Dynamics NAV version.

It is located in:

**C:\Program Files (x86)\Dynamics NAV\110\Modern Development Environment**

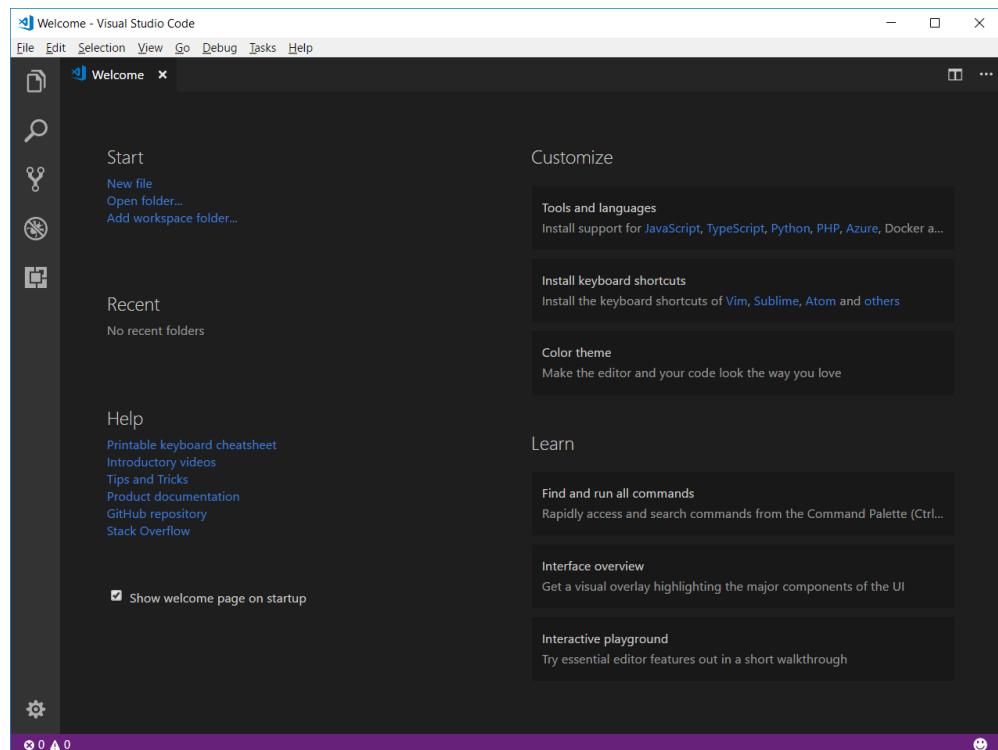


## Solution Development in Visual Studio Code



After the installation, VS Code is ready to use.

## The Visual Studio Code Editor



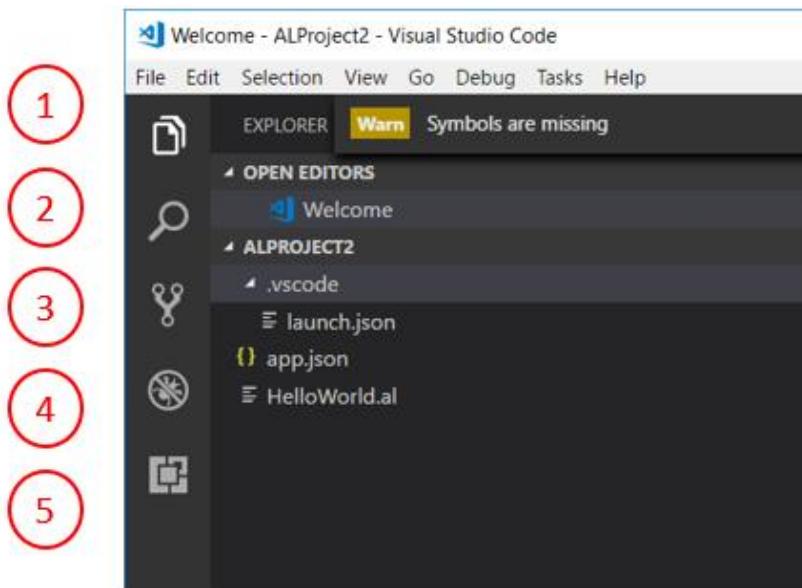
To start a new project, it is necessary to create a new folder structure and a number of settings files. The folder is called a workspace and it will include all files in the project and therefore, files added to the folder are automatically included in the workspace.



Use **Ctrl+Shift+P** and type: **AL:Go!** To initiate a new project, automatically creating the necessary folder structure and settings files

Name the project **SD Seminar**.

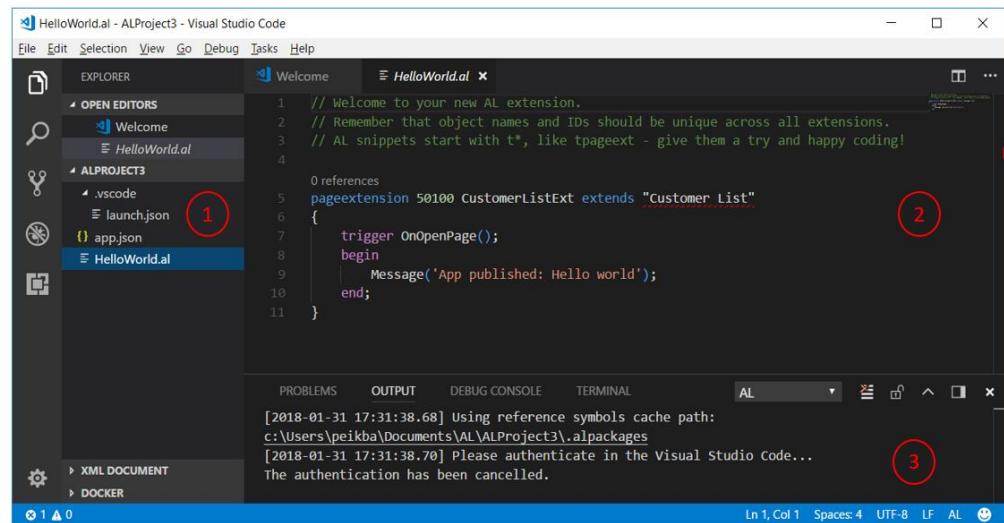
To the left a number of icons, open different windows.



- 1) The explorer window to show all folders and files used in the project.
- 2) The search function to find any text in any files in the project.  
It can also be used for find and replace.
- 3) The source control window. VS Code can integrate seamlessly to source control systems like Team Foundation Server and Git.  
See the chapter on how to set up a Git repository
- 4) The standard Debugger is currently not integrated to the Dynamics 365 BC debugger, and can therefore not be used with AL programming.
- 5) The Extensions window is where it is possible to add extra languages. A number of interesting extensions could be:
  - Docker
  - JSON Tools
  - PowerShell
  - SML Language Support
  - Snippet-creator by Nikita Kunevich

## Solution Development in Visual Studio Code

When clicking the HelloWorld.al file, the view changes:

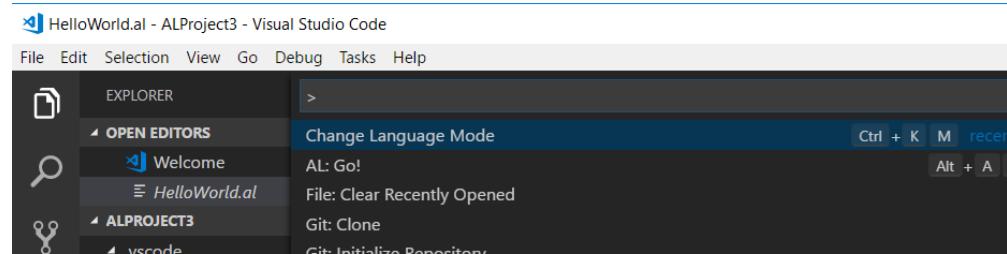


- 1) The explorer window
- 2) The selected file
- 3) The output window showing the latest result or error message

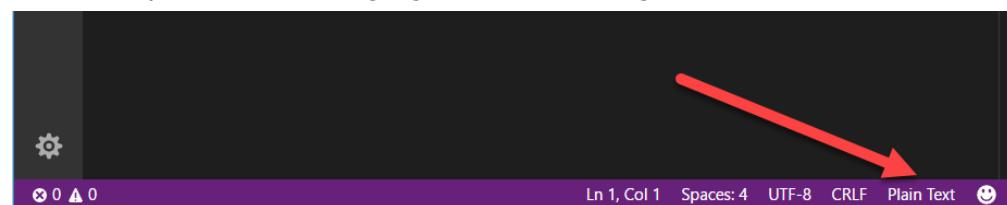
## Automatic language selection

Opening or creating a file with the .al extension will automatically enable Visual Studio Code to “understand” the AL language.

If no file extension has been defined it is necessary to change language to AL by clicking the **Ctrl+Shift+P** and selecting the AL language module.



Another way is to click the language selector in the right side of the status bar.

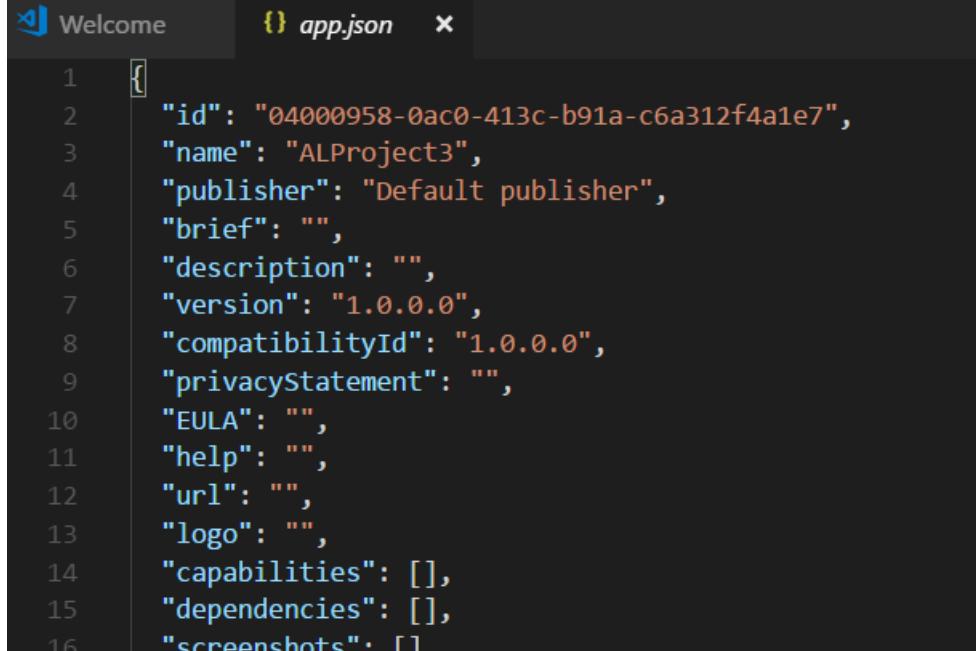


## JSON file settings

There are two JSON files in the project; the app.json file and the launch.json file. The files are automatically generated for your project when using the >AL:Go!

command in the Command Palette (Ctrl+Shift+P).

## The app.json file



```

1  [
2   "id": "04000958-0ac0-413c-b91a-c6a312f4a1e7",
3   "name": "ALProject3",
4   "publisher": "Default publisher",
5   "brief": "",
6   "description": "",
7   "version": "1.0.0.0",
8   "compatibilityId": "1.0.0.0",
9   "privacyStatement": "",
10  "EULA": "",
11  "help": "",
12  "url": "",
13  "logo": "",
14  "capabilities": [],
15  "dependencies": [],
16  "screenshots": []

```

The app.json file is the manifest for the

Setting	Mandatory	Value
id	Yes	The unique ID of the extension. When app.json file is automatically created, the ID is set to a new GUID value.
name	Yes	The unique extension name.
publisher	Yes	The name of your publisher, for example: <b>NAV Partner, LLC</b>
brief	No, but required for AppSource submission	Short description of the extension.
description	No, but required for AppSource submission	Longer description of the extension.
version	Yes	The version of the app package.

## Solution Development in Visual Studio Code

---

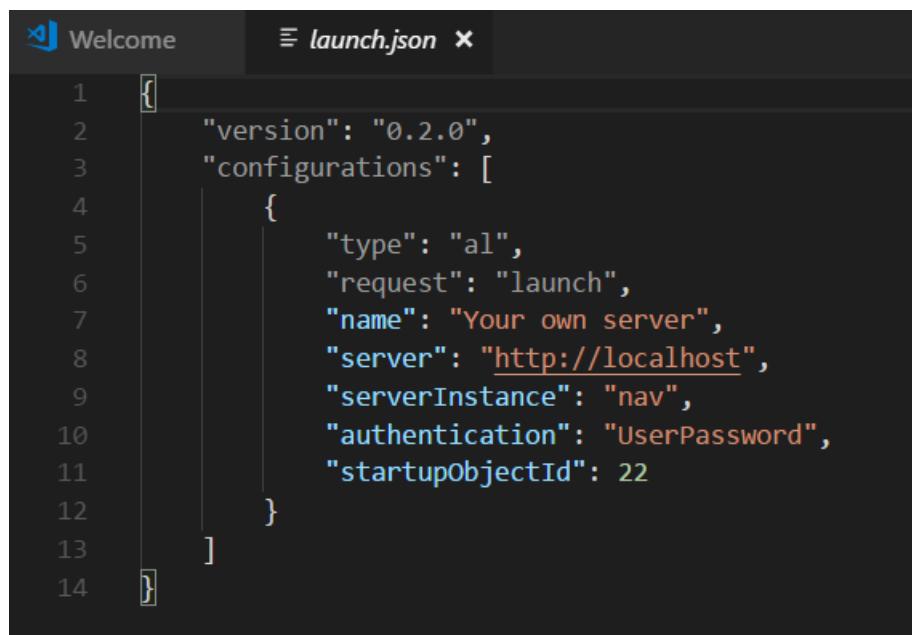
Setting	Mandatory	Value
privacyStatement	No, but required for AppSource submission	URL to the privacy statement for the extension.
EULA	No, but required for AppSource submission	URL to the license terms for the extension.
help	No, but required for AppSource submission	URL to the help for the extension.
url	No	URL of the extension package.
logo	No, but required for AppSource submission	URL to the logo for the extension package.
dependencies	No	List of dependencies for the extension package. For example: "dependencies": [ { "appId": "4805fd15-75a5-46a2-952f-39c1c4eab821", "name": "WeatherLibrary", "publisher": "Microsoft", "version": "1.0.0.0" } ],
screenshots	No	Relative paths to any screenshots that should be in the extension package.
platform	Yes, if system tables are referenced in the extension	The minimum supported version of the platform symbol package file, for example: "11.0.0.0".
application	** (1)	The minimum supported version, for example: "application": "11.0.0.0"
idRange	Yes	A range for application object IDs. For all objects outside the range, a compilation error will be raised.

Setting	Mandatory	Value
showMyCode	No	This is by default set to false and not visible in the manifest. To enable viewing the source code when debugging into an extension, add the following setting: "showMyCode": true
target	No	By default this is Extension. For Dynamics NAV, you can set this to Internal to get access to otherwise restricted APIs. The Dynamics NAV Server setting must then also be set to Internal.

\*\*1 Yes, if base application objects are extended or referenced. The AL package will be compiled against the application that is present on the server that you connect to. This allows you to write a single AL extension for multiple country versions as long as you do not depend on country-specific code. If you do depend on country-specific code you should only try to compile your app against a server set up for that country.

### Launch.json file

The following table describes the settings in the launch.json file. The launch.json file has two configurations depending on whether the extension is published to a local server or to the cloud.



```

1  [
2    {
3      "version": "0.2.0",
4      "configurations": [
5        {
6          "type": "al",
7          "request": "launch",
8          "name": "Your own server",
9          "server": "http://localhost",
10         "serverInstance": "nav",
11         "authentication": "UserPassword",
12         "startupObjectId": 22
13       }
14     ]
}

```

### Publish to local server settings

Setting	Mandatory	Value
name	Yes	"Publish to your own server"
type	Yes	Must be set to ".al". Required by Visual Studio Code.
request	Yes	Request type of the configuration. Must be set to Launch. Required by Visual Studio Code.
server	Yes	The HTTP URL of your server, for example: "http://localhost"
serverInstance	Yes	The instance name of your server, for example: "NAV"
authentication	Yes	Specifies the server authentication method.
startupObjectId	No	Specifies the ID of the object to open after publishing.
startupObjectType	No	Specifies the Type of the object to open after publishing.
schemaUpdateMode	No	Specifies the data synchronization mode when you publish an extension to the development server, for example: "schemaUpdateMode": "Synchronize Recreate"
		The default value is Synchronize.

### Publish to cloud settings

Setting	Mandatory	Value
name	Yes	"Publish to Microsoft cloud sandbox"
type	Yes	Must be set to ".al". Required by Visual Studio Code.
request	Yes	Request type of the configuration. Must be set to launch. Required by Visual Studio Code.
startupObjectId	Yes	Specifies the ID of the object to open after publishing. Only objects of type Page are currently supported.
serverInstance	Yes	The instance name of your server, for example: "US"

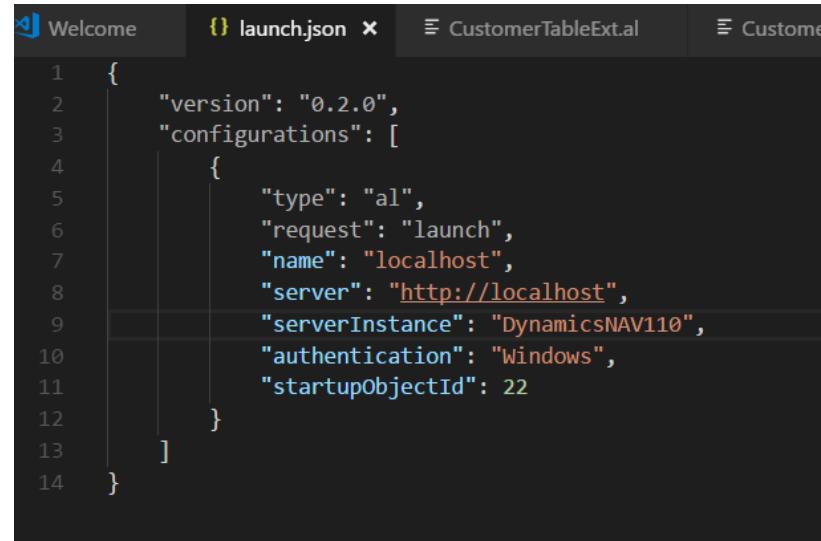
## The platform symbol file

The platform symbol file contains all of the base app objects that your extension builds on. If the symbol file is not present, you will be prompted to download it.

### The first test

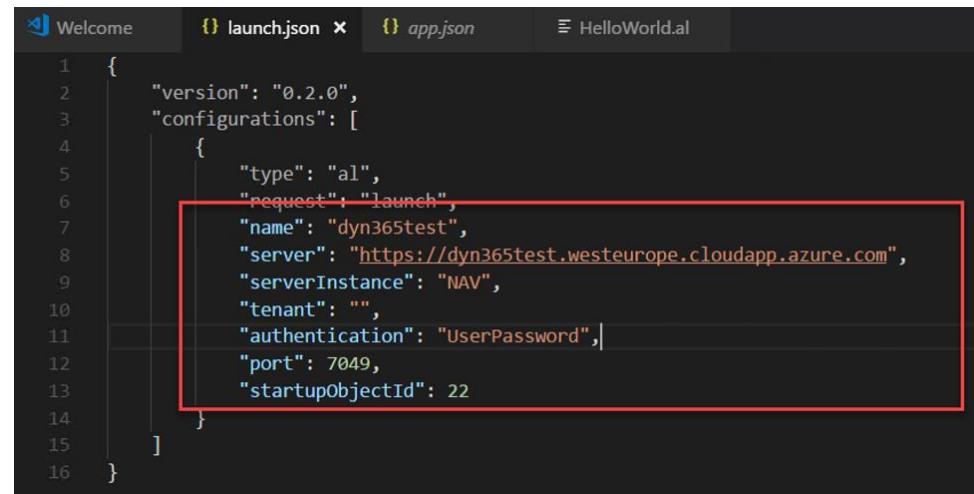
#### Connecting to the database

First thing is to define the connection to the server. It is important that the name equals the server name. If running on the on-premise version of Dynamics NAV 2018, the launch.json file could look like this:



```
1  {
2      "version": "0.2.0",
3      "configurations": [
4          {
5              "type": "al",
6              "request": "launch",
7              "name": "localhost",
8              "server": "http://localhost",
9              "serverInstance": "DynamicsNAV110",
10             "authentication": "Windows",
11             "startupObjectId": 22
12         }
13     ]
14 }
```

Connecting to a Dynamics 365 BC in the cloud, the Launch.json could look like this:



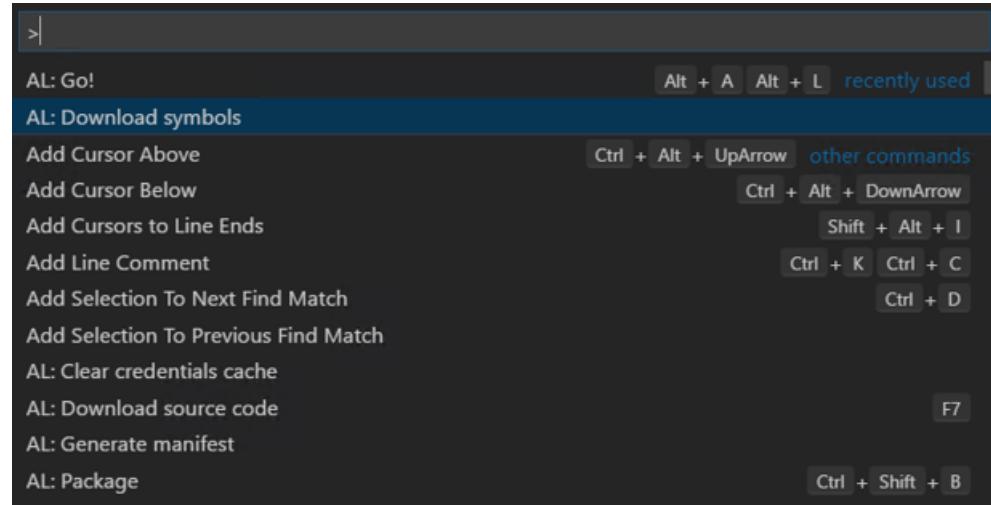
```
1  {
2      "version": "0.2.0",
3      "configurations": [
4          {
5              "type": "al",
6              "request": "launch",
7              "name": "dyn365test",
8              "server": "https://dyn365test.westeurope.cloudapp.azure.com",
9              "serverInstance": "NAV",
10             "tenant": "",
11             "authentication": "UserPassword",
12             "port": 7049,
13             "startupObjectId": 22
14         }
15     ]
16 }
```

Click Ctrl+S to save the file before proceeding.

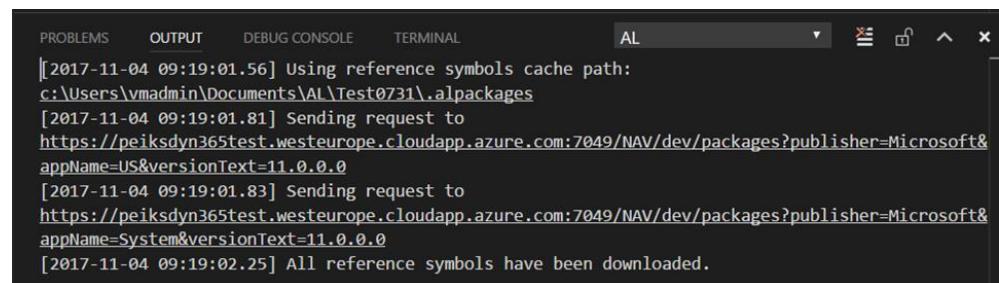
### Downloading the symbols

The symbols are a reference to all the Dynamics 365 BC snippets and all objects in the Dynamics 365 BC database. They need to be downloaded.

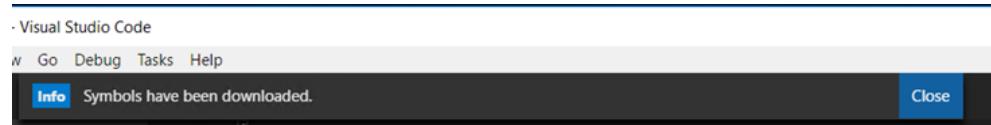
To download the symbols for the project:



If the authentication method is **UserPassword** then give User id and press Enter then give your Password and press enter.

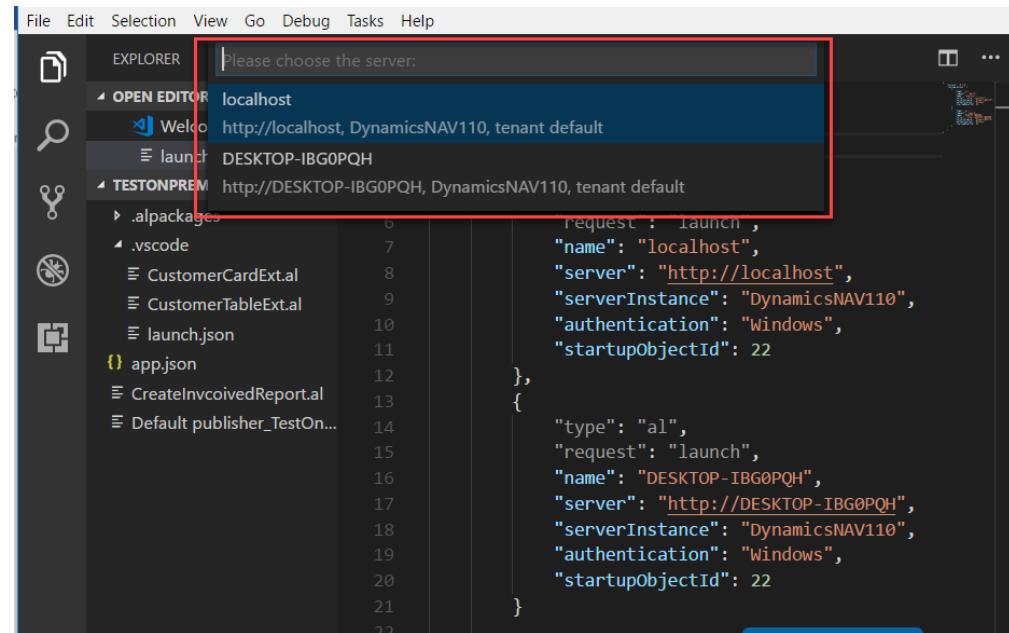


The output windows look like this and in the top of the editor, the message below will appear.



Now the symbols have been downloaded correctly and the editor is ready to use.

It is possible to have multiple configurations in the Launch.json file. In that case the VS Code editor will prompt for the server on downloading the symbols.



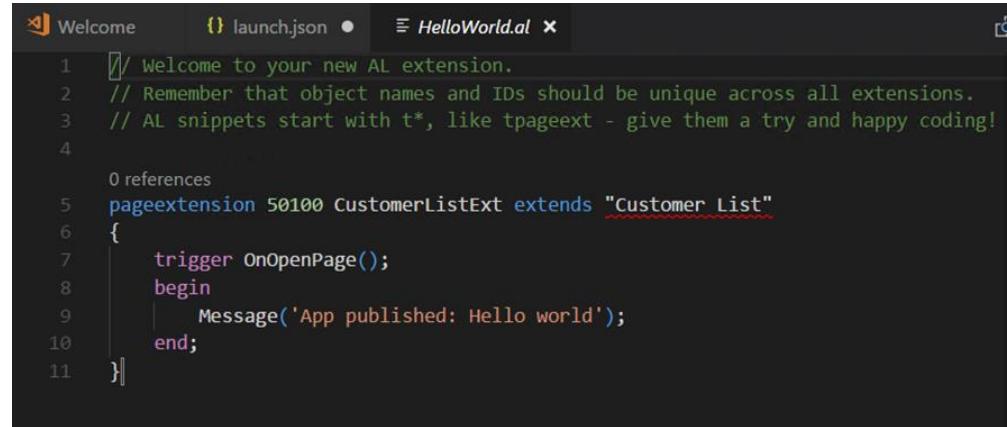
The screenshot shows the Visual Studio Code interface. The top menu bar includes File, Edit, Selection, View, Go, Debug, Tasks, and Help. The Explorer sidebar on the left lists several items: OPEN EDITOR (localhost, http://localhost, DynamicsNAV110, tenant default), launch (DESKTOP-IBG0PQH), and TESTONPREM (http://DESKTOP-IBG0PQH, DynamicsNAV110, tenant default). The main editor area displays a JSON file with the following content:

```
request : launch ,
"name": "localhost",
"server": "http://localhost",
"serverInstance": "DynamicsNAV110",
"authentication": "Windows",
"startupObjectId": 22
},
{
  "type": "al",
  "request": "launch",
  "name": "DESKTOP-IBG0PQH",
  "server": "http://DESKTOP-IBG0PQH",
  "serverInstance": "DynamicsNAV110",
  "authentication": "Windows",
  "startupObjectId": 22
}
```

The status bar at the bottom of the interface shows the message "Please choose the server:" followed by a dropdown menu with two options: "localhost" and "DESKTOP-IBG0PQH".

### Running the first test

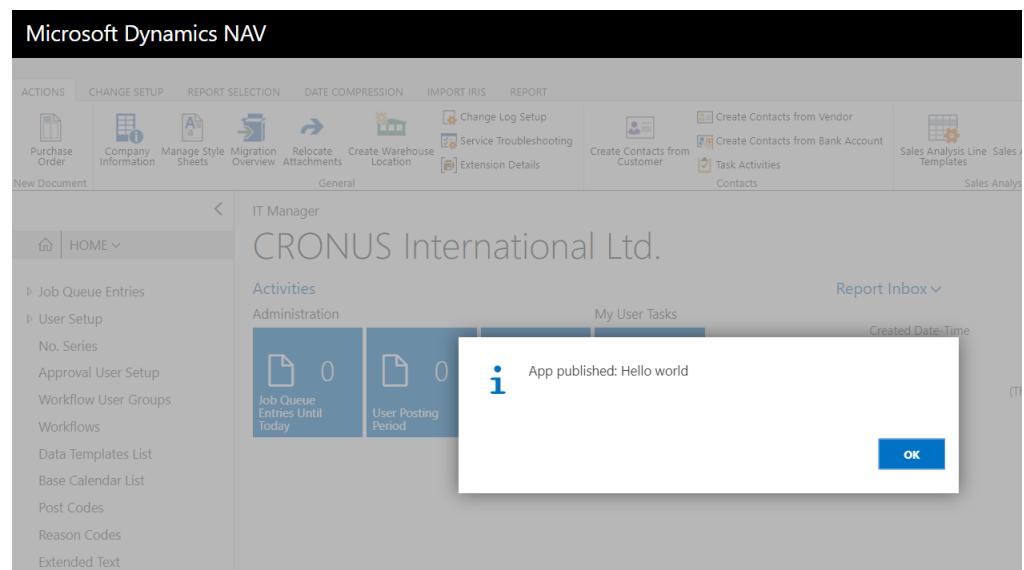
HelloWorld.al is automatically generated .al file to show how an extension could be made:



```
1 // Welcome to your new AL extension.
2 // Remember that object names and IDs should be unique across all extensions.
3 // AL snippets start with t*, like tpageext - give them a try and happy coding!
4
5 pageextension 50100 CustomerListExt extends "Customer List"
6 {
7     trigger OnOpenPage();
8     begin
9         Message('App published: Hello world');
10    end;
11 }
```

When pressing F5 (Debug) the package is created and the extension is sent to the server. Lastly, the web client is opened and page 22 is run.

That will confirm that Visual Studio Code is ready to develop extensions to the Dynamics 365 BC database.



### Lab 2.1 Publish a test extension to Dynamics 365 BC

#### Prerequisite to this lab:

In order to complete this lab, there are a number of prerequisites:

- A Dynamics 365 BC must be available. Both the cloud solution and on-premise version will work.
- There must be access to the Dynamics 365 BC as **super user**
- Visual Studio Code must be installed locally
- The AL Language must be installed in VS Code
- The **“Enable Develop Service Endpoint”** and **“Enable loading application symbol references at server startup”** settings must be activated in the Dynamics 365 BC service tier.

#### High-Level steps:

- Create a new project with the name: “SD Seminar”
- Change the launch.json file to connect to the Dynamics 365 BC database
- Change the app.json file meet the minimum requirements for a Dynamics 365 BC extension. Add “D.E. Veloper” as the Publisher and “SD Seminar” as the Extension Name
- Publish the extension to the database
- Verify that the web client opens and that the “Hello World” message appears
- Verify that the “SD Seminar” extension is available in the Extension Management window.
- Uninstall the “SD Seminar” extension.
- Unpublish the “SD Seminar” extension.
- Verify that the “SD Seminar” extension is gone
- Remove the HelloWorld.al file from the project
- Remove the **“StartupObjectId”: 22** line and the preceding comma.

#### Detailed steps:

- 1) Open VS Code
- 2) Click **Ctrl+Shift+P** or click the menu **View/Command Palette**
- 3) Write or select the **AL:Go!** Command
- 4) Replace **Project\_** with **SD Seminar** and press **Enter**
- 5) Expand the **.vscode** folder
- 6) Open the **launch.json** file
- 7) In the **name** setting, replace “Your own server” with the server name of Dynamics 365 BC server without http:// and any trailing information

```
"name": "dyn365test",
"server": "https://dyn365test.westeurope.cloudapp.azure.com",
```

Or

```
"name": "localhost",
"server": "http://localhost",
```

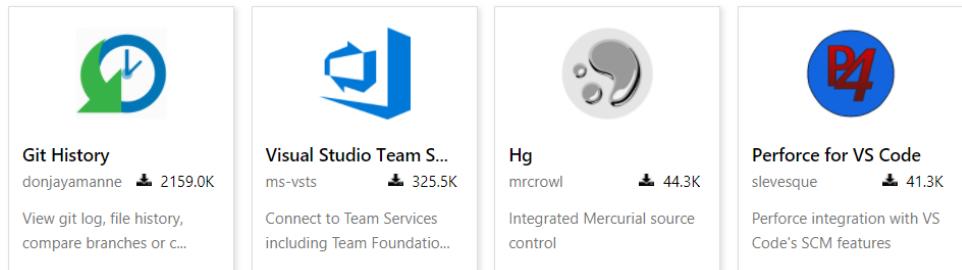
- 8) Enter the URL to the server as shown above
- 9) Change the *ServerInstance* to **DynamicsNAV110**
- 10) Change the *Authentication* to **Windows**
- 11) Then save the file by clicking **Ctrl+S**
- 12) Download the symbols by clicking **Ctrl+Shift+P** or clicking the menu **View/Command Palette**
- 13) Write or select the **AL:Download Symbols** Command
- 14) If Dynamics 365 BC requires user id and password, provide the user id and press **Enter** then give the password and press **Enter** again.
- 15) Verify that the message "All reference symbols have been downloaded" is shown In the Output Window
- 16) Open the **app.json** file
- 17) Change the **publisher** setting to be "D.E. Veloper"
- 18) Change the **name** setting to be "**SD Seminar**"
- 19) Then save the file by clicking **Ctrl+S**
- 20) Open the **HelloWorld.al** file
- 21) Change the "**App published: Hello World**" text to something else
- 22) Then save the file by clicking **Ctrl+S**
- 23) Publish the extension by clicking **F5** (Start Debugging)
- 24) Verify that the message **Success: The package has been published to the server** is shown in the Output Window, and that the Web client opens with a message box.
- 25) Click the search  icon in the Web client and search for the "**Extension Management**" page
- 26) Verify that the "**SD Seminar**" extension is available in the page
- 27) Uninstall the "**SD Seminar**" extension
- 28) Unpublish the "**SD Seminar**" extension
- 29) Verify that the "**SD Seminar**" extension is no longer in the list of extensions
- 30) Close the Web client
- 31) Go back to VS Code
- 32) Stop the debugger on the red button in the debug panel:  

- 33) Delete the "HelloWorld.al" file because it will conflict with the later solution.
- 34) Now open the launch.json file and delete the "**StartupObjectId": 22** line and the comma above.

# Module 3: Setting up a Git repository

## Using Version Control in VS Code

Visual Studio Code has integrated source control and includes Git support out-of-the-box. Many other source control providers are available through extensions on the VS Code Marketplace.



## SCM Providers

VS Code has support for handling multiple Source Control providers simultaneously. For example, you can open multiple Git repositories alongside your TFS local workspace and seamlessly work across your projects. The **SOURCE CONTROL PROVIDERS** list of the Source Control view (**Ctrl+Shift+G**) shows the detected providers and repositories and you can scope the display of your changes by selecting a specific provider.

## Git support

VS Code ships with a Git source control manager (SCM) extension. Most of the source control UI and work flows are common across other SCM extensions so reading about the Git support will help you understand how to use another provider.

### What is Git?

According to Wikipedia, Git is a version control system for tracking changes in computer files and coordinating work on those files among multiple people. It is primarily used for source code management in software development, but it can be used to keep track of changes in any set of files. As a distributed revision control system, it is aimed at speed, data integrity, and support for distributed, non-linear workflows.

Git was created by Linus Torvalds in 2005 for development of the Linux kernel, with other kernel developers contributing to its initial development. Its current maintainer since 2005 is Junio Hamano.

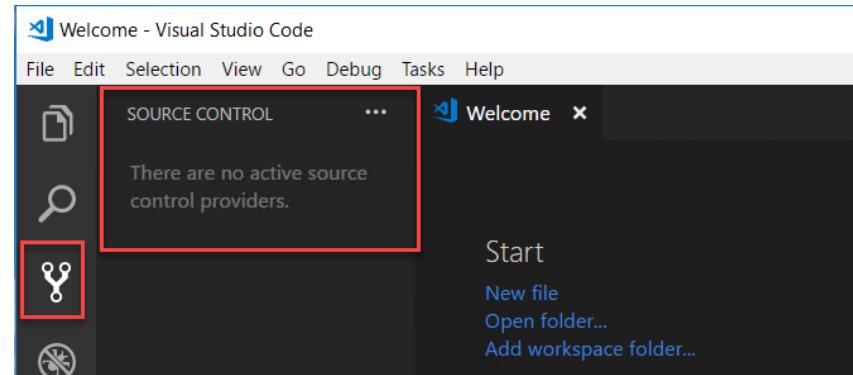
As with most other distributed version control systems, and unlike most client-server systems, every Git directory on every computer is a full-fledged repository with complete history and full version tracking abilities, independent of network access or a central server.

Git is free software distributed under the terms of the GNU General Public License version 2.

Acquiring an account at Github is free, but only integrating to a public repository. For a private repository it is necessary to acquire a paid subscription. Git is here demonstrated integrating to Github, but other solutions like BitBucket.org or GitLab.com would work too.

## Installing Git and initializing the repository

Git must be installed on the computer and clicking the Source Control window only gives this message:

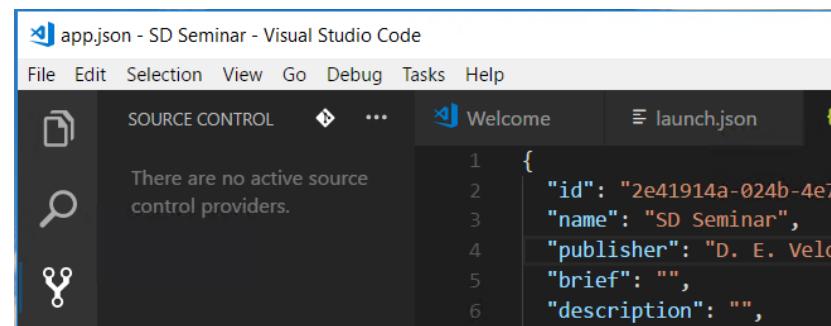


Therefore, Git must be downloaded, and the best place to find it is here:

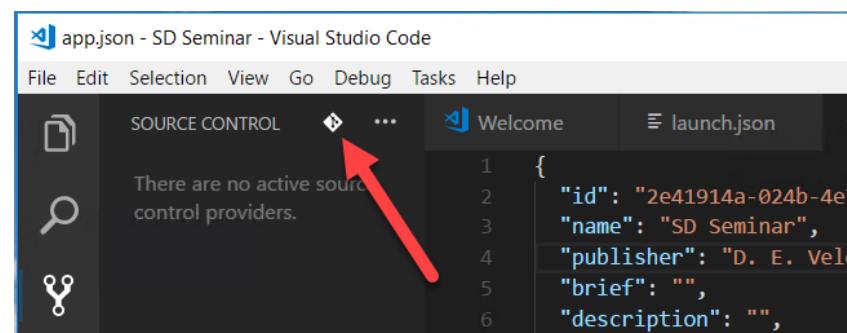
<http://git-scm.com/download/win>

Run the installer with a next-next installation and Git has been installed on the computer.

Now clicking the Source Control windows this message appears:



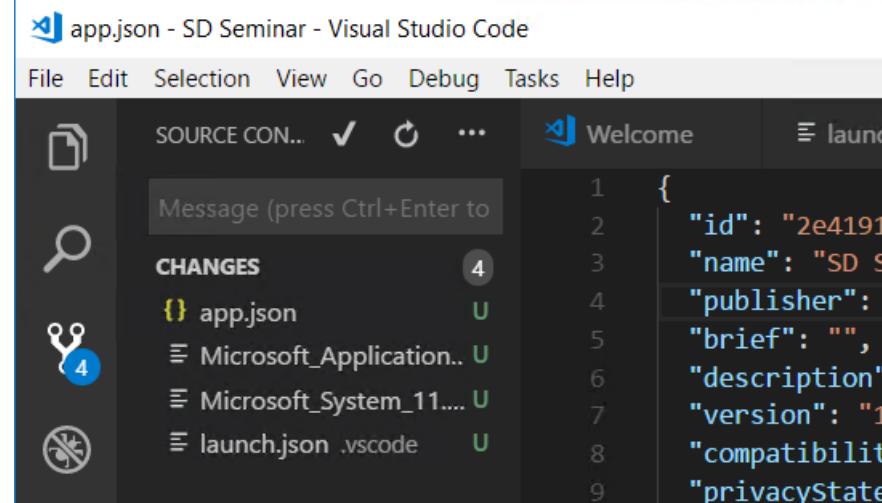
This means that VS Code is almost ready to use.



After opening the **SD Seminar** workspace, simply click the Initialize Repository

button or press **Ctrl+Shift+P** and type **Git: Initialize repository**. Then select the **SD Seminar** folder.

Now the source control looks like this:

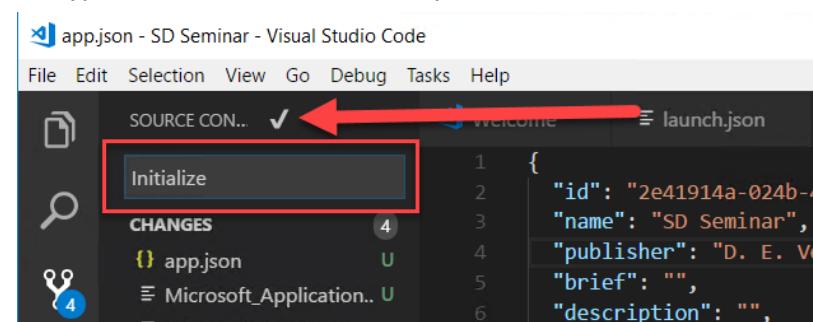


The blue badge with the 4 means that there are 4 uncommitted documents in the folder and it is therefore to commit them.

Every commit is identified by a text. This text will appear on all logs and in the Github repository.

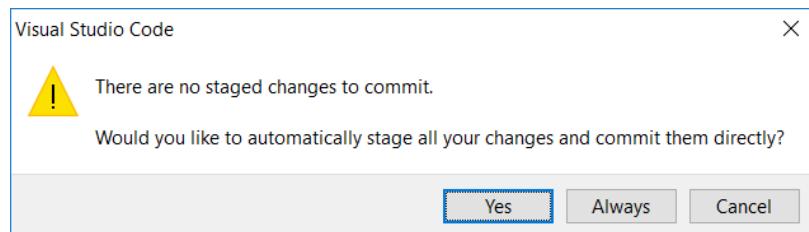
Committing a change means adding it to the local repository. It is possible to keep the changes in local repository only. However, that will deprive you the advantages of an off-site repository. To commit the changes Later, it will be necessary to push the changes to the external repository.

So, type a text in the field to identify the commit and click the checkmark.



## Solution Development in Visual Studio Code

At the message there are no staged changes to commit, click **Always**.



This means that when there are no staged changes, all working tree changes are committed.

Now you get an error:



Which is because you have not identified yourself to Git. Therefore, it is necessary to create an account at:

<https://github.com/>.

Then it is necessary to connect VS Code to the on-line Git repository. Use the Terminal window in the bottom of VS Code:

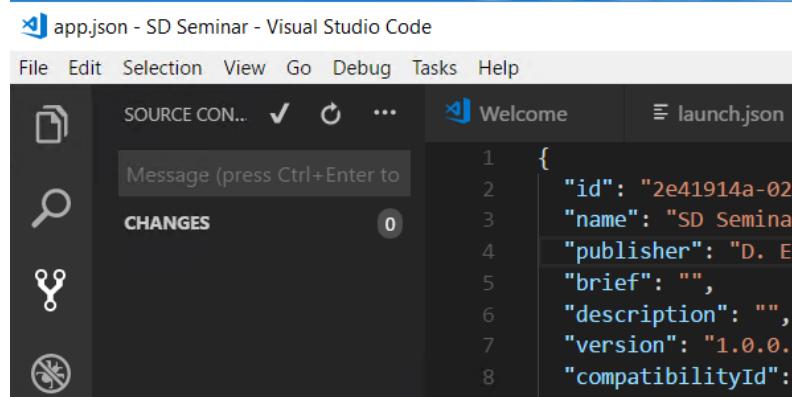
```
Windows PowerShell
Copyright (C) 2016 Microsoft Corporation. All rights reserved.

PS C:\> Git config --global user.email "DeVelooper@gmail.com"

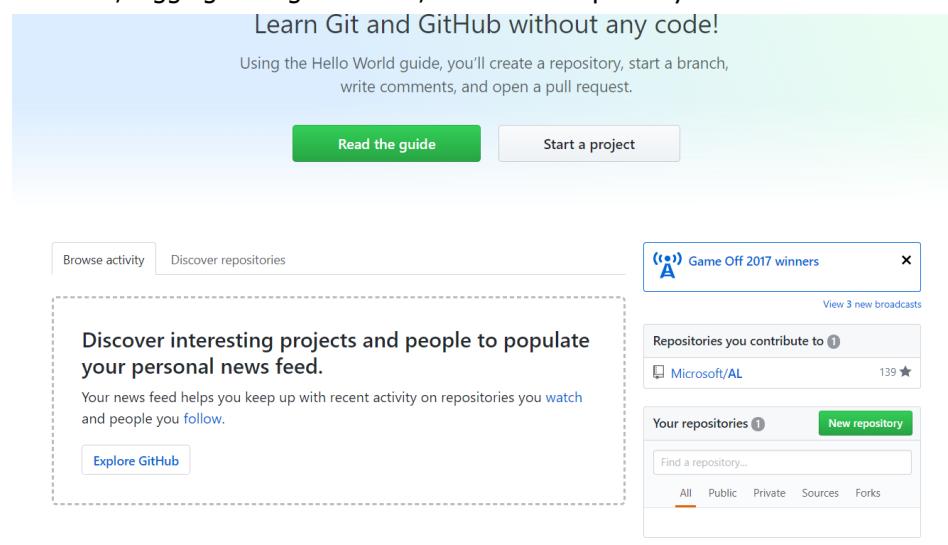
Windows PowerShell
Copyright (C) 2016 Microsoft Corporation. All rights reserved.

PS C:\> Git config --global user.name "DeVelooper"
```

Now, clicking the Source Control Checkmark, all files disappear from the Source Control window.



However, logging in to [github.com](https://github.com), there is no repository called SD Seminar.

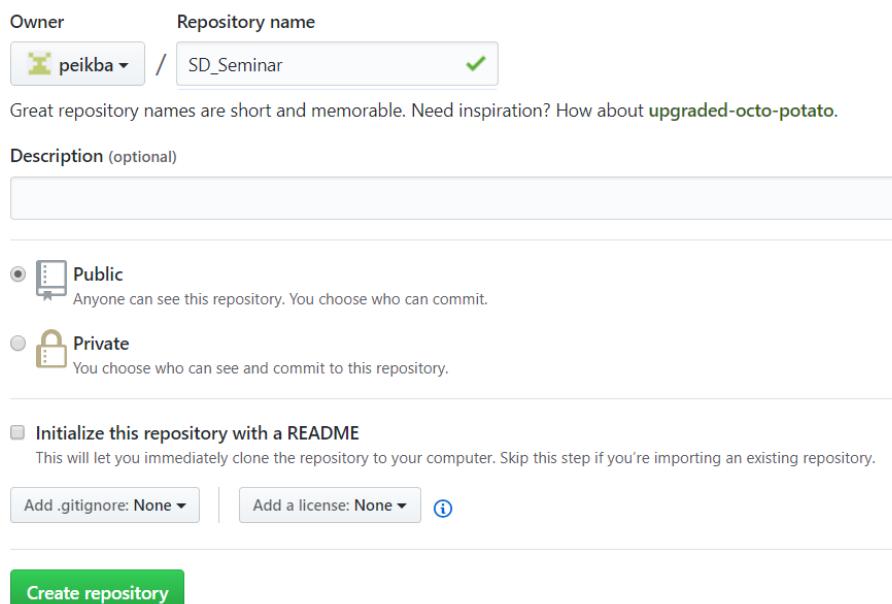


The screenshot shows the GitHub homepage. At the top, a banner reads "Learn Git and GitHub without any code!" with a "Read the guide" button. Below the banner, a "Discover interesting projects and people to populate your personal news feed." section is visible, along with a "Explore GitHub" button. To the right, a "Game Off 2017 winners" broadcast is shown with a "View 3 new broadcasts" link. The main search interface includes a "Repositories you contribute to" section for Microsoft/AL (139 stars), a "Your repositories" section with a "New repository" button, and a search bar with filters for All, Public, Private, Sources, and Forks.

Click the New Repository button and enter SD\_Seminar as the name

### Create a new repository

A repository contains all the files for your project, including the revision history.



The screenshot shows the "Create a new repository" form. It includes fields for "Owner" (peikba), "Repository name" (SD\_Seminar), and a "Description (optional)" text area. Below these are options for "Public" (selected) and "Private" repository types, with a note that anyone can see a public repository. There is also a checkbox for "Initialize this repository with a README" and buttons for "Add .gitignore: None" and "Add a license: None". At the bottom is a large green "Create repository" button.

Click Create Repository.

## Solution Development in Visual Studio Code

Now all that is needed now, is to copy the two commands, and run them in VS Code Terminal window:

Quick setup — if you've done this kind of thing before

or   [https://github.com/peikba/SD\\_Seminar.git](https://github.com/peikba/SD_Seminar.git)

We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).

...or create a new repository on the command line

```
echo "# SD_Seminar" >> README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin https://github.com/peikba/SD_Seminar.git
git push -u origin master
```

...or push an existing repository from the command line

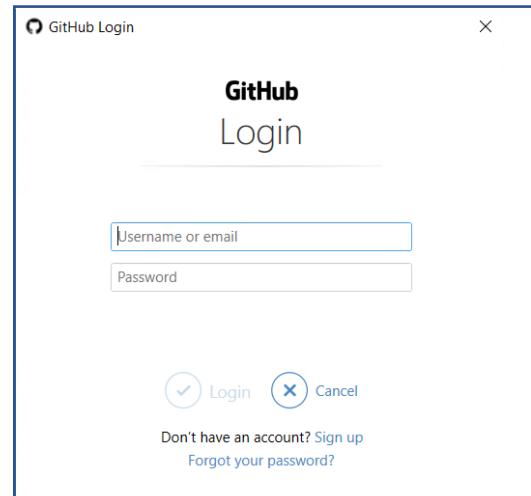
```
git remote add origin https://github.com/peikba/SD_Seminar.git
git push -u origin master
```

Windows PowerShell

Copyright (C) 2016 Microsoft Corporation. All rights reserved.

```
PS C:\> Git remote add origin https://github.com/peikba/
SD_Seminar.git
PS C:\> Git push -u origin master
```

After logging in to Github the Repository is ready



```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
PS C:\Users\Admin\Documents\AL\SD Seminar> git remote add origin https://github.com/peikba/SD_Seminar.git
PS C:\Users\Admin\Documents\AL\SD Seminar> git push -u origin master
Counting objects: 8, done.
Compressing objects: 100% (7/7), done.
Writing objects: 100% (8/8), 3.52 MiB | 576.00 KiB/s, done.
Total 8 (delta 0), reused 0 (delta 0)
To https://github.com/peikba/SD_Seminar.git
 * [new branch]      master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.
PS C:\Users\Admin\Documents\AL\SD Seminar> git
  
```

Now looking at the Github account, the repository is here:

No description, website, or topics provided. [Edit](#)

1 commit 1 branch 0 releases 1 contributor

Branch: master [New pull request](#) [Create new file](#) [Upload files](#) [Find file](#) [Clone or download](#)

peikba Initialize [Latest commit 51354cb 19 minutes ago](#)

File	Commit	Time
.alpackages	Initialize	19 minutes ago
.vscode	Initialize	19 minutes ago
app.json	Initialize	19 minutes ago

This is a public repository, which it is available to everybody. If a more private repository is needed, it is necessary to purchase a subscription at Github.

Now the repository is ready to use, which means that all changes to the files will be tracked. Here a change has been made to the app.json file, and it has been saved. Now the Source Control icon clearly remarks that an uncommitted change has been made. To see the change that has been made, click the filename in the Source Control window and both files are shown with all changes:

File Edit Selection View Go Debug Tasks Help

SOURCE CON... CHANGES app.json

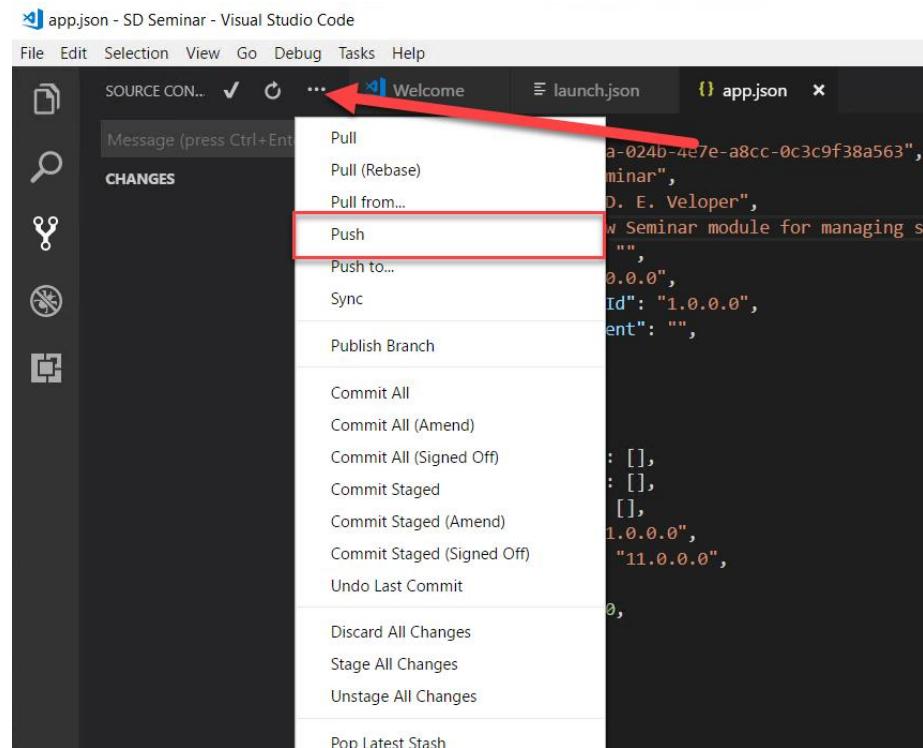
```

1 [ 1
2 "id": "2e41914a-024b-4e7e-a8cc-0c3c9f38a563",
3 "name": "SD Seminar",
4 "publisher": "D. E. Veloper",
5 - "brief": "",
6 + "brief": "A New Seminar module for managing sem
7 "description": "",
8 "version": "1.0.0.0",
9 "compatibilityid": "1.0.0.0",
10 "privacystatement": "",
11 "EULA": "",
12 "help": "",
13 "url": "",
14 "logo": "",
15 "capabilities": [],
16 "dependencies": [],
17 "screenshots": [],
18 "platform": "11.0.0.0",
19 "application": "11.0.0.0",
20 "idRange": {
21   "from": 50100,
22   "to": 50149
23 }
  
```

To add that to the repository, click the Source Control Checkmark and add a short description and it has been added to the local repository.

