Programming in AL

for beginners

Krzysztof Bialowas

MyNAVblog.com

TABLE OF CONTENT

INTRODUCTION

Microsoft Dynamics 356 Business Central is an ERP system that can be easily customized with AL language extensions. This workbook has been prepared to guide you through the basics of the development.

I hope that after reading it and moreover doing all the tasks you will get familiar with the basics of developing in the AL Language. You should know that this is only the start. There is much more in functionalities in the programming language but here you will see the essential which will give you fundaments to build more advanced extensions.

How should you work with it? First of all, it is not a book. I would like to encourage you to print it (if you already do not have hardcopy) and write in it. Make some notes and write down the things which you would like to explore more. That is why I am calling it **WORKBOOK**.

In the next pages, you will see some code which you need to write. I would like to ask you something. **Do not copy it!** Try to write it by yourself. Use this which you can find here as an example and reference. This way you will understand more and get familiar with how to do it later by yourself.

I hope you will like the structure and the content. If so, please share some thoughts on social media such as LinkedIn, Twitter or comment on the blog **MyNAVBlog.com**.

The material here is fully free of charge. If you like you can use it in your organization or preparing the workshops for your colleagues. Please only do not remove copyrights from the materials.

Krzysztof Bialowas
www.MyNAVBlog.com

PROJECT REPOSITORY

Whole code used in this workbook and also the newest version of this workbook you can find on GitHub page:

https://github.com/mynavblog/ALForBeginners

Chapter 1

Preparing the development environment

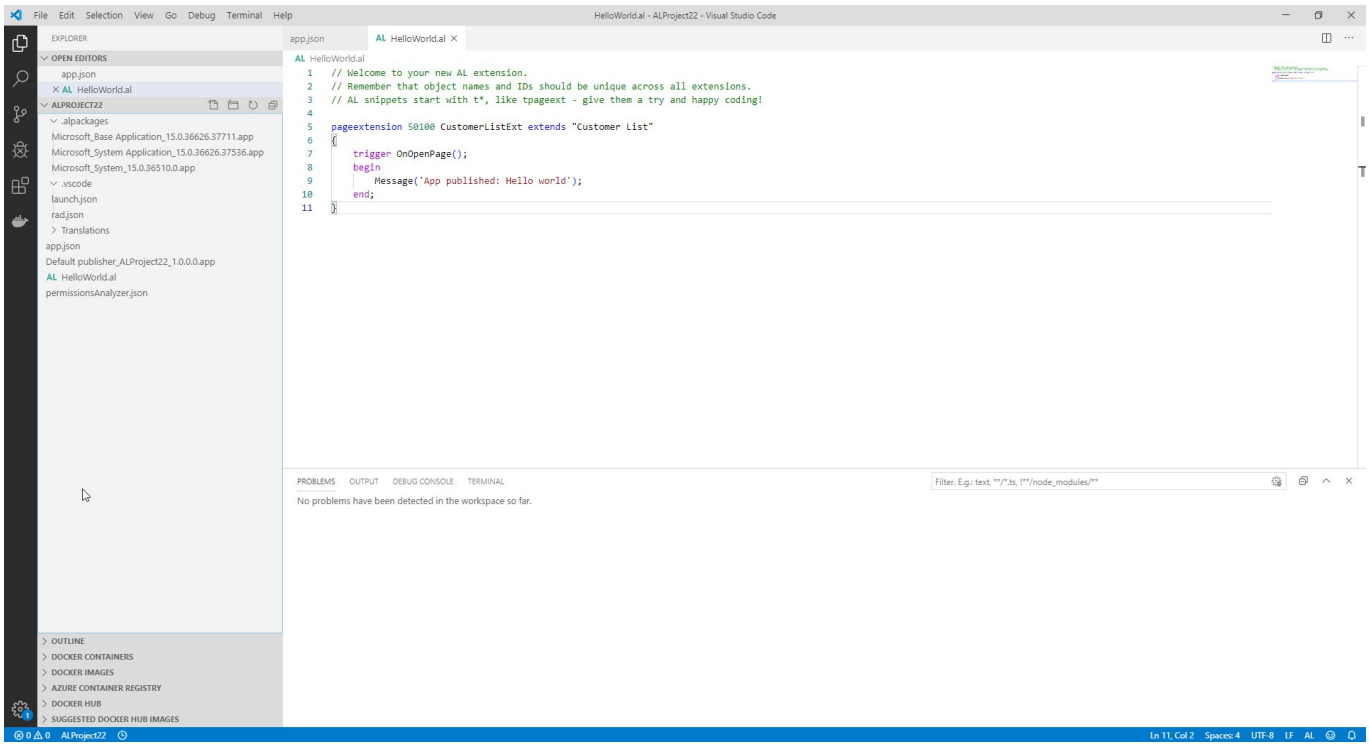
OBJECTIVES

To start building extensions for Microsoft Dynamics 365 Business Central you need to prepare the development environment first. Only Visual Studio Code is mandatory for the development however in this chapter you will get knowledge how to:

- Install Visual Studio Code with extensions for AL language
- Setup development database using Docker Container
- Setup development database directly on Online Sandbox
- Install Git which will be used for source control
- Create the repository on Azure DevOps

VISUAL STUDIO CODE

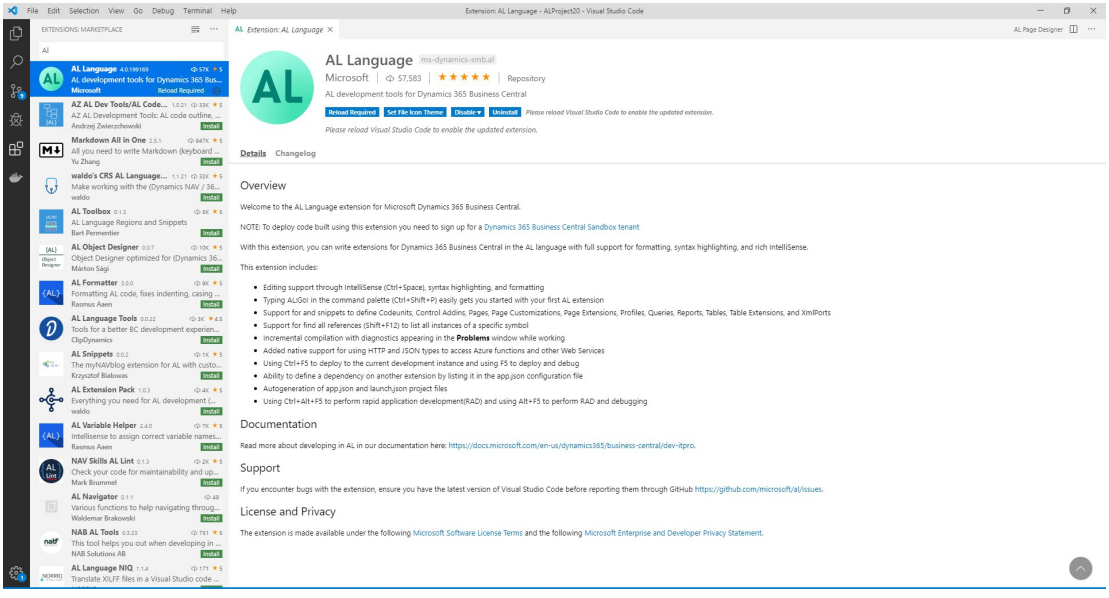
Visual Studio Code is the primary code editor for the Dynamics 365 Business Central development. It allows us to write and debug the code in the AL language in which the extensions are written. In the next chapters, you will get familiar with more functions and how to use it.



Installation

To install Visual Studio Code go to page https://code.visualstudio.com and download the current version for your platform.

After installation open Visual Studio Code and go to Extensions management and find the AL Language. Install it.



Hints

Visual Studio Code gives us the possibility to extend it. In the market place (Extension Management) you can find many useful addons that can be installed to help you working with the AL language.

At this moment it is good to check below extensions. Remember that, it all depends on you, how you would like to configure Visual Studio Code.

- *AZ AL Dev Tools/AL Code Outline*
- *Waldo's CRS AL Language Extension*
- *AL Variable Helper*
- *AL Object Designer*

DOCKER AND LOCAL DEVELOPMENT ENVIRONMENT

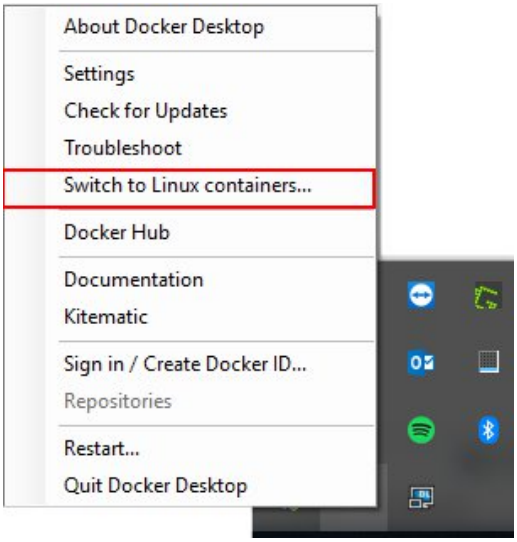
Docker allows you to create local development environments. Each environment is separated and packed in something called container. Thanks to it you can have more than one installations of Dynamics Business Central on your computer.

Creating a container is very easy and can be done using PowerShell Script. At this moment Microsoft supports the containers for almost every version of Business Central.

Installation

To install Docker go to the page https://www.docker.com. And click option Get Started. You will need to create a free account to download Docker.

After installation make sure that your Docker is running in Windows container Mode. To check that, in the tray bar, click right on the Docker icon and check if you have option **Switch to Linux Containers**. If yes then it means that your Docker works as Windows containers. You can also mark this option during the installation.



Creating first Business Central Container

To create a Business Central Docker Container you can use the PowerShell Script. Functions used in it are taken from the NavContainerHelper library. You can find out more about the library on this page https://github.com/microsoft/navcontainerhelper.

The script for creating the container, you can find also on GitHub page for this workbook.

Script

```
install-module navcontainerhelper -force
Import-Module navcontainerhelper
# set accept_eula to $true to accept the eula found here: https://go.microsoft.com/fwlink/?linkid=861843
$accept_eula = $true

$containername = 'bccontainer' #you can use your own name here
$navdockerimage = 'mcr.microsoft.com/businesscentral/sandbox:us'

$credential = get-credential -UserName $env:USERNAME -Message "Using Windows Authentication. Please enter your Windows credentials."


New-NavContainer -accept_eula:$accept_eula `
        -containername $containername `
        -auth Windows `
        -Credential $credential `
        -alwaysPull `
        -doNotExportObjectsToText `
        -usessl:$false `
        -updateHosts `
        -imageName $navdockerimage `
        -assignPremiumPlan `
        -includeTestToolkit `
        -licenseFile 'Please specify your local path for the license file'
```

Hints

The first time running the script can take quite a long time. This is because the image of Business Central needs to be downloaded. It is around 17 Gb.

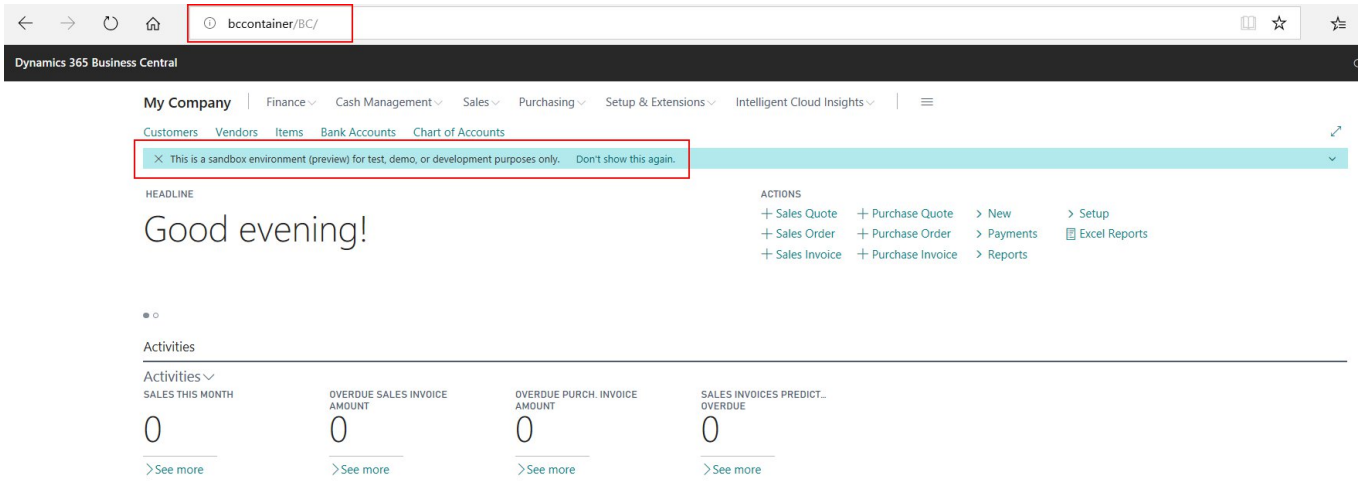
The script above installs the newest version of the Business Central Cloud Sandbox. If you would like to install On-prem version change in the script **$Navdockerimage** *to:*

mcr.microsoft.com/businesscentral/onprem

After the script will be executed, all information needed to connect to your local installation of Bussiness Central will be shown on the screen

Open your browser and go to page http://bccontainer/BC (if you change the container name then you need to put your container name. **BC** in this address is the server instance which by default is always BC).

```
Pulling image mcr.microsoft.com/businesscentral/sandbox:latest-ltsc2016
latest-ltsc2016: Pulling from businesscentral/sandbox
Digest: sha256:89f7818b1cb9cc1b86a4f271ad9d30559e8c4fd102477f20f0c77be54ff8ef36
Using image mcr.microsoft.com/businesscentral/sandbox:latest-ltsc2016
Removing container bccontainer
Removing bccontainer from host hosts file
Removing C:\ProgramData\NavContainerHelper\Extensions\bccontainer
Creating Container bccontainer
Version: 15.0.36626.37711-w1
Style: sandbox
Platform: 15.0.37582.0
Generic Tag: 0.0.9.95
Container OS Version: 10.0.14393.3204 (ltsc2016)
Host OS Version: 10.0.17134.1069 (1803)
The container operating system does not match the host operating system, forcing hyperv isolation.
Using locale en-US
Using hyperv isolation
Disabling the standard eventlog dump to container log every 2 seconds (use -dumpEventLog to enable)
Using license file C:\temp\lic.flf
Files in C:\ProgramData\NavContainerHelper\Extensions\bccontainer\my:
- AdditionalOutput.ps1
- license.flf
- MainLoop.ps1
- SetupNavUsers.ps1
- SetupVariables.ps1
- updatehosts.ps1
Creating container bccontainer from image mcr.microsoft.com/businesscentral/sandbox:latest-ltsc2016
dcd33d998d1030897bee4d884144cfa4171d90a12f3aac3b884979b65dbc7afe
Waiting for container bccontainer to be ready
Initializing...
Setting host.docker.internal to 172.31.169.131 in container hosts file (copy from host hosts file)
Setting gateway.docker.internal to 172.31.169.131 in container hosts file (copy from host hosts file)
Setting kubernetes.docker.internal to 127.0.0.1 in container hosts file (copy from host hosts file)
Setting host.containerhelper.internal to 172.28.32.1 in container hosts file
Starting Container
Hostname is
PublicDnsName is bccontainer
Using Windows Authentication
Assign Premium plan for BCCONTAINER\XXAMR
Container IP Address: 172.28.34.49
Container Hostname  :
Container Dns Name  : bccontainer
Web Client          : http://bccontainer/BC/
Dev. Server         : http://bccontainer
Dev. ServerInstance : BC
Setting bccontainer to 172.28.34.49 in host hosts file

Files:
http://bccontainer:8080/al-4.0.192371.vsix

Initialization took 111 seconds
Ready for connections!
Reading CustomSettings.config from bccontainer
```

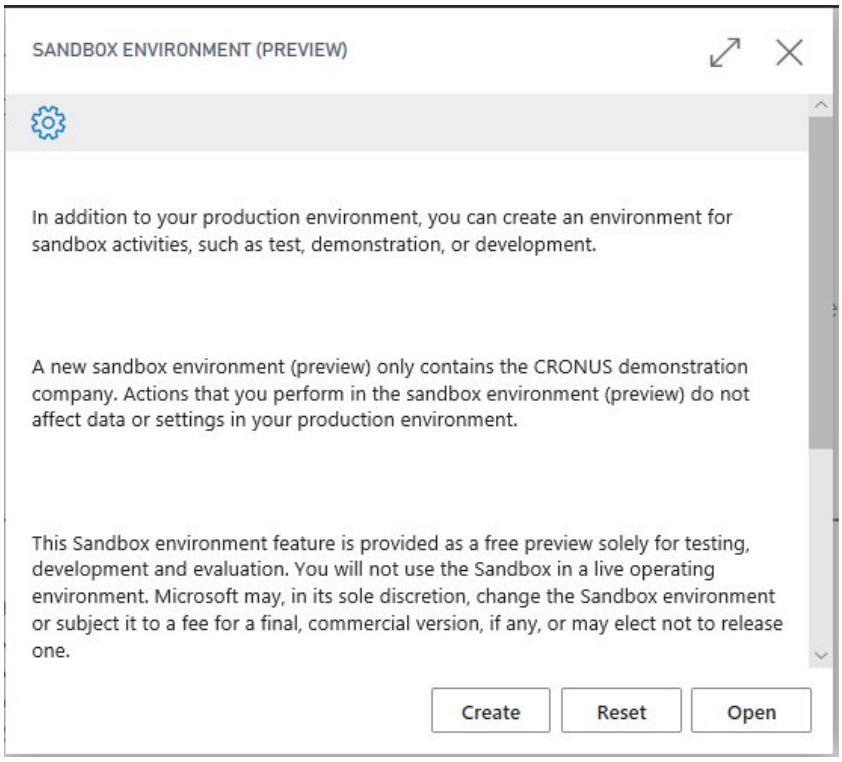


ONLINE SANDBOX ENVIRONMENT

The Sandbox is the alternative to work with Docker. It can be created directly from your production or trial version of Business Central Online. Below steps show how to create the trial version of Business Central and then create a Sandbox.

Go to page https://trials.dynamics.com and find Business Central. Then you can create your trial version for 90 days.

Login to https://businesscentral.dynamics.com and find in Tell me option **Sandbox Environment (Preview)**. There you can create the sandbox.



VERSION CONTROL - GIT

When doing development for Business Central there is no common database where the code is stored.

Git is a distributed version control system. It is used to track the changes in the code. In AL development it will be used, together with the remote repository, to comment on the code, track changes in it and store it in a safe place.

Git allows to skip documenting the changes directly in the code which means code is cleaner. Additionally, it controls when and by whom the changes were done.

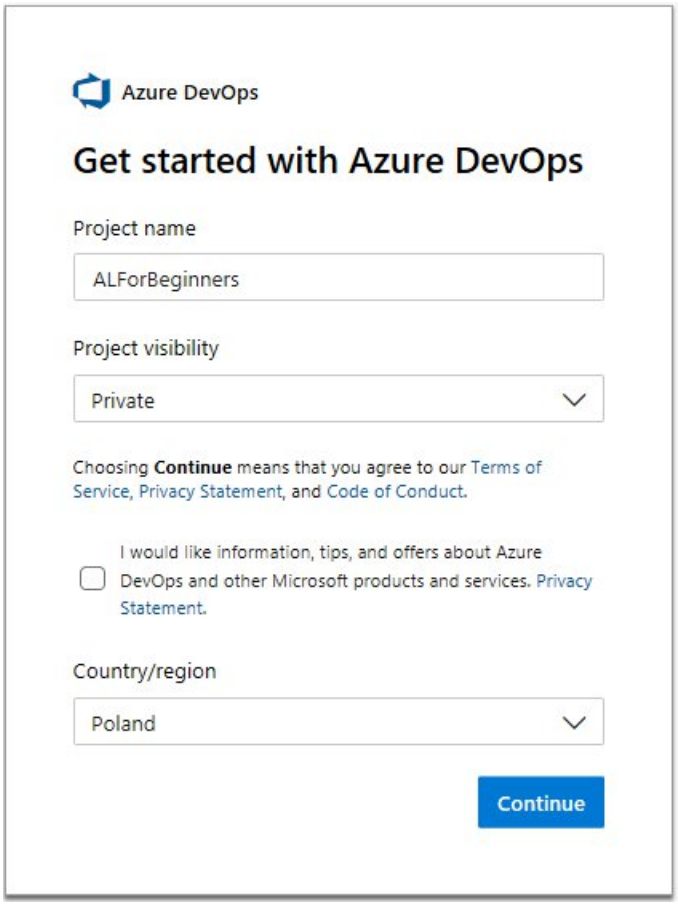
Installation

To install Git go to page https://git-scm.com and download version for your platform. After installation Git is ready to be used.

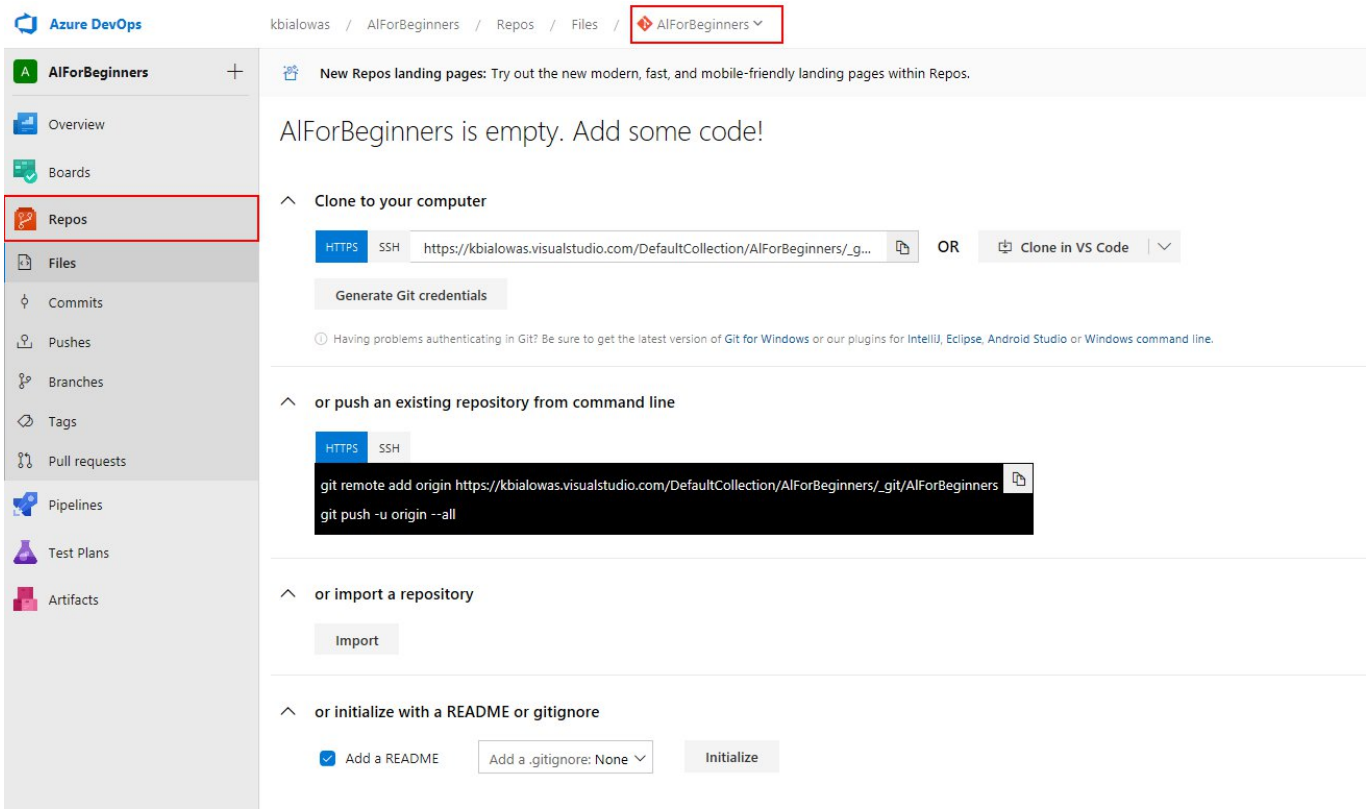
AZURE DEVOPS

Azure DevOps allows hosting your Git repository. It has also many other functions such as managing the team tasks, creating build, release pipelines and much more. However, in this workbook, only repositories functionality will be used and described.

