



Professional Expertise Distilled

Microsoft Dynamics NAV 2015 Professional Reporting

Discover tips and tricks for Dynamics NAV report building

Steven Renders

[PACKT] enterprise 
PUBLISHING professional expertise distilled

www.allitebooks.com

Microsoft Dynamics NAV 2015 Professional Reporting

Discover tips and tricks for Dynamics NAV
report building

Steven Renders



BIRMINGHAM - MUMBAI

Microsoft Dynamics NAV 2015 Professional Reporting

Copyright © 2015 Packt Publishing

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

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

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

First published: September 2015

Production reference: 1110915

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham B3 2PB, UK.

ISBN 978-1-78528-473-1

www.packtpub.com

Credits

Author

Steven Renders

Reviewers

Daniela Bozdoc

Alex Chow

Daniel Rimmelzwaan

Matt Traxinger

Commissioning Editor

Dipika Gaonkar

Acquisition Editors

Purav Motiwalla

Richard Brookes-Bland

Content Development Editor

Shweta Pant

Technical Editor

Saurabh Malhotra

Copy Editors

Kevin McGowan

Rashmi Sawant

Project Coordinator

Sanjeet Rao

Proofreader

Safis Editing

Indexer

Priya Sane

Graphics

Abhinash Sahu

Production Coordinator

Nitesh Thakur

Cover Work

Nitesh Thakur

About the Author

Steven Renders is a Microsoft Certified Trainer with skills that span the business and technical domains. He specializes in Microsoft Dynamics NAV and Microsoft SQL Server. He has more than 15 years of both business and technical experience. He provides training and consultancy that focuses on Microsoft Dynamics NAV, Microsoft SQL Server, business intelligence solutions, Microsoft SQL Server Reporting Services, and database performance tuning.

Furthermore, he is also an expert in Microsoft Dynamics NAV, on which he has already delivered many training sessions. He was also the author of the official Microsoft training material on Dynamics NAV reporting, development, upgrading, and SQL Server performance tuning.

He is the author of the books, *Microsoft Dynamics NAV 2015 Professional Reporting* and *Microsoft Dynamics NAV 2009: Professional Reporting* and also a reviewer of the books, *Programming Microsoft Dynamics NAV 2009*, *Programming Microsoft Dynamics® NAV 2013*, and *Implementing Microsoft Dynamics NAV 2013*.

He has also presented at various Microsoft MSDN and TechNet conferences, NAV Techdays, communities, events, and the MCT Summit.

In 2011, he started his own company, think about IT, which specializes in training and consultancy, helping companies learn, implement, understand, and solve complex business requirements related to IT, both in Belgium and abroad.

His specialties are Microsoft Dynamics NAV, Microsoft SQL Server, Business Intelligence & Reporting, and Power BI.

You can contact him at steven.renders@thinkaboutit.be and through his website (www.thinkaboutit.be). You can also view his LinkedIn profile at <http://be.linkedin.com/in/stevenrenders>, and his Twitter handle is @srenders.

Acknowledgement

There are so many people I would like to thank, who kept me motivated while I was researching and writing this second book.

First of all, a special thanks to my parents, Luc and Martine; my family, Liza, Jan, Ben, Daan, Wout, Lukas and my close friends Merlijn, Vicky, Holbe, Liesbeth, Veerle, Johan, Els, Gita, Niki, and Fynn who always stood behind me and allowed me to spend so much time apart from them.

I would like to thank the team at Packt Publishing, who deserves a lot of gratitude. It was a pleasure working with them, especially Shweta and Saurabh. They helped me a lot and guided the book in the right direction. I'm very thankful and appreciative of their help and guidance.

A big thank you to the team of reviewers (Matt, Daniel, Alex, and Daniela), who volunteered their time, knowledge, and experience by reviewing every chapter and maintaining the quality, accuracy, and flow of the book. You had a very big contribution in making this book a great piece of work that is easy to read and understand.

A special thanks to Vincent and Koen from Plataan. Many years ago, they motivated me to become a Microsoft Certified Trainer and allowed me to deepen my knowledge and experience in the Dynamics community.

Since I started my own company, think about IT, I have been lucky to have worked with a lot of very good and interesting customers, challenging projects, and different types of businesses, which have allowed me to broaden my horizons and expertise, both of which I was able to apply in this book.

I would also like to thank Microsoft and their employees for making fantastic products, such as Dynamics NAV and SQL Server, to come closer together. Both of them are great applications on their own, but combining them has been one of their biggest achievements over the last few years. The way Dynamics NAV is getting more and more integrated with other Microsoft technologies has shaped the future and opened up an almost unlimited window of possibilities and opportunities.

To all the individuals I mentioned earlier and to several colleagues, who have assisted me in one way or the other, especially in challenging me with alternative views, I feel very much indebted to you all (Roel, Steffie, Brecht, Kurt, Luc, Claus, Tarek, Mark, Conny, Frank, Anas, and Aleksandar).

I would like to thank you all!

About the Reviewers

Daniela Bozdoc is an IT professional who has a wide experience as a business analyst with a solid background as a software developer and data and software architect on various technologies. The implementation projects, especially Microsoft Dynamics NAV and Oracle EBS, have brought her excitement, new experiences, and the opportunity to meet and work with interesting people and exceed even the highest expectations.

She is a graduate from the Babes-Bolyai University of Cluj-Napoca, Romania, where she received a bachelor's degree in computer science.

She lives in Romania, where she enjoys spending time with her family and taking pictures of beautiful landscapes and natural eye-catching pieces.

Alex Chow has been working with Microsoft Dynamics NAV, formerly Navision, since 1999. Over the years, he has conducted hundreds of implementations across multiple industries. The size of businesses he has worked for range from small enterprises that earn \$2 million a year to multinational corporations that earn \$500 million a year.

Throughout his Dynamics NAV career, he has often been designated as the primary person responsible for the success and failure of a Dynamics NAV implementation. The fact that he is still in the Dynamics NAV business means that he's been pretty lucky so far. His extensive career in the Dynamics NAV business is an evidence of his success rate and expertise.

With a background in implementing all the functions and modules in and out of Microsoft Dynamics NAV, he has encountered and resolved the most practical and complex requirements and business rules. Through these experiences, he has learned that sometimes you have to be a little crazy to have a competitive edge.

He strongly believes that sharing these experiences and knowledge will benefit the Dynamics NAV community. He writes about his journey at www.dynamicsnavconsultant.com. He is also the founder of AP Commerce, Inc. (www.apcommerce.com) in 2005, a fullservice Dynamics NAV service center. In addition, he has written a book on Dynamics NAV titled *Getting Started with Dynamics NAV 2013 Application Development*.

He lives in Southern California with his beautiful wife and two lovely daughters. He considers himself the luckiest man in the world.

Daniel Rimmelzwaan was born and raised in the Netherlands and moved to the USA at the end of 1999 to be with his new American wife. In Holland, he worked as a Microsoft Access and VBA developer. While looking for a job as a VB developer in the USA, he was introduced to Navision by a "VB Recruiter" and was intrigued by the simplicity of its development tools. He decided to accept a job offer as a Navision developer with the firm intention to continue looking for a "real" developer job.

More than 15 years later, after a couple of stints in the Microsoft partner channel and a few years as a freelancer, he currently works as the chief quality officer for KCP Dynamics Group, an international partner serving customers all over the world, and he enjoys his career more than ever.

Ever since he started working with NAV, he has been an active member of the online communities for NAV, such as mibuso.com, dynamicsuser.net, and the online forums managed by Microsoft. For his contributions to these online communities, he received his first of eleven consecutive Microsoft Most Valuable Professional Awards in July 2005, which was just the second year that the MVP Award was given for NAV. Microsoft gives the MVP award to independent members of technology communities around the world and recognizes people who share their knowledge with other members of the community.

He lives with his wife and two kids in Arizona, USA.

Matt Traxinger graduated from the Georgia Institute of Technology in 2005 with a BS in computer science. After college, he took up a job as an add-on developer using a language he was unfamiliar with for a product he had never heard of—Navision. It turned out to be a great decision.

In the years that followed, he learned all the areas of the product and earned certifications in multiple technical and functional areas of Microsoft Dynamics NAV. He currently works as a development manager for ArcherPoint, a Dynamics NAV solutions provider.

In 2012, he was recognized as a Microsoft MVP and continues to be actively involved in the community, working closely with NAVUG and the Association of Dynamics Professionals to educate the next generation of NAV professionals.

www.PacktPub.com

Support files, eBooks, discount offers, and more

For support files and downloads related to your book, please visit www.PacktPub.com.

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at www.PacktPub.com and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at service@packtpub.com for more details.

At www.PacktPub.com, you can also read a collection of free technical articles, sign up for a range of free newsletters and receive exclusive discounts and offers on Packt books and eBooks.



<https://www2.packtpub.com/books/subscription/packtlib>

Do you need instant solutions to your IT questions? PacktLib is Packt's online digital book library. Here, you can search, access, and read Packt's entire library of books.

Why subscribe?

- Fully searchable across every book published by Packt
- Copy and paste, print, and bookmark content
- On demand and accessible via a web browser

Free access for Packt account holders

If you have an account with Packt at www.PacktPub.com, you can use this to access PacktLib today and view 9 entirely free books. Simply use your login credentials for immediate access.

Instant updates on new Packt books

Get notified! Find out when new books are published by following @PacktEnterprise on Twitter or the *Packt Enterprise* Facebook page.

Table of Contents

Preface	vii
Chapter 1: How Do I Start to Create a Report?	1
What is a report?	2
The request page	6
The report viewer	7
Report development phases	9
The data model phase	9
The layout phase	11
The testing phase	12
Report development tools	12
What do I use to develop the data model?	13
How do I create the report layout?	13
Built-in and custom layouts	14
Building the data model	14
Understanding the report dataset designer	16
Building the dataset	16
Data items and columns – fields, variables, and expressions	17
Including captions and labels	20
IncludeCaption versus FIELDCAPTION	21
How is the dataset flattened?	22
Report triggers	28
What happens when a report runs?	28
The report trigger sequence	29
What is a ProcessingOnly report?	30
Creating the layout	31
Visual Studio versus Report Builder	31
Creating a simple layout in Report Builder	31
Report Builder features	32
Wizards for prototyping	33
Creating a simple layout in Visual Studio	37

Visual Studio features	40
Report formatting, toolbars, and document outline	41
Building and testing the layout	42
Testing pagination and layout in different rendering extensions	43
Testing the report in different clients – Windows, Web, and tablet	43
Reporting design guidelines	43
The request page	44
The report description	45
The report creation workflow	45
Summary	47
Chapter 2: Getting Started with the Tablix	49
<hr/>	
Report items	49
Everything is a Tablix	50
The Document Outline	51
Changing the name of a Tablix	52
List versus Table versus Matrix	53
Filtering and sorting	56
How can I implement filters?	56
How can I implement sorting?	62
Interactive sorting	63
Grouping	66
How can I implement grouping?	66
Adding a parent-child group to a Tablix	66
How do I implement expand/collapse?	72
Adding an adjacent group to a Tablix	74
Formatting report items	82
Using placeholders	85
Important properties – CanGrow and CanShrink	92
Example – create an item dashboard report	93
Summary	97
Chapter 3: Expressions	99
<hr/>	
Using expressions for properties	99
The expression language	104
Simple and complex expressions	104
Symbols used in expression placeholders	106
Collections	106
Understanding the scope of an expression	108
Creating custom functions	112
Typical expression examples	117
Working with dates	117
Working with strings	120

Decision functions	122
Generating page breaks in code	126
Repeating a column header on every page	131
Example – the green-bar-matrix	135
Summary	138
Chapter 4: Data Visualization Techniques	139
<hr/>	
An introduction to data visualization	139
Recipes to implement top x filtering	139
Conditional formatting in a report	145
Analyzing your data with data bars and indicators	149
Using Sparklines to visualize trends	156
Learning how to visualize information with gauges, maps, and charts	159
Using gauges	160
Using charts	164
Using maps	170
Summary	175
Chapter 5: Document Reports	177
<hr/>	
What is a document?	177
The data model	178
Implementing multilanguage	181
Address formatting	184
Including logos	188
The No. of Copies option	191
Totaling and VAT	197
Logging and No. Printed	201
InitializeRequest	203
The layout	203
Filtering the dataset	204
Working with headers and footers	205
GetData and SetData explained	210
Alternative solutions – the mini-document	217
How do I implement page x of y?	219
Summary	222
Chapter 6: Tips and Tricks	223
<hr/>	
Report pagination	223
Show a footer or header on the last page	226
Place at the bottom	229
A fixed number of rows	235
Trans headers and footers	239
Creating links	242

Using a filter	244
Using a bookmark	248
Using the GETURL() function	250
Using internal bookmarks	252
Printing barcodes	253
Report templates	256
Using a report setup table	262
Report logging	263
The fixed header problem	265
Summary	268
Chapter 7: Performance Optimization Techniques	269
Performance recommendations	269
The dataset	269
Captions and labels	270
Remove unused columns	272
Avoid unnecessary rows	278
Report totals	279
Number formatting	281
Applying the correct filters	284
Recommendations according to the version of Dynamics NAV	287
The layout	287
Print layout versus print preview	287
Avoid conditional visibility on a big dataset	288
Best practices when visualizing information	289
Expressions in the page header or footer	291
Complex grouping and aggregate functions	292
Optimization for the chosen rendering format	292
Report design guidelines	293
Implementing hotfixes and rollout updates	293
Alternatives for building a faster dataset	294
Using a temporary table	294
Using a query object for the dataset	301
Summary	307
Chapter 8: Word Report Layouts	309
Introducing the Word report layout	309
Creating a Word report layout	311
Formatting the Word report layout	322
Repeating a table header	326
Using Word templates	327
Optimizing your dataset for Word reports	335
Managing report layouts	338
Custom layouts	339
Editing a Custom RDLC layout	341

The report execution flow	343
The Word report execution flow	344
At design time	344
At runtime	344
Managing layouts in code	344
Scheduling reports	348
Summary	350
Chapter 9: Power BI	351
<hr/>	
Dynamics NAV web services	351
Using Excel	354
Power Pivot	357
Activating Power Pivot in Excel	357
Building a Power Pivot data model	359
Importing data into Power Pivot	360
Creating relations in the Power Pivot data model	366
Power View	369
Power Map	376
Power Query	383
Power BI Designer	383
PowerBI.com	391
Summary	397
Chapter 10: Reporting Services	399
<hr/>	
What are Reporting Services?	399
Installation and configuration	400
Creating a report in SSRS	404
Using SQL Server Data Tools	415
Publishing a report project	417
Implementing reusability	419
Shared data sources and datasets	419
Shared report parts	423
Creating functions	426
Using stored procedures	428
Calling a Dynamics NAV OData web service	431
The next step	434
Caching	435
Subscribing or scheduling	436
Summary	437
Chapter 11: Charts in Dynamics NAV	439
<hr/>	
The generic chart designer	439
Text management	445
Show any list as a chart	447

Table of Contents

Business charts	449
Creating a business chart	450
Drill down your business chart	459
Preserving the user personalization	462
Implementing cues and colored indicators	463
A typical activities page	466
A typical cue table	468
Colored indicators	469
Cue style objects in Dynamics NAV	472
Summary	472
Index	473

Preface

The goal of this book is to introduce and explain the reporting capabilities of Dynamics NAV in detail. Starting from the beginning, this book will introduce you to the report designers and explain how you can create and customize reports in Dynamics NAV. The book also looks at topics in depth to explain and demonstrate the typical issues you may encounter in your daily life regarding reporting and Dynamics NAV using practical real-life scenarios.

After reading this book, you will understand how to manipulate Dynamics NAV for it to produce the reports and analytical data that you want, when you want it, and in the format you want it.

What this book covers

Chapter 1, How Do I Start to Create a Report?, explains how to create a report in Dynamics NAV. This chapter explains that the report development can be done in two steps: by creating the data model and then the layout. It also explains how to include captions and labels. Then, it dives into Visual Studio, explains how to create the layout, and demonstrates the difference between Visual Studio and Report Builder.

Chapter 2, Getting Started with the Tablix, covers how to use the Tablix control when we create the layout of the report. This chapter explains how the Tablix can be used as a List, Table, or Matrix, and demonstrates the differences between them, and also discusses when to use each. This chapter also covers the different techniques of how to filter, sort, and group information in the report layout. It also introduces you to some important properties.

Chapter 3, Expressions, discusses the expressions and how they can be used to generate values for certain properties.

Chapter 4, Data Visualization Techniques, explains that creating a report is not difficult, but making it easy to understand so that you can spot trends and learn from your data takes some consideration. The main goal of a report is to communicate the information clearly and effectively, for example, through graphical means. A report needs to create insights by communicating its key aspects in an intuitive way. In this chapter, you will learn about the different techniques available in Microsoft Dynamics NAV to visualize the information.

Chapter 5, Document Reports, explains how the RDLC report layout for documents, such as sales invoices, is created. We will explore this in detail with the most important workarounds, how and why they are required, and explore some alternative solutions.

Chapter 6, Tips and Tricks, contains tips, tricks, and useful things to know when developing reports or to speed up report development. It also contains recipes, or report design patterns, on how to show a header or footer on the last page, place it at the bottom, use a query in the dataset, optimize the report performance, create hyperlinks, reusable report components, or templates, report scheduling, and how to upgrade reports.

Chapter 7, Performance Optimization Techniques, contains tips, tricks, and recipes on how to optimize or performance tune a report.

Chapter 8, Word Report Layouts, introduces you to the built-in Word report layouts and explains how to customize them. This chapter also explains how to build a new Word layout reusing an existing Word invoice template, how to refactor and upgrade datasets for Word report layouts, and how to schedule a report to execute on the server side.

Chapter 9, Power BI, introduces you to the world of Power BI. Power BI can be used to extract data from Dynamics NAV, via ODATA web services, so that you can create BI reports in Excel, using simple pivot tables and charts, or you can make use of Power Pivot to create a more complex and optimized data model. It also covers Power View, a tool used to build interactive data visualizations on top of a Power Pivot data model. Last but not least, it also introduces Power BI in Office 365 and Q&A, a feature of Power BI in Office 365 to generate reports by simply typing in a question.

Chapter 10, Reporting Services, introduces you to the Reporting Services of SQL Server. This chapter explains how you can use reporting services, as a free report development tool, as an alternative tool to create reports on top of a Dynamics NAV database in SQL Server.

Chapter 11, *Charts in Dynamics NAV*, introduces you to the built-in chart designer in Dynamics NAV. It's frequently used by end users to create charts in Role Centers. This chapter also covers the business charts and how to customize them, as a developer.

What you need for this book

Name of the Software	Actual Name	Download link
Dynamics Nav	Microsoft Dynamics NAV 2015 R2 Management Pack for System Center	https://mbs.microsoft.com/partnersource/global/deployment/downloads/product-releases/msdnav2015download
Report Builder 2014	Microsoft® SQL Server® 2014 Report Builder	http://www.microsoft.com/en-in/download/details.aspx?id=6116
Report Builder 3	Microsoft SQL Server 2008 R2 Report Builder 3.0	http://www.microsoft.com/en-in/download/details.aspx?id=42301
Visual Studio Community edition	Visual Studio Community 2013	https://go.microsoft.com/fwlink/?LinkId=532606&clid=0x409
Microsoft Office 2013	Microsoft Office 2013	
Visual Studio Data Tools	Download Latest SQL Server Data Tools	https://msdn.microsoft.com/en-us/library/mt204009.aspx

Who this book is for

Basically, this book is for everyone who uses Microsoft Dynamics NAV or has an interest in the reporting capabilities of NAV. This book does not have a lot of prerequisites, although it mainly focuses on Dynamics NAV, RDLC, and Business Intelligence.

This does not mean that this book has no technical depth and you don't require any technical skills. On the contrary, many parts of the book will cover the technical aspects, development techniques, and reporting tools for Dynamics NAV in great detail.

If you want to get an impression of what's possible inside and outside the box of Dynamics NAV, then this book will give you a great overview. If you are interested to know how to attach other reporting or business intelligence products to Dynamics NAV, then this book will also give you an overview of these possibilities.