In order to upload the change to the Github repository, click the three dots and select the **Push** menu item.

## Solution Development in Visual Studio Code



On Pushing the changes to the repository, a message appears VS Code should automatically fetch changes. VS Code is able to periodically fetch changes from your remotes. This enables VS Code to show how many changes your local repository is ahead or behind the remote. Starting with VS Code 1.19, this feature is disabled by default and you use the `git.autofetch` setting to enable it.

Now the file has been added to the Github repository and the overview look like this:

A screenshot of a GitHub repository page. At the top, it shows '2 commits', '1 branch', '0 releases', and '1 contributor'. Below that, there are buttons for 'Branch: master', 'New pull request', 'Create new file', 'Upload files', 'Find file', and 'Clone or download'. The main area shows a list of commits: 'peikba Added Description' (latest commit, 13 minutes ago), '.alpackages Initialize' (44 minutes ago), 'vscode Initialize' (44 minutes ago), and 'app.json Added Description' (13 minutes ago).

Clicking the app.json file shows the latest version:

A screenshot of a GitHub file history page for 'app.json'. At the top, it shows 'Branch: master' and the file path 'SD Seminar / app.json'. There are buttons for 'Find file' and 'Copy path'. The history shows a single commit: 'peikba Added Description' (0817573 14 minutes ago). Below the commit, it says '1 contributor'. At the bottom, it shows '23 lines (23 sloc) | 496 Bytes' and buttons for 'Raw', 'Blame', 'History' (which is highlighted with a red box), and file download icons.

And clicking the History button, shows all the different versions:

History for SD\_Seminar / app.json

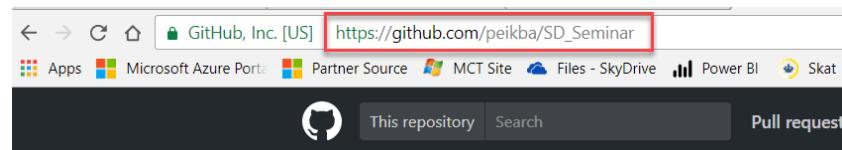
Commits on Feb 2, 2018	
Added Description	 0817573 
Initialize	 51354cb 

## Cloning an existing repository

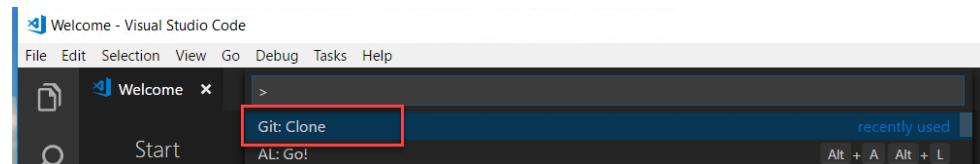
If the repository already exists like in this case:

Branch: master	New pull request	Create new file	Upload files	Find file	Clone or download
 peikba Next Field					Latest commit 89ef306 2 days ago
 .alpackages	Initialize				2 days ago
 .vscode	Initialize				2 days ago
 Tables	Next Field				2 days ago
 app.json	Added Description				2 days ago

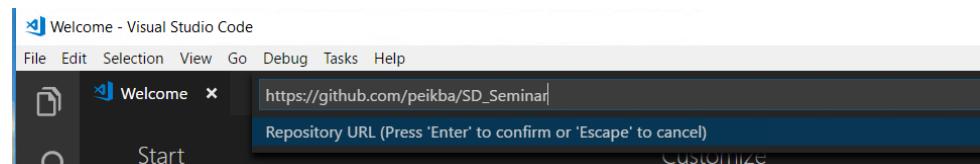
Simply copy the url:



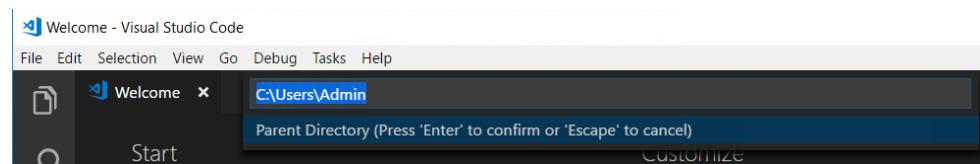
Now open VS Code, close all workspaces, press **Ctrl+Shift+P** and type or select **Git: Clone**



Enter the repository url:



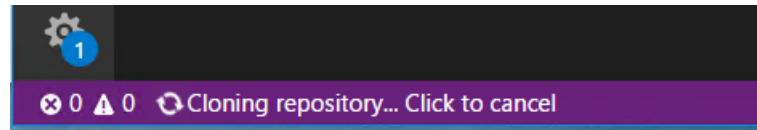
Enter the Parent Directory:



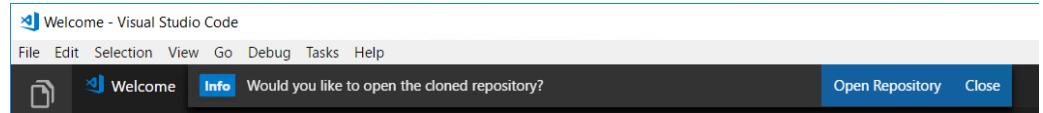
## Solution Development in Visual Studio Code

---

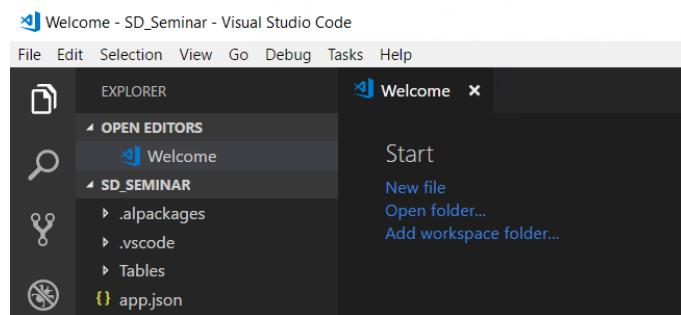
Now the repository will be cloned



On completion, the following message appear:



Now the Repository is cloned to the machine:

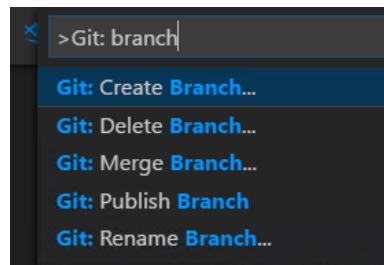


## Other Git functionalities worth mentioning

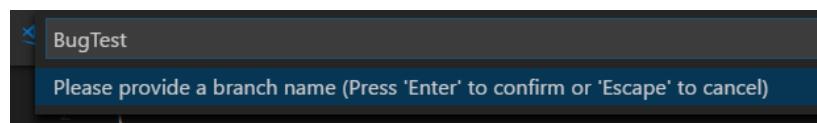
### Create a branch

You can create and checkout branches directly within VS code through **the Git: Create Branch...** and **Git: Checkout to...** commands in the Command Palette (Ctrl+Shift+P).

If you run **Git: Checkout to...**, you will see a dropdown containing all of the branches or tags in the current repository.



The **Git: Create Branch...** command lets you quickly create a new branch. Just provide the name of your new branch and VS Code will create the branch and switch to it.

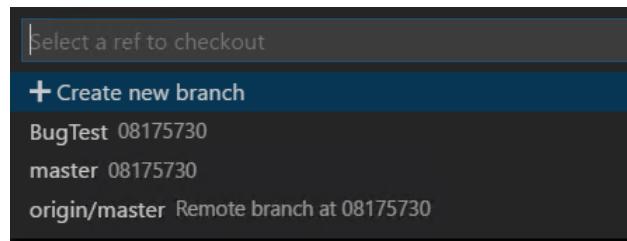


### Change branches

In the status bar the active Branch is shown:

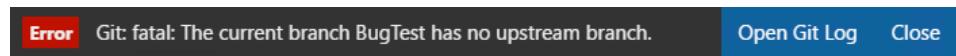


It possible the switch between the branches by clicking the branch name



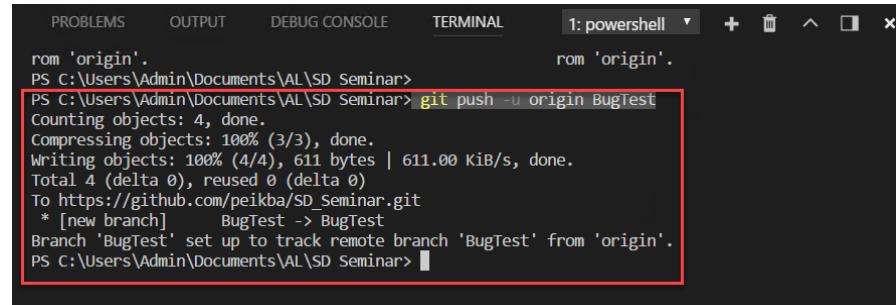
That will show a list of all the installed branches.

Pushing the commit to Github, will give you this error:



So, running the command: **git push -u origin BugTest** in the Terminal window gives this result:

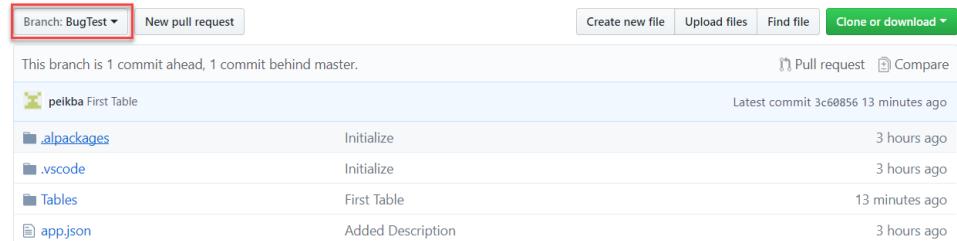
## Solution Development in Visual Studio Code



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL 1: powershell
rom 'origin'.
PS C:\Users\Admin\Documents\AL\SD Seminar> git push -u origin BugTest
PS C:\Users\Admin\Documents\AL\SD Seminar> git push -u origin BugTest
Counting objects: 4, done.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (4/4), 611 bytes | 611.00 KiB/s, done.
Total 4 (delta 0), reused 0 (delta 0)
To https://github.com/peikba/SD_Seminar.git
 * [new branch]      BugTest -> BugTest
Branch 'BugTest' set up to track remote branch 'BugTest' from 'origin'.
PS C:\Users\Admin\Documents\AL\SD Seminar>
```

And now it is possible to use the Push command on both the **BugTest** and the **master** branches.

In Github the branch looks like this:



Branch: BugTest ▾ New pull request Create new file Upload files Find file Clone or download

This branch is 1 commit ahead, 1 commit behind master.

peikba First Table

.alpackages Initialize 3 hours ago

.vscode Initialize 3 hours ago

Tables First Table 13 minutes ago

app.json Added Description 3 hours ago

Pull request Compare Latest commit 3c60856 13 minutes ago

Adding another field to the Seminar table and pushing that to the master repository, it looks like this:



Branch: master ▾ New pull request Create new file Upload files Find file Clone or download

peikba Next Field

.alpackages Initialize 3 hours ago

.vscode Initialize 3 hours ago

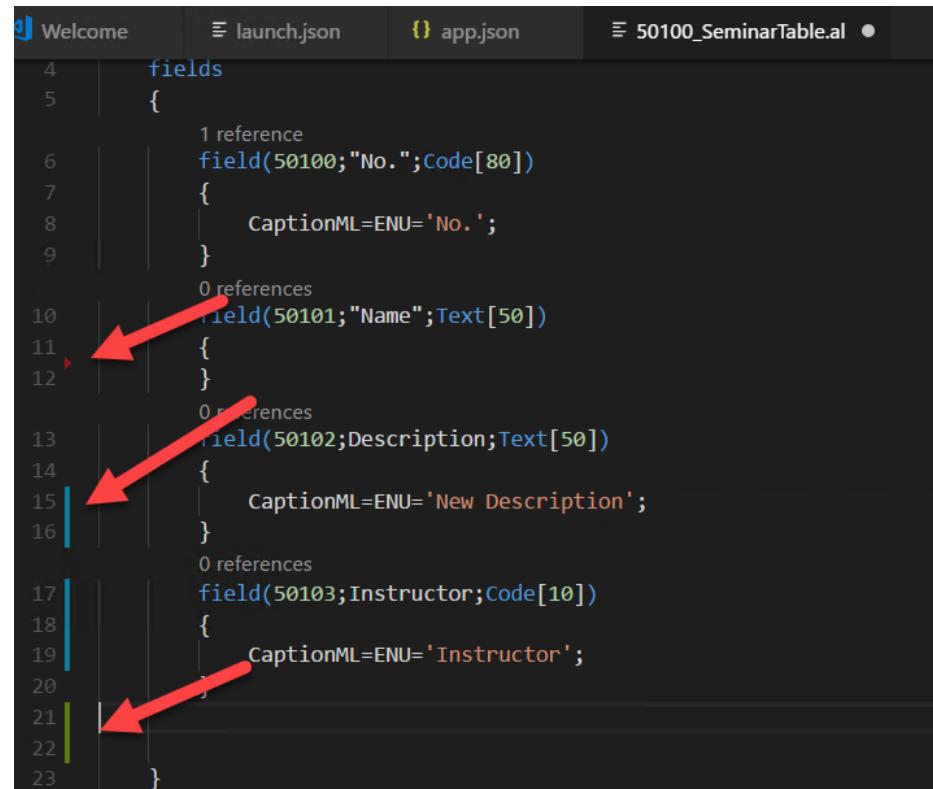
Tables Next Field 5 minutes ago

app.json Added Description 3 hours ago

Latest commit 89ef306 5 minutes ago

## Stage and clean changes

Making some changes to the code, will show a number of gutter indicators:



The screenshot shows a code editor with a dark theme. The gutter on the left side of the code area displays color-coded indicators for each line: a red arrow points to line 11, a blue arrow points to line 15, and a green arrow points to line 21. The code itself is a snippet of AL (Action Language) code defining fields for a table. Lines 11, 15, and 21 are marked as deleted, changed, and inserted respectively.

```
4     fields
5     {
6         1 reference
7         field(50100;"No.");Code[80]
8         {
9             CaptionML=ENU='No.';
10        }
11        0 references
12        field(50101;"Name");Text[50]
13        {
14        }
15        0 references
16        field(50102;Description;Text[50])
17        {
18            CaptionML=ENU='New Description';
19        }
20        0 references
21        field(50103;Instructor;Code[10])
22        {
23            CaptionML=ENU='Instructor';
24        }
25    }
```

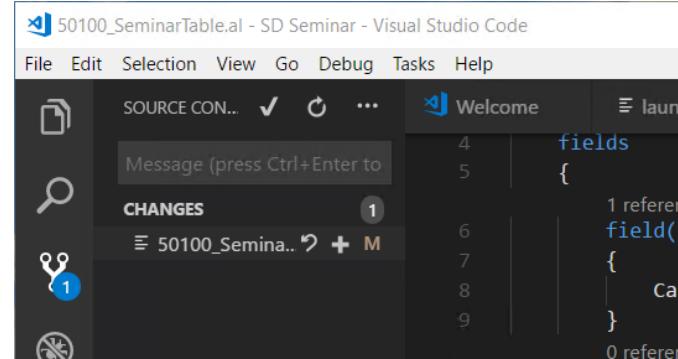
The red indicator shows that a line has been deleted

The blue indicator shows that a line has been changed

The green indicator shows that a line has been inserted

## Solution Development in Visual Studio Code

After saving the file the Source Control window look like this:

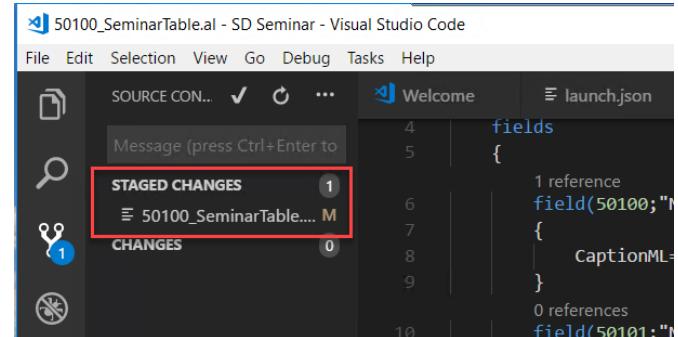


The screenshot shows the Visual Studio Code interface with the Source Control tab selected. The left sidebar shows a file named '50100\_SeminarTable.al'. The 'CHANGES' section shows 1 staged file. The 'STAGED CHANGES' section shows 1 staged file. The main editor area shows a JSON-like code block with the following content:

```
4   fields
5   {
6   1 referer
7   field('
8   {
9   }')
0 references
```

Now it is possible to stage the file by clicking the **+** sign or to revert the changes by clicking the undo **undo** sign.

Staging the file moves it to the staged files:

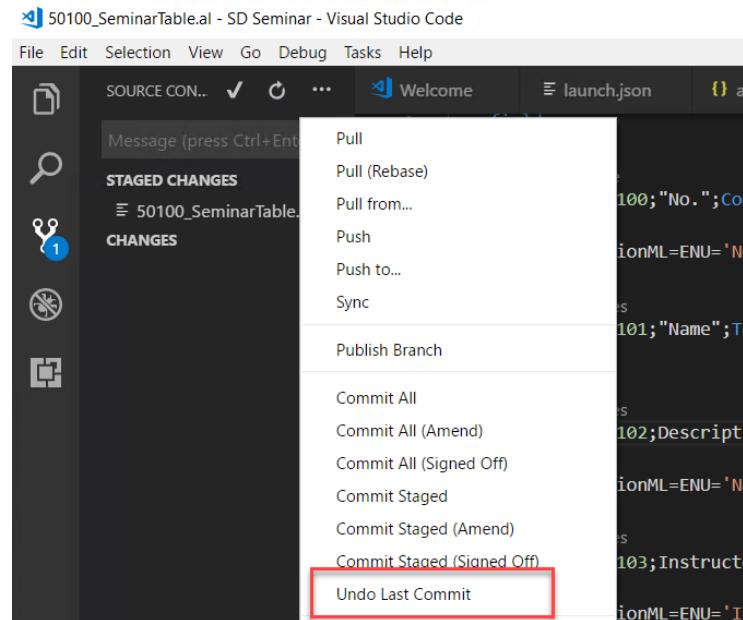


The screenshot shows the Visual Studio Code interface with the Source Control tab selected. The left sidebar shows a file named '50100\_SeminarTable.al'. The 'STAGED CHANGES' section is highlighted with a red box and shows 1 staged file. The 'CHANGES' section shows 0 changes. The main editor area shows the same JSON-like code block as the previous screenshot.

And now it is possible to commit the change.

### Undo a commit

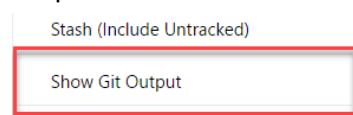
If the file was committed by mistake, it is possible to undo the commit by clicking the **Undo Commit** menu item:



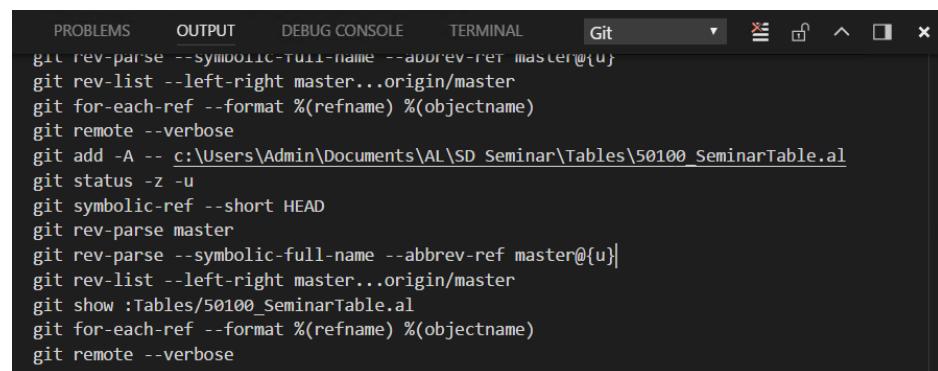
This will roll-back the changes.

### View git commands in Git output

To get an overview of the Git commands that has been run click the Show Git Output menu item.



Now all Git commands and their output is shown in the Output window.

A screenshot of the Visual Studio Code interface showing the Output window. The window title is 'Git'. It displays a list of git commands and their output. The commands include: git rev-parse --symbolic-full-name --abbrev-ref master@{u}, git rev-list --left-right master...origin/master, git for-each-ref --format %(refname) %(objectname), git remote --verbose, git add -A -- c:\Users\Admin\Documents\AL\SD Seminar\Tables\50100\_SeminarTable.al, git status -z -u, git symbolic-ref --short HEAD, git rev-parse master, git rev-parse --symbolic-full-name --abbrev-ref master@{u}, git rev-list --left-right master...origin/master, git show :Tables/50100\_SeminarTable.al, git for-each-ref --format %(refname) %(objectname), and git remote --verbose.

### Lab 3.1 Set up a Git Repository

#### Prerequisite to this lab:

In order to complete this lab, there are a number of prerequisites:

- The project "SD Seminar" must be created
- The Git program must be installed on the computer
- A free account on Github must be available

#### High-Level steps:

- Initialize the repository
- Make the initial commit
- Connect to Git with the email and user name from Github
- Create a new repository in Github
- Push the existing repository to Github
- Test that the source control works by changing a file

#### Detailed steps:

- 1) Open VS Code
- 2) Click the Source Control Icon 
- 3) Click the Initialize Repository icon 
- 4) Press Enter to select the SD Seminar folder
- 5) Enter the text "Initial" in the message field
- 6) Click the Source Control checkmark 
- 7) Click Always to the message box
- 8) Go to the Terminal window in the bottom of the page
- 9) Enter the text `git config --global user.email "Your Github email"`
- 10) Enter the text `git config --global user.name "Your Github user name"`
- 11) Go to the Source Control window again 
- 12) Click the Source Control checkmark 
- 13) Verify that no error message appears
- 14) Login to <http://github.com>
- 15) Create a new repository by clicking **New Repository**
- 16) Enter the name **SD\_Seminar**
- 17) Enter a description if desired
- 18) Click **Create Repository**
- 19) Copy the command **git remote add origin https://....**
- 20) Switch to VS Code
- 21) Paste the command into the Terminal window
- 22) Switch back to the Browser window
- 23) Copy the command **git push -u origin master**

- 24) Switch to VS Code
- 25) Paste the command into the Terminal window
- 26) Login to Github
- 27) Switch back to the Browser window
- 28) Verify that the repository has been synchronized
- 29) Switch to VS Code
- 30) Click the Explorer icon 
- 31) Click the **app.json** file
- 32) Enter a text in the “**brief**”: setting in the **app.json** file
- 33) Save the file 
- 34) Click the Source Control icon 
- 35) Click the **app.json** file
- 36) Verify that two windows open showing the changes before and after.

# Module 4: Data and Process Model

## Module Overview

Companies use Dynamics 365 BC to manage their business operations and processes, such as creating and posting sales invoices, controlling the inventory, handling purchases, and so on. Dynamics 365 BC 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 Dynamics 365 BC standard application. The gaps between the functionality that Dynamics 365 BC 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 Dynamics 365 BC can be developed in Dynamics NAV Development Environment and in C/AL programming language, or it can be developed in VS Code in the AL language in order to provide an add-on solution to avoid interfering with the standard solution. 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 in the Dynamics NAV Development Environment. 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. Changing the standard objects will also complicate the process of upgrading the standard functionality to a newer version.

As opposed to this, the modern development environment provides the opportunity to create solutions that are added to the database as extensions.

## What are extensions?

You can extend and customize a Dynamics 365 BC deployment without modifying the original application objects. With extension packages, you install, upgrade, and uninstall functionality in on-premises deployments or for selected tenants in a multitenant deployment. Customers can easily add or remove horizontal or

customized functionality to their solution. This makes upgrade much easier than past solutions.

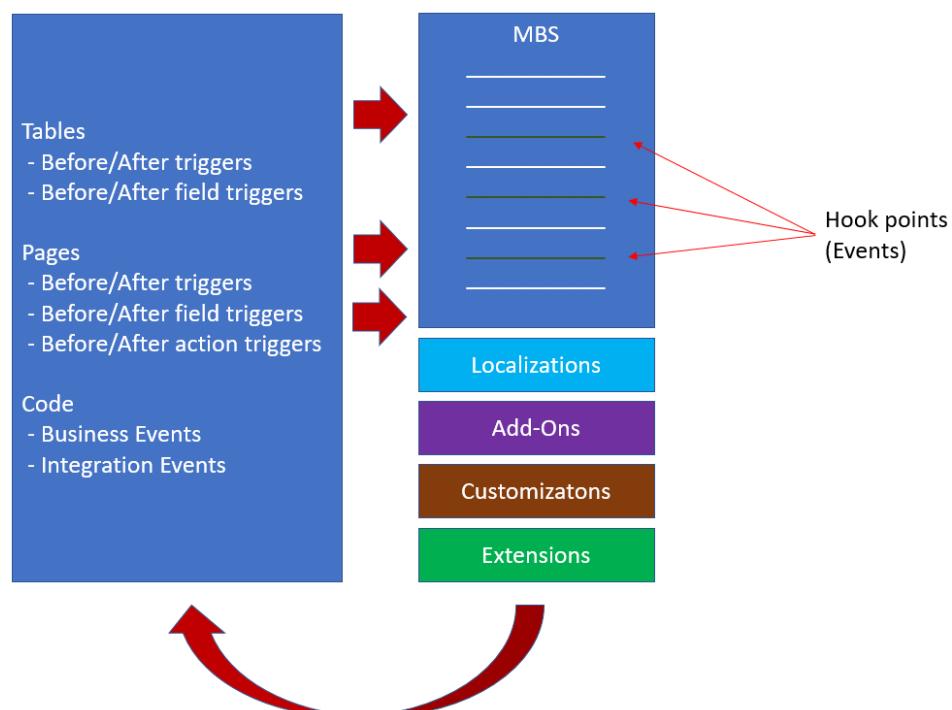
From Dynamics NAV 2016, it is possible to create extensions to customize pages and tables. The main difference from classical development is that source code modifications are not allowed. Instead, you use C/AL events to extend and customize objects. From Dynamics NAV 2017 it is possible to create extensions with these object types:

- Tables
- Pages
- Codeunits
- Reports
- Queries
- XMLports

As well as this, it is possible to extend some existing objects:

- Table extensions
- Page extensions

From the codeunits, it is possible to "hook in" to the existing solution using the so-called events and thereby extend the existing functionality without changing the standard application.



Creating solutions based on extensions demands a different way of thinking. Instead of changing the behavior of the standard functionality, the solution must

be based on add-on functionality.

An example could be:

A solution is needed to send delivery notes to the customers when the warehouse posts a shipment.

The event that contains all the needed information is **OnAfterPostSalesDoc** in codeunit 80 (Sales-Post). Because that event is executed in a place in the posting routine where:

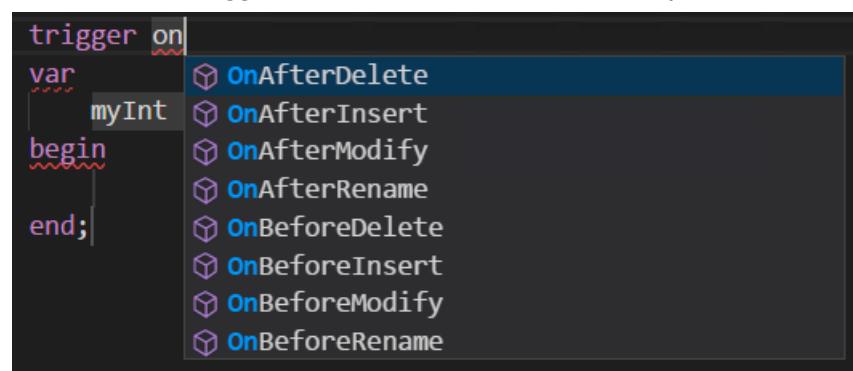
- It is after the **Sales Header** record is deleted
- and the posting has been committed

Therefore, there are no options to make any error and roll-back if the mail cannot be sent. It should not be the warehouse employee that must handle the deviation.

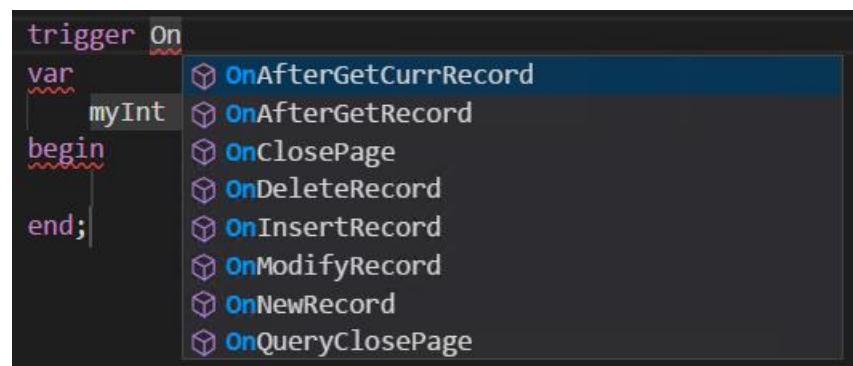
Therefore, it is necessary to make a log or send a mail to the sales person that the information on the customer is incomplete.

### Changing existing code and properties

In table extensions and page extensions it is possible to add fields and code to the fields. However, it is also possible to append code to existing objects triggers. In table extensions it is not possible to change the existing **OnInsert** or **OnModify** triggers, but it is possible to "Hook" in, either before or after the trigger. In table extensions, the triggers are named a little different; they are called:



In the page extensions, the triggers are names like the "original" triggers, but they will not replace the original triggers. They will be executed *after* the original page triggers.



When it comes to properties, a few properties can be altered on all extension objects:

- **Description**
- **Caption/CaptionML**

For tables the following property can be changed from a table extension:

- **DataCaptionFields**

For field properties in tables and pages, the following properties can be changed from extensions:

- **CaptionClass**
- **OptionCaption/OptionCaptionML**
- **ClosingDates**
- **Width**

For page properties, the following properties can be changed from extensions:

- **InstructionalText/InstructionalTextML**
- **DataCaptionExpression**
- **PromotedActionCategories/PromotedActionCategoriesML**

For page fields specifically, the following properties can be changed from extensions:

- **Style**
- **ApplicationArea**
- **Visible**
- **StyleExpr**
- **ToolTip/ToolTipML**
- **Enabled**
- **HideValue**
- **ShowCaption**
- **Importance**
- **QuickEntry**
- **ODataEDMType**

For page groups, the following properties can be changed from extensions:

- **Visible**
- **Enabled**
- **InstructionalText/InstructionalTextML**
- **FreezeColumn**

For page parts, the following properties can be changed from extensions:

- **Visible**

- **Enabled**
- **ToolTip/ToolTipML**

For page action groups, the following properties can be changed from extensions:

- **ToolTip/ToolTipML**
- **Visible**
- **Enabled**

And for page actions, the following properties can be changed from extensions:

- **ToolTip/ToolTipML**
- **Visible**
- **Enabled**
- **ApplicationArea**
- **Promoted**
- **PromotedIsBig**
- **PromotedOnly**
- **PromotedCategory**
- **ShortcutKey**
- **InFooterBar**

In the following we will create a solution as an extension, only utilizing functionality in form of code or properties, that can be created or invoked from VS Code.

## Developing solutions for Dynamics 365 BC

To develop solutions for Dynamics 365 BC, you must understand how the standard application works. You must become familiar with the basic principles that are consistently applied throughout Dynamics 365 BC and across all of its application areas.

Dynamics 365 BC is consistent and intuitive in how it presents data and manages processes. When you customize Dynamics 365 BC, 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 Dynamics 365 BC.

-

## Table Types and Characteristics

As a business management application, Dynamics NAV 365 BC 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 Dynamics NAV 365 BC 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 an 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	Item Vendor, FA Depreciation Book

Type	Remarks	Examples
	table.	
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 an application area.	Purchase Journal, Item Journal

## 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.



The G/L Account table is one important exception to this principle. It is the only master table in Dynamics NAV 365 BC 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 FastTab** 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 default Attribute:

- **Attribute**
- **Value**

Values from these fields are copied to the **Item Attributes** table for that Item, 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** and **DrillDownPageID** 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. It is however displayed as the data caption in the page.

## 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. The SumIndexField can be omitted and in that case the SQL server will still be able to calculate the field, only a little slower.

### 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 an 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.

Most often another property **DelayedInsert** is set to **true** to avoid generating a primary key before the line is ready to be inserted. If **DelayedInsert** is set to **true**, it will hold inserting the line, and thereby generating the next line no., until the user leaves the record. If it is set to **false**, the primary key will be generated on leaving the last of the primary key fields.

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 the 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.



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 Dynamics NAV 365 BC 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 design 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.

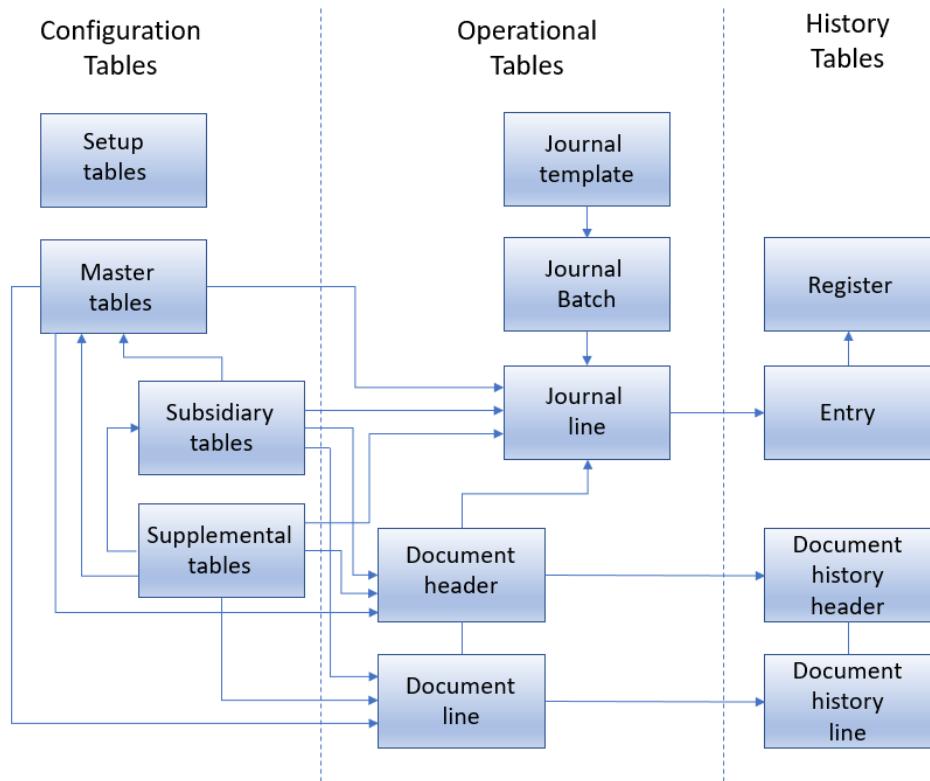
The consistency of data model and data flow patterns is important for both users and developers. When users master one application area and understand data model principles, they can also quickly understand other application areas. As a developer, when you understand the principles of data models and 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
<b>Configuration tables</b>	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.
<b>Operational tables</b>	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.
<b>History tables</b>	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.

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



## Typical Data Triggers

In addition to maintaining data validity through data types and table relationships, Dynamics NAV 365 BC 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 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
<b>Master</b>	<p>Assigns the <b>No.</b> field from the appropriate number series, if the user has not provided the value manually.</p> <p>Assigns the default dimensions for the account type, based on the configuration in the <b>Default Dimension</b> table.</p>
<b>Subsidiary</b>	<p>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.</p>
<b>Journal</b>	<p>Validates the values in Shortcut Dimension 1 and Shortcut Dimension 2 fields.</p>
<b>Document (header)</b>	<p>Assigns the <b>No.</b> field from the appropriate number series if the user has not provided the value manually.</p> <p>Applies certain defaults, such as dates.</p> <p>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.</p>
<b>Document (line)</b>	<p>Makes sure that the status can accept new lines if the document supports different statuses.</p>

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
<b>Master</b>	Sets the <b>Last Date Modified</b> field to the current system date.  <b>Note:</b> For master tables, the <i>OnRename</i> trigger must do the same action
<b>Journal</b>	Makes sure that a modification does not violate business rules for the specific journal type.
<b>Document (header)</b>	Assigns the <b>No.</b> 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.
<b>Document (line)</b>	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
<b>Master</b>	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.
<b>Supplemental</b>	Deletes all subsidiary information for the supplemental record.
<b>Journal</b>	Makes sure that the deletion does not violate business rules for the specific type of journal.
<b>Document (header)</b>	Makes sure that the deletion does not violate any business rules for the specific document, and deletes all document lines and subsidiary document information.
<b>Document (line)</b>	Makes sure that the deletion does not violate any business rules for the specific document, and deletes all subsidiary document line information.
<b>Document History (header)</b>	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.
<b>Document History (line)</b>	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.

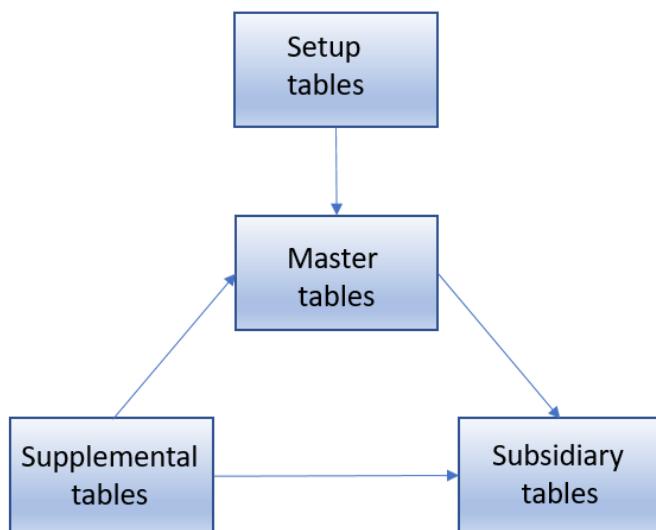
## Standard Data Flow

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.

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



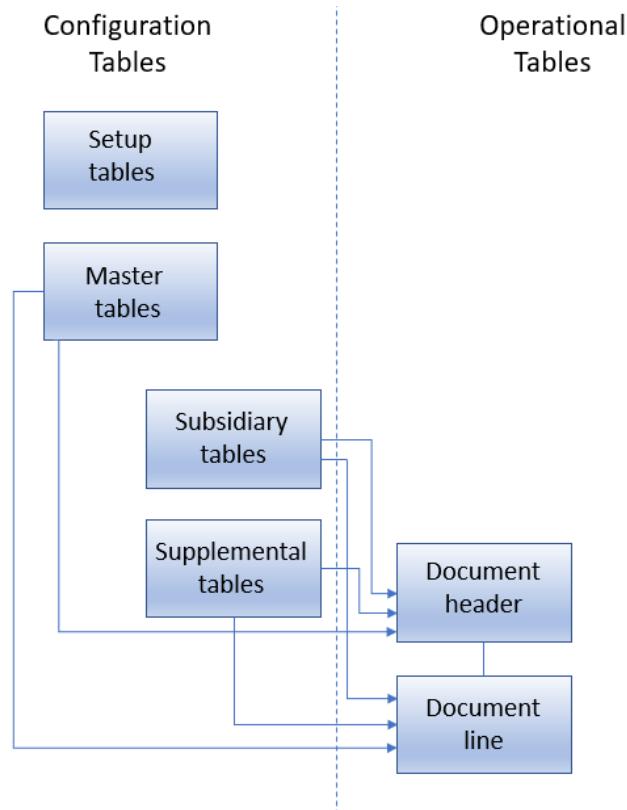
Typically, master tables contain many fields that have relationships to other tables, such as supplemental and other master record tables. When you enter values in these fields, the application may take default values from related tables and assign them to the master record. In addition, when you enter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

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

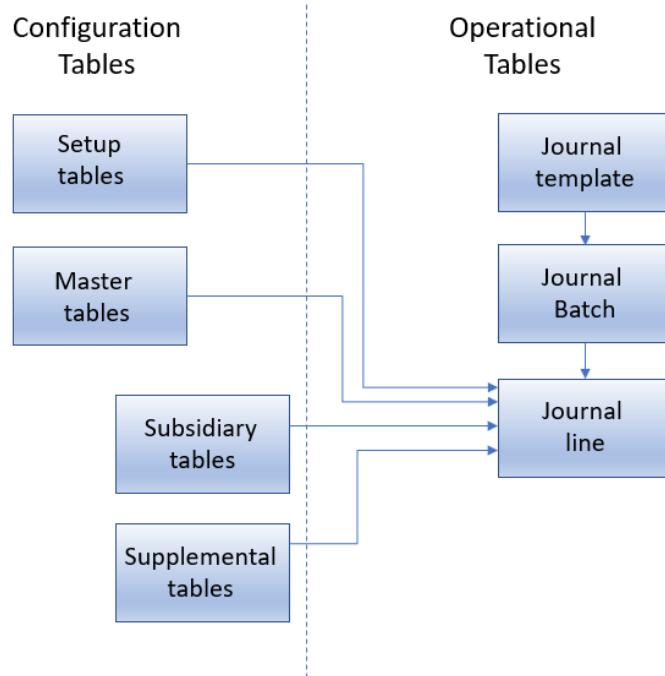


### Journal Creation Data Flow

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.



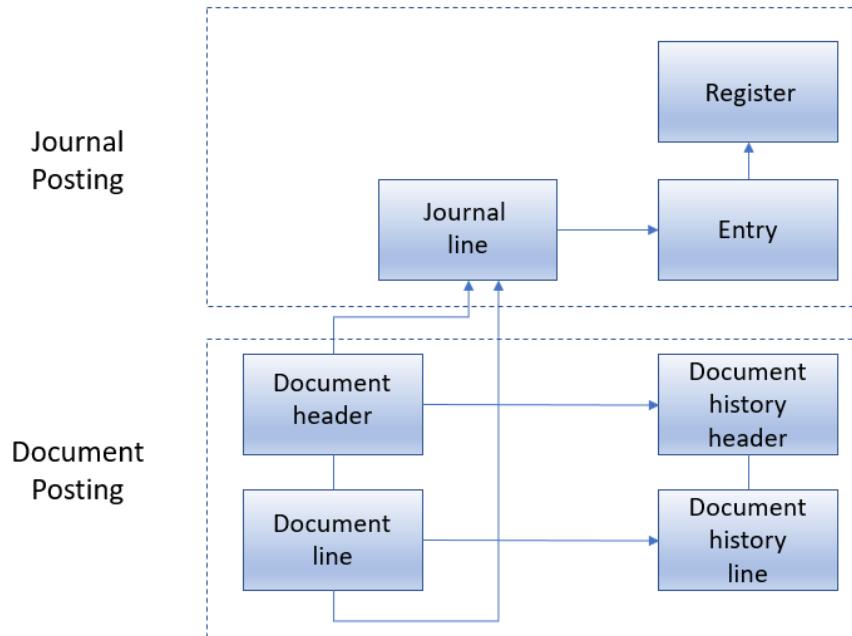
### Posting Data Flow

Posting is one of the most important processes in Dynamics NAV 365 BC. 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.

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



## 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 Dynamics NAV 365 BC. 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
<b>Journal – Post Line</b>	Reads information from a single journal line, and then writes corresponding ledger entry or entries.
<b>Journal – Check Line</b>	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.
<b>Journal – Post Batch</b>	Reads information from all lines in a batch, and then calls the check line and post line for each line in the batch.

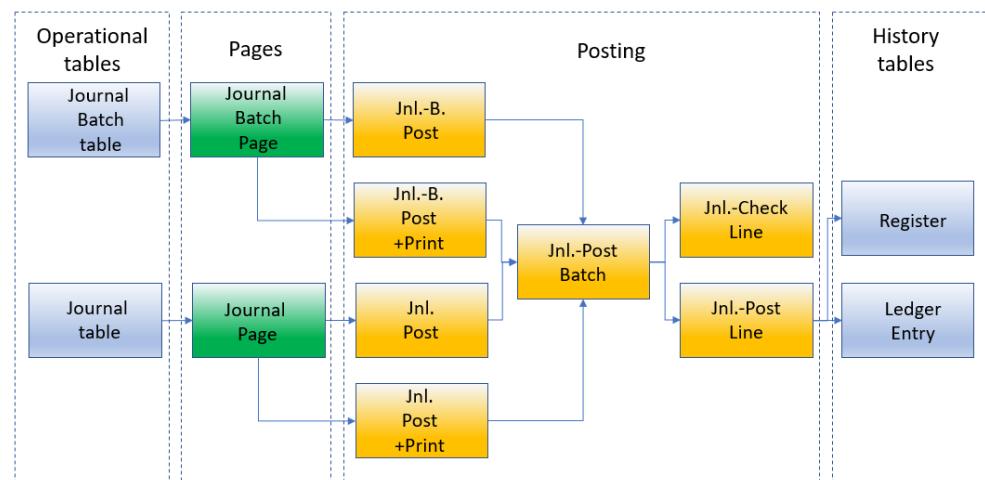
A user can never call any of these codeunits directly. They are always called by another journal or document posting codeunits.

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.
------------------------------	-----------------	---

The following shows the process and data flow of journal posting codeunits.



## 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.

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:

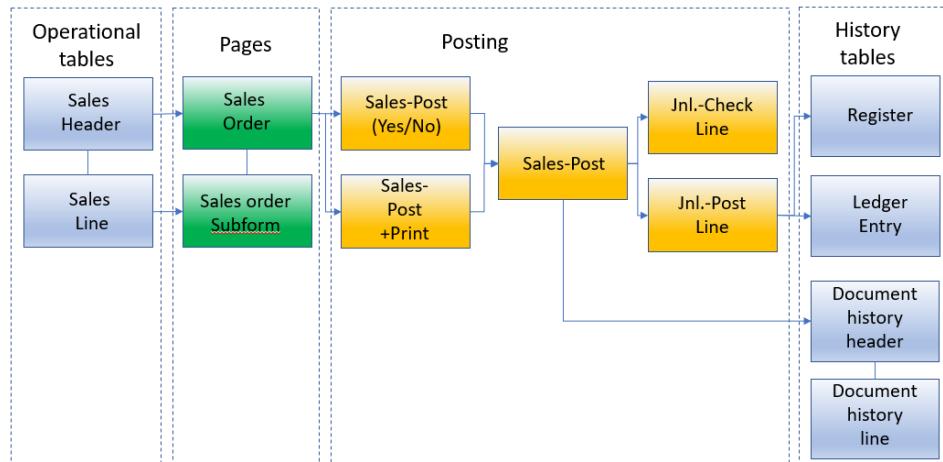
- G/L Entry
- Cust. Ledger Entry
- Item Ledger Entry
- Res. Ledger Entry
- FA. Ledger Entry

## Solution Development in Visual Studio Code

A document posting routine consists of several codeunits.

Codeunit	Remarks
<b>Post (Yes/No)</b>	Asks for a confirmation, and then calls the Post codeunit. Users can call this codeunit from document and document list pages.
<b>Post + Print</b>	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.
<b>Post</b>	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.

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



## Lab 4.1 Prepare the workspace for the Seminar solution

### Prerequisite to this lab:

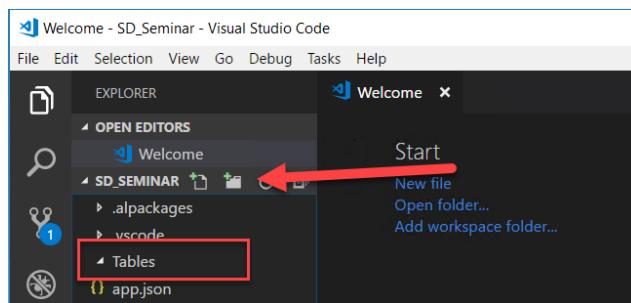
This lab cannot be solved without having completed the previous labs

### High-Level steps:

- Create new folders in the workspace to hold the different object types: **Tables**, **Pages**, **Codeunits**, **Reports**, **Queries** and **XMLports** as well as folders for **Table Extensions** and **Page Extensions**.

### Detailed steps:

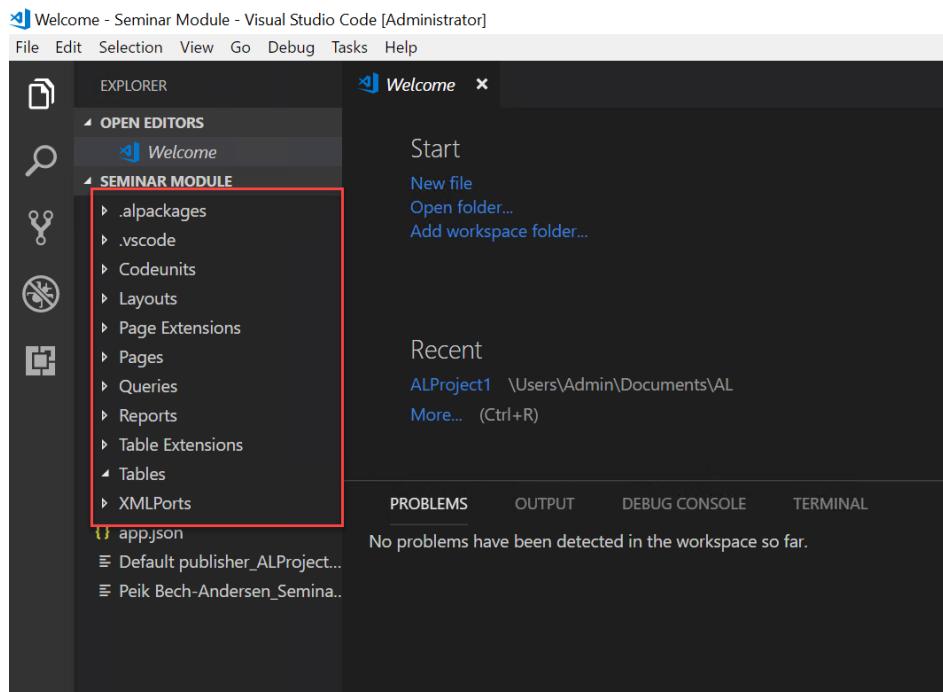
- 1) Open VS Code
- 2) Click the "Create Folder" button:



Make sure that there are no files or folders selected on creating the folders, otherwise the folder will not be created in the root of the workspace.

- 3) Repeat the above action for the following folders:
  - a. **Pages**
  - b. **Codeunits**
  - c. **Reports**
  - d. **Queries**
  - e. **XMLports**
  - f. **Table Extensions**
  - g. **Page Extensions**
  - h. **Layouts**

- 4) Verify that the workspace looks like this:



- 5) Save the workspace using **Ctrl+K S** or using the menu item **File/Save All**

## Module Review

### Module Review and Takeaways

Dynamics NAV 365 BC 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, Dynamics NAV 365 BC 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 Dynamics 365 BC 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 Dynamics NAV 365 BC 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 Dynamics 365 BC, 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.

# Module 5: 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 an 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 an application area in Dynamics 365 BC 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 Dynamics 365 BC, the *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 business logic that executes when certain system events occur. Some of this logic is mandated by Dynamics 365 BC 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 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 Dynamics 365 BC standard functionality. In each module, this “Prerequisite Knowledge” lesson presents Dynamics 365 BC standards and principles that you can apply while customizing the solution for CRONUS International Ltd.

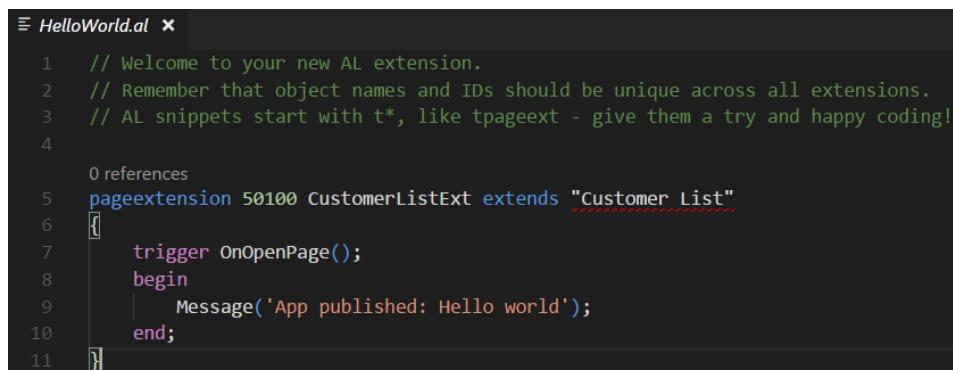
## Triggers

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 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 AL code. It typically contains free-form documentation. This trigger lets you document the changes that you have made to objects.

Working in VS Code the documentation trigger is not available. However, it is possible to insert documentation lines everywhere and in order to keep track of all changes. It is therefore recommended to add a line of documentation to the top of the .al file, describing the change made to the code like demonstrated in the below code.



```
 1 // Welcome to your new AL extension.
 2 // Remember that object names and IDs should be unique across all extensions.
 3 // AL snippets start with t*, like tpageext - give them a try and happy coding!
 4
 5 pageextension 50100 CustomerListExt extends "Customer List"
 6 [
 7     trigger OnOpenPage();
 8     begin
 9         | Message('App published: Hello world');
10     end;
11 ]
```

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

## Multi-Language Support

Dynamics 365 BC is a multilanguage application. This means that a localized version of Dynamics 365 BC 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.

Since VS Code and Dynamics 365 BC is “Work-in-progress”, it has previously been necessary to apply all captions in every language via the **CaptionML** property. However, The support for using the ML properties, such as **CaptionML** and

**TooltipML**, is being deprecated, so it is recommended to refactor your extension to use the corresponding property, such as **Caption** or **Tooltip**, which is being picked up in the .xlf file.

## FilterGroup

Filtergroup can contain a filter for a Record that has been set earlier with the **SetFilter** Function (Record) or the **SetRange** Function (Record). The total filter applied is the combination of all the filters set in all the filtergroups.

When you select a filter group, subsequent filter settings by the **SetFilter** Function (Record) or the **SetRange** Function (Record) apply to that group.

All groups are active at all times. The only way to disable a group is to remove the filters set in that group.

Filters in different groups are all effective simultaneously. For example, if in one group, a filter is set on customer numbers 1000 to 2000, while in another group, a filter is set on customer numbers 1800 to 3000, then only numbers in the range 1800 to 2000 are visible.

Dynamics 365 BC uses the following filter groups internally.

Number	Name	Description
-1	Cross-column	Used to support the cross-column search.
0	Std	The default group where filters are placed when no other group has been selected explicitly. This group is used for filters that can be set from the filter dialogs by the end user.
1	Global	Used for filters that apply globally to the entire application.
2	Form	Used for the filtering actions that result from the following: <ul style="list-style-type: none"> <li>• SETTABLEVIEW Function (Page, Report, XMLport)</li> <li>• SourceTableView Property</li> <li>• DataItemTableView Property.</li> </ul>
3	Exec	Used for the filtering actions that result from the following: <ul style="list-style-type: none"> <li>• SubPageView Property</li> </ul>

		<ul style="list-style-type: none"> <li>• RunPageView Property</li> </ul>
4	Link	<p>Used for the filtering actions that result from the following:</p> <ul style="list-style-type: none"> <li>• DataItemLink Property (Reports)</li> <li>• SubPageLink Property</li> </ul>
5	Temp	Not currently used.
6	Security	Used for applying security filters for user permissions.
7	Factboxes	Used for clearing the state of factboxes.

A filter set in a group different from filter group 0 cannot be changed by a user that uses a filter dialog to set a filter. If, for example, a filter has been set on customer numbers 1000 to 2000 in group 4, then the user can set a filter that delimits this selection further but cannot widen it to include customer numbers outside the range 1000 to 2000.

It is possible to use one of the internally used groups from C/AL. If you do this, you replace the filter that Dynamics 365 BC assumes is in this group. If, for example, you use filter group 4 in a page, you will replace the filtering that is actually the result of applying the **SubPageLink** Property. This could seriously alter the way pages and subpages interact.

Using **FilterGroup 7** may cause factboxes to not work as intended.

```
trigger OnOpenPage();
begin
    FilterGroup(1);
    SetFilter(Type, '%1', Type::Person);
    FilterGroup(0);
end;
```

To reset the filters in filter group 1, you add an empty filter to the group. To add an empty filter, to filter group 1, you must first set the filter group.