If you do not have the Azure DevOps account yet, go to page https://dev.azure.com and Start free. After login you will be able to choose the project name and if it is public or private.



After going to your project's Repos in Azure DevOps, you should see the page with an empty repository. In the next chapters, you will get familiar with how to put and download the code from the repository. Remember that you can create multiple repositories in one project.



Hints

Azure DevOps is free for teams up to five users. If your team is bigger you will need to pay a small fee.

For hosting the repository you also can use GitHub but Azure DevOps gives more functionalities for handling the project and CI/CD.

CHAPTER SUMMARY

In this chapter, we did not do any development but we prepared the environment.

You can choose if you want to publish your extensions in the online sandbox or you want to use a Docker container for that.

You also will be able to manage your code on the remote repository on Azure DevOps.

After the chapter, you have got everything installed for extensions development for Business Central.

Chapter 2

First extension for Business Central

OBJECTIVES

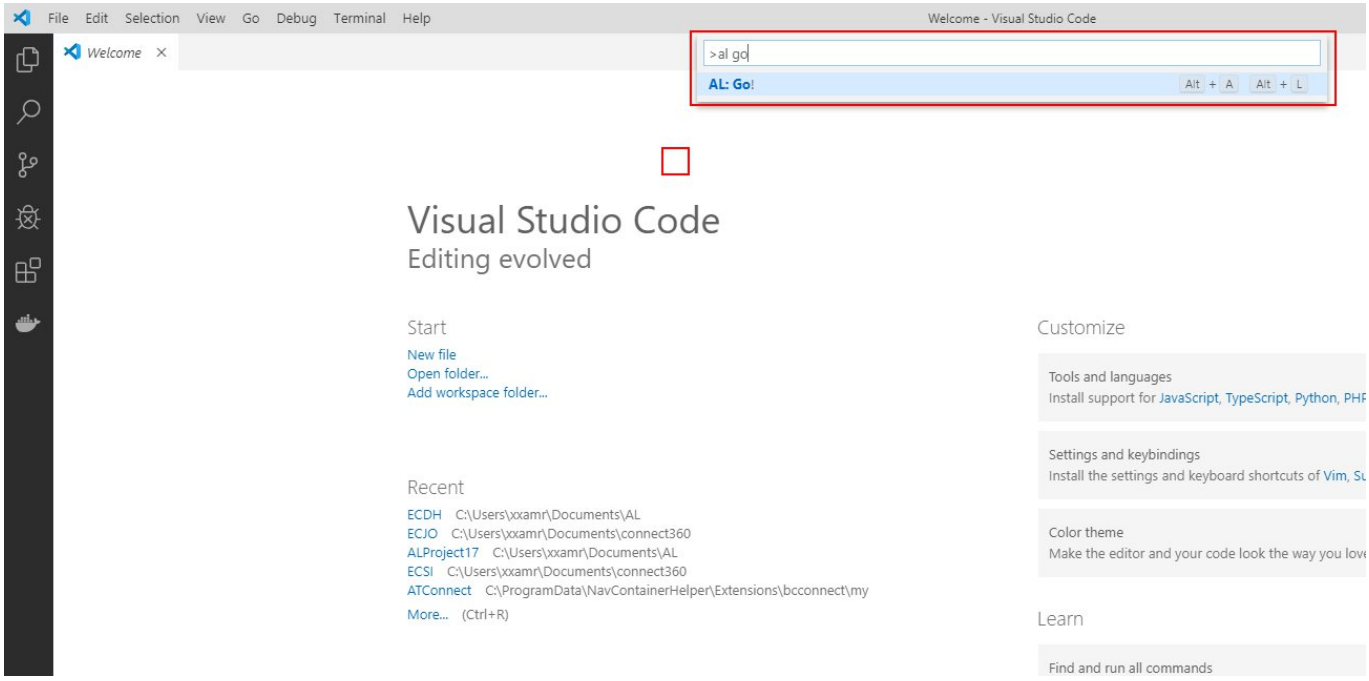
After preparing the environment it is time to do first development for Business Central. In almost every programming language the first program is to show "Hello World" on the screen. In this chapter, you will do the same. The objectives are:

- Create a new AL project in Visual Studio Code
- Connect to Business Central
- Get familiar with app.json file
- Understand what are the dependencies and which are needed
- Publish Hello World extension to the development environment
- Push code to remote repository on Azure DevOps

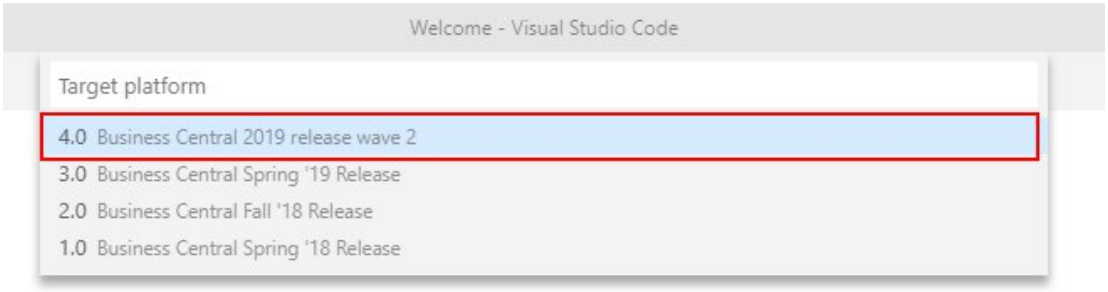
FIRST PROJECT

To create a new project for Business Central open Visual Studio Code and then open Command Pallete. The key shortcut for that is Ctrl+Shift+P.

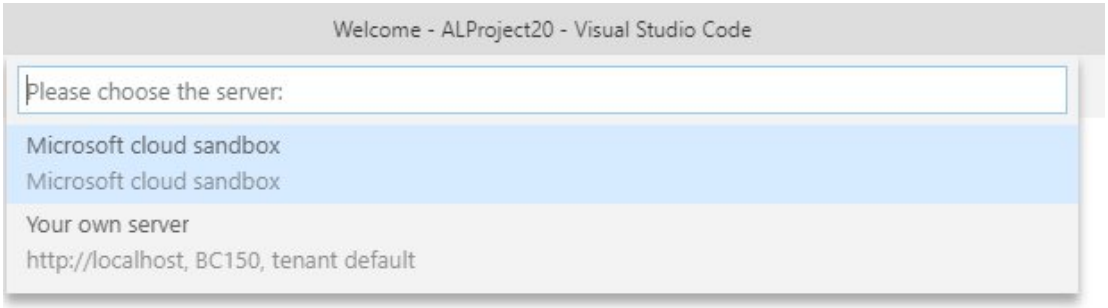
Then write AL: Go!. You can also use the key shortcut Alt+A, Alt+L.



After specifying where the project should be stored, you will need to choose for which platform you are doing development. In this workbook, the examples will be developed for the newest available platform.



The next step is to decide if your development environment will be an online sandbox or your own Docker container. If you would choose the online sandbox you will be prompt to log in to it.



After that your first application is created. For now it will contain primarly three files:

- launch.json
- app.json
- HelloWorld.al

launch.json

This file describes where the development environment is placed. If you using an online sandbox then your **lanuch.json** file should look similar to below.

.vscode > launch.json > ...

```
{
    "version": "0.2.0",
    "configurations": [
        {
            "type": "al",
            "request": "launch",
            "name": "Microsoft cloud sandbox",
            "startupObjectId": 22,
            "startupObjectType": "Page",
            "breakOnError": true,
            "launchBrowser": true,
            "enableLongRunningSqlStatements": true,
            "enableSqlInformationDebugger": true
        }
    ]
}
```

If you decided to use the Docker container then you should choose the option **Your own server**. After that you need to specify what is the address of your server - the same as your container name and what is the service instance name - BC by default for the Docker containers.

Additionally, you need to specify the way of authentication. Either UserPassword or Windows. If you follow the script from the previous chapter then authentication should be Windows.

.vscode > launch.json > ...

```
{
    "version": "0.2.0",
    "configurations": [
        {
            "type": "al",
            "request": "launch",
            "name": "Your own server",
            "server": "http://bccontainer",
            "serverInstance": "BC",
            "authentication": "Windows",
            "startupObjectId": 22,
            "startupObjectType": "Page",
            "breakOnError": true,
            "launchBrowser": true,
            "enableLongRunningSqlStatements": true,
            "enableSqlInformationDebugger": true
        }
    ]
}
```

Hints

The launch.json file has many other options related to debugging.

It is possible to have more than one configuration in the launch.json file at the same time.

app.json

This file provides the most important information about the extension which you develop. You can find here a name and a description of your extension, information about a publisher. Also, you can specify the web addresses for a help site, end-user license agreements (EULA).

app.json > [ ] features

```
{
  "id": "ad368519-0b40-4c4f-ba90-8fa6eff100dc",
  "name": "AL For Beginners",
  "publisher": "MyNAVBlog",
  "version": "1.0.0.0",
  "brief": "",
  "description": "This extension is for workshops related to AL for Beginners",
  "privacyStatement": "",
  "EULA": "http://www.mynavblog.com/eula",
  "help": "http://www.mynavblog.com/help",
  "url": "http://www.mynavblog.com/",
  "logo": "",
  "dependencies": [
    {
      "appId": "63ca2fa4-4f03-4f2b-a480-172fef340d3f",
      "publisher": "Microsoft",
      "name": "System Application",
      "version": "1.0.0.0"
    },
    {
      "appId": "437dbf0e-84ff-417a-965d-ed2bb9650972",
      "publisher": "Microsoft",
      "name": "Base Application",
      "version": "15.0.0.0"
    }
  ],
  "screenshots": [],
  "platform": "15.0.0.0",
  "idRanges": [
    {
      "from": 50100,
      "to": 50149
    }
  ],
  "contextSensitiveHelpUrl": "http://www.mynavblog.com/help/",
  "showMyCode": true,
  "runtime": "4.0",
  "features": ["TranslationFile"]
}
```

In the **app.json** file, you also can find the dependencies to the other applications. For start, you need only two dependencies which are created automatically - System Application and Base Application. Those two will allow you to extend the standard Business Central application.

Hints

The System Application is created from modules. All information about them you can find on page https://docs.microsoft.com/en-us/dynamics365/business-central/dev-itpro/developer/devenv-system-application-overview

When creating the project in app.file you may be asked to add one additional parameter. Add at the end **"features": ["TranslationFile"]** *(as in example above)*

HelloWorld.al

This file is your first code. It simply shows the message when you open the customers list. In later parts of this workbook we will remove this file but for now, let's keep it.