You might be an application developer, a power user, or a technical decision maker. Regardless of your role, I hope that you can use this book to discover the reporting features in Dynamics NAV that are most beneficial to you.

Conventions

In this book, you will find a number of styles of text that distinguish between different kinds of information. Here are some examples of these styles, and an explanation of their meaning.

Code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles are shown as follows: "The report `.rdlc` file is imported into the report object."

New terms and **important words** are shown in bold. Words that you see on the screen, in menus or dialog boxes for example, appear in the text like this: "Use the **View/Layout** menu to open the layout and create the RDLC file."

[ Warnings or important notes appear in a box like this.]

[ Tips and tricks appear like this.]

Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book – what you liked or may have disliked. Reader feedback is important for us to develop titles that you really get the most out of.

To send us general feedback, simply send an e-mail to feedback@packtpub.com, and mention the book title via the subject of your message.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide on www.packtpub.com/authors.

Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

Downloading the color images of this book

We also provide you with a PDF file that has color images of the screenshots/diagrams used in this book. The color images will help you better understand the changes in the output. You can download this file from http://www.packtpub.com/sites/default/files/downloads/4731EN_ColorImages.pdf.

Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books – maybe a mistake in the text or the code – we would be grateful if you would report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting <http://www.packtpub.com/submit-errata>, selecting your book, clicking on the **errata submission form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded on our website, or added to any list of existing errata, under the Errata section of that title. Any existing errata can be viewed by selecting your title from <http://www.packtpub.com/support>.

Piracy

Piracy of copyright material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works, in any form, on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at copyright@packtpub.com with a link to the suspected pirated material.

We appreciate your help in protecting our authors, and our ability to bring you valuable content.

Questions

You can contact us at questions@packtpub.com if you are having a problem with any aspect of the book, and we will do our best to address it.

1

How Do I Start to Create a Report?

In most implementations, the focus is on customizing the Dynamics NAV application to meet the needs of the organization and sometimes also future needs. The effort that is required for reporting in general is often underestimated and unfortunately assigned to the least experienced consultants, who have to create/adapt document reports according to customer requests.

Personally, I believe reporting is one of the most important aspects of an implementation. It should therefore be given importance from the outset, in the analysis phase of the project. The kind of information you want to retrieve from your ERP system and the way you want to retrieve this information has a big impact on the implementation of the system. Doing this correctly at the beginning of a project can, and will, save a lot of time, money, and frustration. The unfortunate reality is that many partners and/or customers look at reporting first when they want to reduce the cost of an ERP implementation project.

This chapter is an introduction to creating reports in Dynamics NAV.

I will start by stating what a report is, and how standard Dynamics NAV includes all sorts of reports. Then, I will explain that report development is always done in three steps: creating the data model, then the layout and, last but not least, testing the report.

When creating the data model, I will guide the user and explain how to create a dataset, starting with a simple dataset consisting of one data item and then make it more complex by introducing multiple data items and explaining/demonstrating the effects on the dataset of the way you build data items.

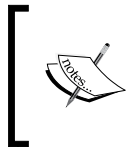
I will also explain how to include captions and labels. Then, I will dive into Visual Studio and explain how to create the layout. I will also explain and demonstrate the difference between Visual Studio and Report Builder.

What is a report?

Reports have several purposes in Dynamics NAV. The purpose of a report is to print or visualize information from a database in an intuitive and structured way. For example, a report could be a list of customers, vendors or items, or a combination of customers and items sold.

Some reports are used to communicate with third parties. These reports are called **document reports**. Examples of document reports are sales invoices, credit memos, and so on. For every document in the application, a document report is also created.

Apart from printing information, some reports have no layout and are used only to process information. These reports are considered batch jobs or processing-only reports. You can compare them to code units but with the advantages of the report dataset designer and request page options.

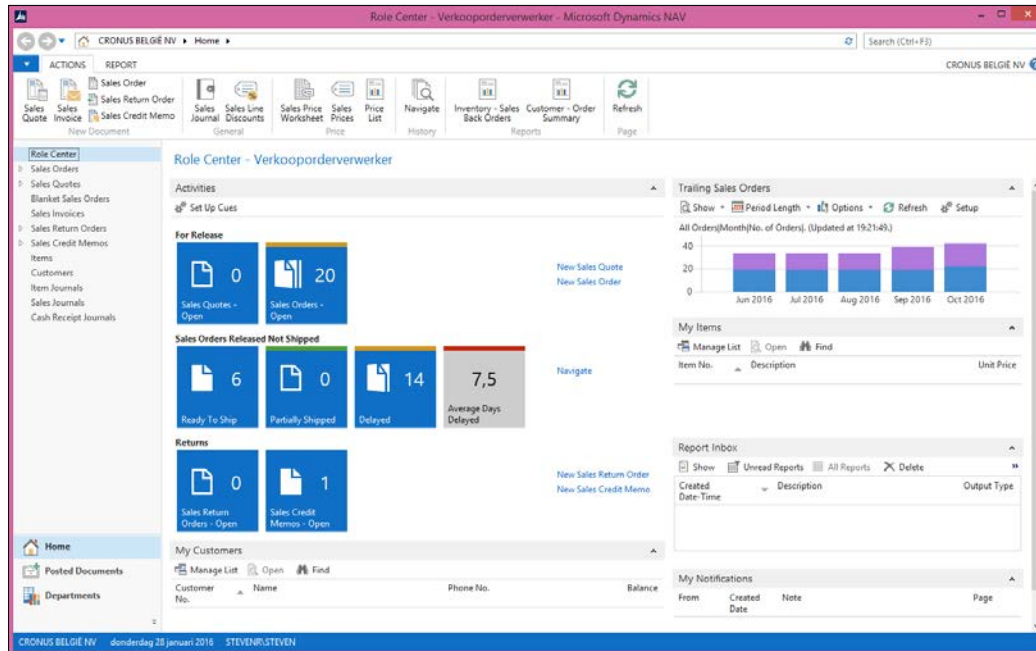


There are many different types of reports in standard NAV. For a more comprehensive listing of the standard reports, please have a look at:

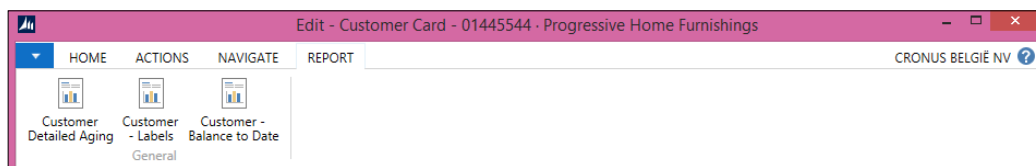
[https://msdn.microsoft.com/en-us/library/hh174014\(v=nav.80\).aspx](https://msdn.microsoft.com/en-us/library/hh174014(v=nav.80).aspx)

From the users' point of view, everything starts with the Dynamics NAV application and their experience will vary depending on the **Role Center** they are assigned to. Dynamics NAV is all about the RoleTailored Client and the RoleTailored Client always opens with a **Role Center** page. A **Role Center** is like a dashboard. It is the starting page in Dynamics NAV and on it you will find the links to all the information you need.

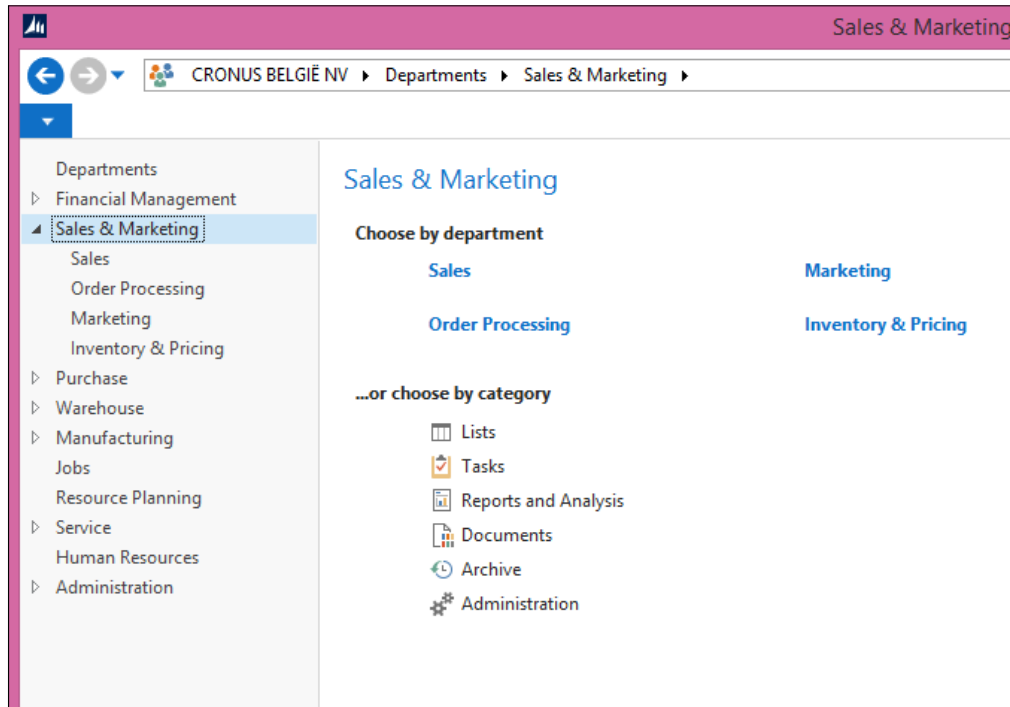
This is an example of the **Role Center** for the **Order Processor** role:



Depending on the **Role Center** you are on, the ribbon will contain different reports. You can also access reports on other pages. For example, you can access reports in the menu at the top of the window on the **Customer Card** page:



When you go to **Departments** in the RTC, you can go to any department, for example, **Sales & Marketing**. When you click on **Department**, different categories appear in the content window:



Click on **Reports and Analysis** and you get an overview of all the reports that are related to **Sales & Marketing**. As you can see, the reports are divided into different groups.

These groups correspond to the groups defined in the menu item as defined in the MenuSuite designer for the **Sales** and **Marketing** menu. In each group, you will find links to reports:

Sales & Marketing, Reports and Analysis

Analysis & Reporting
Sales Budgets
Sales Analysis Reports
Sales Analysis by Dimensions
Production Forecast
Item Dimensions - Detail
Item Dimensions - Total

Sales

Reports

Contacts
Contact List
Contact - Company Summary
Contact - Person Summary
Contact Labels
Questionnaire - Handouts
Questionnaire - Test

Customers
Customer List
Customer Register
Customer - Order Summary
Customer - Order Detail
Customer Labels
Customer Top 10 List
Customer/Item Sales
Sales List
Customer Balance to Date
Customer Trial Balance

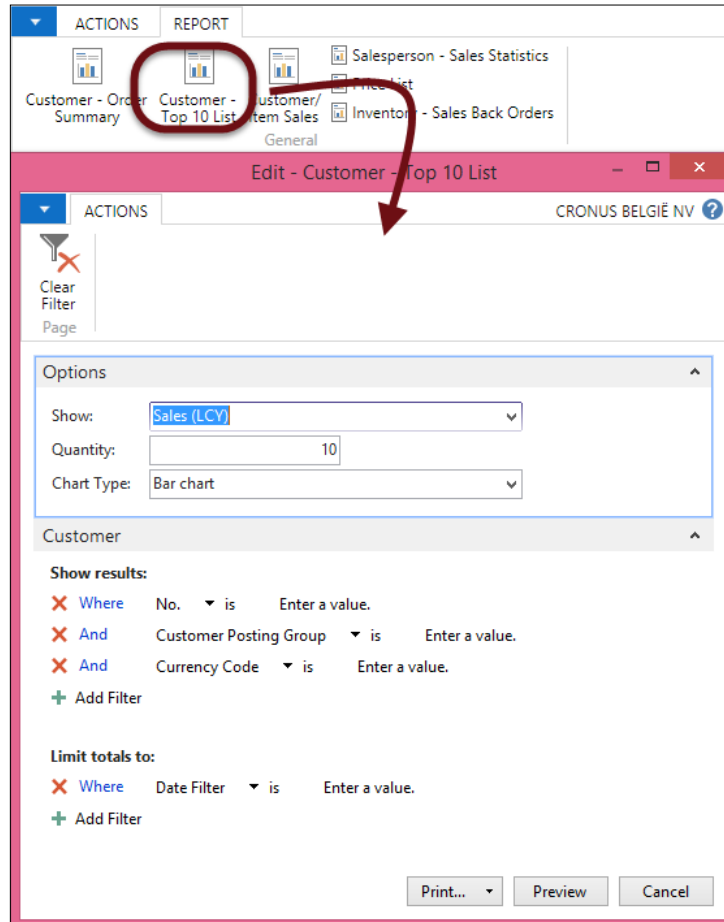
Salespeople/Teams
Sales Statistics
Salesperson Sales Statistics
Salesperson Commission
Salesperson To-dos
Salesperson Opportunities
Team To-dos

Opportunities
Opportunity - List
Sales Cycle - Analysis
Opportunity - Details

This kind of classification of reports is available in every section of the **Departments** suite in the RTC.

The request page

The first thing you see when you run a report, for example the **Customer Top 10 List**, is the request page:



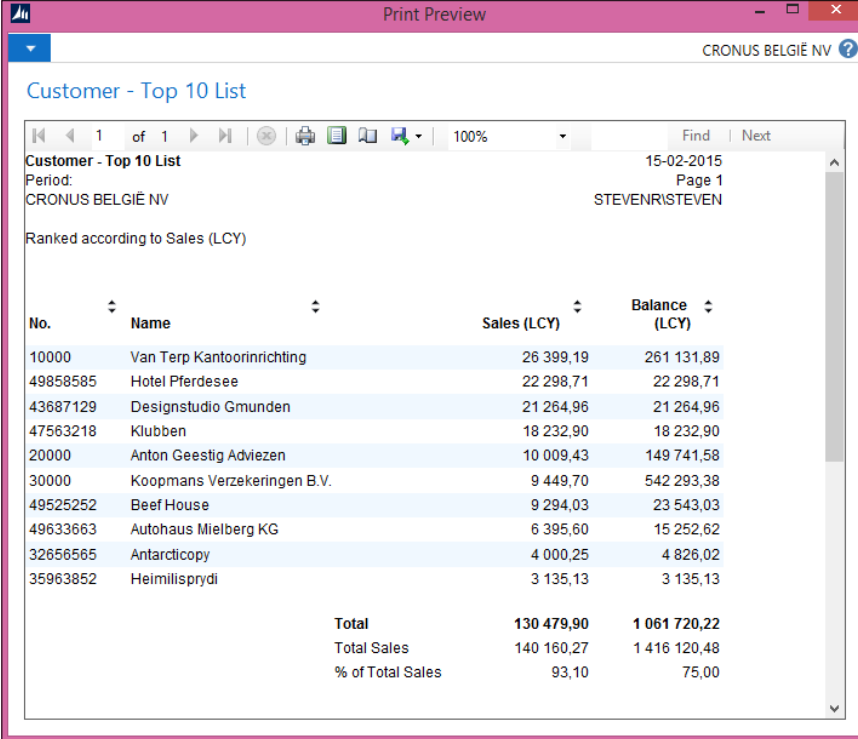
The request page allows you to decide how you would like to view the report. At the top, there's a choice of **Options**. This contains options to change the type of visualization, or the way that the report behaves. Not all reports have an **Options** tab. The request page contains a tab where you can apply filters. In this example, you can apply a filter on specific fields, or on any other field from the underlying table of the report. If the table contains FlowFields, then you also have the option to limit the totals of the FlowFields. In other words, limiting totals means applying FlowFilters to the FlowFields.

To summarize, a request page allows the user to specify options and filters before generating the report layout.

You can then print or export the report to a specific format such as Word, PDF, or Excel with the buttons at the bottom of the request page, or you can schedule the report for it to run later.

The report viewer

When you preview a report from a client computer, the report is displayed using the report viewer:



Customer - Top 10 List

Period: 15-02-2015
Page 1
CRONUS BELGIË NV STEVENRISTEVEN

Ranked according to Sales (LCY)

No.	Name	Sales (LCY)	Balance (LCY)
10000	Van Terp Kantoorinrichting	26 399,19	261 131,89
49858585	Hotel Pferdesee	22 298,71	22 298,71
43687129	Designstudio Gmunden	21 264,96	21 264,96
47563218	Klubben	18 232,90	18 232,90
20000	Anton Geestig Adviezen	10 009,43	149 741,58
30000	Koopmans Verzekeringen B.V.	9 449,70	542 293,38
49525252	Beef House	9 294,03	23 543,03
49633663	Autohaus Mielberg KG	6 395,60	15 252,62
32656565	Antarctcopy	4 000,25	4 826,02
35963852	Heimilispydi	3 135,13	3 135,13
Total		130 479,90	1 061 720,22
Total Sales		140 160,27	1 416 120,48
% of Total Sales		93,10	75,00

 The report viewer is launched from the report request page, but this is explained in a later section. 

The report viewer allows reports to be embedded in Microsoft Dynamics NAV client applications. The report viewer control is installed automatically on:


- Any client computer, for viewing reports from the Microsoft Dynamics NAV client
- The Microsoft Dynamics NAV Server, for using the `SAVEASEXCEL`, `SAVEASWORD`, and `SAVEASPDF` functions
- The computer running the development environment, for compiling reports

The report viewer is used when you preview the report if you are running a report from the Windows client. The report viewer supports user interaction and renders a report as an HTML page behind the scenes.


At the top of the report viewer there's a toolbar that provides navigation, search, export, and print functionality:




You can save a report as an Excel, PDF or Word file. The same report can have a different appearance and functionality, depending on the rendering format that you select. For example, reports that have links, document maps, and bookmarks might not work properly if the report is saved to a file. A report layout in a different file format might include additional pages or white space, depending on how items are aligned.

 It's best to test the development with all render extensions (Preview, Print, PDF, Excel, and Word)

At runtime, users can use the print commands on the ReportViewer toolbar to open a **Print** dialog box, preview the report in print layout, and configure the page setup, prior to printing.

 You can also print a report from within the report viewer but note that, if the function `CurrReport . PREVIEW` is used in the code, the **Print** button will not be available in preview mode.

Print support varies depending on whether you are using the Web server control or Windows Forms control.

 For more information about print support, refer to the following link:
<https://msdn.microsoft.com/en-us/library/ms251693.aspx>

Report development phases

Before you start to develop a report, you should think about what exactly you want to achieve. Ask yourself the following questions:


- Who is the report intended for?
- What is the purpose of the report?
- How is the report going to be used?
- Where should the information come from?
- How should the information be visualized?

It doesn't matter what technology you are using, the development of a report always boils down to two things: the data model and the layout.

The data model phase

The data model is actually the most important phase in report development, because if you get it wrong or if you have to make fundamental changes to it, it usually means you will have to redo the layout. So, take your time to think about the data model thoroughly.

This means that you have to know what information is required in the layout and where it is from. Are you going to use one or more tables? Is the same information available in different tables, in that case, which one are you going to use?

 Remember, in Dynamics NAV, information travels from the master tables, via documents and/or journals, towards ledger entries and posted document tables. Some information is copied from one table to another, in order to keep track of history or changes.

Make a list of all the tables that contain the information and then decide which ones you are going to use. In real life, this probably means you will have to consult an expert or a business user and explain that the information they require resides in multiple tables, so they can make an informed decision as to which table you should use in the report.



Ledgers or Posted Documents?

Dynamics NAV allows posted documents to be deleted after they have been printed. So, if confronted with the choice of using ledger entry tables or posted document tables as data sources for a non-document report, I recommend using ledger entries.

Once you have defined which tables you are going to use, you will need to think about the relations between the tables, how you are going to link the tables, and also in what order? The order will have an influence on the performance of the report, but also on the kind of information that might be missing.