```

trigger OnOpenPage();

begin
    FilterGroup(1);
    SetFilter(Type, '');
    FilterGroup(0);
end;
  
```

## Object numbering

Dynamics 365 BC operates with different object numbers and field numbers depending on the origin of the object or field. Creating new solutions for a Dynamics 365 BC, it is imperative that the object numbers used in the solution do not conflict with the existing objects or fields. Likewise, the solution can not conflict with any other installed extensions or even future extensions. Solutions that are approved with the CfMD approval (Certified for Dynamics) will be assigned a unique number series, which will never conflict with any other solution. The origin of different object numbers can be explained as shown below:

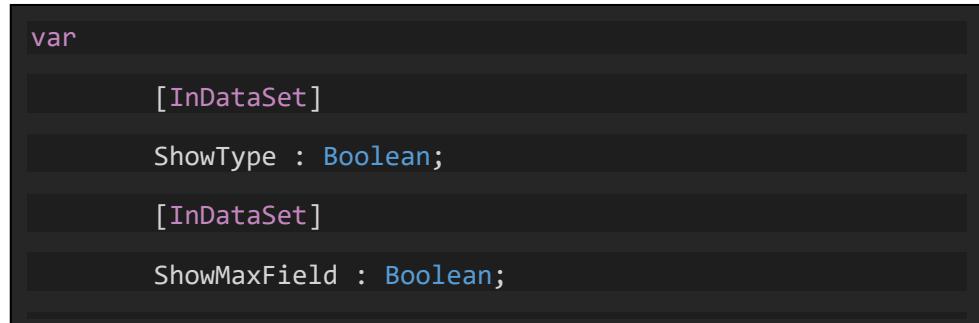
Object Number Series	Origin
0 - 49.999	The standard solution from Microsoft
10.000 - 19.999	Country specific localization from Microsoft
50.000 – 99.999	Object range to be used by all Microsoft partners for customizations. End-user customers has to acquire access to any range used by the Microsoft partner.
50.000 – 50.009	Included in the starter pack: Access to ten tables and codeunits.
50.010 – 50.019	Included in the extended pack: Access to ten tables and codeunits.
50.000 – 50.099	Also included in the starter pack: one hundred pages, reports, Queries and XMLports.
50.100 – 50.199	Also included in the extended pack: one hundred pages and XMLports.
100.000 – 999.999	Testing objects and other objects reserved by Microsoft
1.000.000 – 98.999.999	CfMD approved add-ons
99.000.000 – 99.999.999	Microsoft Standard functionality for the manufacturing module
1.000.000.000	Used by Dynamics NAV versions up till NAV 2009 for storing data on restoring a backup to the system. When the restore is completed all objects and data are moved from the 1.000.000.000 range to the actual ranges.
2.000.000.000	System and virtual objects
123.465.700 – 123.456.799	Objects used only for training purposes

In a normal development situation, the object numbers used will be depending on if the solution is going to be CfMD approved.

For this solution we will use object numbers starting at **123.456.700**. Field numbers in table extensions will also start at **123.456.700**, whereas field numbers in all new tables will start at 10 and increment 10 for each next field.

### The InDataSet property

Due to the Dynamics 365 BC architecture, all code is executed in the service tier. This also means that variables are not available in the page by default. In order to make the variable available on the page, it is necessary to include the variable to the dataset. This is done by setting a property just before assigning the variable:



```
var
  [ InDataSet ]
  ShowType : Boolean;
  [ InDataSet ]
  ShowMaxField : Boolean;
```

## Object Naming

You must use a prefix/suffix when naming all new objects. When you modify a core Dynamics 365 BC object using a Table Extension or Page Extension, the prefix must be defined at the control or field level. Examples could be:

- table 70000000 "CSD Salesperson"
- page 70000000 "CSD Salesperson Card"
- codeunit 70000000 "CSD Salesperson Management"

## 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 Dynamics 365 BC 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.

Therefore, you must make sure that you determine whether there is existing functionality in Dynamics 365 BC that CRONUS International Ltd. can use to represent participants. Following is the minimum information that you must track:

- **Name**
- **Address**
- **Phone Number**
- **Email Address**

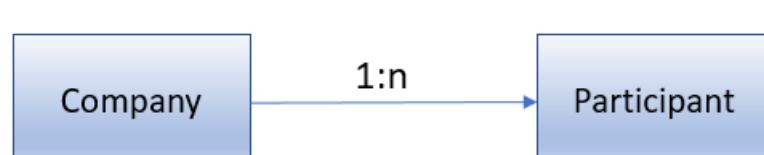
There are many tables in Dynamics 365 BC that include this kind of information about an entity. You must determine whether there is a table that you can use, or if you must develop one.

Following are two candidate tables in Dynamics 365 BC 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:



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 Dynamics 365 BC 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 Dynamics 365 BC standard functionality in the following way:

Required Functionality	Standard Functionality
Participant	Contact
Company	Customer

However, the requirements state that both participants and companies can receive invoices. In Dynamics 365 BC 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 Dynamics 365 BC. Unless there are other options, this approach should always be your last choice.

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 Dynamics 365 BC. It also does not conflict or contradict the requirements. On top of this, a contact of the type **Customer** is automatically created when creating the customer and a contact of the type **Person** is automatically created when entering the name of the contact person on the customer card. This supports the choice of the second solution.

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 Dynamics 365 BC is preferred.

Therefore, instead of customizing the sales area, you should opt for the second solution, and use the standard Dynamics 365 BC functionality.



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 Dynamics 365 BC 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 Dynamics 365 BC 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 Dynamics 365 BC 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 Dynamics 365 BC, 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.

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.



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"> <li>• <b>Resource Type</b></li> <li>• <b>Maximum Participants</b></li> </ul>
Page 77, Resource List	After the <b>Name</b> field, add the <b>Resource Type</b> and <b>Maximum Participant</b> fields. Make sure that you hide the <b>Type</b> field if the page is filtered by <b>Type</b> . Make sure that you hide the <b>Maximum Participant</b> field if the page is filtered to show only instructors.

<b>Page 76, Resource Card</b>	Add the <b>Resource Type</b> field to the <b>General</b> FastTab. After <b>Personal Data</b> , add the <b>Room</b> FastTab. In the <b>Room</b> FastTab, add the <b>Maximum Participants</b> field.
-------------------------------	--

The “Page 77, Resource List” and “Page 76, Resource Card” figure shows pages after development work is completed.

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

Resources					
No.	Name	Type	Resource Type	Quantity Per Day	Base Unit of Measure
LIFT	Lift for Furniture	Machine	Internal	0.00	Hour
LINDA	Linda Martin	Person	Internal	0.00	Hour
MARK	Mark Hanson	Person	Internal	0.00	Hour
MARY	Mary A. Dempsey	Person	Internal	0.00	Hour
TIMOTHY	Timothy Sneath	Person	Internal	0.00	Hour

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

ROOM 01 · Room 01

**General**

Name:	<input type="text" value="Room 01"/>	Last Date Modified:	<input type="text" value="23-03-2018"/>
Type:	<input type="text" value="Machine"/>	Use Time Sheet:	<input type="checkbox"/>
Base Unit of Measure:	<input type="text" value="Hour"/>	Time Sheet Owner User ID:	<input type="text"/>
Search Name:	<input type="text"/>	Time Sheet Approver User ID:	<input type="text"/>
Resource Group No.:	<input type="text"/>	Resource Type:	<input type="text" value="Internal"/>
Blocked:	<input type="checkbox"/>	Quantity Per Day:	<input type="text" value="8"/>

▼ Show more fields

**Invoicing**

**Personal Data**

**Room**

Maximum Participants:

### Lab 5.1: Customize Resource Tables and Pages

#### Scenario

As a part of the development team who works on the implementation project of Dynamics 365 BC at CRONUS International Ltd. You must customize the resource table to meet the requirements in the **Functional Requirements Document** for the project.

#### 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.



All extension names and field names in extensions must be prefixed with a code that is at least three characters long. This is to separate the name from similar extensions to the same table. In solutions published through App Source, a prefix will be assigned to the extension. Here the prefix **CSD** must be used. The prefix only applies to the name, the caption will be as normal to keep semblance to the users. E.g. name: **“CSD Resource Type”** and caption **“Resource type”**



A trick is to prefix all file names with the last two numbers from the object number. Thereby, it is easier to identify the next vacant object number to be used.

## Task 1: Add Fields to the Resource Table

### Prerequisite to this lab:

This lab cannot be solved without having completed the previous labs.

### High Level Steps:

Create a file **00\_ResourceExt.al** with table extension for the **Resource** table with the following fields:

Field No.	Field Name	Data Type	
123456701	CSD Resource Type	Option	Options: Internal, External
123456702	CSD Maximum Participants	Integer	
123456703	CSD Quantity Per Day	Decimal	

Add code to the existing **Profit %** field to test if the **Unit Cost** has been filled.

### Detailed Steps:

#### Create the Table Extension for the Resource Table

- 1) In the VS Code Explorer window, click the **Table Extension** folder, then right-click and select the **New File** menu item
- 2) Enter **00\_ResourceExt.al** as the file name and press enter
- 3) Enter **tt** in the first line and select the **ttableext** snippet
- 4) Enter **123456700** as the **ID** and **"CSD ResourceExt"** as the **name** to extend the **Resource** table and press enter



Note that the **ID** is automatically selected, so enter the number and press the **Tab** key. Now the **MyExtension** is selected and the name can be entered without deleting the text

- 5) Enter documentation right underneath the **tableextension** line  
All documentation lines must be prefixed with // like this:  
**// CSD1.00 - 2018-01-01 - D. E. Veloper**
- 6) In the **fields** section, delete the line  
**// Add changes to table fields here**
- 7) Add a blank line and type **modify("Profit %")** and press [Enter]
- 8) On the next line add a **{**, and VS Code will automatically add the matching **}**. After pressing Enter, the code will format automatically.
- 9) Create the **OnAfterValidate** trigger by typing **ttr** and selecting the **ttrigger** snippet
- 10) Replace the **OnWhat** with **OnAfterValidate**
- 11) Delete the **var** section
- 12) Add the following line between the begin and the end:  
**Rec.TestField("Unit Cost");**
- 13) After the **modify** section, add a blank line and use the **tfields** snippet to add a



It is possible to add validation code before and after the existing validation code of an existing field. The triggers to use are:  
**OnBeforeValidate** and **OnAfterValidate**

new field

- 14) Enter **123456701** as **ID** and press the **Tab** key
- 15) Enter “**CSD Resource Type**” instead of **MyField** and press the **Tab** key



Entering the “, the **MyField** is automatically converted to “**MyField**” and the selection still only includes the **MyField**. This means that it is possible to enter the new field name directly

- 16) Enter **Option** instead of the **Type** and press the **Tab** key
- 17) Replace the **FieldPropertyName** with **Caption** and press the **Tab** key
- 18) Replace the **FieldPropertyValue** with ‘**Resource Type**’ and press the **Enter** key
- 19) Add the following properties:  
**OptionMembers = "Internal","External";**  
**OptionCaption = 'Internal,External';**
- 20) Add field **123456702 “CSD Maximum Participants”** with the type **Integer** as described above
- 21) Add the Caption property to the field without the **CSD**
- 22) Add field **123456703 “CSD Quantity Per Day”** with the type **Decimal** as described above
- 23) Add the Caption property to the field
- 24) Delete the **var** section including the **myint : Integer;** line
- 25) Save the file by clicking **Ctrl+S**
- 26) Verify that the solution looks like this:

```
tableextension 123456700 "CSD ResourceExt" extends Resource
// CSD1.00 - 2018-01-01 - D. E. Veloper
// Chapter 5 - Lab 1-1
{
    fields
    {
        modify("Profit %")
        {
            trigger OnAfterValidate()
            begin
                Rec.TestField("Unit Cost");
            end;
        }
        field(123456701;"CSD Resource Type";Option)
        {
            Caption = 'Resource Type';
            OptionMembers = "Internal","External";
            OptionCaption = 'Internal,External';
        }
        field(123456702;"CSD Maximum Participants";Integer)
        {
            Caption = 'Maximum Participants';
        }
        field(123456703;"CSD Quantity Per Day";Integer)
        {
            Caption = 'Quantity Per Day';
        }
    }
}
```

### Task 2: Add Fields to the Resource Card Page

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.

#### High Level Steps:

Create a file **00\_ResourceCardExt.al** with a page extension for the **Resource Card** page with the following fields:

- Add the field **Resource Type** to the **General** FastTab
- Create a new FastTab **Room** underneath the **Personal Data** FastTab
- Add the field **Maximum Participants** to the **Room** FastTab
- Add code to the **OnOpenPage** trigger to hide the **Type** field if a filter has been set on the **Type** field.

#### Detailed Steps:

##### Create the ResourceCardExt Page Extension

- 1) In the VS Code Explorer window, click the **Page Extension** folder, then right-click and select the **New File** menu item
- 2) Enter **00\_ResourceExtCard.al** as the file name and press enter
- 3) Enter **tpa** in the first line and select the **tpageext** snippet
- 4) Enter **123456700** as the **ID** and **"CSD ResourceCardExt"** as the name to extend the **Resource Card** page and press enter
- 5) Enter documentation right underneath the **pageextension** line  
All documentation lines must be prefixed with **//** like this:  
**// CSD1.00 - 2018-02-01 - D. E. Veloper**
- 6) In the **layout** section, delete the line  
**// Add changes to page layout here**
- 7) Add a blank line and type **a**, select **addlast** and type **General** in parentheses.



This will add the fields after the last field in the **General** FastTab. If another location is desired, it is possible to use **addafter(FieldName)** instead.



To get a list of available fields and groups, press **Ctrl+Space** in the parentheses

- 8) On the next line add a **{**, and VS Code will automatically add the matching **}**. After pressing Enter, the code will format automatically.
- 9) Type **field** and end with **()**
- 10) Start typing **"CSD Resource Type"** and select the field when the Intellisense suggests it.



If Intellisense does not suggest anything, there is an error previously in the file. Check for red markings anywhere above and correct the error. The error is also described in the **Problems** section of the **Output** windows

- 11) Verify that the line looks like this:  
**field("CSD Resource Type"; "CSD Resource Type")**
- 12) On the next line add a **{**, and VS Code will automatically add the matching **}**. After pressing Enter, the code will format automatically.
- 13) Add a blank line after the **field("CSD Resource Type"; "CSD Resource Type")** section and type:

- field("CSD Quantity Per Day";"CSD Quantity Per Day")**
- 14) On the next line add a {, and VS Code will automatically add the matching }. After pressing Enter, the code will format automatically.
  - 15) Outside the addlast(general) section add a new line and type **addafter("Personal Data")**
  - 16) On the next line add a {, and VS Code will automatically add the matching }. After pressing Enter, the code will format automatically.
  - 17) Type **group** and type **"CSD Room"** in the parentheses
  - 18) On the next line add a {, and VS Code will automatically add the matching }. After pressing Enter, the code will format automatically.
  - 19) Add the caption for the **Room** group
  - 20) Type **field** and end with the parenthesis
  - 21) Start typing **"CSD Maximum Participants"** and select the field when the Intellisense suggests it.
  - 22) On the next line add a {, and VS Code will automatically add the matching }. After pressing Enter, the code will format automatically.
  - 23) Go to the **var** section in the bottom of the file to create a global variable.
  - 24) Delete the **myInt : Integer;** line and create a new line
  - 25) Type a [ and press **Ctrl+[Space]** to get a list of options
  - 26) Select the **inDataSet** option to make the variable available to the page as a part of the dataset.
  - 27) Add a variable like this:  
**ShowMaxField: Boolean;**
  - 28) Above the **var** section, add a blank line and type **ttrigger**
  - 29) In the **OnWhat** type **OnAfterGetRecord** and press **[Enter]**
  - 30) Delete the newly created **var** section, because there is no need for local variables
  - 31) In the begin end block, type the following code:

```
trigger OnAfterGetRecordPage();
begin
  ShowMaxField := (Type = Type::Machine);
  CurrPage.Update(false);
end;
```

- 32) Return to the "Room" Group and add the following property:  
**Visible = ShowMaxField;**
- 33) Save the file by clicking **Ctrl+S**
- 34) Verify that the file looks like this:

```
pageextension 123456700 "CSD ResourceCardExt" extends
"Resource Card"

// CSD1.00 - 2018-01-01 - D. E. Veloper

// Chapter 5 - Lab 1-2

// Added new fields:
// - Internal/External
// - Maximum Participants
// Added new FastTab
// Added code to OnOpenPage trigger

{
    layout
    {
        addlast(General)
        {
            field("CSD Resource Type"; "CSD Resource Type")
            {
            }
            field("CSD Quantity Per Day"; "CSD Quantity Per Day")
            {
            }
    }
}
```

```
addafter("Personal Data")
{
    group("Room")
    {
        Visible = ShowMaxField;
        field("CSD Maximum Participants"; "CSD Maximum
            Participants")
        {
        }
    }
}

trigger OnAfterGetRecord();
begin
    ShowMaxField := (Type = Type::Machine);
    CurrPage.Update(false);
end;

var
    [InDataSet]
    ShowMaxField: Boolean;
}
```

### Task 3: Add Fields to the Resource List Page

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.

#### High Level Steps:

Create a new file **01\_ResourceListExt.al** with a page extension of the **Resource List** page:

- Add the field "**CSD Resource Type**" after the **Type** field
- Add the field "**CSD Maximum Participants**" after the "**CSD Resource Type**" field
- Add code to the **OnOpenPage** trigger only to executed if the page is being called from a **SubPageView** or a **RunPageView** Property:
  - Hide the **Type** field if a filter has been set on the **Type** field.
  - Hide the Maximum Participants field if a filter has been set on the **Type** field to show only Machines.

#### Detailed Steps:

##### Create the ResourceListExt Page Extension

- 1) In the VS Code Explorer window, click the **Page Extension** folder, then right-click and select the **New File** menu item
- 2) Enter **01\_ResourceExtList.al** as the file name and press enter
- 3) Enter **tp** in the first line and select the **tpageext** snippet
- 4) Enter **123456701** as the **ID** and "**CSD ResourceListExt**" as the name to extend the **Resource List** page and press enter
- 5) Enter documentation right underneath the pageextension line  
All documentation lines must be prefixed with // like this:  
**// CSD1.00 - 2018-01-01 - D. E. Veloper**
- 6) In the layout section, delete the line  
`// Add changes to page layout here`
- 7) Start by adding two new global variables
- 8) Go to the **var** section in the bottom of the file to create a global variable.
- 9) Delete the **myInt : Integer;** line and create a new line
- 10) Type a [ and press **Ctrl+[Space]** to get a list of options
- 11) Select the **inDataSet** option to make the variable available to the page as a part of the dataset.
- 12) Add a variable like this:  
**ShowMaxField: Boolean;**
- 13) Create a new line
- 14) Type a [ and press **Ctrl+[Space]** to get a list of options
- 15) Select the **inDataSet** option to make the variable available to the page as a part of the dataset.
- 16) Add a variable like this:  
**Showtype: Boolean;**

- 17) Above the **var** section, add a blank line and type **ttrigger**
- 18) In the **OnWhat** type **OnOpenPage** and press **[Enter]**
- 19) Delete the newly created **var** section, because there is no need for local variables
- 20) In the begin end block, type the following code:

```

trigger OnOpenPage();

begin

    FilterGroup(3);

    ShowType := (GetFilter(Type)='');

    ShowMaxField := (GetFilter(Type) =
                      format(Type::machine));

    FilterGroup(0);

end;

```

- 21) In the Layout section, add a blank line and type **m**, select **modify** and start typing **Type** and select the **Type** field when the Intellisense suggests it.
- 22) On the next line add a **{**, and VS Code will automatically add the matching **}**. After pressing Enter, the code will format automatically.
- 23) Add a new property:  
**Visible = ShowType;**
- 24) Outside the **modify(Type)** section add a new line and type  
**addafter("Type")**
- 25) On the next line add a **{**, and VS Code will automatically add the matching **}**. After pressing Enter, the code will format automatically.
- 26) Type **field** and end with the parenthesis
- 27) Start typing **"CSD Resource Type"** and select the **CSD Resource Type** field when the Intellisense suggests it.
- 28) On the next line add a **{**, and VS Code will automatically add the matching **}**. After pressing Enter, the code will format automatically.
- 29) Press enter to create a blank line and type **field**
- 30) Start typing **"CSD Maximum Participants"** and select the **CSD Maximum Participants** field when the Intellisense suggests it.
- 31) On the next line add a **{**, and VS Code will automatically add the matching **}**. After pressing Enter, the code will format automatically.
- 32) Add the following property:  
**Visible = ShowMaxField;**
- 33) Remove all unnecessary documentation lines
- 34) Remove the **action** section
- 35) Save the file by clicking **Ctrl+S**
- 36) Verify that the file looks like this:

## Solution Development in Visual Studio Code

---

```
pageextension 123456701 "CSD ResourceListExt" extends
"Resource List"

// CSD1.00 - 2018-01-01 - D. E. Veloper

// Chapter 5 - Lab 1-3

// Changed property on the Type field

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

// Added code to OnOpenPage trigger

{

    layout

    {

        modify(Type)
        {

            Visible = ShowType;
        }

        addafter(Type)
        {

            field("CSD Resource Type"; "CSD Resource Type")
            {

            }

            field("CSD Maximum Participants"; "CSD Maximum
Participants")
            {

                Visible = ShowMaxField;
            }
        }
    }
}
```

```

trigger OnOpenPage();

begin
    FilterGroup(3);
    ShowType := (GetFilter(Type)=' ');
    ShowMaxField :=
        (GetFilter(Type)=format(Type::machine));
    FilterGroup(0);
end;

var
    [InDataSet]
    ShowType : Boolean;
    [InDataSet]
    ShowMaxField : Boolean;
}

```

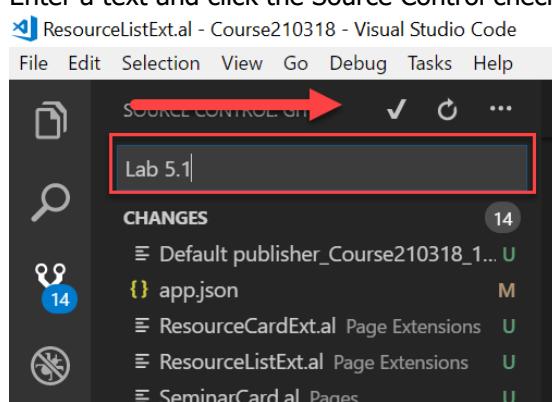
37) In order to keep the changes, it is now necessary to commit the changes and to push them to the off-site repository.

38) Click the menu item File/Save All

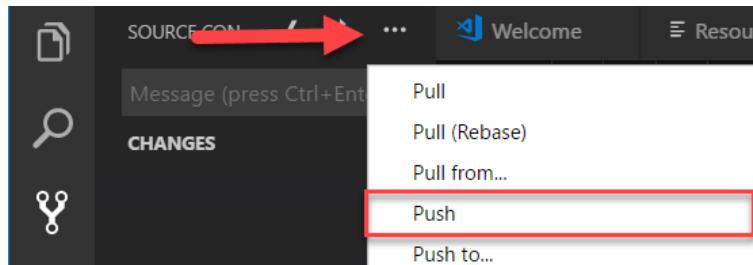


39) Next click the Source Control icon

40) Enter a text and click the Source Control check mark



41) Lastly Push the changes to Github



In the coming chapters this will only be described as  
"Commit the changes to Github"

## Seminars

There is no standard functionality in Dynamics 365 BC that handles Seminar Management requirements. You must develop a completely new application area. A standard application area in Dynamics 365 BC 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 an application area in Dynamics 365 BC. 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 an 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 Dynamics 365 BC 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

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.

```
trigger OnInsert();
var
    NoSeriesMgt: codeunit NoSeriesManagement;
    ResSetup : Record "Resources Setup";
begin
    if "No." = '' then begin
        ResSetup.GET;
        ResSetup.TestField("Resource Nos.");
        NoSeriesMgt.InitSeries(ResSetup."Resource Nos."
            ,xRec."No. Series",0D,"No.","No. Series");
    end;
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

```
procedure AssistEdit() : Boolean;
var
    NoSeriesMgt: codeunit NoSeriesManagement;
    ResSetup : Record "Resources Setup";
begin
    with Resource do begin
        Resource:=Rec;
        ResSetup.get;
        ResSetup.TestField("Resource Nos.");
        IF NoSeriesMgt.SelectSeries(ResSetup."Resource Nos."
            ,OldResource."No. Series","No. Series") then begin
            NoSeriesMgt.SetSeries("No.");
            Rec:=Resource;
            exit(true);
        end;
    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.

```
field(1;"No.");Code[20])
{
    trigger OnValidate();
    var
        NoSeriesMgt: codeunit NoSeriesManagement;
        ResSetup : Record "Resources Setup";
    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 the master records. 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.



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.



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

```
trigger OnModify();
begin
    "Last Date Modified" := TODAY;
end;
```

### 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 Dynamics 365 BC. 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 Dynamics 365 BC, 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 Dynamics 365 BC. 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 <b>Item</b> table when they enter lines in the <b>ItemJournal</b> .
List Page	CardPageID	Specifies which page shows when users double-click lines in the list page, or when users click the <b>New</b> , <b>Edit</b> , <b>View</b> , or <b>Delete</b> actions.
List Page	Editable	Must be set to <b>No</b> , 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.

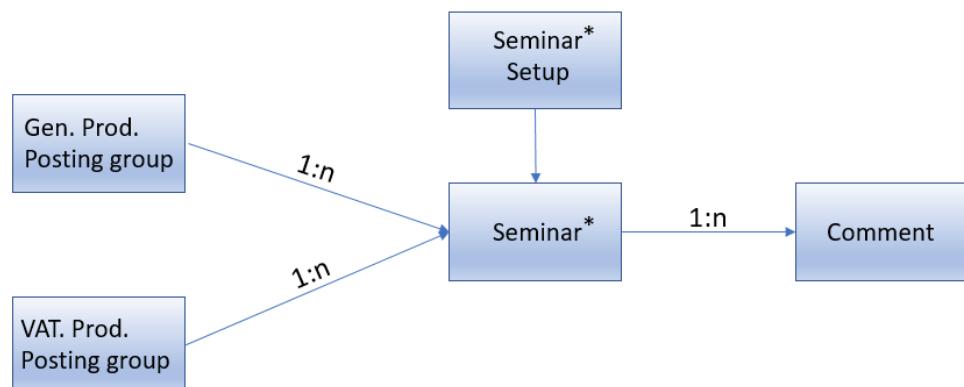
- On occasion, seminars can be overbooked, depending on the capacity of the assigned room.
- Each seminar can be canceled if there are not enough participants.
- The price of each seminar is fixed.

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.



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 Dynamics 365 BC tables.

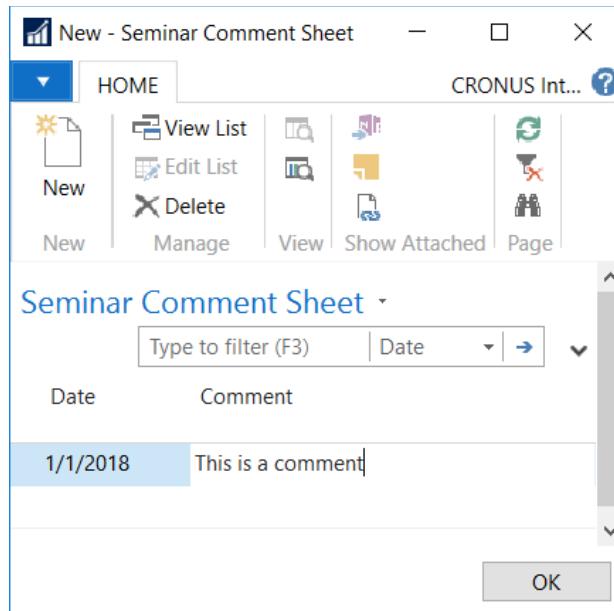
## Development

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 Dynamics 365 BC 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 Dynamics 365 BC, 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.

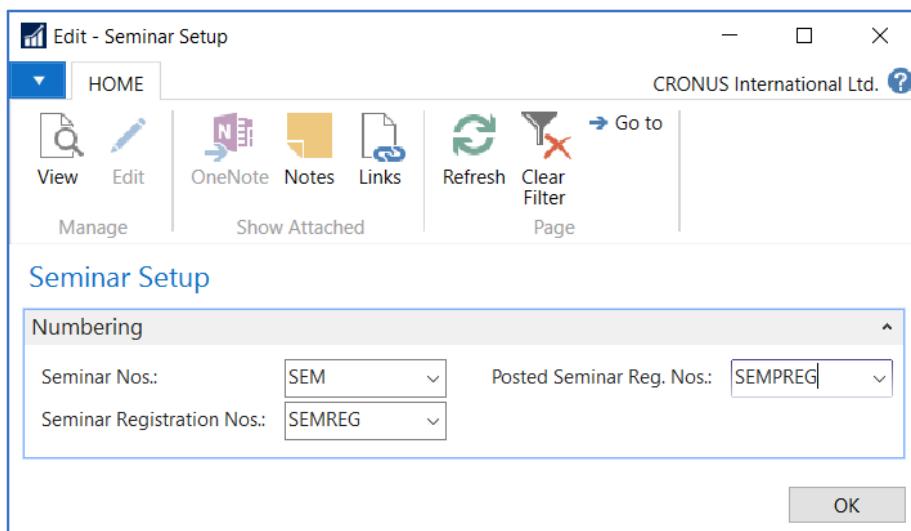
At the very minimum, all card and list pages for master tables must have **Links** and **Notes** FactBoxes.

The **Seminar Comment Sheet** page as shown below, allows users to enter and change the data in the **Seminar Comment Line** table. The similar looking **Seminar Comment List** page is used as Lookup Page and DrillDown Page.



The screenshot shows a Microsoft Dynamics application window titled "New - Seminar Comment Sheet". The ribbon bar at the top has a "HOME" tab selected. Below the ribbon are several buttons: "New" (with a file icon), "View List" (with a list icon), "Edit List" (with a list icon), "Delete" (with a delete icon), "Manage" (with a gear icon), "View" (with a magnifying glass icon), "Show Attached" (with a document icon), and "Page" (with a page icon). The main content area is titled "Seminar Comment Sheet" and contains a table with two columns: "Date" and "Comment". A single row is visible, showing "1/1/2018" in the Date column and "This is a comment" in the Comment column. A "Type to filter (F3)" input field and a "Date" dropdown are located above the table. A "OK" button is at the bottom right of the table area.

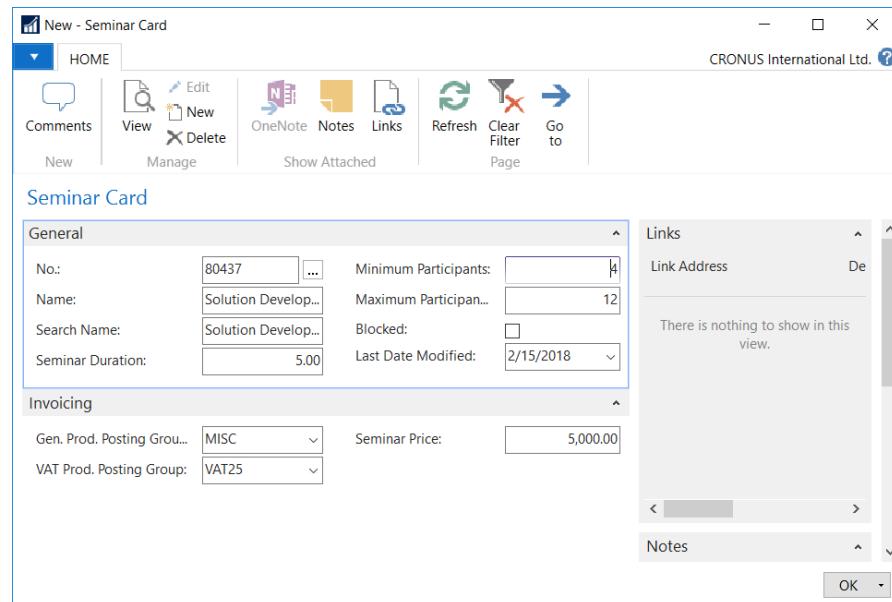
The **Seminar Setup** page as shown below, allows users to configure the Seminar Management application area.



The screenshot shows a Microsoft Dynamics application window titled "Edit - Seminar Setup". The ribbon bar at the top has a "HOME" tab selected. Below the ribbon are several buttons: "View" (with a magnifying glass icon), "Edit" (with a pencil icon), "OneNote" (with a notebook icon), "Notes" (with a yellow square icon), "Links" (with a link icon), "Refresh" (with a circular arrow icon), "Clear Filter" (with a filter icon), and "Page" (with a page icon). The main content area is titled "Seminar Setup" and contains a "Numbering" section. It includes two dropdown menus: "Seminar Nos.: SEM" and "Posted Seminar Reg. Nos.: SEMPREG". Below these are two more dropdown menus: "Seminar Registration Nos.: SEMREG" and another "Posted Seminar Reg. Nos." dropdown. A "OK" button is at the bottom right of the "Numbering" section.

## Solution Development in Visual Studio Code

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



**General**

No.:	80437	Minimum Participants:	4
Name:	Solution Develop...	Maximum Participants:	12
Search Name:	Solution Develop...	Blocked:	<input type="checkbox"/>
Seminar Duration:	5.00	Last Date Modified:	2/15/2018

**Invoicing**

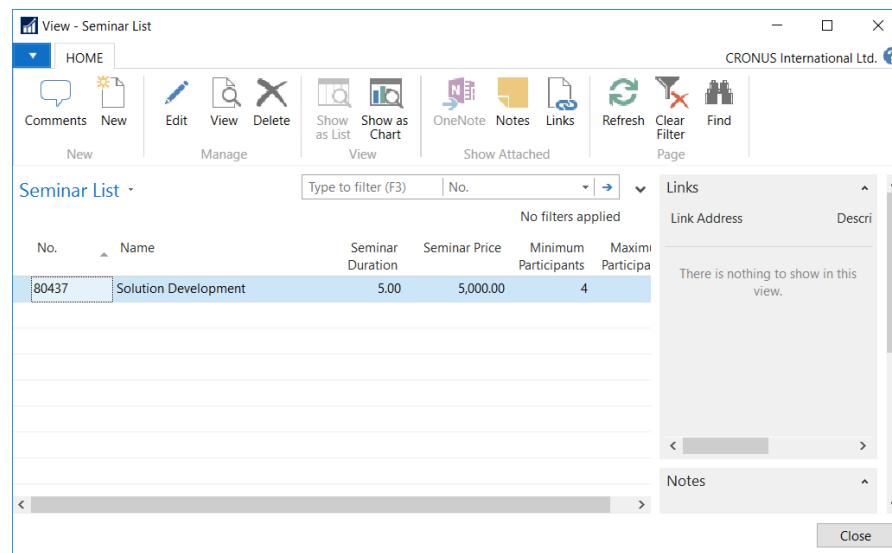
Gen. Prod. Posting Group:	MISC	Seminar Price:	5,000.00
VAT Prod. Posting Group:	VAT25		

**Links**  
Link Address

There is nothing to show in this view.

**Notes**

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.



**Seminar List**

No.	Name	Seminar Duration	Seminar Price	Minimum Participants	Maximum Participants
80437	Solution Development	5.00	5,000.00	4	12

**Links**  
Link Address

There is nothing to show in this view.

**Notes**

## Lab 5.2: 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 Dynamics 365 BC standards for setup tables, master tables, and card and list pages.

### 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.



In the following labs, the description of each task will be less detailed

### Task 1: Create the Seminar Setup Table

#### High Level Steps:

Create a new file **00\_SeminarSetup.al** with table **123456700 "CSD Seminar Setup"** and the following fields:

No.	Field Name	Type	Length	Remarks
10	Primary Key	Code	10	
20	Seminar Nos.	Code	20	Set the <b>TableRelation</b> property to the <b>No. Series</b> table
30	Seminar Registration Nos.	Code	20	Set the <b>TableRelation</b> property to the <b>No. Series</b> table
40	Posted Seminar Reg. Nos.	Code	20	Set the <b>TableRelation</b> property to the <b>No. Series</b> table

- Set the Primary Key to be "Primary Key"

## Solution Development in Visual Studio Code

---

### Detailed Steps:

#### Create the Seminar Setup Table

- 1) In the VS Code Explorer window, click the **Table** folder, then right-click and select the **New File** menu item
- 2) Name the file **00\_SeminarSetup.al** and press **[Enter]**
- 3) Enter **tt** in the first line and select the **ttable** snippet
- 4) Enter **123456700** as the **ID** and “**CSD Seminar Setup**” as the **name** and press enter
- 5) Add the following property:
  - a. **Caption = 'Seminar Setup'**
- 6) Add the fields described above in the fields section by:
  - a. Type **tfie** and select the **tfields** snippet
  - b. Enter the field number then press the **Tab** key
  - c. Type the name and then press the **Tab** key
  - d. Type the field type. Enter the length surrounded by **[]** on code and text fields. then press the **Tab** key
  - e. Replace the **FieldPropertyName = FieldPropertyValue;** line with properties for **Caption**
  - f. If needed add the **TableRelation** property
- 7) Locate the **keys** section and replace **MyField** with **Primary Key**
- 8) Delete the **var** section
- 9) Delete all triggers
- 10) Save the file using **Ctrl+S**
- 11) Verify that the solution looks like this

```
table 123456700 "CSD Seminar Setup"  
// CSD1.00 - 2018-01-01 - D. E. Veloper  
// Chapter 5 - Lab 2-1  
{  
    Caption = 'Seminar Setup';  
  
    fields  
    {  
        field(10;"Primary Key";Code[10])  
        {  
            Caption = 'Primary Key';  
        }  
    }  
}
```

```
field(20;"Seminar Nos.";Code[20])
{
    Caption = 'Seminar Nos.';
    TableRelation = "No. Series";
}

field(30;"Seminar Registration Nos.";code[20])
{
    Caption = 'Seminar Registration Nos.';
    TableRelation = "No. Series";
}

field(40;"Posted Seminar Reg. Nos.";code[20])
{
    Caption = 'Posted Seminar Reg. Nos.';
    TableRelation = "No. Series";
}

keys
{
    key(PK;"Primary Key")
    {
        Clustered = true;
    }
}
```

### Task 2: Create the Seminar Table

#### High Level Steps:

Create a new file **01\_Seminar.al** with table **123456701 "CSD Seminar"**

- Add the following fields:

No.	Field Name	Type	Length	Remarks
10	No.	Code	20	
20	Name	Text	50	
30	Seminar Duration	Decimal		Set the <b>DecimalPlaces</b> property to <b>0:1</b>
40	Minimum Participants	Integer		
50	Maximum Participants	Integer		
60	Search Name	Code	50	
70	Blocked	Boolean		
80	Last Date Modified	Date		Set the <b>Editable</b> property to <b>false</b>
90	Comment	Boolean		Set the <b>Editable</b> property to <b>false</b>
100	Seminar Price	Decimal		Set the <b>AutoFormatType</b> property to 1. This makes sure that the value is always formatted as an amount
110	Gen. Prod. Posting Group	Code	10	Set the <b>TableRelation</b> property to the <b>Gen. Product Posting Group</b> table
120	VAT Prod. Posting Group	Code	10	Set the <b>TableRelation</b> property to the <b>VAT Product Posting Group</b> table
130	No. Series	Code	10	Set the <b>Editable</b> property to <b>false</b> and the <b>TableRelation</b> property to the <b>No. Series</b> Table

- Configure the **Comment** field as a **FlowField** which shows if related records exist in the **Seminar Comment Line** table.
- Add keys to the **Seminar** table. Set the primary key to the **No.** field and add a secondary key for the **Search Name** field.
- Compile, save, and then close the table.

### Detailed Steps:

#### Create the Seminar Table

- 1) In the VS Code Explorer window, click the **Table** folder, then right-click and select the **New File** menu item
- 2) Name the file **01\_Seminar.al** and press **[Enter]**
- 3) Enter **tt** in the first line and select the **ttable** snippet
- 4) Enter **123456701** as the **ID** and "CSD Seminar" as the **name** and press enter
- 5) Add the following property:
  - Caption = 'Seminar'**
- 6) Add the fields described above in the fields section by:
  - Type **tfie** and select the **tfields** snippet
  - Enter the field number then press the **Tab** key
  - Type the name and then press the **Tab** key
  - Type the field type. Enter the length surrounded by **[]** on code and text fields. then press the **Tab** key
  - Replace the **FieldPropertyName = FieldPropertyValue;** line with properties for **Caption**
- 7) Locate the **Last Date Modified** field and the **No. Series** field and add the following property:

```
Editable=false;
```

- 8) Locate the **Comment** field and add the following properties:

```
Editable=false;
FieldClass=FlowField;
//CalcFormula=exist("Seminar Comment Line" where("Table
// Name"= const("Seminar"), "No."=Field("No."));
```



The code will not compile until the Seminar Comment Line table is created in Lab 5.3. Either leave the code with red underline or disable the lines by prefixing them with **//**. Then later they can be removed after completing Lab 5.3

- 9) Locate the **keys** section and replace **MyField** with "No."
- 10) Save the file using **Ctrl+S**
- 11) Verify that the file looks like this:

```
table 123456701 "CSD Seminar"
// CSD1.00 - 2018-01-01 - D. E. Veloper
// Chapter 5 - Lab 2-2
{
    Caption='Seminar';

    fields
    {
        field(10;"No.";Code[20])
        {
            Caption='No.';
        }
        field(20;Name;Text[50])
        {
            Caption='Name';
        }
        field(30;"Seminar Duration";Decimal)
        {
            Caption='Seminar Duration';
            DecimalPlaces=0:1;
        }
        field(40;"Minimum Participants";Integer)
        {
            Caption='Minimum Participants';
        }
    }
}
```

```
    field(50;"Maximum Participants";Integer)
    {
        Caption='Maximum Participants';
    }
    field(60;"Search Name";Code[50])
    {
        Caption='Search Name';
    }
    field(70;Blocked;Boolean)
    {
        Caption='Blocked';
    }
    field(80;"Last Date Modified";Date)
    {
        Caption='Last Date Modified';
        Editable=false;
    }
    field(90;Comment;Boolean)
    {
        Caption='Comment';
        Editable=false;
        //FieldClass=FlowField;
        //CalcFormula=exist("Seminar Comment Line"
        //where("Table Name"= const("Seminar"),
        //      "No."=Field("No.")));
    }
    field(100;"Seminar Price";Decimal)
    {
        Caption='Seminar Price';
        AutoFormatType=1;
    }
```

```
    field(110;"Gen. Prod. Posting Group";code[10])
    {
        Caption='Gen. Prod. Posting Group';
        TableRelation = "Gen. Product Posting Group";
    }
    field(120;"VAT Prod. Posting Group";code[10])
    {
        Caption='VAT Prod. Posting Group';
        TableRelation = "VAT Product Posting Group";
    }
    field(130;"No. Series";Code[10])
    {
        Editable=false;
        Caption='No. Series';
        TableRelation = "No. Series";
    }
}

keys
{
    key(PK;"No.")
    {
        Clustered = true;
    }
    key(Key1;"Search Name")
    {
    }
}
```



### Task 3: Add Code to the Seminar Table

#### High Level Steps:

1. Add the global variables for the **Seminar Setup**, **Seminar 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 as described in the design pattern previously.
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 as described in the design pattern previously.
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 global 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) Locate the global variables section (**var**) in the bottom of the file, remove the **myInt : Integer;** line and enter the following lines:

```
var
  SeminarSetup : Record "CSD Seminar Setup";
  //CommentLine : record "CSD Seminar Comment Line";
  Seminar : Record "CSD Seminar";
  GenProdPostingGroup: Record "Gen. Product Posting Group";
  NoSeriesMgt : Codeunit NoSeriesManagement;
```

- 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) In the **OnInsert** trigger, enter the following code:

```
trigger OnInsert();
begin
  if "No."=' ' then begin
    SeminarSetup.get;
    SeminarSetup.TestField("Seminar Nos.");
    NoSeriesMgt.InitSeries(SeminarSetup."Seminar
    Nos.",xRec."No. Series",0D,"No. ","No. Series");
  end;
end;
```

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

```
trigger OnModify();
begin
  "Last Date Modified":=Today;
end;
```

```
trigger OnRename();
begin
    "Last Date Modified":=Today;
end;
```

- 6) Add the code to the **OnDelete** trigger to delete any comment lines for the seminar record being deleted.

In the **OnDelete** trigger, enter the following code:

```
trigger OnDelete();
begin
    //CommentLine.Reset;
    //CommentLine.SetRange("Table Name",
    //CommentLine."Table Name":=Seminar);
    //CommentLine.SetRange("No.", "No.");
    // CommentLine.DeleteAll;
end;
```

- 7) Create an **OnValidate** trigger of the **No.** field, and 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.



The code will not compile until the Seminar Comment Line table is created in Lab 5.3. Either leave the code with red underline or disable the lines by prefixing them with `//`. Then later they can be removed after completing Lab 5.3

- 8) Locate the “**No.**” field after all properties, type **ttr** and select the **ttrigger** snippet.
- 9) Delete the **var** section, then enter the following code between the **begin** and **end**:

```

trigger OnValidate();
begin
  if "No." <> xRec."No." then begin
    SeminarSetup.GET;
    NoSeriesMgt.TestManual(SeminarSetup."Seminar Nos.");
    "No. Series" := '';
  end;
end;

```

- 10) Locate the **“Name”** field. After all properties, type **ttr** and select the **trigger** snippet.
- 11) Delete the **var** section, then enter the following code between the **begin** and **end**:

```

trigger OnValidate();
begin
  if ("Search Name"=UpperCase(xRec.Name)) or
    ("Search Name"='') then
    "Search Name":=Name;
end;

```

- 12) Create an **OnValidate** trigger of the **Gen. Prod. Posting Group** field, and 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.
- 13) In the **OnValidate** trigger of the **Gen. Prod. Posting Group** field, enter the following code:

```
trigger OnValidate();

begin
    if (xRec."Gen. Prod. Posting Group" <>
        "Gen. Prod. Posting Group") then begin
        if  GenProdPostingGroup.ValidateVatProdPostingGroup
            (GenProdPostingGroup, "Gen. Prod. Posting Group") then
            Validate("VAT Prod. Posting Group",
                     GenProdPostingGroup."Def. VAT Prod. Posting Group");
    end;
end;
```

- 14) In the Seminar table, create a new global 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**.
- 15) Go to the end of the code and create a new line just before the last }
- 16) Type **tpro** and select the **tprocedure** snippet
- 17) Remove the local in front of procedure to make the procedure global
- 18) Replace the **MyProcedure** with **AssistEdit**
- 19) Add **: Boolean** after the **()** to indicate a return value of the type **Boolean**
- 20) Remove the **var** section
- 21) Enter the following code in the procedure:

```
procedure AssistEdit() : Boolean;
begin
  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;
end;
```

- 1) Save the file using **Ctrl+S**

### Create the Seminar Pages

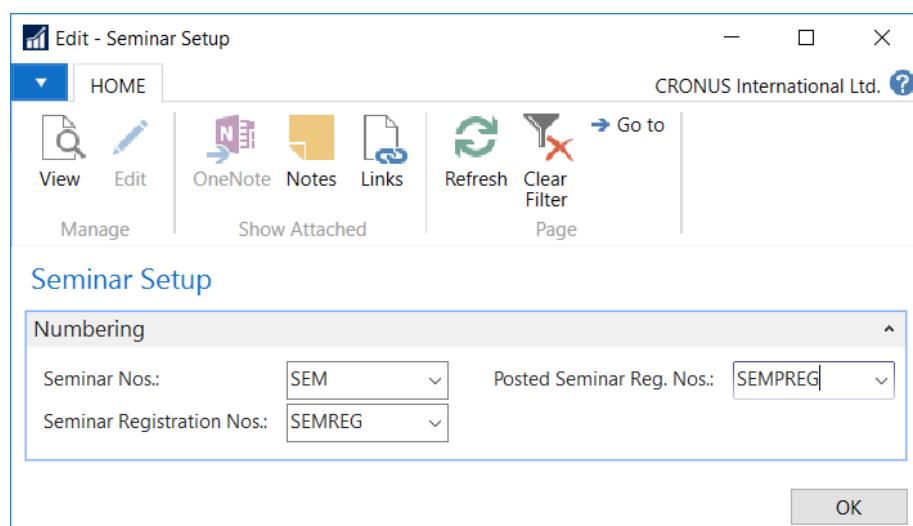
#### Exercise Scenario

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 Dynamics 365 BC master page functionality. This includes the RoleTailored client navigation and number series selection functionality.

### Task 4: Create the Seminar Setup Page

#### High Level Steps:

Create the “**CSD Seminar Setup**” page as shown below and assign the **ID** of **123456700** to it.



- Set properties so it is not possible to insert and delete records
- Set properties so the page is given the **Administration** category
- Add code to the **OnOpenPage** trigger to insert a new record, if there is not a record in the Seminar Setup table.
- Compile, save, and then close the page.

#### Detailed steps:

- 1) In the VS Code Explorer window, click the **Page** folder, then right-click and select the **New File** menu item
- 2) Name the file **00\_SeminarSetup.al** and press **[Enter]**
- 3) Enter **tpa** in the first line and select the **tpage, Page of type card** snippet
- 4) Enter **123456700** as the **ID** and “**CSD Seminar Setup**” as the **name** and press enter
- 5) Set the Source table to be “**CSD Seminar Setup**”
- 6) Add a **Caption** property “**Seminar Setup**”
- 7) Add the **InsertAllowed** property to be **false**
- 8) Add the **DeleteAllowed** property to be **false**
- 9) Add the **UsageCategory** to be **Administration**
- 10) Change the **group** name to be **Numbering**
- 11) Add the following fields to the page in the Numbering group:
  - a. Seminar Nos.

- b. Seminar Registration Nos.
  - c. Posted Seminar Reg. Nos.
- 12) Go to the end of the code and create a new line just before the last }  
13) Type **ttr** and select the **ttrigger** snippet  
14) Replace the **OnWhat** with **OnOpenPage and** add the following code to the trigger:

```
trigger OnOpenPage();

begin
    if not get then begin
        init;
        insert;
    end;
end;
```

- 15) Save the file using **Ctrl+S**

- 16) Verify that the file now looks like this:

```
page 123456700 "CSD Seminar Setup"
// CSD1.00 - 2018-01-01 - D. E. Veloper
// Chapter 5 - Lab 2-3
{
    PageType = Card;
    SourceTable = "CSD Seminar Setup";
    Caption='Seminar Setup';
    InsertAllowed = false;
    DeleteAllowed = false;
    UsageCategory = Administration;
```

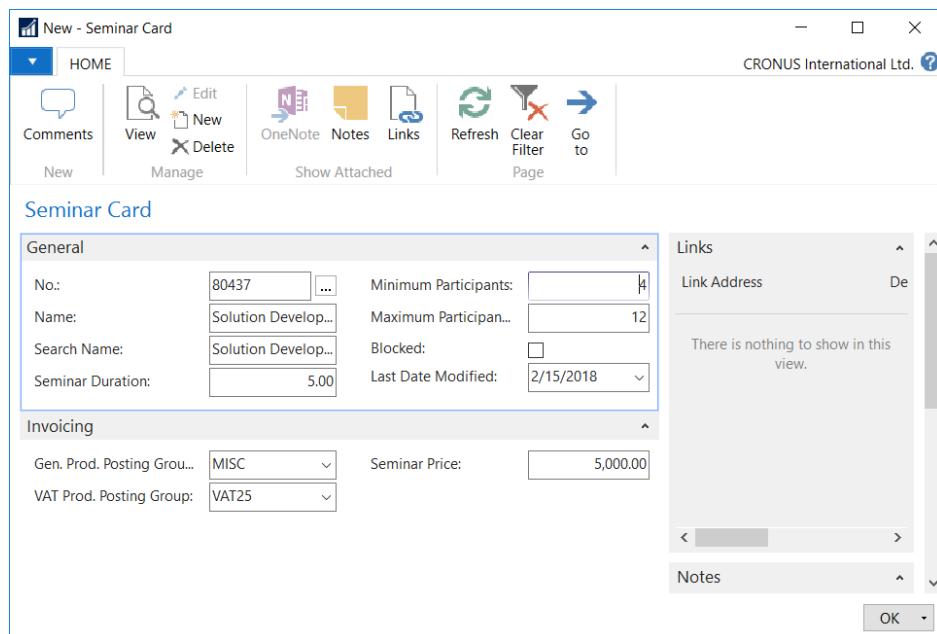
```
layout
{
    area(content)
    {
        group(Numbering)
        {
            field("Seminar Nos.;" "Seminar Nos.")
            {
            }
            field("Seminar Registration Nos.;" "Seminar Registration Nos.")
            {
            }
            field("Posted Seminar Reg. Nos.;" "Posted Seminar Reg. Nos.")
            {
            }
        }
    }
}

trigger OnOpenPage();
begin
    if not get then begin
        init;
        insert;
    end;
end;
```

## Task 5: Create the Seminar Card Page

### High Level Steps:

Create the “**CSD Seminar Card**” page as shown below and assign the **ID** of **123456701** to it.



- Add the fields to the page as shown above
- Add two FactBoxes to the page: **RecordLinks** and **Notes**
- 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**.
- Add an action to the page, to show the “**CSD Seminar Comment Sheet**” page for the seminar that is currently displayed in the page.
- Compile, save, and then close the page.

### Detailed Steps:

- 1) In the VS Code Explorer window, click the **Page** folder, then right-click and select the **New File** menu item
- 2) Name the file **01\_SeminarCard.al** and press **[Enter]**
- 3) Enter **tpa** in the first line and select the **tpage, Page of type card** snippet
- 4) Enter **123456701** as the **ID** and “**CSD Seminar Card**” as the **name** and press enter
- 5) Set the Source table to be “**CSD Seminar**”
- 6) Add a **Caption** property as **Seminar**
- 7) Change the **group** name to be **General**
- 8) Add the following fields to the page in the **General** group:
  - a. **No.**
  - b. **Name**
  - c. **Search Name**
  - d. **Seminar Duration**

- e. **Minimum Participants**
- f. **Maximum Participants**
- g. **Blocked**
- h. **Last Date Modified**

- 9) Add a new group after the **General** group and name it **Invoicing**
- 10) Add the following fields to the page in the **Invoicing** group:
  - a. **Gen. Prod. Posting Group**
  - b. **VAT Prod. Posting Group**
  - c. **Seminar Price**
- 11) Locate the end of the area (content) and create a new line after the **}**
- 12) Type **area(FactBoxes)** and press **[Enter]**
- 13) On the blank line type **{** and press **[Enter]**
- 14) Enter the two FactBoxes as shown below

```
area(FactBoxes)
{
    systempart("Links";Links)
    {
    }
    systempart("Notes";Notes)
    {
    }
}
```

- 15) Locate the **"No."** field and
- 16) Type **ttr** and select the **OnAssistEdit** trigger
- 17) In the trigger enter the following code:

```
field("No.;"No.")
{
    AssistEdit = true;
    trigger OnAssistEdit();
    begin
        if AssistEdit then
            CurrPage.Update;
    end;
}
```



If the Intellisense do not show the **AssistEdit** procedure, it is probably because, it has not been made global in the **Seminar** table.