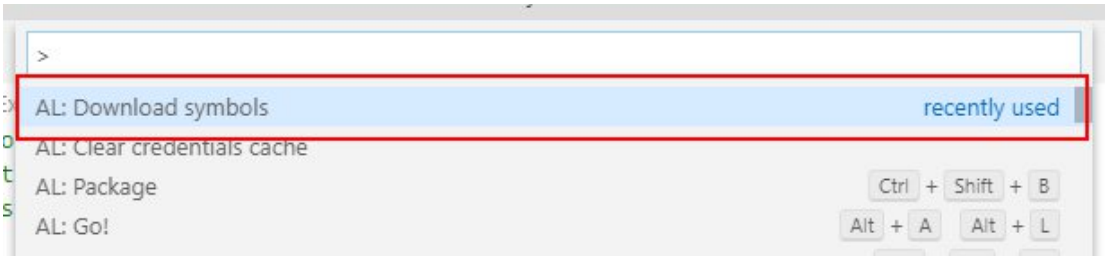
```
AL HelloWorld.al > ...
1   // Welcome to your new AL extension.
2   // Remember that object names and IDs should be unique across all extensions.
3   // AL snippets start with t*, like tpageext - give them a try and happy coding!
4
    0 references
5   pageextension 50100 CustomerListExt extends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

DOWNLOAD SYMBOLS AND RUN THE EXTENSION

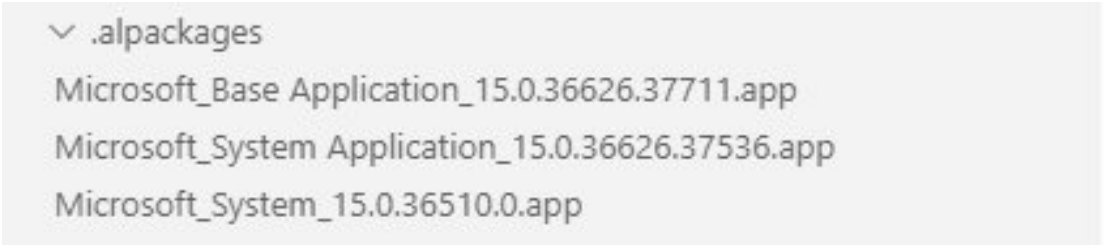
Probably, when you created a new project and open HelloWorld.al file, you have few errors. One of them says that *the target page "Customer List" for the extension is not found*. This is because you do not have symbols downloaded.

Symbols can be found in the folder **.alpackages**. To download them you need to open Command Pallet (Ctrl+Shift+P) and run function **AL: Download symbols**.

If you run the function for the first time and you are using an online sandbox then you will need to login to your environment.

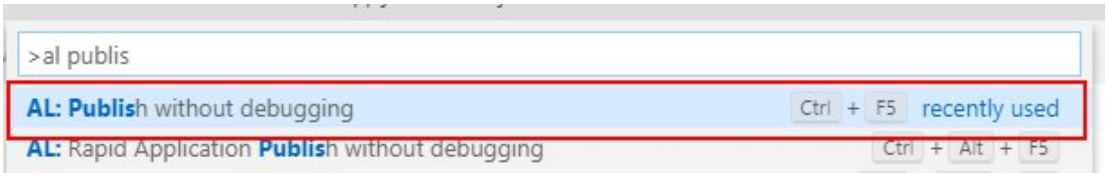


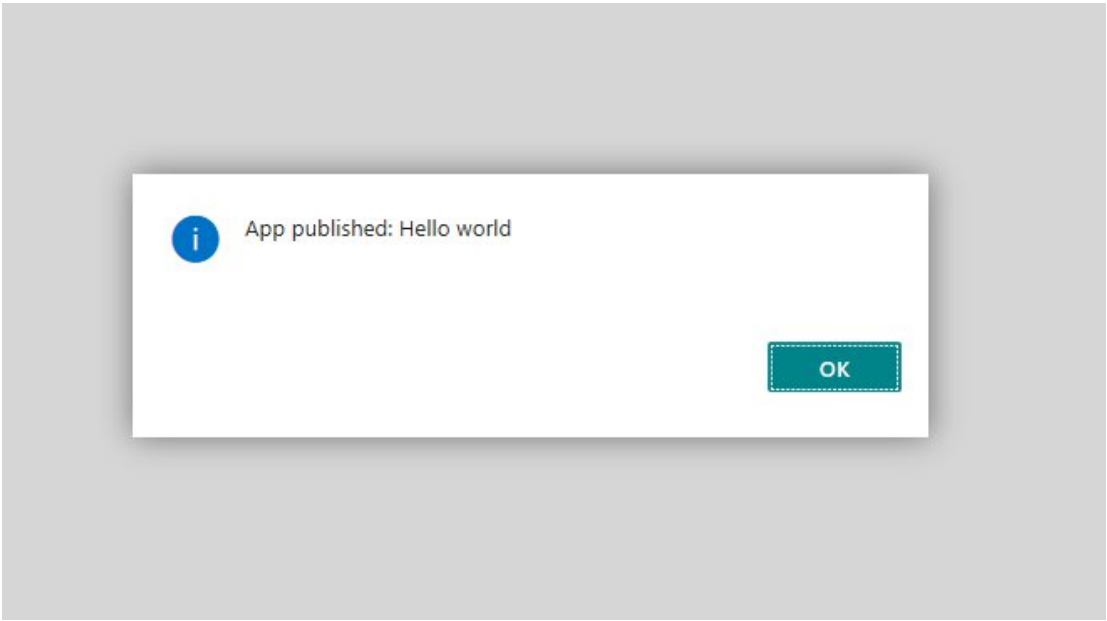
After the symbols will be downloaded, you should be able to see the app files in the **.alpackages** folder. If all is correct at least two files should be created - one for System Application and one for Base Application.



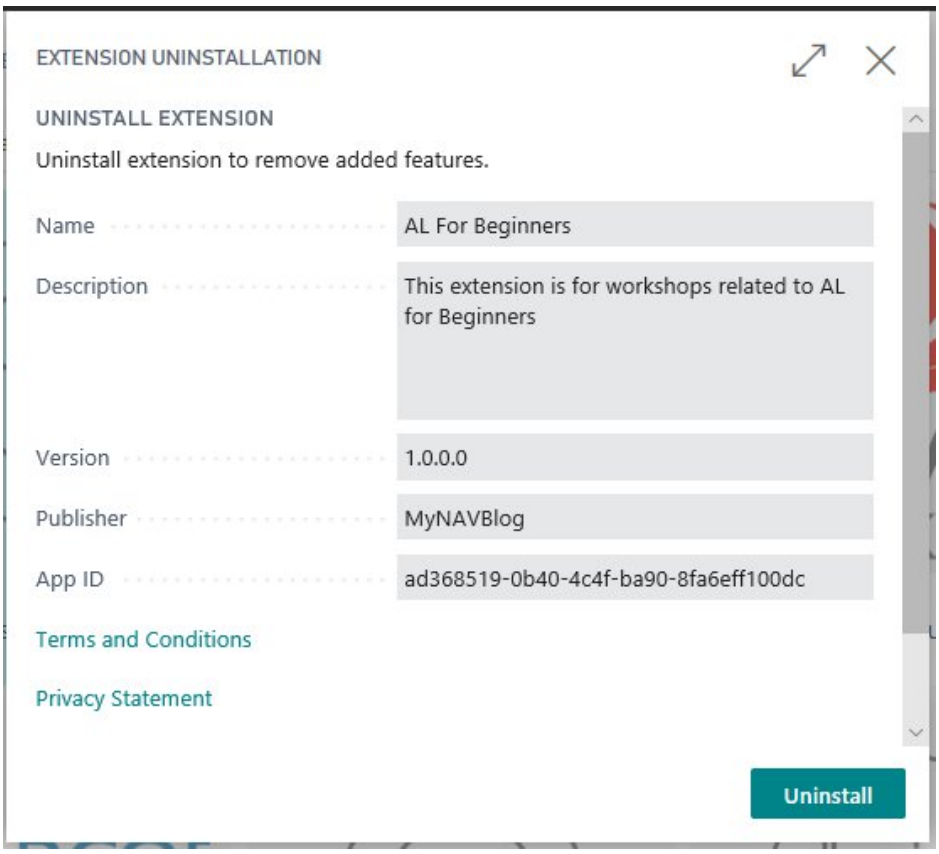
Now you can see the results of your first extension. To publish the extension open Command Pallet (Ctrl+Shift+P) and run function **AL: Publish without debugging**.

The web page with your Business Central will be opened and you should see the message every time when you open Customer List.





You will be able to see your extension in the Extensions Management. Go and try to find it. It should be automatically installed. As you can see all information from the app.json file are present here.



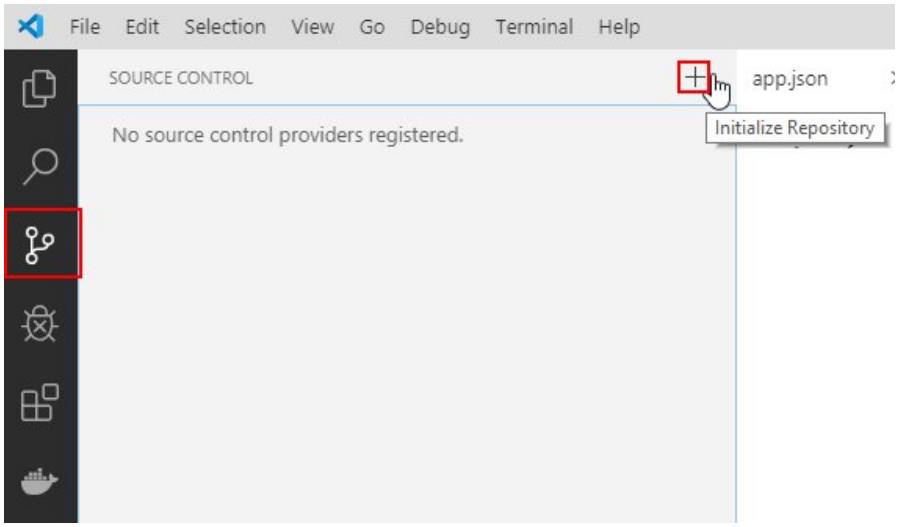
PUSH CODE TO THE REPOSITORY ON AZURE DEVOPS

As you already know, there is no common database doing development for extensions for Business Central. It means that at this moment the source code of your extension is only on your laptop. Think what would happen if something goes wrong?

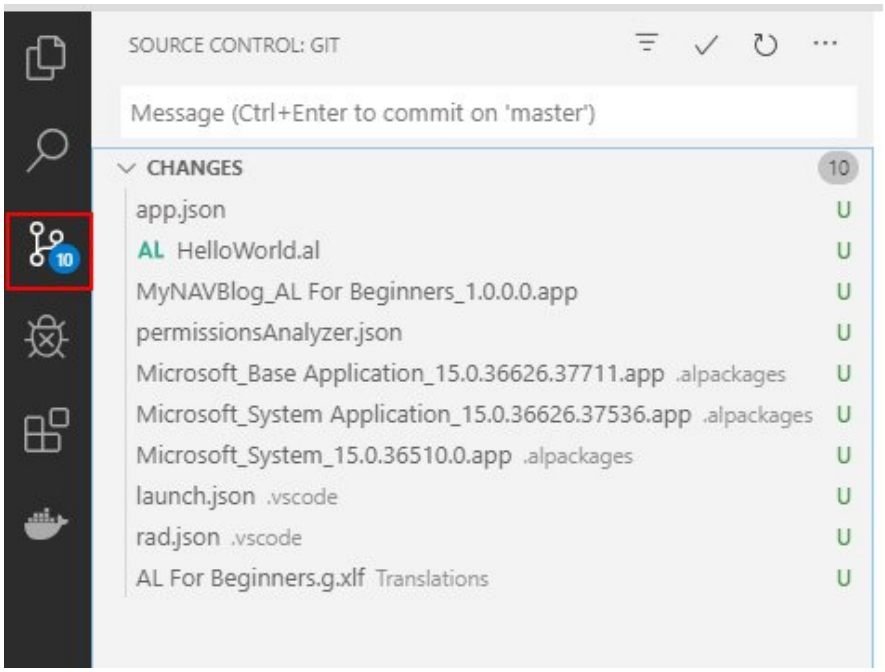
That is why the next step is to push the code to the remote repository. In the previous chapter, you already created it on Azure DevOps. Now you need to do first commit and send the code to secure location.

Init local repository

The first step to connect to a remote Git repository is to create a local one. To do so, open the Source Control menu and init repository. Then pick the folder with Git repository.



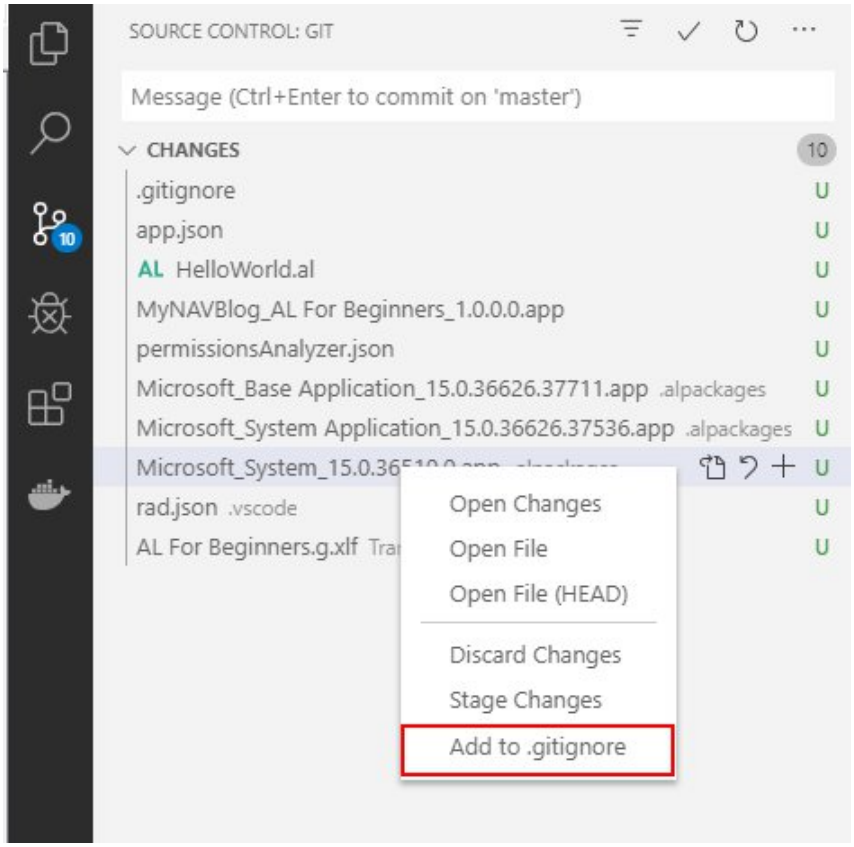
After that, you should see in the Source Control menu your source control is Git. Additionally, you can see all the changes which you did in the project. For now, all objects are new in the repository that is why there is letter U next to the changes.



.gitignore

Some of the files are not needed to be tracked in the source control. Such files as launch.json or all files with extension app are changing quite often. That is why it is good to use the **.gitignore** function. Such files which are marked as ignored will not be sent to the repository.

To add a file to **.gitignore** you need to right-click on the file and use action **Add to .gitignore**.



Hints

In each project **.gitignore** *is almost exactly the same. That is why you can use below code in your* **.gitignore** *file*

Code

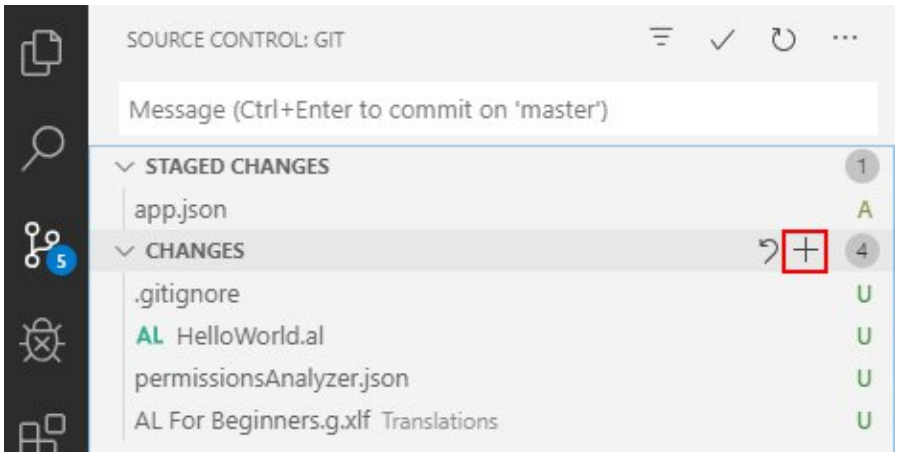
```
.alpackages/
.altemplates/
.alcache/
.vscode/
*.app
```

Stage and commit the changes

Before sending changes to the remote repository the first step is to add (stage) the changes which should be sent. Then you need to give some title for those changes.

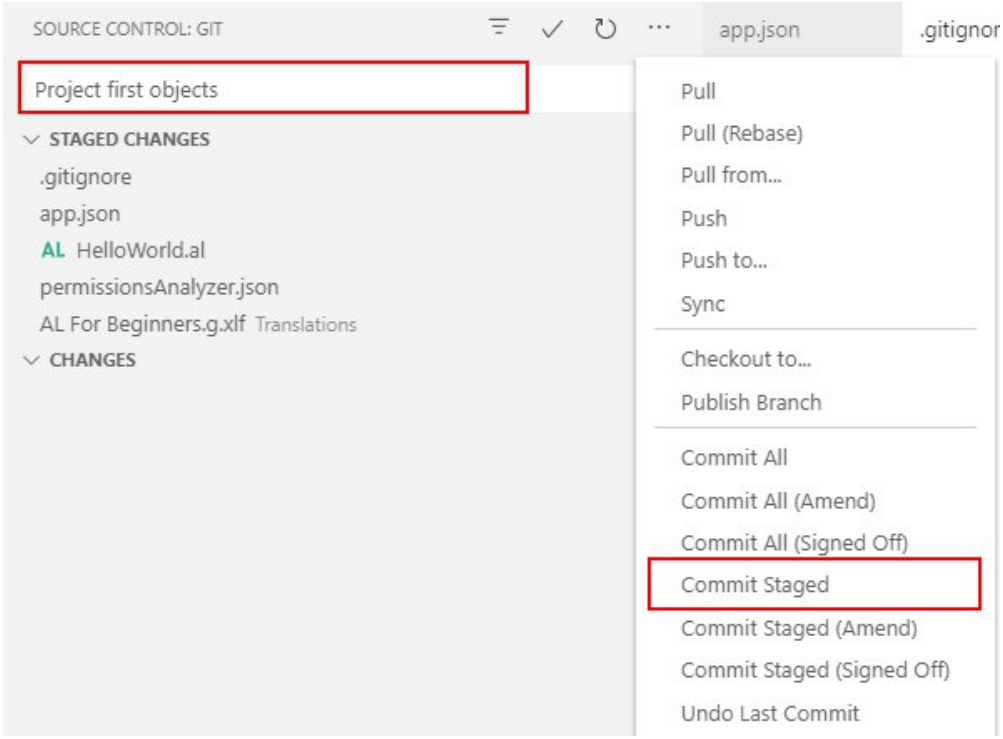
To stage the changes, you need to mark the files, and from the menu choose action **Stage Changes**. If you want to stage all the changes you can click the action **Stage All Changes**.

All staged changes will be shown under **Staged Changes**.



To commit the changes you need to write some message. It should give some meaning to all people who are involved in the project. Also to you. When you will see the history of the project, you will easily be able to tell why you did the change in code.

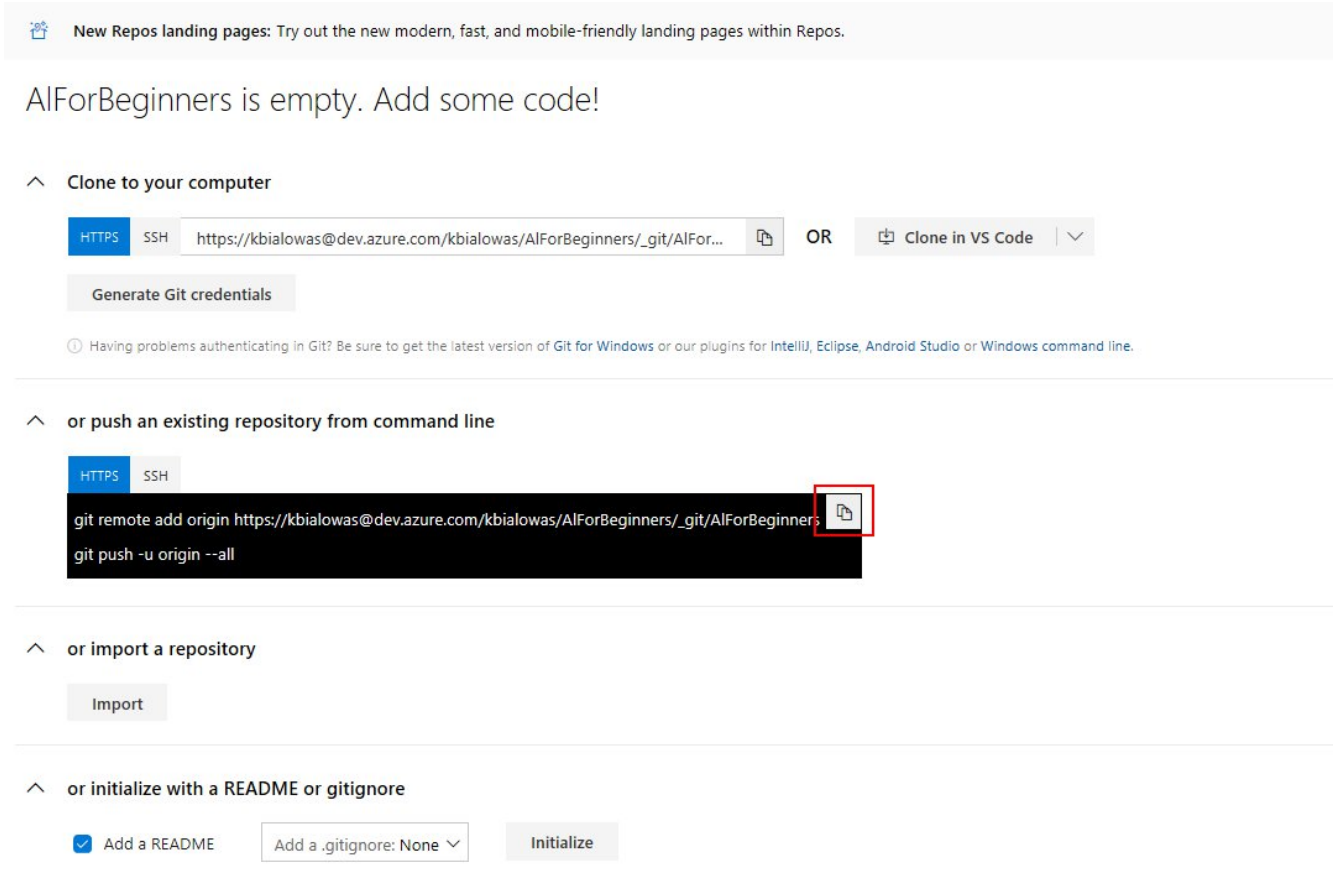
After writing the message choose from the menu action **Commit Staged**. If you would run the action in the Source Control menu you will see that you do not have any changes. Which means that from the last commit there are no new changes.



Connect local repository with remote

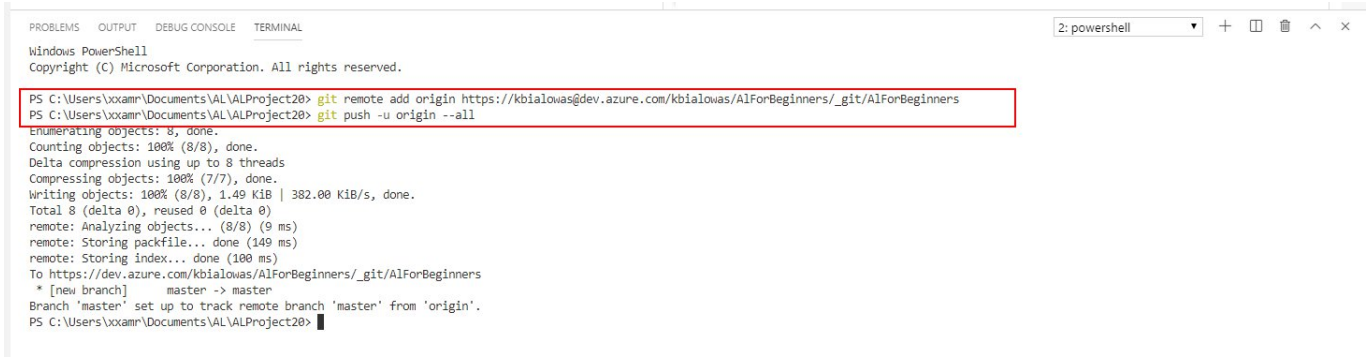
This step is mandatory to send the files to the remote repository. You need to do it only once when you start your project and you did not push any changes to Azure DevOps yet.

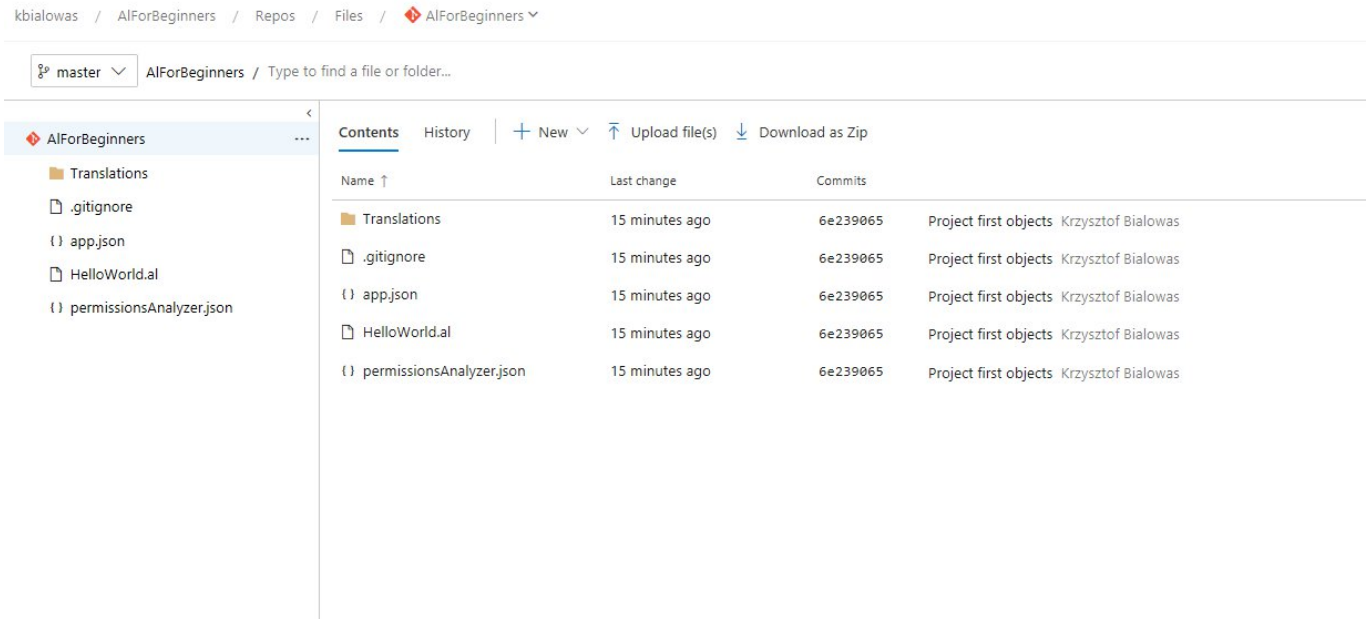
In the previous chapter, you create a repository on Azure DevOps. Open the page and copy two functions that are shown on a black background.



Paste those two functions in the Terminal Console which is visible in Visual Studio Code and run them. You will need to put your credentials for Azure DevOps.

After that, you should see that all changes have been push to the remote repository. You can check it by refreshing your repository page. Now instead of the previous view, you should see the same files as in your project.





CHAPTER SUMMARY

In this chapter, you developed the first extension for Business Central in the AL language. Congratulations!

Now you also know what you can find in the app.json and lanuch.json.

You get familiar with how to stage the changes in the project and commit them.

All the code done so far is secure in the repository in Azure DevOps.

Chapter 3

Extension overview, project settings and structure

OBJECTIVES

In this chapter, you will find out the best practices needed to start developing the extensions in the AL language. You will also get information on what you will be working on in the next chapters. The objectives are:

- Get familiar with the extension overview
- Enable the Code Analysis Tool
- Get familiar with the affixes
- Get familiar with object naming best practices

BONUS REGISTRATION EXTENSION

Your company just decided to start a new project - **Bonus Registration Extension for Business Central**, and your management decided that you will be working on it. Isn't that great?

They want to have that extension to be published on the **AppSource**, where all extensions for Business Central are listed.

What the extension should do?

Here are some bullet points from your system architect:

> *First of all, the user should be able to create the Bonus Card. It should be possible to tell which Bill-to Customer the bonus is created and what is starting and ending date of it.*

> *The bonus should be calculated only for items. Users should be able to decide if the bonus is for particular items or for all items.*

> *It must be possible to tell what is the bonus percent per item.*

> *There should be two statuses for the Bonus Card: Open when the bonus is not calculated, Released when the bonus is calculated and the Bonus Card is not editable.*

> *The users should be able to see for which posted sales lines bonus was granted. On the Bonus Card, users should see a total amount of the bonus.*

> *Users should be able to print simple report for the Bonus Card.*

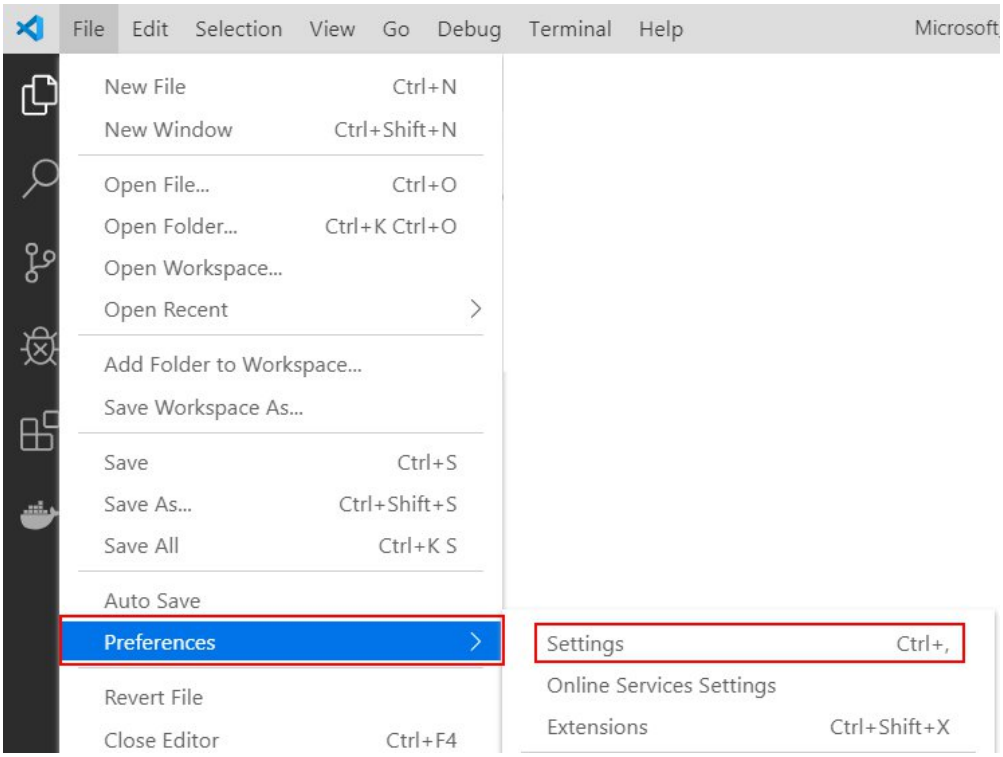
Quite a lot of points but in the next chapters you will find out how to develop such an extension for Business Central.

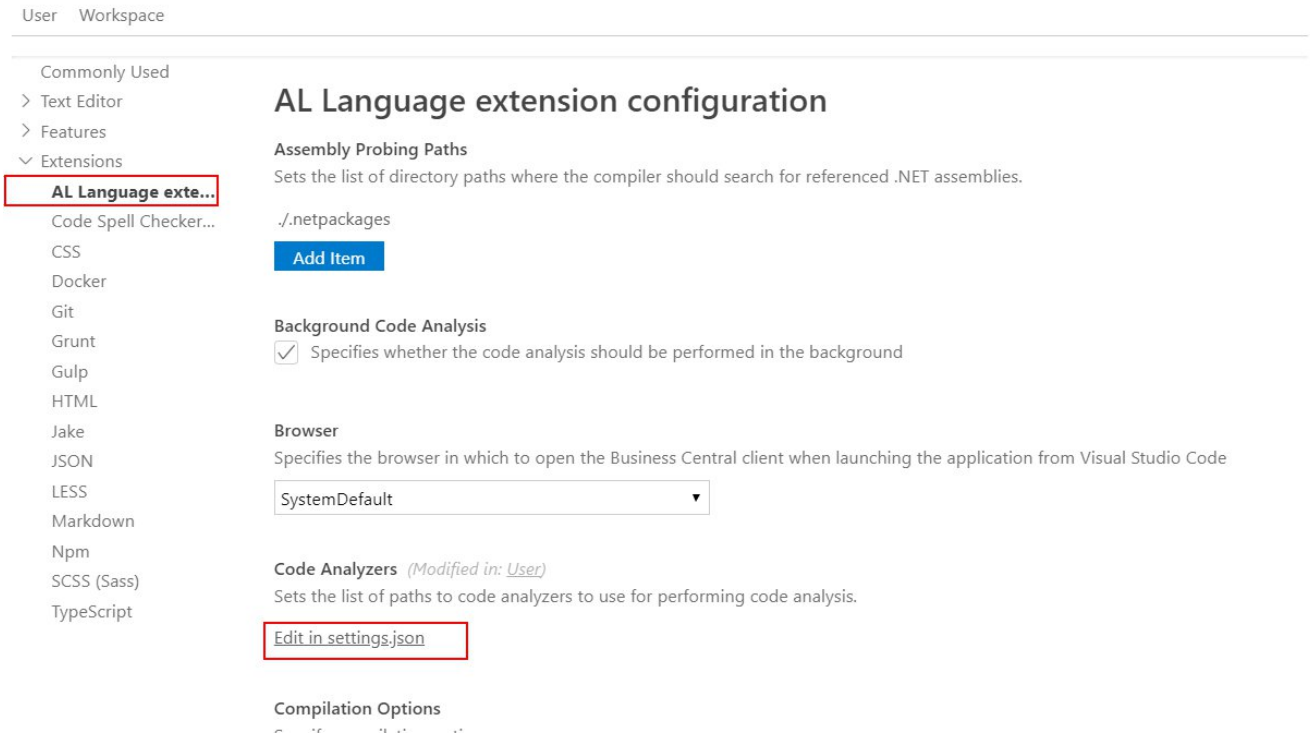
CODE ANALYSIS TOOL

In the AL language, there are provided four code analysis sets. The code analyzer is responsible for checking if the code which is developed is done according to Microsoft guidelines and if the provided code would not give you problems when publishing your extension in AppSource or Business Central tenant. Below you can find names of the sets:

- CodeCop
- AppSourceCop
- UICop
- PerTenantExtensionCop

To enable code analysis open **settings.json** file. You can do it either from the menu File, Preferences and then Settings or by writing in the Command Pallet - **Open Settings (JSON)**.





In the **settings.json** you must add two parameters. The first one is **al.enableCodeAnalysis** which enables the functionality and second is **al.codeAnalyzers**. This one says which code analyzers should be active.

The code example you can find below.

Code

```
"al.enableCodeAnalysis": true,
"al.codeAnalyzers": ["${AppSourceCop}","${CodeCop}"]
```

Hints

For the extensions which are developed for AppSource, the AppSourceCop is mandatory. However, it is always good to have also CodeCop turn on because it shows the guidelines for code.

What is checked for **AppSourceCop** *you can find on this page: https://docs.microsoft.com/en-us/dynamics365/business-central/dev-itpro/developer/analyzers/appsourcecop*

What is checked for **CodeCop** *you can find on this page: https://docs.microsoft.com/en-us/dynamics365/business-central/dev-itpro/developer/analyzers/codecop*

AFFIXES

Each new object which you develop for the Business Central extension needs to have own prefix or suffix. It is mandatory to avoid object naming conflicts. Also if you extend standard objects such as page or table for each field, control or action you need to add the prefix or suffix.

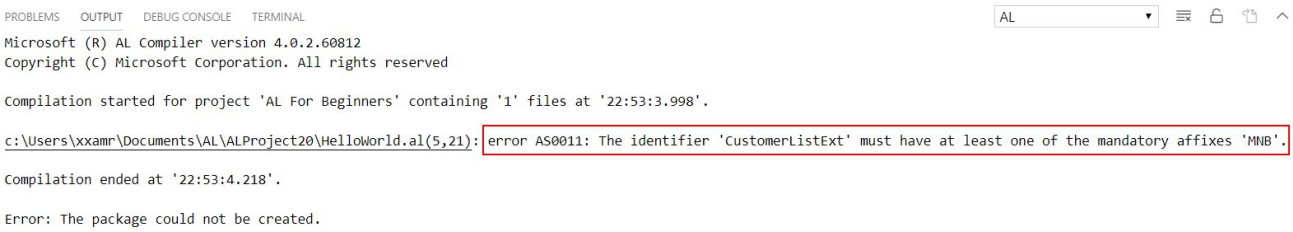
The general rule is that affix should contain at least three characters. To control if all necessary objects and controls have the affix you can use the AppSourceCop code analyzer.

To enable checking the affixes create a new file in your project - **AppSourceCop.json** and add the setup below. In the example, the **MNB** is the affix that must be added to all objects.

Code