For example, if you are asked to create a report to display inventory by location, the questions you should ask are:

- What is inventory, is it the sum of the quantities in the item ledger entry table?
- What quantity field should I use: **Quantity**, **Remaining Quantity**...?
- What is a location? Is it the **Location Code** field in the **Location** table or something else?
- Do I need to include items for which there is no inventory?
- Do I need to include locations for which there is no inventory?
- Do I have to display the item number or also the description?
- Should I include any translations, substitutions, variants, or cross-references?

The answers to these questions will define which tables and fields you are going to include in the data model and in which order you are going to iterate over them.



A good idea is to make a draft drawing of the layout of the report you want to create on a piece of paper. Write down the fields that need to be visible on the report and then find out which table they are from. After that, if there are multiple tables, find out how the tables are related and write that down. Having an **entity relationship (ER)** model helps a lot here, especially in a customized database.

In this way, when you open the designer, you already know what you need to do. Both novice and experienced developers make the mistake of not thinking before they begin. It can then get confusing very quickly.

A good suggestion is to have a requirements session with the user, during which you can create a mock-up of the report. Then you can define each field, column, grouping, sorting and printing option. Based on this session, you can then create the data model for the report. This mock-up is also referred to as a format design document.

The layout phase

Although the data model is important for the reasons I just explained, the layout of the report determines how the user will perceive it. So, if the layout is not easy to interpret, or if you can't see the wood from the trees, however cleverly you construct the data model, the report is not going to be used.

There are many out of the box reports in the Dynamics NAV application and most of them are never used. One of the reasons is that they have an inadequate layout.

How do I visualize the information such that the report clearly reveals its intention and the user quickly finds what he or she is looking for? That's the most important question to ask when creating the layout.

Creating a report is not difficult, but making it easy to understand, so that you can spot trends and learn from your data, takes some consideration. The main goal of a report is to communicate information clearly and effectively, for example, graphically. A report needs to create insights by communicating its key points in an intuitive way.

Using the example of the inventory by location report, you might consider how you are going to visualise the inventory. Are you going to display a number, or a data bar? Is it important to include a key performance indicator, for example compare the current inventory with the reorder point (or some other important value)? Or are we using items with an expiry date? If so, do items close to expiry need a different color? Then, in what order are you going to display the locations and items?

The testing phase

Testing is a phase that is often neglected, for different reasons. The most frequent excuse for the lack of tests is not having enough time. That might actually be true when you are developing the report, but, in the end, when users complain about bugs and missing functionality, you will wish you had tested more thoroughly.

Of course, this needs to be specified in your report's design document. Its test criteria should mention which formats the report needs to be tested in.

Another reason tests are usually dismissed or poorly carried out is a lack of understanding of the business case. How is the user going to use the report? Is all the information on there, and is it correct? As a developer in the NAV world, you have to put yourself in the shoes of the user. Only then will you truly understand if what you developed is ready or not.

Tests should include export into different formats (PDF, EXCEL, WORD) and actually printing the report on a printer. Test it on different clients: Windows, web, and tablet.

Using the example of the inventory by location report, a test verifies if the inventory is correct and corresponds to the inventory on the item card. Are there any locations or items missing from the report?

Report development tools

The development of reports in Dynamics NAV is done with different tools. You use the report dataset designer to create the dataset, which opens from the object designer in the Dynamics NAV Development Environment. You can choose to use either Visual Studio or Report Builder to create the layout.

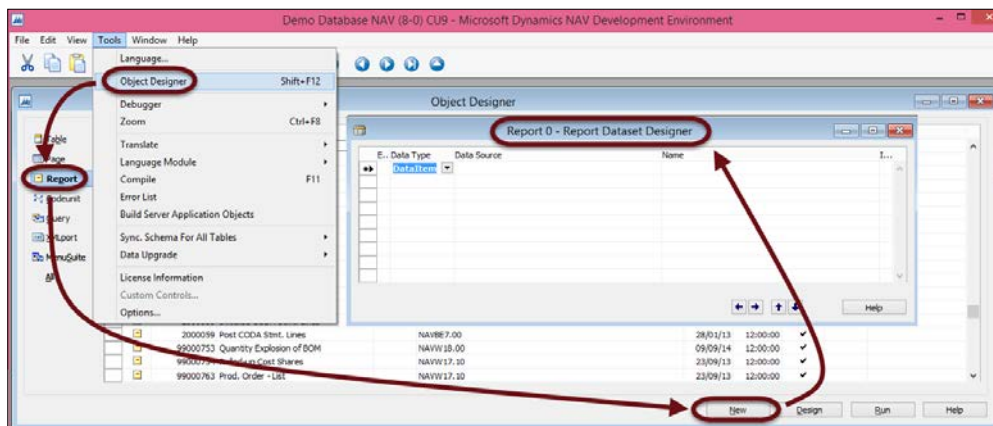


Depending on your version of Dynamics NAV you will have to use different versions of Visual Studio.

What do I use to develop the data model?

You use Report Dataset Designer in the Microsoft Dynamics NAV Development Environment to define the dataset of a report. This is how you open it:

1. In the development environment, on the **Tools** menu, choose **Object Designer**.
2. In **Object Designer**, choose **Report**, and then choose **New**.



How do I create the report layout?

There are two types of report layouts: client report definition (RDLC) layouts and Word layouts. To create an RDLC layout, you use Visual Studio Report Designer or Report Builder from the Microsoft Dynamics NAV Development Environment.

The RDLC layout is the most flexible. By this, I mean that, from a technical point of view, you have the ability to use expressions to determine how and when data should be visualized. The Word layout is restrictive and imposes limitations on the way you create the dataset.

In this chapter, I will focus on the RDLC layout. The Word layout is explained in *Chapter 8, Word Report Layouts*.

Built-in and custom layouts

A report can have multiple layouts. In the development environment, a report can have one RDLC layout and one Word layout. These are the built-in layouts, because they are a part of the report object and are stored inside the report object. You can see this when you export the report object to a text file, as the RDLC and Word layout are then included.

A user can also create a custom layout with the Dynamics NAV application, which is based on the built-in layout. The idea is that a user can customize the built-in layouts according to their needs. In this way, a user can switch between different layouts for the same report. These custom layouts are not stored in the report object, they are stored in a separate table: 9650 report layouts.



In a multi-tenant Microsoft Dynamics NAV deployment, the built-in report layouts are stored in the application database because they are part of the report objects. Therefore, built-in report layouts are available to all tenants. Custom report layouts are stored in the business data database, therefore they are specific to the tenant. This enables you to create separate report layouts for each tenant.

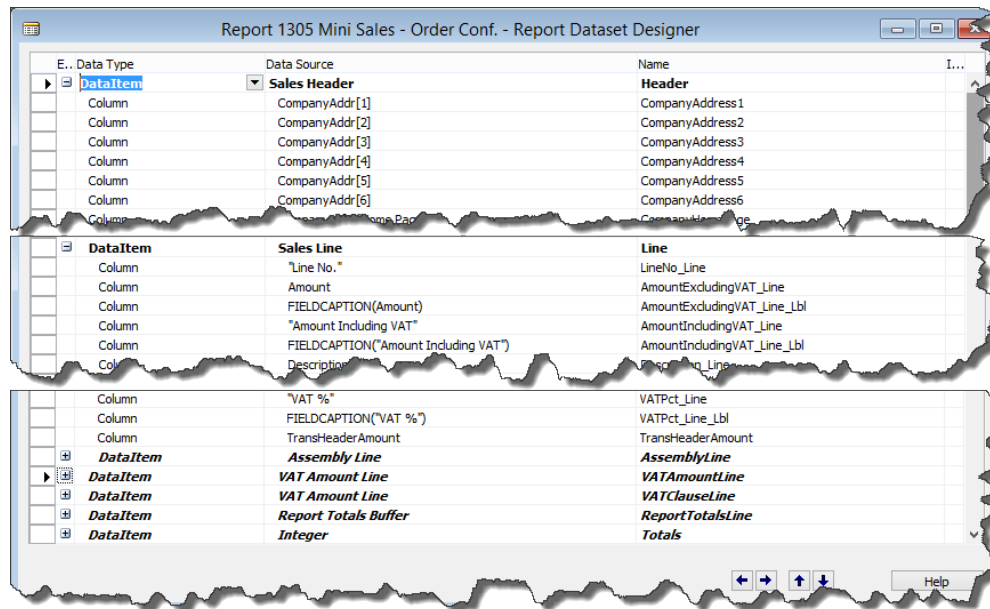
I recommend not creating too many report layouts. It is very convenient to have multiple layouts but it can get confusing and difficult to manage.

Building the data model

The data model of a report is designed in the report dataset designer and will become the dataset for the layout. The runtime dataset is flat and is generated from the data items (tables). The layout will be rendered on top of the dataset.

Flat means that the dataset consists of rows and columns. For example, when you combine two tables that have a one-to-many relationship, such as the sales header and sales line, the dataset will consist of the columns from both tables and a row for every line, in which the columns from the header table are repeated. So, at runtime, the dataset looks different when compared to the definition of the dataset in the report dataset designer, where data items are indented. I will explain in more detail later in this chapter how the dataset is flattened and columns from different data items are combined.

The following screenshot is the dataset of the **Report 1305 Mini Sales - Order**. As you can see, it contains many tables and columns:



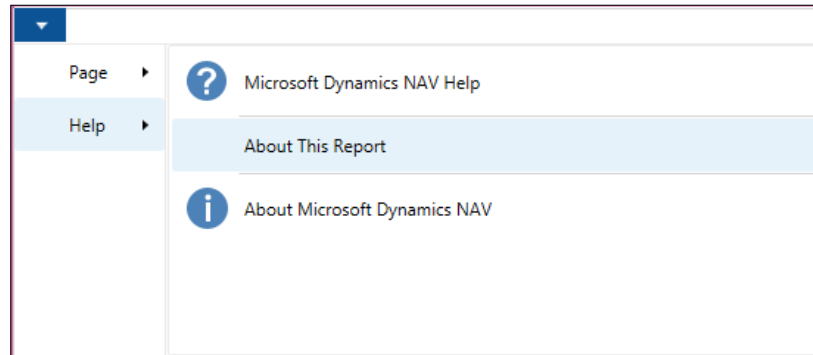
The following is a screenshot of the dataset of this report at runtime:

The screenshot shows the runtime dataset for 'About This Report: Order Confirmation'. It displays a table with columns and rows, representing the data items at runtime.

rdDate...	OrderNoCapti...	HomePageCa...	EmailCaption	BilltoCustNo...	PricesInclVAT...	DimText	DimensionLo...	HeaderDimCa...	SalesLineAmt...	SalesLineAmt...	Desc_SalesLine	NNCSalesLine...	NNCSa
rd Date	Order No.	Home Page	E-Mail	Bill-to Custom...	Prices Includin...	<>	<>	<>	<>	<>	<>	<>	<>
rd Date	Order No.	Home Page	E-Mail	Bill-to Custom...	Prices Includin...	<>	<>	<>	1983,50	##0,00	INNSBRUCK Ka...	1983,50	##0,00
rd Date	Order No.	Home Page	E-Mail	Bill-to Custom...	Prices Includin...	<>	<>	<>	904,60	##0,00	INNSBRUCK Ka...	2888,10	##0,00
rd Date	Order No.	Home Page	E-Mail	Bill-to Custom...	Prices Includin...	<>	<>	<>	<>	<>	<>	<>	<>

As you can see, the dataset at runtime consists of rows and columns. There is a column for every column in the dataset designer, and a row for every record from the data items. This is referred to as a flat dataset, as against the data items, which are indented.

To visualize the dataset of a report at runtime use *Ctrl + Alt + F1*, or **About This Report**, as shown in the following screenshot:



When you select **About This Report**, the system tells you that you have to run the report again to see the data. Actually, you have to enable this feature before you can see the dataset. This means, of course, that you have to run your report twice, just to be able to see the contents of the dataset.



The information in the **About This Report** page can also be exported to Microsoft Word or Excel or to an e-mail in Microsoft Outlook. A user can then forward this information to the helpdesk or whoever is providing support.

If you click on the **About This Report** feature in the request page of the report, then it is also enabled. After that, at runtime, you can click on **About This Report** and the dataset will contain data. In this way, you don't have to run the report twice to be able to see the dataset.

Understanding the report dataset designer

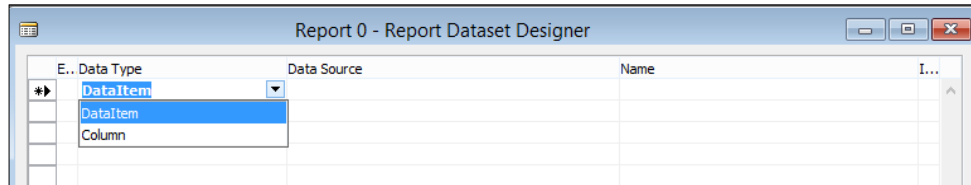
In this section, I will explain how to use the report dataset designer to create a data model for your report.

Building the dataset

Building the dataset is an important process because it determines where and how the data becomes available in the runtime dataset and so will have an impact on how you design the layout.

Data items and columns – fields, variables, and expressions

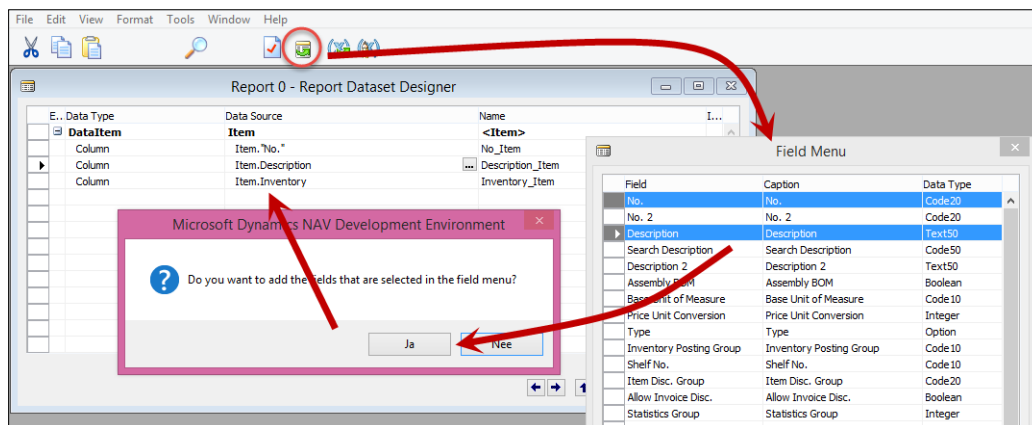
You design the dataset in the **Report Dataset Designer**, based on data items and columns. Data items link to tables in the database and columns link to fields or expressions:




A column can be a field in a table, but it can also be an expression, a variable or a text constant.

Let's start with an example and create an item list report. The idea is to display a list of items, so I will use the **Item** table as my data item and I will add the number, description and inventory as columns.

In the dataset designer, add a **Data Type DataItem** and **Data Source Item** line, then use the field menu button to select the fields from the **Item** table:



The **Field Menu** allows you to select any of the data item fields and add them to the report without having to type in the **Name** of the field.

 In the **Field Menu** window, select one or more fields that you want to add to the report dataset. Select multiple fields by holding down the *Shift* key or the *Ctrl* key. Choose the **OK** button to add the selected fields to the dataset.

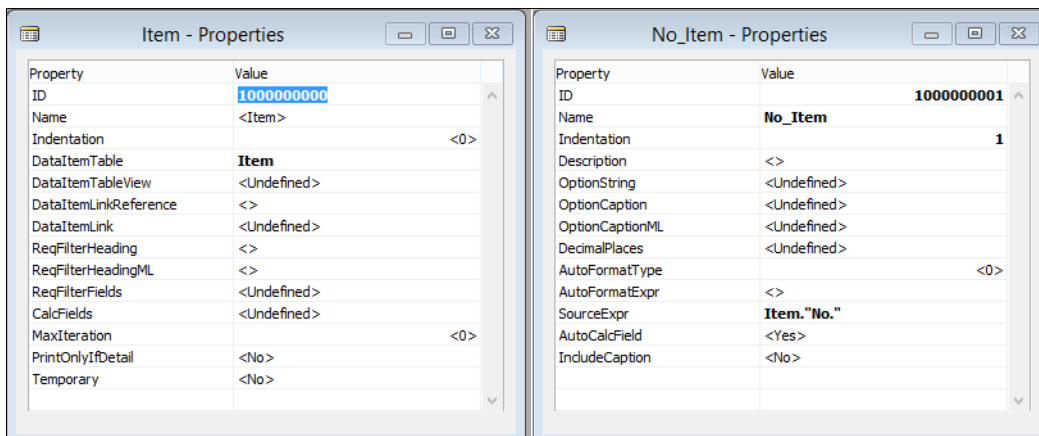
All names in the Name column must be unique and **Common Language Specification (CLS)** compliant. You will notice that, when using the **Field Menu**, the field name consists of the field name, an underscore, and the data item name.

 More information about the Common Language Specification is available in the MSDN Library at <https://msdn.microsoft.com/en-us/library/12a7a7h3.aspx>.

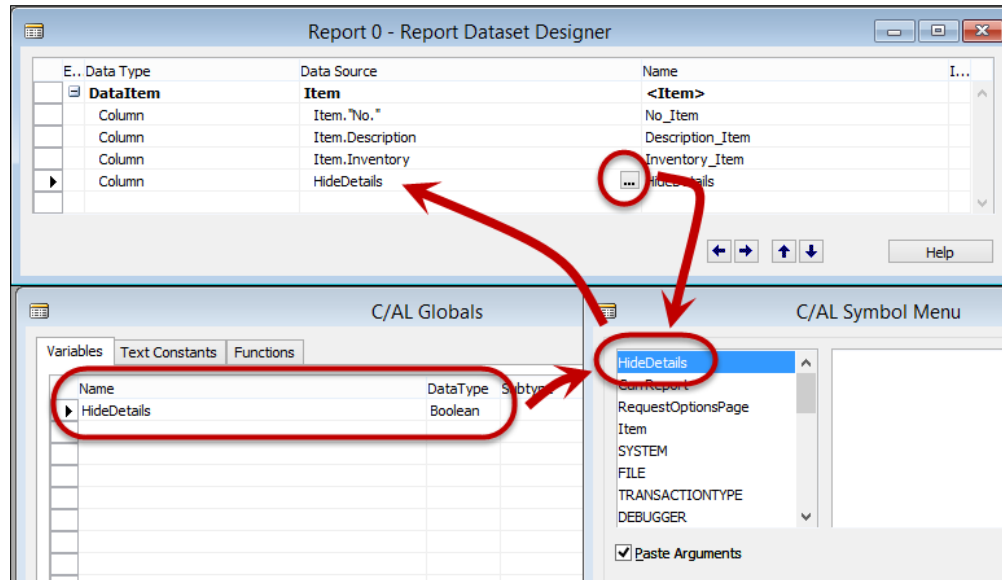
After you have selected the data items and columns you want to add in the dataset, open the properties window. Here you can set the data item and field properties.

Use *Shift + F4* or the property button at the top to open the properties. When you select a data item or a field, the property window displays the properties of the selected data item or field. When you go to the last line in the dataset designer, referred to as the first empty line, the property window displays the report properties.

The following image shows the properties of a data item and of a field:



You can also include a variable in the report dataset. Define the variable in **C/AL Globals** and then add it by typing in its name or using the assist-edit button in the data source column where you can select it from the **C/AL Symbol Menu**:



An example of this report is available in the object: Packt - CH01-1

A column can also be the result of an expression, here are some examples:

- `FORMAT (TODAY, 0, 4)`
- `Item.Description + '-' + Item.No_`
- `STRSUBSTNO (DocumentCaption, CopyText)`
- `- ("Line Amount" - "Inv. Discount Amount" - "Amount Including VAT")`

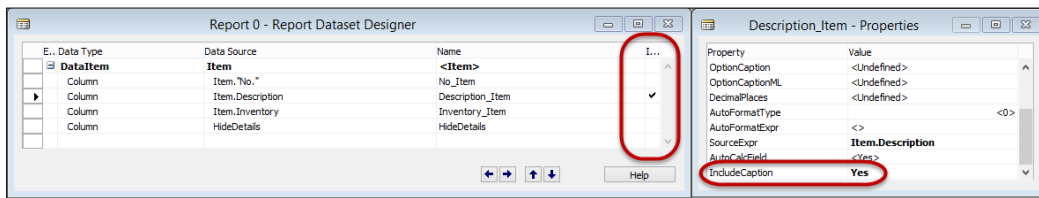
Including captions and labels

You are not only going to display data in the report layout but also field names and textual information. These names and text will be displayed in the user's own language, since Dynamics NAV is a multilanguage application. You can therefore use captions and labels when you design the dataset. Captions and labels are sent as parameters to the report layout. They are not actually included in the dataset, because their value is the same for every record and we don't want this information to repeat because that would increase the size of the dataset unnecessarily.