- 18) To add an action to the page, to show the **Comment Sheet** page for the seminar that is currently displayed in the page, locate the **action** section of the page.
- 19) Replace the **Processing** action container with **Navigation**
- 20) Delete the default **action** section.
- 21) In the **area(Navigation)** section, add a new section: **group("&Seminar")**
- 22) In the **group("&Seminar")** Section add an **action("Co&mment")**
- 23) In the **action("Co&mment")** section add the following properties:

```
area(Navigation)
{
    group("&Seminar")
    {
        action("Comments")
        {
            RunObject=page "Seminar Comment Sheet";
            //RunPageLink = "Table Name"= const(Seminar),
            // "No."=field("No.");
            Image = Comment;
            Promoted = true;
            PromotedIsBig = true;
            PromotedOnly = true;
        }
    }
}
```



The Ampersand (&) represents the shortcut key to be used with the action



The code will not compile until the Seminar Comment Line table is created in Lab 5.3. Either leave the code with red underline or disable the lines by prefixing them with **//**. Then later they can be removed after completing Lab 5.3

- 24) Delete the **var** section
- 25) Save the file using **Ctrl+S**

- 26) Verify that the file looks like this:

```
page 123456701 "CSD Seminar Card"  
// CSD1.00 - 2018-01-01 - D. E. Veloper  
// Chapter 5 - Lab 2-4 & Lab 2-5  
{  
    PageType = Card;  
    SourceTable = "CSD Seminar";  
  
    layout  
    {  
        area(content)  
        {  
            group(General)  
            {  
                field("No."; "No.")  
                {  
                    trigger OnAssistEdit();  
                    begin  
                        if AssistEdit then  
                            CurrPage.Update;  
                    end;  
                }  
                field(Name; Name)  
                {  
                }  
                field("Search Name"; "Search Name")  
                {  
                }  
            }  
        }  
    }  
}
```

```
        field("Seminar Duration";
              "Seminar Duration")
        {
        }
        field("Minimum Participants";
              "Minimum Participants")
        {
        }
        field("Maximum Participants";
              "Maximum Participants")
        {
        }
        field(Blocked; Blocked)
        {
        }
        field("Last Date Modified";
              "Last Date Modified")
        {
        }
    }
    group(Invoice)
    {
        field("Gen. Prod. Posting Group";
              "Gen. Prod. Posting Group")
        {
        }
        field("VAT Prod. Posting Group";
              "VAT Prod. Posting Group")
        {
        }
    }
}
```

```
        field("Seminar Price"; "Seminar Price")
    {
        }
    }

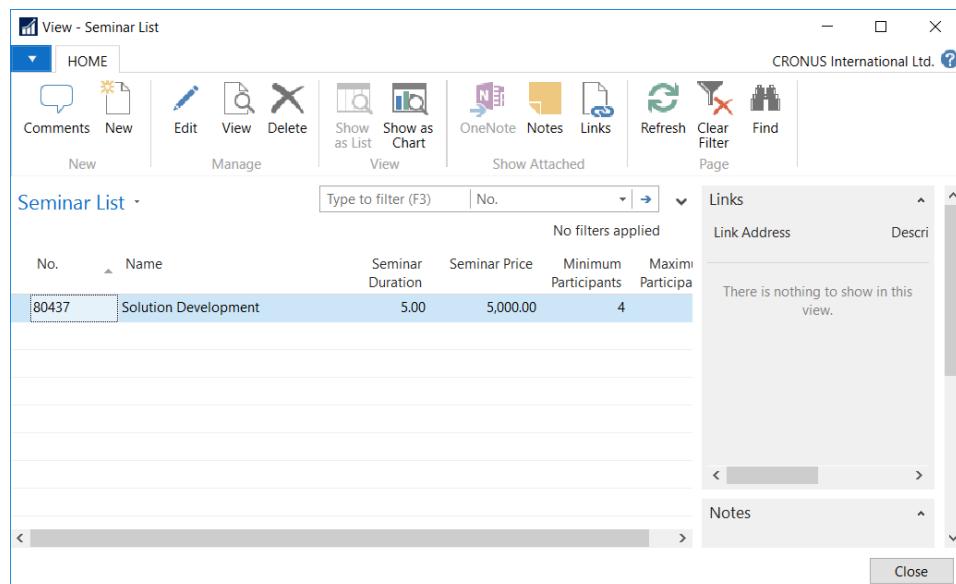
area(FactBoxes)
{
    systempart("Links"; Links)
{
}
systempart("Notes"; Notes)
{
}
}
```

```
actions
{
    area(Navigation)
    {
        group("&Seminar")
        {
            action("Comments")
            {
                RunObject=page "Seminar Comment Sheet";
                RunPageLink = "Table
Name"=const(Seminar),"No."=field("No.");
                Image = Comment;
                Promoted = true;
                PromotedIsBig = true;
                PromotedOnly = true;
            }
        }
    }
}
```

### Task 6: Create the Seminar List Page

#### High Level Steps:

Create the “**CSD Seminar List**” page as shown below and assign the **ID** of **123456702** to it.



- Add an action to the page to show the “**CSD Seminar Comment Sheet**” page for the seminar that is selected in the list.
- Add the **CardPageID** action to the list page
- Compile, save, and then close the page.
- Return to the **CSD Seminar** table and add the **LookupPageID** and **DrillDownPageID** properties to point to the **CSD Seminar List** page.
- Compile, save, and then close the page.

#### Detailed Steps:

- 1) In the VS Code Explorer window, click the **Page** folder, then right-click and select the **New File** menu item
- 2) Name the file **02\_SeminarList.al** and press **[Enter]**
- 3) Enter **tpa** in the first line and select the **tpage, Page of type list** snippet
- 4) Enter **123456702** as the **ID** and “**CSD Seminar List**” as the **name** and press enter
- 5) Set the Source table to be “**CSD Seminar**”
- 6) Add a **Caption** property
- 7) Add the **Editable** property to be **false**
- 8) Add the CardPageID to be **123456701**
- 9) Add the **UsageCategory** to be **Lists**
- 10) Add the following fields to the page in the **Repeater(Group)**:
  - a. **No.**
  - b. **Name**
  - c. **Seminar Duration**
  - d. **Seminar Price**
  - e. **Minimum Participants**
  - f. **Maximum Participants**

- 11) Locate the end of the area (content) and create a new line after the }
- 12) Copy the **area(FactBoxes)** from the Seminar Card page
- 13) Copy the **actions** section from the Seminar Card page
- 14) Save the file using **Ctrl+S**
- 15) Commit the changes to Github
- 16) Verify that the file looks like this:

```
page 123456702 "CSD Seminar List"
// CSD1.00 - 2018-01-01 - D. E. Veloper
// Chapter 5 - Lab 2-6
{
    Caption='Seminar List';
    PageType = List;
    SourceTable = "CSD Seminar";
    Editable = false;
    CardPageId = 123456701;
    UsageCategory = Lists;

    layout
    {
        area(content)
        {
            repeater(Group)
            {
                field("No."; "No.")
                {
                }
                field(Name; Name)
                {
                }
            }
        }
    }
}
```

```
        field("Seminar Duration";
              "Seminar Duration")
    {
    }

    field("Seminar Price"; "Seminar Price")
    {
    }

    field("Minimum Participants";
          "Minimum Participants")
    {
    }

    field("Maximum Participants";
          "Maximum Participants")
    {
    }

}

area(FactBoxes)
{
    systempart("Links"; Links)
    {
    }

    systempart("Notes"; Notes)
    {
    }

}
}
```

```
actions
{
    area(Navigation)
    {
        group("&Seminar")
        {
            action("Comments")
            {
                RunObject=page "Seminar Comment Sheet";
                RunPageLink = "Table
Name"=const(Seminar),"No."=field("No.");
                Image = Comment;
            }
        }
    }
}
```

- 17) Return to the **CSD Seminar** table and add the **LookupPageID** and **DrillDownPageID** properties to point to the **CSD Seminar List** page.
- 18) Compile, save, and then close the page.

### Lab 5.3: Creating Seminar Comment Line Table and Pages

#### Scenario

There are two types of comment lines in Dynamics 365 BC; The **Comment Line** table (97) is the table used with all the master data tables. An equivalent table exist for the document tables, e.g. the comment lines for the Sales module is called table (44) **Sales Comment Line**.

In the **Comment Line** table, the primary key contains an option field referencing which table the comment belongs to:

```
OptionMembers = "G/L Account", "Customer", "Vendor", "Item",  
"Resource", "Job", "", "Resource Group", "Bank Account2",  
"Campaign", "Fixed Asset", "Insurance", "Nonstock Item",  
"IC Partner";
```

The second part of the primary is the **"No."** field, which have been given a conditional table relation to all the different master tables they are related to.

In order to implement a **Comment Line** table with the Seminar Management module leaves two options:

- 1) If the design patterns should be followed to the T, it is necessary to extend the **Table Name** field with another option: "**Seminar**" and to extend the table relation to also include the **Seminar** table. However, it is not possible to modify the **OptionMembers** or the **TableRelation** property in VS Code, and the change has to be made in the classic development environment making an upgrade even more difficult.



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.



Never add options other than in the end of the option string. All options are saves in the database as Integer values and adding an option in the middle of the string will change the functionality for that option e.g. in the posting routines.

- 2) A new table can be created as a copy of the **Comment Line** (97) table and the **Seminar** option can be entered as the first option in the **Table Name** field. This requires a new table and two new pages to implement it in the Seminar Management Module. In order to implement a comment table for both the master data table and the coming document tables, an extra field is added to the part of the primary key: **Document Line No.**

Since changing the standard functionality of Dynamics 365 BC is out of the

question, we will implement solution two for the comment lines for the Seminar table.

## Task 1: Create the Seminar Comment Line Table

High Level Steps:

Create a new file **04\_SeminarCommentLine.al** with table **123456704 "CSD Seminar Comment Line"** and the following fields:

No.	Field Name	Type	Length	Remarks
10	Table Name	Option		Option members: Seminar,Seminar Registration Header,Posted Seminar Reg. Header
20	Document Line No.	Integer		
30	No.	Code	20	Conditional table relation: if ("Table Name"=CONST( Seminar)) "CSD Seminar"
40	Line No.	Integer		
50	Date	Date		
60	Code	Code	10	
70	Comment	Text	80	

Set the Primary Key to be **Table Name,Document Line No.,No.,Line No.**

Detailed Steps:

### Create the Seminar Setup Table

- 1) In the VS Code Explorer window, click the **Table** folder, then right-click and select the **New File** menu item
- 2) Name the file **04\_SeminarCommentLine.al** and press **[Enter]**
- 3) Enter **tt** in the first line and select the **ttable** snippet
- 4) Enter **123456704** as the **ID** and **"CSD Seminar Comment Line"** as the **name** and press enter
- 5) Add the following property:
  - a. **Caption = 'Seminar Comment Line'**
- 6) Add the fields described above in the fields section by:

- a. Type **tfie** and select the **tfields** snippet
  - b. Enter the field number then press the **Tab** key
  - c. Type the name and then press the **Tab** key
  - d. Type the field type. Enter the length surrounded by [] on code and text fields. then press the **Tab** key
  - e. Replace the **FieldPropertyName = FieldPropertyValue;** line with properties for **Caption**
- 7) Locate the “**No.**” field add the **TableRelation** property as a conditional table relation:

```
field(30;"No. ";Code[20])  
{  
    Caption='No.';  
    TableRelation;if ("Table Name"=CONST(Seminar)) "CSD  
Seminar";  
}
```

That way, it is prepared for later when the comments are to be used with the documents

- 8) Locate the **keys** section and replace **MyField** with **Table Name,Document Line No.,No.,Line No.**
- 9) Delete the **var** section
- 10) Delete all triggers
- 11) Save the file using **Ctrl+S**

- 12) Verify that the solution looks like this

```
table 123456704 "CSD Seminar Comment Line"  
{  
    Caption='Seminar Comment Line';  
  
    fields  
    {  
        field(10;"Table Name";Option)  
        {  
            Caption='Table Name';  
            OptionMembers="Seminar","Seminar Registration  
Header","Posted Seminar Reg. Header";  
            OptionCaption='Seminar,Seminar Registration  
Header,Posted Seminar Reg. Header';  
        }  
    }  
}
```

```
field(20;"Document Line No.";Integer)
{
  Caption='Document Line No.';

}
field(30;"No.";Code[20])
{
  Caption='No.';

  TableRelation=if ("Table Name"=CONST(Seminar))
    "CSD Seminar";
}
field(40;"Line No.";Integer)
{
  Caption='Line No.';

}
field(50;Date;Date)
{
  Caption = 'Date';

}
field(60;Code;Code[10])
{
  Caption = 'Code';

}
field(70;Comment;Text[80])
{
  Caption = 'Comment';

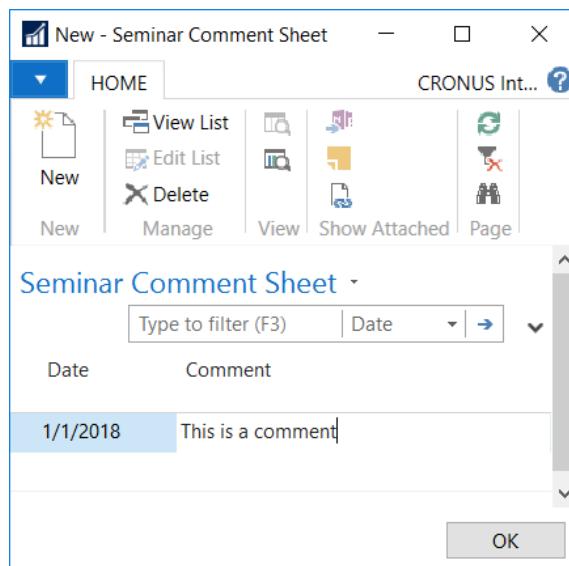
}
}
```

```
keys
{
  key(PK;"Table Name","Document Line No.", "No.", "Line No.")
  {
    Clustered = true;
  }
}
```

## Task 2: Create the Seminar Comment Sheet Page

### High Level Steps:

Create the “**CSD Seminar Comment Sheet**” page as shown below and assign the **ID** of **123456706** to it.



- Add the **Code** field after the **Date** field but set the **Visible** property to **false**
- Set properties so the page is given the **Tasks** category
- Compile, save, and then close the page.

Then create an exact copy of the “**CSD Seminar Comment Sheet**” page and name it **123456707 “CSD Seminar Comment List”**. Change the caption and make the page not editable.

### Detailed steps:

- 1) In the VS Code Explorer window, click the **Page** folder, then right-click and select the **New File** menu item
- 2) Name the file **06\_SeminarCommentSheet.al** and press **[Enter]**
- 3) Enter **tpa** in the first line and select the **tpage, Page of type list** snippet
- 4) Enter **123456706** as the **ID** and “**CSD Seminar Comment Sheet**” as the **name** and press enter
- 5) Set the Source table to be “**CSD Seminar Comment Line**”
- 6) Add a **Caption** property
- 7) Delete the **FactBoxes** area and the **actions** section
- 8) Add the following fields to the page in the **Repeater** group:
  - a. Date
  - b. Code
  - c. Comment



The Fastest way to add fields to a page is by typing:

**Field()**

**{**

**}**

And then copy that a number of times. Then all that needs filling is the field name, the expression will be added automatically.

## Solution Development in Visual Studio Code

---

- 9) Set the **Visible** property for the **Code** field to be **false**
- 10) Save the file using **Ctrl+S**
- 11) Verify that the file looks like this:

```
page 123456706 "CSD Seminar Comment Sheet"  
{  
    Caption = 'Seminar Comment Sheet';  
    PageType = List;  
    SourceTable = "CSD Seminar Comment Line";  
  
    layout  
    {  
        area(content)  
        {  
            repeater(Group)  
            {  
                field(Date;Date)  
                {  
                }  
                field(Code;Code)  
                {  
                    Visible=false;  
                }  
                field(Comment;Comment)  
                {  
                }  
            }  
        }  
    }  
}
```

- 12) Copy the **06\_SeminarCommentSheet.al** file by right-clicking on the file in the Explorer window
- 13) Then Right-Click the Pages folder and select **Paste**
- 14) Rename the new File to **07\_SeminarCommentList.al**

- 15) Change the ID to **123456707**
- 16) Change the name to "**CSD Seminar Comment List**"
- 17) Change the Caption property to **Seminar Comment List**
- 18) Add a new property **Editable = false**
- 19) Save the file using **Ctrl+S**
- 20) Return to the **CSD Seminar Comment Line** table and set the **CSD Seminar Comment List** Page as both the **LookupPageId** and the **DrillDownPageId**:

```
table 123456704 "CSD Seminar Comment Line"
{
    Caption='Seminar Comment Line';
    LookupPageId = "CSD Seminar Comment List";
    DrillDownPageId = "CSD Seminar Comment List";
```

- 21) Next, Make sure that all files are saved by clicking the **File/Save All** menu item
- 22) Then click the Source Control icon  The number might be different depending on the last commit.
- 23) Enter the text **Lab 5.3** in the Message field
- 24) Then click the Commit checkmark 
- 25) Lastly, Push the changes to the off-site repository

### 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 Dynamics 365 BC standards to the master tables and pages to make sure that users have a consistent experience across the application.

## Module 6: 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 Dynamics 365 BC. 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 Dynamics 365 BC, 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

The objectives are:

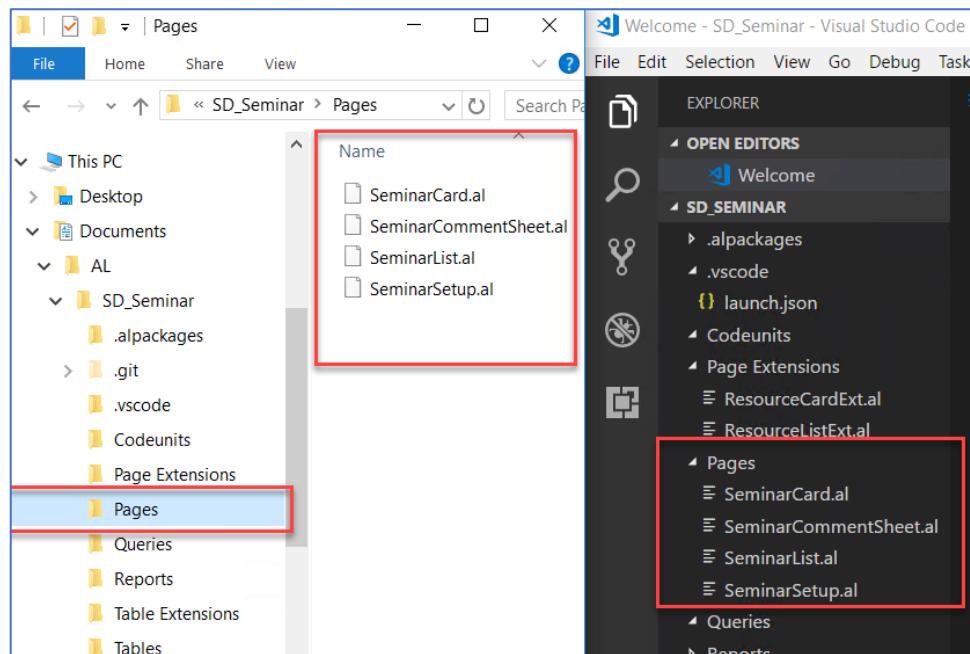
- Import and export of files to and from the workspace
- Using the Txt2AI Conversion Tool
- Multilanguage Functionality in Extensions
- Use document pages
- Use virtual tables
- Use temporary tables
- Review the various types of tables
- Review different page and table AL functions
- Create additional 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 Dynamics 365 BC.

### Import and export of files to and from the workspace.

The explorer window of VS Code contains the workspace of the project. The workspace is a mirror of the folder that was created in Lab 2.1, which means that all files created in the later labs are all represented in both the workspace and the folder:



The explorer window can be managed with dragging and dropping files into the workspace. It is possible to add files with other extensions than .al into the Explorer window e.g. a PowerShell files. In that case, the file will be included in the project but it will not be added to the app. This can be useful, if the PowerShell script is used with the project.



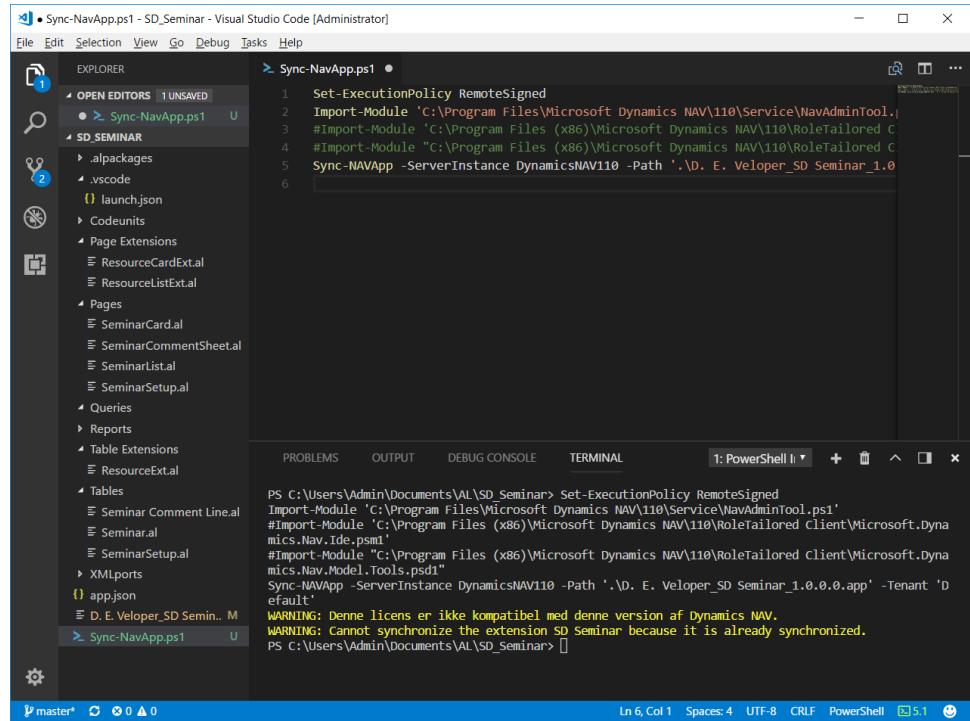
It is possible to install a PowerShell extension in VS Code so the script can be executed directly from VS Code



In order to execute PowerShell scripts directly from VS Code, The VS Code editor must be started in an elevated state by running it as administrator.



In some VS Code versions, it is not possible to drag and drop files directly to the Explorer window. In that case the folder must be opened with a File Explorer



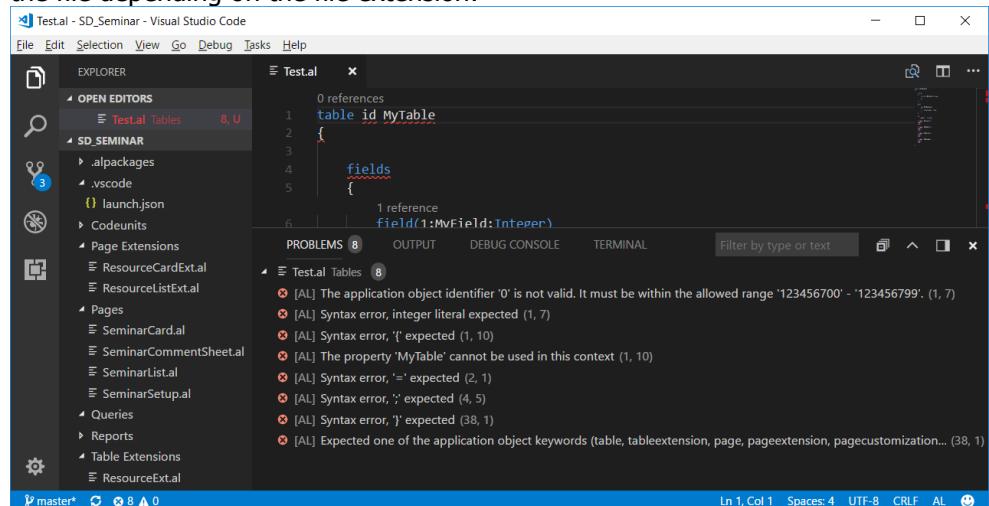
```

Sync-NavApp.ps1 - SD_Seminar - Visual Studio Code [Administrator]
File Edit Selection View Go Debug Tasks Help
EXPLORER OPEN EDITORS 1 UNSAVED
Sync-NavApp.ps1
SD SEMINAR
.alpackages
.vscode
.launch.json
Codeunits
Page Extensions
ResourceCardExt.al
ResourceListExt.al
Pages
SeminarCard.al
SeminarCommentSheet.al
SeminarList.al
SeminarSetup.al
Queries
Reports
Table Extensions
ResourceExt.al
Tables
Seminar Comment Line.al
Seminar.al
SeminarSetup.al
XMLports
app.json
D. E. Velooper_SD Seminar.. M
Sync-NavApp.ps1
Sync-NavApp.ps1 •
1 Set-ExecutionPolicy RemoteSigned
2 Import-Module 'C:\Program Files\Microsoft Dynamics NAV\110\Service\NavAdminTool.ps1'
#Import-Module 'C:\Program Files (x86)\Microsoft Dynamics NAV\110\RoleTailored client\Microsoft.Dynamics.Nav.Ide.psml'
#Import-Module 'C:\Program Files (x86)\Microsoft Dynamics NAV\110\RoleTailored client\Microsoft.Dynamics.Nav.Model.Tools.psdt'
Sync-NAVApp -ServerInstance DynamicsNAV110 -Path '.\D. E. Velooper_SD Seminar_1.0.app' -Tenant 'D default'
WARNING: Denne licens er ikke kompatibel med denne version af Dynamics NAV.
WARNING: Cannot synchronize the extension SD Seminar because it is already synchronized.
PS C:\Users\Admin\Documents\AL\SD_Seminar>

```

Adding files that already exist in the workspace will give a warning before overwriting the existing file.

After adding the file to the VS Code workspace, VS Code will immediately validate the file depending on the file extension:



```

Test.al - SD_Seminar - Visual Studio Code
File Edit Selection View Go Debug Tasks Help
EXPLORER OPEN EDITORS
Test.al Tables 8, U
SD SEMINAR
.alpackages
.vscode
.launch.json
Codeunits
Page Extensions
ResourceCardExt.al
ResourceListExt.al
Pages
SeminarCard.al
SeminarCommentSheet.al
SeminarList.al
SeminarSetup.al
Queries
Reports
Table Extensions
ResourceExt.al
Test.al •
1 references
table id MyTable
2 {
3
4   fields
5   {
6     1 reference
7       field:1:MvField:Integer.
PROBLEMS 8 OUTPUT DEBUG CONSOLE TERMINAL
Filter by type or text
8
1 [AL] The application object identifier '0' is not valid. It must be within the allowed range '123456700' - '123456799'. (1, 7)
2 [AL] Syntax error, integer literal expected (1, 7)
3 [AL] Syntax error, '[' expected (1, 10)
4 [AL] The property 'MyTable' cannot be used in this context (1, 10)
5 [AL] Syntax error, '=' expected (2, 1)
6 [AL] Syntax error, ';' expected (4, 5)
7 [AL] Syntax error, ';' expected (38, 1)
8 [AL] Expected one of the application object keywords (table, tableextension, page, pageextension, pagecustomization... (38, 1)

```

It is not possible to publish and install the package as long as there is an error in one of the .al files.



If it is necessary to publish and install a package that contains an incomplete .al file, the file can be renamed to e.g. .xal. then it will be excluded from the project.

It is possible to handle multiple files using the **Shift** or **Ctrl** keys with the mouse, but only with VS Code version 1.20 and later.

### Using the Txt2AI Conversion Tool

The Txt2AI conversion tool allows you to take existing Dynamics NAV objects that have been exported in **.txt** format and convert them into the new **.al** format. The **.al** format is used when developing extensions for Dynamics 365 BC. Converting the objects consists of following two steps:

- Exporting the objects from C/SIDE in a cleaned format.
- Splitting the objects into separate files
- Converting the objects to the new syntax.

The txt2al.exe file was installed together with the “Modern Development Environment” when Dynamics 365 BC was installed.

The program has been placed in the folder:

**C:\Program Files (x86)\Dynamics NAV\110\RoleTailored Client**

To perform the conversion using the “Classic” Development Environment:

- Export the objects as txt file.
- Open a command prompt as administrator
- Run the tool from the command line using the following syntax:  
`txt2al --source --target --rename --type --extensionStartId`

After the conversion, the **.al** files can be dragged directly into the workspace.

There might be a number of problems in the conversion that needs addressing, they will be listed during conversion:

```
The value of property Name of element with ID 1902613707 has
not been correctly exported by C\SIDE.

The value of property Name of element with ID 1900383207 has
not been correctly exported by C\SIDE.

The value of property Name of element with ID 1905767507 has
not been correctly exported by C\SIDE.

The value of property Name of element with ID 140 has not
been correctly exported by C\SIDE.
```

The parameters used with the txt2al.exe program is:

Parameter	Description
<code>--source=Path</code>	Required. The path of the directory containing the delta files.
<code>--target=Path</code>	Required. The path of the directory into which the converted AL files will be placed.
<code>--rename</code>	Rename the output files to prevent clashes with the source <b>.txt</b> files.

--type=ObjectType	The type of object to convert. Allowed values: Codeunit, Table, Page, Report, Query, XmlPort
--extensionStartId	The starting numeric ID of the extension objects (Default: 70000000). It will be incremented by 1 for each extension object.
--help	Show help screen.

The process can be automated using a PowerShell script in VS Code:

```
Set-ExecutionPolicy RemoteSigned

# Import the module for the Export-NAVApplicationObject
cmdLet

Import-Module 'C:\Program Files (x86)\Microsoft Dynamics
NAV\110\RoleTailored Client\Microsoft.Dynamics.Nav.Ide.psm1'

# Import the module for the Split-NAVApplicationObjectFile
cmdLet

Import-Module 'C:\Program Files (x86)\Microsoft Dynamics
NAV\110\RoleTailored
Client\Microsoft.Dynamics.Nav.Model.Tools.psd1'

# Export objects from database

Export-NAVApplicationObject -DatabaseServer 'localhost' -
DatabaseName 'Demo Database NAV (11-0)' -Filter
'Type=Page;Id=21..22' -Path 'c:\temp\Al-
Conversion\Allobjects.txt' -Username 'administrator' -
Password 'password'

# Split the txt file into separate txt files per object

Split-NAVApplicationObjectFile -Source 'C:\temp\Al-
Conversion\Allobjects.txt' -Destination 'c:\temp\Al-
Conversion\cal'

# Create the .al files

$Command = "C:\Program Files (x86)\Microsoft Dynamics
NAV\110\RoleTailored Client\txt2al.exe"

& $Command --source c:\temp\Al-Conversion\cal --target
c:\temp\Al-Conversion\al
```



VS Code must be run as administrator, and the Development Environment must be installed, with access to the Dynamics 365 BC database in order to execute the script

### Multilanguage Functionality in Extensions

In previous versions of Dynamics NAV, it was necessary to maintain all languages directly in the Development Environment. When you created messages for the user, you had to make sure that the text and the object names in the messages were enabled for multilanguage functionality.

If your code had to display any errors, confirmations, or messages, the text messages could not be entered directly in the C/AL code. This made the code dependent on the language that you used initially. Any users who run Dynamics NAV in another language would be unable to understand the messages.

In order to display text in a dialog box, a page, or a report, you had to define such text as a text constant. It was imperative that the **Caption** and the **CaptionML** properties were filled in order to be able to export the language layer to a file for translation.

Dynamics 365 BC handles multilanguage in a different way. Instead of storing all texts in different languages in the code, the texts are stored in XLIFF files. XLIFF (XML Localization Interchange File Format) is an XML-based format created to standardize the way localizable data are passed between tools during a localization process and a common format for CAT tool files.

XLIFF forms part of the Open Architecture for XML Authoring and Localization (OAXAL) reference architecture.

You can still display the user interface (UI) in different languages, but the support for using the **ML** properties, such as **CaptionML** and **TooltipML**, is being deprecated, so it is recommended to refactor all extensions to use the corresponding properties, such as **Caption** or **Tooltip**, which is being picked up in the .xlf file.



You can use the new translation files approach only for objects from your extension. For translating the base application, you still need to use the .TXT files approach

To add a new language to the extension you have built, you must first enable the generation of XLIFF files. The XLIFF file extension is .xlf. The generated XLIFF file contains the strings that are specified in properties such as **Caption** and **Tooltip**.

#### Generating the XLIFF file

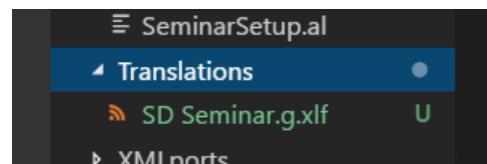
To enable generation of the translation file, you must add a setting in the manifest. In the **app.json** file of your extension, add the following line:

```

{} app.json •
1   {
2     "id": "2e41914a-024b-4e7e-a8cc-0c3c9f38a563",
3     "name": "SD Seminar",
4     "publisher": "D. E. Veloper",
5     "brief": "A New Seminar module for managing seminars",
6     "description": "",
7     "version": "1.0.0.0",
8     "privacyStatement": "",
9     "EULA": "",
10    "help": "",
11    "url": "",
12    "logo": "",
13    "capabilities": [],
14    "dependencies": [],
15    "screenshots": [],
16    "platform": "11.0.0.0",
17    "application": "11.0.0.0",
18    "features": ["TranslationFile"],
19    "idRange": {
20      "from": 123456700,
21      "to": 123456799
22    }
23  }

```

Next time the package is created, a new folder Translations and a .xlf file has been created as well:



Clicking the file, shows all the captions in the special .xlf format:

```

{} app.json SD Seminar.g.xlf •
1  <?xml version="1.0" encoding="utf-8"?>
2  <xliff version="1.2" xmlns="urn:oasis:names:tc:xliff:document:1.2" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="urn:oasis:names:tc:xliff:document:1.2 http://www.oasis-open.org/committees/standards/xliff_1_2.xsd">
3    <file datatype="xml" source-language="en-US" original="SD Seminar">
4      <body>
5        <group id="body">
6          <trans-unit id="Page 2391461621 - Property 2879900210" maxwidth="999" size-unit="px">
7            <source>Seminar Comment Sheet</source>
8            <note from="Developer" annotates="general" priority="2" />
9            <note from="Xliff Generator" annotates="general" priority="3">Page - Page</note>
10           </trans-unit>
11           <trans-unit id="Page 1173659623 - Property 2879900210" maxwidth="999" size-unit="px">
12             <source>Seminar List</source>

```

But in order to actually work with the XLIFF file, it is necessary to install a special tool.



You can have only one .xlf file per language. If you translate your extension to multiple languages, you must have a translation file per language. There is no enforced naming on the file, but a suggested good practice is to name it <extensionname>.<language>.xlf.

Microsoft does not recommend any specific translation tool, but a couple of choices could be:

- Pootle
- Weblate
- PoEditor
- xliff.brightec.co.uk
- crowdin.net

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

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

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

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

Type	Naming Convention	Example
Document Table (Header)	Name of Transaction or Document + Header	<b>Sales Header</b> (Table 36)
Document Table (Line)	Name of Transaction or Document + Line	<b>Sales Line</b> (Table 37)
Document Page	Name of Document Represented	<b>Sales Invoice</b> (Page 43)
Document Subpage	Name of Document Represented + Subform	<b>Sales Invoice Subform</b> (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 AL code in page objects. The following table presents the page functions.

Function	Remarks
<b>CurrPage.SAVERECORD</b>	Saves the current record to the database.
<b>CurrPage.UPDATE</b>	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.
<b>CurrPage.SETSELECTIONFILTER</b>	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.
<b>CurrPage.ACTIVATE</b>	Brings the current page into the focus of the user.
<b>CurrPage.CLOSE</b>	Closes the current page.

## Virtual Tables

A virtual table contains system information. You cannot change the data in virtual

tables. You can only read the information. Virtual tables are not stored in the database but are computed by Dynamics NAV at run time.

### Using Virtual Tables

You can use the same methods to access information in virtual tables as you use when you are working with ordinary tables. For example, you can use filters to get subsets or ranges of integers or dates from the Integer virtual table or the Date virtual table.

A system administrator usually uses these virtual tables. These tables give the system administrator information about the users who are currently connected to the database and the current state of the system.

The virtual tables provide such information as:

- Integers in the range –1,000,000,000 to 1,000,000,000.
- Dates in a given period.
- Overview of the operating system files.
- Overview of the logical disk drives.
- Overview of the operating system files that store the database.

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
<b>Object</b>	Contains the list of all objects in a Dynamics NAV 2013 application.
<b>Date</b>	Contains the list of all periods (days, weeks, months, quarters, or years) between January 03, 0001, and December 31, 9999.
<b>Integer</b>	Contains the list of all integer numbers.
<b>AllObjWithCaption</b>	Contains the list of all objects in a Dynamics NAV 2013 application, together with their captions.

<b>Field</b>	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
<b>Period Type</b>	The type of the period (Day, Week, Month, Quarter, Year).
<b>Period Start</b>	The date of the first day in the period.
<b>Period End</b>	The date of the last day in the period.
<b>Period No.</b>	The number of the period within the parent. For example, any period of type Day, where the day is Monday, has the <b>Period No.</b> equal to 1; or any period of type Month, where the month is February, has the <b>Period No.</b> equal to 2.
<b>Period Name</b>	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.

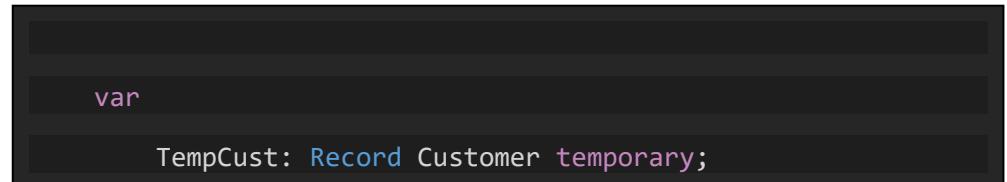
The write transaction principle that applies to ordinary database tables does not apply to temporary tables: **COMMIT** does not affect temporary tables, and **ERROR** does not roll back any earlier changes to the data in the temporary tables.

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 global **var** or the locals **var** section, define a variable with the data type Record. Select a table as the subtype.
2. Add the property temporary after the table name



```
var
TempCust: Record Customer temporary;
```

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 Dynamics 365 BC when you create a new database. The system tables track different system-related information. Following are a few examples:

- Security permissions
- Object metadata
- Record links
- Charts

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

## Registrations

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

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

## Solution Design

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

- Users must be able to schedule seminars. Each seminar has a starting date, an allocated seminar room, an assigned instructor, the minimum and the maximum participants, and the price. The minimum participants and the price information are always taken from the seminar master record. The maximum participants are taken as the lower number of maximum participants of the seminar and maximum participants of the room.
- A seminar cannot be scheduled in a room that cannot hold at least the minimum number of participants for the seminar.
- For each scheduled seminar, users must enter additional comments, such as necessary equipment, or other special requirements.
- It must be possible to assign additional expenses to a scheduled seminar, such as catering expenses or equipment rentals.

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

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

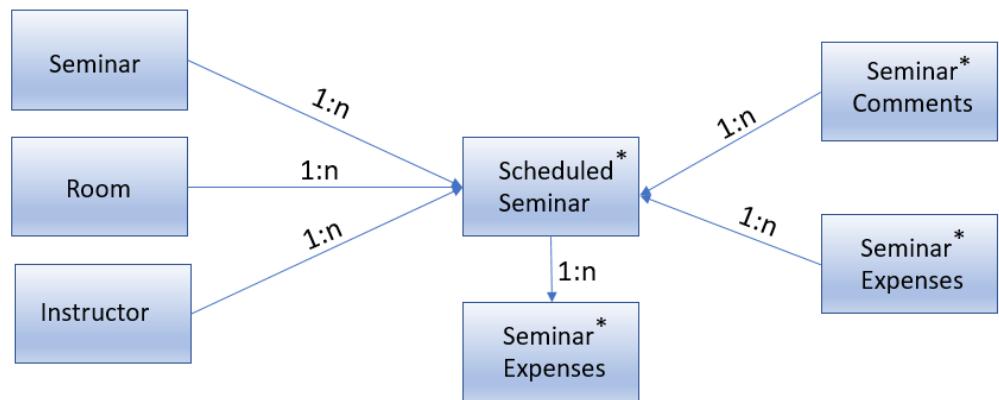
- Users must be able to register one or more participants for scheduled seminars. For each registered participant, it must be possible to specify if additional expenses must be invoiced for this registration. The default is Yes.
- If the room maximum capacity exceeds the maximum participants that are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

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

## Solution Development in Visual Studio Code

- 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.

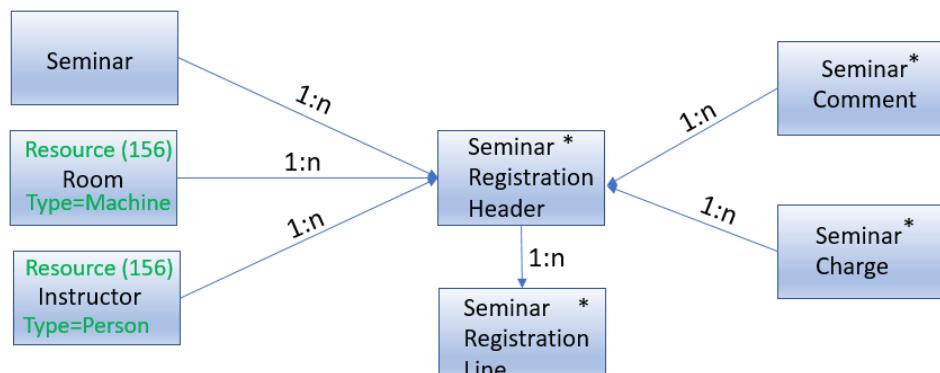


The diagram suggests that the best solution for managing registrations is to use the document functionality of Dynamics 365 BC with the following tables:

- 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 (\*).



## 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 Dynamics 365 BC principles for document tables and pages. You must provide all of the functionality that users experience with other document pages elsewhere in the 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

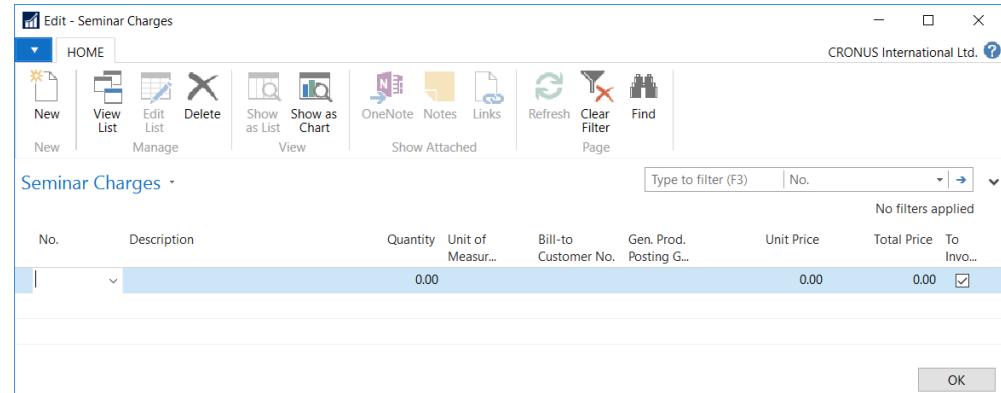
To support the seminar registration functionality, you must develop the following new tables.

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

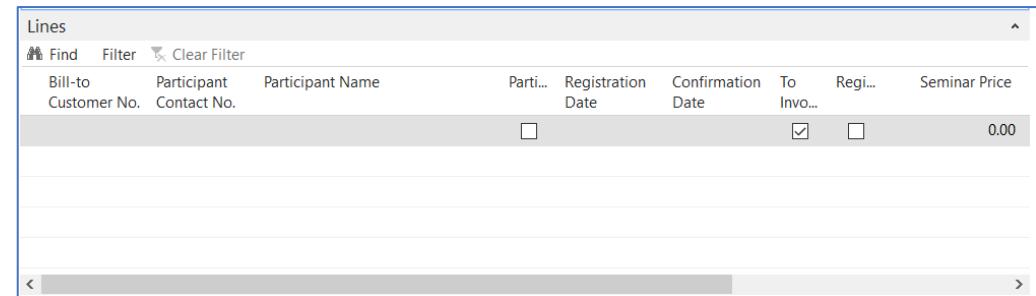
The **Seminar Charges** page as shown below permits the entry of charges for a seminar.



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 Reg. Subpage** figure shows the Seminar Reg. Subpage embedded in the **Seminar Registration** page. 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.



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

The following image shows the **Seminar Details FactBox** page embedded in the **Seminar Registration** page. This page is never used directly. Use this page as a page part on the Seminar Registration List page.

### Seminar Registration

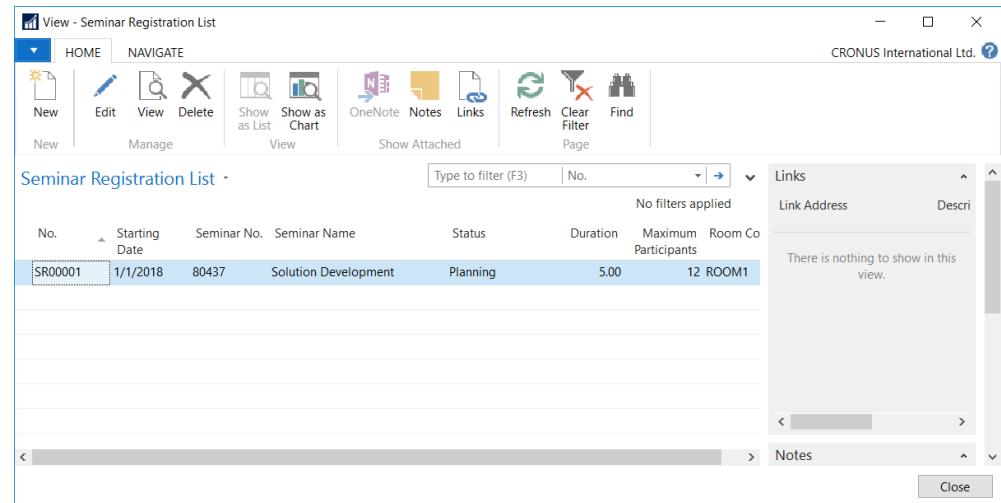
The screenshot shows the 'Seminar Registration' page. On the left, there is a 'General' section with fields for No., Starting Date, Seminar No., Seminar Name, Instructor Code, Instructor Name, Posting Date, Document Date, Status, Duration, Minimum Participants, and Maximum Participants. On the right, there is a 'Seminar Details' factbox with fields for No., Name, Seminar Duration, Minimum Participants, Maximum Participants, and Seminar Price. A red box highlights the 'Seminar Details' factbox.

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 screenshot shows the 'New - Seminar Registration - SR00001' page. The top navigation bar includes 'HOME', 'NAVIGATE', and 'CRONUS International Ltd.' with a help icon. The main area starts with a 'General' section containing fields for No. (SR00001), Starting Date (1/1/2018), Seminar No. (80437), Seminar Name (Solution Development), Instructor Code (LINDA), Instructor Name (Linda Martin), Posting Date (1/23/2020), Document Date (1/23/2020), Status (Planning), Duration (5.00), Minimum Participants (4), and Maximum Participants (12). Below this is a 'Lines' section with a table showing a single row: Bill-to Customer No. (10000), Participant Contact No. (CT100140), Participant Name (David Hodgson), Part... (checkbox), Registration Date (1/23/2020), and Confirmation Date (checkbox). At the bottom is a 'Seminar Room' section with fields for Room Code (ROOM1) and Room Post Code. A factbox on the right titled 'Seminar Details' shows the same data as the General section. Another factbox titled 'Links' is present but empty. A message in the 'Links' factbox says 'There is nothing to show in this view.'

## Solution Development in Visual Studio Code

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



The screenshot shows a web-based application titled "View - Seminar Registration List". The interface includes a top navigation bar with "HOME" and "NAVIGATE" buttons, and a ribbon menu with "New", "Edit", "View", "Delete", "Show as List", "Show as Chart", "View", "OneNote", "Notes", "Links", "Refresh", "Clear Filter", and "Find" options. A search bar at the top right says "Type to filter (F3)" and "No. [dropdown]". Below the search bar, a message says "No filters applied". The main content area displays a table titled "Seminar Registration List" with the following data:

No.	Starting Date	Seminar No.	Seminar Name	Status	Duration	Maximum Participants	Room Co
SR00001	1/1/2018	80437	Solution Development	Planning	5.00	12	ROOM1

On the right side of the screen, there are two expandable sections: "Links" and "Notes". The "Links" section is collapsed and displays the message "There is nothing to show in this view." The "Notes" section is also collapsed and displays the message "There is nothing to show in this view." A "Close" button is located at the bottom right of the "Notes" section.

## Lab 6.1: Importing, Reviewing and Completing Seminar Registration Tables

### Scenario

another developer at your company, who works on the implementation project of Dynamics 365 BC at CRONUS International Ltd has created tables to manage seminar registrations. You import the files into the workspace and review the tables to make sure that they follow all Dynamics 365 BC standards. You make any necessary corrections to the tables, their properties and fields, and their code.

### Import the Starter Objects

#### Exercise Scenario

Import the objects from the text file provided. You know that the text file import overwrites any objects in the database. Therefore, you first review the contents of the file to make sure that it only creates new objects.

### Task 1: Review the .al Files

#### High Level Steps:

Review the .al files from the folders:

**Solution Development Course Objects\Mod06\Starter**

without importing them to the workspace.

Check that the files contain the following objects:

Type	Object Id	Data Type
Table	123456710	CSD Seminar Registration Header
Table	123456711	CSD Seminar Registration line
Table	123456712	CSD Seminar Charge
Page	123456710	CSD Seminar Registration
Page	123456711	CSD Seminar Reg. Subpage
Page	123456713	CSD Seminar Registration List
Page	123456717	CSD Seminar Details FactBox
Page	123456724	CSD Seminar Charges

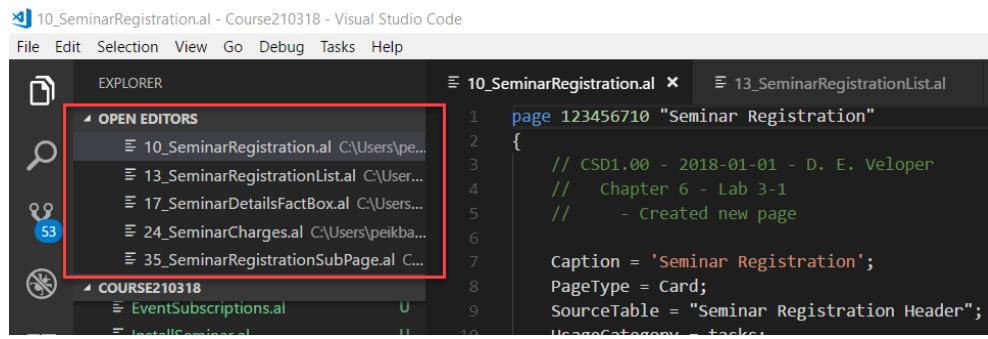
- Make sure that no imported objects exist in the database.

#### Detailed Steps:

##### Investigate the .al files from on the desktop

- 1) In the VS Code Explorer window, make sure that all files are saved and there are no open files in the editor
- 2) Click the **File/Open File** menu item
- 3) Open the **Solution Development Course Objects\Mod06\Starter\Tables** folder on the Desktop and select all the files and click OK

## Solution Development in Visual Studio Code



```
1 page 123456710 "Seminar Registration"
2 {
3 // CSD1.00 - 2018-01-01 - D. E. Developer
4 // Chapter 6 - Lab 3-1
5 // - Created new page
6
7 Caption = 'Seminar Registration';
8 PageType = Card;
9 SourceTable = "Seminar Registration Header";
10 UsageCategory = Task;
```

- 4) This will open the files in VS Code and give you a chance of investigating the files without changing the project.



Opening the files the in the editor does not make them part of the workspace and if the package is published the files will be ignored. In order to include them to the workspace, they must be copied to the workspace manually

- 5) Note down all objects the object id and type that are contained in the file.

Type	Object Id	Data Type
Table	123456710	CSD Seminar Registration Header
Table	123456711	CSD Seminar Registration line
Table	123456712	CSD Seminar Charge

- 6) Make sure that no imported objects exist in the database
- 7) Close all the files again
- 8) In the VS Code **Explorer** window, Click the **File/Open File** menu item
- 9) Open the **Solution Development Course Objects\Mod06\Starter\Pages** folder on the Desktop and select all the files and click OK
- 10) Note down all objects the object id and type that are contained in the file.
- 11) Make sure that it is only the pages listed below

Type	Object Id	Data Type
Page	123456710	CSD Seminar Registration
Page	123456711	CSD Seminar Reg. Subpage
Page	123456713	CSD Seminar Registration List
Page	123456717	CSD Seminar Details FactBox
Page	123456724	CSD Seminar Charges

- 12) Make sure that no imported objects exist in the database
- 13) Close all the files again

## Task 2: Import the .al Files into the workspace

### High Level Steps:

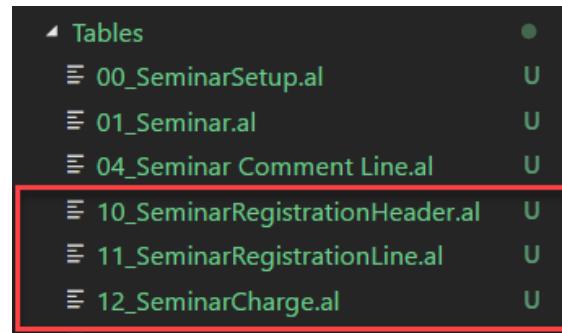
Drag the .al files from **Solution Development Course Objects\Mod06\Starter\Tables** into the Tables folder in the workspace  
Then drag the .al files from

**Solution Development Course Objects\Mod06\Starter\Pages**  
into the Pages folder in the workspace

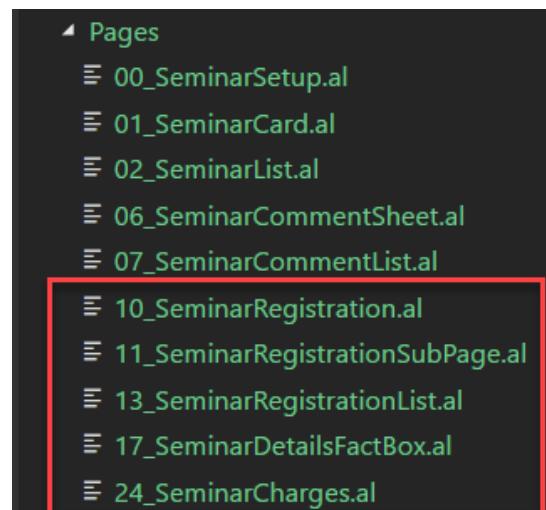
**Detailed Steps:**

**Import all the .al files from on the desktop and place in the workspace**

- 1) Open the **Solution Development Course Objects\Mod06\Starter\Tables** folder on the Desktop in a File Explorer
- 2) Mark all files and drag them into the **Tables** folder of the project
- 3) Verify that they are placed in the Tables folder and that they are all green:



- 4) Then open the **Solution Development Course Objects\Mod06\Starter\Pages** folder on the Desktop in a File Explorer
- 5) Mark all files and drag them into the **Pages** folder of the project
- 6) Verify that they are placed in the Pages folder and that they are all green:



### 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 3: Reviewing Header Table and Field Properties

#### High Level Steps:

- 1) Open the file containing the **Seminar Registration Header** table.
- 2) Add the table **Caption** property
- 3) Check each field and add the **Caption** property for each field
- 4) Correct the **Field Name** and **Length** properties for the **Instructor Code** and **Room Code** fields. The names must include to include the name **Resource No.** and the length must match the **No.** field from the **Resource** table, which is 20.
- 5) Save the file using **Ctrl+S**

#### Detailed Steps:

- 1) Open the **10\_SeminarRegistrationHeader.al** file
- 2) Create a new line just above the fields section and add the following line

```
Caption = 'Seminar Registration Header';
```

```
fields
```

- 3) Locate the first field and add the Caption property for the field:

```
fields
{
    field(1;"No.";Code[20])
    {
        Caption = 'No.';
```

- 4) Repeat this procedure for all fields in the **Seminar Registration Header** table, adding the captions



When the caption is added, the field is included in the multilanguage functionality and the field will be included in the XLIFF file when it is exported

- 5) Locate the **Instructor Code** field and make the following changes
  - a. Change the name to **Instructor Resource No.**
  - b. Change the caption to **Instructor Resource No.**
  - c. Change the length **20** to match the "No." field from the **Resource** table

- 6) Locate the **Room Code** field and make the following changes
- Change the name to **Room Resource No.**
  - Change the caption to **Room Resource No.**
  - Change the length **20** to match the “**No.**” field from the **Resource** table



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

- 7) Locate the **Instructor Name** field  
 8) Verify that the CalcFormula looks like this:

```
CalcFormula = Lookup(Resource.Name WHERE ("No."=
FIELD("Instructor Code"),
Type=CONST(Person)));
Editable = false;
FieldClass = FlowField;
```

- 9) In the CalcFormula property, replace the words “Instructor Code” with “Instructor Resource No.”

```
CalcFormula = Lookup(Resource.Name WHERE ("No."=
FIELD("Instructor Resource No."),
Type=CONST(Person)));
Editable = false;
FieldClass = FlowField;
```

- 10) Now save the file using **Ctrl+S** and locate the page files that were affected by the change: **10\_SeminarRegistration.al** and **13\_SeminarRegistrationList.al**.