```
{
  "mandatoryAffixes": ["MNB"]
}
```

Now when you would try to Publish (Ctrl+F5) the extension one more time, then you will see an error similar to below.



Hints

The affix which you will use in your project must be registered in Microsoft. Otherwise, if you would have got a conflict with another company then that one which has affix register wins.

To register the prefix or suffix send an email to **d365val@microsoft.com**. *It takes around one or two business days to register the affix.*

FILE NAMING

In the AL language, one of the practice is to create one file per object. In theory, there is a possibility that in one file is more than one object however, it is not common practice.

The file name can be created as:

<ObjectNameExcludingPrefix>.<ObjectTypeName>.al

In the below table, you can find the name for most common object types.

Object Type	Object Type Name
Table	Table
Table Extension	TableExt
Page	Page
Page Extension	PageExt
Report	Report
Codeunit	Codeunit
Enum	Enum
Enum Extension	EnumExt
XMLPort	Xmlport

Hints

If your extension is bigger, then you can think to put the files in the folders per object type. It allows you to control more the objects in the project.

In the table above you can find only the most common objects. Full list with examples you can find on the page https://docs.microsoft.com/en-us/dynamics365/business-central/dev-itpro/compliance/apptest-bestpracticesforalcode

CHAPTER SUMMARY

In this chapter, you found out what is your task. It will be explained in detail in the next chapters.

You know now how to use the code analyzers.

You know what is prefix/suffix and when you need to use it. Also, you know how to register the prefix for your extension.

You get familiar with how you should create new file names in the project.

Chapter 4

Tables and Pages

OBJECTIVES

In this chapter, you will develop base tables and pages for the Bonus Registration Extension. The objectives are:

- Get familiar with objects type table and page
- Understand what is Init(), Insert(), Get() adn Reset()
- Get familiar with if...else statement
- Develop the setup for the extension
- Develop the bonus card

TABLES OVERVIEW

A table is an object where you can store the data. A table in AL language contains:

- Table properties
- Set of fields
- Keys
- Global variables
- Table triggers

Each table needs to have a unique number and name. It means that in the database there cannot be two tables with the same name or the same number - even if the name is part of a different extension.

Table properties

The table properties are added for the whole object. There are four properties which you will need to know when starting development for the AL language.

Name	Description
DataClassification	It is responsible for the classification of the table in terms of GDPR. It is mandatory and should be different than **ToBeClassified.**
Caption	Caption for the table. It should not contain the prefix or suffix.
DrillDownPageId	The name of the page which will be shown when the user will use function DrillDown on the page. For example when clicking the calculated field.
LookupPageId	The name of the page which will be shown when the user will use function Lookup on the page. For example when clicking more options.

Table fields

The fields in the table can have different types. Therefore, you do not need to be worried if a field which is defined as a numeric field (integer or decimal), can store also other characters than numbers.

There is quite a long list of field types, but only a few are needed to begin the development. In the below table you can find types that are used in most cases.

The normal class fields, similar to the table, have got properties. For each field type you can, with the property Editable, decide if a field is editable or not. Two of them are mandatory - **DataClassification** and **Caption**. The caption of the field will be used as the default caption when you use that field. The **DataClassification** property is needed for GDPR and has to be different than **ToBeClassified**. In the table below you will find the properties which are good to be considered when adding the field in your extension.

Type	Description	Addtional properties
Code	It is an alphanumeric field that is automatically converted to uppercase. It is used for storing mostly unique values (very often as primary key) of the record such as customer number or code in a dictionary. It has got, in most cases, 20 or 10 characters.	Setting **NotBlank** will force you to put not empty value in the field. Adding **TableRelation** will allow you to Lookup the values from a different table.
Text	It is a text field that is used to store such things as descriptions or addresses. In most cases, it is a length of 50 characters but can store up to 2048 chars. A common practice is to create the second field for description if 50 chars are not enough.	
Date	It stores the date. Automatically it will show the calendar to choose the date.	
Integer	It is a numeric field that stores whole numbers.	Setting **BlankZero** will show empty value instead of zero. Setting **MinValue** and **MaxValue** will force you to use only values in the range.
Decimal	It is a numeric field that stores decimal numbers.	Setting **DecimalPlaces** will allows you to define the minimum and the maximum number of decimal places. You can use the same properties as with Integer type.
Boolean	It stores values 0 or 1. Automatically on the page, it is shown as switch between yes and no.	
Enum	This field type shows the list of options defined in the separate object - Enum. When creating the field you need to specify which Enum you would like to use.	

FlowFields

The **FlowFields** are a special class of fields. You can set them with the property **FieldClass**. For those fields, you cannot set **DataClassification**.

It allows to do easy mathematic operations, such as count, sum or get maximal or minimal value. It also allows you to show value from a different table (lookup) or check if the record exists. To tell what the **FlowField** should show, you can use property **CalcFormula**.

For the **FlowFields** you should always set the editable property to false.

Field triggers

The field triggers allow you to add the code directly to the field. There are two triggers assigned to the field. The first one - **OnLookup()**, allows you to add the code when a user clicks on page lookup function. However, the second one is more important. It triggers when the user will put value in the field. It is called **OnValidate()**.

Table Keys

In the table, you need to define at least one key which will be the primary key. It means that it will not be allowed to insert the same value twice in the field. You can, in an easy way, get the record in code by using the method **Get()** and specifying the value of the primary key.

You can add more than one key but only one can be set as the primary key. However, you can have got more than one field in the primary key. You can use keys to sort the data in the code for better performance optimization.

Table triggers

Four table triggers tell what code is triggered when you do one of the following things.

Operation	Trigger
Insert the record	When inserting the record, **OnInsert()** trigger will be executed.
Modify the record	When saving the modified the record, **OnModify()** trigger will be executed.
Delete the record	When deleting the record, **OnDelete()** trigger will be executed.
Change primary key	When chaniging value in any field which is in the primary key, **OnRename()** trigger will be executed.

CREATE BONUS SETUP TABLE

In some cases, there is a need to have a setup table for the extension. Your first task will be to create such a table, later you will also create a page for the table.

The setup tables have a unique structure comparing to other tables. They always store only one record and have one field as a primary key which is called Primary Key and is always blank.

In extensions, you will use a setup table to store the number of series which will be assigned to a new bonus.

Hints

In all examples in this workbook, all objects will have prefix MNB (myNAVBlog).

For workbook purposes, you can use object numbers from 50100 to 50150. The range is defined in the app.json file. For your extension, you need to ask Microsoft to assign you your range.

The snippets are predefined parts of code which will be added when you will click on it. When you will start typing **t...** *(for example* **ttable***) you will get help what are the snippets.*

By pressing **Ctrl+Space** *you can run* **IntelliSense** *which is a code-completion aid.*

Task

1. Create new file **BonusSetup.Table.al**
2. Use a snippet **ttable** to create a new table. Add table number and table name "MNB Bonus Setup"
3. Create a new field **"Primary Key"** - use type Code[10]. Remember about properties.
4. Create a new field **"Bonus Nos."** - use type Code[20]. Remember about properties.
5. To field "Bonus Nos." add **TableRelation** to **"No. Series"**
6. Make sure that the field "Primary Key" is set as primary key
7. Delete unused code

Code

```
table 50100 "MNB Bonus Setup"
{
    DataClassification = CustomerContent;
    Caption = 'Bonus Setup';
    fields
    {
        field(1; "Primary Key"; Code[10])
        {
            Caption = 'Primary Key';
            DataClassification = CustomerContent;
        }
        field(2; "Bonus Nos."; Code[20])
        {
            Caption = 'Bonus Nos.';
            DataClassification = CustomerContent;
            TableRelation = "No. Series";
        }
    }
    keys
    {
        key(PK; "Primary Key")
        {
            Clustered = true;
        }
    }
}
```

CREATE BONUS HEADER TABLE

Your bonus card will be created from two tables. The first one will be a header, where you will store the information about customer, status, starting and ending date. In the second table, you will store information about the granted bonus.

Task

1. Create new file **BonusHeader.Table.al**
2. Use a snippet **ttable** to create a new table. Add table number and table name "MNB Bonus Header"
3. Create a new field **"No."** - use type Code[20]. Remember about properties. This field will be the primary key.
4. Create a new field **"Customer No."** - use type Code[20]. Remember about properties and add **TableRelation** to **"Customer"**
5. Create a new field **"Starting Date"** - use type Date
6. Create a new field **"Ending Date"** - use type Date

Code

```
table 50101 "MNB Bonus Header"
{
    Caption = 'Bonus';
    DataClassification = CustomerContent;

    fields
    {
        field(1; "No."; Code[20])
        {
            DataClassification = CustomerContent;
            Caption = 'No.';
        }
        field(2; "Customer No."; Code[20])
        {
            DataClassification = CustomerContent;
            Caption = 'Customer No.';
            TableRelation = Customer;
        }
        field(3; "Starting Date"; Date)
        {
            DataClassification = CustomerContent;
            Caption = 'Starting Date';
        }
        field(4; "Ending Date"; Date)
        {
            DataClassification = CustomerContent;
            Caption = 'Ending Date';
        }
    }
```

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

Task

1. Create new file **BonusStatus.Enum.al**
2. Use a snippet **t**enum to create a new enum. Add number and name "MNB Bonus Status"
3. Create value 0 with Caption "Open".
4. Create value 1 with Caption "Released".

Code

```
enum 50100 "MNB Bonus Status"
{
    Extensible = true;
    value(0; Open)
    {
        Caption = 'Open';
    }
    value(1; Released)
    {
        Caption = 'Released';
    }
}
```

Task

1. Open file **BonusHeader.Table.al**
2. Add new field "Status" with Type Enum and choose enum "MNB Bonus Status"

Code

```
field(5; Status; Enum "MNB Bonus Status")
{
    DataClassification = CustomerContent;
    Caption = 'Status';
}
```

CREATE BONUS LINE TABLE

The Bonus Line table will store information about granted bonus percent. It will be connected with the header by using the Document No. field. Users will be able to put bonuses either for all items or for one particular. If a user would choose a particular item, it will be possible to choose a number from the list.

This table will have multiple fields in the primary key.

Task

1. Create new file **BonusType.Enum.al** and create new enum "MNB Bonus Type"
2. Create value 0 with Caption "All Items".
3. Create value 1 with Caption "Item".

Code

```
enum 50101 "MNB Bonus Type"
{
    Extensible = true;
    value(0; "All Items")
    {
        Caption = 'All Items';
    }
    value(1; "Item")
    {
        Caption = 'Item';
    }
}
```

Task

1. Create new file **BonusLine.Table.al**
2. Use a snippet **ttable** to create a new table. Add table number and table name "MNB Bonus Line"
3. Create a new field **"Document No."** - use type Code[20]. Remember about properties and add **TableRelation** to **"MNB Bonus Header"**
4. Create a new field **"Type"** - use type Enum. Remember about properties and add Enum "MMNB Bonus Type"
5. Create a new field **"Item No."** - use type Code[20] and add **TableRelation** to **"Item"** but only if Type is Item.
6. Create a new field **"Bonus Perc."** - use type Integer. Allow only values from 0 to 100
7. Add fields **"Document No."**, **"Type"** and to **"Item No."** to the primary key

```
table 50102 "MNB Bonus Line"
{
    DataClassification = CustomerContent;
    Caption = 'Bonus Line';

    fields
    {
        field(1; "Document No."; Code[20])
        {
            DataClassification = CustomerContent;
            Caption = 'Document No.';
            TableRelation = "MNB Bonus Header";
        }
        field(2; Type; Enum "MNB Bonus Type")
        {
            DataClassification = CustomerContent;
            Caption = 'Type';
        }
        field(3; "Item No."; Code[20])
        {
            DataClassification = CustomerContent;
            Caption = 'Item No.';
            TableRelation = if (Type = filter(Item)) Item;
        }
        field(4; "Bonus Perc."; Integer)
        {
            DataClassification = CustomerContent;
            Caption = 'Bonus Perc.';
            MinValue = 0;
            MaxValue = 100;
        }
    }

    keys
    {
        key(PK; "Document No.", Type, "Item No.")
        {
            Clustered = true;
        }
    }
}
```

PAGES OVERVIEW

On pages, users interact with data. They can insert, modify or delete data. Additionally, they can run custom actions.

A page in the AL lanugage contains:

- Page properties
- Set of controls
- Set of actions
- Global variables
- Page triggers

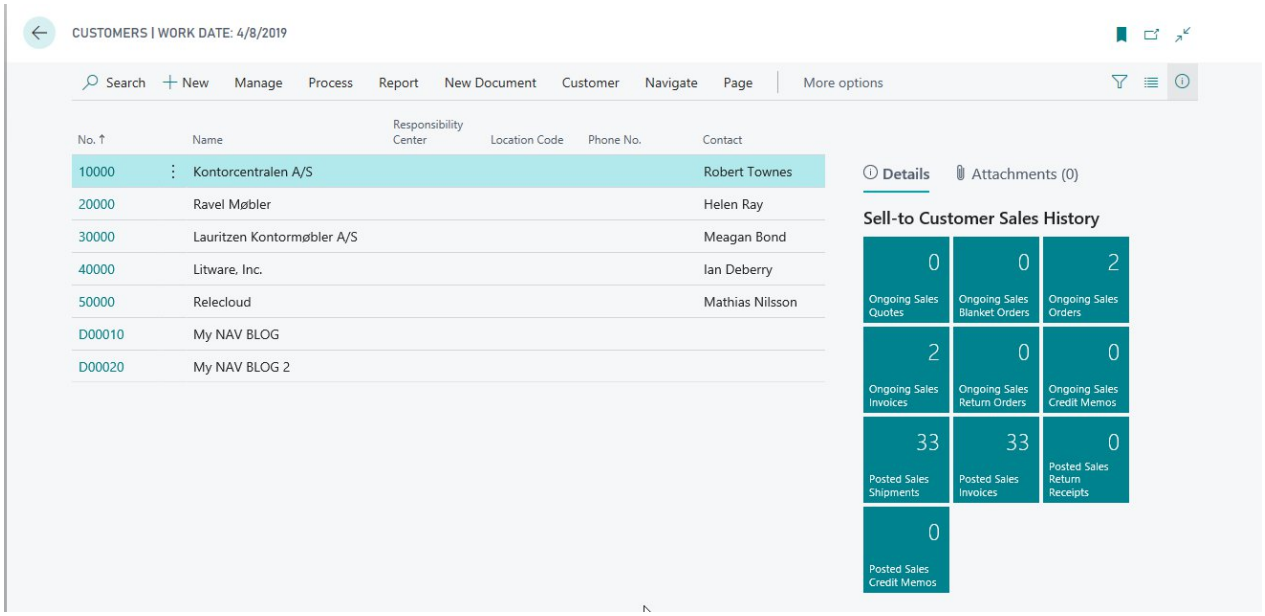
In Business Central there are different types of pages. The most important are explained below.

List page

The list page is used when more than one record needs to be shown on the page. This kind of page is, in most cases, visible from the menu or Tell Me functionality. For data such as customers, vendors or purchase orders (so data from master tables and transactional data) lists are not editable.

The lists which present dictionaries, for example, payment terms or user setup, are editable.

Example of a list page you can find below.

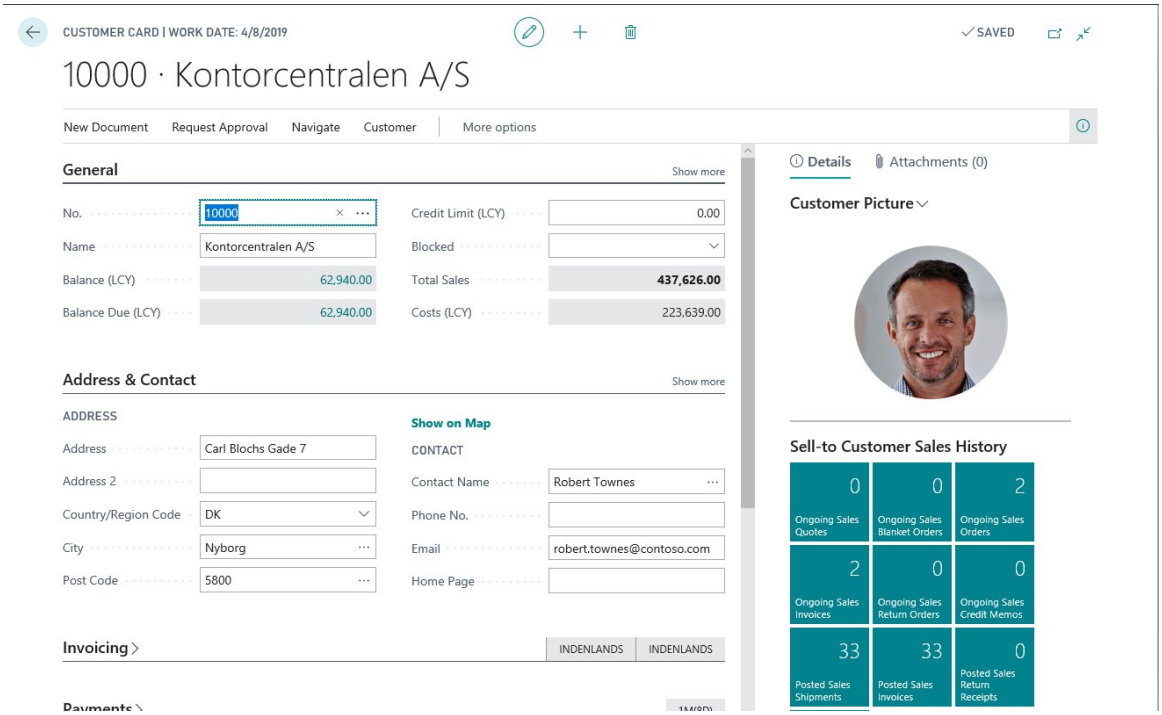


Card page

The card page is used when only one record needs to be shown on the page. This type is used, in most cases, for the master data. For example a customer, a vendor or an item. Such pages are, in general, editable but cannot be accessed from the menu. Only from the list page for particular master data.

The setup pages, such as Inventory Setup are also the card pages. Those can be accessed from the menu or Tell me functionality, are editable but a user cannot delete or insert a record directly.

Example of a card page you can find below.

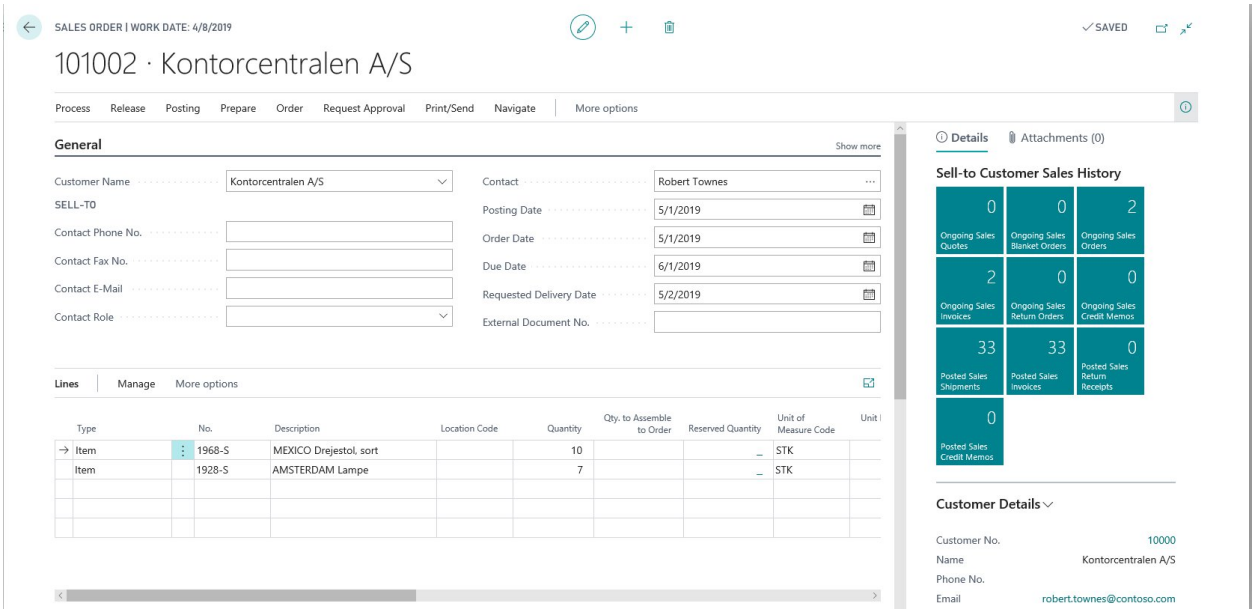


Document page

The document page is used when there should be a header and lines presented. This type is used, in most cases, for the documents such as orders, invoices, posted invoices.

In practice, such a page is created using two separated pages. The first one with type **Document** where data from the header table is presented. The second one presents lines and is created with type **ListPart**. Then such a page is added to the Document page as a part.

Example of Document page with ListPart page you can find below.



Page properties

Each page has own properties. There are common properties that can be used on all types of pages. In the below table you will find the most useful properties. Some of the properties are only for the particular type of page. If so then it is stated in the table.

Name	Description
PageType	This property defines the type of the page. The most common types you can find above.
Caption	Caption for the page. It should not contain the prefix or suffix.
SourceTable	The property describes from which table data is presented on the page. It can be only one table set per page.
UsageCategory	If the page should be visible from Tell Me functionality, this property is mandatory. Additionally, you will need to fill the **ApplicationArea** property if you want that page will be seen in the Tell Me. The **ApplicationArea** property describes in which areas of the system the page is visible.
Editable	With this property, you can tell if the page is editable or not. By default it is editable.
ContextSensitiveHelpPage	If you provide the help for the extension, you can put here the direct name of the page, which will be open to help users. This is mandatory if your extension would be published on AppSource.
DeleteAllowed, InsertAllowed, ModifyAllowed	The purpose of those three properties should be rather clear. They tell if the page should allow delete, insert or modify records. By default, those properties are set to true.
CardPageId	This property is only for pages with type list. You can decide which page will be open as a card page for the list.

Page controls

On each page in the layout section, you will find the controls. Controls are assigned to one of two areas - **Content** or **FactBox**.

In the Content area, you will find the fields which should be shown on the page. Those fields are grouped in **FastTabs** (in case of card and document pages) or in the repeater (in case of list pages).

Fields have own properties. For now, only two are very important. The first is **ApplicationArea**. It describes in which area of the system field should be shown. The second one is the **ToolTip**. It gives users basic help for the field.

It is also possible to add a part control. It will allow you to embed another page on your main page. You should put the same properties as the normal field. Also, define what is the page name, that should be added. Additionally, specify the link between the pages (for example the same document number). Examples of the parts are lines for a bonus card or a purchase order.

In the **FactBox** area, you can show parts the same way as in the Content area. **FactBox** is shown on the right side of the page. It is very common to use a FactBox to show more data about the record which is present in the main window. Normally the page which is present in the fact box has a special type of page - **CardPart**.

Page actions

On each page, you can add special actions. The actions are visible in the top part of the page. You can decide, what will happen when the user will click the action. It can be that the new page will be opened, the report will be run or some code will be triggered.

Actions are grouped in the areas. There are four areas that you can choose: **Creation**, **Navigation**, **Processing**, **Reporting**. Actions that are used to open related data, should be placed in the **Navigation** area. Those which do some action, for example posting a document, change status, should be placed in the **Processing** area. Related reports should be placed in the **Reporting** area.

Actions, the same as controls, need some properties to work correctly. In the table below you can find the properties needed for actions.

Name	Description
ApplicationArea	This property is mandatory. It says in which area of the system action will be visible.
Caption	Caption for the action. It should not contain the prefix or suffix.
Image	The property tells what is the image for the action.
Promoted	If action is set as promoted then it is shown on the Home tab. If you want that action is shown only in the **Home** tab, then set property PromotedOnly to true. When action is promoted, then you can put it in the proper category using property **PromotedCategory**. There are three standard categories: **New**, **Process**, **Report**.
RunObject	If you want to run the object, for example, another page or report, you need to specify the object type and name. The link between your page and the object can be set in the property **RunPageLink**. In case you use **RunObject** property do not write code in trigger **onAction()**.
ToolTip	ToolTip for the action is mandatory if your extension would be published on AppSource. It gives the user quick help with what the action does.

Page triggers

Similar to the table, there are few triggers on the page. Not all are used so often. That is why, in this workbook, only the most important will be described. In triggers, you can write a code that will be run when triggered.

Operation	Trigger
Open page	When opening the page, **OnOpenPage()** trigger will be executed.
Close page	When closinig the page, **OnClosePage()** trigger will be executed.
Put focus on the record	When putting focus on the record, **OnAfterGetCurrRecord()** trigger will be executed.

Hints

It is not common to put code in triggers related to insert, modify or delete the data on the page. Rather the code is put on the table directly.

It is not often to put the captions on pages. If you put the caption on the table then it will be used in all pages.

It is good practice to assign an image to action. You can see how the image looks directly in the Visual Studio Code.

It is possible to add code to fields on the pages as well. There are triggers **OnValidate()** *and* **OnLookup()** *but it is more common to put code directly on the table.*

CREATE BONUS SETUP PAGE

The Bouns Setup page will present data from the Bouns Setup table. You will create a card page. Since in setup there is only one record, the users will not be able to create or delete the record. Because of that, you will write code typical for setup pages in trigger **OnOpenPage()**.

On the page, you will create one FastTab - Numbering, which is the same as on standard setup pages. For example on the Inventory Setup page.

Task

1. Create a new file **BonusSetup.Page.al** and create new page "MNB Bonus Setup" using snippet **tpage** and choosing card page
2. Add page property for **Caption**. Additionally, add **UsageCategory** - Administration and **ApplicationArea** - All
3. Make sure that the page is using **"MNB Bonus Setup**" table as the source. And it is not allowed to insert or delete the record
4. Add new group in the **Content** area with caption Numbering
5. Add in the group field **"Bonus Nos."**. Remember about **ApplicationArea** and **ToolTip**

Code