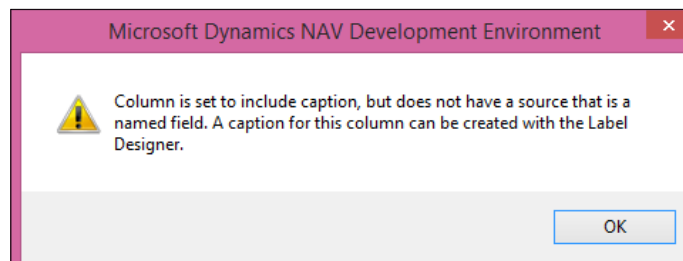
To include a caption:

1. Select the row in the dataset.
2. Open the properties window (*Shift + F4*).
3. Enter **Yes** in the **IncludeCaption** property.

Alternatively, use the **IncludeCaption** checkbox in the report dataset designer:



If you select a row that contains an expression or a variable, then you will get the following error when you activate the **IncludeCaption** property for that row:

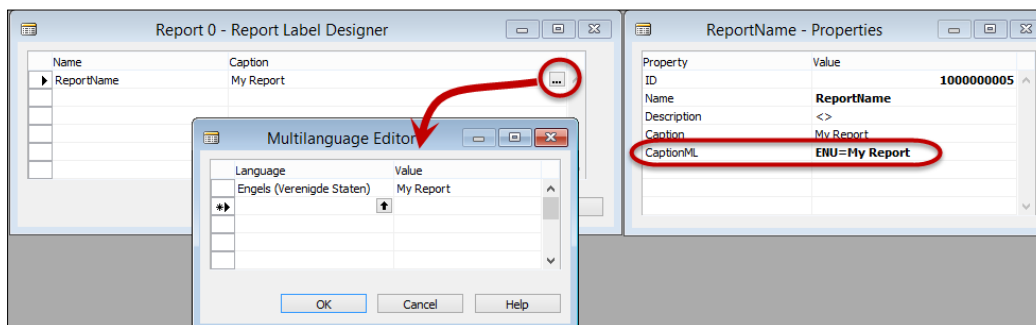


You can only use the **IncludeCaption** property on table fields. For other fields, you can define a label.

To include a label:

1. Go to **View, Labels** to open the **Report Label Designer**.
2. Add a new label in the **Report Label Designer** by entering a **Name** and a **Caption**.
3. Then, in the **Label** properties, use the **CaptionML** property to translate the label into other languages.

Alternatively, you can use the assist-edit button to open the label **Multilanguage Editor**:



How to see the properties



To see the properties of a column or data item, you have to first select it. Make sure it is selected by clicking on it with your mouse. Then, you can click on the properties button at the top of the screen, or press *Shift* and the *F4* function key. Now, the property window opens and displays the appropriate properties.

IncludeCaption versus FIELDCAPTION

Captions are sent as parameters and in the language of the user. In some cases, for example in document reports, you may want the captions to be in the language of the recipient, not in the language of the user. To do that, you use the `FIELDCAPTION` function, and add the caption to the dataset as an extra column. Then, you can determine the language via `C/AL` code.



Keep in mind that adding captions to the dataset with `FIELDCAPTION` increases the size of the dataset, so only do this when it is really necessary. In previous versions of Dynamics NAV, captions were added by default in the dataset, so using `IncludeCaption` and `labels` is a performance improvement.

An example of the use of `FIELDCAPTION` can be found in most document reports:

- Sales: Quote
- Order: Confirmation
- Sales: Invoice
- Sales: Credit Memo
- Sales: Shipment

Examples of the `FIELDCAPTION` function are shown in the following screenshot:

Column	<code>FIELDCAPTION(Description)</code>	<code>Description_SalesShptLineCaption</code>
Column	<code>FIELDCAPTION(Quantity)</code>	<code>Qty_SalesShptLineCaption</code>
Column	<code>FIELDCAPTION("Unit of Measure")</code>	<code>UOM_SalesShptLineCaption</code>
Column	<code>FIELDCAPTION("No.")</code>	<code>No_SalesShptLineCaption</code>

How is the dataset flattened?

At runtime, a report dataset consists of rows and columns. If you only have one data item, then the columns in the dataset designer become the columns of the runtime dataset and the rows from the data item (or table) become the rows in the dataset. In this situation, the runtime dataset looks the same as when you simply run the table.

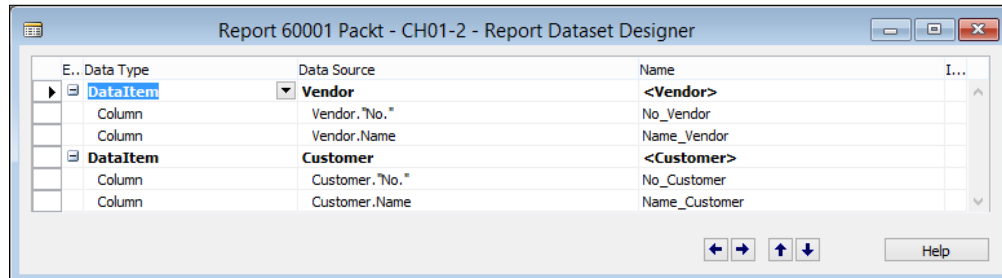
If you combine multiple data items, then the data items at design time can be indented, or not, and this results in a different dataset at runtime. Since the runtime dataset is two dimensional, consisting of rows and columns, information is repeated over multiple rows.

In this section, I will explain and demonstrate how the runtime, flat, dataset is generated, using indented or non-indented data items.

Using multiple data items in a report is a common pattern in most reports and understanding how the multi-data-item dataset at design time is converted into the flat two-dimensional dataset at runtime is very important. Understanding this process is, in my opinion, the most important part of RDLC report development, because it determines how you build the layout and which filters you need to apply in the RDLC layout.

Unrelated tables or multiple data items, without indentation

As an example, let's start with a dataset that consists of two data items, **Vendor** and **Customer**:



When we run this report and display the dataset with the **About This Report** feature, it shows this:

About This Report: Packt - CH01-2

No_Vendor	Name_Vendor	No_Customer	Name_Customer
61000	Megapoel	<>	<>
62000	De Loper	<>	<>
CON1020	Cronus Cardoxy Sales	<>	<>
CON1030	Cronus Cardoxy Procurement	<>	<>
<>	<>	01121212	Spotsmeyer's Furnishings
<>	<>	01445544	Progressive Home Furnish...
<>	<>	01454545	New Concepts Furniture
<>	<>	01905893	Candoxy Canada Inc.
<>	<>	01905899	Elkhorn Airport

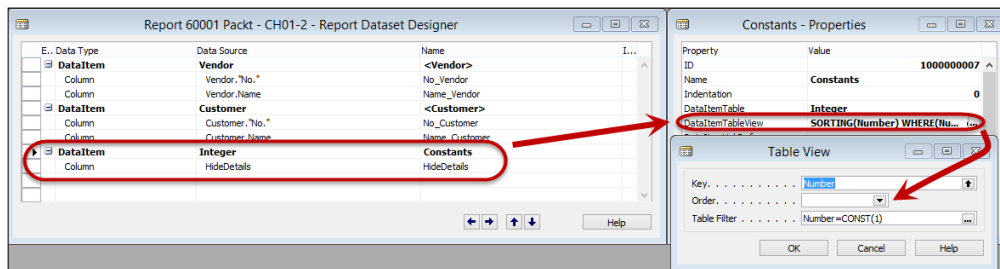
In the dataset there is a column for every column in the report dataset designer: **No_Vendor**, **Name_Vendor**, **No_Customer**, **Name_Customer**.

When the report processes the data items it starts with the first one (on top), which is **Vendor**. For all vendors, it fetches their **No** and **Name** and adds it to the dataset. The customer columns remain empty. Then, after the **Vendor** data item has been completely processed, the system starts with the **Customer** data item and does the same. The result sets of the two data items then follow each other in the dataset, stuck together.

How Do I Start to Create a Report?

Now imagine that I need to create a request page for this report to include an option to display details. Then, I would create a variable `HideDetails`, add it to the request page, and also add it to the dataset. If I added this variable to the dataset as an extra column, then we would have a problem. Are we going to add it to the Customer or Vendor data item? If we add it to the customer data item, it will be available at runtime, but not in the vendor rows. Now, as this is a variable that contains a constant value, we only need it once and there's no point in repeating its value on every row, because that would increase the size of the dataset and so decrease performance.

The solution is to include an extra data item that will only add one row to the dataset. To do this, you can use the integer table, as shown in the following screenshot:



Then the dataset becomes this:

About This Report: Packt - CH01-2

No_Vendor	Name_Vendor	No_Customer	Name_Custo...	HideDetails
<>	<>	49858585	Hotel Pferdesee	<>
<>	<>	50000	Waterschap Oost	<>
<>	<>	60000	Hifi-speciaalzaak	<>
<>	<>	61000	SoundVision	<>
<>	<>	62000	De Onderdelen...	<>
<>	<>	CON1020	Cronus Cardox...	<>
<>	<>	CON1030	Cronus Cardox...	<>
<>	<>	<>	<>	No

The integer data item adds one row at the end of the dataset and this contains the value of the `HideDetails` variable.



The expression `SORTING (Number) WHERE (Number=CONST (1))` in the `DataItemTableView` property of the Integer data item makes sure that the outer data item produces exactly one row from the Integer table.

Use the following expression to retrieve this value in the layout of the report:

```
=Last (Fields!HideDetails.Value, "DataSet_Result")
```

Unless you add the integer data item as the first data item in the report dataset designer, the row is added to the beginning and the expression becomes this:

```
=First (Fields!HideDetails.Value, "DataSet_Result")
```

Remember that we have an extra row in the dataset containing our variable `HideDetails`. This row should be filtered out in the tables in the layout that display the Vendors and Customers.



An example of this report is available in the object: Packt - CH01-2 Using an integer data item, and filtering it to add one or more rows in the dataset, is a common pattern in report design. Instead of filtering on a constant value, you can also set the filter on the integer data item via the C/AL code in the integer data item `OnPreDataItem` trigger. In that way, you can set it at runtime, depending on an option in the request page. In document reports, this is usually how the `NoOfCopies` option is implemented. I will come back to this pattern in the *Chapter 5, Document Reports*.

Related tables or multiple data items with indentation

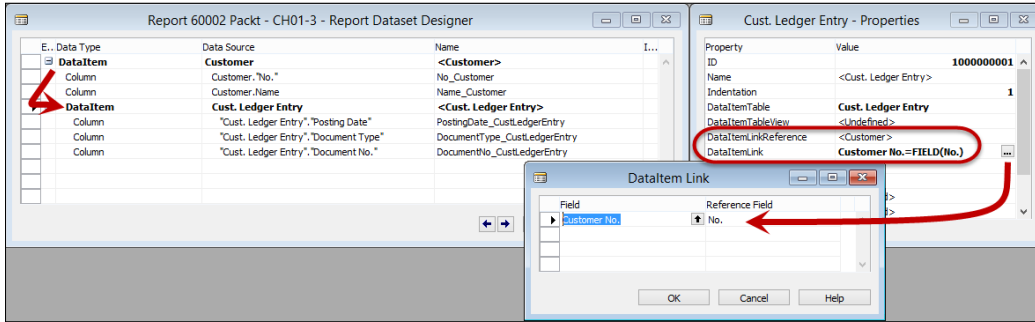
Let's create a report with a dataset that contains two data items as an example: **Customer** and **Customer Ledger Entry**.



In document reports, this pattern is applied a lot, because, in a document report, we have many data items that need to be visualized in different tables (or sections) of the report.

How Do I Start to Create a Report?

The **Customer Ledger Entry** needs to be linked to the **Customer**, so for each customer we can see their individual entries. Linking data items is done by indenting them in the report dataset designer and then, in the indented data item setting, the link fields in the property `DataItemLink`, as follows:



Then, when you run the report, the dataset becomes:

About This Report: Packt - CH01-3

No_Customer	Name_Customer	PostingDate...	DocumentTyp...	DocumentNo...
01121212	Spotsmeyer's Furnishings	<>	<>	<>
01445544	Progressive Home Furnishings	25/01/2016	Invoice	103023
01454545	New Concepts Furniture	31/12/2015	Invoice	00-17
01905893	Candoxy Canada Inc.	<>	<>	<>
01905899	Elkhorn Airport	<>	<>	<>
01905902	London Candoxy Storage Campus	<>	<>	<>
10000	Van Terp Kantoorinrichting	31/12/2015	Invoice	00-1
10000	Van Terp Kantoorinrichting	31/12/2015	Invoice	00-11
10000	Van Tern Kantoorinrichting	31/12/2015	Invoice	00-16

Customers that don't have ledger entries are shown, but the ledger entry columns are empty. Customers that have ledger entries are shown and, for every ledger entry, there's a row in the resulting dataset.

As you can see, if a customer has multiple ledger entries, then, for every ledger, a row is added to the dataset and the columns for the customers are repeated on each of these rows. This is called the flattening of the dataset, the columns of the parent record are repeated for every child record.

To filter out customers that don't have ledger entries, you can use the `PrintOnlyIfDetail` property. You need to set this on the top data item, in this example, the `Customer`:

The screenshot shows the Report Dataset Designer interface. The top window displays the data item hierarchy: `Customer` (DataItem) with columns `No_Customer` and `Name_Customer`, and `Cust. Ledger Entry` (DataItem) with columns `"Cust. Ledger Entry", "Posting Date"`, `"Cust. Ledger Entry", "Document Type"`, and `"Cust. Ledger Entry", "Document No."`. The `PrintOnlyIfDetail` property for the `Customer` data item is highlighted in red and set to `Yes`. Below, the 'About This Report' window shows a table of data with columns `No_Customer`, `Name_Customer`, `PostingDate...`, `DocumentTyp...`, and `DocumentNo...`. A red arrow points from the `PrintOnlyIfDetail` property to the table, indicating the effect of the property.

No_Customer	Name_Customer	PostingDate...	DocumentTyp...	DocumentNo...
01445544	Progressive Home Furnishings	25/01/2016	Invoice	103023
01454545	New Concepts Furniture	31/12/2015	Invoice	00-17
10000	Van Terp Kantoorinrichting	31/12/2015	Invoice	00-1
10000	Van Terp Kantoorinrichting	31/12/2015	Invoice	00-11
10000	Van Terp Kantoorinrichting	31/12/2015	Invoice	00-16
10000	Van Terp Kantoorinrichting	31/12/2015	Invoice	00-3
10000	Van Terp Kantoorinrichting	31/12/2015	Invoice	00-6
10000	Van Terp Kantoorinrichting	31/12/2015	Invoice	00-9
10000	Van Terp Kantoorinrichting	10/01/2016	Invoice	103015

The property `PrintOnlyIfDetails` specifies whether to print data in a report for the parent data item when the child data item does not generate any output. If there are more than two data items, then the report iterates through each parent child relationship in the same way.



An example of this report is available in the object: `Packt - CH01-3`

If you are going to create a layout for this dataset and you want to see the ledger entries per customer, you do this by creating a group in the table and grouping on `Customer No.`

This is a very common design for the datasets in Dynamics NAV reports. It is used with header and line tables, master and ledger tables, and also in document reports.



If you omit the **DataItemLink** and **DataItemLinkReference** between the **Customer** and **Customer Ledger Entry** table, which effectively disconnects the two tables, then the resulting dataset is much larger, since it includes all possible combinations of headers and lines, with absolutely no regard to their possible table relations.

When using multiple data items with indentation, you can also apply the technique of including an integer data item. In that case, investigate the resulting dataset before you create the report layout because, depending on the result, you might want to move the integer data item to the top or bottom to get a better dataset.

Report triggers

A report, like other objects in Dynamics NAV, contains triggers. These triggers are fired when specific events happen and allow you, as a developer, to have code executed at those moments. In this section, I will explain the different triggers and in what order they are fired.

What happens when a report runs?

When you run any report, the **OnInitReport trigger** is called first. This trigger performs any processing that is necessary before the report is run, and so before any data is read, and before the request page is shown to the user.

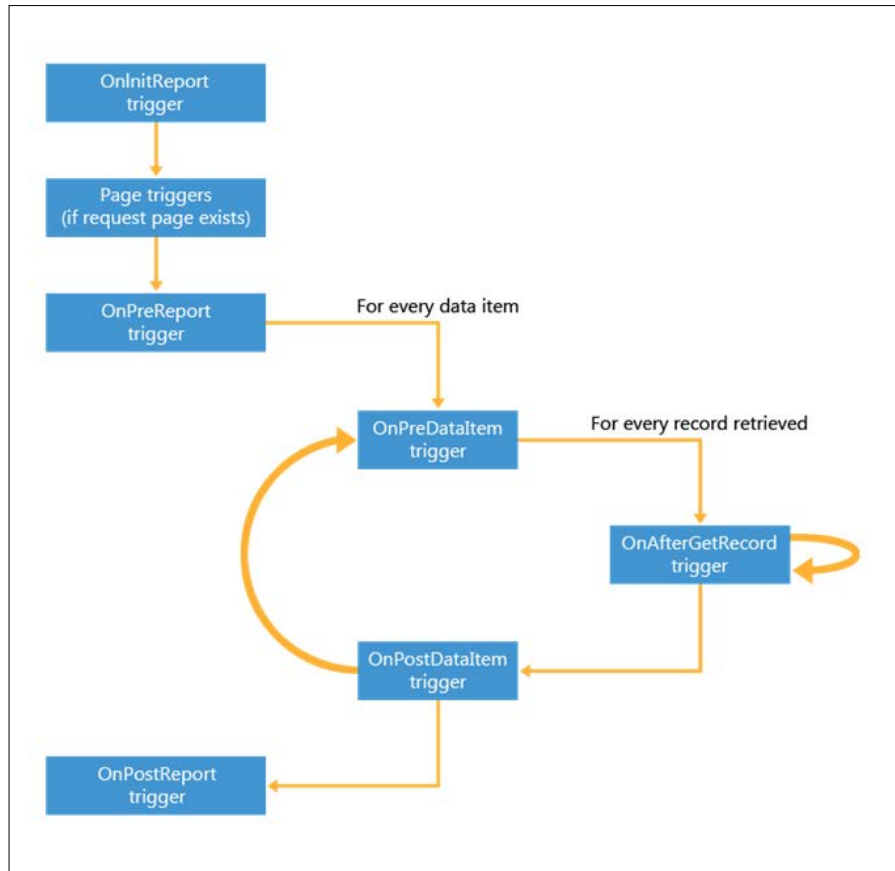
Next, the request page for the report is run, if it is defined. Here, you select the options that you want for this report.

If you decide to continue, the **OnPreReport trigger** is called. At this point, no data has yet been processed. You can use this trigger to initialize variables or fetch information from the database via C/AL code. The *Company Information* table is usually queried in this trigger to retrieve company information like the name, VAT number, company logo, and so on.

When the **OnPreReport trigger** has been executed, the first data item is processed. When the first data item has been processed, the next data item, if there is any, is processed in the same way.

When there are no more data items, the **OnPostReport trigger** is called to do any necessary post-processing.

The following is a visual representation of the report execution flow:



The report trigger sequence

The **OnPreDataItem trigger** is executed before any data is retrieved from the database. This trigger is used to filter the data item dynamically.

The **OnPostDataItem trigger** is executed after the data item has been processed, meaning after all records have been fetched from the table. This trigger usually contains no code, or just code to clean up variables or filters.