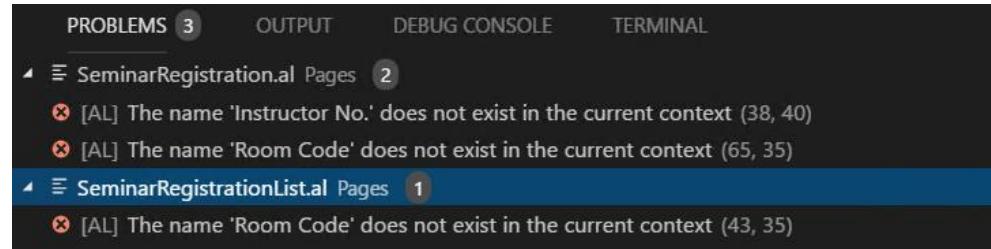
- 11) Verify that they are marked with red in the Explorer window:

≡ 07_SeminarCommentList.al	0
≡ 10_SeminarRegistration.al	2, U
≡ 11_SeminarRegistrationSubPage.al	U
≡ 13_SeminarRegistrationList.al	1, U
≡ 17_SeminarDetailsFactBox.al	U

- 12) Open the **Problems** window

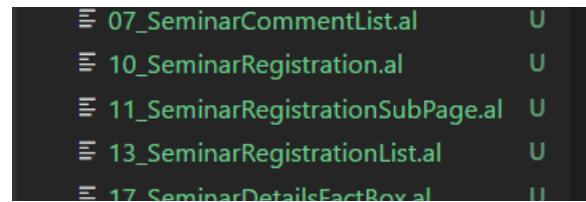
## Solution Development in Visual Studio Code

---



13) Find the Instructor Code field and the Room Code fields and change the field name.

14) Verify that the file names both have changed to green in the Explorer window.



15) Save both files using **Ctrl+S**

## Task 4: Review the Header Code

High Level Steps:

1. Review the **Seminar Registration Header** table Documentation.
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 if the Delete function is called from the **Seminar Registration** page.
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 Dynamics 365 BC 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 the lab you made in the **Master Tables and Pages** chapter.

## Solution Development in Visual Studio Code

---

### Detailed Steps:

- 1) Open the **10\_SeminarRegistrationHeader.al** file
- 2) Review the Seminar Registration Header table documentation.
- 3) Locate the top of the file
- 4) Make sure that the documentation is defined.

```
table 123456710 "CSD Seminar Reg. Header"  
// CSD1.00 - 2018-01-01 - D. E. Velooper  
// Chapter 6 - Lab 1  
// - Created new table
```

- 5) Review the **OnInsert** trigger.
- 6) Make sure that the **OnInsert** trigger contains the code that initializes the number series that is based on the Seminar Setup table.
- 7) The code should look exactly the same as the following example:

```
trigger OnInsert();  
begin  
    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;  
end;
```

- 8) Check whether there is any code that initializes the fields to certain default values
- 9) 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.");

```



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

- 10) Go to the end of the file, just before the **AssistEdit** procedure and add a new line
- 11) Type **tproc** and select the **tprocedure** snippet
- 12) Delete the **var** section
- 13) Name the procedure **InitRecord** 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.
- 14) In the **OnInsert** trigger, select the code described above and press **Ctrl+X**
- 15) In the **OnInsert** code, where the deleted code was present, type **InitRecord**.
- 16) Go to the **InitRecord** procedure and paste the content between the begin and end
- 17) Verify that the code looks like this now:

```

trigger OnInsert();
begin
  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;
end;

```

```
local procedure InitRecord();
begin
    if "Posting Date" = 0D then
        "Posting Date" := WORKDATE;
    "Document Date" := WORKDATE;
    SeminarSetup.GET;
    NoSeriesMgt.SetDefaultSeries("Posting No. Series",
        SeminarSetup."Posted Seminar Reg. Nos.");
end;
```



If the indentation of the code was destroyed during the copy/paste operation, it is possible to right-click the background of the file and select the Format Document function. Then the indentation will be restored.

- 18) In the **OnDelete** trigger, make sure that only the seminars in status **Canceled** can be deleted. This should only apply if the Delete function is called from the **Seminar Registration** page. If it is called from the code, any document can be deleted.
- 19) At the beginning of the **OnDelete** trigger, verify that the value of the Status field is **Canceled**. Use the **TestField** function.
- 20) Insert the following line of code before all other code in the OnDelete trigger:

```
trigger OnDelete();
begin
    if (CurrFieldNo>0) then
        TestField(Status,Status::Canceled);
```



The **CurrFieldNo** variable is a system variable that automatically is set to the field number of the cursor position in the page. E.g. if the cursor is in the **Starting Date** field **CurrFieldNo** will be set to 2. If the delete function is called from the code like: **SemRegHeader.Delete(true)**, then **CurrFieldNo** will be set to 0

- 21) 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.
- 22) 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.

```
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);
```

- 23) Review the code that deletes all **Seminar Charge** records for the current **Seminar Registration Header**

```
SeminarCharge.RESET;  
SeminarCharge.SETRANGE("Document No.", "No.");  
if not SeminarCharge.ISEMPTY then  
  ERROR(Text006, SeminarCharge.TABLECAPTION);
```

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

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

- 25) Review the **OnValidate** trigger for the **No.** field, to understand how it makes sure that changes to the **No.** field are permitted.

Make sure that the following block of code exists

```
trigger OnValidate();
begin
  if "No." <> xRec."No." then begin
    SeminarSetup.GET;
    NoSeriesMgt.TestManual(SeminarSetup.
      "Seminar Registration Nos.");
  end;
end;
```



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.

- 26) 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**

Make sure that the following code exists.

```
trigger OnValidate();
begin
  if "Starting Date" <> xRec."Starting Date" then
    TestField(Status, Status::Planning);
```



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

- 27) 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.

Make sure that the following block of code exists.

```

If "Seminar No." <> xRec."Seminar No." then begin
    SeminarRegLine.RESET;
    SeminarRegLine.SETRANGE("Document No.", "No.");
    SeminarRegLine.SETRANGE(Registered, TRUE);
    if not SeminarRegLine.ISEMPTY then
        ERROR(
            Text002,
            "Seminar No." );

```

- 28) 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.

```



**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

- 29) 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.
- 30) 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

## Solution Development in Visual Studio Code

---

```
if "Room Resource Code" = '' 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 Code");
"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";
```

31) 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 and 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;

```

- 32) Review the **OnValidate** triggers for the **Room Post Code** and **Room City** fields to understand how standard Dynamics 365 BC functionality populates the **Post Code**, **City**, **County**, and **Country** from **Post Code** or **City** values. Make sure that the following code exists.

```

trigger OnValidate();
begin
    PostCode.ValidatePostCode("Room City", "Room Post
    Code", "Room County", "Room Country/Reg. Code",
    (CurrFieldNo <> 0) and GUIALLOWED);
end;

```



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

- 33) 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.

## Solution Development in Visual Studio Code

---

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 repeat
      SeminarRegLine.VALIDATE("Seminar Price",
        "Seminar Price");
      SeminarRegLine.Modify;
    until SeminarRegLine.NEXT = 0;
  Modify;
end;
```

- 34) 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.

Make sure that the following code exists

```

trigger OnValidate();

begin
    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.", '');
end;

```



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



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

- 35) 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.

Make sure that the following block of code exists.

## Solution Development in Visual Studio Code

---

```
trigger OnLookup();

begin
  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;
end;
```

- 36) Review the **AssistEdit** function to make sure that it contains the code that resembles the one that you wrote in **Master Tables and Pages**  
Make sure that the following code exists

```
procedure AssistEdit(OldSeminarRegHeader: Record "CSD
Seminar Reg. Header"): Boolean;
begin
  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;
end;
```

### Task 5: Reviewing Line Table Code and Field Properties

#### Reviewing Line Table Code

##### Exercise Scenario

The Dynamics 365 BC 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 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 Dynamics 365 BC. 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 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:

In the **11\_SeminarRegistrationLine.al** review the **GetSeminarRegHeader** and the **UpdateAmount** procedures. Next review the **OnInsert** and the **OnDelete** triggers. Lastly, review the field triggers of the different fields.

#### GetSeminarRegHeader

The point of the **GetSeminarRegHeader** procedure is to standardize the retrieval of the header variable, and to ensure that the header is only retrieved if necessary.

```
local procedure GetSeminarRegHeader();
begin
    if SeminarRegHeader."No." <> "Document No." then
        SeminarRegHeader.Get("Document No.");
end;
```

#### UpdateAmount

Just like with the **GetSeminarRegHeader** procedure, the point of the **UpdateAmount** procedure is to standardize the calculation of the amounts. The procedure will then be called from many places in the object.

```

local procedure UpdateAmount();
begin
    GLSetup.Get;
    Amount := Round("Seminar Price" - "Line Discount Amount",
                    GLSetup."Amount Rounding Precision");
end;

```



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.

### The OnInsert Trigger

The **OnInsert** trigger first retrieves the **Seminar Registration Header** record by the call to the **GetSeminarRegHeader** function. Next the **Registration Date** is set to the **WorkDate**. 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.

```

trigger OnInsert();
begin
    GetSeminarRegHeader;
    "Registration Date" := WorkDate;
    "Seminar Price" := SeminarRegHeader."Seminar Price";
    Amount := SeminarRegHeader."Seminar Price";
end;

```

### The OnDelete Trigger

The **OnDelete** trigger makes sure that only the lines that are not registered can be deleted. 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.

```
trigger OnDelete();
begin
    TestField(Registered, false);
end;
```

### The Bill-to Customer No. field - OnValidate

The **OnValidate** trigger on the **Bill-to Customer No.** field makes sure that you cannot change the customer for the registered line. 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.

```
trigger OnValidate();
begin
    if "Bill-to Customer No."<>xRec."Bill-to Customer No."
    then begin
        if Registered then begin
            ERROR(RegisteredErrorTxt,
                  FieldCaption("Bill-to Customer No."),
                  FieldCaption(Registered), Registered);
        end;
    end;
end;
```

### The Participant Contact No. field - OnValidate

The **OnValidate** trigger in the **Participant Contact No.** field makes sure that the contact selected by the user is related to the customer that is specified in the **Bill-to Customer No.** field.

It 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.

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.

```

trigger OnValidate();
begin
  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(WrongContactErrorTxt, Contact."No.",
          Contact.Name, "Bill-to Customer No.");
      end;
    end;
  end;
end;

```

### The Participant Contact No. field - OnLookup

The code in the **OnLookup** trigger for the **Participant Contact No.** filters the contacts that are related to the customer, referenced in the **Bill-to Customer No.** field. The code then modally shows the **Contact** page so the user can look up a value.

The **Participant Contact No. – OnLookup** trigger also uses the **Contact Business Relation** table to filter the contacts that belong to the referenced customer.

It is possible to 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**.

```
trigger OnLookup();
begin
    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
            "Participant Contact No." := Contact."No.";
    end;
    CalcFields("Participant Name");
end;
```



The **Run** function calls the page, and then immediately continues executing 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 Seminar Price field - OnValidate

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.

The **Seminar Price – OnValidate** trigger calls the **OnValidate** trigger on the **Line Discount %** to run the calculation code there.

```

trigger OnValidate();
begin
    VALIDATE("Line Discount %");
end;

```



The **AutoFormatType** property on the **Seminar Price** field defines the appearance of the field. The explanation to the **AutoFormatType** property can only be found in the **Developer and IT Pro help**, which can be found here: [https://msdn.microsoft.com/en-us/library/dn789723\(v=nav.90\).aspx](https://msdn.microsoft.com/en-us/library/dn789723(v=nav.90).aspx)

### The Line Discount % field and the Line Discount Amount field – OnValidate

The **Line Discount % - OnValidate** and **Line Discount Amount – OnValidate** triggers calculates one of the discount values that are based on the other value.

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

```

trigger OnValidate();
begin
    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;
end;

```

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

```
trigger OnValidate();

begin
    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;
end;
```

### The Amount field – OnValidate

The **Amount – OnValidate** trigger calculates the **Line Discount Amount** field based on the price and the amount specified.

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**.

Based on the value of the **Line Discount Amount** field, the application calculates the **Line Discount %** field.

```
trigger OnValidate();

begin
    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;
end;
```

### Demonstration: Reviewing the Seminar Charge Table

Many tables in Dynamics 365 BC frequently relate to several 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:

In the **12\_SeminarCharge.al** file review the field triggers that contain code.

#### The Type field – OnValidate

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.

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**.

```
trigger OnValidate();
var
    OldType : Integer;
begin
    if Type <> xRec.Type then begin
        OldType:=Type;
        Init;
    end;
end;
```

#### The No. field – OnValidate

The **No. – OnValidate** trigger shows how the values for several fields are defaulted from other master records.

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. When you use master records, you first must make sure that the record is not blocked.

The information defaults from either the **G/L Account** or **Resource** tables, depending on the value in the **Type** field.

```
trigger OnValidate();

begin
  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;
end;
```

### The Quantity, Unit Price, and Total Price fields - OnValidate

The **Quantity**, **Unit Price**, and **Total Price** fields are interrelated.

- When the user changes either the **Quantity** or the **Unit Price**, the system calculates the **Total Price**.
- When the user changes the **Total Price**, the system calculates the **Unit Price**.
- 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**.

```
field(6;Quantity;Decimal)
{
    DecimalPlaces = 0:5;

    trigger OnValidate();
    begin
        "Total Price" := Round("Unit Price" * Quantity,0.01);
    end;
}

field(7;"Unit Price";Decimal)
{
    AutoFormatType = 2;
    MinValue = 0;

    trigger OnValidate();
    begin
        "Total Price" := Round("Unit Price" * Quantity,0.01);
    end;
}
```

```

field(8;"Total Price";Decimal)
{
    AutoFormatType = 1;
    Editable = false;

trigger OnValidate();
begin
    if (Quantity<>0) then
        "Unit Price":= Round("Total Price" / Quantity,0.01)
    else
        "Unit Price":=0;
end;

```

### The Unit of Measure Code field – OnValidate

The **Unit of Measure Code** influence the price calculations.

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.

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.

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.

```
trigger OnValidate();

begin
  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;
        ResourceUofM.Get("No.", "Unit of Measure Code");
        "Qty. per Unit of Measure" := ResourceUofM."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;
  if CurrFieldNo=FieldNo("Unit of Measure Code") then
    begin
      Validate("Unit Price");
    end;
end;
```

## Lab 6-2: Reviewing the Seminar Comment Line Table and Pages

In Dynamics 365 BC, all master records and documents enable users to define free-text comments.

Dynamics 365 BC 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.

### Task 1: Reviewing the Seminar Comment Line Table and Pages

#### High Level Steps:

Review and complete the **Seminar Comment Line** Table

Review the primary key of the **Seminar Comment Line** Table and explain the function of the last field in the key.

Add conditional table relation for the **Seminar Registration Header** table

Add a new function that is named **Setup.NewLine** setting the date to be the working date if there are no comments available.

Then open the file for the **Seminar Comment Sheet** page and verify that the **AutoSplitKey** property has been set.

Make sure that the **Setup.NewLine** procedure is called from the **OnNewRecord** trigger

Add field captions to the table and to all fields

#### Detailed Steps:

1) Open the **Seminar Comment Line** file

2) Set the **Caption** property to the table

3) Go through all fields and add the **Caption** property from the name

4) Locate the **No.** field and extend the **TableRelation** property with the following line:

```
else if ("Table Name"=const("Seminar Registration Header"))
"Seminar Registration Header"
```

Making the total line look like this:

```
field(30;"No.";Code[20])
{
    Caption='No.';
    TableRelation=if ("Table Name"=CONST(Seminar)) "Seminar"
        else if ("Table Name"=const("Seminar
        Registration Header")) "Seminar Registration
        Header";
}
```

## Solution Development in Visual Studio Code

---

- 5) Locate the **keys** section of the file
- 6) Verify that the Primary Key (PK) has been set to the following:

```
key(PK;"Table Name","Document Line No.","No.","Line No.")  
{  
    Clustered = true;
```

- 7) Locate the end of the file and create a new line just before the last }
- 8) Type **tpro** and select **tprocedure** to create a new procedure
- 9) Remove the local property to make the procedure global
- 10) Name the procedure **Setup.NewLine**

```
procedure Setup.NewLine()  
var  
    SeminarCommentLine: Record "CSD Seminar Comment Line";  
  
begin  
    SeminarCommentLine.SetRange("Table Name", "Table Name");  
    SeminarCommentLine.SetRange("No.", "No.");  
    SeminarCommentLine.SetRange("Document Line No.",  
        "Document Line No.");  
    SeminarCommentLine.SetRange("Date", WorkDate);  
    if SeminarCommentLine.IsEmpty then  
        Date:=WorkDate;  
end;
```

- 11) Save the file using **Ctrl+S**
- 12) Open the file containing the **Seminar Comment Sheet** page
- 13) Add a new property for the page:  
**AutoSplitKey=true;**
- 14) Go to the end of the file and create a new line just before the last }
- 15) Type **ttr** and select **ttrigger** to create a new procedure
- 16) Replace the **OnWhat** with **OnNewRecord(BelowxRec: Boolean)**
- 17) Remove the **var** section of the trigger
- 18) Call the Setup.NewLine function between the begin and the end

- 19) Save the file using **Ctrl+S**  
20) Verify that the file looks like this:

```
page 123456706 "CSD Seminar Comment Sheet"
{
    Caption = 'Seminar Comment Sheet';
    PageType = List;
    SourceTable = "Seminar Comment Line";
    UsageCategory= Lists;
    AutoSplitKey=true;

    layout
    {
        area(content)
        {
            repeater(Group)
            {
                field(Date;Date)
                {
                }
                field(Code;Code)
                {
                    Visible=false;
                }
                field(Comment;Comment)
                {
                }
            }
        }
    }
}
```

```
trigger OnNewRecord(BelowxRec: Boolean)
begin
    Setup.NewLine;
end;
}
```



If the Intellisense does not suggest the **Setup.NewLine** procedure, it is because the procedure has not been made global

## Lab 6.3: Create Seminar Registration Pages

The developer for the partner company that is implementing Dynamics 365 BC for CRONUS International Ltd is now ready to complete the work and develop the pages for managing seminar registrations.

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 must be created. Also, it is necessary to develop a FactBox that shows details about a seminar. Use this FactBox to decorate the document and list page.

Finally, any necessary code must be added, and the pages must be linked to enable users to move from the list to the document, and to access the related information, such as comments and charges.

### Exercise 1: Review the Pages

#### Exercise Scenario

Several pages have already been created, but some of them has not been finished. The objects previously imported included:

- Seminar Registration page. Incomplete and controls must be added to this page.
- Seminar Registration List page.
- Seminar Reg. Subpage 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 any issues found is corrected.

### Task 1: Review the Seminar Registration Pages

#### High level steps:

- Review the Seminar Details FactBox page.
- Review the Seminar Reg. Subpage page.
- Review the Seminar Registration page.

#### Detailed steps:

##### **Review the Seminar Details FactBox page**

- 1) Open the file containing the **Seminar Details FactBox** page
- 2) Verify that the **PageType** property is set to **CardPart**. **FactBoxes** must be of **CardPart** or **ListPart** type.
- 3) Verify that the **SourceTable** property is set to **Seminar**. This FactBox shows

information about one seminar.

- 4) Verify that all fields are added directly under the **Content** area. There can not be a group control because **CardPart** pages do not use groups.
- 5) Delete the **actions** section because FactBoxes do usually not use actions.
- 6) Save the file using **Ctrl+S**

### Review the Seminar Reg. SubPage page

- 1) Open the file containing the **Seminar Reg. Subpage** page
- 2) Set the **PageType** property to **ListPart**. Document subpages must be of the **ListPart** type.



Developers frequently make the mistake and set the **PageType** property for subpages to **List**. Selecting an incorrect page type causes the **SetAutoSplitKey** functionality to fail

- 3) Verify that the **SourceTable** property is set to **Seminar Registration Line**.
- 4) 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**.
- 5) Set the **AutoSplitKey** property to **true**. All document subpages must set this property to enable Dynamics 365 BC to automatically assign the values in the **Line No.** field.
- 6) Verify that there is a group control of type **Repeater** under the **Content** area. The repeater contains the fields shown by the subpage.
- 7) Save the file using **Ctrl+S**

### Review the Seminar Registration page

- 1) Open the file containing the **Seminar Registration** page
- 2) Set the **PageType** property to **Document**. Document pages must not be of the Card type.



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

- 3) Verify that the **SourceTable** property is set to **Seminar Registration Header**
- 4) 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 yet.



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

- 5) Select the **Seminar Details FactBox** page part control.
- 6) Check the SubPageLink property.
- 7) Verify that the value looks like this:

```
area(factboxes)
{
    part("Seminar Details FactBox";"Seminar Details FactBox")
    {
        SubPageLink="No."=field("Seminar No.");
    }
}
```



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

- 8) Save the file using **Ctrl+S**

### Exercise 2: Completing the Document Pages

#### Exercise Scenario

Now that the review has been completed, the development of the seminar registration management pages can be completed too.

### Task 2: Complete the Document Pages

#### High Level Steps:

- Design the Seminar Registration page
- Add the subpage for the **Seminar Reg. Subpage** page, and name the control **SeminarRegistrationLines**
- Link the **SeminarRegistrationLines** page part to the main page
- Link the **Customer Details FactBox** page part to the **SeminarRegistrationLines** page part

#### Detailed Steps:

- 1) Open the file containing the **Seminar Registration** page
- 2) Locate the **Seminar Room** group and create a new line above the group with the same indentation as the group
- 3) Add the **Caption** and **SubPageLink** properties as shown below:

```
part(SeminarRegistrationLines;"Seminar Registration
Subpage")
{
    Caption='Lines';
    SubPageLink="Document No."= field("No.");
}
```

- 4) Locate the **FactBox** area
- 5) Create a new row underneath the **Seminar Details FactBox** page
- 6) Add a new part with the same indentation as the **Seminar Details FactBox** page
- 7) On the next line add a **{**, and VS Code will automatically add the matching **}**. After pressing Enter, the code will format automatically
- 8) Enter **("Seminar Details FactBox";"Seminar Details FactBox")** for the part.
- 9) Add the **Provider** property to point to the **SeminarRegistrationLines** control
- 10) Add the SubPageLink to link the Customer **No.** field on the **Customer Details FactBox** page to the **Bill-to Customer No.** field on the **Seminar Reg. Subpage** page
- 11) Verify that the code looks like this:

```

area(factboxes)
{
    part("Seminar Details FactBox";"Seminar Details FactBox")
    {
        SubPageLink="No."=field("Seminar No.");
    }

    part("Customer Details FactBox";"Customer Details
        FactBox")
    {
        Provider=SeminarRegistrationLines;
        SubPageLink="No."=field("Bill-to Customer No.");
    }
}

```



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.

- 12) Save the file using **Ctrl+S**
- 13) Next, make sure that all files are saved by clicking the **File/Save All** menu item
- 14) Then click the Source Control icon  The number might be different depending on the last commit.
- 15) Enter the text **Lab 6.3** in the Message field
- 16) Then click the Commit checkmark 
- 17) Lastly, Push the changes to the off-site repository

### 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 Dynamics 365 BC 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 Dynamics 365 BC.

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.

## Module 7: Posting

### Module Overview

Transactional systems, such as Dynamics 365 BC, records 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, Dynamics 365 BC 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 Dynamics 365 BC 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 Dynamics 365 BC consist of the following:

- Document tables
- Journal tables

Posted tables in Dynamics 365 BC 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 routines 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 Dynamics 365 BC.

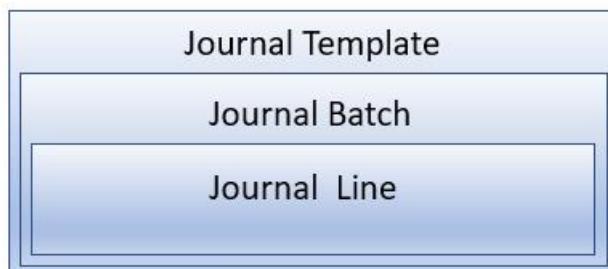
### Journal Tables

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
<b>Journal Template</b>	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.

<b>Journal Batch</b>	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.
<b>Journal Line</b>	Journal line tables store the information about the transaction itself.

Lines belong to batches, and batches belong to templates. The “Journal Structure” is shown below.



## 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 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 Dynamics 365 BC, 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. The main rule is that the **Entry No.** of a Ledger Entry table must start at 1 and it must be consecutive and unbroken.



Exceptions to this rule are the **Customer Ledger Entry** and **Vendor Ledger Entry** tables. They will be given the same number as the **General Ledger Entry**

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

The 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.



You do not protect the **Ledger Entry** tables directly by making the table fields noneditable. Instead, you must protect each field from 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:

- checks journal lines
- 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 Dynamics 365 BC, 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 line 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.



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.

### Check Line

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.

The posting process in the **Post Line** codeunit performs many write operations. It also adds many locks, some of them explicit. The Check Line guarantees the highest possible concurrency between transactions. Calling the **Check Line** codeunit first in the posting process, causes the problematic journal to fail before any locks are added.

This codeunit is called by both the **Post Batch** codeunit and the **Post Line** codeunit.

### Post Batch

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.

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 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 <b>JournalBatches</b> page.	3	233, Gen. Jnl.-B.Post
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.



This function utilizes that a global variable is initialized with the first call of the object and is kept in memory as long as the calling object is in scope.

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.



The Post Batch codeunit always respects any filters that the user has applied to the **Journal Line** table in the **Journal** page. This allows users to only post sections of a batch, instead of the whole batch.

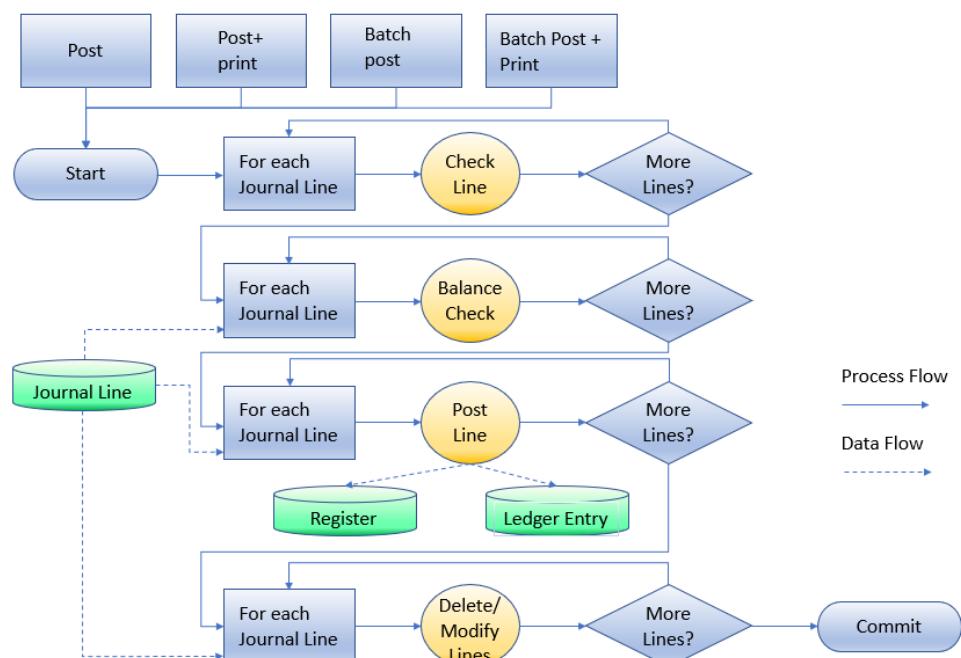
The **Post Batch** codeunit can then begin checking each journal line in the batch by calling the **Check Line** codeunit for each line. As soon as all lines are checked, they can be posted by calling the **Post Line** codeunit for each line. By then, the codeunit has looped through all the records two 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.

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 diagram below outlines the steps in the Posting Routine when the Post Batch codeunit is called.



### 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 additional checks, like 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 Dynamics 365 BC, 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 journal**, the **Resource journals**, and the **General Journal** for the customer.

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.

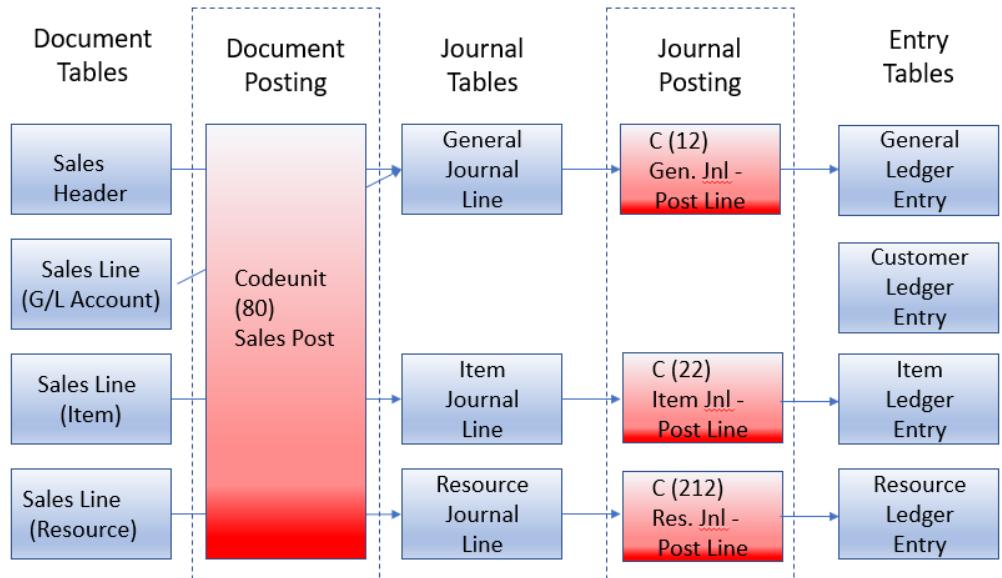
In this example, the document posting routine calls the **Gen. Jnl.-Post Line** codeunit at least two times: one time for the **Item Jnl.-Post Line** codeunit, and one time for the **Res. Jnl.-Post Line** codeunit.

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 diagram below 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.



## 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:

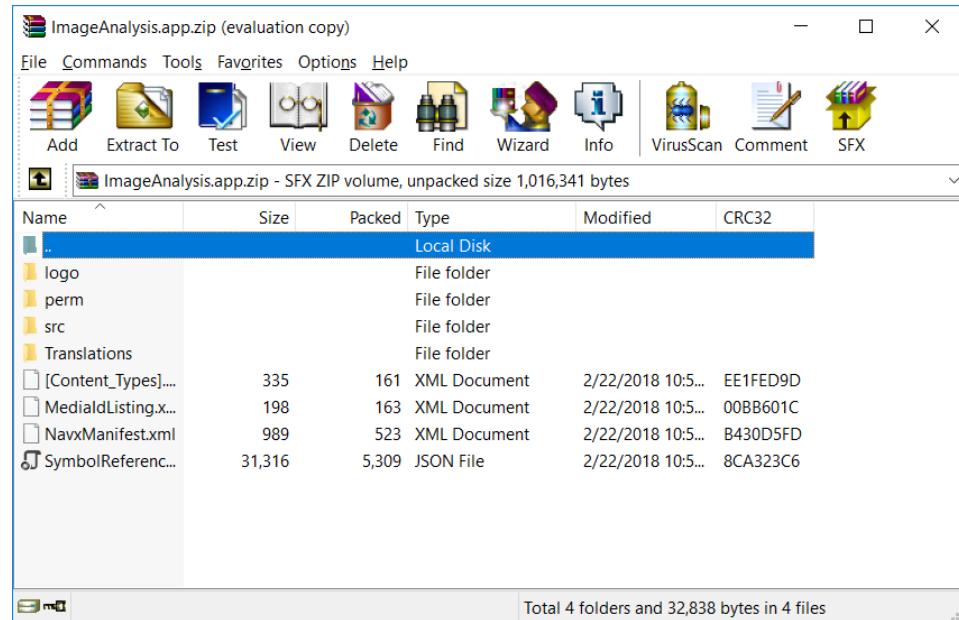
- Determines the document type and validates the information on the sales header and lines.
  - The codeunit determines the posted document numbers and updates the header.
  - This section ends with a **Commit** function call.
- Locks the appropriate tables.
- Inserts the **Shipment Header** and **Invoice- 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 depending on the

### Sales & Receivable Setup.

- 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**.
  - 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.

### Securing the source code

The source code for Dynamics 365 BC has always been available to all certified Dynamics NAV partners for the standard application and the customizable object range. Many Independent Software Vendors (ISVs) allow the Microsoft Partners to view and even alter their source code. This procedure changes a bit now. Not that the source code is available to any Dynamics NAV partner, because every app file is actually a **.zip** file containing all the **.al** files:



Changing the extension to be **.zip** and opening it with WinRAR will show you all the files in the extension.

However, it is not possible to make changes to the **.app** file and republish it to the server.

You are therefore dependent of the ISV company's ability to stay alive to provide future upgrades.

## Solution Development in Visual Studio Code

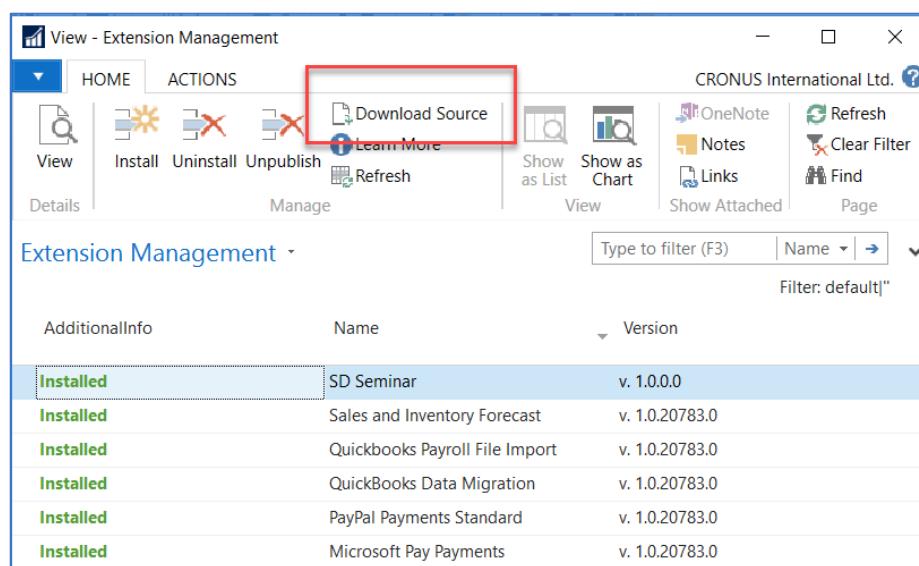
Developing a solution whether it is sold through app source or directly as an add-on solution, the source code should be secured by opening Escrow account.

1Click Factory describes the Escrow service like this:

*"Source code escrow is the deposit of the source code of software with a third-party escrow agent. It ensures maintenance of the software if the solution licensor files for bankruptcy or otherwise fails to maintain and update the software as promised in the software license agreement. As Microsoft states, this requirement is aimed to protect the investment of the customer now and in the future."*

*The Software Escrow requirement is mandatory for CfMD approval if the solution is not offered free of charge and not sold by subscription."*

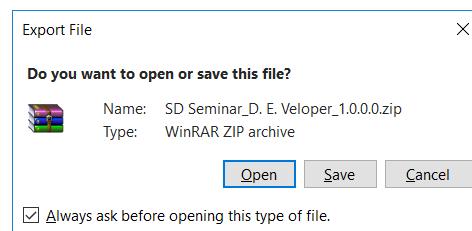
Even if the solution is not to be certified, it is essential to create a structure for the files used for development. For the owner of the extension, or for extensions created in the 50.000 – 99.999 object range, it is possible to download the source code directly from the **Extension Management** page at the customer site:



The screenshot shows the 'View - Extension Management' page. At the top, there are tabs for 'HOME' and 'ACTIONS'. Under 'ACTIONS', there are several icons: 'View', 'Install', 'Uninstall', 'Unpublish', 'Download Source' (which is highlighted with a red box), 'Learn More', 'Refresh', 'OneNote', 'Notes', 'Links', 'Show Attached', and 'Find'. Below this is a search bar with 'Type to filter (F3)' and 'Name' dropdown. The main area is titled 'Extension Management' and shows a list of installed extensions. The columns are 'AdditionalInfo', 'Name', and 'Version'. The list includes:

AdditionalInfo	Name	Version
Installed	SD Seminar	v. 1.0.0.0
Installed	Sales and Inventory Forecast	v. 1.0.20783.0
Installed	Quickbooks Payroll File Import	v. 1.0.20783.0
Installed	QuickBooks Data Migration	v. 1.0.20783.0
Installed	PayPal Payments Standard	v. 1.0.20783.0
Installed	Microsoft Pay Payments	v. 1.0.20783.0

The download is a zip file:



Then it is possible to continue directly from there.

However, it is recommendable to maintain a version structure and always make sure that the customers run the latest version. That way it is possible to control changes to the table structures via either Install codeunits or upgrade codeunits.



An **Install Codeunit** is a codeunit that is run every time the extension is installed to the server.

An **Upgrade codeunit** is run if the version number in the app.json file is changed. More about both later.

## Code Comments

Together with the general comments that are provided generally in the .al file, it is important to provide comments in the code at the lines where a change for a new version is made. Do this only when changing an existing object, not when you create a new .al file.

The key is to mark the changed code with the same reference number as used in the documentation of the object.

```
page 123456702 "Seminar List"
// CSD1.00 - 2018-01-01 - D. E. Veloper
// Chapter 5 - Lab 3-6
// Version 1.0.1.123 Added field
{
    Caption='Seminar List';
    PageType = List;
    SourceTable = Seminar;
    Editable = false;
    CardPageId = 123456701;
    UsageCategory = Lists;

    layout
    {
        0 references
        area(content)
        {
            0 references
            repeater(Group)
            {
                // >> 1.0.1.123
                0 references
                field(Comment;Comment)
                {
                }
                // << 1.0.1.123
                field("No.;" "No.")
            }
        }
    }
}
```

### Single Line Modification

```
trigger OnValidate();

begin
    if "Seminar No." <> xRec."Seminar No." then begin
        SeminarRegLine.Reset;
        // >> 1.0.1.123
        //SeminarRegLine.SetRange("Document No.", "No.");
        SeminarRegLine.SetRange("Document No.", "No. Series");
        // << 1.0.1.123
        SeminarRegLine.SetRange(Registered, true);
```

If you add or change a whole block of code, mark the change as follows.

### Multiple Lines Modification

```
trigger OnValidate();

begin
    if "Seminar No." <> xRec."Seminar No." then begin
        SeminarRegLine.Reset;
        // >> 1.0.1.123
        /* 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);
        */
        SeminarRegLine.GET("No.",10000);
        // << 1.0.1.123
```

### Code removal

Never delete the original extension 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.

## 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 Dynamics 365 BC 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 Dynamics 365 BC, 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.



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 demands to the terminology of Dynamics 365 BC, 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. 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.

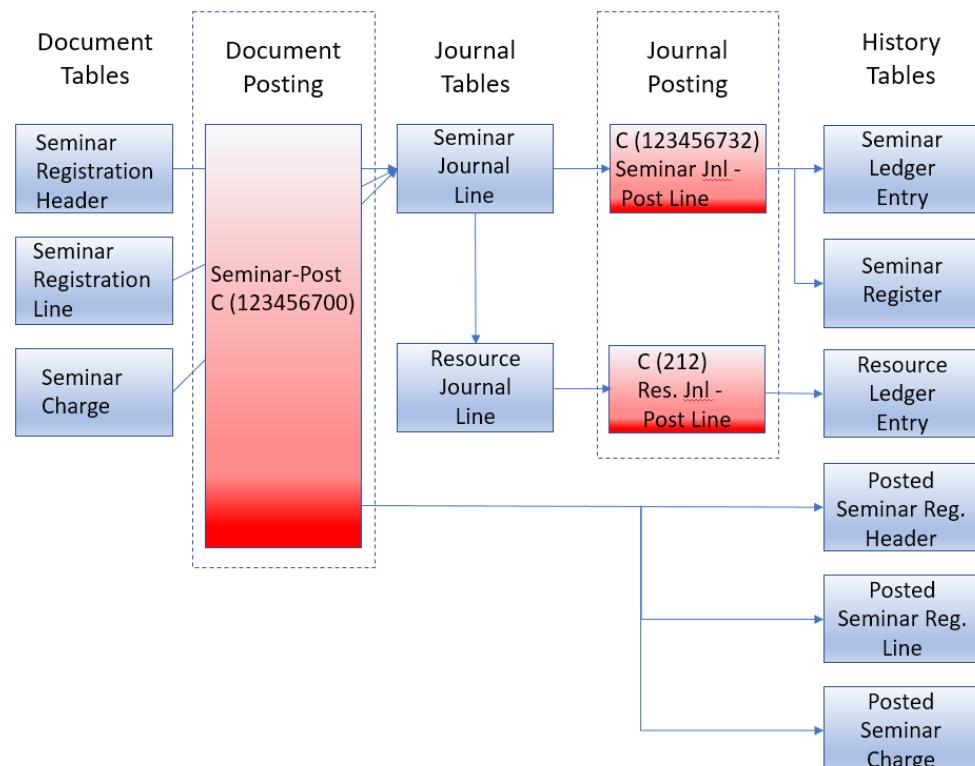
When you introduce posting functionality, you must follow Dynamics 365 BC standard conceptual, data model, and user interface principles. This means that you must provide at least the journal posting infrastructure that consists of the following:

- 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 figure below shows the entities that are involved in the seminar registration posting process and the data flow between them.



## 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 Dynamics 365 BC 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 CSD Posted Seminar Reg. Header	Holds the information for the completed (posted) seminar. Takes the data from the <b>Seminar Registration Header</b> table during posting.
123456719 CSD Posted Seminar Reg. Line	Holds the detailed information for the completed (posted) seminar. Takes the data from the <b>SeminarRegistrationLine</b> table during posting.
123456721 CSD Posted Seminar Charge	Holds charges that are related to the completed (posted) seminar.
123456731 CSD Seminar Journal Line	Let you post the seminar ledger entries.
123456732 CSD Seminar Ledger Entry	Keeps the transaction details for all posted seminars.
123456733 CSD Seminar Register	Keeps the transaction log for posted seminars.

You must also change the following tables.

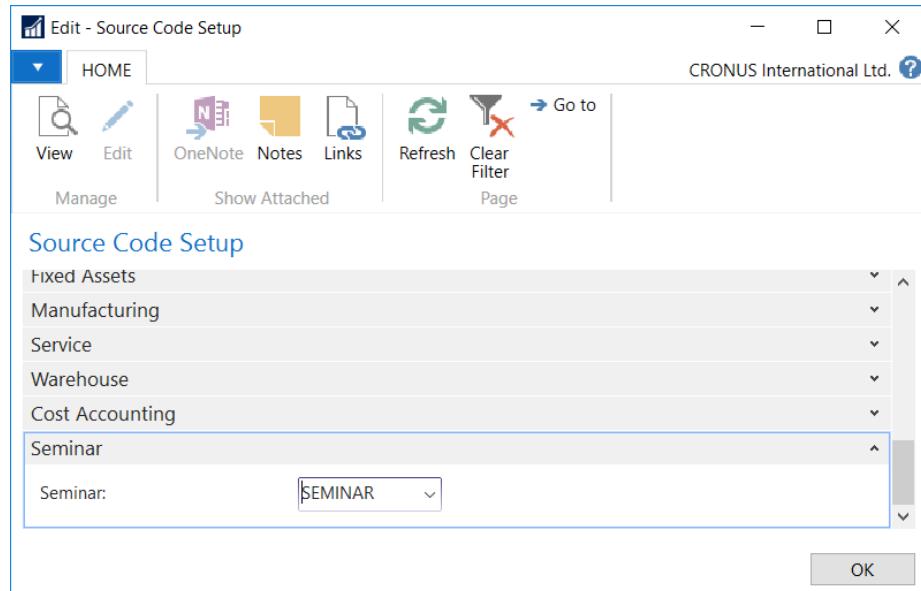
Table	Remarks
203 Res. Ledger Entry	Add fields to link the resource ledger entries to the seminars and to the posted seminar registration documents.
207 Res. Journal Line	Add fields to support posting of seminar information during resource journal posting.
242 Source Code Setup	Add a field to support the audit trail source code for the seminar registration transaction history.

### Pages

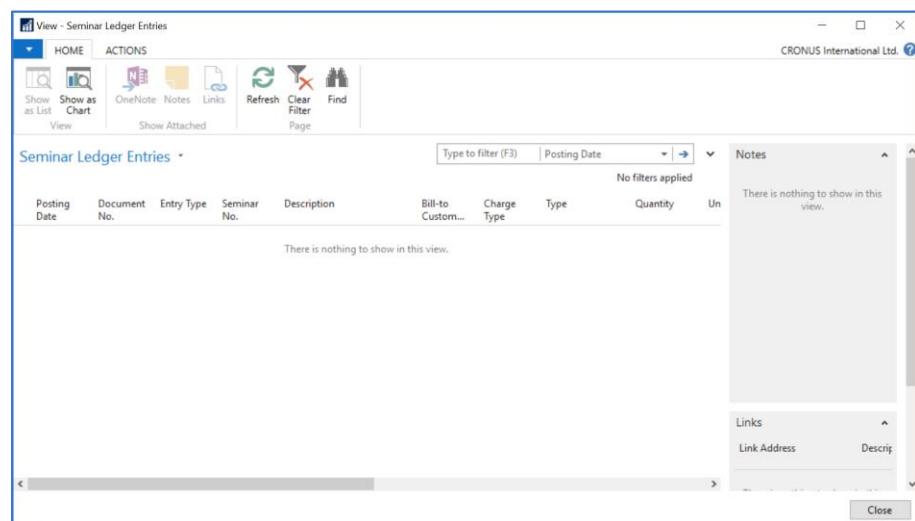
The pages for the seminar registration posting and the navigation between them

reflect the relationships that are shown in the previous 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:

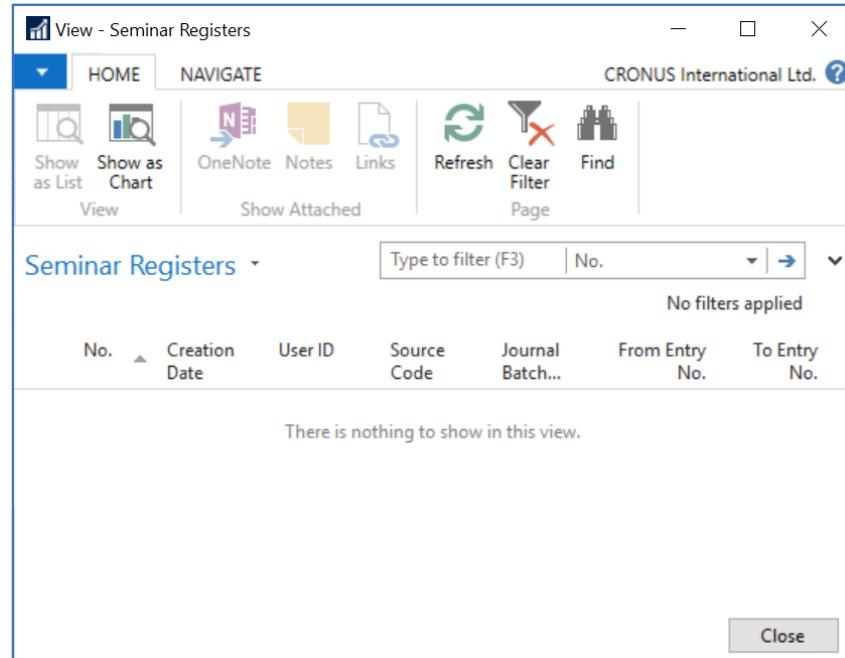


The **Seminar Ledger Entries** page displays the ledger entries:

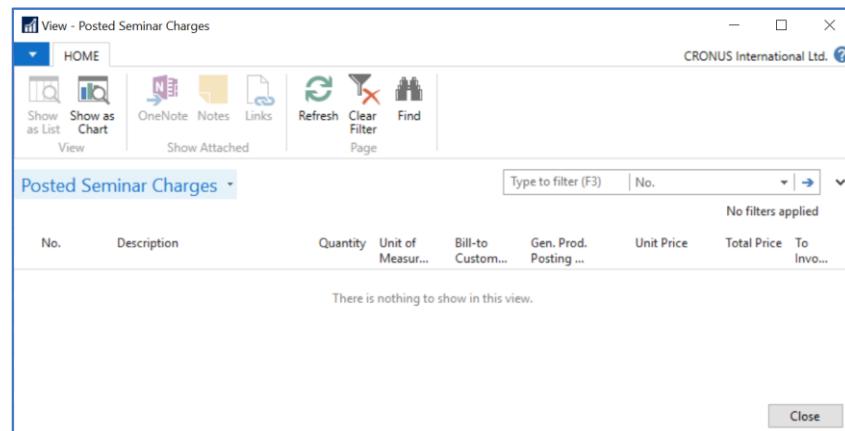


## Solution Development in Visual Studio Code

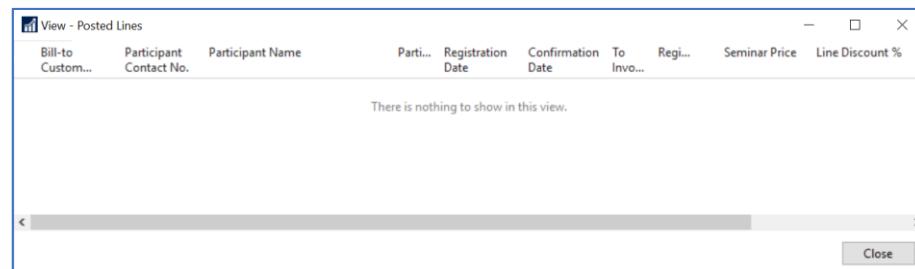
The **Seminar Registers Page** displays the registers that are created when seminar registrations are posted.



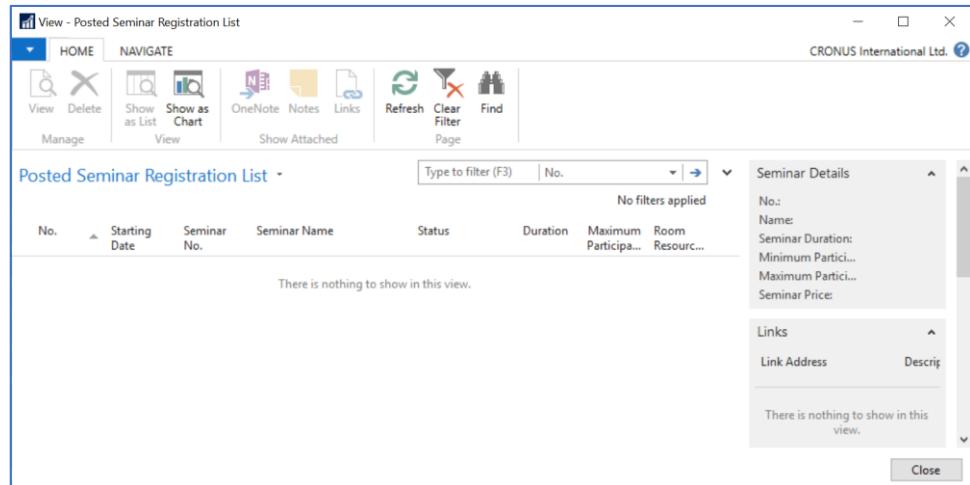
The **Posted Seminar Charges Page** image shows the charges that are related to a posted seminar registration.



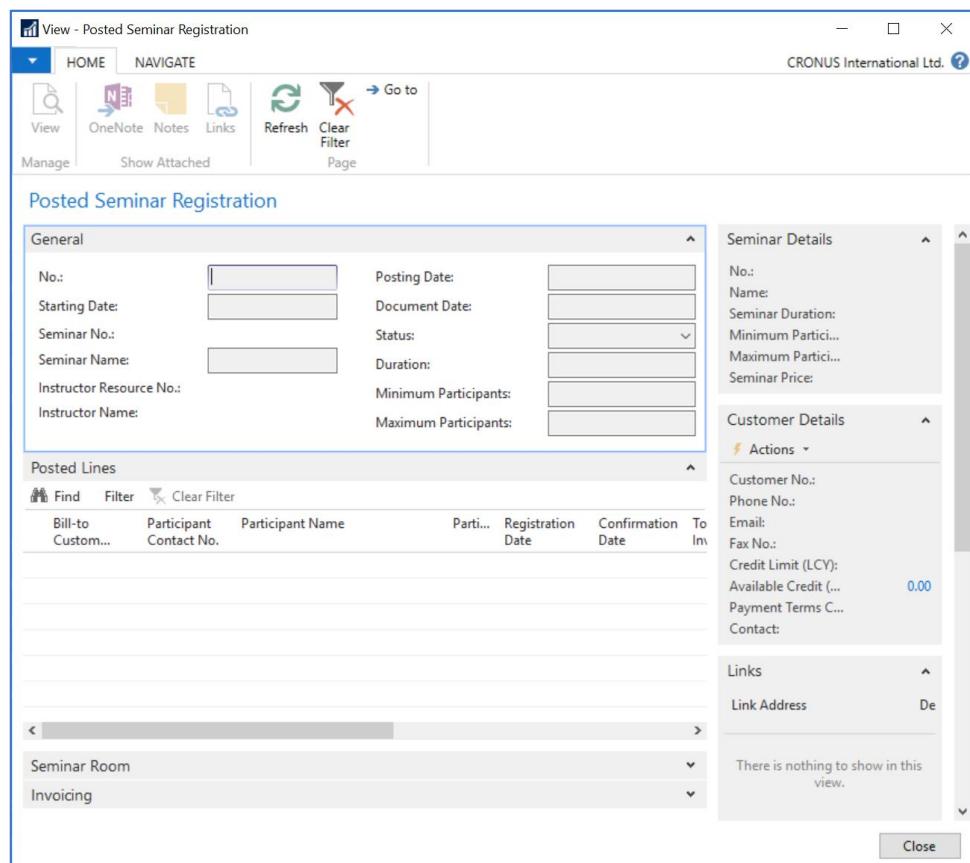
The **Posted Seminar Reg. Subform page** shows the lines for the Posted Seminar Registration document page.



The **Posted Seminar Reg. List Page** displays a list of posted seminar registrations.



The **Posted Seminar Registration Page** displays the posted seminar registration.



### 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 CSD Seminar Jnl.-Check Line	<ul style="list-style-type: none"><li>Verifies the data validity of a seminar journal line before the posting routine posts it by doing the following:</li><li>The codeunit checks that the journal line is not empty and that there are values for the <b>Posting Date</b>, <b>Instructor Resource No.</b>, and <b>Seminar No.</b> fields.</li><li>Depending on whether the line is posting an Instructor, a Room, or a Participant, the codeunit checks that the applicable fields are not blank.</li></ul> <p>The codeunit also verifies that the dates are valid.</p>
123456732 CSD Seminar Jnl.-Post Line	<ul style="list-style-type: none"><li>Performs the posting of the <b>Seminar Journal Line</b>. The codeunit creates a <b>Seminar Ledger Entry</b> for each <b>Seminar Journal Line</b> and creates a <b>Seminar Register</b> to track which entries are created during the posting</li></ul>

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 CSD Seminar-Post	<p>Posts the complete seminar registration that includes the resource posting and seminar posting.</p> <ul style="list-style-type: none"> <li>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.</li> <li>The codeunit creates a new <b>Posted Seminar Reg. Header</b> record and <b>Posted Seminar Reg. Line</b> records.</li> <li>The codeunit then runs the job journal posting, and posts seminar ledger entries for each participant, for the instructor, and for the room.</li> </ul> <p>Finally, the codeunit deletes the records from the document tables. This includes the header, lines, comment lines, and charges.</p>
123456701 CSD Seminar-Post (Yes/No)	<p>Interacts with users and confirms that they want to post the registration. If users confirm the posting, the codeunit runs the <b>Seminar-Post</b> codeunit.</p>

# Lab 7.1: Reviewing and Completing the Journal and Ledger Tables

## Scenario

The team that is implementing Dynamics 365 BC for CRONUS International Ltd. Isaac is in charge of developing the base version of tables and user interface objects.

It is now necessary to review the objects to make sure that they follow all Dynamics 365 BC architectural principles and best practices. Any necessary corrections must be made to the tables, their properties and fields, and their code.

By this point, you should already be familiar with the basics of the Dynamics 365 BC 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.

## Reviewing the Import File Contents and Importing the Objects

### Exercise Scenario

Prior to importing objects, you should review the contents of the import files to make sure that no existing objects will be overwritten. Start the review process by opening the .al files. This enables you to review the contents of the file before adding the files to the workspace.