```
page 50100 "MNB Bonus Setup"
{
    PageType = Card;
    ApplicationArea = All;
    UsageCategory = Administration;
    SourceTable = "MNB Bonus Setup";
    Caption = 'Bonus Setup';
    DeleteAllowed = false;
    InsertAllowed = false;

    layout
    {
        area(Content)
        {
            group(Numbering)
            {
                Caption = 'Numbering';
                field("Bonus Nos."; "Bonus Nos.")
                {
                    ApplicationArea = All;
                    Tooltip = 'Specifies number series what will be used for bonus numbers.';
                }
            }
        }
    }
}
```

BASIC METHODS: RESET(), INIT(), INSERT() AND GET()

When you will write the code, you will need to do operations on the records. Four first methods you can find below.

The four methods which you will use in first place, are **Reset()**, **Init()**, **Insert()** and **Get()**.

The **Reset()** method is used to remove all filters which are applied to the record variable. Thanks to that you can be sure that the set of records which you will get is not filtered in any way.

The **Init()** method you will use when you want to start writing to the database a new record.

Normally, after init and assigning values to the fields, you will use method **Insert()**. By specifying parameter in method **Insert()**, you can decide if code from OnInsert() trigger will be run. If you will leave brackets empty, then it means, that code will not be triggered.

Method **Get()** is used to get the record values from the database by using the primary key fields. In the parameters, you should add values of the fields in the same order as in the defined key. Using this method will always return you only one record.

IF...ELSE STATEMENT

The **if...else** statement is very common in any programming language. It allows you to run some code only if some condition or conditions are true. With the **else** part of the statement,you can tell what should happen if the conditions are not met. Examples you can find below.

Hints

The **else** *part is not always needed.*

You can use **and**/**or** *if there are more conditions.*

If some condition should not be true then you can use **not** *before it.*

If more than one line should be executed after if statement then put lines between **begin** *and* **end**. *If there is only one line then do not use* **begin** *and* **end**.

Code

```
if a > b then begin
...
end;

if (a > b) and not (c < b) then
   Message('Hello World');

if (a > b) and not (c < b) then
   Message('Hello World')
else begin
   ...
end;
```

ADD ONOPENPAGE( ) TRIGGER

Your Bonus Setup Page at this moment does not allow users to insert or delete the record. This is why you need to put some code in trigger **OnOpenPage()** that if the record will not be found then it should be initialized and insert with default value in the primary key.

Hints

In the setup tables, there is only one record and the value of the primary key empty. Therefore you do not need to put any value in brackets in method **Get()**. *For example, if you have variable* **SalesSetup** *which represents Sales & Receivable Setup then running* **SalesSetup.Get()** *will retrieve the record from the database.*

If you write code on the page, you do not need to specify the variable for which you are running methods. If you do not do so, then by default, it is for the current table which is defined in the source table.

Task

1. Open the file **BonusSetup.Page.al** and add new trigger OnOpenPage(). Add it at the end of the object. You can use snippet **ttrigger**.
2. Add code which will first reset all filters and then check if record does not exist.
3. If record does not exist, then init new record and insert it with empty field for primary key.

Code

```
trigger OnOpenPage()
begin
  Reset();
  if not Get() then begin
    Init();
    Insert();
  end
end;
```

Hints

If you want to see the effect of your code and how your Bonus Setup looks like, you can publish your code running **Ctrl+F5**.

CREATE BONUS LIST PAGE

The Bonuses list page will present all available, at this moment fields, from the Bonus Header table. Also, you will add new action which will open the Customer Card page for customer specified in the bonus.

Page, which you will create, will be type list and will be not editable. Later you will add the **CardPageId** to it to open the Bonus Card page.

Task

1. Create a new file **BonusList.Page.al** and create new page "MNB Bonus List" using snippet **tpage** and choosing list page
2. Add page property for **Caption**. Additionally, add **UsageCategory** - **List** and **ApplicationArea** - All
3. Make sure that page is not editable and the source table is **"MNB Bonus Header"**
4. Add all fields to the page from the table. Remember to add **ToolTip** and **ApplicationArea** for all fields.
5. Add new action **"Customer Card"** in **Navigation** area. Action should open the customer card for the customer specified in the **"Customer No."** field. To create an action you can use snippet **taction**.

Code

```
page 50101 "MNB Bonus List"
{
    PageType = List;
    Caption = 'Bonuses';
    ApplicationArea = All;
    UsageCategory = Lists;
    SourceTable = "MNB Bonus Header";
    Editable = false;

    layout
    {
        area(Content)
        {
            repeater(Control1)
            {
                field("No."; "No.")
                {
                    ApplicationArea = All;
                    ToolTip = 'Specifies the bonus number.';
                }
                field("Customer No."; "Customer No.")
                {
                    ApplicationArea = All;
                    ToolTip = 'Specifies the customer number.';
                }
                field("Starting Date"; "Starting Date")
                {
                    ApplicationArea = All;
                    ToolTip = 'Specifies the starting date.';
                }
                field("Ending Date"; "Ending Date")
                {
                    ApplicationArea = All;
                    ToolTip = 'Specifies the ending date.';
                }
                field(Status; Status)
                {
                    ApplicationArea = All;
                    ToolTip = 'Specifies the bonus status.';
                }
            }
        }
    }
```

```
    actions
    {
        area(Navigation)
        {
            action(CustomerCard)
            {
                ApplicationArea = All;
                Caption = 'Customer Card';
                Image = Customer;
                Promoted = true;
                PromotedCategory = Process;
                RunObject = page "Customer Card";
                RunPageLink = "No." = field("Customer No.");
                ToolTip = 'Opens customer card.';
            }
        }
    }
}
```

Task

1. Open the file **BonusHeader.Table.al** and assign **"MNB Bonus List"** to the **DrillDownPageId** and **LookUpPageId** properties.

Code

```
DrillDownPageId = "MNB Bonus List";
LookupPageId = "MNB Bonus List";
```

CREATE BONUS CARD PAGE

The Bonus Card page will present all available, at this moment fields, from the Bonus Header table. Also, you will add new action which will open the Customer Card page for customer specified in the bonus.

Page, which you will create, will be type car and will be editable.

Task

1. Create a new file **BonusCard.Page.al** and create a new page **"MNB Bonus Card"** using snippet **tpage** and choose a card page template. Make sure to change the type to **Document**.
2. Add page property for Caption.
3. Make sure that page source table is **"MNB Bonus Header"**
4. Create a group **General** in the **Content** area. Add all fields to it from the table. Remember to add **ToolTip** and **ApplicationArea** for all fields.
5. Copy the action **"Customer Card"** from **"MNB Bonus List"** and add into the "MNB Bonus Card".

Code

```
page 50102 "MNB Bonus Card"
{
    PageType = Document;
    SourceTable = "MNB Bonus Header";
    Caption = 'Bonus Card';

    layout
    {
        area(Content)
        {
            group(General)
            {
                Caption = 'General';
                field("No."; "No.")
                {
                    ApplicationArea = All;
                    ToolTip = 'Specifies bonus number.';
                }
                field("Customer No."; "Customer No.")
                {
                    ApplicationArea = All;
                    ToolTip = 'Specifies bonus customer number.';
                }
                field("Starting Date"; "Starting Date")
                {
                    ApplicationArea = All;
                    ToolTip = 'Specifies bonus starting date.';
                }
                field("Ending Date"; "Ending Date")
                {
                    ApplicationArea = All;
                    ToolTip = 'Specifies bonus ending date.';
                }
                field(Status; Status)
                {
                    ApplicationArea = All;
                    ToolTip = 'Specifies bonus status.';
                }
            }
        }
    }
```

```
    actions
    {
        area(Navigation)
        {
            action(CustomerCard)
            {
                ApplicationArea = All;
                Caption = 'Customer Card';
                Image = Customer;
                Promoted = true;
                PromotedCategory = Process;
                RunObject = page "Customer Card";
                RunPageLink = "No." = field("Customer No.");
                ToolTip = 'Opens customer card.';
            }
        }
    }
}
```

Task

1. Go to the file **BonusList.Page.al** and add the property **CardPageId**. Choose "MNB Bonus Card" as the card page.
2. Publish the extension and add new bonus. Check if actions are working properly.

Code

```
CardPageId = "MNB Bonus Card";
```

CREATE BONUS SUBFORM PAGE

The Bonus Card page, at this point, has only a header. You need also to add the lines to define the bonus rules. For that new page with the type **ListPart** is needed. Later you will need to add the page, as a part, to the **"MNB Bonus Card"** page.

Task

1. Create a new file **BonusSubform.Page.al** and create a new page **"MNB Bonus Subform"** using snippet **tpage** and choose a list page template. Make sure to change the type to ListPart.
2. Add page property for Caption - **Lines**.
3. Make sure that page source table is **"MNB Bonus Line"**
4. Add fields **"Type"**, **"Item No."** and **"Bonus Perc."**. Remember to add the **ApplicationArea** and **ToolTip** properties.

Code

```
page 50103 "MNB Bonus Subform"
{
    PageType = ListPart;
    SourceTable = "MNB Bonus Line";
    Caption = 'Lines';

    layout
    {
        area(Content)
        {
            repeater(Lines)
            {
                field(Type; Type)
                {
                    ApplicationArea = All;
                    Tooltip = 'Specifies type of the bonus assigned.';
                }
                field("Item No."; "Item No.")
                {
                    ApplicationArea = All;
                    Tooltip = 'Specifies item number for which bonus is assigned.';
                }

                field("Bonus Perc."; "Bonus Perc.")
                {
                    ApplicationArea = All;
                    Tooltip = 'Specifies bonus percent.';
                }
            }
        }
    }
}
```

Task

1. Open the file **BonusCard.Page.al** and in the **Content** area add new part with name **Lines**.
2. Set that the part shows page **"MNB Bouns Subform"** and add link between the header and lines that **"No."** in header is the same as **"Document No."** in the lines.
3. Remember to add **ApplicationArea** property.

Code

```
part(Lines; "MNB Bonus Subform")
{
    ApplicationArea = All;
    SubPageLink = "Document No." = field("No.");
}
```

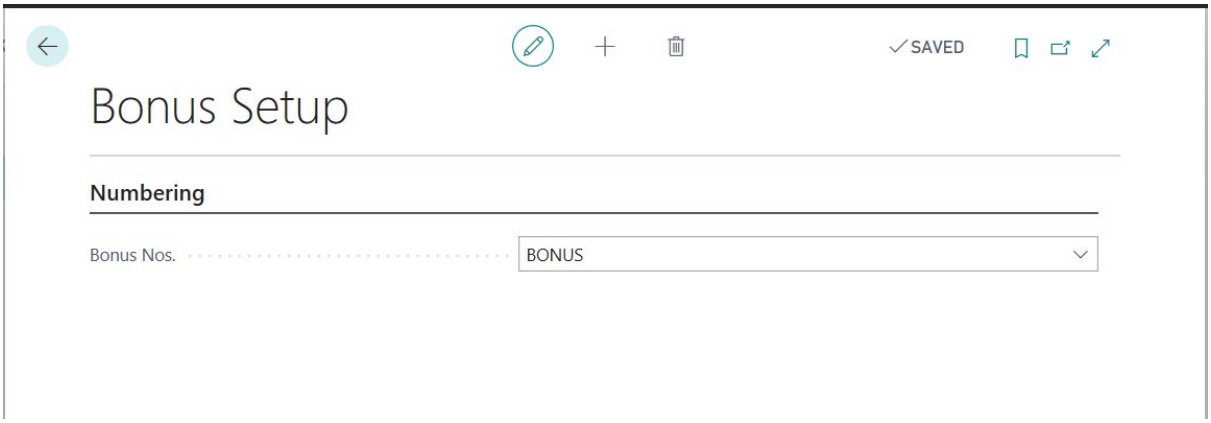
CHAPTER SUMMARY

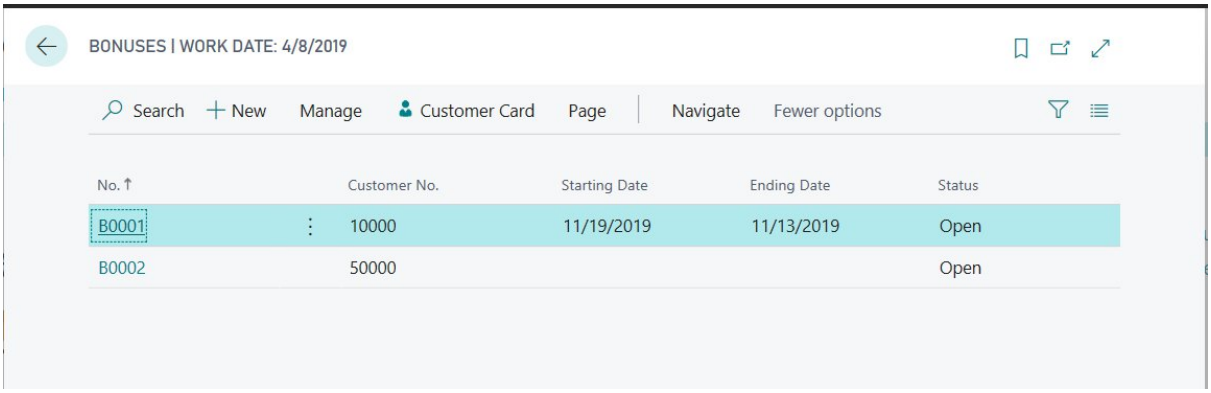
In this chapter, you understood how to develop pages and tables.

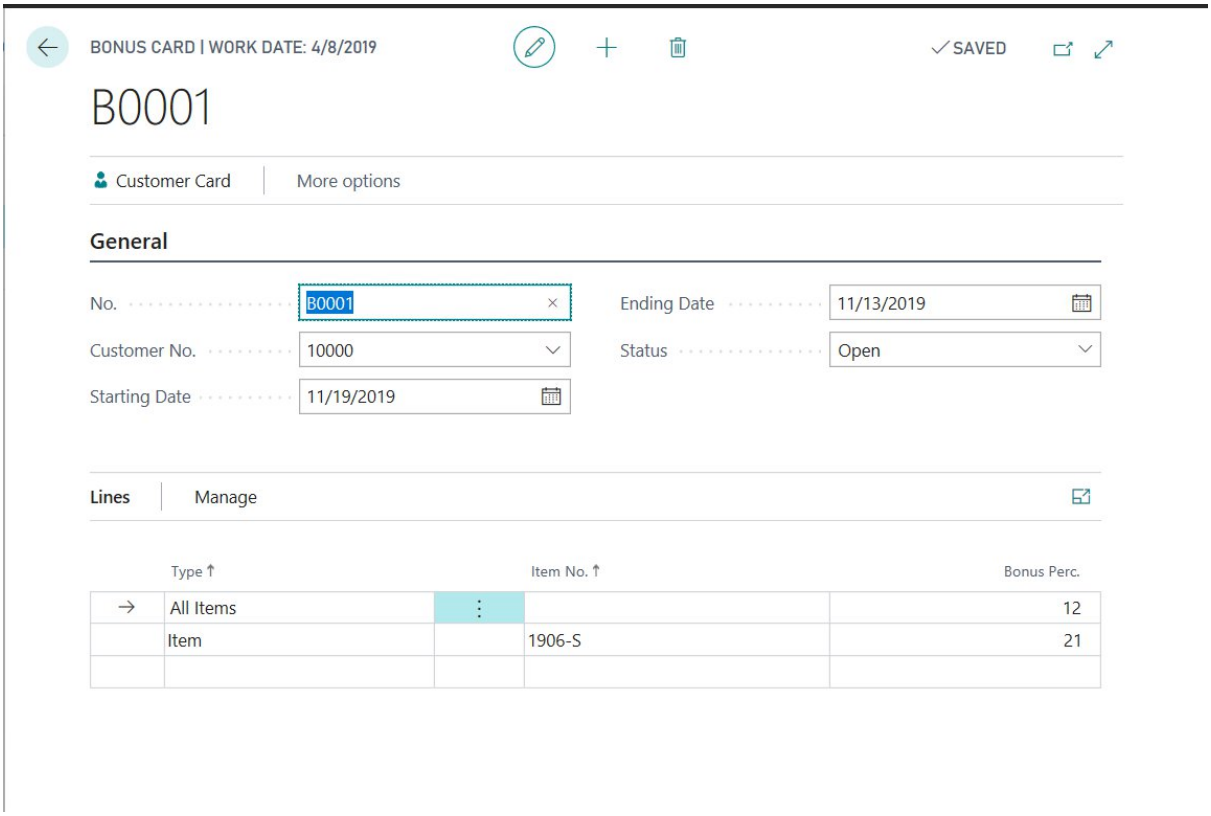
You know how to write an if...else statement.

You got familiar with the first AL language methods.

You developed the first tables and pages. How the pages look like after this chapter you can find below.







Hints

Remember to push your changes to the remote repository. You can use function **push** *from the source control menu.*

To see the development you can publish your extension to Business Central.

Chapter 5

Table Extensions and Page Extensions

OBJECTIVES

In this chapter, you will get information how to add new fields to the standard tables. Also how to show new fields on standard pages. The objectives are:

- Get familiar with objects type table extension and page extension
- Get familiar with SetFilter(), SetRange() methods
- Get familiar with FindFirst(), FindLast(),FindSet() and IsEmpty() methods
- Understand how to show messages to the user
- Develop the table extension for the Customer table
- Develop the new action for the Customer Card

TABLE EXTENSION OVERVIEW

A table extension is an object which allows you to modify the standard table. A table extension in AL language may contain:

- Properties
- Fields
- Keys
- Global variables
- Table extension triggers

Table extension properties

At this moment the table extension properties will not be described. Table extension allows for changing the standard properties. However not all properties can be changed. You can click **Ctrl+Space** in the object to check which properties can be changed.

Table extension fields

You can create fields in table extension if you need to add something to the standard object. You are doing it the same way as adding fields to the new table.

There is also possible to change the standard fields. However, only a few properties can be changed. You cannot change the code which is triggered in the standard object, but you can add your code to the standard field in one of two triggers - **OnBeforeValidate()**, **OnAfterValidate()**. The code will be trigger either before or after standard code.

Table extension keys

You can add new keys to the table extension. But you cannot mix standard fields with your new fields. You can add the keys which contain only added fields.

Table extension triggers

Similar to code for standard fields, you cannot change code for standard triggers that are in the table. However, you can add code which will be run before or after standard code. For example to add code which will be run when deleting the record you can use **OnBeforeDelete()** or **OnAfterDelete()** triggers.

PAGE EXTENSION OVERVIEW

A page extension is an object which allows you to modify the standard page. A page extension in AL language may contain:

- Properties
- Controls
- Actions
- Global variables
- Page extension triggers

Page extension properties

At this moment the page extension properties will not be described. Page extension allows for changing the standard properties. However not all properties can be changed. You can click **Ctrl+Space** in the object to check which properties can be changed.

Page extension controls

You can add new fields, created with table extension or standard fields, that are not present yet on the page. Adding the controls is very similar to adding controls to the standard page. However, you need to specify in which place the control should be added.

There is also possible to change the standard controls. However, only a few properties can be changed. You cannot change the code which is triggered in the standard object, but you can add your code to the standard control. The code will be trigger either before or after standard code.

Page extension actions

Similar to the controls, in the page extension, you can create new actions or modify existing ones. When you modify the standard action you can add the code which will be triggered after or before running the action.

Page extension triggers

In the page extension, you cannot change code for standard triggers that are on the page. However, you can add code which will be run before or after standard code.

Hints

When adding table extensions and page extensions remember to put the prefix/suffix to all fields, controls, and actions.

In your extension, you can have only one extension to the standard object. It means that for example to Customer List you can have only one page extension.

When adding new controls or actions always use **addfirst** *or* **addlast** *postions.*

GET RECORDS BESED ON FILTERS

In the previous chapter, it was described how to get the record based on the primary key. But sometimes you need to get the record or set of records based on other fields in the table.

Examples of below methods you can find at the end of this topic.

SetRange()

The **SetRange()** method allows you to add a simple filter. You can either put the range of values that you want to filter, or you can put only one value. It means that the filter will be set to one particular value.

SetFilter()

The **SetFilter()** method allows you to add a more complex filter. You can put in filter any combination of the operators such as <, >, .. , &, |, and =. In the filter string, you can use also value replacement like %1, %2 and so on.

FindFirst(), FindLast()

You can retrieve the first or last record after filtering records in the database using one of the methods: **FindFirst()** or **FindLast()**.

FindSet()

You can retrieve the set of records within the filters using **FindSet()** method. Later you can use statement **Repeat..Until** to move between records.

IsEmpty()

The **IsEmpty()** method allows you to check if there exist records within the filters. Remember that **IsEmpty()** does not retrieve the record but only values true or false.

Code

```
SalesLine.SetRange("Bill-to Customer No.", "No.");

SalesLine.SetFilter(Type, '%1|%2', SalesLine.Type::Item, SalesLine.Type::"G/L Account");
```

Hints

FindFirst() *and* **FindLast()** *also can be used with Repeat...Until statement but because of the performance of the database, it should not be used.*

If you expect that there may be no records in the set use **IsEmpty()** *method before running* **FindSet()**. *It will optimize database performance.*

You can use **FindFirst()**, **FindLast()**, **FindSet()** *and* **IsEmpty()** *as a condition in the if statement.*

MESSAGE() AND ERROR()

If you can show a message to the user, you can use method **Message()**. You can also show an error on the screen. For that, you can use method **Error()**. When you use the **Error()** method then the transaction will be stopped and rollback.

In both methods, you can use the word replacement as %1, %2, etc.

You never should hardcode the message shown on the screen. All text values should be stored in the global variables in the object. You should create a **Label** variable and assign the message or error text to it.

Examples you can find below.

Code