In between the **OnPre** and **OnPost DataItem triggers**, the data is processed on a record by record basis. The **OnAfterGetRecord trigger** is executed after a record is fetched from the table, but before it is added to the dataset.


Understanding the flow of report triggers and data item triggers is crucial when deciding where to put C/AL code. C/AL code should not be executed when it is not necessary. For example, if you can choose between the **OnAfterGetRecord trigger** and the **OnPreDataItem trigger**, you should choose the **OnPreDataItem trigger**. This is because the **OnAfterGetRecord trigger** will execute for every single record that is retrieved from the database.

What is a ProcessingOnly report?

A processing-only report is a report that does not print but just processes data or C/AL code as in batch processes that require user input. Processing table data is not limited to processing-only reports. Reports that print can also change records. This section applies to those reports as well.

It is possible to specify a report to be "Processing Only" by changing the `ProcessingOnly` property of the Report object. The report functions as it is supposed to (processing data items), but it does not generate any printed output.

When the `ProcessingOnly` property is set, the request page for the report changes slightly, as the **Print** and **Preview** buttons are replaced with an **OK** button. The **Cancel** and **Help** buttons remain unchanged. When the `ProcessingOnly` property is set, you cannot create a layout.

 If you want to remove the layout from a report set the `ProcessingOnly` property to yes. You will get a prompt that the layout will be removed. Then, set the `ProcessingOnly` property back to No.

There are advantages to using a report to process data rather than a code unit:

The request page functionality that allows the user to select options and filters for data items is readily available in a report, but difficult to program in a code unit.

Using the report designer features ensures consistency. Instead of writing code to open tables and retrieving records, report data items can be used.

Creating the layout

Now it is time to see how to create a layout for a report. In Dynamics NAV, we have a choice of either Visual Studio or Report Builder. Let's start by comparing the two.

Visual Studio versus Report Builder

In the Dynamics NAV development environment you can set the **Use Report Builder** option, via **Tools/Options**. If set to **No**, then the system will use Visual Studio to create the report layout, otherwise Report Builder will be used.



By default, the option is set to **No**, even when Visual Studio is not installed. You have to manually set this to **yes** if you want to use Report Builder.

Report Builder is (normally) installed together with Dynamics NAV when you install the development environment. Visual Studio has to be installed separately. Report Builder is free and can be downloaded from the Microsoft website. To be able to create report layouts with Visual Studio in Dynamics NAV 2013, you need at least the Professional edition of Visual Studio, which is not free.



More information about which version of Visual Studio is right for your environment is available here:

<http://blogs.msdn.com/b/nav/archive/2013/12/19/microsoft-visual-studio-2013-now-supported-for-rdlc-report-design.aspx>

[https://msdn.microsoft.com/en-us/library/dd301054\(v=nav.80\).aspx#DevEnv](https://msdn.microsoft.com/en-us/library/dd301054(v=nav.80).aspx#DevEnv)

As from Dynamics NAV 2015 you can use the Visual Studio Community Edition 2013, which is free and can be downloaded from <https://www.visualstudio.com/en-us/news/vs2013-community-vs.aspx>.

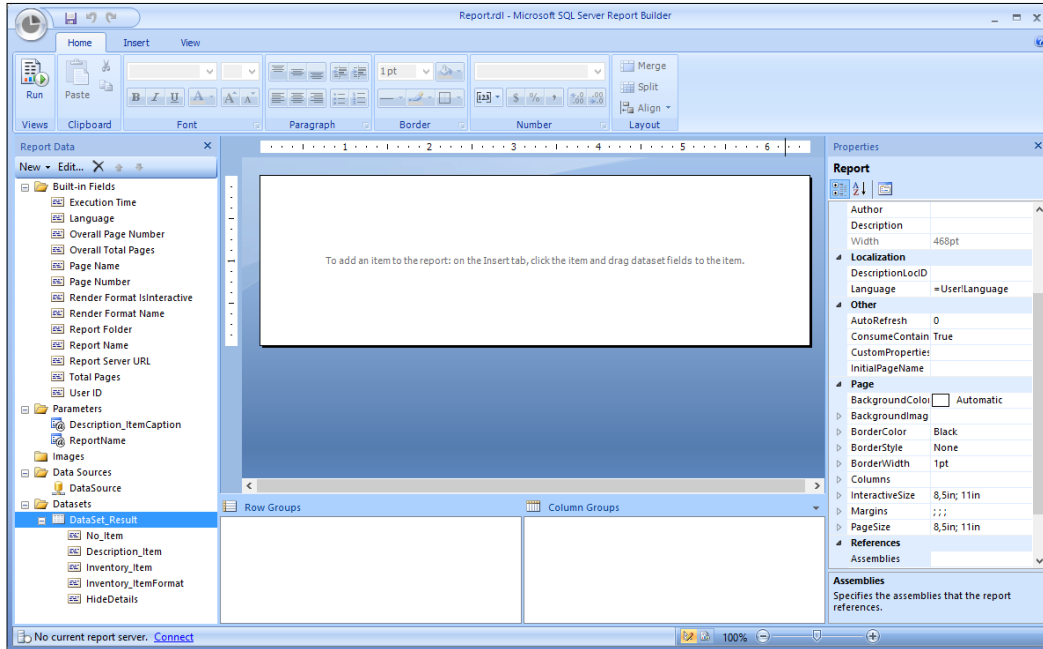
Creating a simple layout in Report Builder

In this section, we are going to use Report Builder to create a report layout. Make sure you set the option **Use Report Builder** to **Yes**.

I will start with the report Packt - CH01-1, which contains a data item for the integer table, and I will create a simple layout to display an item list.

How Do I Start to Create a Report?

From the report dataset designer, click on **View, Layout** to open Report Builder:



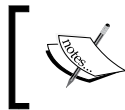
Report Builder features

On the left-hand side you can see the **Report Data** section in Report Builder. It contains:

- **Built-in Fields:** The built in fields can be considered as constants or variables. They are not a part of the dataset and can be used in the layout, in textboxes or in expressions.
- **Parameters:** Parameters contain the labels and captions that were added via the `IncludeCaption` property on a column, or via the label designer.
- **Images:** Images can be imported into the report layout as embedded images.
- **Data Sources:** In Dynamics NAV RDLC reports, there is always exactly one data source. At runtime it links to the report dataset, generated by the report dataset designer.
- **Datasets:** In Dynamics NAV RDLC reports, there is always exactly one dataset. It has the name `Dataset_Result` and contains the columns that were added in the report dataset designer.

At the top you can see the ribbon. It contains three sections:

- **Home:** Here you can see formatting toolbars and buttons. You can use them to format the layout of textboxes in your report.
- **Insert:** Here you can see the toolbox. It contains report items, which are divided into **Data Regions**, **Data Visualizations**, **Report Items**, **Sub Reports** and **Header/Footer**.



Sub Reports are not available when designing Dynamics NAV RDLC reports. They are meant to be used with Reporting Services and require a SQL Server Report Server.

- **View:** Here you can customize the Report Builder look and view. You can disable or enable **Report Data**, **Grouping**, **Properties** and **Ruler**.

At the bottom, you can see the **Grouping** pane. It contains **Row and Column** groups. You can use them when you want to add grouping to your report layout.

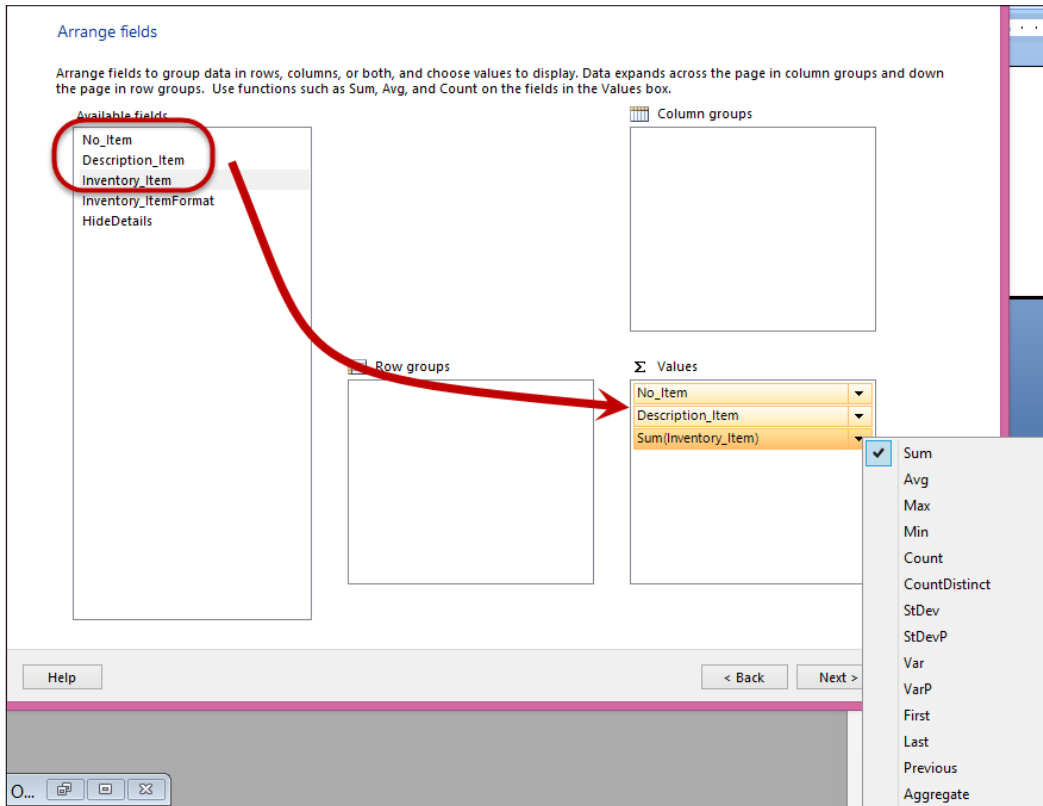
On the right you can see the **Properties**. Depending on what you select in the layout, you can see and edit its properties.

Wizards for prototyping

Report Builder contains wizards to help you to create a report layout. Let's use them to create our item list.

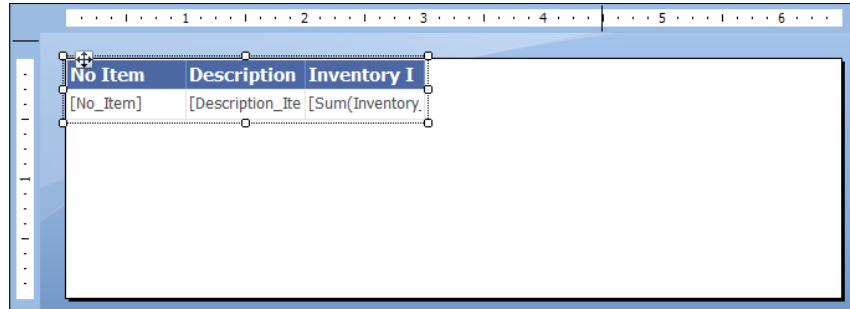
1. In the **Insert** tab in the ribbon, click on **Insert**, then **Table**, and then **Table Wizard**.
2. In the **Choose a Dataset** window select **Dataset_Result** and click on **Next**.

3. In the **Arrange fields** window you can now see the columns on the left. You can drag them to the right and put them into **Values**, **Column Groups** and/or **Row groups**. Let's drag the fields into **Values**:



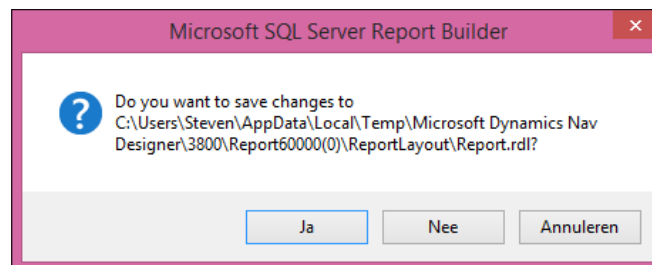
4. For the inventory, click on the little arrow and select **Sum**. Click on **Next**.
5. In the **Choose the Layout** window, you can select grouping and totaling options, but only if you added a row and/or column group in the previous window. Click on **Next**.
6. In the **Choose a Style** window, you can select a style template. Click on **Finish**.

7. The wizard has ended and it has added a table to the body of the report:

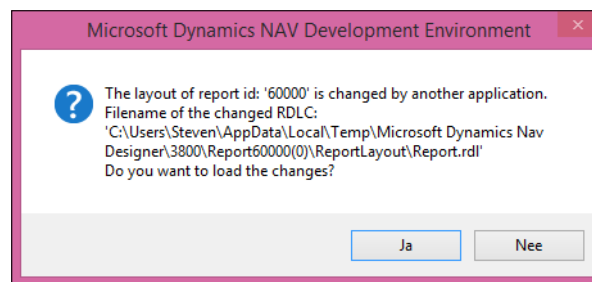


Now, we are going to save and test the layout.

In Report Builder, click on the **Save** button (or *Ctrl + S*). Then close report builder. If you forget to save it, Report Builder displays the following message:



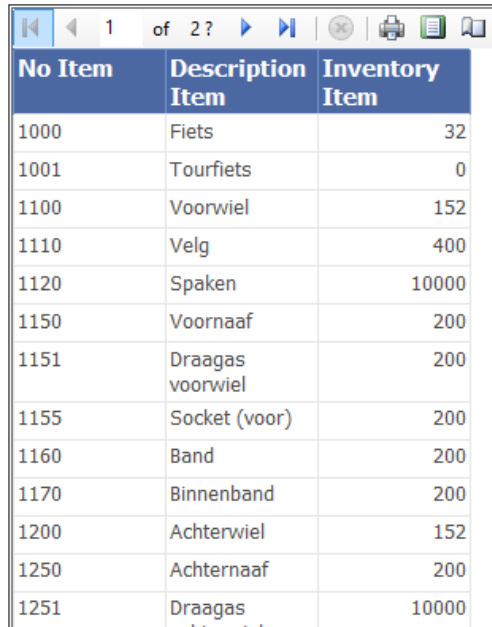
As you can deduce from this message, Dynamics NAV has generated a `report.rdlc` file and opened it in Report Builder. Then, we made changes to it, via the wizard. Now, it's asking us to save our modifications. This `report.rdlc` file will then be imported back into the report dataset designer. This happens when you click on a line in the report dataset designer:



How Do I Start to Create a Report?

Click on **Yes** (if you click on **No** the changes are not imported and will be lost.)
After you click on **Yes**, the system will parse the `report.rdlc` file for errors.
Some errors will be detected at this stage. Now you need to save the layout in the report object in the Dynamics NAV database. Click on **File, Save** (or *Ctrl + S*). When you save it, with the compile option checked, the system will parse the complete report and some errors might be detected.

Now run the report via **File, Run** (or *Ctrl + R*). The request page opens.
Click on **Preview**:



The screenshot shows a report preview window with a table. The window title bar includes navigation icons and the text "1 of 2?". The table has three columns: "No Item", "Description Item", and "Inventory Item". The data rows are as follows:

No Item	Description Item	Inventory Item
1000	Fiets	32
1001	Tourfiets	0
1100	Voorwiel	152
1110	Velg	400
1120	Spaken	10000
1150	Voornaaf	200
1151	Draagas voorwiel	200
1155	Socket (voor)	200
1160	Band	200
1170	Binnenband	200
1200	Achterwiel	152
1250	Achternaaf	200
1251	Draagas	10000

To further enhance the report layout, you can reopen it via **View, Layout**.



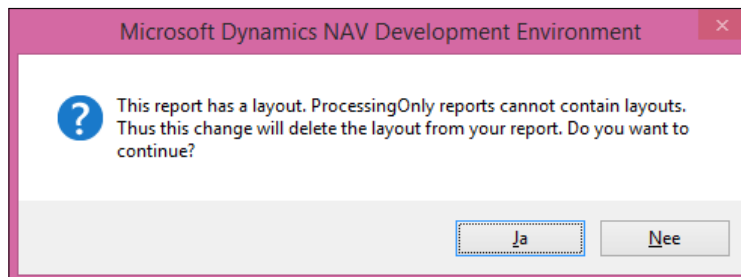
You don't have to close Report Builder if you know in advance the layout is not ready and you only want to test or preview it. In that case, save the layout in Report Builder and minimize Report Builder, don't close it. Then, click on a data item to import the layout, confirm and save, and run the report. If you need to make more changes to the layout, simply return to Report Builder, make your changes and follow the same steps to import your changes back into the report dataset designer.

It's very easy to create a simple layout using the wizards in Report Builder. I use it a lot for prototyping. You can then further enhance the report layout in Visual Studio.

Creating a simple layout in Visual Studio

In this section we are going to use Visual Studio to create a report layout. Make sure you set the option **Use Report Builder** to **No**.

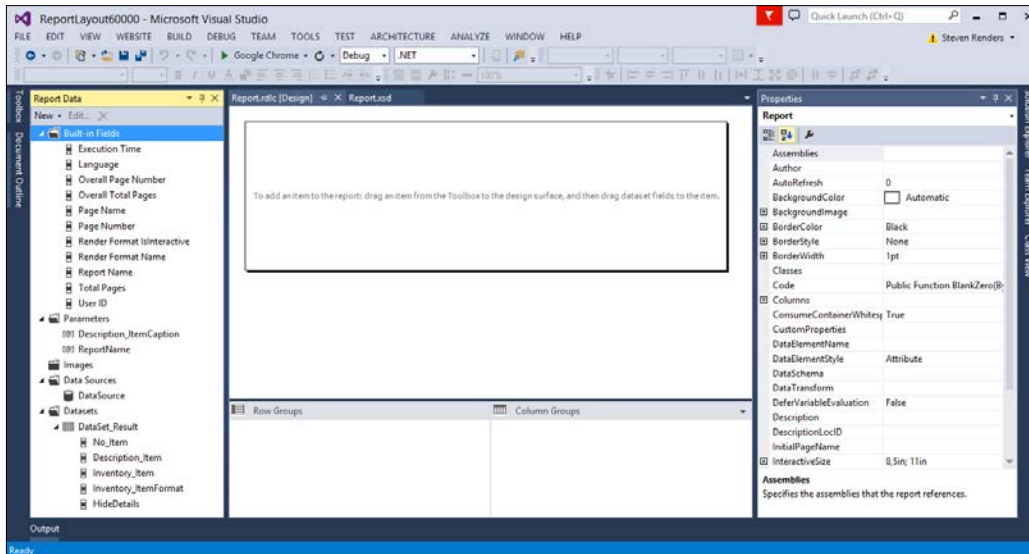
I will start with the **Item List** report I created with Report Builder, but first, I will remove that layout by setting the `processingonly` property to `yes`. The following message is displayed:



How Do I Start to Create a Report?

After confirming and deleting the current layout, don't forget to set the property back to No.

Next, click on **View, Layout** to open Visual Studio:



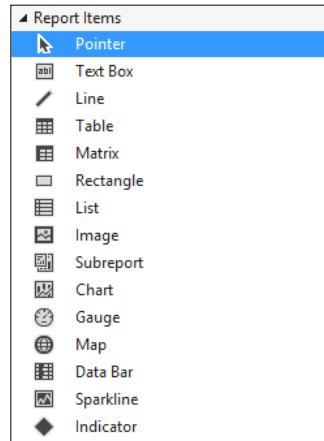
In Visual Studio you can see similar sections to those in Report Builder.



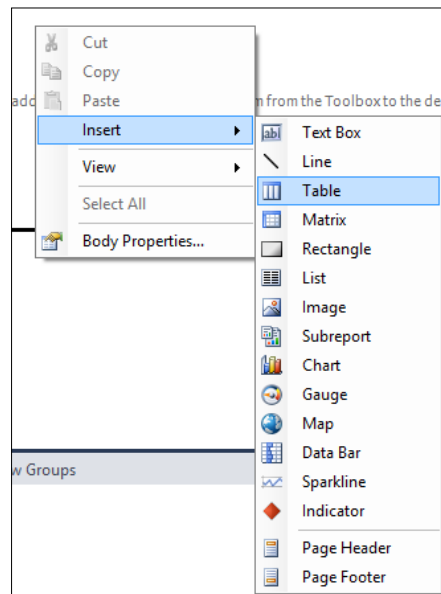
The Report Data window needs to be opened manually by selecting **View, Report Data** (or *Ctrl + Alt + D*). You will need to do this every time you open Visual Studio.