You may notice that the import files do not include the **Seminar Journal Template** or **Seminar Journal Batch** table, or the **Seminar Journal** page. This is because the **Seminar Journal Line** 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.

## Task 1: Preview the .al files Contents

### High Level Steps:

Review the .al files from the folders:

**Solution Development Course Objects\Mod07\Starter A\Tables**

without importing them to the workspace.

Check that the files contain the following objects:

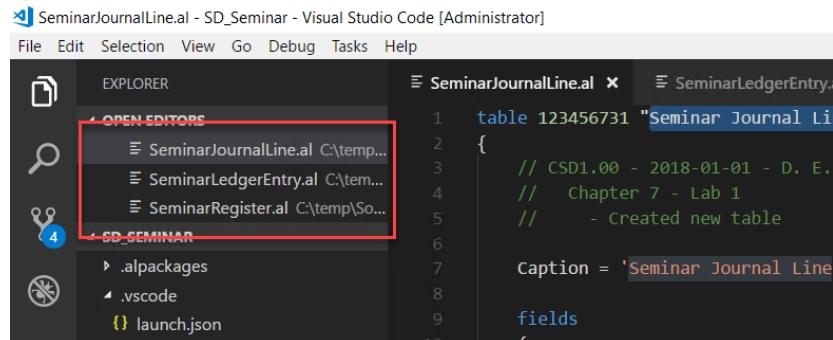
Type	Object Id	Data Type
Table	123456731	CSD Seminar Journal Line
Table	123456732	CSD Seminar Ledger Entry
Table	123456733	CSD Seminar Register

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

### Detailed Steps:

#### Investigate the .al files from on the desktop

- 1) In the VS Code Explorer window, make sure that all files are saved and there are no open files in the editor
- 2) Click the **File/Open File** menu item
- 3) Open the **Solution Development Course Objects\Mod07\Starter A\Tables** folder on the Desktop and select all the files and click OK



- 4) This will open the files in VS Code and give you a chance of investigating the files without changing the project.
- 5) Note down all objects the object id and type that are contained in the file.

Type	Object Id	Data Type
Table	123456731	CSD Seminar Journal Line
Table	123456732	CSD Seminar Ledger Entry
Table	123456733	CSD Seminar Register

- 6) Make sure that no imported objects exist in the database
- 7) Close all the files again

## Task 2: Import the .al Files into the workspace

### High Level Steps:

Drag the .al files from

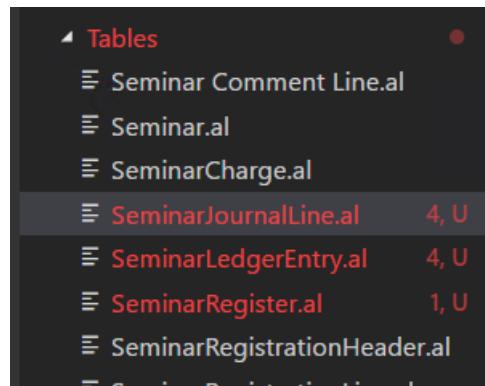
## Solution Development in Visual Studio Code

**Solution Development Course Objects\Mod07\Starter A\Tables**  
into the Tables folder in the workspace  
Review the files and fix the errors until the files are green

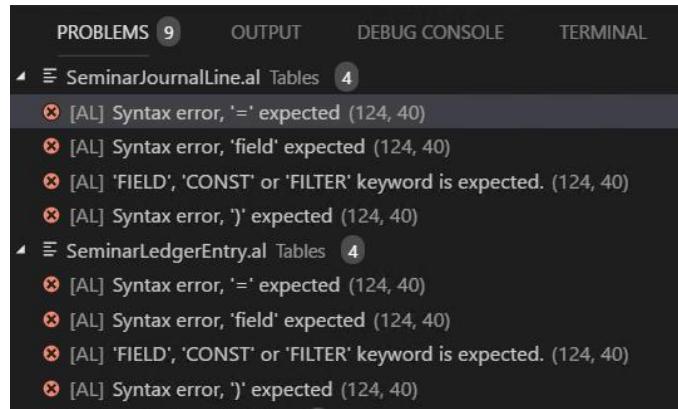
### Detailed Steps:

**Import all the .al files from on the desktop and place in the workspace**

- 1) Open the **Solution Development Course Objects\Mod07\Starter A\Tables** folder on the Desktop in a File Explorer
- 2) Mark all files and drag them into the **Tables** folder of the project
- 3) Verify that they are placed in the Tables folder and that they are all red:



- 4) Click the **Problems** window and click the first line:



- 5) Fix the following type of conversion errors:

- TableRelation = IF (**Source Type**=CONST(Seminar)) Seminar;

Must be changed into:

TableRelation = IF ("**Source Type**"=CONST(Seminar)) Seminar;



Markin the both words **Source Type** and typing "", will automatically set the quotes before and after

Any object id must be changes from the ID to the Name:

- UserMgt : Codeunit "418";

Must be changed into:

UserMgt : Codeunit "User Management";

- 6) Verify that all files are green now:

Tables	
≡ Seminar Comment Line.al	●
≡ Seminar.al	
≡ SeminarCharge.al	
≡ SeminarJournalLine.al	U
≡ SeminarLedgerEntry.al	U
≡ SeminarRegister.al	U

- 7) Remove all unused sections, e.g. **fieldsgroups**

## 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 Dynamics 365 BC principles. Start with the Seminar Journal Line table.

### Task 3: Review Table and Field Properties

#### High Level Steps:

Review the fields of the **Seminar Journal Line**

Verify that the primary key of the **Seminar Journal Line** consists of the following fields: **Journal Template Name**, **Journal Batch Name** and **Line No.** fields in the correct order.

#### Detailed Steps:

- 1) Select the **31\_SeminarJournalLine.al** file
- 2) Verify that the following fields are present in the **Seminar Journal Line** 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 <b>Posting Date</b> . 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 Dynamics 365 BC 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.

- 3) Verify that the primary key of the **Seminar Journal Line** consists of the following fields: **Journal Template Name**, **Journal Batch Name** and **Line No.** fields in the correct order by locating the **var** section at the end of the file.

### Task 4: Review the Table Code

#### High Level Steps:

- Check whether the table contains the necessary functions for the journal line tables.
- Create and define the **EmptyLine** function. The function must return a Boolean value defining if the line is empty.
- 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:**

All Journal Line tables must contain the **EmptyLine** function, which is called during posting to make sure no empty lines are posted.

- 1) Select the **31\_SeminarJournalLine.al** file
- 2) Locate the end of the file
- 3) Add an empty line just above the last and bracket ( **}** )
- 4) Create a new procedure by typing the snippet **tpro** and selecting **procedure**
- 5) Remove the **local** property to enable the procedure to be called from outside the object
- 6) Change the name to be **EmptyLine**
- 7) Add the **: Boolean** after the parentheses to enable the return value
- 8) Remove the **var** section
- 9) Enter the following code between the begin and the end

```
procedure EmptyLine() : Boolean;
begin
    exit(("Seminar No." = '') AND (Quantity = 0));
end;
```



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.

- 10) Locate the Posting Date field
- 11) Create a blank line after the Caption property
- 12) Add a new trigger by typing **ttr** and selecting **trigger**
- 13) Replace the **OnWhat(Parameters)** with **OnValidate()**
- 14) Remove the **var** section of the trigger
- 15) Add the following code in the trigger:

```
Caption = 'Posting Date';

trigger OnValidate();
begin
    Validate("Document Date", "Posting Date");
end;
```

## Reviewing Other Tables

### Exercise Scenario

After reviewing the **Seminar Journal Line** table and correcting the issues that caused the gap between development teams work and Dynamics 365 BC standards and best practices, you want to review the remaining journal posting tables: the **Seminar Ledger Entry** and **Seminar Register** tables.

### Task 5: Review the Seminar Ledger Entry Table

#### High Level Steps:

- Review all the fields in the **Seminar Ledger Entry** table to note any differences between the actual **Seminar Ledger Entry** table fields and the required set of fields described below.
- Correct the issues that were noted in the previous step and save the table.

#### Detailed Steps:

- 1) Review all the fields in the **Seminar Ledger Entry** table to note any differences between the actual Seminar Ledger Entry table fields and the required set of fields. 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
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
21	Res. Ledger Entry No.	Integer	
22	Source Type	Option	

Field No.	Field Name	Data Type	Length
23	Source No.	Code	20
24	Journal Batch Name	Code	10
25	Source Code	Code	10
26	Reason Code	Code	10
27	No. Series	Code	10
28	User ID	Code	50



Notice that the actual set of fields does not match this table. Apparently, the **Seminar Ledger Entry** table has been created by copying the **Seminar Journal Line** table. You now must correct these issues.

- 2) Verify the following changes:

Issue	Remarks
Entry No.	This field does not exist. Instead, there are the <b>Journal Template Name</b> and <b>Line No.</b> fields.
No. Series	This field does not exist. Instead, there is the <b>Posting No. Series</b> field.
User ID	This field does not exist.
Primary key	Primary key must only include the <b>Entry No.</b> field.

- 3) Correct the issues that were noted in the previous step and save the table.
- 4) From the list of fields, delete the **Journal Template Name** and **Line No.** fields.
- 5) Create the **Entry No.** field as the first field in the table.
- 6) Create the **User ID** field as the last field in the table.
- 7) Rename the **Posting No. Series** field to **No. Series**.
- 8) Set the **Caption** properties for any fields that you created or changed to match the field's Name.
- 9) Set the **TableRelation** property of the **User ID** field to relate to the **User Name** field of the table **User**.
- 10) Set the **ValidateTableRelation** property of the **User ID** field to **false**.
- 11) In the **OnLookup** trigger for the **User ID** field, enter the code that calls the **LookupUserID** function of the **User Management** codeunit.
- 12) Verify that the result look like this:

```
field(36;"User Id";code[50])
{
    TableRelation=user where("User Name"=field("User Id"));
    ValidateTableRelation=false;

trigger OnLookup();
var
    UserMgt : Codeunit "User Management";
begin
    usermgt.LookupUserID("User Id");
end;
}
```

- 14) Set the primary key to **Entry No.**
- 15) Rerumber 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.
- 16) Save the file using **Ctrl+S**

## Task 6: Review the Seminar Register Table

### High Level Steps:

- Review all the fields in the **Seminar Register** table to note any differences between the actual **Seminar Register** table fields and the required set of fields described below.
- Correct the issues that were noted in the previous step and save the table.

### Detailed Steps:

- 1) Review all the fields in the **Seminar Register** table to note any differences between the actual Seminar Ledger Entry table fields and the required set of fields. 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	50
7	Journal Batch Name	Code	10

2) Note the following issues:

Issue	Remarks
No.	The field does not exist. Instead, the field <b>Entry No.</b> is there. By convention <b>Register</b> tables always have the primary key field named <b>No.</b>
Journal Template Name	This field is not necessary in the register. The source of the transaction is kept in the <b>Source Code</b> field.

- 3) Change the **Name** and the **Caption** properties for the **Entry No.** field to **No.**
- 4) Delete the **Journal Template Name** field.
- 5) Rerumber 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.
- 6) Change the length of the **User ID** field to 50 characters
- 7) Remove the **Journal Template Name** field from Key 3
- 8) Save the file using **Ctrl+S**

## Customize the Source Code Setup Table and page

### Exercise Scenario

#### Exercise Scenario

In Dynamics 365 BC every transaction must leave a clear and obvious audit trail. The core feature for auditing transactions in Dynamics 365 BC 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 Dynamics 365 BC 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 Dynamics 365 BC.



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 7: Extend the Source Code Setup Table

#### High Level Steps:

- Create a new file **01\_SourceCodeSetupExt.al** in the **Table Extension** folder
- Create a table extension **123456701 "CSD SourceCodeSetupExt"**
- Add the **Seminar** field as a code 10 and a table relation to the **Source Code** table

#### Detailed Steps:

- 1) Create a new file **01\_SourceCodeSetupExt.al** in the **Table Extension** folder
- 2) Create a table extension by typing **tta** and selecting **tableextension**
- 3) Give it the number and name: **123456701 "CSD SourceCodeSetupExt"**
- 4) Replace **MyTargetTable** with the **Source Code Setup** table
- 1) Add a blank line in the **fields** section
- 2) Add the field as shown below:

```
fields
{
    field(123456700;"CSD Seminar";Code[10])
    {
        Caption='Seminar';
        TableRelation="Source Code";
    }
}
```

- 3) Update the documentation

```
tableextension 123456701 "CSD SourceCodeSetupExt" extends  
"Source Code Setup"  
  
// CSD1.00 - 2012-06-15 - D. E. Veler  
  
// Chapter 7 - Lab 1-7  
  
// - Added new fields:  
  
// - Seminar
```

- 4) Save the file using **Ctrl+S**

## Task 8: Extend the Source Code Setup Page

### High Level Steps:

- Create a new file **01\_SourceCodeSetupExt.al** in the **PageExtensions** folder
- Create a table extension **123456701 "CSD SourceCodeSetupExt"**
- Add a new group called Seminar
- Add the **Seminar** field as a code 10 and a table relation to the **Source Code** table

### Detailed Steps:

- 1) Create a new file **01\_SourceCodeSetupExt.al** in the **Page Extensions** folder
- 2) Create a table extension by typing **tpa** and selecting **pageextension**
- 3) Give it the number and name: **123456701 "CSD SourceCodeSetupExt"**
- 4) Replace **MyTargetPage** with the **Source Code Setup** page
- 5) Add a blank line in the **layout** section
- 6) Add the new group after the Cost Accounting group by typing:  
**7) addafter("Cost Accounting")**
- 8) Add the new group in the **addafter("Cost Accounting")** section by typing:  
**9) group(SeminarGroup)**
- 10) Add the caption **Seminar** to the group
- 11) After the **addafter("Cost Accounting")** section, add the seminar field as the first field in the SeminarGroup section by typing:
- 12) **addfirst(SeminarGroup)** and add the seminar field in the **addfirst(SeminarGroup)** section
- 13) Verify that the result look like this:

## Solution Development in Visual Studio Code

---

```
addafter("Cost Accounting")
{
    group(SeminarGroup)
    {
        Caption='Seminar';
    }
}
addfirst(SeminarGroup)
{
    field(Seminar;CSD Seminar)
    {
    }
}
```



The Seminar group and the seminar field needs to be given different identifiers. That is to ensure that we can reference them with the **addbefore** or **addafter** commands. The caption can be the same though.

14) Update the documentation

```
pageextension 123456702 "CSD SourceCodeExt" extends "Source
Code Setup"
// CSD1.00 - 2012-06-15 - D. E. Veloper
// Chapter 7 - Lab 1-8
```

15) Save the file using **Ctrl+S**

## Lab 7.2: 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.



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 mandatory and they are as follows.

Codeunit	Remarks
Seminar Jnl.-CheckLine	This codeunit checks each line before posting.
Seminar Jnl.-Post Line	This codeunit posts each line.
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.

### 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.



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:

- Create a new file **31\_SeminarJnlCheckLine.al** in the **Codeunits** folder
- Create codeunit **123456731, CSD Seminar Jnl.-Check Line**, and set the properties to specify the **CSD Seminar Journal Line** as the source table for this codeunit.

#### Detailed Steps:

- 1) Create a new file **31\_SeminarJnlCheckLine.al** in the Codeunits folder
- 2) Type **tco** and select **codeunit**
- 3) Replace **id** with **123456731** and **MyCodeunit** with "**CSD Seminar Jnl.-Check Line**"
- 4) Set the **TableNo** property to "**CSD Seminar Journal Line**" to make the table the source table for this codeunit.
- 5) Verify that the result looks like this:

```
codeunit 123456731 "CSD Seminar Jnl.-Check Line"
{
    TableNo="CSD Seminar Journal Line";

    trigger OnRun();
    begin
        end;

    var
        myInt : Integer;
}
```

## Task 2: Declare the Variables and Text Constants

### High Level Steps:

- Declare the global variables for the **G/L Setup** and **User Setup** tables
- Declare two date variables to keep track of allowed posting period starting and ending dates
  - o AllowPostingFrom
  - o AllowPostingTo

Declare the text constants that display errors if entries are posted on closing dates, or outside the allowed posting periods.

### Detailed Steps:

- 1) Locate the **var** section of the file
- 2) Detete the **myInt : Integer;** line
- 3) Add the lines as shown below:

```
var
  GLSetup : Record "General Ledger Setup";
  UserSetup : Record "User Setup";
  AllowPostingFrom : Date;
  AllowPostingTo : Date;
  ClosingDateTxt : Label 'cannot be a closing date.';
  PostingDateTxt : label 'is not within your range of
                           allowed posting dates.';
}
```



Text constants are defined differently in C/Side and in AL.  
 In C/Side they will be declared as textconst types. This also applies to objects converted with the Txt2AL.exe conversion tool.  
 They must be changed to the label type

### Task 3: Create the RunCheck Function

#### High Level Steps:

- Create the global **RunCheck** procedure 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.
- From the **OnRun** trigger, call the **RunCheck** procedure.

#### Detailed Steps:

- 1) Locate the end of the file just before the end-bracket ( } )
- 2) Add a new line and create a new procedure by typing **tpro** and selecting **procedure**
- 3) Remove the **local** property
- 4) Replace the **MyProcedure** with **RunCheck**
- 5) In the parentheses add the parameter for receiving a Seminar Journal Line record by reference by adding the following:  
**var SemJnlLine : Record "CSD Seminar Journal Line"**
- 6) 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.
- 7) Verify that the procedure looks like this:

```
local procedure RunCheck(var SemJnlLine : Record "CSD Seminar Journal Line");  
  
var  
    myInt : Integer;  
  
begin  
    With SemJnlLine do begin  
  
    end;  
end;
```

- 8) Locate the **OnRun** trigger.
- 9) Add the following line to the trigger:

```
trigger OnRun();  
  
begin  
    RunCheck(Rec);  
  
end;
```

## Task 4: Add Code to the RunCheck Function

### High Level Steps:

- 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.
- Make sure that the **Posting Date**, **Resource No.**, and **Seminar No.** fields are not empty.
- Depending on the value of the **Charge Type field**, make sure that the **Instructor Resource No.**, **Room Resource No.**, and **Participant Contact No.** fields are not empty.
- If the line is Chargeable, make sure that the **Bill-to Customer No.** field is not blank.
- Show an error if the **Posting Date** is a closing date.
- 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.
- Show an error if the **Document Date** field is a closing date
- Then save the codeunit.

### Detailed Steps:

- 1) Add a new line as the first line in the **With** block
- 2) Enter code to test whether the **Seminar Journal Line** is empty by using the **EmptyLine** function. If the line is empty, the function exits

```
if EmptyLine then
    exit;
```



For all other steps in this task, keep adding the code to the **RunCheck** function trigger, just before the **end** of the **With** block

- 3) Test that the **Posting Date**, **Instructor Resource No.**, and **Seminar No.** fields are not empty.

```
TestField("Posting Date");
TestField("Instructor Resource No.");
TestField("Seminar No.");
```

- 4) Depending on the value of the **Charge Type** field, make sure that the **Instructor Code**, **Room Resource No.**, and **Participant Contact No.** fields are not empty.

```
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;
```

- 5) If the line is **Chargeable**, make sure that the **Bill-to Customer No.** field is not blank.

```
if Chargeable then
    TestField("Bill-to Customer No.');
```

- 6) Show an error if the Posting Date is a closing date

```
if "Posting Date" = ClosingDate("Posting Date") then
    FieldError("Posting Date",ClosingDateTxt);
```

- 7) 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.

```

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 := DMY2Date(31,12,9999);
  end;
  if ("Posting Date" < AllowPostingFrom) OR
    ("Posting Date" > AllowPostingTo) then
    FieldError("Posting Date", PostingDateTxt);

```



This check is a standard check in all journal posting codeunits. The variables **AllowPostingFrom** and **AllowPostingFrom** are global variables in the codeunit. This means that, because of the testing for if **AllowPostingFrom** and **AllowPostingFrom** are set, it will only be executed the first time the **Seminar Check-Line** codeunit is run. Second time it is run, the values will already be in memory from the first run.



When a date must be hardcoded, it is best practice to use the **DMY2Date** command. This mitigates any problems when handling multiple regional settings.

## Solution Development in Visual Studio Code

---

8) Show an error if the Document Date field is a closing date, and then

```
if ("Document Date" <> 0D) then  
  if ("Document Date" = CLOSINGDATE("Document Date")) then  
    FIELDERROR("Document Date", PostingDateTxt);
```

9) Add the necessary documentation:

```
// CSD1.00 - 2018-01-01 - D. E. Veloper  
// Chapter 7 - Lab 2-1
```

10) Save the file using **Ctrl+S**

## Create the Seminar Jnl.-Post Line Codeunit

### Exercise Scenario

Now the **Seminar Jnl.-Post Line** codeunit must be created. 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.
- Populate the ledger entry from the fields of the Seminar Journal Line table and Insert it.



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 5: Create the Codeunit

### High Level Steps:

- Create a new file **32\_SeminarJnlPostLine.al** in the Codeunits folder
- Create the codeunit **123456732, CSD Seminar Jnl.-Post Line**
- Set the properties to specify the **CSD Seminar Journal Line** as the source table for this codeunit.

### Detailed Steps:

- 1) Create a new file **32\_SeminarJnlPostLine.al** in the Codeunits folder
- 2) Type **tco** and select **codeunit**
- 3) Replace **id** with **123456732** and **MyCodeunit** with "**CSD Seminar Jnl.-Post Line**"
- 4) Set the **TableNo** property to "**CSD Seminar Journal Line**" to make the table the source table for this codeunit.
- 5) Verify that the result looks like this:

```
codeunit 123456732 "CSD Seminar Jnl.-Post Line"
{
    TableNo = "CSD Seminar Journal Line";

    trigger OnRun();
    begin
    end;
```

### Task 6: Declare the Variables

#### High Level Steps:

- 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) Locate the **var** section of the file
- 4) Detete the **myInt : Integer;** line
- 5) Add the lines as shown below:

```
var
  SeminarJnlLine : Record "CSD Seminar Journal Line";
  SeminarLedgerEntry : Record "CSD Seminar Ledger Entry";
  SeminarRegister : Record "CSD Seminar Register";
  SeminarJnlCheckLine : Codeunit "CSD Seminar Jnl.-Check Line";
  NextEntryNo : Integer;
```

## Task 7: Create the Functions

### High Level Steps:

- Create the **RunWithCheck** and **Code** functions.
- The **RunWithCheck** function receives a **Seminar Journal Line** as a parameter **SeminarJnlLine2** by reference.
- Enter code in the appropriate trigger so that when the program runs codeunit **123456732, CSD Seminar Jnl.-Post Line**, it runs the **RunWithCheck** function for the current record.
- Enter code in the **RunWithCheck** function trigger so that the function copies the **SeminarJnlLine** from the **SeminarJnlLine2** record, runs the **Code** function based on the **SeminarJnlLine** record, and then restores the **SeminarJnlLine2** record back from the **SeminarJnlLine** record.

### Detailed Steps:

- 1) Locate the end of the file just before the end-bracket ( **}** )
- 2) Add a new line and create a new procedure by typing **tpro** and selecting **procedure**
- 3) Replace the **MyProcedure** with **RunWithCheck**
- 4) Remove the **local** property
- 5) In the parentheses add the parameter for receiving a Seminar Journal Line record by reference by adding the following:  
**var SeminarJnlLine : Record "CSD Seminar Journal Line"**
- 6) 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.
- 7) Verify that the procedure looks like this:

```
procedure RunWithCheck(var SeminarJnlLine2 : Record "CSD Seminar Journal Line");
var
    myInt : Integer;
begin
    with SeminarJnlLine2 do begin
```

- 8) Repeat this procedure to create a new local procedure **Code**. This procedure is created without a parameter.
- 9) In the **Code** procedure, the **with** block must refer to the **SeminarJnlLine** record
- 10) Enter code in the appropriate trigger so that when the program runs codeunit **123456732, CSD Seminar Jnl.-Post Line**, it runs the **RunWithCheck** function for the current record.
- 11) 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.

12) Verify that the procedure looks like this:

```
procedure RunWithCheck(var SeminarJnlLine2 : Record "CSD Seminar Journal Line");
var
    myInt : Integer;
begin
    with SeminarJnlLine2 do begin
        SeminarJnlLine:=SeminarJnlLine2;
        Code();
        SeminarJnlLine2:=SeminarJnlLine;
    end;
end;
```



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 passed as a local variable 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 8: Add Code to the Code Function

### High Level Steps:

- Check whether the **SeminarJnlLine** is empty by using the **EmptyLine** function. If it is empty, the function exits.
- Runs the **RunCheck** function of the **SeminarJnlCheckLine** codeunit.
- 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.
- If the **Document Date** is empty, the set the **Document Date** to the **Posting Date**.
- 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.
- 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 following code in the **with** block. Check whether the **SeminarJnlLine** is empty by using the **EmptyLine** function. If it is empty, the function exits



For all successive steps in this task, keep adding the code to the Code function trigger, just before the **end** of the **with** block

- 2) Run the **RunCheck** function of the **SeminarJnlCheckLine** codeunit.
- 3) 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.

```

if NextEntryNo = 0 then begin
    SeminarLedgerEntry.LockTable;
    if SeminarLedgerEntry.FindLast then
        NextEntryNo := SeminarLedgerEntry."Entry No.";
    NextEntryNo := NextEntryNo + 1;
end;

```



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

- 4) If the **Document Date** is empty, the set the **Document Date** to the **Posting Date**.
- 5) 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.



If the **No.** field of the **SeminarRegister** record is zero, then the register record has not yet been created.

- 6) 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;

```



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 **Next EntryNo.**, 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 **Next EntryNo.** 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.

- 8) Verify that the code looks like this:

```

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;

```

## Solution Development in Visual Studio Code

---

```
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."Posting No. Series" :=
    "Posting No. Series";
```

9) Add the necessary documentation:

```
// CSD1.00 - 2018-01-01 - D. E. Veloper
// Chapter 7 - Lab 2-8
```

10) Save the file using **Ctrl+S**

## 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 9: Create the Seminar Ledger Entries Page

### High Level Steps:

- Create a new file **21\_SeminarLedgerEntries.al** in the **Pages** folder
- Create a noneditable list page for the **Seminar Ledger Entry** table by using the **tpa** snippet.
- Name the page as **123456721, CSD Seminar Ledger Entries**
- Then add the fields to the page. Save and close it

### Detailed Steps:

- 19) In the VS Code Explorer window, click the **Page** folder, then right-click and select the **New File** menu item
- 20) Name the file **21\_SeminarLedgerEntries.al** and press **[Enter]**
- 21) Enter **tpa** in the first line and select the **tpage, Page of type list** snippet
- 22) Enter **123456721** as the **ID** and **CSD Seminar Ledger Entries** as the **name** and press enter
- 23) Set the Source table to be **CSD Seminar Ledger Entry**
- 24) Add a **Caption** property
- 25) Add the **Editable** property to be **false**
- 26) Add the **UsageCategory** to be **Lists**

- 11) In the repeater(Group) section, add the following fields

```
repeater(Group)
  field("Posting Date";"Posting Date")
  field("Document No.;"Document No.")
  field("Document Date";"Document Date")
  field("Entry Type";"Entry Type")
  field("Seminar No.;"Seminar No.)
  field(Description;Description)
  field("Bill-to Customer No.;"Bill-to Customer No.)
  field("Charge Type";"Charge Type")
  field(Type;Type)
```

```
field(Quantity;Quantity)
field("Unit Price";"Unit Price")
field("Total Price";"Total Price")
field(Chargeable;Chargeable)
field("Participant Contact No.;"Participant Contact No.)
field("Participant Name;"Participant Name")
field("Instructor Resource No.;"Instructor Resource No.)
field("Room Resource No.;"Room Resource No.)
field("Starting Date;"Starting Date")
field("Seminar Registration No.;"Seminar Registration
No.)
field("Entry No.;"Entry No.)
```



In the window above, all { and } brackets have been removed for space.

The fastest way to create many fields in a page is to create three lines:

```
Field()
{
}
```

Copy the lines and paste them 20 times. Then insert the field names

12) Add the **Record Links** and **Notes** system FactBoxes.

13) Set the **Visible** property for the **Document Date** field to **false**.

14) Add the necessary documentation:

```
// CSD1.00 - 2018-01-01 - D. E. Veloper
// Chapter 7 - Lab 2-9
```

15) Save the page using **Ctrl+S**

## 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 10: Create the Seminar Reg.-Show Ledger Codeunit

### High Level Steps:

- Create a new file **34\_SeminarRegShowLedger.al** in the **Codeunits** folder
- Name the codeunit **123456734, CSD Seminar Reg.-Show Ledger**
- Set the properties to specify the Seminar Register as the source table for this codeunit.
- Declare a global variable for the **CSD Seminar Ledger Entry** table.
- Enter code in the appropriate trigger so that when the program runs the codeunit, the codeunit runs the **CSD Seminar Ledger Entries** page showing only those entries between the **From Entry No.** field and the **To Entry No.** field on the Seminar Register

### Detailed Steps:

- 1) Create a new file **34\_SeminarRegShowLedger.al** in the Codeunits folder
- 2) Create the codeunit by typing **tco** and selecting codeunit
- 3) Replace **id** with **123456734** and **MyCodeunit** with **CSD Seminar Reg.-Show Ledger**
- 4) Set the **TableNo** property to specify the **CSD Seminar Register** as the source table for this codeunit.
- 5) Locate the **var** section and replace the **myInt : Integer;** line with **SeminarLedgerEntry : Record "CSD Seminar Ledger Entry";**
- 6) In the onRun trigger, enter the following code:

```
trigger OnRun();
begin
    SeminarLedgerEntry.SETRANGE("Entry No.", "From Entry No.",
        "To Entry No.");
    page.Run(Page::"Seminar Ledger Entries",
```

- 7) Add the necessary documentation:

```
// CSD1.00 - 2018-01-01 - D. E. Veloper
// Chapter 7 - Lab 2-10
```

- 8) Save the codeunit using **Ctrl+S**

### Task 11: Create the Seminar Registers Page

#### High Level Steps:

- Create a new file **22\_SeminarRegisters.al** in the **Pages** folder
- Create a noneditable list page for the **Seminar Register** table by using the **tpa** snippet.
- Name the page as **123456722, CSD Seminar Registers**
- Then add the fields to the page.
- Add an action to the **Navigation** action container to run the **Seminar Reg.-Show Ledger** codeunit.
- Save and close it

#### Detailed Steps:

- 1) In the VS Code Explorer window, click the **Page** folder, then right-click and select the **New File** menu item
- 2) Name the file **22\_SeminarRegisters.al** and press **[Enter]**
- 3) Enter **tpa** in the first line and select the **tpage, Page of type list** snippet
- 4) Enter **123456722** as the **ID** and **CSD Seminar Registers** as the **name** and press enter
- 5) Set the Source table to be **CSD Seminar Registers**
- 6) Add a **Caption** property
- 7) Add the **Editable** property to be **false**
- 8) Add the **UsageCategory** to be **Lists**
- 9) In the repeater(Group) section, add the following fields

```
field("No.;" "No.")  
field("Creation Date;" "Creation Date")  
field("User ID;" "User ID")  
field("Source Code;" "Source Code")  
field("Journal Batch Name;" "Journal Batch Name")
```

- 10) Add the **Record Links** and **Notes** system FactBoxes.
- 11) Add a new action **Seminar Ledgers** in the **Navigation** area
- 12) Set the **Image** property to be **WarrantyLedger**
- 13) Set **RunObject** property to run **the Seminar Reg.-Show Ledger** codeunit.
- 14) Verify that the code looks like this:

```
area(Navigation)
{
    action("Seminar Ledgers")
    {
        Image=WarrantyLedger;
        RunObject=codeunit SeminarRegShowLedger;
```

15) Add the necessary documentation:

```
// CSD1.00 - 2018-01-01 - D. E. Veler
// Chapter 7 - Lab 2-11
```

16) Save the page using **Ctrl+S**

# Lab 7.3: 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 Dynamics 365 BC, 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 Dynamics 365 BC 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.



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.

## 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 file and save it under a new name. Then open the file and give it 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:

- Create the **Posted Seminar Reg. Header** table by copying the file containing

the **Seminar Registration Header** table.

- Change the file name to **18\_PostedSeminarRegHeader.al**.
- Replace the **id** with **123456718** and the name with **CSD Posted Seminar Reg. Header**.
- Remove all the code from the table.
- Delete and rename fields to match the standards for posted document header tables and add the **User ID** and **Source Code** fields.
- Add code to the **OnLookup** trigger of the **User ID** field to run the **LookupUser** function of the **User Management** codeunit.
- Correct the calculation formula for the **Comment** field.
- Remove all global variables
- Set the **Caption** property for the table to match its **Name**, and then save and close the table.

#### Detailed Steps:

- 1) Locate the **10\_SeminarRegistrationHeader.al** file
- 2) Press **Ctrl+C** and **Ctrl+V** on the file name
- 3) Rename the new file to **18\_PostedSeminarRegHeader.al** and press **[Enter]**
- 4) Replace the **ID** with **123456718** and replace the **name** with the name with **CSD Posted Seminar Reg. Header**
- 5) Change the **Caption** property to match the name
- 6) Delete all **OnValidate** and **OnLookup** triggers for all fields in the file
- 7) Delete the **OnDelete**, the **OnInsert** triggers
- 8) Delete the **InitRecord** and the **AssistEdit** procedures
- 9) Delete all Global variables
- 10) Delete the **Posting No.** field
- 11) Add a new field with Id 29 and name **User Id** with type code length 50
- 12) Set the table relation to **User Id** field for the **User** table
- 13) Set the **ValidateTableRelation** to be false
- 14) Create the **OnLookup** trigger for the **User Id** field
- 15) Add a local variable **UserMgt** to be codeunit **User Management**
- 16) Enter the following code in the trigger:

```
field(29;"User Id";Code[50])
```

```
{
  Caption='User Id';
  TableRelation=User;
  ValidateTableRelation=false;
}
```

- 17) Add a new field with Id 30 and name **Source Code** with type code length 10

```
field(30;"Source Code";Code[10])  
{  
    Caption='Source Code';  
    TableRelation="Source Code";  
}
```

- 18) Set the table relation to **Source Code** field for the **Source Code** table  
19) Locate the Comment field and correct the calculation formula to:

```
Field(22; Comment; Boolean)  
{  
    Caption = 'Comment';  
    CalcFormula = Exist ("Seminar Comment Line" where  
        ("Table Name" = const ("Posted Seminar Registration"),  
        "No." = Field ("No."));  
    Editable = false;  
    FieldClass = FlowField;  
}
```

- 20) Change the documentation to:

```
// CSD1.00 - 2018-01-01 - D. E. Veloper  
// Chapter 7 - Lab 3-1
```

- 21) Save the file Using **Ctrl+S**

## Task 2: Correct the Seminar Comment Line table

### High Level Steps:

- Locate the **04\_SeminarCommentLine.al** file
- Add the Posted Seminar Registration Table to the table relation of the No. field
- Save and close the file.

### Detailed Steps:

- 1) Locate the **04\_SeminarCommentLine.al** file
- 2) Locate the **No.** field.
- 3) Change the table relation to include the Posted Seminar Registration:

```
TableRelation=if ("Table Name"=CONST(Seminar)) "Seminar"  
else if("Table Name"=const("Seminar Registration"))  
"Seminar Registration Header"  
else if("Table Name"=const("Posted Seminar Registration"))  
"Posted Seminar Reg. Header";
```

- 4) Update the documentation to:

```
// CSD1.00 - 2018-01-01 - D. E. Veloper  
// Chapter 5 - Lab 2-1  
// Chapter 7 - Lab 3-2
```

- 5) Save the file Using **Ctrl+S**

### Task 3: Create the Posted Seminar Reg. Line Table

#### High Level Steps:

- Create the **Posted Seminar Reg. Line** table by copying the file containing the **Seminar Registration Line** table.
- Change the file name to **19\_PostedSeminarRegLine.al**.
- Replace the **id** with **123456719** and the name with **CSD Posted Seminar Reg. Line**.
- Correct the table relation for the **Document No.** field.
- Remove all the code from the table.
- Remove all global variables
- Set the **Caption** property for the table to match its **Name**, and then save and close the table.

#### Detailed Steps:

- 1) Locate the **11\_SeminarRegistrationLine.al** file
- 2) Press **Ctrl+C** and **Ctrl+V** on the file name
- 3) Rename the file **19\_PostedSeminarRegLine.al** and press **[Enter]**
- 4) Replace the **ID** with **123456719** and replace the **name** with the name with **CSD Posted Seminar Reg. Line**
- 5) Change the **Caption** property to match the name
- 6) Delete all **OnValidate** and **OnLookup** triggers for all fields in the file
- 7) Delete the **OnDelete**, the **OnInsert** triggers
- 8) Delete all procedures
- 9) Delete all Global variables

10) Change the documentation to:

```
// CSD1.00 - 2018-01-01 - D. E. Veler  
// Chapter 7 - Lab 3-3
```

11) Save the file Using **Ctrl+S**

## Task 4: Create the Posted Seminar Charge Table

### High Level Steps:

- Create the **Posted Seminar Charge** table by copying the file containing the **Seminar Charge** table.
- Change the file name to **21\_PostedSeminarCharge.al**.
- Replace the **id** with **123456721** and the name with **CSD Posted Seminar Charge**.
- Correct the table relation for the **Document No.** field.
- Remove all the code from the table.
- Remove all global variables
- Set the **Caption** property for the table to match its **Name**, and then save and close the table.

### Detailed Steps:

- 12) Locate the **12\_SeminarCharge.al** file
- 13) Press **Ctrl+C** and **Ctrl+V** on the file name
- 14) Rename the file **21\_PostedSeminarCharge.al** and press **[Enter]**
- 15) Replace the **ID** with **123456721** and replace the **name** with the name with **CSD Posted Seminar Charge**
- 16) Change the **Caption** property to match the name
- 17) Delete all **OnValidate** and **OnLookup** triggers for all fields in the file
- 18) Delete the **OnDelete**, the **OnInsert** triggers
- 19) Delete all Global variables
- 20) Change the documentation to:

```
// CSD1.00 - 2018-01-01 - D. E. Veloper
// Chapter 7 - Lab 3-4
```

- 21) Save the file Using **Ctrl+S**

## 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, the development team have developed the following pages for accessing the posted document information: Posted Seminar Registration, Posted Seminar Reg. List, and Posted Seminar Charges.

## Task 5: Import the Posted Registration page Objects

### High Level Steps:

Drag the .al files from  
**Solution Development Course Objects\Mod07\Starter C\Pages**  
 into the **Pages** folder in the workspace  
 Review the files and fix the errors until the files are green

### Detailed Steps:

**Import all the .ai files from on the desktop and place in the workspace**

- 1) Open the **Solution Development Course Objects\Mod07\ Starter C\Pages** folder on the Desktop in a File Explorer
- 2) Mark all files and drag them into the **Pages** folder of the project
- 3) Verify that they are placed in the Tables folder and that they are all red
- 4) Review the files and fix the errors until the files are green
- 5) Save the files Using **Ctrl+S**

## Lab 7.4: 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.



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 <b>Ledger Entry</b> table, you must first add those fields to the matching <b>Journal Line</b> table.
Codeunit	212	Res. Jnl.-Post Line	To move the fields from

			a <b>Journal Line</b> table to the matching <b>Ledger Entry</b> 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 <b>Ledger Entries</b> page.

## Extend the Objects

### Exercise Scenario

You start by adding the necessary fields to the Res. Ledger Entry and Res. Journal Line tables by creating Table Extensions, and then add the fields to the Resource Ledger Entries page by creating a Page Extension.

Lastly, a new codeunit with an Even Subscription interacting with the Res. Jnl.-Post Line codeunit.

## Task 1: Add fields to the Res. Ledger Entry Table

### High Level Steps:

Create a new file **03\_ResourceLedgerEntryExt.al** in the **Table Extensions** folder

Create a table extension **123456703** with the name **ResourceLedgerEntryExt** to extend the **Res. Ledger Entry** table

Add the two fields:

- **123456700 "CSD Seminar No."** Code 20, with relation to the **Seminar table**
- **123456701 "CSD Seminar Registration No."** Code 20, with relation to the **Seminar Registration Header table**

### Detailed Steps:

- 1) Create a new file **03\_ResourceLedgerEntryExt.al** in the **TableExtensions** folder
- 2) Create a table extension using the **ttab** snippet
- 3) Replace **id** with **123456703**
- 4) Replace **MyExtension** with **ResourceLedgerEntryExt**
- 5) Replace **MyTargetTable** with **Res. Ledger Entry**
- 6) In the fields section add the following code:

```
field(123456700;"CSD Seminar No.";code[20])
{
    Caption='Seminar No.';
    TableRelation="CSD Seminar";
}
```

```
field(123456701;"CSD Seminar Registration No.";code[20])
```

```
{
```

```
    Caption='Seminar Registration No. ';
```

```
    TableRelation="CSD Seminar Reg. Header";
```

```
}
```

7) Change the documentation to:

```
// CSD1.00 - 2018-01-01 - D. E. Veloper
```

```
// Chapter 7 - Lab 4-1
```

8) Save the file Using **Ctrl+S**

### Task 2: Add fields to the Res. Journal Line Table

#### High Level Steps:

Create a new file **04\_ResJournalLineExt.al** in the **Table Extensions** folder

Create a table extension **123456704** with the name **ResJournalLineExt** to extend the **Res. Journal Line** table

Add the two fields:

- **123456700 "CSD Seminar No."** Code 20, with relation to the **Seminar table**
- **123456701 "CSD Seminar Registration No."** Code 20, with relation to the **Seminar Registration Header** table

#### Detailed Steps:

- 1) Create a new file **04\_ResJournalLineExt.al** in the **Table Extensions** folder
- 2) Create a table extension using the **ttab** snippet
- 3) Replace **id** with **123456704**
- 4) Replace **MyExtension** with **ResJournalLineExt**
- 5) Replace **MyTargetTable** with **Res. Journal Line**
- 6) In the fields section add the following code:

```
field(123456700;"CSD Seminar No.";code[20])  
{  
    Caption='Seminar No.';  
    TableRelation="CSD Seminar";  
}  
  
field(123456701;"CSD Seminar Registration No.";code[20])  
{  
    Caption='Seminar Registration No.';  
    TableRelation="CSD Seminar Reg. Header";  
}
```

- 7) Change the documentation to:

```
// CSD1.00 - 2018-01-01 - D. E. Veloper  
// Chapter 7 - Lab 4-2
```

- 8) Save the file Using **Ctrl+S**



When the fields with same **Nos.** and **Names** exist in multiple tables or table extensions, you can copy them from one table or table extension to another. Here, instead of creating fields, you can copy the fields from the **Res. Ledger Entry** Extension.

## Task 3: Add fields to the Res. Ledger Entry Page

### High Level Steps:

Create a new file **03\_ResourceLedgerEntryExt.al** in the **PageExtensions** folder

Create a table extension 123456703 with the name **ResourceLedgerEntryExt** to extend the **Res. Ledger Entries** Page

Add the two fields:

- **123456700 CSD Seminar No.**
- **123456701 CSD Seminar Registration No.**

### Detailed Steps:

- 1) Create a new file **03\_ResourceLedgerEntryExt.al** in the **PageExtensions** folder
- 2) Create a table extension using the **tpag** snippet and select the **pageext** snippet
- 3) Replace **id** with **123456703**
- 4) Replace **MyExtension** with **ResourceLedgerEntryExt**
- 5) Replace **MyTargetPage** with **Resource Ledger Entries**
- 6) In the **layout** section create an **addlast(Content)** section
- 7) In the **addlast(Content)** section add the two fields:
  - a. **CSD Seminar No.**
  - b. **CSD Seminar Registration No.**
- 8) Remove all unused sections
- 9) Change the documentation to:

```
// CSD1.00 - 2018-01-01 - D. E. Veloper
```

```
// Chapter 7 - Lab 4-3
```

- 10) Verify that the file looks like this:

## Solution Development in Visual Studio Code

---

```
pageextension 123456703 "CSD ResourceLedgerEntryExt" extends
"Resource Ledger Entries"

// CSD1.00 - 2018-01-01 - D. E. Veloper

// Chapter 7 - Lab 4-3

{

    layout

    {

        addlast(Content)

        {

            field("Seminar No.;"CSD Seminar No.")

            {

                }

            field("Seminar Registration No.;

                "CSD Seminar Registration No.")

                {

                    }

                }

            }

        }

    }

}
```

11) Save the file Using **Ctrl+S**

## Task 4: Create the EventSubscriptions codeunit

### High Level Steps:

- Create a new file **39\_EventSubscriptions.al** in the **Codeunits** folder
- Create a new codeunit **123456739** with the name **CSD EventSubscriptions**
- Create a new **EventSubscription** to intercept codeunit 212  
**OnBeforeResLedgEntryInsert**
- Create a local procedure **PostResJnlLineOnBeforeResLedgEntryInsert**
- Copy all parameters from the **OnBeforeResLedgEntryInsert** procedure in the **Res. Jnl.-Post Line** codeunit
- Transfer the the two fields from the **Res. Journal Line** table to the **Res. Ledger Entry** table:
  - o **CSD Seminar No.**
  - o **CSD Seminar Registration No.**

### Detailed Steps:

- 1) **39\_EventSubscriptions.al** in the **Codeunits** folder
- 2) Create a new codeunit **123456739** with the name **CSD EventSubscriptions** using the **tco** snippet and select the **codeunit** snippet
- 3) Replace **id** with **123456739**
- 4) Replace **MyCodeunit** with "**CSD EventSubscriptions**"
- 5) Delete everything in the codeunit except the braces ({}{})
- 6) Create an event subscription using the **teven** snippet end selecting **tevents**
- 7) Replace **ObjectType::ObjectType** with **ObjectType::Codeunit**
- 8) Replace **ObjectId** with **212**
- 9) Delete '**OnSomeEvent**' and press **Ctrl+[Space]**
- 10) Select **OnBeforeResLedgEntryInsert**
- 11) Replace '**ElementName**' with '' (two single quotes)
- 12) Replace **SkipOnMissingLicense** with **true**
- 13) Replace **SkipOnMissingPermission** with **true**
- 14) Create a blank line underneath the **EventSubscriber**
- 15) Create a local procedure using the snippet **tpro**
- 16) Replace **MyProcedure** with  
**PostResJnlLineOnBeforeResLedgEntryInsert**
- 17) Create a local variable **C212 : codeunit "Res. Jnl.-Post Line"**
- 18) In the procedure type **C212.RunWithCheck** and ignore the error line
- 19) Click the **RunWithCheck** and click **F12** (Go to Definition)
- 20) Copy the parameters from the **OnBeforeResLedgEntryInsert** procedure to the parameters in the **PostResJnlLineOnBeforeResLedgEntryInsert** procedure
- 21) Delete the **C212.RunWithCheck** line
- 22) Delete the **C212 : codeunit "Res. Jnl.-Post Line"** variable
- 23) Add the code as shown below:

```
[EventSubscriber(ObjectType::Codeunit, 212,
'OnBeforeResLedgEntryInsert', '', true, true)]

local procedure PostResJnlLineOnBeforeResLedgEntryInsert
  (var ResLedgerEntry : Record "Res. Ledger Entry";
   ResJournalLine : Record "Res. Journal Line");

begin
  ResLedgerEntry."Seminar No.":=
    ResJournalLine."Seminar No.";
  ResLedgerEntry."Seminar Registration No.":=
    ResJournalLine."Seminar Registration No.";
end;
```

24) Add the following documentation to:

```
// CSD1.00 - 2018-01-01 - D. E. Velooper
// Chapter 7 - Lab 4-4
```

25) Save the file using **Ctrl+S**

## Lab 7.5: Creating the Codeunits for Document Posting

### Scenario

The codeunits for seminar registration posting has been partially created. The Seminar-Post (Yes/No) codeunit has been completed. For the Seminar-Post codeunit the variables and functions has been declared. You must complete the Seminar- Post and Seminar-Post (Yes/No) codeunits.

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.

### 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 Seminar Post Files

#### High Level Steps:

- Drag the .al files from **Solution Development Course Objects\Mod07\Starter E\Codeunits** into the **Codeunits** folder in the workspace
- Review the files and fix the errors until the files are green

#### Detailed Steps:

##### **Import all the .al files from on the desktop and place in the workspace**

- 1) Open the **Solution Development Course Objects\Mod07\ Starter E\Codeunits** folder on the Desktop in a File Explorer
- 6) Mark all files and drag them into the **Codeunits** folder of the project
- 7) Verify that they are placed in the folder and that they are all red
- 8) Review the files and fix the errors until the files are green
- 9) Save the files Using **Ctrl+S**

### Task 2: Complete the CopyCommentLines Function

#### High Level Steps:

- In the **CopyCommentLines** procedure trigger, enter the code that finds records in the **Seminar Comment Line** table that matches the specified

## Solution Development in Visual Studio Code

---

**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:

- 1) Locate the **CopyCommentLines** procedure
- 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**.
- 3) Insert the following code in the **CopyCommentLines** procedure:

```
local procedure CopyCommentLines(FromDocumentType :  
    Integer;ToDocumentType : Integer;FromNumber :  
    Code[20];ToNumber : Code[20]);  
  
begin  
    SeminarCommentLine.Reset;  
    SeminarCommentLine.SetRange("Table Name",  
        FromDocumentType);  
    SeminarCommentLine.SetRange("No.",FromNumber);  
    if SeminarCommentLine.FindSet then repeat  
        SeminarCommentLine2:=SeminarCommentLine;  
        SeminarCommentLine2."Table Name":=ToDocumentType;  
        SeminarCommentLine2."No.":=ToNumber;  
        SeminarCommentLine2.Insert;  
    until SeminarCommentLine.Next=0;  
end;
```

## Task 3: Complete the **CopyCharges** Function

### High Level Steps:

- 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.**.



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.

- 2) Insert the following code in the **CopyCharges** procedure:

```
local procedure CopyCharges(FromNumber : Code[20]; ToNumber :
Code[20]);
begin
  SeminarCharge.Reset;
  SeminarCharge.SetRange("Document No.", FromNumber);
  if SeminarCharge.FindSet then repeat
    PstdSeminarCharge.TransferFields(SeminarCharge);
    PstdSeminarCharge."Document No." := ToNumber;
    PstdSeminarCharge.Insert;
  until SeminarCharge.Next=0;
end;
```

## Task 4: Complete the PostResJnlLine Function

### High Level Steps:

- In the **PostResJnlLine** function trigger, enter **with** code block for the **SeminarRegHeader** record variable.
- 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*.

- Assigns the **Document No.** from **No.** field on the **PstdSeminarRegHeader** record variable.
  - Assigns the **Resource No.** from the **Resource** record parameter.
- In the **with** code block, append the code that assigns the following field values from the seminar **Registration Header** record:
- Posting Date
  - Reason Code
  - Description
  - Gen. Prod. Posting Group
  - 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**.
- 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. 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.
- 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.

```
local procedure PostResJnlLine(Resource : Record Resource) :  
  Integer;  
begin  
  with SeminarRegHeader do begin  
      
      
    end;  
  end;
```

- 2) In the **WITH** code block, enter the code that does the following:
  - a. Makes sure that the **Quantity Per Day** field on the **Resource** record is not empty
  - b. Initializes a **Resource Journal Line** record.

- c. Sets its **Entry Type** to **Usage**.
- d. Assigns the **Document No.** from **No.** field on the **PstdSeminarRegHeader** record variable.
- e. Assigns the **Resource No.** from the **Resource** record parameter.

Enter the following code:

```
with SeminarRegHeader do begin
  ResJnlLine.Init;
  ResJnlLine."Entry Type":=ResJnlLine."Entry Type"::Usage;
  ResJnlLine."Document No.":=PstdSeminarRegHeader."No.";
  ResJnlLine."Resource No.":=Resource."No.";
end;
```

- 3) In the **with** code block, append the code that assigns the following field values from the seminar **Registration Header** record:
- a. Posting Date
  - b. Reason Code
  - c. Description
  - d. Gen. Prod. Posting Group
  - e. 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**.

Enter 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. 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.

```
ResJnlLine.Quantity := Duration *
    Resource."Quantity Per Day";
ResJnlLine."Total Cost" := ResJnlLine."Unit Cost" *
    ResJnlLine.Quantity;
ResJnlLine."Seminar No." := "Seminar No.";
```

- 5) After the **with** block, find the last **Resource Ledger Entry**, and return its **Entry No.** field value as the function return value

```
ResLedgEntry.FindLast;
exit(ResLedgEntry."Entry No.");
```

- 6) Save the file using **Ctrl+S**

## Task 5: Complete the PostSeminarJnlLine Function

### High Level Steps:

- In the **PostSeminarJnlLine** function trigger, enter **with** code block for the **SeminarRegHeader** record variable.
- 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.
- To the **with** code block, append the code that compares the **ChargeType** parameter to all possible option values that it can have.
- 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.
- 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.
- 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

- 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**.

- If **ChargeType** is *Charge*, then assign the fields to the **SeminarJnlLine** record variable from the **SeminarCharge** record variable. Assign the following fields:
  - Description
  - Bill-to Customer No.
  - Type
  - Quantity
  - Unit Price
  - Total Price
  - ChargeableChargeable is set from **To Invoice**.
- 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.

```
local procedure PostSeminarJnlLine(ChargeType : Option  
Instructor,Room,Participant,Charge);  
begin  
  with SeminarRegHeader do begin  
      
  end;  
end;
```

- 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:

- a. Seminar No.
- b. Posting Date
- c. Document Date
- d. Document No.
- e. Charge Type
- f. Instructor Resource No.
- g. Starting Date
- h. Seminar Registration No.

- i. Room Resource No.
- j. Source Type
- k. Source Code
- l. Reason Code
- m. Posting No.

```

with SeminarRegHeader do
begin
  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.

```
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.

```

case ChargeType of
  ChargeType::Instructor :
    Begin
      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);
    end;

```

- 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.

```

ChargeType::Room :
begin
  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);
end;

```

- 6) If the **ChargeType** is *Participant*, assign the fields to the **SeminarJnlLine**

record variable from the **SeminarRegLine** record variable. Assign the following fields:

- a. Bill-to Customer No.
- b. Participant Contact No.
- c. Participant Name
- d. Description
- e. Chargeable
- f. Unit Price
- g. 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**.

```
ChargeType::Participant :  
begin  
    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;  
end;
```

- 7) If **ChargeType** is *Charge*, then assign the fields to the **SeminarJnlLine** record variable from the **SeminarCharge** record variable. Assign the following fields:
- a. Description
  - b. Bill-to Customer No.

- c. Type
- d. Quantity
- e. Unit Price
- f. Total Price
- g. Chargeable

Chargeable is set from To Invoice.

```
ChargeType::Charge :  
begin  
    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";  
end;
```

- 8) After the **case** block, post the **SeminarJnlLine** through the **Seminar Jnl-Post Line** codeunit.

```
SeminarJnlPostLine.RunWithCheck(SeminarJnlLine);
```

### Task 6: Complete the PostCharges Function

#### High Level Steps:

- 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**.

```
local procedure PostCharges();
begin
    SeminarCharge.reset;
    SeminarCharge.SetRange("Document No.",
                           SeminarRegHeader."No.");
    if SeminarCharge.FindSet(false,false) then repeat
        PostSeminarJnlLine(3); // Charge
        until SeminarCharge.next = 0;
    end;
```

## Task 7: Add Code to the OnRun Trigger

### High Level Steps:

- 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.
- 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.
- If there are no lines for the current document, throw an error.
- Open a dialog box to show the posting progress.
- 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.
- Assign the **SourceCode** variable from the **Seminar** field of the **Source Code Setup** table.
- 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.
- Update the dialog box.
- Copy the comment lines and charges from the registration header to the posted registration header, by calling the **CopyCommentLines** and **CopyCharges** functions.
- Set the **LineCount** variable to zero and prepare the loop for the registration lines of the current registration header.
- 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.
- 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.

## Solution Development in Visual Studio Code

---

- Delete the registration header, lines, comments, and charges.
- 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.

```
trigger OnRun();
begin
    ClearAll;
    SeminarRegHeader := Rec;
    with SeminarRegHeader do begin
    end;
    Rec := SeminarRegHeader;
end;
```

- 2) In the **with** block, make sure that the following fields are not empty and that the **Status** field value is *Closed*:

- a. Posting Date
- b. Document Date
- c. Seminar No.
- d. Duration
- e. Instructor Resource No.
- f. Room Resource No.

```
with SeminarRegHeader do begin
    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.

```

    SeminarRegLine.Reset;
    SeminarRegLine.SetRange("Document No.", "No.");
    if SeminarRegLine.IsEmpty then
        Error(Text001);

```

- 4) Open a dialog box to show the posting progress.

```

    Window.Open('#####' + Text002);
    Window.Update(1, StrSubstNo('%1 %2', Text003, "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.