```
var
    HelloWorldErr: Label 'Hello World';
...
Error(HelloWorldErr);

var
    CustomerNumberMsg: Label 'Customer Number is %1';
...
Message(CustomerNumberMsg, Customer."No.");
```

EXTEND CUSTOMER LIST

The system architect just checked what you did so far and is happy. But there are new requirements:

> *On the Customer List page, the user should be able to see the number of bonuses assigned to the customer. And the user should be able to navigate to a list of bonuses assigned to the customer.*

> *It should not be possible to delete the customer if there is any bonus assigned to the customer.*

Task

1. Create a new file **Customer.TableExt.al** and create a new **table extension "MNB Customer"** using snippet **ttableext**. Make sure that it extend Customer table.
2. Add new field "MNB Bonuses" which should be integer type.
3. Change the field class to **FlowField** and add **CalcFormula**. It should count records from **"MNB Bonus Header"** table where **"Customer No."** field is the same as **"No."** from Customer table.
4. Make sure the field is not editable.

Code

```
tableextension 50100 "MNB Customer" extends Customer
{

    fields
    {
        field(50100; "MNB Bonuses"; Integer)
        {
            Caption = 'Bonuses';
            FieldClass = FlowField;
            CalcFormula = count ("MNB Bonus Header" where("Customer No." = field("No.")));
            Editable = false;
        }
    }
}
```

Task

1. Open the file **Customer.TableExt.al** and add new global variable type label.
2. Add the text to variable *'You cannot delete Customer %1 because there is at least one Bonus assigned.'*.
3. Add new trigger **OnBeforeDelete()**. Add a local variable to the trigger which will be type Record of **"MNB Bonus Header"**.
4. Add the code which will filter the "MNB Bouns Header" to check if there are any records for the customer.
5. If even one record exists then error should be shown. It should show the customer **No.** in the error.

Code

```
var
    BonusExistsErr: Label 'You can not delete Customer %1 because there is at least one Bonus assigned.';

    trigger OnBeforeDelete()
    var
        MNBBonusHeader: Record "MNB Bonus Header";
    begin
        MNBBonusHeader.SetRange("Customer No.", "No.");
        if not MNBBonusHeader.IsEmpty() then
            Error(BonusExistsErr, "No.");
    end;
```

Task

1. Create a new file **CustomerList.PageExt.al** and create a new **page extension "MNB Customer List"** using snippet **tpageext**. Make sure that it extend Customer table.
2. Add new field "MNB Bonuses" at the end of **Control1**.
3. Remember to add **ToolTip** and **ApplicationArea** properties.
4. Create new action **MNBBonuses**. Action should open the **Bonus List** page where **"Customer No."** is the same as the customer No.
5. Remember to add proper properties to the action.

Code

```
pageextension 50100 "MNB Customer List" extends "Customer List"
{
    layout
    {
        addlast(Control1)
        {
            field("MNB Bonuses"; "MNB Bonuses")
            {
                ApplicationArea = All;
                ToolTip = 'Shows number of assgined bonuses to customer.';
            }
        }
    }
```

```
    actions
    {
        addlast(navigation)
        {
            action(MNBBonuses)
            {
                Caption = 'Bonuses';
                ApplicationArea = All;
                Image = Discount;
                RunObject = page "MNB Bonus List";
                RunPageLink = "Customer No." = field("No.");
            }
        }
    }
}
```

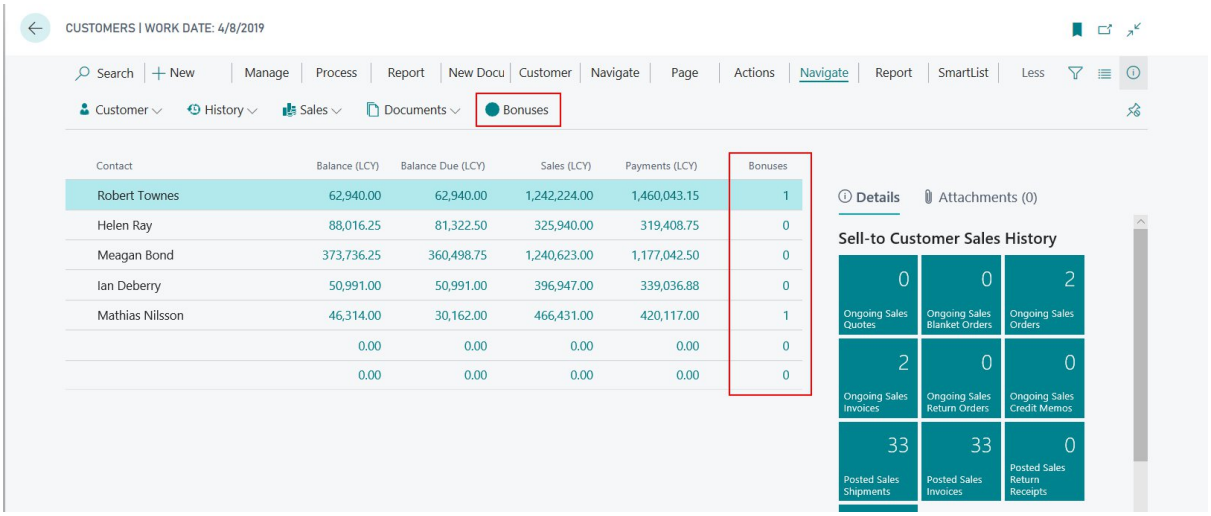
CHAPTER SUMMARY

In this chapter, you understood how to develop pages and tables extensions.

You know how to show messages and errors to the user.

You got familiar how to filter and retrieve records from the database based on assigned filters.

You developed the first table and page extension. How the Customer List looks like after this chapter you can find below.



Chapter 6

Codeunits, procedures and events subscribers

OBJECTIVES

In this chapter, you will get information on how to add procedures to objects. Also, how to create codeunits and events. The objectives are:

- Understand how to create codeunits
- Get familiar with procedures
- Get familiar with event subscribers
- Understand how to use method repeat..until
- Develop table and page to gather the bonus entries
- Develop code to calculate the bonus from sales invoices

CODEUNIT OVERVIEW

A codeunit is an object which allows you to write a code that will be executed. In the codeunit, you have one standard trigger called onRun(). This trigger, as the name says, is triggered when you run the codeunit.

Procedures

In the codeunits, you can write the procedures - which you can also name functions. At this moment you need to remember that you can create a local or a global procedure. The local one you will be able to use only in the object. Global one you can run also from different objects.

Each procedure can have parameters that you can define in the procedure header. If you want to modify a parameter in the function, you can pass by reference.

The procedure can also return some value. For that, you need to specify the type of variable which is returned. The return value you can specify in the brackets of method **exit()**.

Hints

You can also declare procedures in other types of objects.

Full description of the procedures you can write on page https://docs.microsoft.com/en-us/dynamics365/business-central/dev-itpro/developer/devenv-al-methods .

Event subscribers

In the AL language, you cannot modify the standard code of the application. Sometimes, you will see that you need to add some code when some standard code is triggered. For example, if someone posts the sales invoice you want to calculate the bonus.

To do so, you need to subscribe to an event that is published by Microsoft in the base application extension. The publishers always contain some parameters which you can use. Example of using the publisher in standard code you can find below. One publisher, to which you can subscribe, is just before and second one is right after insert the invoice line when posting sales documents.

As you can see both events allows you to retrieve such data as sales invoice header and line.

```
    end else
        if SalesShptLine.Get("Shipment No.", "Shipment Line No.") then begin
            SalesInvLine."Order No." := SalesShptLine."Order No.";
            SalesInvLine."Order Line No." := SalesShptLine."Order Line No.";
        end;
    OnBeforeSalesInvLineInsert(SalesInvLine, SalesInvHeader, xSalesLine, SuppressCommit);
    SalesInvLine.Insert(true);
    OnAfterSalesInvLineInsert(
      SalesInvLine, SalesInvHeader, xSalesLine, ItemLedgShptEntryNo, WhseShip, WhseReceive, SuppressCommit,
      SalesHeader, TempItemChargeAssgntSales);
    CreatePostedDeferralScheduleFromSalesDoc(xSalesLine, SalesInvLine.GetDocumentType,
      SalesInvHeader."No.", SalesInvLine."Line No.", SalesInvHeader."Posting Date");
end;
```

To subscribe to the publisher you can use snippet **teventsub**. Later you will need to specify the object in which the publisher is and put the publisher's name. In the procedure parameters you do not need to specify all parameters but only those which you will use. Remember only that the name of the parameters should be the same as in the publisher.

Hints

To not make a mistake in the parameters you can copy them from the publisher directly. In the object where it is published the definition of the events is most likely at the end of the object.

You can declare the event subscriber in any object but it is good to keep them in the codeunits to keep your code clean.

The name of the procedure in the subscriber event is not checked however it is good to use pattern: Run + name of the publisher. For example for publisher **OnAfterSalesInvLineInsert** *put name* **RunOnAfterSalesInvLineInsert**.

REPEAT ... UNTIL

A **repeat...until** statement gives you the possibility to run the same code in the loop until the statement which you want will not be true.

Typically, the statement is used to navigate the records. An example of it you can find below. You can filter the records before the statement to get in the loop only records which you would like to process. To check if the record is the last one in the set you can use **Next() = 0** statement.

Code

```
ItemChargeAssgntSales.Reset();
ItemChargeAssgntSales.SetRange("Document Type", "Document Type");
ItemChargeAssgntSales.SetRange("Document No.", "Document No.");
ItemChargeAssgntSales.SetFilter("Qty. to Assign", '<>0');
if ItemChargeAssgntSales.FindSet() then
  repeat
    TempItemChargeAssgntSales.Init();
    TempItemChargeAssgntSales := ItemChargeAssgntSales;
    TempItemChargeAssgntSales.Insert();
  until ItemChargeAssgntSales.Next() = 0;
```

CREATE BONUS ENTRY TABLE AND BONUS ENTRIES PAGE

To store what bonus has been calculated for the bonus card you need to prepare first the table and page where you will have the entries. In the next points, you will write the code which will insert data to this table. The table and page should not be editable. Users should be able to open the entries from the bonus card and list.

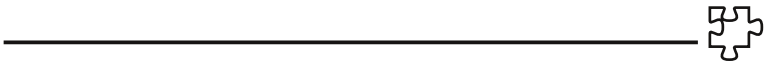
Task

1. Create a new file **BonusEntry.Table.al** and add new table **"MNB Bonus Entry"**.
2. Create fields **"Entry No."** - Integer, **"Bonus No."** - Code[20] with table relation to "MNB Bonus Header", **"Document No."** - Code[20] with table relation to "Sales Invoice Header", **"Item No."** - Code[20] with table relation to "Item", **"Posting Date"** - Date, **"Bouns Amount"** - Decimal
3. Make sure that all fields are not editable.
4. Make sure that **"Entry No."** is the primary key.

Code

```
table 50103 "MNB Bonus Entry"
{
    DataClassification = CustomerContent;
    Caption = 'Bonus Entry';

    fields
    {
        field(1; "Entry No."; Integer)
        {
            DataClassification = CustomerContent;
            Caption = 'Entry No.';
            Editable = false;
        }
        field(2; "Bonus No."; Code[20])
        {
            DataClassification = CustomerContent;
            Caption = 'Bonus No.';
            Editable = false;
            TableRelation = "MNB Bonus Header";
        }
        field(3; "Document No."; Code[20])
        {
            DataClassification = CustomerContent;
            Caption = 'Document No.';
            Editable = false;
            TableRelation = "Sales Invoice Header";
        }
        field(4; "Item No."; Code[20])
        {
            DataClassification = CustomerContent;
            Caption = 'Item No.';
            Editable = false;
            TableRelation = Item;
        }
```

```
        field(5; "Posting Date"; Date)
        {
            DataClassification = CustomerContent;
            Caption = 'Posting Date';
            Editable = false;
        }
        field(6; "Bonus Amount"; Decimal)
        {
            DataClassification = CustomerContent;
            Caption = 'Bonus Amount';
            Editable = false;
        }
    }
    keys
    {
        key(PK; "Entry No.")
        {
            Clustered = true;
        }
    }
}
```



Task

1. Create a new file **BonusEntries.Page.al** and add new **page "MNB Bonus Entries"**. It should be based on table **"MNB Bonus Entry"** and should be type **List**.
2. Make sure that it is not possible to edit the list and that user cannot do any operation on it (insert, modify and delete)
3. Add all fields from the table to the page.

Code

```
page 50104 "MNB Bonus Entries"
{
    PageType = List;
    SourceTable = "MNB Bonus Entry";
    Editable = false;
    DeleteAllowed = false;
    InsertAllowed = false;
    ModifyAllowed = false;
    Caption = 'Bonus Entries';

    layout
    {
        area(Content)
        {
            repeater(Control1)
            {
```

```
            field("Entry No."; "Entry No.")
            {
                ApplicationArea = All;
                ToolTip = 'Specifies entry number for the ledger.';
            }
            field("Bonus No."; "Bonus No.")
            {
                ApplicationArea = All;
                ToolTip = 'Specifies bonus number.';
            }
            field("Document No."; "Document No.")
            {
                ApplicationArea = All;
                ToolTip = 'Specifies sales invoice number.';
            }
            field("Item No."; "Item No.")
            {
                ApplicationArea = All;
                ToolTip = 'Specifies item number.';
            }
            field("Posting Date"; "Posting Date")
            {
                ApplicationArea = All;
                ToolTip = 'Specifies sales invoice posting date.';
            }
            field("Bonus Amount"; "Bonus Amount")
            {
                ApplicationArea = All;
                ToolTip = 'Specifies calculated bonus amount.';
            }
        }
    }
  }
}
```

Task

1. Open the file **BonusCard.Page.al** and add new action **BonusEntries**.
2. The action should open the page "MNB Bonus Entries". Only the entries for the particular bonus card should be shown.
3. Copy the action to **"MNB Bonus List"** page.

Code

```
action(BonusEntries)
{
    ApplicationArea = All;
    Caption = 'Bonus Entries';
    Image = Entry;
    Promoted = true;
    PromotedCategory = Process;
    RunObject = page "MNB Bonus Entries";
    RunPageLink = "Bonus No." = field("No.");
    ToolTip = 'Opens bonus entries.';
}
```

ASSIGNING VALUE TO THE FIELD

There are two basic methods to assign value to the field. You can assign a value directly or you can force the system to validate the field. If you will choose to validate the field it means that the code which is in the table in trigger **OnValidate()** for the field will be executed. Examples of methods you can find below.

Code

```
Customer."Credit Limit (LCY)" := 0;


Customer.Validate("Credit Limit (LCY)" , 0);
```

Hints

If you assigning the value to the fields in the tables which are posted documents or ledger entries, in most cases, you can assign value without validating it.

CREATE CODEUNIT TO CALCULATE BONUS

You will create the codeunit which will calculate and insert the bonus entries. It will be run whenever the user will post the sales invoice for the item. It needs to check if there are any bonuses for the customer in status Released within the date ranges defined in the Starting and Ending Date.

Remember that there might be more than one bonus assigned to the customer. You need to check if the bonus exists for all items and if exists for one particular item.

If you would find the match, then you should insert the new record in the **"MNB Bonus Entry"** table. Make sure that you will add the proper entry number. You will need to calculate bonus as Line Amount * Bonus Percent / 100.

Task

1. Create a new file **BonusCalculation.Codeunit.al** and create a codeunit. For that, you can use snippet **tcodeunit**. Codeunit should have the name **"MNB Bonus Calculation"**.
2. Create a new event subscriber. You can use snippet **teventsub**. Make sure to subscribe to the codeunit **"Sales-Post"** to publisher **"OnAfterSalesInvLineInsert"**.
3. Change name procedure to **RunOnAfterSalesInvLineInsert** and add only parameter - *var SalesInvLine: Record "Sales Invoice Line"*.
4. Create new local procedure CalculateBonus and put it in the event subscriber. Make sure to pass by reference the "Sales Invoice Line" record.
5. In the procedure **CalculateBonus** check if "Sales Invoice Line" has type different than Item - if yes then exit from the procedure.
6. In the procedure, **CalculateBonus** check if exists any bonus in status Released, for the same customer as "Bill-to Customer No." in the sales invoice line. Check also if the "Posting Date" of the sales invoice line is between "Starting Date" and "Ending Date".
7. If you find the bonuses (remember that can be more than one) for assigned filter then **check if in the bonus exists the line for all items** - if yes then insert the record to table "MNB Bonus Entry"
8. If you find the bonuses (remember that can be more than one) for assigned filter then **check if in the bonus exists for the particular item** - if yes then insert the record to table **"MNB Bonus Entry"**.
9. You can create a new procedure to insert the bonus to the "MNB Bonus Entry Table". Thanks to that you will have only one function to insert the data.

Code

```
codeunit 50100 "MNB Bonus Calculation"
{
    [EventSubscriber(ObjectType::Codeunit, Codeunit::"Sales-Post", 'OnAfterSalesInvLineInsert', '', true, true)]
    local procedure RunOnAfterSalesInvLineInsert(var SalesInvLine: Record "Sales Invoice Line")
    begin
        CalculateBonus(SalesInvLine);
    end;

    local procedure CalculateBonus(var SalesInvLine: Record "Sales Invoice Line")
    var
        MNBBonusHeader: Record "MNB Bonus Header";
    begin
        if SalesInvLine.Type <> SalesInvLine.Type::Item then
            exit;

        MNBBonusHeader.SetRange("Customer No.", SalesInvLine."Bill-to Customer No.");
        MNBBonusHeader.SetRange(Status, MNBBonusHeader.Status::Released);
        MNBBonusHeader.SetFilter("Starting Date", '..%1', SalesInvLine."Posting Date");
        MNBBonusHeader.SetFilter("Ending Date", '%1..', SalesInvLine."Posting Date");
        if MNBBonusHeader.IsEmpty() then
            exit;

        if MNBBonusHeader.FindSet() then
            repeat
                FindBonusForAllItems(MNBBonusHeader, SalesInvLine);
                FindBonusForOneItem(MNBBonusHeader, SalesInvLine);
            until MNBBonusHeader.Next() = 0;
    end;

    local procedure FindBonusForAllItems(var MNBBonusHeader: Record "MNB Bonus Header"; var SalesInvLine: Record "Sales Invoice Line")
    var
        MNBBonusLine: Record "MNB Bonus Line";
    begin
        MNBBonusLine.SetRange("Document No.", MNBBonusHeader."No.");
        MNBBonusLine.SetRange(Type, MNBBonusLine.Type::"All Items");
        if MNBBonusLine.FindFirst() then
            InsertBonusEntry(MNBBonusLine, SalesInvLine);
    end;
```

```
    local procedure FindBonusForOneItem(var MNBBonusHeader: Record "MNB Bonus Header"; var SalesInvLine: Record "Sales Invoice Line");
    var
        MNBBonusLine: Record "MNB Bonus Line";
    begin
        MNBBonusLine.SetRange("Document No.", MNBBonusHeader."No.");
        MNBBonusLine.SetRange(Type, MNBBonusLine.Type::Item);
        MNBBonusLine.SetRange("Item No.", SalesInvLine."No.");
        if MNBBonusLine.FindFirst() then
            InsertBonusEntry(MNBBonusLine, SalesInvLine);
    end;

    local procedure InsertBonusEntry(var MNBBonusLine: Record "MNB Bonus Line"; var SalesInvLine: Record "Sales Invoice Line")
    var
        MNBBonusEntry: Record "MNB Bonus Entry";
        EntryNo: Integer;
    begin
        if MNBBonusEntry.FindLast() then
            EntryNo := MNBBonusEntry."Entry No." + 1
        else
            EntryNo := 1;