If **Report Data** is not available in the **View** window, then click somewhere in the body of the report to make it available in the **View** menu. Visual Studio automatically populates the toolbars with options depending on what you have selected in the layout.

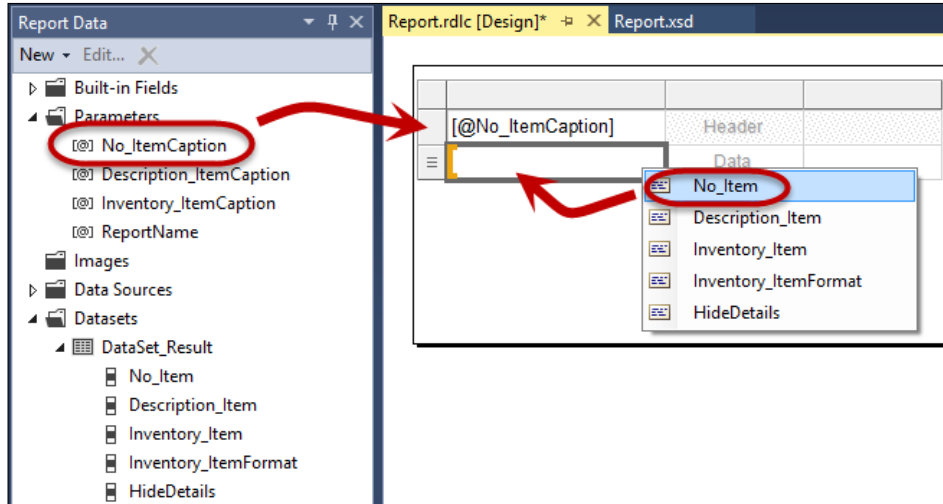
To create a table layout, I will use the **Toolbox**. To open the Toolbox click on **View, Toolbox**:



Here, I will drag the table onto the report body. Alternatively, right-click in the report body and select **Insert, Table**:



To populate the table I will use the captions on the first row, which is the header row, and I will use the value fields from the dataset. From the parameters, drag the parameter into the textbox on the header row. Then, in the corresponding textbox on the detail row, click on the drop-down box to select the value:



There are many other ways of doing this, but I find this the easiest. Otherwise sometimes Visual Studio might create extra columns.

Repeat this process for all the columns. Then select the first row via its handle and make it bold. Next, we need to save our layout in Visual Studio, minimize it, and then go back into the report dataset designer and import it into the report object, just like we did with Report Builder. Then you can save and run the report to see the result.

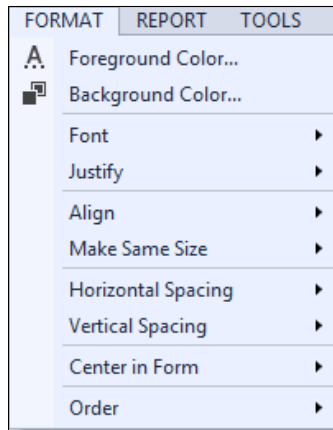
Visual Studio features

Visual Studio contains many features to make report editing a pleasant experience. I will introduce these features in this and the following chapters.

Report formatting, toolbars, and document outline

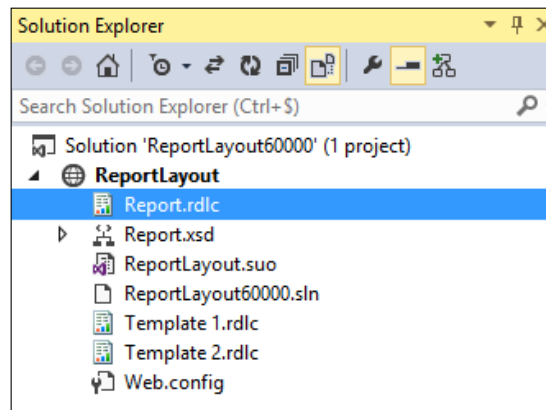
In Visual Studio, you can change the layout of the report designer. You can move around the report data, properties and other sections. You can enable and disable toolbars, and so on. It's much more flexible than the report builder.

There is also the **Format** menu:



This menu contains many formatting options. They are especially useful when you need to align textboxes, for example in a document report.

There's also the **Solution Explorer**:



It contains the complete solution. Be careful, because only the `report.rdlc` file is imported into the report object. Any changes to the other items in the **Solution Explorer** are not saved. One advantage is that we can add extra template reports in the **Solution Explorer**. These templates can contain tables, lists, and matrices which you can pre-format and then copy and paste into your report layout. I will explain later how you can create and include these templates.

Building and testing the layout

I would like to point out that there is a big difference between what you see and experience when you print an enhanced report onscreen and what you see and expect, when you actually print it on paper, or export it to Excel, Word or PDF.

When you run a report in the report viewer then it is fully functional. What I mean by this is that all interactive features the report contains are available to the end user. An enhanced report can, for example, contain hyperlink actions, expand/collapse functionality, drilldowns, and so on. When the user prints the report on paper, that interactivity is lost. If the main usage of a report is to print it, as document reports usually are, then don't spend much time adding interactivity to the report, because it will almost never be used and so will not be a very good return on investment.

Don't enhance too much



From personal experience, I have seen many customers migrate towards enhanced reports as they are eager to implement a lot of interactive features in reports, and because it looks like a good thing to do. It is possible, so why not implement it?

Remember that adding functionality to reports should be based upon business requirements and not only because it looks nice. The more functionality you implement in reports, the more difficult it will be to maintain these reports.

Apart from the functional reasons online reports are better than printed reports, there's also the financial aspect. By using online reports, you will spend less on paper, ink, storage, binding, and distribution.

The ability of RDLC to export a report to PDF and/or Excel can help, but you must also remember that not all of the interactive features will still work in PDF and/or Excel. For example, drill down is available in Excel, but not in the PDF format.

Testing pagination and layout in different rendering extensions

Some reports look very different in online mode as compared to being printed on paper or exported to Excel or PDF. For example, totals at the bottom, page breaks, running totals, headers, and footers are items to pay attention to. That's why it's very important to test all of these scenarios and explain to the end user how to use the report.

Even when the user specifically asks for only Excel and PDF it is important to test other renderings and run any issues that you see by the users, to make sure they don't come back later if they see a problem with the other rendering options.

Testing the report in different clients – Windows, Web, and tablet


Just as you test different rendering extensions for a report, you should also test it in different clients. The actual report content will be the same, but the request page might be different or sometimes not shown at all.

Reporting design guidelines

Microsoft has published design guidelines you can follow when you create a layout for a report. This is especially important when you develop document and list reports. By following these guidelines, you can make sure that your reports are:

- Simple and clean
- Easy to scan and read
- Professional and consistent

Applying these guidelines has the advantage that you can use a minimum set of rules, by using the default formatting options available in Visual Studio. You can also create report layout templates that you can apply by report type so that all your reports have a standard look and feel and give a consistent user experience.

 More information about the report design guidelines is available here:
[https://msdn.microsoft.com/en-us/library/jj651616\(v=nav.70\).aspx](https://msdn.microsoft.com/en-us/library/jj651616(v=nav.70).aspx)

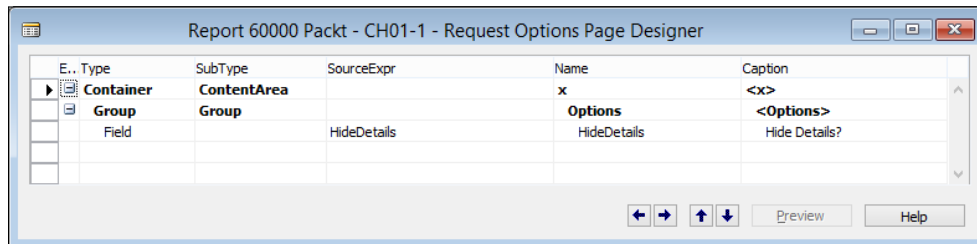
Microsoft followed these guidelines, in some reports, when they developed the report layouts in Dynamics NAV 2013. In most implementations of Dynamics NAV, document reports are customized and this process can be time-consuming. The idea was to provide layouts that could be used out of the box in real life. I will leave it up to you to decide if this is true.

The request page

Before a report is run in the Windows client, a request page is shown. Here, you can enter filters, define sorting, and specify options for the report. A request page is automatically created when you create a report. The **Request Options Page** tab has to be created by the report developer.

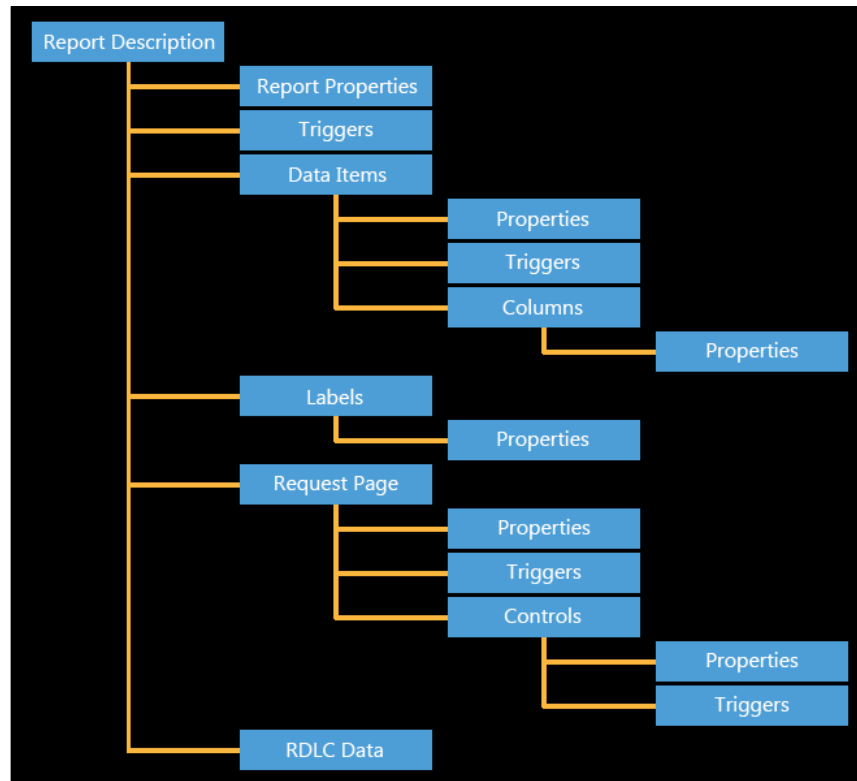
To design the request page options tab, click on **View, Request Page** in the report dataset designer. This opens the request page designer. It is actually a page designer in which we usually create an options `Fasttab`.

Here's an example:



The report description

The following illustration shows the components of a report and how they are related:



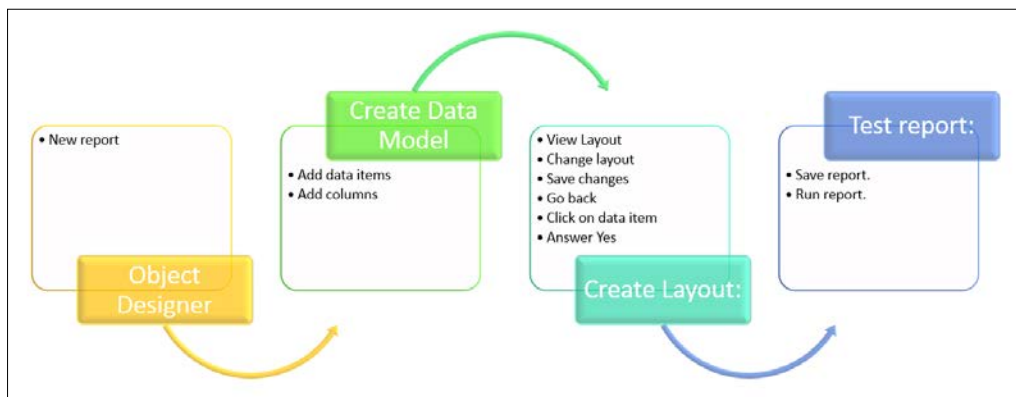
The report creation workflow

Let's summarize what we have learned about how to create a report. The steps required to create an RDLC report are:

1. Create a new report in the object designer.
2. Create the data model in the report dataset designer
 1. Add a data item in the data item designer.
 2. Add fields from the data item as columns in the report dataset designer.

3. Create the report layout:
 1. Use the **View/Layout** menu to open the layout and create the RDLC file.
 2. Make changes to the layout, and add new controls from the toolbox.
 3. Save the changes in the layout.
 4. Go back to the report dataset designer and click on one of the data items.
 5. Answer **Yes** to importing the RDLC file.
4. Test the report:
 1. Save your report.
 2. Run your report.

These steps can be visualised in the following diagram:



Summary

In this chapter, we created our first report, using Report Builder and the Visual Studio report designer. The chapter provided you with a good understanding of the steps required to get started when creating report layouts.

Remember that developing a report can be broken down into different phases: the data model, the layout and testing.

We can conclude that the RDLC layout is a very big step forward in regards to report design functionality. It has enormous potential and added value and features that we can now use and apply when designing reports in Dynamics NAV.

The choice between Report Builder and Visual Studio depends on your experience. If you are a less experienced developer and you want to make use of wizards to create a report layout, then Report Builder is a good starting point. Visual Studio has a lot more features and is much more flexible. So, as you go along, you will probably favor Visual Studio over Report Builder or, to get the best of both worlds, you can combine the two. Of course, Report Builder is free and Visual Studio is not, so sometimes your budget will determine which one you are going to use, depending if you are using Dynamics NAV 2013 or 2015. But Visual Studio has so much added value that it will have a great return on investment.

In the next chapter, we will explore the Tablix data region. I will explain what a Tablix is and how it can be used to create a list, table or matrix.

Last, but not least, we will learn about grouping, sorting and formatting report items.

2

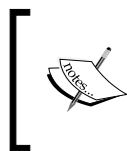
Getting Started with the Tablix

In the previous chapter, we learned how to create our first report. We went through the different report development phases and introduced the report development tools.

This chapter is all about using the Tablix control when creating a layout for the report. It will start by explaining that the Tablix can be used as a List, Table, or Matrix, and then demonstrate the differences between them and when to use which. This chapter also covers different techniques on how to filter, sort, and group information in the report layout. The chapter also introduces some important textbox properties.

Report items

To design the layout of a report in Visual Studio you can use the toolbox, in which you will find all the controls you can use on your report. It's with this toolbox, by clicking on a control and dropping (or by drag and drop or drag and draw), that you can add to the report. You can also right-click on the layout (the body, header, or footer) and insert a control.



There are multiple ways of achieving the same goal. In this book, I will use and demonstrate the steps I apply on a daily basis in real life. Keep in mind that there might be different ways to do the same thing, but I will not document all of them.

The controls in the toolbox can be divided into report items and data regions. Report items represent information, which might come from the dataset. Examples of these types of report items are lines and rectangles. Images and textboxes are independent report items that can be connected to a field from the dataset, but it's not mandatory.

Static report items are items on a report that are not connected to a dataset. A textbox, for example, represents a text constant for labels or comments in a report. The static report items are as follows:

- Pointer
- Textbox
- Line
- Rectangle
- Image

When you drag a field from the dataset in the report body, the system will create a textbox that references the field from the dataset. But, when you run the report, the textbox will not be repeated for every record in the dataset.

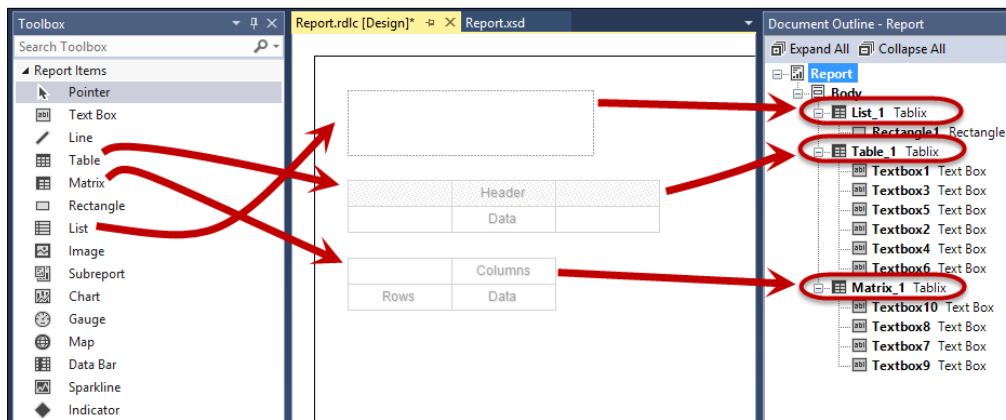
Data regions are areas in a report that contain data from the dataset that is repeated, in the form of multiple records. The data regions are:

- List
- Table
- Matrix
- Chart

So, if you want to display multiple records from the dataset, you need to select one of these data regions from the toolbox, add it to the report body and then fill it with fields from the dataset. Only then, with a data region, will the records be repeated.

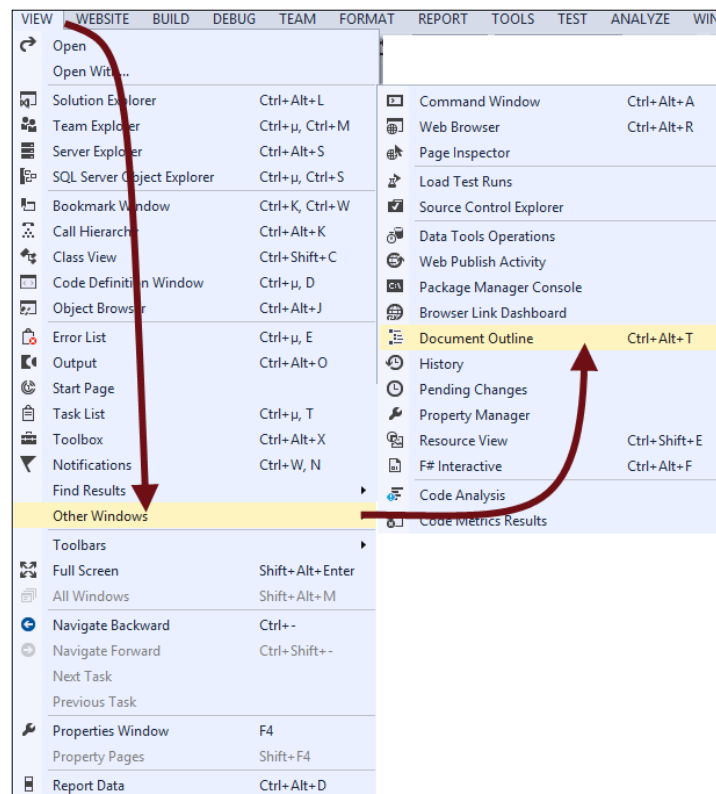
Everything is a Tablix

If you already have some experience in developing RDLC report layouts in Dynamics NAV 2013 or later, then you will surely have noticed that when you select a **List**, **Table**, or **Matrix** from the toolbox and add it to the report body, it shows up as a Tablix in its properties, as you can see in the following screenshot:



The Document Outline

To open the **Document Outline** window, go to **View, Other Windows**, and select **Document Outline**, or use the shortcut: *Ctrl + Alt + T*:

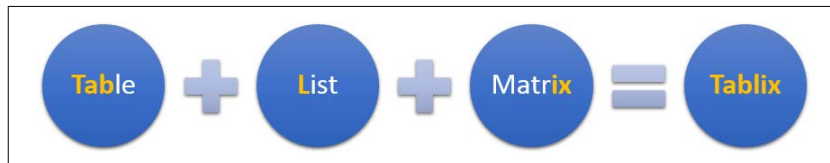




Tablix naming

To see the difference in the properties I recommend changing the name of your **List**, **Table**, or **Matrix**, as I did in this example, to reflect the template you used to create it. An explanation of how you can do this is available in the section, *Changing the name of a Tablix*.