```

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.

```

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.

```
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.

```
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.

```
CopyCommentLines(  
SeminarCommentLine."Table Name"::"Seminar Registration",  
SeminarCommentLine."Table Name"::"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.

```
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.

```

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.

```

// 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.

```
Delete(true);
```



Using the true parameter on the Delete command fires the OnDelete trigger on the Seminar Registration Header. Thereby all records for the seminar registration header in the tables:

Seminar Registration Line

Seminar Charge and

Seminar Comment Line

Will also be deleted

14) Add the following documentation to:

```
// CSD1.00 - 2018-01-01 - D. E. Veloper
```

```
// Chapter 7 - Lab 4-7
```

15) Save the file using **Ctrl+S**

## 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 Dynamics 365 BC, users must be able to start posting from the Document and the List pages for documents.

To meet the Dynamics 365 BC user experience standards, you must add the Post action to the Seminar Registration and Seminar Registration List pages.

## Task 8: Modify the Seminar Registration Pages

### High Level Steps:

- Add an action to the **Seminar Registration** page that runs the **Seminar-Post (Yes/No)** codeunit.
- Add an action to the **Seminar Registration List** page that runs the **Seminar-Post (Yes/No)** codeunit.

### Detailed Steps:

- 1) Locate the file containing the **Seminar Registration List** page
- 2) Locate the action section
- 3) Add a new action

```
action("&Post")
{
    Caption='&Post';
    Image=PostDocument;
    Promoted=true;
    PromotedIsBig=true;
    PromotedCategory=Process;
    ShortcutKey=F9;
    RunObject=codeunit "Seminar-Post (Yes/No)";
}
```

- 4) Add the following documentation to the file:

```
// Chapter 7 - Lab 4-8
// Added Action Post
```

- 5) Save the file using **Ctrl+S**
- 6) Locate the file containing the **Seminar Registration** page
- 7) Locate the action section

- 8) Add a new action Post as described above



Actions can be copied from one page to another by using **Ctrl+C** and **Ctrl+V**

- 9) Add the following documentation to the file:

```
// Chapter 7 - Lab 4-8  
// Added Action Post
```

- 10) Save the file using **Ctrl+S**

- 11) Next, make sure that all files are saved by clicking the **File/Save All** menu item



- 12) Then click the Source Control icon The number might be different depending on the last commit.

- 13) Enter the text **Lab 7.5** in the Message field



- 14) Then click the Commit checkmark

- 15) Lastly, Push the changes to the off-site repository

## Module Review

### Module Review and Takeaways

There are two types of posting routines in Dynamics 365 BC : 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 Dynamics 365 BC 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 Dynamics 365 BC 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 Dynamics 365 BC 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 Dynamics 365 BC 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 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

# Module 8: 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 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. Dynamics 365 BC 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. 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 is possible in the Development Environment, but it is not possible in extensions made in VS Code.
- Change any properties that only control how data is displayed or formatted.
- Change the **TableRelation** and **ValidateTableRelation** 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.



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

If one of the following changes is made to the table structure after it has been installed once, it is not possible to send the extension to Dynamics 365 BC:

- 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.

If you try to make a prohibited change, Dynamics NAV Development Environment warns you, and then prevents the change.

In order to handle structural changes to the tables, it is necessary to increase the version number in the app.json and run a Start-NavAppDataUpgrade in PowerShell or to back up the data in temporary tables and then publish the extension to the Dynamics 365 BC with the property  
**"schemaUpdateMode": "Recreate"**  
 In the launch.json file.



Adding the property **"schemaUpdateMode": "Recreate"**  
In the launch.json file will delete all data in all tables and extended fields in the extension.

## 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	Enables the Navigate feature to search for new ledger entry and posted document types.
Master Pages	<ul style="list-style-type: none"><li>• Create new documents from master pages.</li><li>• Access open documents from master pages.</li><li>• Access ledger entries from master pages.</li></ul>
Transactions	<ul style="list-style-type: none"><li>• Access Navigate from posted document pages.</li><li>• Access Navigate from ledger entries.</li></ul>

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 Dynamics 365 BC 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 Dynamics 365 BC, 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. Dynamics 365 BC standard functionality provides many examples of integrating features when you develop your custom application functionality.

The three most common user tasks in Dynamics 365 BC 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 Dynamics 365 BC 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.

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 Dynamics 365 BC 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 Dynamics 365 BC 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:

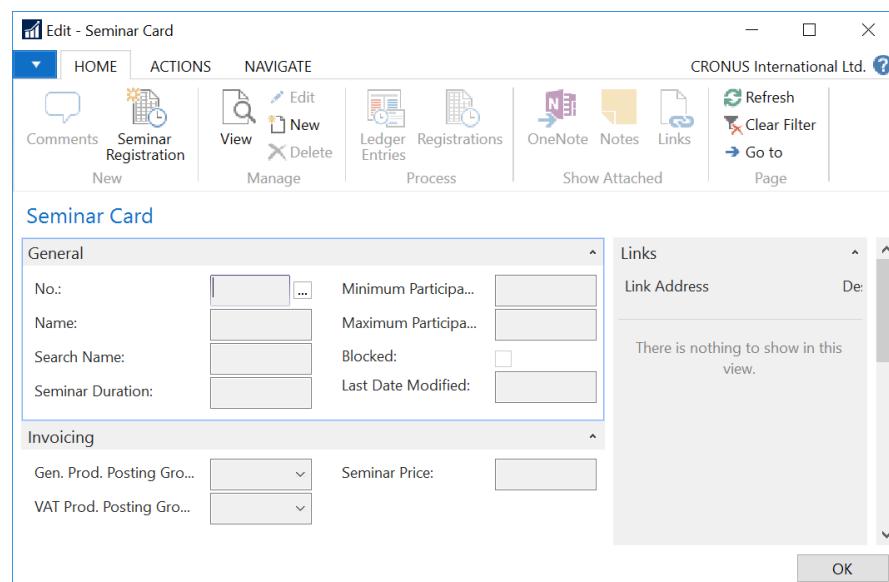
- Checks whether there is a filter on the Seminar No. field.
- If there is a filter on the Seminar No. field, the code checks whether the filter is to a single Seminar No. value.
- If the filter is to a single value, the code validates the Seminar No. field to the value in the filter on the Seminar No. 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)
  - New Document Items (container)
    - Seminar Registration (action)

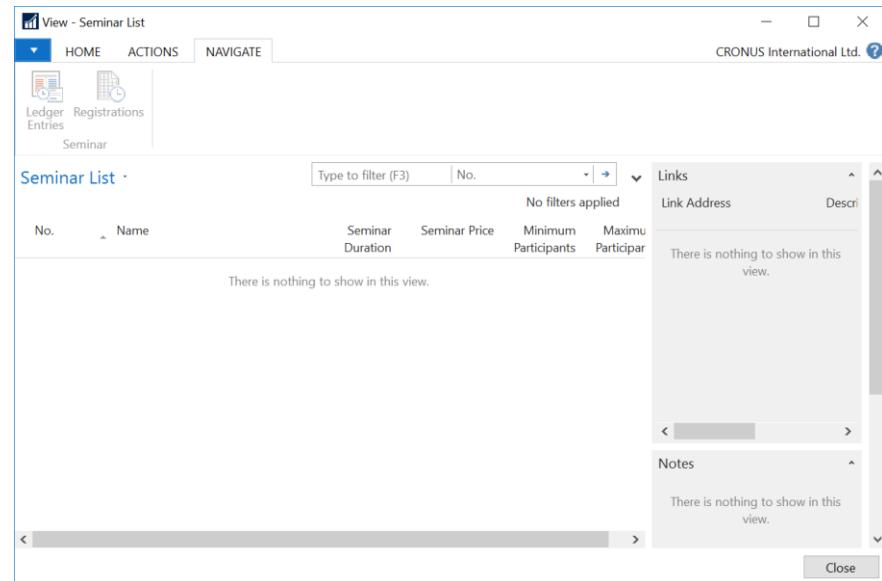
The **Seminar Card Page** shows the Seminar Card page after customization, with the Seminar Registration action in the New group on the Home tab.



The **Seminar List Page** shows the page after customization, with the **Ledger Entries** action and the **Registrations** action in the New group on the **Navigate** tab.

## Solution Development in Visual Studio Code

---



## Lab 8.1: Integrating Seminar Features

### Scenario

The developer team working on the implementation of Dynamics 365 BC for CRONUS International Ltd. is responsible for developing page customizations for Seminar Management integration.

They must integrate Seminar Management features to enable standard navigation between master data, documents, and posted information.

### Customize Seminar Registration Master Pages

#### Exercise Scenario

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, the Seminar table must also be customized. If there is a filter on the Seminar No. field, and the filter applies to a single Seminar No., the Seminar No. field validates to the value in the filter.

### Task 1: Customize the Seminar Registration Header Table

#### High Level Steps:

- Create code in the **Seminar Registration Header** table to apply the record link filter to the **Seminar No.** field.

#### Detailed Steps:

- 1) Locate the file containing the **Seminar Registration Header** table
- 2) Locate the **OnInsert** trigger and create a blank line just before the end of the trigger
- 3) Insert the following code in the trigger.

```
// >> Lab 8 1-1
if GetFilter("Seminar No.") <> '' then
    if GetRangeMin("Seminar No.") = GetRangeMax("Seminar No.")
        then
            Validate("Seminar No.",GetRangeMin("Seminar No."));
// << Lab 8 1-1
```

- 4) Save the file using **Ctrl+S**

### Task 2: Customize the Seminar Card Page

#### High Level Steps:

- Add an action to create a new Seminar Registration from the Seminar Card page.
- Add a new action as the first action in the Seminar group, to run the Seminar Ledger Entries page for the current Seminar record.
- Add a new group and an action to run the Seminar Registration List page for the current Seminar record.

#### Detailed Steps:

- 1) Locate the file containing the **Seminar Card** page
- 2) Locate the **actions** section and create a blank line just after **Comments** action
- 3) Add a new action for the **Ledger Entries** action:

```
// > Lab 8 1-2

action("Ledger Entries")
{
    Caption='Ledger Entries';

    RunObject=page "Seminar Ledger Entries";
    RunPageLink="Seminar No."=field("No.");
    Promoted=true;
    PromotedCategory=Process;
    ShortcutKey="Ctrl+F7";
    Image=WarrantyLedger;
}
```

- 4) Add another new action for the **Registrations**

```

// >> Lab 8 1-2

action("&Registrations")
{
    Caption='&Registrations';
    RunObject=page "Seminar Registration List";
    RunPageLink="Seminar No."=field("No.");
    Image=Timesheet;
    Promoted=true;
    PromotedCategory=Process;
}
// << Lab 8 1-2

```

- 5) Locate the end of the **Navigation** area and create a blank line outside the area
- 6) Create a new **Processing** area
- 7) Create a new **Seminar Registration** action in the new area

```

// >> Lab 8 1-2

area(Processing)
{
    action("Seminar Registration")
    {
        RunObject= page "Seminar Registration";
        RunPageLink="Seminar No."=field("No.");
        RunPageMode=Create;
        Image=NewTimesheet;
        Promoted=true;
        PromotedCategory=New;
    }
}
// << Lab 8 1-2

```

- 8) Save the file using **Ctrl+S**

### 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:

- 1) Locate the file containing the **Seminar Card** page
- 2) Copy all the actions made in Lab 8 1-2
- 3) Locate the file containing the **Seminar List** page
- 4) Locate the actions section
- 5) Paste the actions in in the same place as in Lab 8 1-2
- 6) Verify that it looks like this:

```
// >> Lab 8-2

action("Ledger Entries")
{
    RunObject=page "Seminar Ledger Entries";
    RunPageLink="Seminar No."=field("No.");
    Promoted=true;
    PromotedCategory=Process;
    ShortcutKey="Ctrl+F7";
    Image=WarrantyLedger;
}

action("&Registrations")
{
    RunObject=page "Seminar Registration List";
    RunPageLink="Seminar No."=field("No.");
    Image=Timesheet;
    Promoted=true;
    PromotedCategory=Process;
}

// << Lab 8-2
```

```
    }
}

// >> Lab 8-2

area(Processing)
{
    action("Seminar Registration")
    {
        RunObject= page "Seminar Registration";
        RunPageLink="Seminar No."=field("No.");
        RunPageMode=Create;
        Image=NewTimesheet;
        Promoted=true;
        PromotedCategory=New;
    }
}

// << Lab 8-2
```

- 7) Save the file using **Ctrl+S**

## 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 Dynamics 365 BC, and one of its most versatile features.

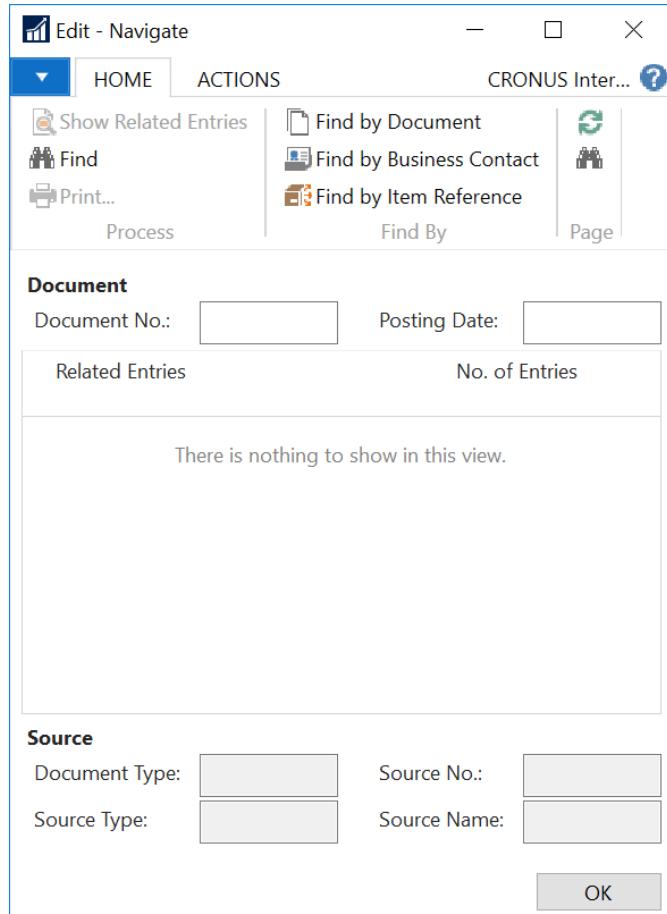
When users post a transaction, they rely on the Navigate feature when they must trace the results of the transaction. Users access the Navigate feature from any page that displays any type of posted entries or documents. The Navigate feature displays every resulting entry.

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**.



The page uses table 265, **Document Entry** as its source, and sets the property to Yes, to only work with temporary in-memory data.

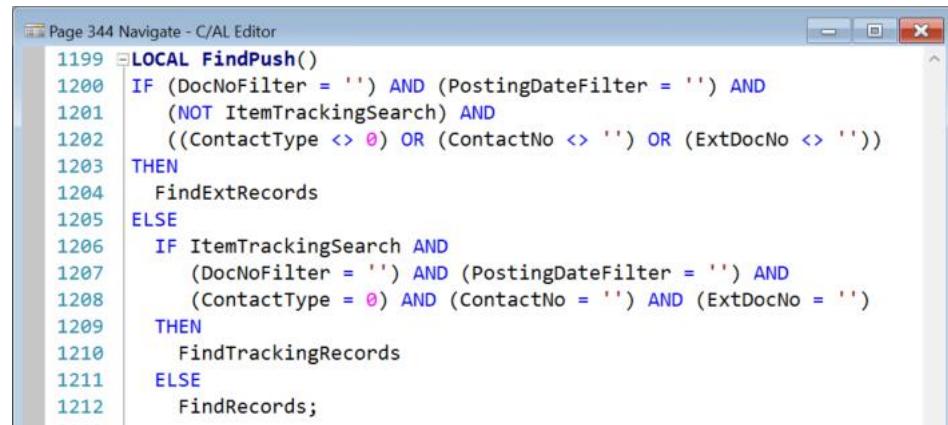
Table 265 Document Entry - Table Designer				
E...Field No.	Field Name	Data Type	Length	Description
1	Table ID	Integer		
2	No. of Records	Integer		
3	Document No.	Code	20	
4	Posting Date	Date		
5	Entry No.	Integer		
6	Table Name	Text	100	
7	No. of Records 2	Integer		
8	Document Type	Option		
9	Lot No. Filter	Code	20	
10	Serial No. Filter	Code	20	

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.



```
Page 344 Navigate - C/AL Editor
1199 LOCAL FindPush()
1200 IF (DocNoFilter = '') AND (PostingDateFilter = '') AND
1201     (NOT ItemTrackingSearch) AND
1202     ((ContactType <> 0) OR (ContactNo <> '') OR (ExtDocNo <> ''))
1203 THEN
1204     FindExtRecords
1205 ELSE
1206     IF ItemTrackingSearch AND
1207         (DocNoFilter = '') AND (PostingDateFilter = '') AND
1208         (ContactType = 0) AND (ContactNo = '') AND (ExtDocNo = '')
1209     THEN
1210         FindTrackingRecords
1211     ELSE
1212         FindRecords;
```

The **FindPush** function calls either the **FindExtRecords** or **FindRecords** function, depending on which filter fields the user populated with values.



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 Dynamics 365 BC, or by a user. There is no basic difference in the 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 SalesInvHeader.ReadPermission then begin

    SalesInvHeader.Reset;

    SalesInvHeader.SetFilter("No.",DocNoFilter);

    SalesInvHeader.SetFilter("Posting Date",

        PostingDateFilter);

    InsertIntoDocEntry(Database::"Sales Invoice Header",

        0,Text003,SalesInvHeader.Count);

end;

```

- 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.
  - Resets the appropriate transaction (posted document or ledger entry) table.
  - Sets the appropriate table key to make the search easier.
  - 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.
  - 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.

### 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.

```
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 Dynamics 365 BC 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 extending 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.

### Codeunit

The **Navigate** page has been designed for this and it includes Integration Events in both the **FindRecords** and **ShowRecords** procedures. The Integration event in the **FindRecords** procedure is **OnAfterNavigateFindRecords** and in **ShowRecords** it is **OnAfterNavigateShowRecords**. The change can therefore be done by creating a codeunit with **Integration Subscriptions** for both procedures.

### 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 8.2: Changing Objects to Integrate with Navigate

### Scenario

After integrating Seminar Management features the standard application functionality features must be integrated. The most important standard feature is Navigate. Enable **Navigate** to search for **Seminar Management** transaction records and make **Navigate** available in **Seminar Management** pages.

### Customize Tables

#### Exercise Scenario

To enable Navigate to search for Seminar Management transactions in the most efficient way, add a new key to the Seminar Ledger Entry table.

### Task 1: Modify the Seminar Ledger Entry Table

#### High Level Steps:

- Add the key with Document No. and Posting Date fields to the Seminar Ledger Entry table.

#### Detailed Steps:

- 1) Locate the file containing the **Seminar Ledger Entry** table
- 2) Locate the **keys** section and create a blank line just after the **key1** section
- 3) Add the following key:

```
key(key2;"Document No.", "Posting Date")
{
}
```

- 4) Add the necessary documentation

```
// Chapter 8 - Lab 2-1
// Added key2
```

- 5) Save the file using **Ctrl+S**

### Customize the Navigate Page

#### Exercise Scenario

Create a codeunit with an **Integration Subscription** so the Navigate page so that it can search for **Posted Seminar Reg. Header** and **Seminar Ledger Entry** records.

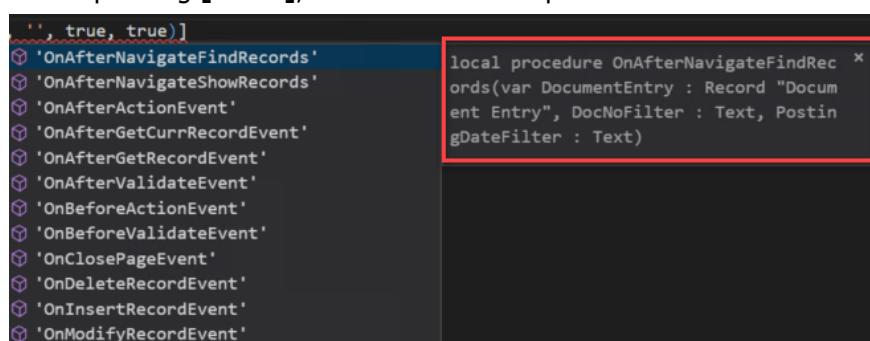
### Task 2: Create the Event Subscriptions

#### High Level Steps:

- Locate the **39\_EventSubscriptions.al** file in the codeunits folder
- Create an **EventSubscriber** for **FindRecords** using the Integration Event **OnAfterNavigateFindRecords** in the **Navigate** page (344)
- Create a local procedure **ExtendNavigateOnAfterNavigateFindRecords**
- Add code to the procedure to enable Navigate to find records from both the **Seminar Ledger Entries** and the **Posted Seminar Reg. Header** tables
- Create an **EventSubscriber** for **ShowRecords** using the Integration Event **OnAfterNavigateShowRecords**
- Create a local procedure **ExtendNavigateOnAfterNavigateShowRecords**
- Add code to the procedure to enable Navigate to show records from both the **Seminar Ledger Entries** and the **Posted Seminar Reg. Header** tables

#### Detailed Steps:

- 1) Locate the **39\_EventSubscriptions.al** file in the codeunits folder
- 2) Create an **EventSubscriber** for **FindRecords** using the Integration Event **OnAfterNavigateFindRecords** in the **Navigate** page (344) using **tiven** and selecting the **tevents sub** snippet just before the end of the file
- 3) Replace **ObjectType** with **Page**
- 4) Replace **ObjectId** with **344**
- 5) Delete 'OnSomeEvent' and click **Ctrl+[Space]** Select **OnAfterNavigateFindRecords**
- 6) Before pressing **[Enter]**, make a note of the parameters for the event.



- 7) Press **[Enter]** to select
- 8) Replace '**ElementName**' with ''
- 9) Replace **SkipOnMissingLicense** with **true**
- 10) Replace **SkipOnMissingPermission** with **true**
- 11) Create a local procedure **ExtendNavigateOnAfterNavigateFindRecords** using the **tpro** snippet

12) Add the parameters to the procedure

```
var DocumentEntry : Record "Document Entry"
DocNoFilter : Text
PostingDateFilter : Text
```

13) Create local variables:

```
SeminarLedgerEntry : record "CSD Seminar Ledger Entry"
PostedSeminarRegHeader : record "CSD Posted Seminar Reg.
Header"
```

14) Add code to the procedure to enable Navigate to find records from the **Posted Seminar Reg. Header** table

```
[EventSubscriber(ObjectType::Page, 344,
'OnAfterNavigateFindRecords', '', true, true)

local procedure ExtendNavigateOnAfterNavigateFindRecords
(var DocumentEntry : Record "Document Entry";
DocNoFilter : Text;
PostingDateFilter : Text);

var
SeminarLedgerEntry : record "CSD Seminar Ledger Entry";
PostedSeminarRegHeader : record "CSD Posted Seminar Reg.
Header";
DocNoOfRecords : Integer;
NextEntryNo : Integer;

begin
if PostedSeminarRegHeader.ReadPermission then begin
PostedSeminarRegHeader.Reset;
PostedSeminarRegHeader.SetFilter("No.",DocNoFilter);
PostedSeminarRegHeader.SetFilter("Posting Date",
PostingDateFilter);
DocNoOfRecords:= PostedSeminarRegHeader.Count;

```

```

With DocumentEntry do begin
    if DocNoOfRecords = 0 then
        exit;
    if FindLast then
        NextEntryNo := "Entry No." + 1
    else
        NextEntryNo := 1;
    Init;
    "Entry No." := NextEntryNo;
    "Table ID" := Database::"Posted Seminar Reg. Header";
    "Document Type" := 0;
    "Table Name" := COPYSTR(PostedSeminarRegHeader .
        TableCaption, 1, MAXSTRLEN("Table Name"));
    "No. of Records" := DocNoOfRecords;
    Insert;
end;
end;
end;

```

15) Add the similar code to the procedure to enable **Navigate** to find records from the **Seminar Ledger Entry** table by copying the previous code, replacing the `PostedSeminarRegHeader` variable with `SeminarLedgerEntry` and replacing the `SetFilter("No.")` with `SetFilter("Document No.")`.

16) Create an event subscription for **ShowRecords** using the Integration Event **OnAfterNavigateShowRecords**. Make a note of the needed parameters:

<ul style="list-style-type: none"> <li>⌚ 'OnAfterNavigateShowRecords'</li> <li>⌚ 'OnAfterActionEvent'</li> <li>⌚ 'OnAfterGetCurrRecordEvent'</li> <li>⌚ 'OnAfterGetRecordEvent'</li> <li>⌚ 'OnAfterValidateEvent'</li> </ul>	<pre>local procedure OnAfterNavigateShowRecords(TableID : Integer, DocNoFilter : Text, PostingDateFilter : Text, ItemTrackingSearch : Boolean)</pre>
--	--

17) Create a local procedure **ExtendNavigateOnAfterNavigateShowRecords** on the **Navigate** page (344) using the same procedure as above.

18) Add code to the procedure to enable **Navigate** to show records from both the **Seminar Ledger Entries** and the **Posted Seminar Reg. Header** tables

```
[EventSubscriber(ObjectType::Page,
344,'OnAfterNavigateShowRecords' , '' , true, true)]
local procedure ExtendNavigateOnAfterNavigateShowRecords
(TableID : Integer;
DocNoFilter : Text;
PostingDateFilter : Text;
ItemTrackingSearch : Boolean);
var
SeminarLedgerEntry : record "CSD Seminar Ledger Entry";
PostedSeminarRegHeader : record "CSD Posted Seminar Reg.
Header";
begin
case TableID of
Database::"Posted Seminar Reg. Header":
Page.Run(0,PostedSeminarRegHeader);
Database::"Seminar Ledger Entry":
Page.Run(0,SeminarLedgerEntry);
end;
end;
```



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.

### Task 3: Define Default Lookup and Drilldown Pages for Tables

#### High Level Steps:

- Define the default lookup and drilldown pages for the Posted Seminar Reg. Header table.
- Define the default lookup and drilldown pages for the Seminar Ledger Entries table.

#### Detailed Steps:

- 1) Locate the file containing the **Posted Seminar Reg. Header** table
- 2) Add the following property

```
Caption = Posted Seminar Reg. Header;  
LookupPageId="Posted Seminar Reg. List";  
DrillDownPageId="Posted Seminar Reg. List";
```

- 3) Add the necessary documentation

```
// Chapter 8 - Lab 2 - 3  
// Added LookupPageId & DrilldownPageId properties
```

- 4) Locate the file containing the **Seminar Ledger Entry** table

- 5) Add the following property

```
Caption = 'Seminar Ledger Entry';  
LookupPageId="Seminar Ledger Entries";  
DrillDownPageId="Seminar Ledger Entries";
```

- 6) Add the necessary documentation

```
// Chapter 8 - Lab 2 - 3  
// Added LookupPageId & DrilldownPageId properties
```

- 7) Save the file using **Ctrl+S**

## Customize Pages

### Exercise Scenario

Finally, it is necessary to change the relevant Seminar Management pages, by adding the Navigate action to run the Navigate page to each of them.

## Task 4: Customize the Posted Seminar Registration Page and the Seminar Ledger Entry Page

### High Level Steps:

- Add the **Navigate** action to the **Processing** area on the **Posted Seminar Registration** page.
- 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.
- Repeat the process for the **Seminar Ledger Entries** page

### Detailed Steps:

- 1) Locate the file containing the **Posted Seminar Registration** page
- 2) Locate the **actions** section
- 3) Add a new area: **area(Processing)**
- 4) In the new area create a new action: "**&Navigate**"
- 5) In the new action section, create a new **OnAction** trigger
- 6) Create a local variable **Navigate : page Navigate**
- 7) Add code to the trigger to set the date filter with the function on **Navigate SetDoc**
- 8) Run the **Navigate** page

```

action("&Navigate")
{
    Caption='&Navigate';
    Image=Navigate;
    Promoted=true;
    PromotedCategory=Process;
    trigger OnAction();
    var
        Navigate : page Navigate;
    begin
        Navigate.SetDoc("Posting Date", "No.");
        Navigate.RUN;
    end;
}

```

- 9) Add the necessary documentation

```
// Chapter 8 - Lab 2 - 4  
// Added Action Navigate
```



Repeat the process for the **Seminar Ledger Entries** page only using the Document No. field instead of the No. Field

- 18) Lastly, make sure that all files are saved by clicking the **File/Save All** menu item

- 19) Then click the Source Control icon  The number might be different depending on the last commit.

- 20) Enter the text **Lab 8.2** in the Message field

- 21) Then click the Commit checkmark 

- 22) Lastly, Push the changes to the off-site repository

## 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 Dynamics 365 BC. Users frequently depend on this feature to search for specific transactions or analyze the results of a single transaction.

# Module 9: 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

## 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 Dynamics 365 BC, 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

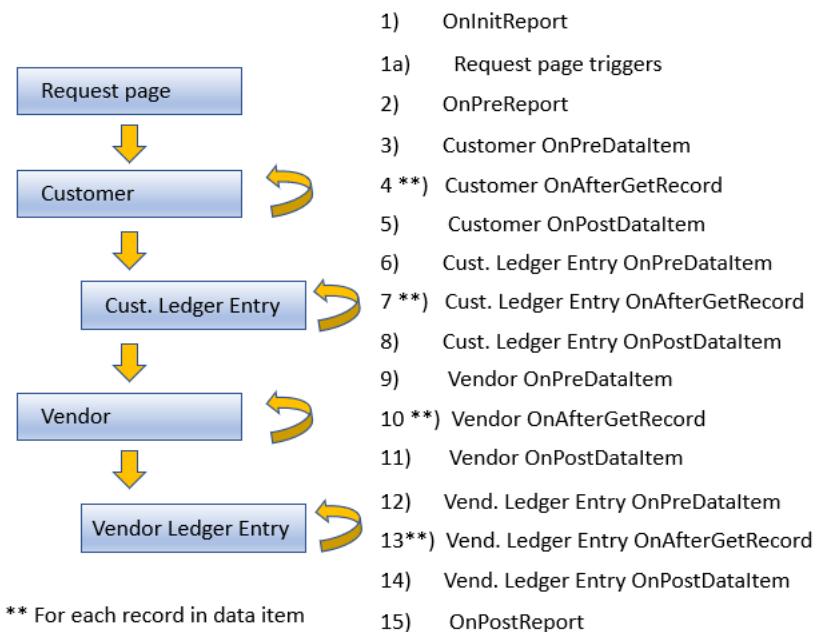
Each report object consists of several elements that can contain the following triggers:

- 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.
- 2) If the **OnInitReport** trigger does not end the processing of the report, the **Request** page for the report runs. 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
- 10) 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.



### 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 Dynamics 365 BC 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 allowed to be printed in preview mode.



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 any Dynamics 365 BC client. This makes sure that any functionality that depends on the **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 **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** button remains unchanged.

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.



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 Participants 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.

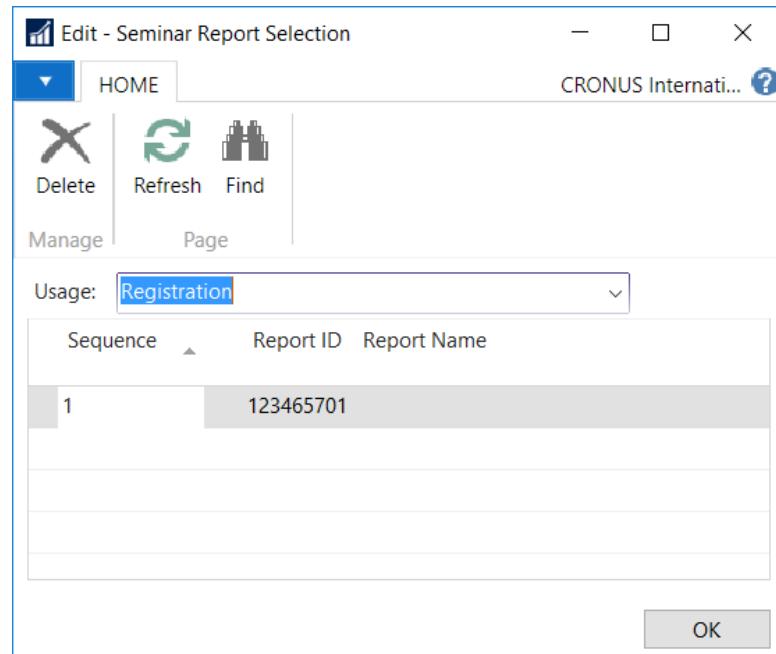
### Solution Design

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

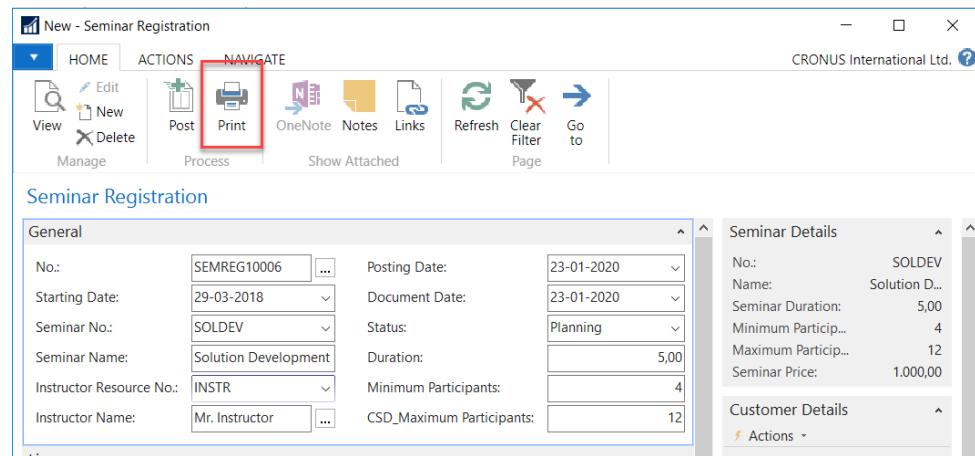
The Seminar Report Selection page displays the available seminar reports.



Add the Seminar Reg. List Participant List report.

Add the Seminar Report Selection page to the departments menu.

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** for an example.



## 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 9.1: 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.

### Part A: The Report Dataset

#### Task 1: Add a field to the Seminar Registration Header table

##### Prerequisite to this lab:

In order to complete this lab, it is necessary to create Seminars and Seminar Registration Headers and lines for testing.

##### High Level Steps:

Add a field to the **CSD Seminar Registration Header** table (123456710).

Field No.	Field Name	Data Type	
40	No. Printed	Integer	Not Editable

##### Detailed Steps:

###### Alter the Seminar Registration Table

- 5) Locate the file containing the Seminar Registration Table
- 6) Add the field at the end of the field list:

```
field(40;"No. Printed";Integer)
{
    Caption='No. Printed';
    Editable=false;
}
```

- 7) Add the necessary documentation

```
// Chapter 9 - Lab 1-1
// - Added new field "No. Printed"
```

- 4) Save the file using **Ctrl+S**

### Task 2: Create a codeunit to increment the No. Printed field just added to the Seminar Registration Header table

#### High Level Steps:

- Create a new file **02\_SeminarRegPrinted.al** in the Codeunits folder
- Create the Seminar Registration-Printed codeunit (123456702).

#### Detailed Steps:

- 1) Create a new file **02\_SeminarRegPrinted.al** in the Codeunits folder
- 2) Create a new codeunit using the **tcodeunit** snippet
- 3) Replace **Id** with **123456702** and **MyCodeunit** with **CSD SeminarRegPrinted**
- 4) Set the property to specify the **CSD Seminar Registration Header** table as the source table for this codeunit.
- 5) In the **OnRun** procedure, add the following code:

```
codeunit 123456702 "CSD SeminarRegPrinted"  
{  
    TableNo="CSD Seminar Reg. Header";  
  
    trigger OnRun();  
    begin  
        Find;  
        "No. Printed" += 1;  
        Modify;  
        Commit;  
    end;  
}
```

- 6) Add the necessary documentation

```
//  Chapter 9 - Lab 1-2  
//      - Added Codeunit
```

- 7) Save the file using **Ctrl+S**

## Task 3: Create the actual dataset in Report Dataset Designer

### High Level Steps:

- Create a file **SeminarRegParticipantList.al** in the in the Reports folder.
- Create a new report 123456701 **SeminarRegParticipantList**
- Add a data item for the **CSD Seminar Registration Header** table.
- Add sorting and filtering to the **CSD Seminar Registration Header** data item.
- Add the fields below to the Seminar Registration Line data item.
- Add a data item for the **CSD Seminar Registration Line** table.
- Add the fields below to the **CSD Seminar Registration Line** data item.
- Use the property of the columns so that captions are available for the fields of both data items.
- Set the property for the report so that the caption is **Seminar Reg.- Participant List**.
- 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.
- Add a **Label** to the Report.
- Publish the Extension to Dynamics 365 BC using **F5**

### Detailed Steps:

1. Create a file **01\_SeminarRegParticipantList.al** in the in the Reports folder.
2. Create a new report using the **treport** snippet
3. Replace **Id** with **123456701** and **MyReport** with "**CSD SeminarRegParticipantList**"
4. Set the **UsageCategory** property to be **ReportsAndAnalysis**
5. Locate the first dataitem and replace **DataItemName** with **CSD Seminar Registration Header**
6. Replace **SourceTableName** with **CSD Seminar Registration Header**
7. Add sorting and filtering to the **CSD Seminar Registration Header** data item.

```

dataitem(SeminarRegistrationHeader;
         "Seminar Registration Header")
{
    DataItemTableView=sorting("No.");
    RequestFilterFields="No.", "Seminar No.";
}

```

8. Add the fields below to the **CSD Seminar Registration Header** data item.

- a. No.
  - b. Seminar No.
  - c. Seminar Name
  - d. Starting Date
  - e. Duration
  - f. Instructor Name
  - g. Room Name
9. Use the property of the columns so that captions are available for the fields of both data items.

```
column(No_; "No.")  
{  
    IncludeCaption=true;  
}
```

10. Add a data item for the **Seminar Registration Line** table after the last column but still within the **SeminarRegistrationHeader** dataitem.
11. Set the **DataItemTableView** property so that the data item sorts by Document No. and Line No.
12. Set 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

```
column(Room_Name; "Room Name")  
{  
    IncludeCaption=true;  
}  
dataitem(SeminarRegistrationLine;  
    "Seminar Registration Line")  
{  
    DataItemTableView=sorting("Document No.", "Line No.");  
    DataItemLink="Document No.=field("No.");  
}  
}
```

13. Add the fields below to the **Seminar Registration Line** data item
- a. Bill-to Customer No.
  - b. Participant Contact No.

- c. Participant Name
14. Add a dataitem after the dataitem containing the **SeminarRegistrationHeader** table
  15. Link the dataitem to the **Company Information** table
  16. Add the **Name** column to the data item as shown below

```
dataitem("Company Information";"Company Information")  
{  
    column(Company_Name;Name)  
}  
}
```

17. Set the property for the report so that the caption is **Seminar Reg.- Participant List**.
18. Add the property **DefaultLayout** to be **RDLC**
19. Set the **UsageCategory** property to be **ReportsAndAnalysis**
20. Add a Label to the Report.

```
labels  
{  
    SeminarRegistrationHeaderCap =  
        'Seminar Registration List'  
}
```

21. Save the file using **Ctrl+S**
22. Publish the Extension to Dynamics 365 BC using **F5**

### Part B: The Report Layout

#### Exercise Scenario

To render a report from inside the RoleTailored client, you must create a (RDLC) report layout.

### Task 4: Use Microsoft SQL Server Report Builder to create a layout for the report

#### High Level Steps:

- Open the Dynamics 365 BC Windows client.
- Search for **Custom Report Layout**
- Add a new line for the report **Seminar Reg.- Participant List**
- Click **Edit Layout**.
- Add a table to the body of the report.
- Add fields to the table.
- Add group header rows to the table.
- Add fields to the group header.
- Format the table.

#### Detailed Steps:

##### Open SQL Server Report Builder.

- 1) Open the Dynamics 365 BC Windows client.
- 2) Search for **Custom Report Layout**
- 3) Add a new line for the report **Seminar Reg.- Participant List**
- 4) Click **Edit Layout**.
- 5) Maximize the SQL Server Report Builder
- 6) Add a table to the body of the report by selecting the **Insert** tab and click **Table**.
- 7) Select the **Insert Table** control and click the body to insert it onto the body of the report.
- 8) Position the table in the upper-left corner in the body of the report.
- 9) In the **Report Data** window, open the **Parameters** folder.
- 10) From the **Parameters** folder, select the field **Bill\_to\_Customer\_No\_Caption**, and then drag it onto the table on the text box on the first row, first column (in the Header Row).
- 11) Repeat this for the following fields:
  - a. **Participant\_Contact\_No\_Caption**
  - b. **Participant\_NameCaption**
- 12) The Header row in the table now contains the caption fields for the fields from the **Seminar Registration Line** table.
- 13) In the second row of the table, in the first text box, click the field selector, and then select the following field from the shortcut menu:  
**Bill\_to\_Customer\_No**
- 14) Repeat this for the following fields:
  - a. **Participant\_Contact\_No**

b. **Participant\_Name**

- 15) In the Row Groups window at the bottom of the screen, select **Details** > **Add Group** > **Parent Group**.
- 16) In the Tablix group window, select **No\_** in the Group By field and select the **Add Header** option.
- 17) Delete the first column of the table.
- 18) Right-click the detail row of the table, and then select **Insert Row, Outside Group - Above**.
- 19) Repeat this step until there are seven empty group header rows in the table.
- 20) In the first row of the table, select the three text boxes.
- 21) Right-click, and then click **Cut**.
- 22) On the first text box on the header row just above the **Details** row, right-click, and then click **Paste**.
- 23) Delete the top row
- 24) The table should look like below

	[@Bill_to_Custome]	[@Participant_Con]
[@Bill to Customer]	[@Participant Contact]	[@Participant Name]

- 25) In the first column of the table, in the empty group header rows, add the following Parameters:
  - a. No\_Caption
  - b. SeminarNo\_Caption
  - c. SeminarName\_Caption
  - d. StartingDate\_Caption
  - e. Duration\_Caption
  - f. InstructorName\_Caption
  - g. RoomName\_Caption
- 26) In the second column of the table, in the empty group header rows add the following fields:
  - a. No\_
  - b. SeminarNo
  - c. SeminarName
  - d. StartingDate
  - e. Duration

- f. InstructorName
- g. RoomName

27) 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.

28) Add a filter to the table to exclude blank lines:

- a. Click the tablix, then right-click the left-margin of the table
- b. Select Tablix Properties
- c. Select Filters and Add a new filter
- d. Click the Expression button



- e. Add the following line in the expression

**=Len(Fields!Seminar\_No\_.Value)>0**

- f. Change **Text** to **Boolean**, the **operator** to **=** and the **Value** to **true**



- g. Then click **OK**

## Task 5: Add a Page Header

### High Level Steps:

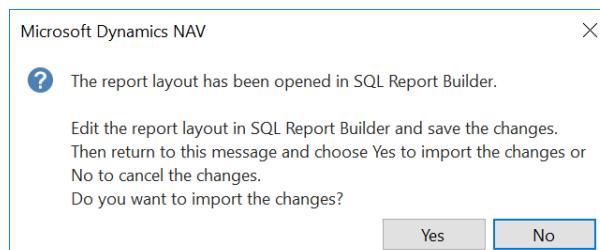
- Add a page header to the report.
- Format the title.

### Detailed Steps:

- 1) Add a page header to the report
- 2) In the Page Header, add four textboxes: one on the left side and three on the right side.
- 3) In the text box on the left side of the page header drag the **SeminarRegistrationHeaderCap** parameter in to the field
- 4) In the three text boxes on the right side, add the following expressions:
  - a. Globals!ExecutionTime
  - b. Globals!PageNumber & " " of " " & Globals!TotalPages
  - c. User!UserID

Format the title

- 5) In the text box on the Page Header that contains the title, set the Color to blue, and then make the text Bold.
- 6) Lastly, drag the Company\_Name field from the dataset and position it under the blue title
- 7) Go to the expression of the field and change  
`=First(Fields!Company_Name.Value, "DataSet_Result")`  
 To  
`=Last(Fields!Company_Name.Value, "DataSet_Result")`
- 8) Close and save the report
- 9) At the confirmation dialog below, click yes to import the layout to the **Custom Report Layout**



- 10) Then click the **Export Layout** action. This will export the .rdl file.
- 11) Save the file to the **Layouts** folder in the VS Code workspace as **SeminarRegParticipantList.rdl**.
- 12) After the export, delete the Custom Report Layout for the **Seminar Reg.- Participant List** report



It is necessary to include the layout in the workspace because it will not be an integrated part of the extension and thereby automatically be included on publishing and installing the extension to another database.

- 13) Lastly, return to the **SeminarRegParticipantList.al** fil in VS Code.
- 14) Add the report property:  
**RDLCLayout='./Layouts/SeminarRegParticipantList.rdl';**
- 15) Save the file using **Ctrl+S**

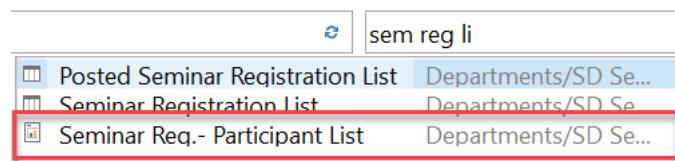
## Task 6: Run the Report

## High Level Steps:

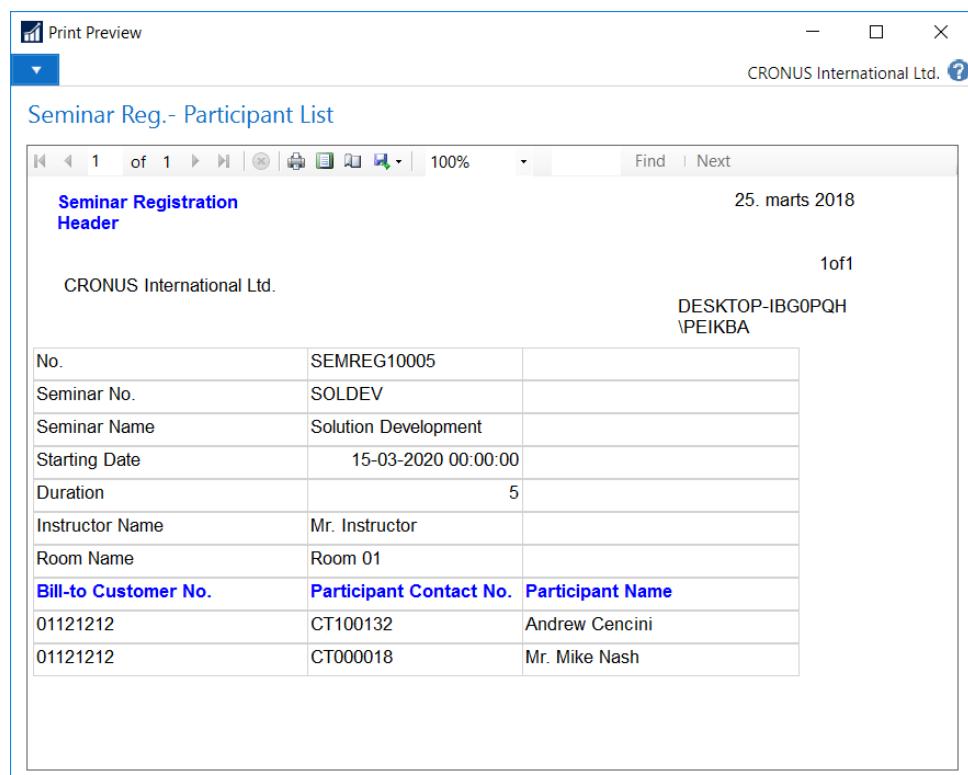
- Publish the extension
  - Start the Dynamics 365 BC Windows client
  - Search for the **Seminar Reg.- Participant List** report
  - Run it to confirm that the layout looks as shown below

## Detailed Steps:

- 1) Publish the extension using **Ctrl+F5**
  - 2) Start the Dynamics 365 BC Windows client



- 3) **4) Search for the Seminar Reg.- Participant List report**  
5) Run it to confirm that the layout looks as shown below



### **Part C: Report Selections Table and Page**

#### **Task 7: Import the Seminar Report Selections Table and Page**

##### **High Level Steps:**

- Import the report selection objects from the Mod09 Starter A folder and place them in the relevant folders

##### **Detailed Steps:**

- 1) Import all the .al files from the desktop and place in the workspace
- 2) Open the **Solution Development Course Objects\Mod09\Starter A\Tables** folder on the Desktop in a File Explorer
- 3) Mark all files and drag them into the Tables folder of the project
- 4) Open the **Solution Development Course Objects\Mod09\Starter A\Pages** folder on the Desktop in a File Explorer
- 5) Mark all files and drag them into the Pages folder of the project
- 6) Open the **Solution Development Course Objects\Mod09\Starter A\Codeunits** folder on the Desktop in a File Explorer
- 7) Mark all files and drag them into the Codeunits folder of the project

## Task 8: Add the print action to the Seminar Registration Page

### High Level Steps:

- Add the Print action to the Seminar Registration Pages calling the **PrintReportSelection** procedure from the **Seminar Report Selections** table

### Detailed Steps:

- 1) Locate the file containing the **Seminar Registration** page
- 2) Add a new **Print** action in the **Seminar Registration** group in the **navigation** area
- 3) Verify that the action looks like this:

```

action("&Print")
{
    Caption = '&Print';
    Image = Print;
    Promoted = true;
    PromotedIsBig = true;
    PromotedCategory = Process;
    trigger OnAction();
    var
        SeminarReportSelection: Record
        "CSD Seminar Report Selections";
    begin
        SeminarReportSelection.PrintReportSelection
            (SeminarReportSelection.Usage::Registration, Rec);
    end;
}

```

- 4) Save the file using **Ctrl+S**
- 5) Repeat the above for the **Seminar Registration List** page

### Part D: Testing

#### Task 9: Test Report Selections

##### High Level Steps:

- Build the extension and publish it to the server
- Test the application

##### Detailed Steps:

- 6) Build the extension and publish it to the server using **Ctrl+F5**
- 7) If there are no Seminar Registration records, create one by selecting **SD Seminars > Registrations** from the **Departments** menu. Complete all fields in the General and Seminar Rooms FastTabs and add at least two participants.
- 8) 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.
- 9) Open the Seminar Registration page again, and then view the record prepared in Step 1. Click Print.
- 10) The Request page should open with the No. set to the current registration. Select Print or Preview to print and check the report.
- 11) Try to print the report by using different parameters and different registration data to verify that it works correctly.

## Lab 9.2: Review the Invoice Posting Batch Job

### Scenario

As soon as a seminar concludes, the registrations must be posted in preparation for billing. A *ProcessingOnly* report has been created to generate Dynamics NAV standard format invoices that are ready for billing and posting.

### Part A: Review the Invoice Posting Batch Job

#### Task 1: Import the Create Invoices report object

##### High Level Steps:

- Import the report object from the **Mod09 Starter B** folder and place them in the relevant folders

##### Detailed Steps:

- 1) Import all the .al files from the desktop and place in the workspace
- 2) Open the **Solution Development Course Objects\Mod09\Starter B\Reports** folder on the Desktop in a File Explorer
- 3) Mark all files and drag them into the Tables folder of the project

#### Task 2: Review the Create Invoices report

##### High Level Steps:

- In Explorer window, locate report **Create Seminar Invoices**, and then Review the AL code as described below.
- Verify the property to specify that this report is for processing- only
- Verify which global variables have been defined for the report.
- Verify which global text constants are defined.
- Verify the fields included in the Request page
- Read and explain the code in the OnOpenPage of the Request page
- Read and explain the code in the **FinalizeSalesInvHeader** procedure.
- Read and explain the code in the **InsertSalesInvHeader** procedure.
- Read and explain the code in the **OnPreDataItem** trigger
- Read and explain the code in the **OnAfterGetRecord** trigger
- Read and explain the code in the **OnPostDataItem** trigger

##### Detailed Steps:

- 1) In Explorer window, locate report Create Seminar Invoices, and then Review the AL code as described below.
- 2) Verify the property to specify that this report is for processing- only

```
report 123456700 "CSD Create Seminar Invoices"
{
    // CSD1.00 - 2018-01-01 - D. E. Velper
    // Chapter 9 - Lab 2
    // - Created new report
```

- 3) Verify which global variables have been defined for the report.

```
var
    CurrencyExchRate : Record "Currency Exchange Rate";
    Customer : Record Customer;
    GLSetup : Record "General Ledger Setup";
    SalesHeader : Record "Sales Header";
    SalesLine : Record "Sales Line";
    SalesSetup : Record "Sales & Receivables Setup";
    SalesCalcDiscount : Codeunit "Sales-Calc. Discount";
    SalesPost : Codeunit "Sales-Post";
    CalcInvoiceDiscount : Boolean;
    PostInvoices : Boolean;
    NextLineNo : Integer;
    NoofSalesInvErrors : Integer;
    NoofSalesInv : Integer;
    PostingDateReq : Date;
    docDateReq : Date;
    Window : Dialog;
    Seminar : Record "CSD Seminar";
```

- 4) Verify which global text constants are defined.

```
var

    Text000 : Label 'Please enter the posting date.';

    Text001 : Label 'Please enter the document date.';

    Text002 : Label 'Creating Seminar Invoices...\\';

    Text003 : Label 'Customer No.      #1#####\\';

    Text004 : Label 'Registration No.  #2#####\\';

    Text005 : Label 'The number of invoice(s) created is

                    %1.';

    Text006 : Label 'not all the invoices were posted. A

                    total of %1 invoices were not posted.';

    Text007 : Label 'There is nothing to invoice.';
```

- 5) Verify the fields included in the Request page

```
area(content)

{

    group(Options)

    {

        Caption = 'Options';

        field(PostingDateReq;PostingDateReq)

        {

            Caption = 'Posting Date';

        }

        field(docDateReq;docDateReq)

        {

            Caption = 'document Date';

        }

    }

}
```

```
    field(CalcInvoiceDiscount;CalcInvoiceDiscount)
    {
        Caption = 'Calc. Inv. Discount';
    }
    field(PostInvoices;PostInvoices)
    {
        Caption = 'Post Invoices';
    }
```

- 6) Read and explain the code in the OnOpenPage of the Request page

```
trigger OnOpenPage();
begin
    if PostingDateReq = 0D then
        PostingDateReq := WorkDate;
    if docDateReq = 0D then
        docDateReq := WorkDate;
    SalesSetup.Get;
    CalcInvoiceDiscount := SalesSetup."Calc. Inv.
    Discount";
end;
```

- Set the **Posting Date** and the **Document Date** to be the working date
  - Get the **SalesSetup** record
  - Set the **CalculateInvoiceDiscount** variable from the sales setup
- 7) Read and explain the code in the FinalizeSalesInvHeader procedure.

```
local procedure FinalizeSalesInvoiceHeader();
begin
  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;
```

- Calculate the Invoice Discount in ant
- Update the **NoOfSalesInv** counter
- Post the invoice if requested from the request page

- 8) Read and explain the code in the InsertSalesInvHeader procedure.