        MNBBonusEntry.Init();
        MNBBonusEntry."Entry No." := EntryNo;
        MNBBonusEntry."Bonus No." := MNBBonusLine."Document No.";
        MNBBonusEntry."Document No." := SalesInvLine."Document No.";
        MNBBonusEntry."Item No." := SalesInvLine."No.";
        MNBBonusEntry."Posting Date" := SalesInvLine."Posting Date";
        MNBBonusEntry."Bonus Amount" := SalesInvLine."Line Amount" * MNBBonusLine."Bonus Perc." / 100;
        MNBBonusEntry.Insert();
    end;
}
```

CHAPTER SUMMARY

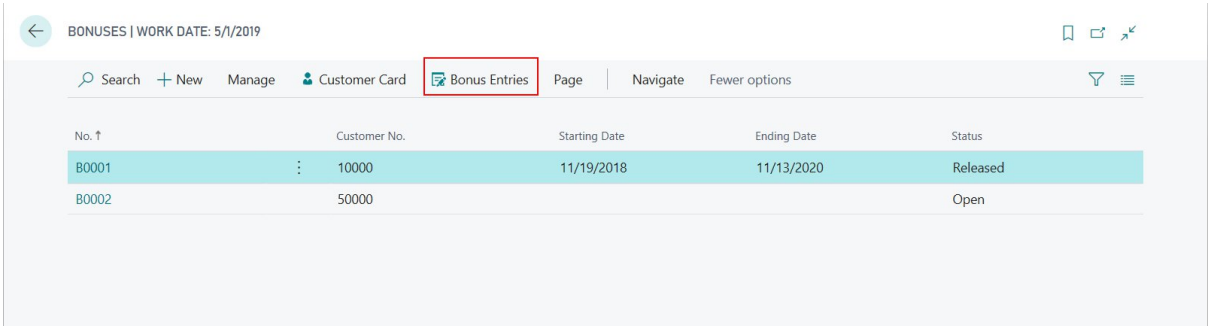
In this chapter, you understood how to develop codeunit and procedures.

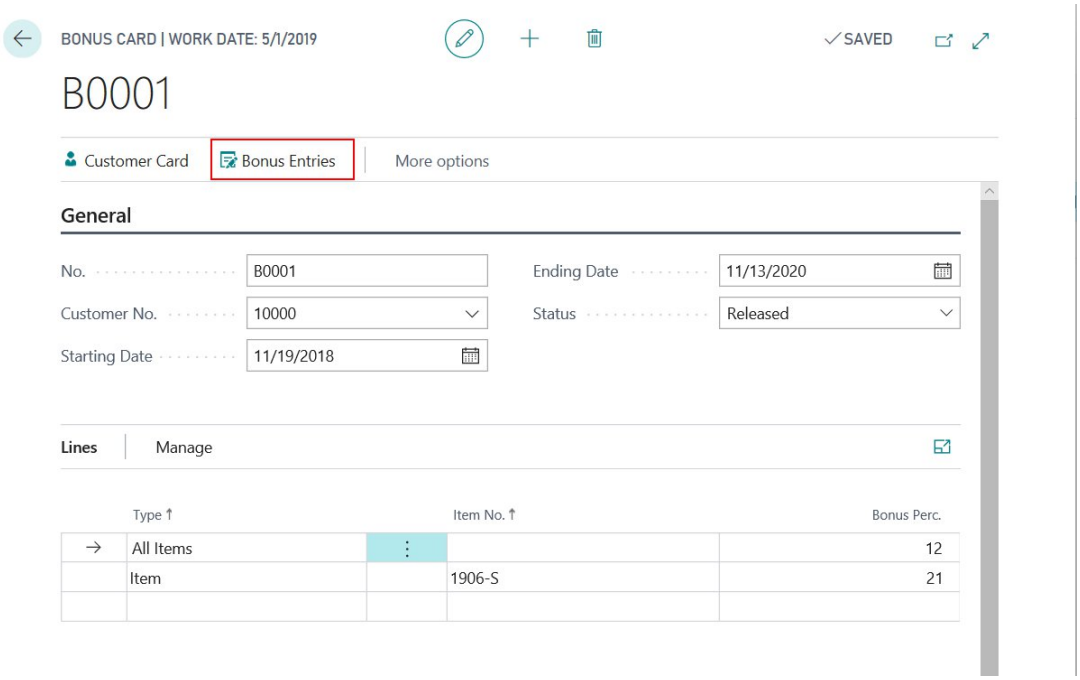
You know how to create an event subscriber to standard code.

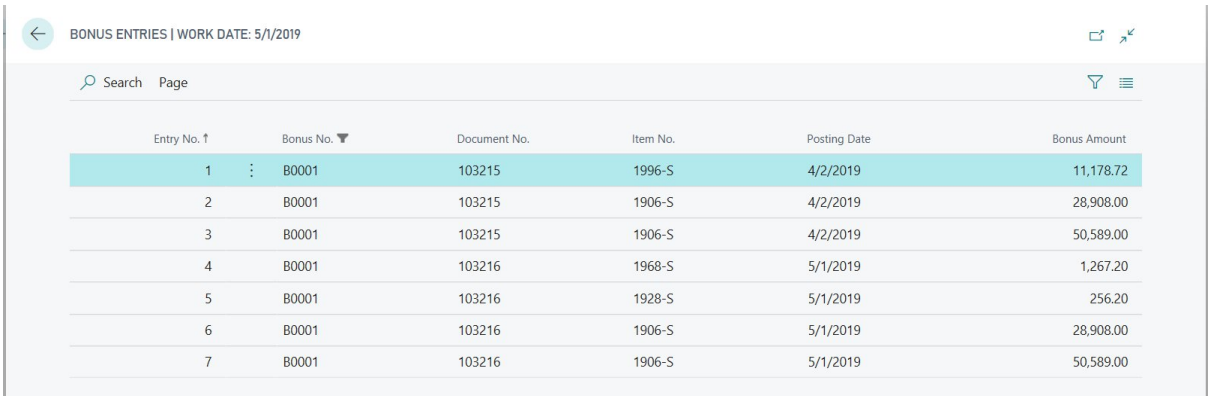
You got familiar with how to use basic loops in the code.

You developed an event that is run when the user posts the sales invoice. Now your bonuses are calculated and you can start thinking about improvements to the bonus extension.

Below you can find the added page and action in this chapter.







Chapter 7

Automated tests

OBJECTIVES

In this chapter, you will get information on how to write simple automated test for your extension. The objectives are:

- Understand why you need to write automated tests
- Get familiar with the automated test structure
- Get familiar with method TestField()
- Develop and run your first and simple automated test
- Find out where to get more information about the automated tests

Hints

For this part, to finish all tasks, you will need to have your own installation on Docker.

The examples shown in this chapter will be simple. If you would like to get more information about automated tests you can go and check the book **Automated Testing in Microsoft Dynamics 365 Business Central** *by* **Luc van Vugt**.

AUTOMATED TESTS OVERVIEW

The automated tests are the tests that can be run multiple times to check if the code which has been written is matching the requirements. The advantage of such tests is that you write them once and then you can run them when it is needed. For example to check if your functionality works with the newest version which will be available for your customers after the next upgrade.

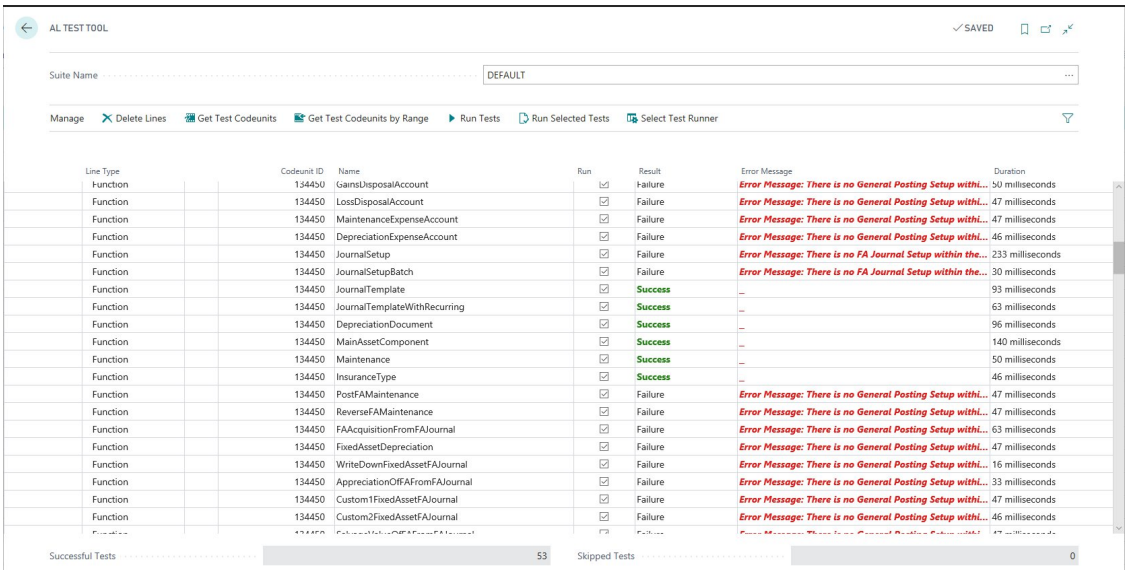
Of course, the automated tests will not replace manual testing but can be a benefit for your process of writing the extensions.

There is no external tool for automated tests in Business Central. You can write them directly in the Visual Studio Code using the same language that you use for the extension development. However, Microsoft prepared useful libraries that allow you to write tests faster.

If you plan to put your extension to the **AppSource**, then writing the automated tests is mandatory. In the validation process, you need to provide a separate application (extension) which contains only tests for your main extension.

At this moment it is not said how many tests you need to provide but covering as much as possible of your code is a beneficial for you.

You can run the automated tests directly in Business Central. For that go to the **AL Test Tool** page and get test codeunits. If you installed in your Docker container the test tool kit, then you should be able to see standard tests provided by Microsoft.



Test codeunit

To write the automated tests, you need to put them in the codeunits. You can have multiple tests in the same codeunit. This allows you to group tests for the same functionality.

It is required to set the codeunit property **Subtype** to be **Test**. Then the codeunit will be visible in the Business Central **AL Test Tool**.

Each test in the test codeunit is written as one procedure. You need to specify that the procedure is the **Test**. Only such procedures will be added to the **AL Test Tool** as test lines.

Example of the automated test with mark properties you can find below.

```
codeunit 50130 "MNB Purchase Create Documents"
{
    Subtype = Test;

    [Test]
    [TransactionModel(TransactionModel::AutoCommit)]
    procedure CheckNotPossibleToCreateOrder()
    var
        Item: Record Item;
        PurchaseHeader: Record "Purchase Header";
        PurchaseLine: Record "Purchase Line";
        MNBItemStatus: Record "MNB Item Status";
    begin
        //[Scenario] Check if not possibe to create Purchase Order
        Initialize();

        //[Given] Item with Item Status and Create Purchase Order set to false
        MNBLibraryItemStatus.CreateItemWithItemStatus(Item, MNBItemStatus);
        MNBItemStatus.Validate("Create Purchase Order", false);
        MNBItemStatus.Modify(true);

        //[When] Creating document
        asserterror LibraryPurchase.CreatePurchaseDocumentWithItem(PurchaseHeader, PurchaseLine,
        PurchaseHeader."Document Type"::"Order", '', Item."No.", LibraryRandom.RandDecInDecimalRange(1, 100, 1),
        '', LibraryRandom.RandDate(10));

        //[Then] Error is shown
        Assert.ExpectedError(StrSubstNo('%1 action is not possible for Item %2. Item Status is: %3',
            MNBItemStatus.FieldCaption("Create Purchase Order"), Item."No.", MNBItemStatus.Code));
    end;
}
```

Test structure

The automated test contains below sections:

- [Scenario]
- [Given]
- [When]
- [Then]

In the section **[Scenario]** you should describe what is the purpose of the test, for example, to check if the field is editable or if the bonus is calculated properly.

The section **[Given]** is responsible for the basic data which needs to be present in the system before executing the test. In the automated tests, you can use the provided libraries to prepare random data. You can have multiple **[Given]** sections - one for the one type of data.

In the **[When]** section you describe the action which is tested. For example, opening the page or post the sales document.

The section **[Then]** describes the expected result of the test. If the result is negative then the error is shown. To compare the expected behavior and result of **[When]** you can use the standard library **Assert**.

Hints

If you expect that there will be an error, use method **asserterror** *to catch the error.*

In the automated tests you can use any language but it is the best to use English. Then you do not need to translate the expected errors.

CHECK IF STARTING DATE IS EDITABLE IN RELEASE STATUS

It is time to write the first automated test. For that you need to create a new project in the AL Language. Fo purpose of this workbook you will create simple test which will check if it is possible to change the Starting Date after the bonus was released.

Task

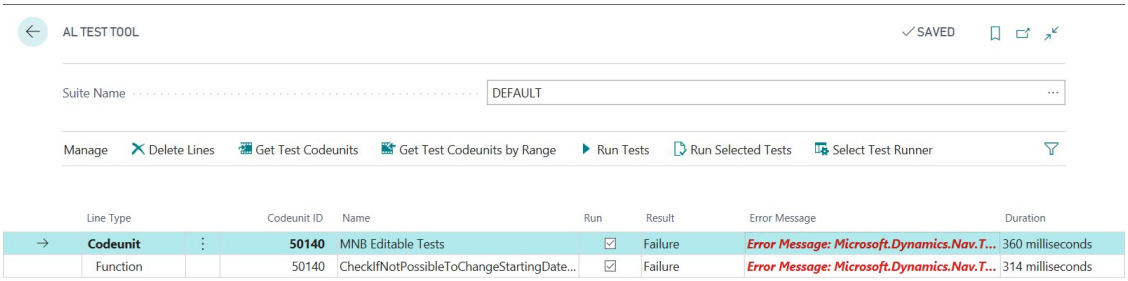
1. Create a new AL project in Visual Studio Code and remove the HelloWorld.al file
2. In the **app.json** file add dependencies to your Bonus extension that you prepared, **Library Assert**, **Tests-TestLibraries**. Remember to add the id, name, publisher, and version of the extension. All the data you can find in the **Extension Management** page.
3. Create a new file **EditableTests.Codeunit.a**l and create new codeunit. You can use snippet tcodeunit for that.
4. Add number for Codeunit 50140 and make sure that sybtype of the codeunit is Test.
5. Create new global variables in the codeunit **LibraryUtility**: Codeunit "Library - Utility", **LibraryRandom**: Codeunit "Library - Random", **Assert**: Codeunit Assert.
6. Create a new procedure and name its name (without spaces) *"Check If Not Possible To Change Starting Date In Released Status"*.
7. Add the same [Scenario] (with spaces).
8. Add [Given] - *Bouns Header Exists in status Released*.
9. In the section [Given], init new **"MNB Bonus Header"**, assign value to the "No." field. For that use function **GetGlobalNoSeriesCode()** from **"Library - Utility"** codeunit. Then assign a status to be Released and insert the record.
10. Add [When] - *Validate the Starting Date directly in the code*.
11. In the [When] section validate the Starting Date field with a random value. For that use function **RandDate()** from the **"Library - Random"** codeunit. Remember that in this place you expect the error so you need to put in the line asserterror.
12. Add [Then] - *Error is shown that you cannot change the Starting Date in Released status*.
13. In the [Then] section add function **ExpectedError()** from the **Assert** codeunit. In the expected error add *'Status must be equal to "Open"'*.

Code

```
codeunit 50140 "MNB Editable Tests"
{
    Subtype = Test;

    [Test]
    procedure CheckIfNotPossibleToChangeStartingDateInReleasedStatus()
    var
        MNBBonusHeader: Record "MNB Bonus Header";
    begin
        //[Scenario] Check If Not Possible To Change Starting Date In Released Status

        //[Given] Bouns Header Exists in status Released
        MNBBonusHeader.Init();
        MNBBonusHeader."No." := LibraryUtility.GetGlobalNoSeriesCode();
        MNBBonusHeader.Status := MNBBonusHeader.Status::Released;
        MNBBonusHeader.Insert();

        //[When] Validate the Starting Date directly in the code
        asserterror MNBBonusHeader.Validate("Starting Date", LibraryRandom.RandDate(10));

        //[Then] Error is shown that you cannot change the Starting Date in Released status
        Assert.ExpectedError('Status must be equal to "Open"');
    end;

    var
        LibraryUtility: Codeunit "Library - Utility";
        LibraryRandom: Codeunit "Library - Random";
        Assert: Codeunit Assert;
}
```

Task

1. Publish your extension and go to AL Test Tool.
2. Get the codeunit which you created and run the test.



CHECKING VALUE WITH TESTFIELD()

You can check if the field has proper value and show an error with the if statement and with the error message. This allows you to show the custom error message.

However, there is one method, that can do the same as custom code does. It is called **TestField()**. In the parameters of the method, you define the field which you testing and you can specify an expected value. If the value will be different, then the standard message will be shown.

With this method, you can also test if the field has an empty value. Then in the method parameters, you specify only the field that you are testing.

Examples you can find below.

```
Customer.TestField(Name);

Item.TestField(Blocked, false);
```

ADDING STATUS CHECK ON BONUS CARD

Your automated test at this moment is failed. It is due to that there is no check of status when you change the fields in the table **"MNB Bonus Header"**. Now you will add the check. For that you can create one function and to each field add new trigger **OnValidate()**.

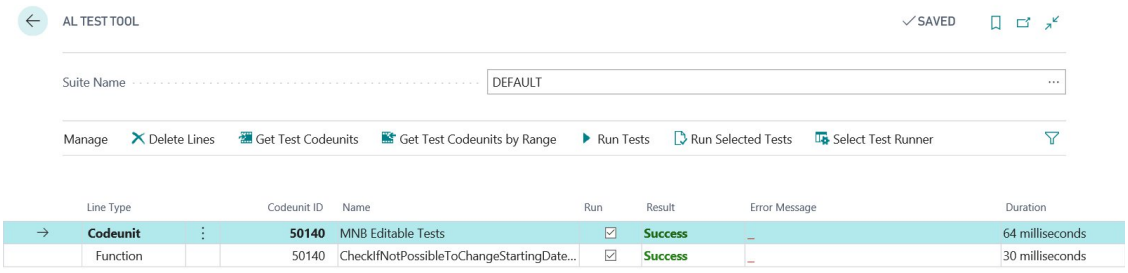
Task

1. Open the file **BonusHeader.Table.al** and create new procedure TestStatusOpen().
2. Add code whihc check if the **Status** field has got value **Open**.
3. Add trigger **OnValidate()** to each field except Status. For that you can use snippet **ttrigger**.
4. In the event add the procedure which you created.
5. Publish your extension and run the prepared automated test one more time.

Code

```
table 50101 "MNB Bonus Header"
{
    Caption = 'Bonus';
    DataClassification = CustomerContent;
    DrillDownPageId = "MNB Bonus List";
    LookupPageId = "MNB Bonus List";

    fields
    {
        field(1; "No."; Code[20])
        {
            DataClassification = CustomerContent;
            Caption = 'No.';
            NotBlank = true;
            trigger OnValidate()
            begin
                TestStatusOpen();
            end;
        }
        field(2; "Customer No."; Code[20])
        {
            DataClassification = CustomerContent;
            Caption = 'Customer No.';
            TableRelation = Customer;
            trigger OnValidate()
            begin
                TestStatusOpen();
            end;
        }
        field(3; "Starting Date"; Date)
        {
            DataClassification = CustomerContent;
            Caption = 'Starting Date';
            trigger OnValidate()
            begin
                TestStatusOpen();
            end;
        }

        field(4; "Ending Date"; Date)
        {
            DataClassification = CustomerContent;
            Caption = 'Ending Date';
            trigger OnValidate()
            begin
                TestStatusOpen();
            end;
        }
```

```
    field(5; Status; Enum "MNB Bonus Status")
    {
        DataClassification = CustomerContent;
        Caption = 'Status';
    }
    field(6; "Customer Name"; Text[50])
    {

        Caption = 'Customer Name';
        FieldClass = FlowField;
        CalcFormula = lookup (Customer.Name where("No." = field("Customer No.")));
    }
}

keys
{
    key(PK; "No.")
    {
        Clustered = true;
    }
}
procedure TestStatusOpen()
begin
    TestField(Status, Status::Open);
end;
}
```



CHAPTER SUMMARY

In this chapter, you understood how to develop the automated tests.

You know how to use TestField() method.

You added the code to check if the fields can be editable when status is Released.

Chapter 8

Multilanguage extension

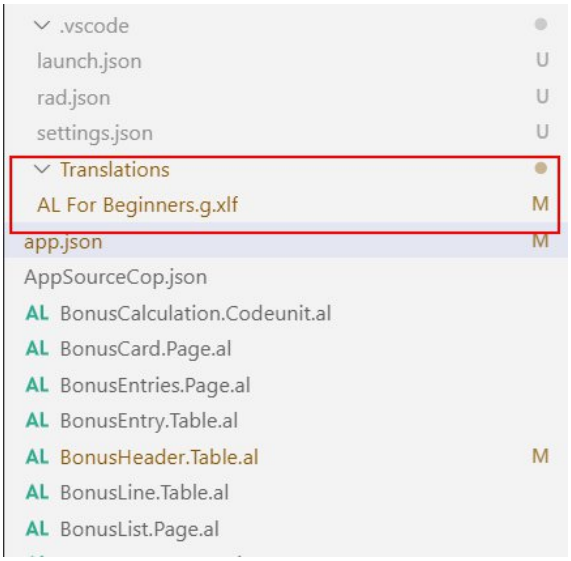
OBJECTIVES

In this chapter, you will get information on how to translate your extension. The objectives are:

- Get information how to translate the extension
- Get information which properties do not use

TRANSLATE THE EXTENSION

To translate the extension, the first step is to add in the app.json file, the feature property - **TranslationFile**. After that when you will publish the extension, automatically will be created the file with the extension **xlf**.



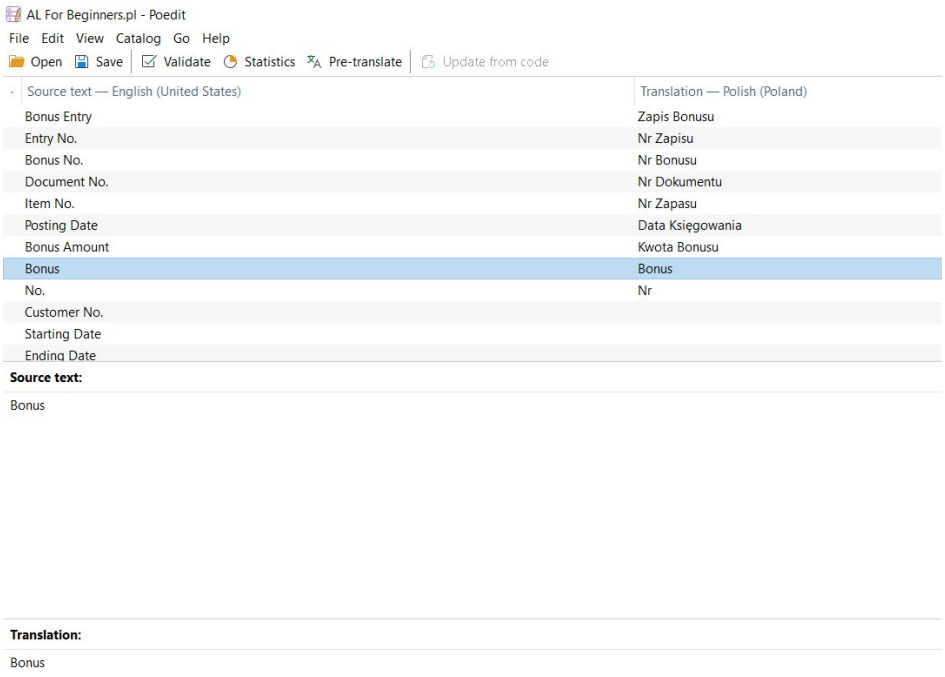
The file is Localization Interchange File Format (**XLIFF**) and has got the XML structure. Each text which you define in the **caption**, **tooltip** or **label** is automatically generated in the file.

The file in the header you can find what is the source and target language.

```
<?xml version="1.0" encoding="utf-8"?>
<xliff version="1.2" xmlns="urn:oasis:names:tc:xliff:document:1.2" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:
  <file datatype="xml" source-language="en-US" target-language="en-US" original="AL For Beginners">
    <body>
      <group id="body">
        <trans-unit id="Table 2915650444 - Property 2879900210" size-unit="char" translate="yes" xml:space="preserve">
          <source>Bonus Entry</source>
          <note from="Developer" annotates="general" priority="2"></note>
          <note from="Xliff Generator" annotates="general" priority="3">Table MNB Bonus Entry - Property Caption</note>
        </trans-unit>
        <trans-unit id="Table 2915650444 - Field 883711285 - Property 2879900210" size-unit="char" translate="yes" xml:space=
          <source>Entry No.</source>
          <note from="Developer" annotates="general" priority="2"></note>
          <note from="Xliff Generator" annotates="general" priority="3">Table MNB Bonus Entry - Field Entry No. - Property Ca
        </trans-unit>
        <trans-unit id="Table 2915650444 - Field 3518116046 - Property 2879900210" size-unit="char" translate="yes" xml:space=
          <source>Bonus No.</source>
          <note from="Developer" annotates="general" priority="2"></note>
          <note from="Xliff Generator" annotates="general" priority="3">Table MNB Bonus Entry - Field Bonus No. - Property Ca
        </trans-unit>
```

If you want to translate the file, you can find on the Internet many applications that will allow you to open the XLIFF file. One which can be recommended is **Poedit** which can be found on page www.poedit.net.

After you finish the translation put the file with the new language in the same folder as the automatically generated.



Hints

Across the AL Language, you can find the CaptionML or ToolTipML properties. They could also be used for the multilanguage functionality however you should treat them as obsolete.

If you do not want that to translate the label you can add to it the property locked.

CHAPTER SUMMARY

In this chapter, you understood how to translate the extension.

You know that you should not use CaptionML and ToolTipML properties.

Chapter 9

Reports and the Word Layout

OBJECTIVES

In this chapter, you will get overview how to develop the reports in Word. The objectives are:

- Get information how to crate a report object
- Get familiar with Word layout report
- Develop report showing the bonus entries

REPORTS OVERVIEW

A report is an object where you can present data from multiple tables. Business Central reports are used to show the data in a custom way (for example the Top 10 Customers report), but also to print documents (for example Sales Invoice).

When developing the report you can choose if the layout of the report is based on Word or RDLC template. In this chapter, you will get familiar with how to build the report using Microsoft Word. However, the structure of the report is the same for both types of layouts.

Without additional development, the user can see the preview of the report on the screen, print it or send it to one of the formats: PDF, Word or Excel (the last only if the report is RDLC).

There is also a special type of report which does not have got the layout but only processes the data in the background. For example, Calculate Depreciation.

A report in AL language contains:

- Report properties
- Dataitems with columns
- Report triggers
- Global variables
- Request page

Report properties

The report properties are added for the whole object. Below you can find the properties which you will need to know when starting development for the AL language.

Name	Description
Caption	Caption for the report. It should not contain the prefix or suffix.
DefaultLayout	With this property you can decide is the report has got Word or RDLC layout.
RDLCLayout, WordLayout	In one of this property (depends on **DefaultLayout** property) you need to specify the file with the layout.
UsageCategory	If the report should be visible from Tell Me functionality, this property is mandatory. Additionally, you will need to fill the **ApplicationArea** property if you want that report will be seen in the Tell Me. The **ApplicationArea** property describes in which areas of the system the report is visible.
ProcessingOnly	With this property, you can can decide if the report shows any layout or only process the data.

Dataitems and columns

In the **dataset** of the report, you can define the **dataitems**. Each of **dataitem** represents the set of records from the table. Inside **dataitem**, you can specify **columns** or another dataitem.

The columns in the **dataitem** represent the fields from the table. You can also put in the columns the variables which you define as globals.

If you specify the dateitem in other dataitem, the report will run it for all records in which are defined in the dataset. To link to dataitems you need to set the property **DataItemLink** where you specify how the two dataitems are connected. For example, you can use it to show for the **Sales Header** only **Sales Lines** where **Document No.** is the same as **No.** in the header.

In the table below you can find other useful properties for the dataitems.

Name	Description
RequestFilterFields	You can define the fields in which the user can define the filters. The fields are shown on the request page when open the report.
DataItemTableView	You can define what sorting and filters are set to the dataitem. If you define **DataItemTableView** but you will not define **RequestFilterFields** then the dataitem is not shown at all on the requested page. It means that the user cannot add filters for this dataitem.
PrintOnlyIfDetail	If the child dataitem does not have any records in set (is empty) then you can decide with this property if the parent dataitem should be print or not.

In the dataitem you can define three triggers. The first one is triggered before the dataitem retrieves the data. It is called **OnPreDataItem()** and is triggered only once when dataitem is accessed. You can use it for example to set additional filters with **SetRange()** or **SetFilter()** methods.

The second one, **OnAfterGetRecord()**, is triggered on each record in the set. You can use it to calculate some data. Commonly, most of the code is written in this trigger.

The last one is triggered when exiting from the dataitem. It is called **OnPostDataItem()**.

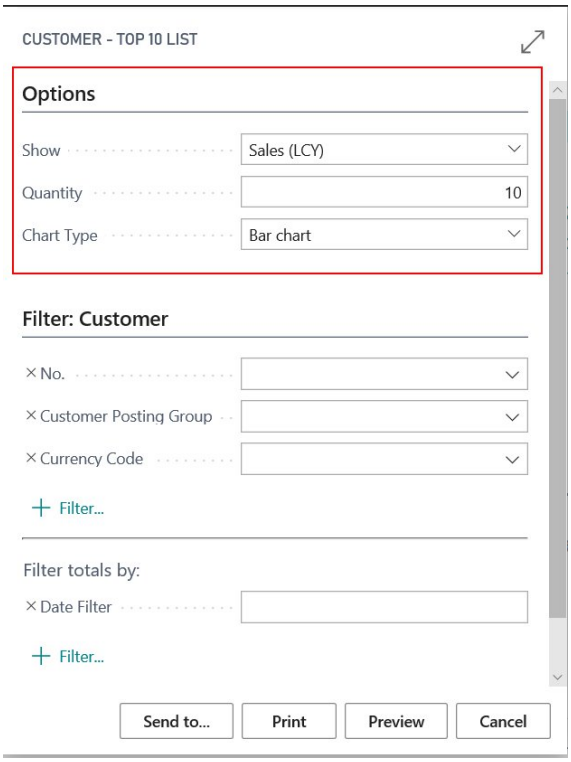
Report triggers

There three report triggers which you can use when developing the reports. The first one is called **OnInitReport()** and is triggered before the request page is open. The second one is called **OnPreReport()** and is run after closing the request page and before run the dataset. The last one is named **OnPostReport()** and is triggered after the dataset is generated.

Request page

Typically reports have got the request page. This is the special type of page that you define directly in the report. It is presented to the users when the report is opened. You can, similar to the standard page, add the controls and the triggers in the same way. However, in most cases, this page contains only global variables and is not based on any table. You can use this page as the option to run the report.

Example of the request page you can see below.



CALCSUMS()

Sometimes doing development in the AL language you need to calculate sum from records set. You can use statement **repeat...until** and add the values to one variable. However, it is not always the best option.

Much easier you can do it with method **CalcSums()**. If you have a decimal or integer field, you can set the filters on the record variable and run the method on a specific field. After that, you can assign to your global variable value of the field. Instead of value which is in one record, you will get a sum of values in the records set.

Example you can find below.

Code