This was not the case in earlier versions and the reason is that we are now on RDLC version 2010. In RDLC version 2008, the Microsoft SQL Server team decided to merge the List, Table, and Matrix data regions into a new object called a Tablix. Of course they needed a fancy name for the new object and came up with Tablix because:



The change you need to make in your mind is that you need to consider the List, Table, and Matrix controls that you find in the toolbox as templates for the Tablix.

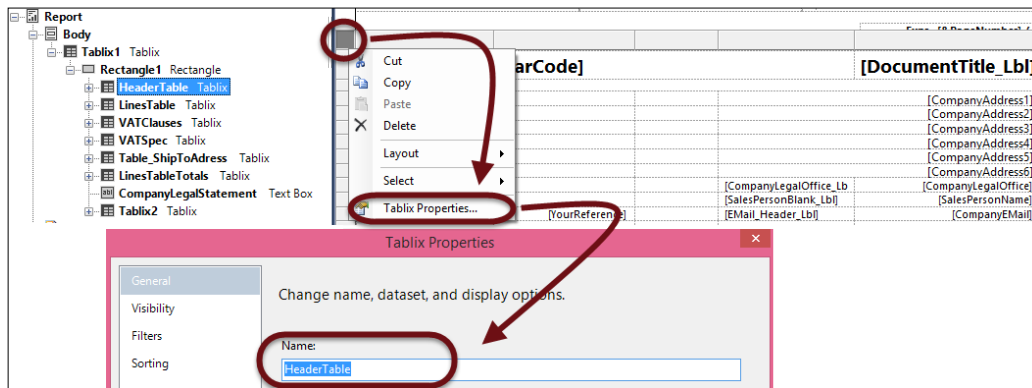
- A List is a Tablix template to display information in a free form layout
- A Table is a Tablix template to display detailed information in a grid layout
- A Matrix is a Tablix template to display grouped information in a grid layout

I find this a great improvement because you can upgrade a table into a matrix. This means you don't have to lose any work from your layout if, during report development, you notice that the table you created actually needs to become a matrix. Instead of deleting the table and creating a matrix, you can simply convert the table into a matrix. On top of that, the Tablix has some other improvements which make it a very flexible report component.

Changing the name of a Tablix

To change the name of a Tablix, you need to click on the Tablix control so that the gray bars become visible at the top and on the left. Then, in the corner on the left top, right-click so that the dropdown menu appears. In the menu, select the **Tablix Properties** option.

The window that opens contains **Tablix Properties** and, in the **Name** property, replace the name with a proper name, as in the following example:

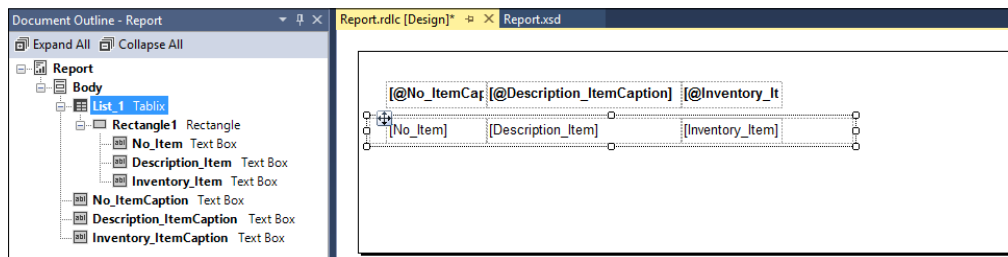


The example comes from **Report 1306 Mini Sales Invoice**. This report was added in Dynamics NAV 2015 and, in this report, Microsoft used proper names for the Tablixes in the report. This is not the case in all reports.

List versus Table versus Matrix

So, let's have a closer look at the Tablix and investigate the difference between a List, a Table, and a Matrix.

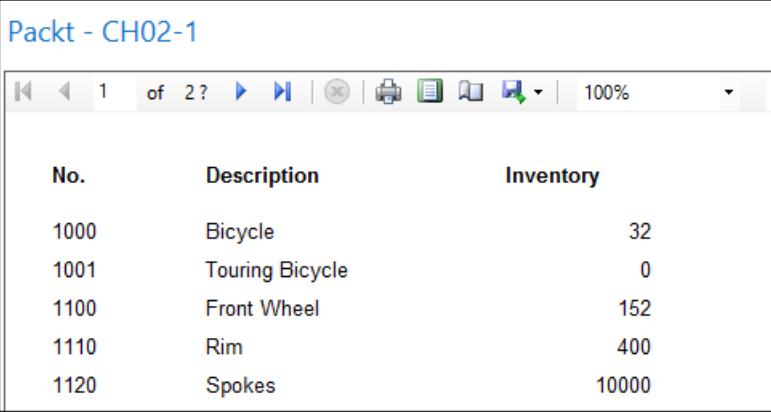
The List is actually completely different from the Table and the Matrix. When you add a List to your report, you will see a List that contains a Rectangle. In the following image you can see this in the **Document Outline**:



This example is available in the object: Packt - CH02-1

In this example, I added a List from the Toolbox to the report body. Then I dragged and dropped three fields from the **Report Data** in the List, which I renamed as `List_1`. Then, I dragged the captions from the **Parameters** on top of the List. Next, I spent some time aligning the textboxes in the list with the textboxes above the list.

The result looks as follows:



No.	Description	Inventory
1000	Bicycle	32
1001	Touring Bicycle	0
1100	Front Wheel	152
1110	Rim	400
1120	Spokes	10000

You will probably find this a cumbersome way to create a simple list report, and it is. Any changes in the layout require you to realign the textboxes, which is time consuming. That is why a List data region is almost never used for these types of reports. We will use the List for other things. In *Chapter 5, Document Reports* we will see why we sometimes need the List.

The List can be used as a free form control if you need to position elements exactly, for example when they need to be printed on preformatted paper. Unless you have a really good reason to use a List, I would not recommend it.



Be careful

When moving textboxes around in a List with your mouse, it can happen that you move a textbox outside of the list, where you actually don't see it and it seems still in the list. That is why the **Document Outline** window is very interesting. In there you can see what is in the List and what is not. An alternative is to use the `Parent` property. It will contain the name of the report item that is the parent of the one you selected.

Usually, we want to organize our database fields in rows and columns. To achieve that goal, the Table and Matrix are much better than the List.

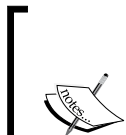
In the previous chapter, when we created our first layout, there was an example on how we use a table to create a list. Also, when you use the wizard in **Report Builder**, it will use a Table in most cases to create the layout.

When you add a table from the toolbox on the layout, it contains two rows and three columns. By using the right mouse button you can add or remove columns and rows. There are three types of rows: **Details**, **Header** and **Footer** rows. The type of row that you right-click on to add a new one is the type of row that will be added. A table can contain multiple header, detail, and footer rows. This is especially useful when you want to show or hide a row dynamically. You will often add multiple rows to a table and then decide via an expression when they should be hidden or visible.



Actually, there are other types of rows, for example, group headers and group footers. These are available when you group detail information in a Tablix.

A table can contain detail data or aggregated data. Aggregating data means that you will create a group in the table, for example, by Item Number, and then calculate an aggregation of a detail value, for example the sum of the inventory by item.



Detail data is the data that you see in the report dataset about this report feature. Grouped data is data that you will organize or group by a field or expression, that you define. The dataset always contains the information at the detail level. You add grouping, filtering, and sorting in the layout of the report to organize the information.

The main difference between a Table and a Matrix is that a table can contain one or more groups and a matrix has to contain at least one row and one column group. This means that a Matrix allows you to create a layout with a variable number of columns. Another difference is that, since in a Matrix data is always grouped, you don't have the detail level, as you do in a Table. If you want to see details when using a Matrix, I recommend including a drill-through. You can do this with the textbox action properties. If you click on one of the textboxes in the Matrix and link it to a table or another report that contains the details, it would be similar to a flow field in Dynamics NAV.



An example of how you can create links in reports is available in Chapter 6, *Tips and Tricks*, in the section, *Creating links*.

There are different types of groups and I will discuss them in the section about grouping in this chapter.

Filtering and sorting

Filtering and sorting are important concepts when running a report and, in many cases, are offered to the users as options in the request page.

How can I implement filters?

For every data item in your dataset there will be a separate tab in the request page allowing the users to apply filters on any of the fields of the tables. You can use the following properties to manage this:

- **ReqFilterFields:** The fields you put in here are the default fields that will be shown for this data item in the request page.
- **DataItemTableView:** If you select a key in this property then the data item disappears from the request page, unless you have selected **ReqFilterFields**. The user will not be able to select a sorting order for this data item.
- **ReqFilterHeading** and **ReqFilterHeadingML:** Both can be used to change the name `FastTab` for the data items in the request page. By default, the table name is used.

Now, sorting and filtering is almost always something a developer is going to apply. The question then is, where do we apply sorting and filtering? There are several options, we can filter in the report dataset designer by making use of data item properties or triggers, or we can apply filters in the report's RDLC layout.

The answer to this question is that you should filter as soon as possible, which means in the report dataset designer. This is because we don't want rows and columns to appear in the reports dataset if they are not needed. The dataset is sent to the client, over the network. The bigger the dataset, the more memory the client will require and, at a certain point in time, it can run out and cause an out of memory exception.

For example, if you want to create a top ten customer list, you can use a table filter in the table data region in the RDLC layout. Let's assume you have about 100,000 customers in the customer table. Then, at runtime, all of them will be fetched from the customer table, sent to the dataset, and then filtered in the RDLC layout. I think it's better to fetch only the rows that you will actually use from the database and send them to the dataset.

When is it better to filter in the RDLC layout? I would answer, when you don't know the filter value and/or filter field. You can use that for dynamic filtering because the filter value and the filter field can be expressions in RDLC. Another reason is when you want to give the user the ability to dynamically filter the dataset.

The same is valid in SQL Server side reporting services. If you filter your query, you will get better performance than when you filter your dataset.

The only reason to filter in the layout should be to link a Tablix to a particular set of records in the dataset. This is done a lot in document reports.

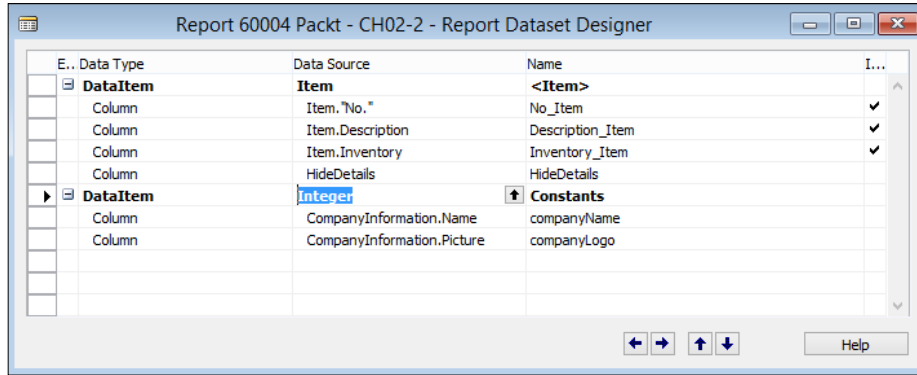


In some cases, this concept means that you might have to redevelop your report's dataset to allow for dynamic filtering. An example is the Customer top 10 report (111).

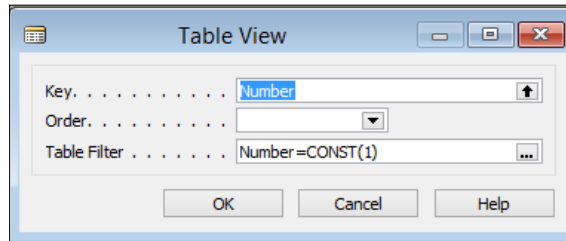
Top-ten filtering is done in the dataset and not in the layout, although doing it in the layout is very easy, much easier than how it is done in the report dataset designer. The way that this report is developed, as well as how to apply dynamic filtering using an integer data item or temporary table, will be explained in a later chapter.

In this section, I will focus on how you can apply sorting and filtering in the Tablix. Examples of how you can apply filters in C/AL code in the dataset are available in *Chapter 7, Performance Optimization Techniques*.

Imagine you have the following dataset:



In this dataset there are two data items: **Item** and **Integer**. The **Integer** data item, which I have named `Constants`, contains the company name and logo. It is filtered via its `DataTableView` property, as follows:



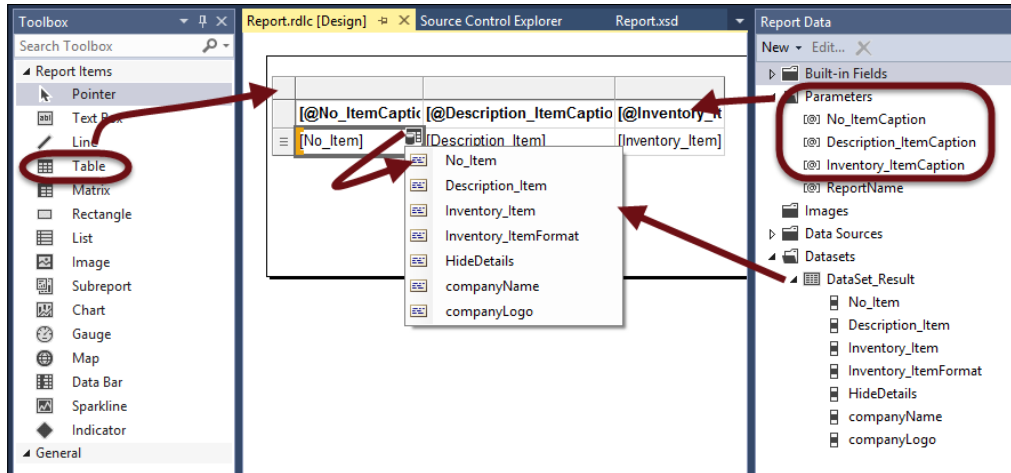
By applying this filter to the **Integer** table, I make sure that only one row is added to the dataset. Now I'm going to create the following layout:

[@No_ItemCaption]	[@Description_ItemCaption]	[@Inventory_ItemCaption]
[No_Item]	[Description_Item]	[Inventory_Item]



Instead of using an integer data item, in this example you can use the `Company Information` directly as a data item, because it's a setup table and it only contains one row. If you need to add any other information from other setup tables to the dataset later, you will need to redesign it, and use an integer data item. So, as a best practice, I recommend using an integer data item at the end or at the beginning of your dataset.

Next, in the layout, select a **Table** from the toolbox and drop it into the report body. Then, from the **Report Data** from the **Parameters**, drag and drop the **No**, **Description**, and **Inventory** in to the first row, or header row, of the Tablix. Then, in the second row, or detail row, click on the icon at the right top of the textbox so you can see a dropdown menu. There you can select the corresponding field from the dataset. Do this for all the columns, as in the following screenshot:



Save the layout in Visual Studio, import it into the dataset designer, save the report object, and run it. Then, the result will contain all the rows from the dataset and, because the Tablix should only show the columns with item values, the last row doesn't make sense:

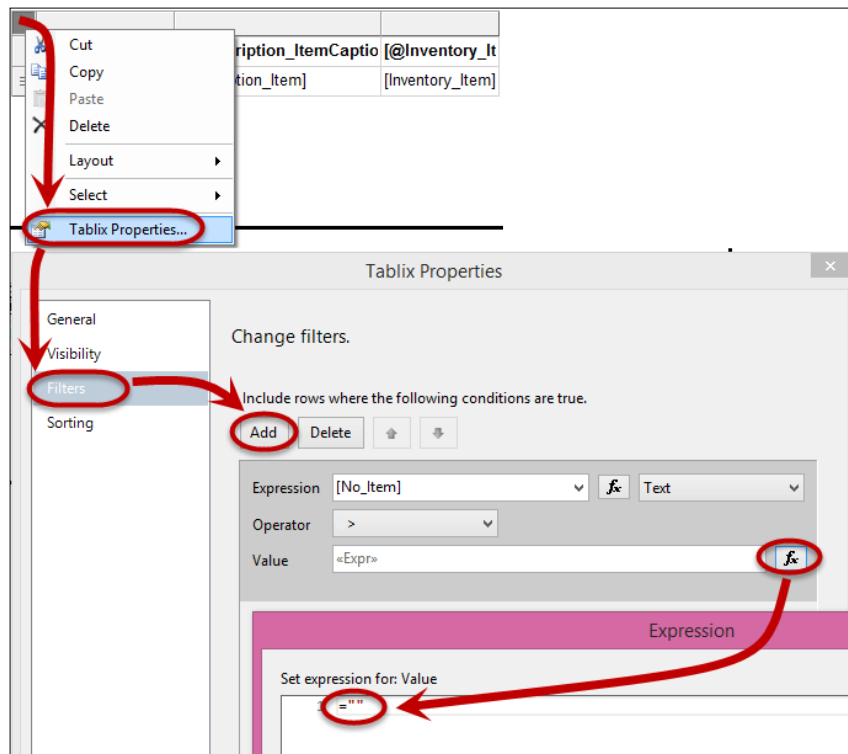
Item ID	Description	Quantity
LS-81	Loudspeaker, Walnut, 80W	0
LS-MAN-10	Manual for Loudspeakers	140
LS-S15	Stand for Loudspeakers LS-150	60
LSU-15	Base speaker unit 15" 100W	28
LSU-4	Tweeter speaker unit 4" 100W	100
LSU-8	Middletone speaker unit 8" 100W	15
SPK-100	Spike for LS-100	78



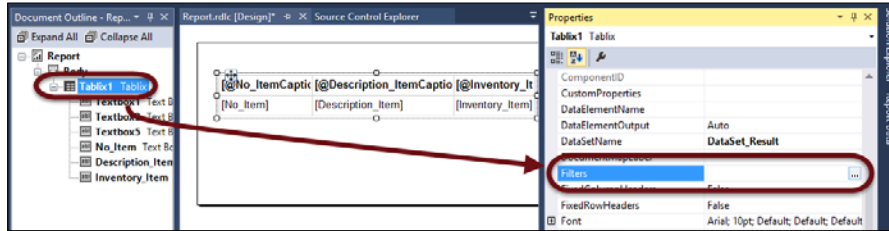
When you include multiple tables in the dataset, the Tablix will show all of the rows, and you have to make sure each Tablix is filtered to show only the rows that you are interested in showing.

This row at the end of the Tablix is the row from the integer data item. To filter it out of the Tablix, you can apply a filter in the Tablix properties.

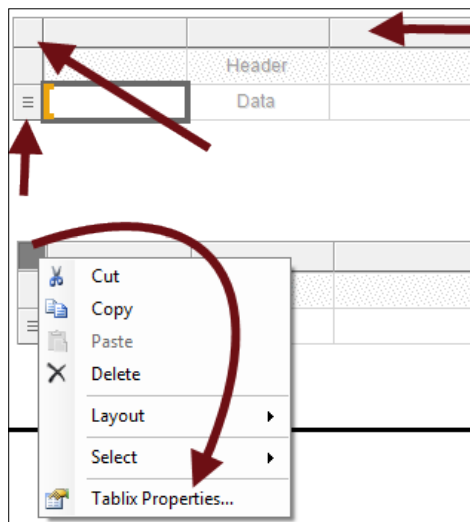
To open the Tablix properties, you first need to select the Tablix. You can select it with the mouse or select it in the dropdown box in the property list. When the table is selected you right-click the handle at the left top and select **Tablix Properties** in the dropdown menu that appears. You will then see the **Tablix properties** in a popup window, and can set the **Filters**, as shown in the next screenshot:



Another way to access the filters is by clicking the ellipsis button next to **Filters**, from the properties page, as in the following screenshot:



To see the Tablix handles, you need to select it, then the gray bars become visible. These bars are referred to as handles. There's a handle at the top, one on the left and a small box in the corner at the left top. If you right-click the handle in the corner at the left top, it displays a drop-down menu. In this menu you can open **Tablix Properties...**, as in the following screenshot.