```
local procedure InsertSalesInvoiceHeader();
begin
    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;
    end;
    NextLineNo := 10000;
end;
end;
```

- Initialize the SalesHeader variable
- Set the Document Type to select the correct number series
- Insert(true) to fetch the next number from the number series
- Validate the Bill-to Customer No, the posting date, the document date and the currency code.
- Modify and commit the record
- Set the NextLineNo for the lines

9) Read and explain the code in the OnPreDataItem trigger

```
trigger OnPreDataItem();
begin
    if PostingDateReq = 0D then
        ERROR(Text000);
    if docDateReq = 0D then
        ERROR(Text001);
    Window.Open(
        Text002 +
        Text003 +
        Text004);
end;
```

- Throw and error if the posting date or the document date from the request page is empty

10) Read and explain the code in the OnAfterGetRecord trigger

```
trigger OnAfterGetRecord();
begin
    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.");
        end;
    end;
end;
```

```
if SalesHeader."No." <> '' then
    FinalizeSalesInvoiceHeader;
    InsertSalesInvoiceHeader;
end;
Window.Update(2,"Seminar Registration No.");
```

- If the customer no has changed, get the new customer rec
- If the new customer is blocked, add it to the error counter
- Update the dialog
- If the sales header no is not empty then finalize the previous sales header
- Then initialize a new sales header
- Update the dialog again

11) The OnAfterGetRecord trigger continued

```
case Type of
    Type::Resource:
        begin
            SalesLine.Type := SalesLine.Type::Resource;
        end;
    "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;
```

- Fill the No. on the sales line depending on the charge type
- 12) The OnAfterGetRecord trigger continued

```

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;

```

- Fill out the primary key from the sales header and set the next line no
- Validate the No. field with the value set in the previous section
- Set the description either to the description from the Seminar Ledger Entry or from the seminar name

13) The OnAfterGetRecord trigger continued

```

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;

```

- Set the **Unit Price** from the **Seminar Ledger Entry**

- Test if the **Currency factor** is set if the **Currency Code** is filled
- Calculate the currency **Unit Price**
- Validate the **Quantity** from the **Seminar Ledger Entry**
- Insert the sales line
- Update the **NextLineNo**

14) Read and explain the code in the OnPostDataItem trigger

```
trigger OnPostDataItem();
begin
    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;
end;
```

- Close the dialog and give a message if there is nothing to post
- Finalize the Sales Invoice header and give a message to user

## **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.

# Module 10: Role Tailoring

## Module Overview

In Dynamics 365 BC, 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. Dynamics 365 BC has more than 20 Role Centers. Each Role Center is represented by a page object.

When you develop custom application areas for Dynamics 365 BC, 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

Dynamics 365 BC 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.

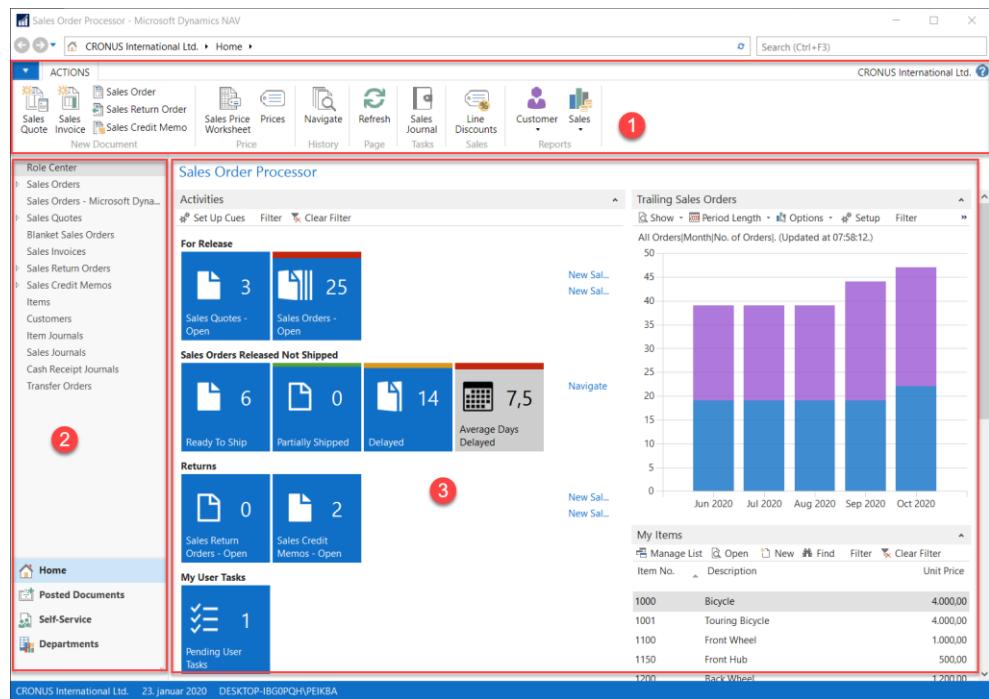
Research shows that the greatest challenges to users are frequent interruptions and balancing multiple concurrent tasks. Dynamics 365 BC helps resolve this issue with a RoleTailored design that is combined with a familiar and easy-to-use interface. Instead of mapping to the structure of the database, 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.

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.

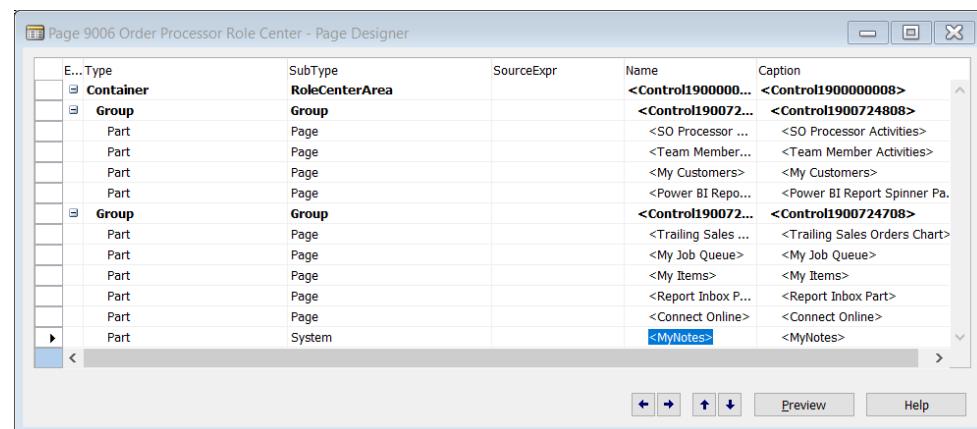
# Solution Development in Visual Studio Code



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.

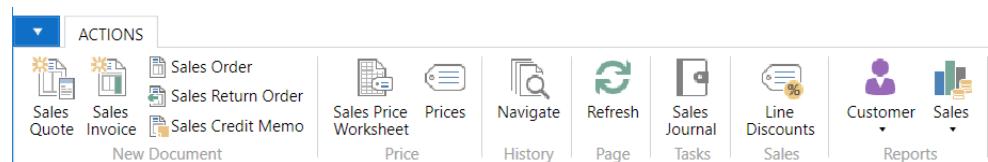


You cannot write any trigger code on pages of type RoleCenter. This applies not only to page triggers, such as OnOpenPage or OnClosePage, but also to action triggers. Any action that is defined in a Role Center page may only use the RunObject property. Any code in the triggers prevents you from compiling or saving the page.

## Actions and Ribbon

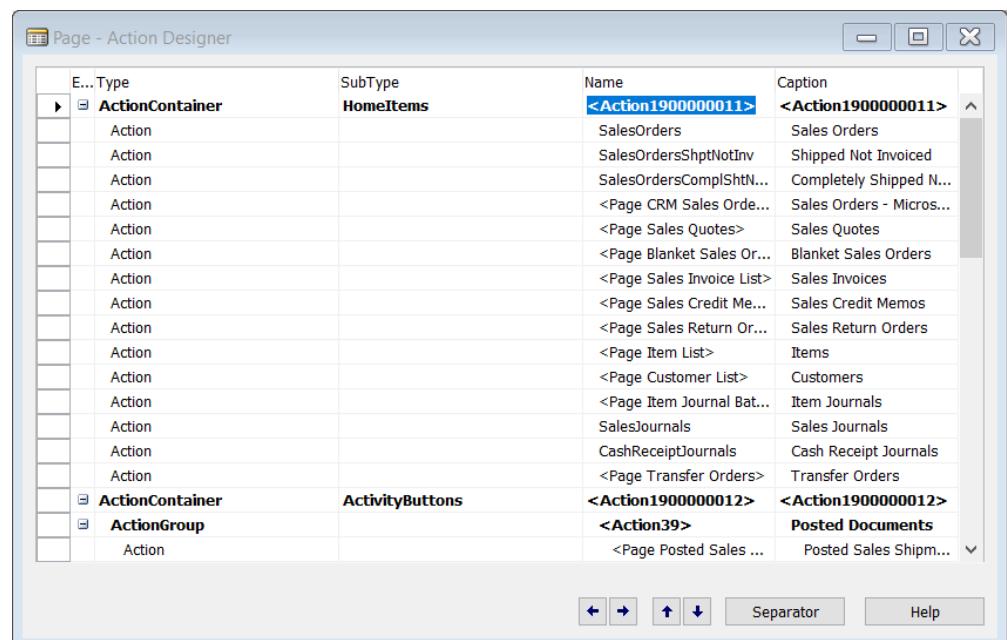
Actions let users access features of an application, such as a List Place, or run a function, such as posting an invoice. The ribbon organizes action types into tabs and groups for quick and easy access by users. The ribbon is available from most page types.

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.



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	<b>Actions</b> tab, New Document group
ActionItems	<b>Actions</b> tab
RelatedInformation	<b>Navigate</b> tab
Reports	<b>Report</b> tab

HomeItems	<b>Home</b> menu of the <b>Navigation</b> pane
ActivityButtons	Other menus of the <b>Navigation</b> pane, such as <b>Posted Documents</b>

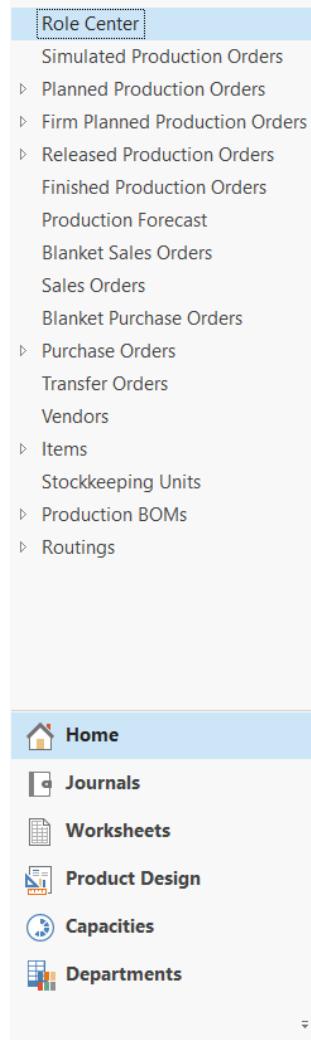


If an action container of type *ActionItems*, *RelatedInformation* or *Reports* contains action groups, these groups appear as separate group in the **Actions**, **Navigate** and **Report** tabs. Any actions that do not belong to an action group and are defined at the action container level, appear in 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 has been assigned 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 menu items. 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.



## 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.

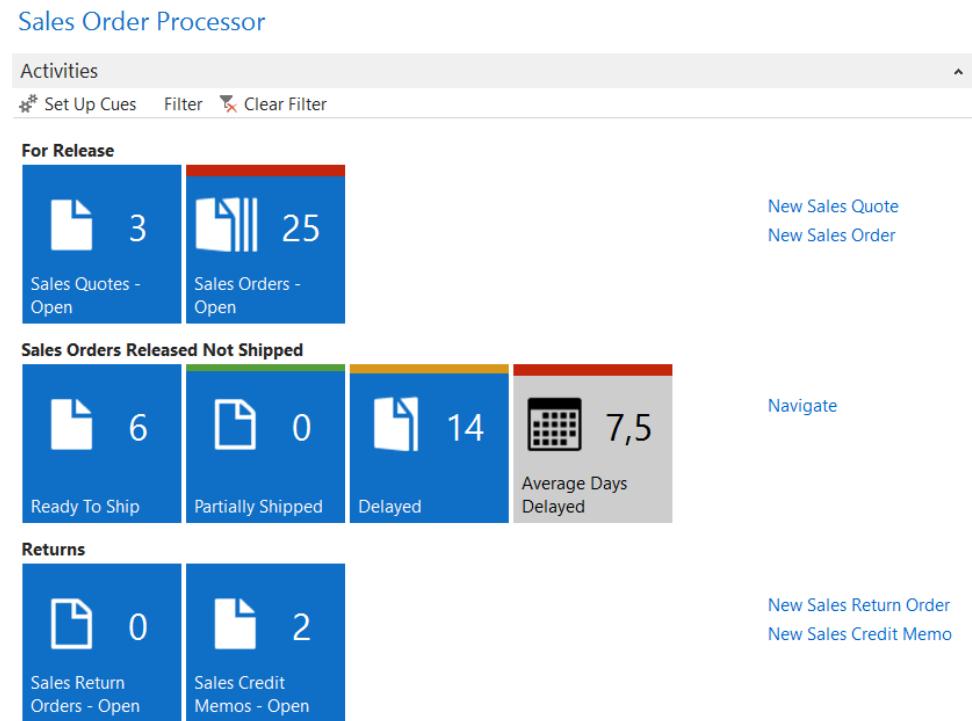
The Activities part is defined as a *CardPart* page and uses groups of type *CueGroup* to show the cues. Every cue is defined as a field in the source table of the Activities page.

**CueGroup** is the only control type in Dynamics 365 BC 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

## Solution Development in Visual Studio Code

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.



A field must be of type *Decimal* or *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.

## My 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.

My Customers		
<span>Manage List</span> <span>Open</span> <span>New</span> <span>Find</span> <span>Filter</span> <span>Clear Filter</span>		
C..	Name N.	Pho... No.
2...	Selangorian Ltd.	96.049,99
3...	John Haddock In...	349.615,40

My Items		
<span>Manage List</span> <span>Open</span> <span>New</span> <span>Find</span> <span>Filter</span> <span>Clear Filter</span>		
Item No.	Description	Unit Price
1000	Bicycle	4.000,00
1001	Touring Bicycle	4.000,00
1100	Front Wheel	1.000,00
1150	Front Hub	500,00
1200	Back Wheel	1.200,00

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.

The entries in the My Customers can be used as filters by assigning the term:

**%MyCustomers**

In the No filter of any list, report, query, XMLport or Chart.

## Solution Development in Visual Studio Code

Customers

Show results:

Where No. is %MyCustomers

Limit totals to:

Where Date Filter is 01-01-20..31-12-20

No.	Name	Responsibility Center	Location Code	Phone No.	Contact
01445544	Progressive Home Furnishings		YELLOW		Mr. Scott Mitchell
01454545	New Concepts Furniture		YELLOW		Ms. Tammy L. McDonald
10000	The Cannon Group PLC	BIRMINGHAM	BLUE		Mr. Andy Teal
20000	Selangorian Ltd.				Mr. Mark McArthur
30000	John Haddock Insurance Co.				Miss Patricia Doyle

Leaving the No. filter field shows how the term is changed into the primary key from all the entries in the My Customers table separated by the sign |.

Customers

Show results:

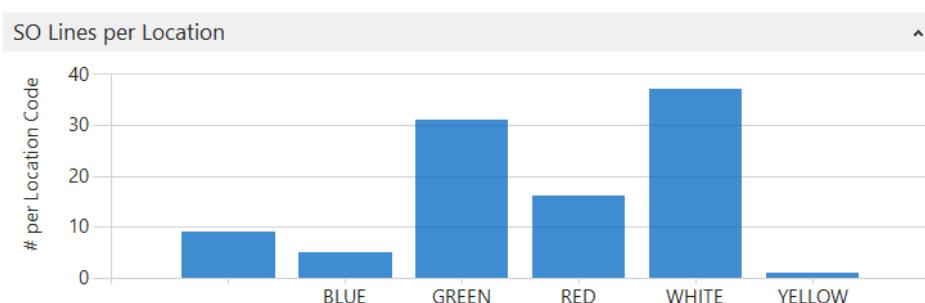
Where No. is 01445544|01454545|10000|

Add Filter

## Charts

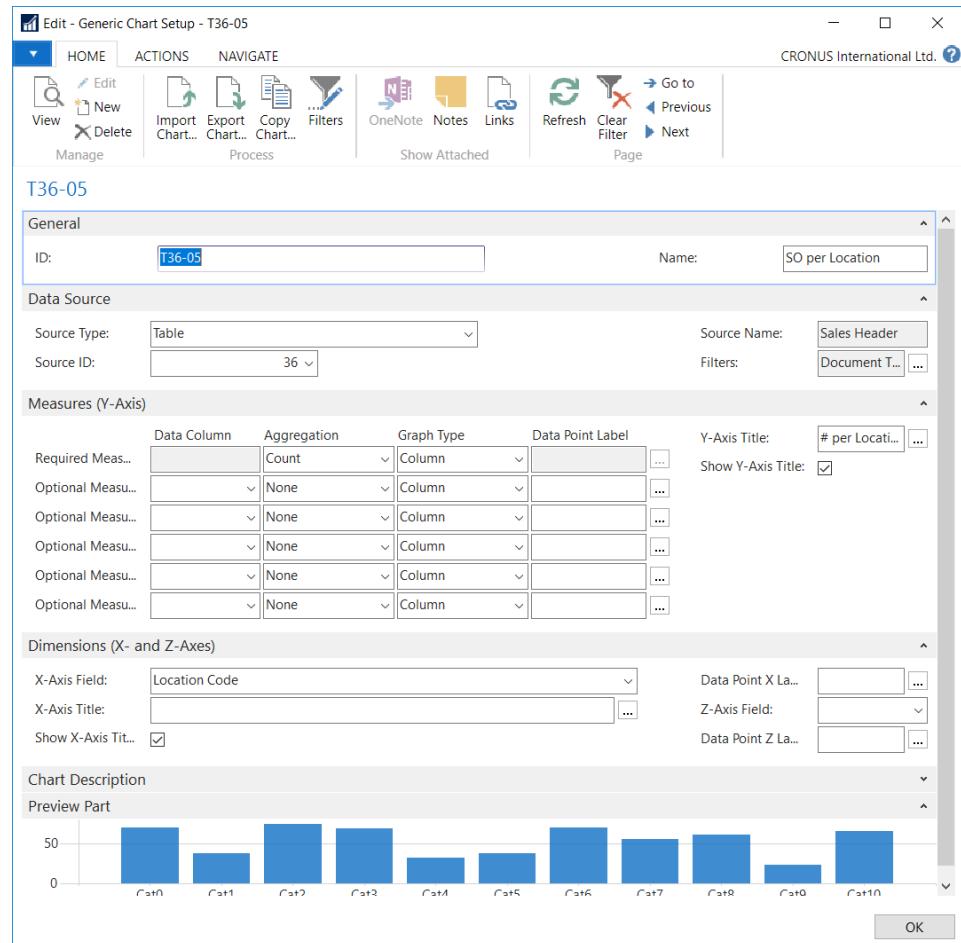
Charts are graphical representations of business data combinations that you can view in list places, FactBoxes, and Role Centers. A Role Center may contain one or more charts that display information that is relevant to a profile.

Below is shown a chart that users can add to their Role Centers.



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.

Below is shown the SO per Location chart in the Generic Chart 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**

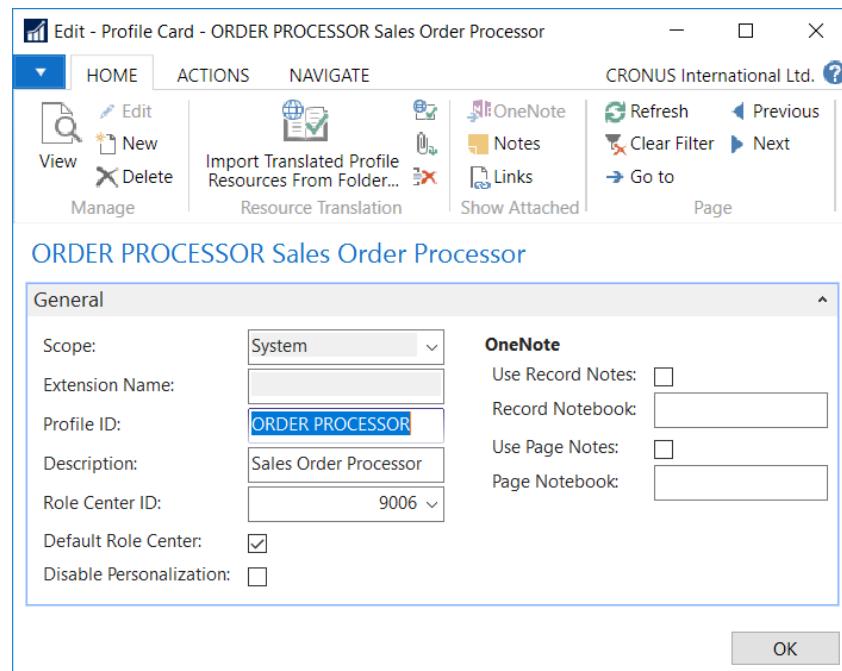
**> RoleTailored Client > Profiles.**

## Solution Development in Visual Studio Code

Use the Profiles page for an overview of all profiles.

Profiles		Type to filter (F3)	Profile ID			
Profile ID	Description	Scope	Extension Name	Role Center ID	Def... Role...	P...
ACCOUNTANT	Accountant	System		9027	<input type="checkbox"/>	
ACCOUNTING MANAGER	Accounting Manager	System		9001	<input type="checkbox"/>	
ACCOUNTING SERVICES	Outsourced Accounting Manager	System		9023	<input type="checkbox"/>	
AP COORDINATOR	Accounts Payable Coordinator	System		9002	<input type="checkbox"/>	
AR ADMINISTRATOR	Accounts Receivable Administrator	System		9003	<input type="checkbox"/>	
BOOKKEEPER	Bookkeeper	System		9004	<input type="checkbox"/>	
BUSINESS MANAGER	Business Manager	System		9022	<input type="checkbox"/>	
DISPATCHER	Dispatcher - Customer Service	System		9016	<input type="checkbox"/>	
IT MANAGER	IT Manager	System		9018	<input type="checkbox"/>	
MACHINE OPERATOR	Machine Operator - Manufacturing Comprehensive	System		9013	<input type="checkbox"/>	
ORDER PROCESSOR	Sales Order Processor	System		9006	<input checked="" type="checkbox"/>	
ORDER PROCESSOR - NC	Sales Order Processor - Non-configured	System		9006	<input type="checkbox"/>	

Use the **Profile Card** page to create new profiles or change existing ones.



Edit - Profile Card - ORDER PROCESSOR Sales Order Processor

HOME ACTIONS NAVIGATE CRONUS International Ltd. ?

View Edit New Delete Import Translated Profile Resources From Folder... Resource Translation OneNote Notes Links Show Attached Refresh Previous Clear Filter Next Go to Page

ORDER PROCESSOR Sales Order Processor

General

Scope:	System
Extension Name:	
Profile ID:	ORDER PROCESSOR
Description:	Sales Order Processor
Role Center ID:	9006
Default Role Center:	<input checked="" type="checkbox"/>
Disable Personalization:	<input type="checkbox"/>

OneNote

Use Record Notes:   
Record Notebook:   
Use Page Notes:   
Page Notebook:

OK

## 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 Dynamics 365 BC. 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.

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 CSD Seminar Cue	Contains the flow fields for the cues on the Activities page part of the <b>Role</b>

	<b>Center</b> page.
123456741 CSD My Seminar	Contains the list of seminars that each user has included in the <b>My Seminars</b> list.

### Pages

You must create the following pages:

Page	Remarks
123456740 CSD Seminar Manager Role Center	The <b>Role Center</b> page.
123456741 CSD Seminar Manager Activities	The Activities page part for the <b>Role Center</b> page.
123456742 CSD My Seminars	A list part for the <b>Role Center</b> page that lets users manage their list of favorite seminars.

Below is shown the Seminar Manager Role Center with all its components.

The screenshot shows the Seminar Manager RoleCenter in Microsoft Dynamics NAV. The interface is organized into several sections:

- Top Navigation:** Includes the company name "CRONUS International Ltd.", a "Home" link, a search bar, and a "Role Center" button.
- Toolbar:** Features a "New Document" button and a "General" button.
- Main Content Area:**
  - Seminar Manager Activities:** Displays "Registrations" (1 Planned, 0 Registered) and "For Posting" (0 Closed).
  - My Customers:** A list of customers with their names and balance (LCY). The list includes:
 

Name	Balance (LCY)
0. Progressiv...	1.499,03
0. New Conc...	222.241,32
1. The Cann...	168.364,41
2. Selangoria...	96.049,99
3. John Hadd...	349.615,40
  - My Notifications:** A list of notifications from "DESKTOP-IBG0POQH\PEL" dated "20-04-2020" indicating successful completion of tasks.
  - My Seminars:** A list of seminars with columns for Name, Duration, and Price. It notes "There is nothing to show in this view."
  - Report Inbox:** A list of reports with columns for Show, Unread Reports, All Reports, Delete, Show Queue, Created Date-Time, Description, and Output Type. It notes "There is nothing to show in this view."
- Bottom Navigation:** Includes links for "Home", "Posted Documents", and "Departments".

The status bar at the bottom shows "CRONUS International Ltd. 23. januar 2020 DESKTOP-IBG0POQH\PEIKBA".

# Lab 10.1: Create the Seminar Manager Role Center

## Scenario

As a part of the development team who works on the implementation project of Dynamics 365 BC at 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 Dynamics 365 BC Role Center page.

## 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.

Field No.	Field Name	Data Type	Length	Property
10	Primary Key	Code	10	
20	Planned	Integer		CalcFormula=Count("Seminar Registration Header" where(Status=const(Planning)))
30	Registered	Integer		CalcFormula=Count("Seminar Registration Header" where(Status=const(Registration)))
40	Closed	Integer		CalcFormula=Count("Seminar Registration Header" where(Status=const(Closed)))

### Detailed Steps:

#### Create the Seminar Setup Table

- 13) In the VS Code Explorer window, click the **Table** folder, then right-click and select the **New File** menu item
- 14) Name the file **40\_SeminarCue.al** and press **[Enter]**
- 15) Enter **tt** in the first line and select the **ttable** snippet
- 16) Enter **123456740** as the **ID** and “**CSD Seminar Cue**” as the **name** and press **enter**
- 17) Add the following property:
  - a. **Caption = 'Seminar Cue'**
- 18) Add the fields described above in the fields section by:
  - a. Type **tfie** and select the **tfields** snippet
  - b. Enter the field number then press the **Tab** key
  - c. Type the name and then press the **Tab** key
  - d. Type the field type. Enter the length surrounded by **[]** on code and text fields. then press the **Tab** key
  - e. Replace the **FieldPropertyName = FieldPropertyValue;** line with properties for **Caption**
  - f. If needed add the additional property
- 19) Locate the **keys** section and replace **MyField** with **Primary Key**
- 20) Delete the **var** section
- 21) Delete all triggers
- 22) Update the documentation
- 23) Save the file using **Ctrl+S**
- 24) Verify that the solution looks like this

```
table 123456740 "CSD Seminar Cue"  
// CSD1.00 - 2018-01-01 - D. E. Veloper  
// Chapter 10 - Lab 1 - 1  
// - Created new page  
{  
    DataClassification = ToBeClassified;  
    Caption='Seminar Cue';  
  
    fields  
    {  
        field(10;"Primary Key";Code[10])  
        {  
            DataClassification = ToBeClassified;  
        }  
    }  
}
```



```

        field(20;Planned;Integer)
        {
            Caption = 'Planned';
            FieldClass=FlowField;
            CalcFormula=Count("Seminar Registration Header"
                where(Status=const(Planning)));
        }

        field(30;Registered;Integer)
        {
            Caption = 'Registered';
            FieldClass=FlowField;
            CalcFormula=Count("Seminar Registration Header"
                where(Status=const(Registration)));
        }

        field(40;Closed;Integer)
        {
            Caption = 'Closed';
            FieldClass=FlowField;
            CalcFormula=Count("Seminar Registration Header"
                where(Status=const(Closed)));
        }

    }

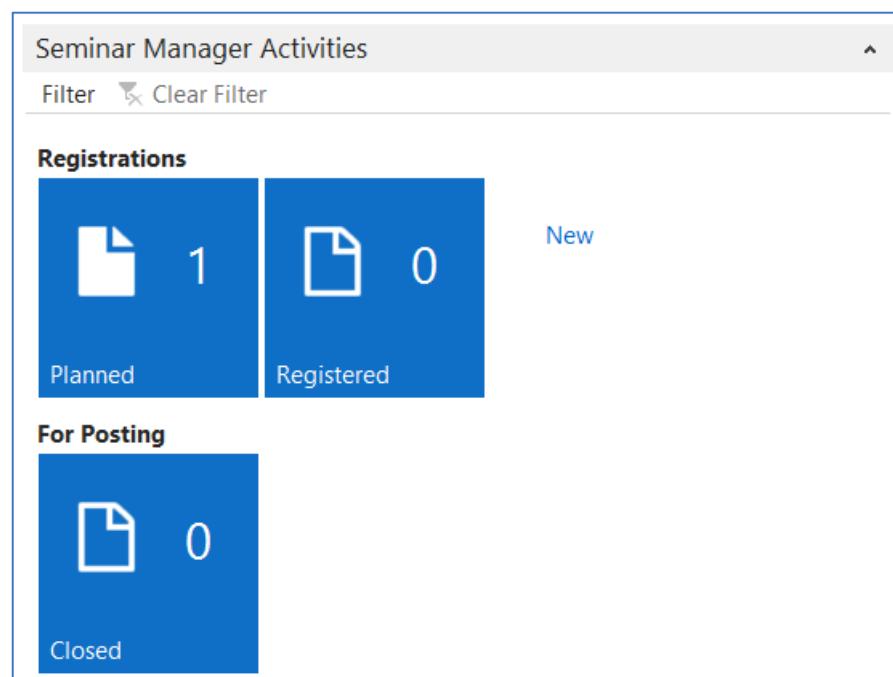
    keys
    {
        key(PK;"Primary Key")
        {
            Clustered = true;
        }
    }
}

```

### Task 2: Create the Seminar Manager Activities Page

#### High Level Steps:

Create the **CSD Seminar Manager Activities** page as shown below and assign the **ID** of **123456740** to it.



- Set the property to be *Card Part*
- Set properties so it is not possible to insert and delete records
- Set properties so the SourceTable is the **CSD Seminar Cue** table
- Add code to the **OnOpenPage** trigger to insert a new record, if there is not a record in the Seminar Setup table.
- Create two cue groups (Registrations, "For Posting")
- Add all FlowFields to the page
- Add an action to the Registrations cue group to create a new registration header
- Compile, save, and then close the page.

#### Detailed steps:

- 26) In the VS Code Explorer window, click the **Page** folder, then right-click and select the **New File** menu item
- 27) Name the file **40\_SeminarManagerActivities.al** and press **[Enter]**
- 28) Enter **tpa** in the first line and select the **tpage, Page of type card** snippet
- 29) Enter **123456740** as the **ID** and **CSD Seminar Manager Activities** as the **name** and press enter
- 30) Change the **PageType** to be **CardPart**
- 31) Set the Source table to be **CSD Seminar Cue**
- 32) Add a **Caption** property
- 33) Add the **Editable** property to be **false**
- 34) Change the **group** to be **cuegroup**

- 35) Change the **cuegroup** name to be **Registrations**
- 36) Add the following fields to the page in the Numbering group:
  - a. Planned
  - b. Registered
- 37) Add an **actions** section inside the cue group section
- 38) Add an **action** with name New in the **actions** section
- 39) Add a new **cuegroup** with name **For Posting**
- 40) Add the following fields to the page in the Numbering group:
  - a. Closed
- 41) Go to the end of the code and create a new line just before the last }
- 42) Type **ttr** and select the **ttrigger** snippet
- 43) Replace the **OnWhat** with **OnOpenPage and** add the following code to the trigger:

```
trigger OnOpenPage();  
  
begin  
    if not get then begin  
        init;  
        insert;  
    end;  
end;
```

- 44) Update the documentation
- 45) Save the file using **Ctrl+S**
- 46) Verify that the solution looks like this:

```
page 123456740 "Seminar Manager Activities"
// CSD1.00 - 2018-01-01 - D. E. Veloper
// Chapter 10 - Lab 1 - 2
// - Created new page
{
    PageType = CardPart;
    SourceTable = "CSD Seminar Cue";
    Caption='Seminar Manager Activities';

    layout
    {
        area(content)
        {
            cuegroup(Registrations)
            {
                Caption='Registrations';
                field(Planned;Planned)
                {
                }
                field(Registered;Registered)
                {
                }
            }
            actions
            {
                action(New)
                {
                    Caption='New';
                    RunObject=page "Seminar Registration";
                    RunPageMode=Create;
                }
            }
        }
    }
}
```

```
}

cuegroup("For Posting")
{
    field(Closed;Closed)
    {
    }
}

trigger OnOpenPage();
begin
    if not get then begin
        init;
        insert;
    end;
end;
}
```

### My Seminars table and 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 3: Create the My Seminar Table

#### High Level Steps:

1. Create the My Seminar table.

Field No.	Field Name	Data Type	Length	Table Relation
10	User ID	Code	50	User
20	Seminar No.	Code	20	Seminar

2. Define table relations on fields.
3. Define the appropriate primary key for the My Seminar table.

#### Detailed Steps:

##### Create the My Seminar Table

- 25) In the VS Code Explorer window, click the **Table** folder, then right-click and select the **New File** menu item
- 26) Name the file **41\_MySeminar.al** and press **[Enter]**
- 27) Enter **tt** in the first line and select the **ttable** snippet
- 28) Enter **123456741** as the **ID** and “**CSD My Seminar**” as the **name** and press **enter**
- 29) Add the following property:
  - Caption = ‘My Seminar’**
- 30) Add the fields described above in the fields section by:
  - Type **tfie** and select the **tfields** snippet
  - Enter the field number then press the **Tab** key
  - Type the name and then press the **Tab** key
  - Type the field type. Enter the length surrounded by [] on code and text fields. then press the **Tab** key
  - Replace the **FieldPropertyName = FieldPropertyValue;** line with properties for **Caption**
  - If needed add the additional property
- 31) Locate the **keys** section and replace **MyField** with **User ID,Seminar No.**
- 32) Delete the **var** section
- 33) Delete all triggers and update the documentation
- 34) Save the file using **Ctrl+S**
- 35) Verify that the solution looks like this

```
table 12346741 "My Seminars"

// CSD1.00 - 2018-01-01 - D. E. Veloper
// Chapter 10 - Lab 1 - 3
// - Created new page
{

    DataClassification = ToBeClassified;

    Caption='My Seminars';

}

fields

{

    field(10;"User Id";Code[50])
    {
        Caption = 'User Id';
        TableRelation=User;
        DataClassification = ToBeClassified;
    }

    field(20;"Seminar No. ";Code[20])
    {
        Caption = 'Seminar No.';
        TableRelation=Seminar;
        DataClassification = ToBeClassified;
    }

}

keys

{

    key(PK;"User Id","Seminar No.")

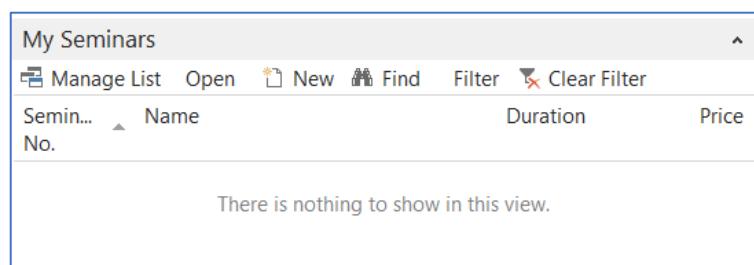
}

}
```

### Task 4: Create the My Seminar Page

#### High Level Steps:

Create the **CSD My Seminar** page as shown below and assign the **ID** of **123456741** to it.



- Set the property to be *List Part*
- Set properties so the **SourceTable** is the **My Seminars** table
- Add only **The Seminar No.** field to the page
- Add code to the **OnOpenPage** trigger to only show records from the actual user
- Then create a global variable for the Seminar record
- Add code to the **OnAfterGetRecord** trigger to get the Seminar Record
- Add the fields Name, Duration and price from the seminar record to the page
- Compile, save, and then close the page.

#### Detailed steps:

- 1) In the VS Code Explorer window, click the **Page** folder, then right-click and select the **New File** menu item
- 2) Name the file **41\_MySeminars.al** and press **[Enter]**
- 3) Enter **tpa** in the first line and select the **tpage, Page of type list** snippet
- 4) Enter **123456741** as the **ID** and **CSD My Seminars** as the **name** and press enter
- 5) Change the **PageType** to be **ListPart**
- 6) Set the Source table to be **CSD My Seminars**
- 7) Add a **Caption** property
- 8) Add the following field to the page in the **Repeater** group:
  - a. Seminar No.
- 9) Go to the end of the code and create a new line just before the last **}**
- 10) Create a **var** section for global variables
- 11) Create a global variable to the **CSD Seminar** record
- 12) Go to the end of the code and create a new line just before the last **}**
- 13) Type **ttr** and select the **ttrigger** snippet
- 14) Replace the **OnWhat** with **OnOpenPage** and add the following code to the trigger:

```

trigger OnOpenPage();
begin
    SetRange("User Id",UserId);
end;

```

- 15) Type **ttr** and select the **ttrigger** snippet  
 16) Replace the **OnWhat** with **OnAfterGetRecord** and add the following code to the trigger:

```

trigger OnAfterGetRecord();
begin
    if Seminar.get("Seminar No.") then;
end;

```

- 17) Type **ttr** and select the **ttrigger** snippet  
 18) Replace the **OnWhat** with **OnNewRecord** and add the following code to the trigger:

```

trigger OnNewRecord(BelowxRec : Boolean);
begin
    Clear(Seminar);
end;

```

- 19) Type **tpr** and select the **tprocedure** snippet  
 20) Replace the **MyProcedure** with **OpenSeminarCard** and add the following code to the trigger:

```

local procedure OpenSeminarCard();
begin
    if Seminar."No."<>'' then
        Page.Run(Page::"Seminar Card",Seminar );
end;

```

- 21) Return to the **repeater** section and add the following fields to the page  
 a. Seminar.Name  
 b. Seminar.Duration  
 c. Seminar.Price  
 22) Locate the **action** section in the **actions** section

## Solution Development in Visual Studio Code

---

- 23) Replace the **ActionName** with **Open**  
24) In the **OnAction** trigger, add the following line in the **begin end** block

```
action(Open)
{
    trigger OnAction();
    begin
        OpenSeminarCard;
    end;
}
```

- 25) Lastly, delete the **FactBox** area  
26) Save the file using **Ctrl+S**  
27) Verify that the solution looks like this:

```
page 123456741 "CSD My Seminars"

// CSD1.00 - 2018-01-01 - D. E. Veloper
// Chapter 10 - Lab 1 - 4
// - Created new page
{

    PageType = Listpart;

    SourceTable = "CSD My Seminars";

    Caption='My Seminars';

    layout

    {

        area(content)

        {

            repeater(Group)

            {

                field("Seminar No.;"Seminar No.")

                {

                }

                field(Name;Seminar.Name)

                {

                }

                field(Duration;Seminar."Seminar Duration")

                {

                }

                field(Price;Seminar."Seminar Price")

                {

                }

            }

        }

    }

}
```

```
actions
{
    area(processing)
    {
        action(Open)
        {
            trigger OnAction();
            begin
                OpenSeminarCard;
            end;
        }
    }
}

var
    Seminar : Record "CSD Seminar";

trigger OnOpenPage();
begin
    SetRange("User Id",UserId);
end;

trigger OnAfterGetRecord();
begin
    if Seminar.get("Seminar No.") then;
end;
```

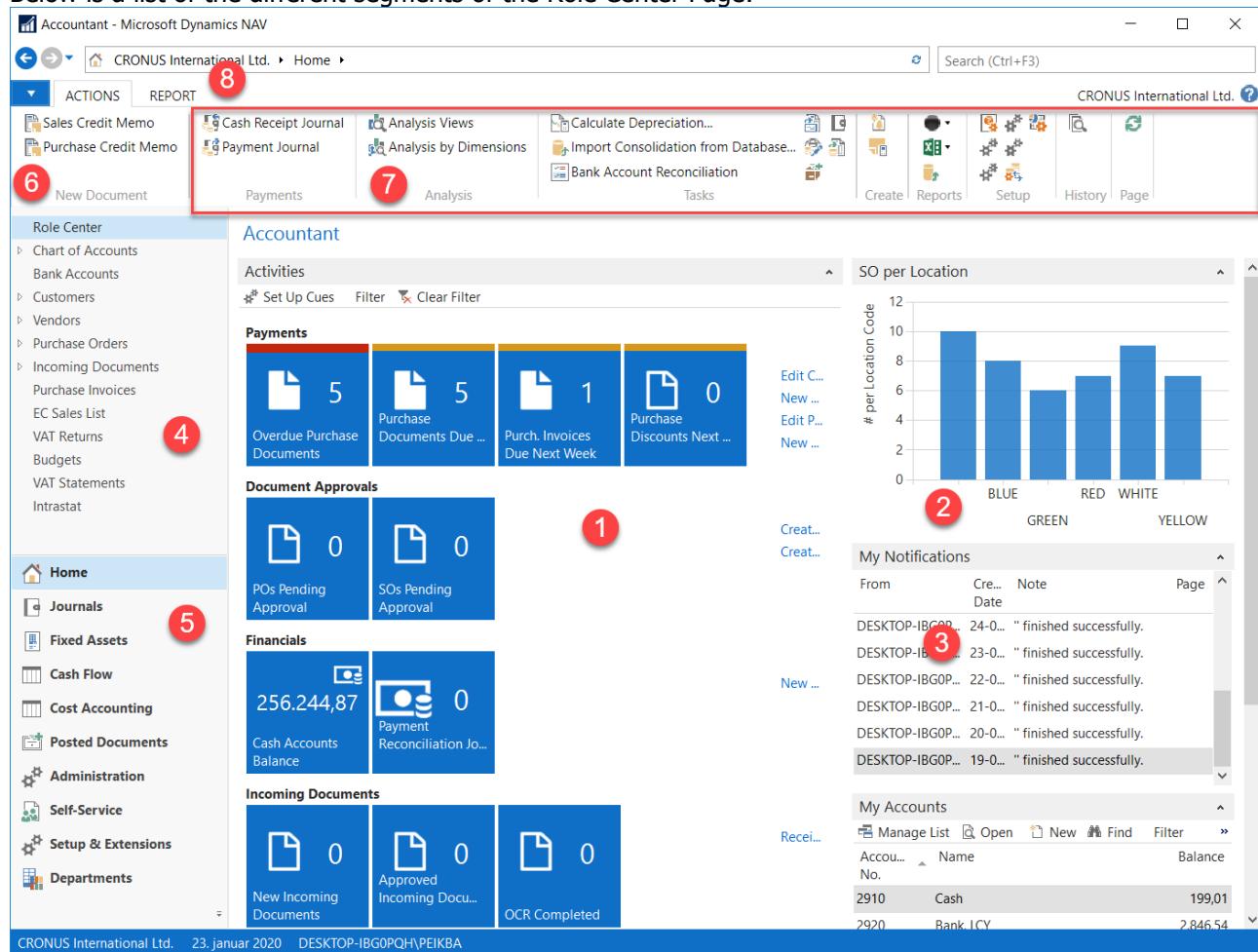
```
trigger OnNewRecord(BelowxRec : Boolean);
begin
  Clear(Seminar);
end;
local procedure OpenSeminarCard();
begin
  if Seminar."No."<>'' then
    Page.Run(Page::"Seminar Card",Seminar );
end;
}
```

## The Role Center Page

### Exercise Scenario

After having completed development of all supporting tables and pages, it is time to define the **Seminar Manager Role Center** page. For this page, the design of the page layout and various page parts, must be arranged in a manner that is consistent with standard Dynamics 365 BC Role Center pages. The navigation pane elements must be defined. This includes the Home items and any additional menus. Finally, it is necessary to define the ribbon actions for the Role Center page. The sections of the Role Center are named a little differently than previously.

Below is a list of the different segments of the Role Center Page:



The **content area** in the **layout** section contains the following items:

The Column1 **group**

- 1) Page parts

The Column2 Group

- 2) Chartparts
- 3) Systemparts

Page parts

The **actions** section

The **Embedding area**

- 4) Actions for the **Home** menu

The **Sections area**

- 5) Groups for each activity button and actions for each action under the button

The **Creation** area

- 6) Actions for all actions in the **New Documents** section

The **Processing** area

- 7) Actions for **action** items

The **Reporting** area

- 8) Actions that contains reports

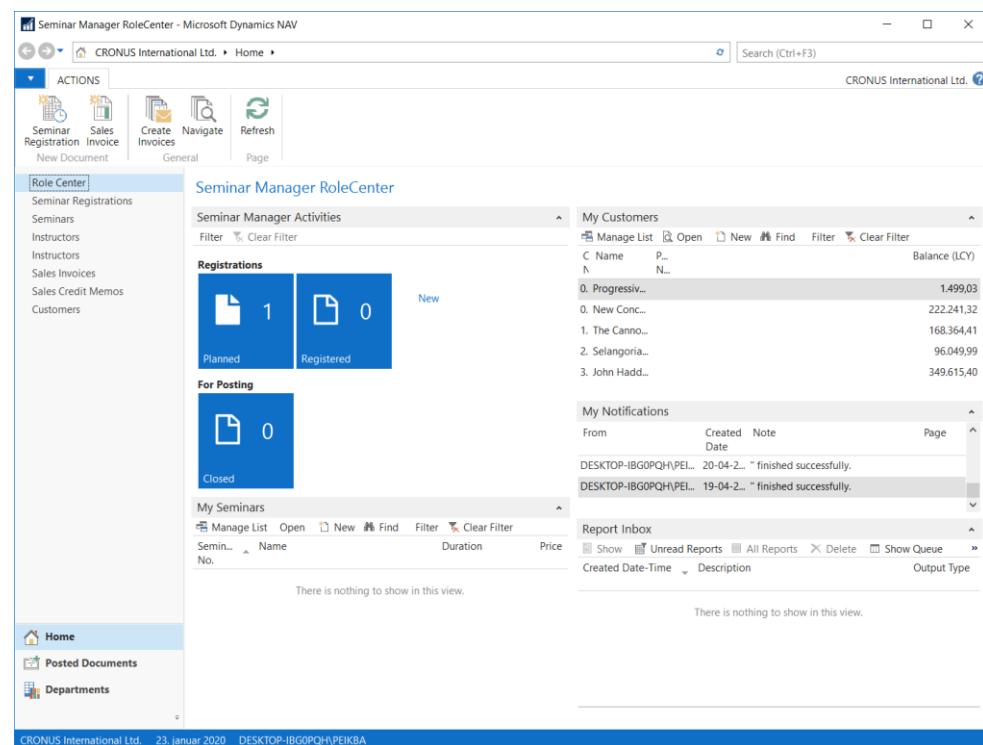
The **Navigation** area

- Action that previously would be positioned in the **Navigate** section (Related Information - none shown here)

### Task 5: Create the Seminar Manager Role Center Page

#### High Level Steps:

- 1 Create the **CSD Seminar Manager RoleCenter** page
- 2 assign the **ID** of **123456742** to it.
- 3 Set the **PageType** to be **RoleCenter**
- 4 Add the page parts as shown below
- 5 Add actions to provide the relevant Home items, Activity buttons and Action items



### Detailed steps:

- 1) In the VS Code Explorer window, click the **Page** folder, then right-click and select the **New File** menu item
- 2) Name the file **42\_SeminarManagerRoleCenter.al** and press **[Enter]**
- 3) Enter **tpa** in the first line and select the **tpage, Page of type card** snippet
- 4) Enter **123456742** as the **ID** and **CSD Seminar Manager RoleCenter** as the **name** and press enter
- 5) Change the **PageType** to be **RoleCenter**
- 6) Remove the Source table property
- 7) Add a **Caption** property
- 8) Locate the area and change the **Content** to **RoleCenter**
- 9) Locate the **group** section
- 10) Replace the **GroupName** with **Column1**
- 11) Remove the **field** section
- 12) Add the following page parts to the page in the **Column1** group:
  - a. Part: Seminar Manager Activities
  - b. Part: My Seminars
- 13) Add an extra group part after the **Column1** group section
- 14) Name the new group **Column2**
- 15) Add the following page parts to the page in the **Column2** group:
  - a. Part: My Customers
  - b. SystemPart: MyNotes
  - c. Part: ReportInbox
- 16) Locate the **actions** section
- 17) Create a new **Embedding** area
- 18) Create actions for
  - a. Seminar Registrations
    - i. page "Posted Seminar Reg. List"
    - ii. Image List
  - b. Seminars
    - i. Page "Seminar List"
    - ii. Image List
  - c. Instructors
    - i. page "Resource List"
    - ii. Image List
    - iii. Filter Type = Person
  - d. Rooms
    - i. page "Resource List"
    - ii. Image List
    - iii. Filter Type = Machine
  - e. Sales Invoices
    - i. page "Sales Invoice List"
    - ii. Image List
  - f. Sales Credit Memo
    - i. page "Sales Credit Memos"
    - ii. Image List
  - g. Customers

- i. Page "Customer List"
- ii. Image List

19) Create a new **Sections** area

20) Create a new **Posted Documents** group

21) Create actions for

- a. Posted Seminar Registrations
  - i. page "Posted Seminar Reg. List"
  - ii. Image List
- b. Posted Sales Invoices
  - i. page " Posted Sales Invoices "
  - ii. Image List
- c. Posted Credit Memos
  - i. page " Posted Credit Memos "
  - ii. Image List
- d. Registers
  - i. page "Seminar Registers"
  - ii. Image List

22) Create a new **Creation** area

23) Create actions for

- a. Seminar Registration
  - i. page "Seminar Registration"
  - ii. Image NewTimeSheet
  - iii. RunPageMode=Create
- b. Sales Invoice
  - i. page "Sales Invoice"
  - ii. Image NewSalesInvoice
  - iii. RunPageMode=Create

24) Create a new **Processing** area

- a. Create Invoices
  - i. Report "Create Seminar Invoices"
  - ii. Image CreateJobSalesInvoice
- b. Sales Invoice
  - i. Page "Navigate"
  - ii. Image Navigate

25) Delete the **var** section for global variables

26) Save the file using **Ctrl+S**

27) Verify that the solution looks like this:

```
page 123456742 "CSD Seminar Manager RoleCenter"

// CSD1.00 - 2018-01-01 - D. E. Velper
// Chapter 10 - Lab 1 - 5
// - Created new page
{

    PageType = RoleCenter;

    Caption='Seminar Manager RoleCenter';

    layout
    {
        area(RoleCenter)
        {
            group(Column1)
            {
                part(Activities;"Seminar Manager Activities")
                {
                }

                part(MySeminars;"My Seminars")
                {
                }
            }

            group(Column2)
            {
                part(MyCustomers;"My Customers")
                {
                }

                systempart(MyNotifications;MyNotes)
                {
                }

                part(ReportInbox;"Report Inbox Part")
                {
                }
            }
        }
    }
}
```

```
actions
{
    area(embedding)
    {
        action(SeminarRegistrations)
        {
            Caption = 'Seminar Registrations';
            Image = List;
            RunObject = Page "Posted Seminar Reg. List";
            ToolTip = 'Create seminar registrations';
        }
        action(Seminars)
        {
            Caption = 'Seminars';
            Image = List;
            RunObject = Page "Seminar List";
            ToolTip = 'View all seminars';
        }
        action(Instructors)
        {
            Caption = 'Instructors';
            RunObject = Page "Resource List";
            RunPageView = WHERE(Type = const(Person));
            ToolTip = 'View all resources registeres as
                        persons';
        }
    }
}
```

```
action(Rooms)
{
    Caption = 'Instructors';
    RunObject = Page "Resource List";
    RunPageView = WHERE(Type = const(Machine));
    ToolTip = 'View all resources registeres as
machines';
}
action("Sales Invoices")
{
    ApplicationArea = Basic,Suite;
    Caption = 'Sales Invoices';
    Image = Invoice;
    RunObject = Page "Sales Invoice List";
    ToolTip = 'Register your sales to customers';
}

action("Sales Credit Memos")
{
    Caption = 'Sales Credit Memos';
    RunObject = Page "Sales Credit Memos";
    ToolTip = 'Revert the financial transactions
involved when your customers want
to cancel a purchase;;';
}
```

```
action(Customers)
{
    Caption = 'Customers';
    Image = Customer;
    RunObject = Page 22;
    ToolTip = 'View or edit detailed information for the
               customers that you trade with.';

}
area(Sections)
{
    group("Posted Documents")
    {
        Caption = 'Posted Documents';
        Image = FiledPosted;
        ToolTip = 'View history for sales, shipments, and
                   inventory.';

        action("Posted Seminar Registrations")
        {
            Caption = 'Posted Seminar Registrations';
            Image = Timesheet;
            RunObject = Page "Posted Seminar Reg. List";
            ToolTip = 'Open the list of posted
                       Registrations.';

        }
    }
}
```

```
action("Posted Sales Invoices")
{
    Caption = 'Posted Sales Invoices';
    Image = PostedOrder;
    RunObject = Page "Posted Sales Invoices";
    ToolTip = 'Open the list of posted sales
                invoices.';
}

action("Posted Sales Credit Memos")
{
    Caption = 'Posted Sales Credit Memos';
    Image = PostedOrder;
    RunObject = Page 144;
    ToolTip = 'Open the list of posted sales credit
                memos.';
}

action("Registers")
{
    Caption = 'Seminar Registers';
    Image = PostedShipment;
    RunObject = Page "Seminar Registers";
    ToolTip = 'Open the list of Seminar Registers.';
}

}
```

```
area(Creation)
{
    action(NewSeminarRegistration)
    {
        Caption='Seminar Registration';
        Image=NewTimesheet;
        RunObject= page "Seminar Registration";
        RunPageMode=Create;
    }
    action(NewSalesInvoice)
    {
        Caption='Sales Invoice';
        Image=NewSalesInvoice;
        RunObject= page "Sales Invoice";
        RunPageMode=Create;
    }
}
area(Processing)
{
    action(CreateInvoices)
    {
        Caption='Create Invoices';
        Image=CreateJobSalesInvoice;
        RunObject= report "Create Seminar Invoices";
    }
}
```

```
action(Navigate)
{
    Caption='Navigate';
    Image=Navigate;
    RunObject= page Navigate;
    RunPageMode=Edit;
}
}
}
}
```

28) Next, Make sure that all files are saved by clicking the **File/Save All** menu item



29) Then click the Source Control icon  The number might be different depending on the last commit.

30) Enter the text **Lab 10.1** in the Message field



31) Then click the Commit checkmark 

32) Lastly, Push the changes to the off-site repository

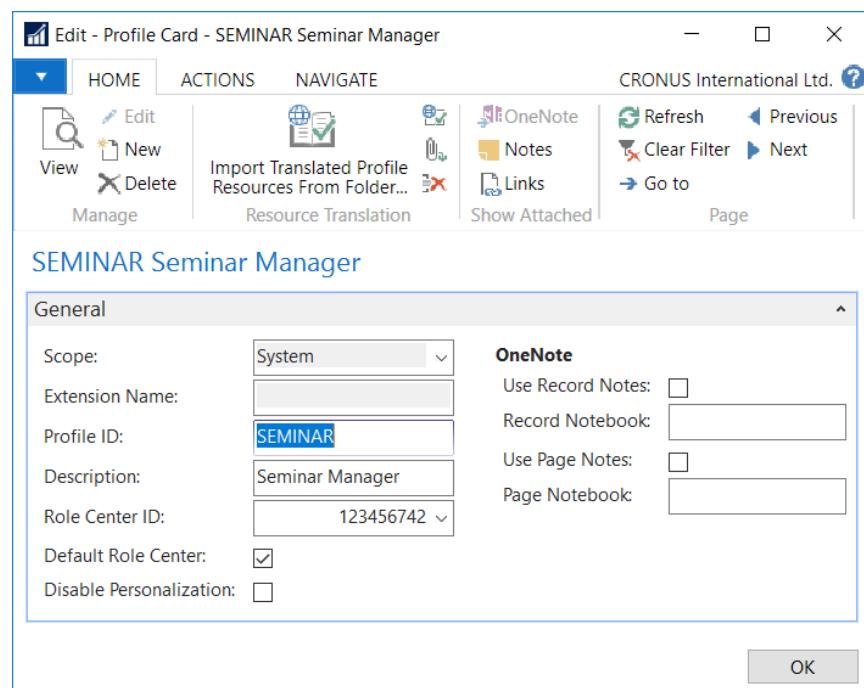
### Task 6: Create a new profile and assign the Role Center page

#### High Level Steps:

- 1 Create a new **Seminar Manager** profile in the Dynamics 365 BC client
- 2 Link the **CSD Seminar Manager RoleCenter** page (**123456742**) to it.
- 3 Set the profile to be default
- 4 Restart the Dynamics 365 BC client

#### Detailed steps:

- 1) Open the Dynamics 365 BC client
- 2) Locate the **Profile** list at **Departments > Administration > Application Setup > RoleTailored Client** or type **Prof** in the upper right corner and select the **Profiles** page
- 3) Create a new profile
- 4) Name the profile **Seminar**
- 5) Give the description **Seminar Manager**
- 6) Enter the Role Center ID **123456742**
- 7) Set the profile to be **Default Role Center**
- 8) Restart the client



## Module Review

### Module Review and Takeaways

RoleTailored clients for Dynamics 365 BC 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.