```
MNBBonusEntry.SetRange("Bonus No.", "No.");
MNBBonusEntry.CalcSums("Bouns Amount");
AmountSum := MNBBonusEntry."Bouns Amount";
```

BONUS OVERVIEW REPORT

The system architect would like to get a simple report which shows the information from the header of Bonus Card such as Number, Customer No., Starting Date and Ending Date. Also, the report has to show the entries for the bonus and present the total amount of the bonus granted. You choose to create a report with the Word layout.

Task

1. Create a new file **BonusOverview.Report.al** and create a new report **"MNB Bonus Overview"**. You can use for that snippet **treport**.
2. Add the properties to the report such as the **Caption**, **UsageCategory** and **ApplicationArea**.
3. Add the dataitem which is based on the **"MNB Bonus Header"** table. Put that user see the filters for **Customer No.** and **No.** fields.
4. Add the columns to the dataitem which are for fields **No.**, **Customer No.**, **Starting Date** and **Ending Date**.
5. Add child dataitem which is based on the **"MNB Bonus Entry"**. It should be linked with the previous dataitem that it shows only records from the exact **"MNB Bonus Header"**.
6. Add **Posting Date** field as filter for **"MNB Bonus Entry"** dataitem
7. Add the columns to the dataitem which are for fields **Document No.**, **Item No.**, **Posting Date** and **Bonus Amount**.
8. Add the global variables for each caption of the field which will be present on the report. Add those labels as columns to the dataitem **"MNB Bonus Header"**.

Code

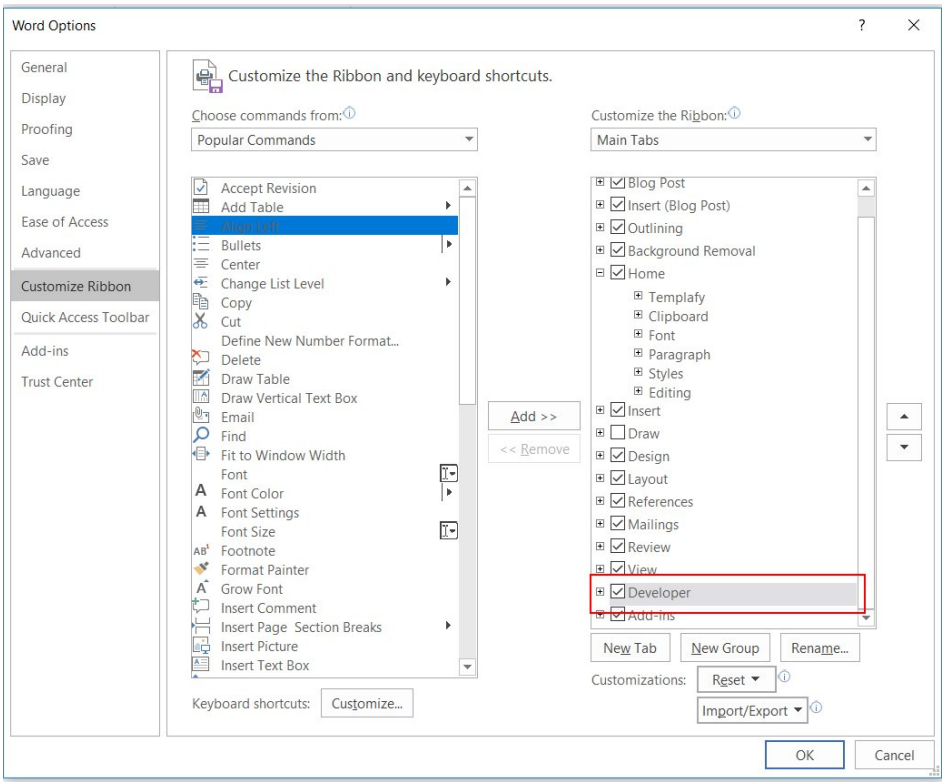
```
report 50100 "MNB Bonus Overview"
{
    UsageCategory = ReportsAndAnalysis;
    ApplicationArea = All;
    Caption = 'Bonus Overview';

    dataset
    {
        dataitem("MNB Bonus Header"; "MNB Bonus Header")
        {
            RequestFilterFields = "No.", "Customer No.";

            column(No_; "No.")
            {

            }
            column(Customer_No_; "Customer No.")
            {

            }
            column(Starting_Date; Format("Starting Date", 0))
            {

            }
```

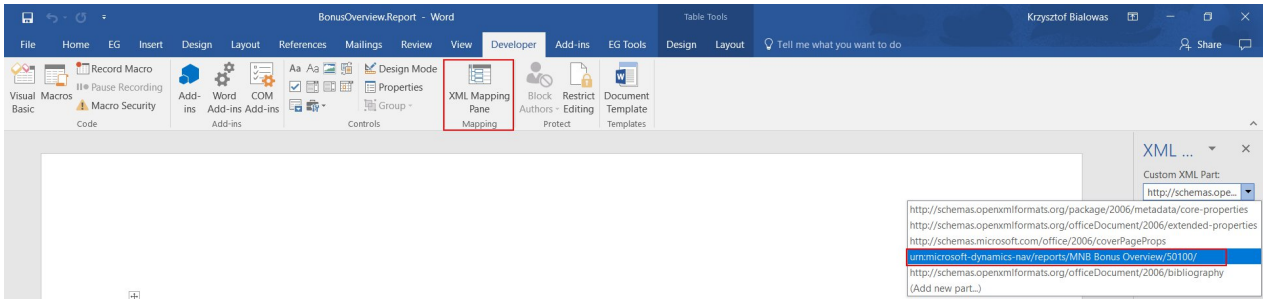
```
column(Ending_Date; Format("Ending Date", 0))
{

}
column(BonusNoCaptionLbl; BonusNoCaptionLbl)
{

}
column(CustomerNoCaptionLbl; CustomerNoCaptionLbl)
{

}
column(PostingDateCaptionLbl; PostingDateCaptionLbl)
{

}
column(DocumentNoCaptionLbl; DocumentNoCaptionLbl)
{

}
column(BonusAmountCaptionLbl; BonusAmountCaptionLbl)
{

}
column(ItemNoCaptionLbl; ItemNoCaptionLbl)
{

}
column(StartingDateCaptionLbl; StartingDateCaptionLbl)
{

}
column(EndingDateCaptionLbl; EndingDateCaptionLbl)
{

}
dataitem("MNB Bonus Entry"; "MNB Bonus Entry")
{
   DataItemLink = "Bonus No." = field("No.");
   RequestFilterFields = "Posting Date";

   column(Document_No_; "Document No.")
   {

   }
   column(Posting_Date; Format("Posting Date", 0))
   {

   }
```

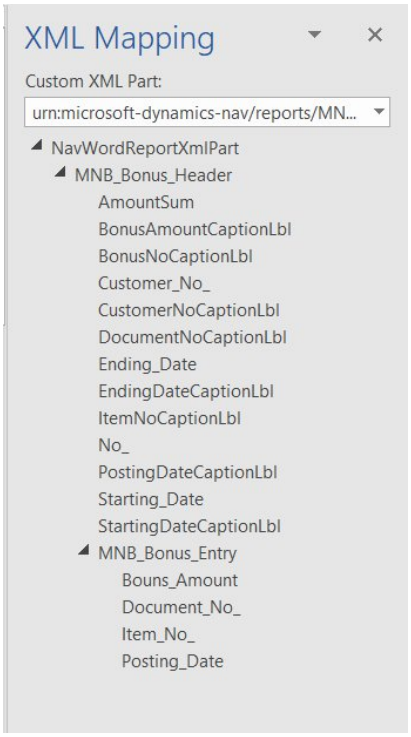
```
            column(Item_No_; "Item No.")
            {

            }
            column(Bonus_Amount; "Bonus Amount")
            {

            }
        }
    }
}
var
    BonusNoCaptionLbl: Label 'Bonus No.';
    CustomerNoCaptionLbl: Label 'Customer No.';
    PostingDateCaptionLbl: Label 'Posting Date';
    DocumentNoCaptionLbl: Label 'Document No.';
    BonusAmountCaptionLbl: Label 'Amount';
    ItemNoCaptionLbl: Label 'Item No.';
    StartingDateCaptionLbl: Label 'Starting Date';
    EndingDateCaptionLbl: Label 'Ending Date';
}
```

WORD LAYOUT

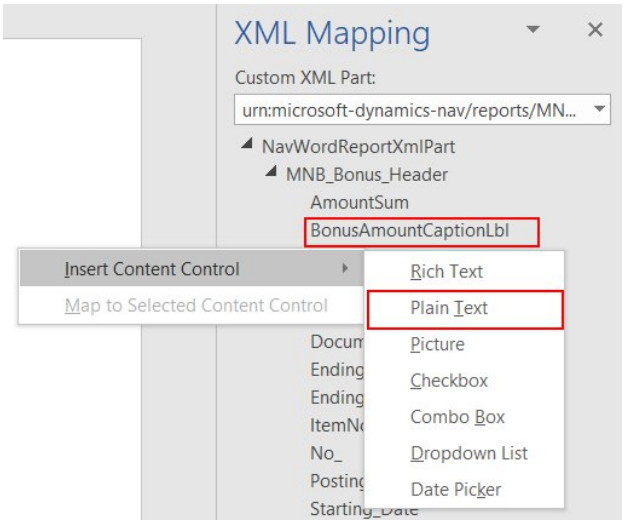
To develop reports in Word you need to activate first the tab Developer. This allows you to use the XML schema and add the fields which are generated in the AL Language.

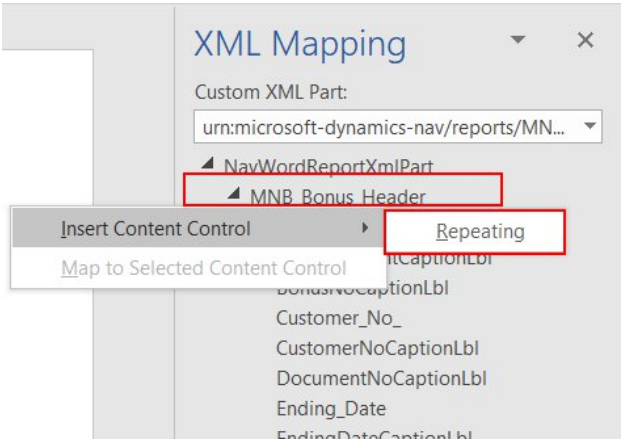






If you want to add the fields to the report you can add the field or the repeater. If you add the repeater you need to do it in the Word table. Then you will be able to show in it fields from all records.





Task

1. Open the file **BonusOverview.Report.al** and add to it that default layout is Word.
2. Add property WordLayout and put the name of the file which will contain the layout. Put value **BonusOverview.Report.docx**.
3. Open Command Pallete and run function AL: Package. It should generate the file **BonusOverview.Report.docx**.
4. Open the **BonusOverview.Report.docx** file in Word.
5. Add the one table cell to the report. It will be used as a placeholder for **"MNB Bonus Header"** if you want you can remove border for it. Mark it and add repeater for **"MNB Bonus Header"**.
6. Inside the reapeater add controls for header fields. Remember to use caption controls as caption for the value fields
7. Add the Word table inside the repeater. It should have header, one data row and a footer. It should have four columns. In a header add controls for the caption fields.
8. Mark the data row and add repeater for **"MNB Bonus Entry".** In the four columns add the controls for value fields.

Code

```
report 50100 "MNB Bonus Overview"
{

...

    DefaultLayout = Word;
    WordLayout = 'BonusOverview.Report.docx';

...
```

CustomerNoCaptionLbl: Customer_No_
BonusNoCaptionLbl: No_
StartingDateCaptionLbl: Starting_Date
EndingDateCaptionLbl: Ending_Date

DocumentNoCaptionLbl	ItemNoCaptionLbl	PostingDateCaptionLbl	BonusAmountCaptionLbl
Document_No_	Item_No_	Posting_Date	Bonus_Amount

Task

1. Open the file **BonusOverview.Report.al** and inside the dataitem for **"MNB Bonus Header"** add a new trigger **OnAfterGetRecord()**.
2. Add new global variable with type decimal - **AmountSum**.
3. In the trigger add calculation of "Bonus Amount" sum. For that create a new local variable **MNBBonusEntry** with typer record of **"MNB Bonus Entry"**.
4. Set proper filters to the variable and use method CalcSums() to calculate the sum of "Bonus Amount" field for the **"MNB Bonus Header"**. Assign the value to **AmountSum** variable.
5. Add to the dataitem for **"MNB Bonus Header"** column with the **AmountSum variable.**
6. Open Command Pallete and run function AL: Package. It should generate the file **BonusOverview.Report.docx**.
7. Open the **BonusOverview.Report.docx** file in Word.
8. Add the **AmountSum** control in the footer of the Bonus.

Code

```
trigger OnAfterGetRecord()
        var
          MNBBonusEntry: Record "MNB Bonus Entry";
        begin
          MNBBonusEntry.CopyFilters("MNB Bonus Entry");
          MNBBonusEntry.SetRange("Bonus No.", "No.");
          MNBBonusEntry.CalcSums("Bouns Amount");
          AmountSum := MNBBonusEntry."Bouns Amount";
        end;
    }
  }

...

var
  AmountSum: Decimal;
```

Hints

In the code above method **CopyFilters()** *has been used. The purpose of it is to apply all filters to the* **MNBBonusEntry** *variable which the user has chosen on the request page of the report. This way the amount will be showing the correct sum for the lines which are present on the report.*

CustomerNoCaptionLbl: Customer_No_
BonusNoCaptionLbl: No_
StartingDateCaptionLbl: Starting_Date
EndingDateCaptionLbl: Ending_Date

DocumentNoCaptionLbl	ItemNoCaptionLbl	PostingDateCaptionLbl	BonusAmountCaptionLbl
Document_No_	Item_No_	Posting_Date	Bonus_Amount
			AmountSum

CHAPTER SUMMARY

In this chapter, you get familiar how the reports are built in the AL Lanuguage

You know how to create a report and what are the types of layouts.

You developed simple report showing all entries for the bonus. Example of the report you can see below.

Customer No.: 10000
Bonus No.: B0001
Starting Date: 11/19/18
Ending Date: 11/13/20

Document No.	Item No.	Posting Date	Amount
103215	1996-S	04/02/19	11,178.72
103215	1906-S	04/02/19	28,908.00
103215	1906-S	04/02/19	50,589.00
103216	1968-S	05/01/19	1,267.20
103216	1928-S	05/01/19	256.20
103216	1906-S	05/01/19	28,908.00
103216	1906-S	05/01/19	50,589.00
			171,696.12

Customer No.: 50000
Bonus No.: B0002
Starting Date:
Ending Date:

Document No.	Item No.	Posting Date	Amount
			0.00

Chapter 10

Additional tasks for bonus extension

OBJECTIVES

In this chapter, you will find more tasks for the bonus extension. The objectives are:

- Get fill automatically Bonus No.
- Add flowfields to the Bonus Header table
- Add FactBox to list and card page

WORKDATE(), TODAY(), TIME() AND CURRENTDATETIME()

If you would like to get the date or time from the system, you can use one of the methods that are present in the AL language. The method **Date()** and **Time()** give you either current date or time. The method **CurrentDateTime()** gives you both values which you can put to field type **DateTime**.

The method **WorkDate()** is widely used across the system. It returns the date which use has set in the settings as a work date.

GET NEXT NUMBER FROM THE NO. SERIES

For now, when creating a new bonus you need to add manually the bonus No. But you created the setup to get the value from specific the number of series. Now you need to add the code which will handle this functionality.

Task

1. Open the file **BonusHeader.Table.al** and new trigger **OnInsert()**.
2. Add two variables to the trigger. On for record **"MNB Bonus Setup"** and second for the codeunit **NoSeriesManagement**.
3. Add the code that if **No.** field is empty then the record from **"MNB Bonus Setup"** is get and it is checked if **"Bonus Nos."** field has any value.
4. Use function **InitSeries** from the codeunit **NoSeriesManagement** to fill the No. Use **WorkDate() in one of the parameters** and as a "No. Series" the field **"Bonus Nos."** from the setup.

Code

```
trigger OnInsert()
var
    MNBBonusSetup: Record "MNB Bonus Setup";
    NoSeriesManagement: Codeunit NoSeriesManagement;
begin
    if "No." = '' then begin
        MNBBonusSetup.Get();
        MNBBonusSetup.TestField("Bonus Nos.");
        NoSeriesManagement.InitSeries(MNBBonusSetup."Bonus Nos.", MNBBonusSetup."Bonus Nos.", WorkDate(), "No.", MNBBonusSetup."Bonus Nos.");
    end;
end;
```

REC AND XREC

If you are in the object, for example, table or page, to get the current value of the field you just write the name of the field. It is the same as you would before the name write **Rec**."Name of the field". For example to get the "No." field you can either use "No." or Rec."No.".

You can also get value before modification. For that, you need to use **xRec** instead od Rec. For example to get the value of the "No." field before modification you can use xRec."No.".

Task

1. Open the file **BonusHeader.Table.al** and go to the trigger **OnValidate()** for the **No.** field.
2. Add two variables to the trigger. On for record **"MNB Bonus Setup"** and second for the codeunit **NoSeriesManagement**.
3. Add the code that if **No.** is changed from previous value then the record from **"MNB Bonus Setup"** is get and it is checked if **"Bonus Nos."** field has any value.
4. Use function **TestManual** from the codeunit **NoSeriesManagement** to check if the No. Series can be fill manually.

Code

```
trigger OnValidate()
var
    MNBBonusSetup: Record "MNB Bonus Setup";
    NoSeriesManagement: Codeunit NoSeriesManagement;
begin
    TestStatusOpen();
    if "No." <> xRec."No." then begin
        MNBBonusSetup.Get();
        MNBBonusSetup.TestField("Bonus Nos.");
        NoSeriesManagement.TestManual(MNBBonusSetup."Bonus Nos.");
    end;
end;
```

ADD CUSTOMER NAME TO THE BONUS HEADER

At this moment the user cannot see directly the customer name when being on the **Bonus Card**. You will add a new field **Customer Name**. This field will be a lookup field that shows the field directly from the Customer table.

To update the field you need to use method **CalcFields()**. The function you should add **OnValidate()** trigger for Customer No. field.

Task

1. Open the file **BonusHeader.Table.al** and new a new field **"Customer Name"**. Field should be type **Text[50]**.
2. Make sure that field is not editable and the class of the field is **FlowField**.
3. Add property to the field **CalcFormula** where you lookup to **Customer** table, the field **"Name"** where **"No."** is the same as the **"Customer No."** field.
4. To the field "Customer No." in the trigger **OnValidate()** add **CalcFields()** method for the field **"Customer Name"**.
5. Add the field **"Customer Name"** to **"MNB Customer Card"**.

Code

```
field(6; "Customer Name"; Text[50])
{
    Caption = 'Customer Name';
    FieldClass = FlowField;
    Editable = false;
    CalcFormula = lookup (Customer.Name where("No." = field("Customer No.")));
}

...

trigger OnValidate()
begin
    TestStatusOpen();
    CalcFields("Customer Name");
end;
```

ADD BONUS STATISTICS FACTBOX

Now to get the total bonus amount user needs to run the report or need to export and manually calculate the total bonus. You will add the field with total bonus amount to the **"MNB Bonus Header"** and then present it in the **FactBox** on the Bonus Card.

Task

1. Open the file **BonusHeader.Table.al** and add a new field **"Bonus Amount"**. Field should be type **Decimal**.
2. Make sure that field is not editable and the class of the field is **FlowField**.
3. Add property to the field **CalcFormula** where you sum field **"Bonus Amount"** from table **"MNB Bonus Entry"** where **"Bonus No."** is the same as **"No."**.
4. Create a new file **BonusStatistics.Page.al** and create new page type **CardPart**. For that you can use the snippet **tpage**. It should not has got the **UsageCategory**.
5. Make sure that page is based on **"MNB Bonus Header"** table. Add only two fields - **"No."** and **"Bonus Amount"**.
6. Open the file **BonusCard.Page.al** and add a new area **FactBoxes**.
7. Add to the area new part **"MNB Bonus Statistics"** and link part that **"No."** is the same as **"No."**.

Code

```
field(7; "Bonus Amount"; Decimal)
{
    Caption = 'Bonus Amount';
    Editable = false;
    FieldClass = FlowField;
    CalcFormula = sum ("MNB Bonus Entry"."Bouns Amount" where("Bonus No." = field("No.")));
}
```

Code

```
page 50105 "MNB Bonus Statistics"
{
    PageType = CardPart;
    SourceTable = "MNB Bonus Header";
    Caption = 'Bonus Statistics';

    layout
    {
        area(Content)
        {
            field("No."; "No.")
            {
                ApplicationArea = All;
                ToolTip = 'Specifies the bonus number.';
            }
            field("Bonus Amount"; "Bonus Amount")
            {
                ApplicationArea = All;
                ToolTip = 'Specifies the bonus totat amount';
            }
        }
    }
}
```

Code

```
area(FactBoxes)
{
    part(MNBBonusStatistics; "MNB Bonus Statistics")
    {
        ApplicationArea = All;
        SubPageLink = "No." = field("No.");
        Caption = 'Statistics';
    }
}
```

CHAPTER SUMMARY

In this chapter, you get familiarwit Rec and xRec methods.

You know how to use **WorkDate()** and **Today()** methods.

You developed the FactBox for Bonus Card. Below you can see the example of the page.