If you right-click on the left top handle of the Tablix, you can open its properties and add a filter. The filter in the preceding screenshot filters the Tablix with a condition that says that the **Item Number** must not be empty, and so you filter out the rows that don't belong to the **Item Data Item**.

Filtering on values



If you type the character 0 in the **Value** cell, by default, this evaluates to the string 0. To compare a numeric expression with the number 0, use the expression syntax which begins with an equal sign: =0.

When previewing a report, you may see a runtime error from data type mismatches that may be similar to:

"The processing of FilterExpression for the [data set name] cannot be performed. Cannot compare data of types System.Int32 and System.String. Please check the data type returned by the FilterExpression."



This example is available in the object: Packt - CH02-2. You can run the report to see the final result, or you can remove the filter in the Tablix to see it with an empty row at the end.

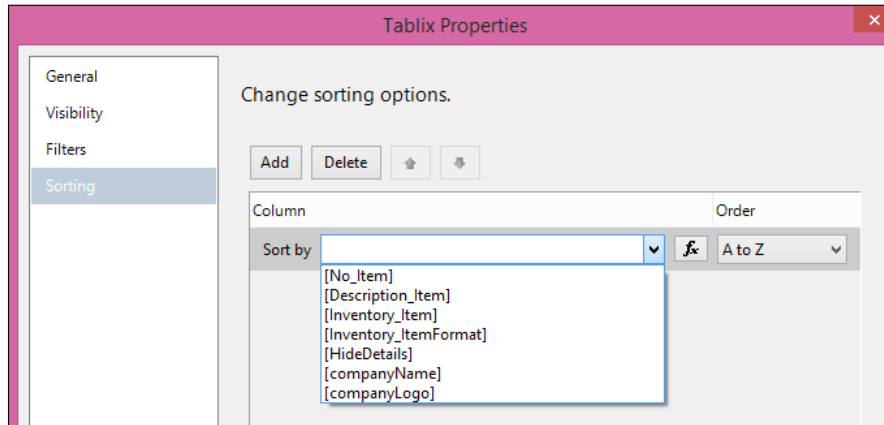
How can I implement sorting?

Sorting can be applied in the same window where you apply filters in the Tablix. Below **Filters** you can see **Sorting** and, in there, you can sort on any field from the dataset, or even an expression.

To sort the table, perform the following steps:

1. Select the **Sorting** tab in the **Tablix Properties** popup.
2. Select the field you want to sort on in the drop-down box in the column named **Expression**.
3. Select a **Direction**, ascending or descending, via the **Order** button (**A to Z** or **Z to A**).
4. Click on the **Ok** button to apply the sorting.

The following screenshot shows the sorting tab in the **Tablix Properties** popup window:



You can even define multiple columns on which to sort, if it is applicable. By using an expression to sort (or filter) you can make it dynamic. For example, you can use a parameter in the request page to let the user decide.

The **Sorting** and **Filters** that you apply in a Tablix are valid for all its members. This can be overridden if you are including one or more groupings in the Tablix.

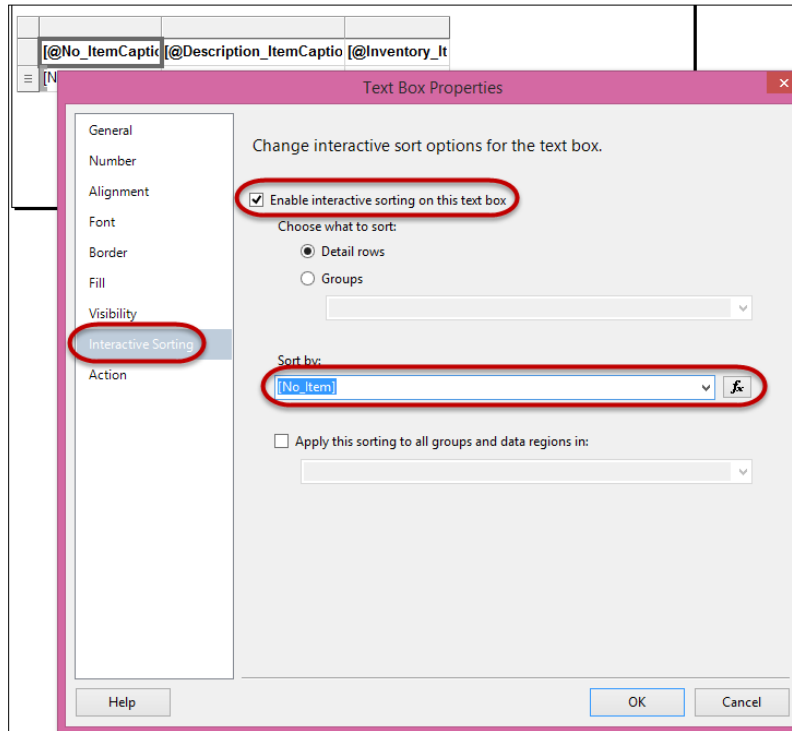


If you are applying sorting in the layout of your report, there's no point in giving the user the ability to sort in the request page via the sort button on the data item tab, because whatever the user selects will be overridden at runtime by the sorting you apply in the layout. In that case, it's better to remove the sorting button from the request page by selecting a key in the `DataItemTableView`. Otherwise you will be confusing the user.

Interactive sorting

With interactive sorting we enable the user to sort on any column in a Tablix by adding sorting buttons to certain textboxes.

Open the textbox properties and activate **Interactive Sorting** for the selected textbox, as shown in the following screenshot:



Usually, you activate the interactive sorting option for textboxes located in the Tablix header. You can choose any dataset field to sort on, it can even be an expression. The rows you will sort in this example are the Tablix detail rows. Once you have created a group in a Tablix, you can also select the group as the sort target.

If you apply interactive sorting on all the column headers in the Tablix, the result will look as follows:

No.	Description	Inventory
1000	Bicycle	32
1001	Touring Bicycle	0
1100	Front Wheel	152
1110	Rim	400
1120	Spokes	10000

As you can see, up/down arrows are added in every column. When you run the report, use the *Shift* and/or *Ctrl* buttons to combine multiple sorting.



Be consistent

Theoretically, it is possible to define interactive sorting for the **No** field when clicked and then sort on **Description**. This will confuse the user, so try to avoid it.



Interactive sorting is only available when you preview a report in the report viewer. It is not available when you print the report or export it to PDF, Excel, or Word.

The example report that includes filtering and interactive sorting is available in the object: Packt - CH02-3.

To apply dynamic filtering in the report, use the request page. The following screenshot shows how you can apply a filter on the item description field:

No.	Description	Inventory
1100	Front Wheel	152
1151	Axle Front Wheel	200
1200	Back Wheel	152
1251	Axle Back Wheel	10000
1320	Chain Wheel Front	100
1330	Chain Wheel Back	100
1710	Hand rear wheel Brake	200
1720	Hand front wheel Brake	200
1908-S	LONDON Swivel Chair, blue	305
1920-S	ANTWERP Conference Table	96

In the preceding example, the filter string is:

@*w*

The @ sign means it's case insensitive and the * sign means any character. So the applied filter says, any description that contains a w.

Grouping

When you add data to a Tablix, in many cases you will add groupings to create a different view on the data. When you add a group it is better to give it a proper name, so you know why it is there. You can create multiple and different types of groups in a Tablix and, in this section I will explain what a group is and the difference between parent-child and adjacent groups.

How can I implement grouping?

Groups can be created manually, but sometimes they are created automatically, for example when you use the wizard in report builder or when you drop a field on the grouping pane. Groups are also structured in hierarchies. The hierarchies are defined by the relationships and can be horizontal or vertical. When you create groups, the rows that they contain become dynamic.

In a Tablix, there are also static rows. Static rows don't belong to a group. Static rows are used to display totals or labels. They will only display once. A dynamic row is a part of a group, and is usually rendered more than once. Textboxes in a dynamic row usually contain aggregated data. The detail row in a Tablix is the innermost row and it displays the detail data, which is not aggregated.



More in-depth information about static and dynamic rows is available here:

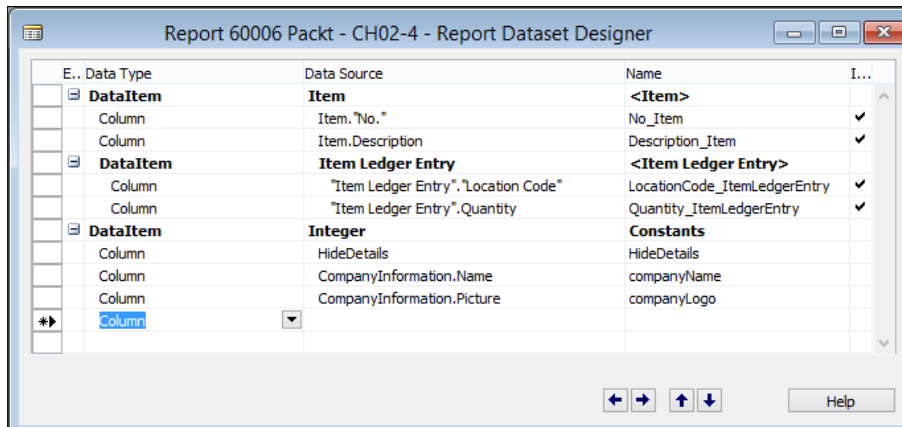
<https://msdn.microsoft.com/en-us/library/ee240753.aspx>

Adding a parent-child group to a Tablix

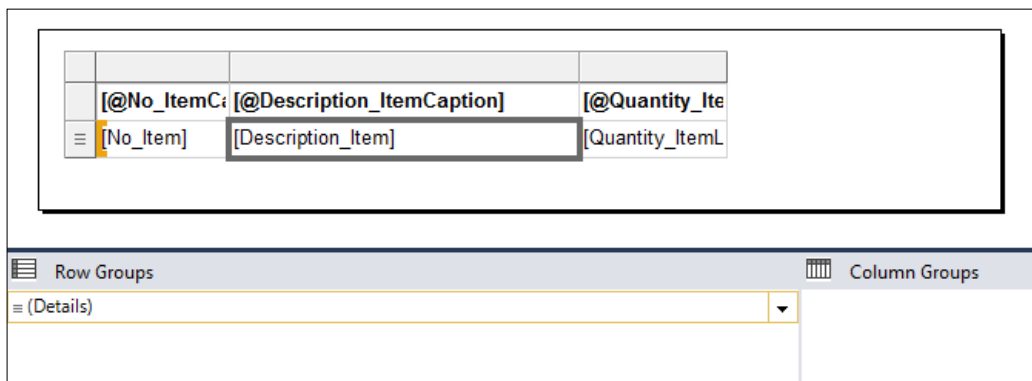
When you add a group to a Tablix you can do it in two ways, you can use a parent-child hierarchy or an adjacent one.

A parent-child hierarchy can be compared to a tree structure. You organize your data in a tree structure so that you can expand and collapse it. The idea behind it is that you want to create a summary of your data according to a particular field. For example, if you create a list of items and add a group to the location, you can then create a summary of the inventory by location.

1. To get started, create a new report with the following dataset:

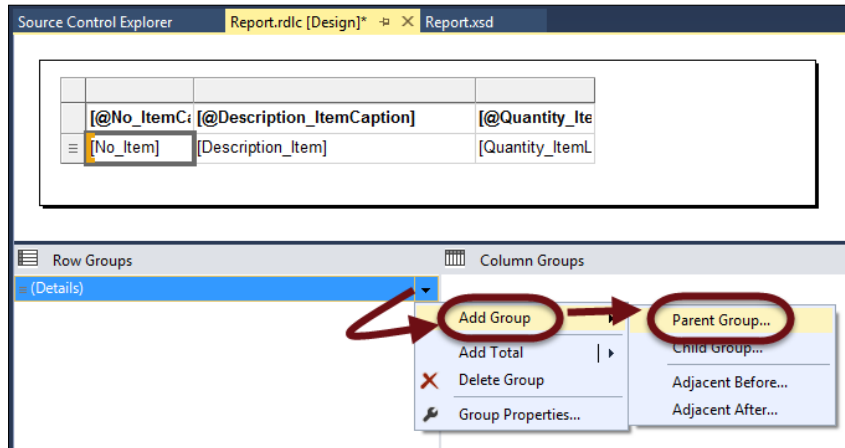


2. Then, in the layout add a Tablix to the body, that contains the following fields:

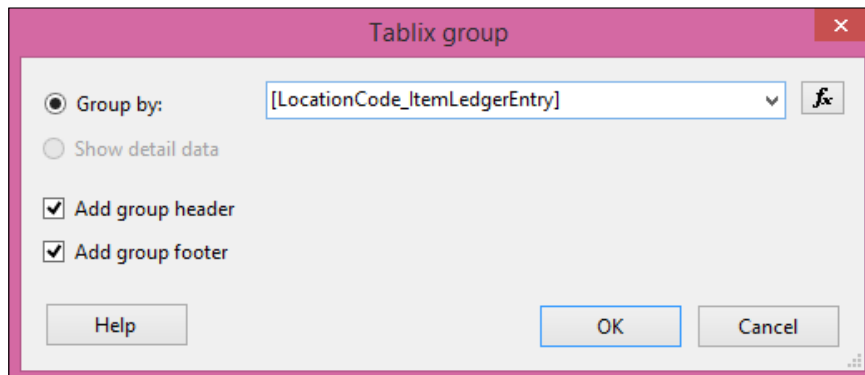


Let's have a look and see how we can create this type of grouping.

1. To create a group, click on the **Details** arrow in the **Row Groups** window:



2. In the window that opens, select the field to group on, like this:



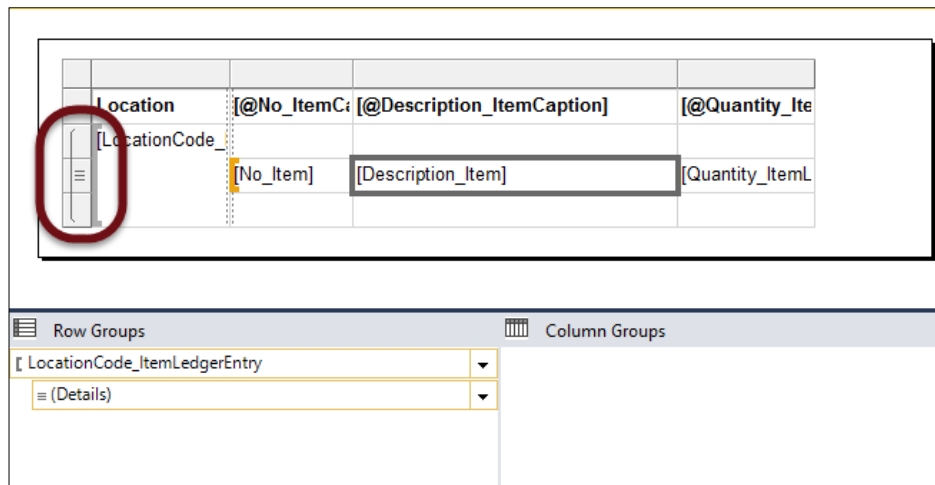
3. The option **Add group header/footer** adds an extra header and/or footer row to the Tablix. As you can see, when a group is added in a Tablix, it is made visible by the grouping line in the row handles.



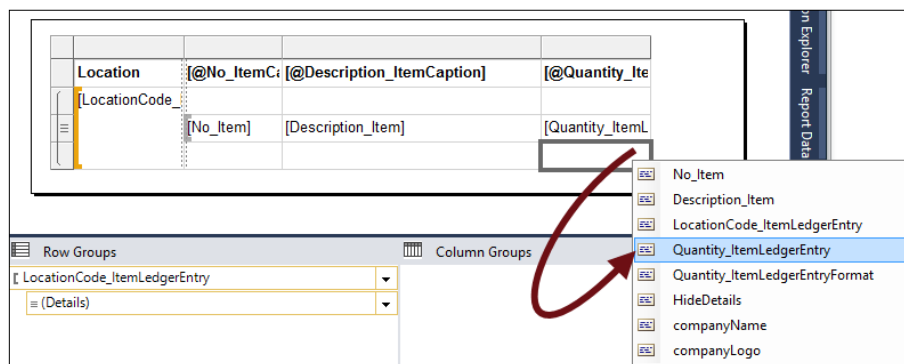
Row and column handles

Row and column handles become visible at the top and left side of a Tablix when you click on them. They are gray and you can click (or right-click) on them. The row handles also contain indicators to show the type of row.

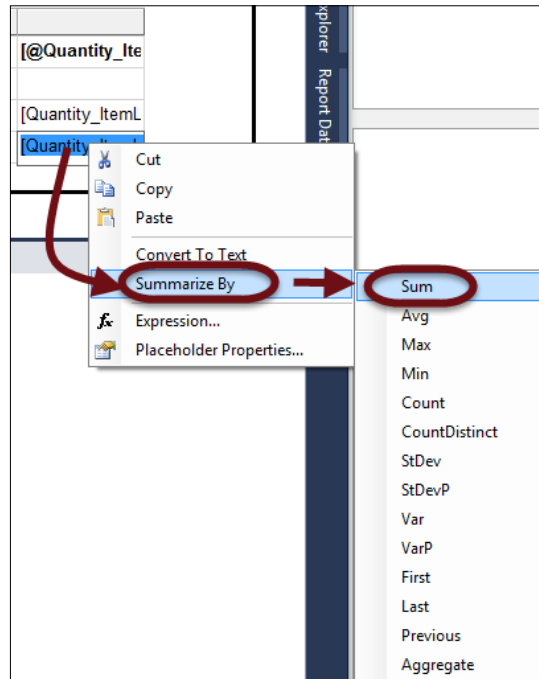
- This vertical line shows where the group starts and ends and it resembles a big bracket:



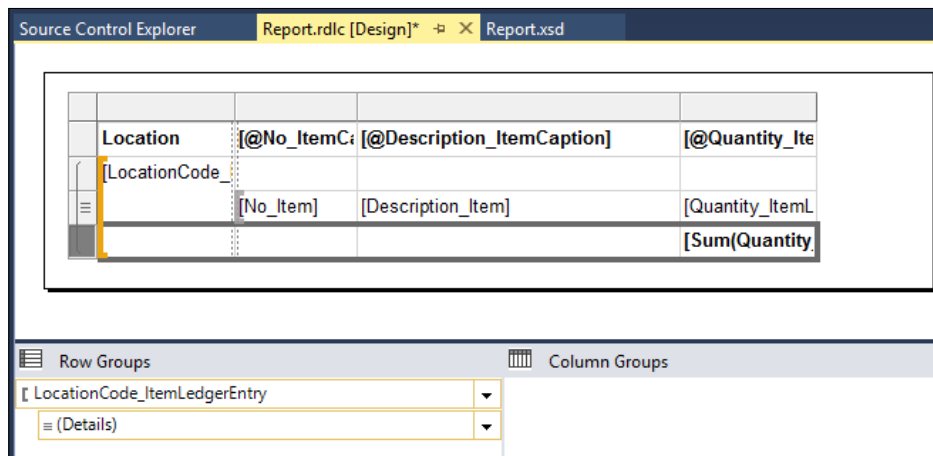
- Now you have two extra rows, a group header, and a group footer row. In the group footer row, below the quantity field on the detail row, select the quantity from the dropdown:



- Now, after you have selected the quantity field in the group footer row, make sure you aggregate the numerical values on the footer (or header) row, as in the following example:



- The row group's window now shows the group and the result is as follows:





If it is not already the case, I recommend changing the name of the group to a meaningful name. This is important when you are going to apply multiple nested groupings and makes it easier to see which group is which.

8. We have now added a group to our Tablix and, when we run the report, it will resemble this:

Location Code Item Ledger Entry	No.	Description	Quantity
BLUE			
	1900-S	PARIS Guest Chair, black	52
	1906-S	ATHENS Mobile Pedestal	70
	1908-S	LONDON Swivel Chair, blue	234
	1920-S	ANTWERP Conference Table	38

	8924-W	Server - Enterprise Package	-1
	LS-75	Loudspeaker, Cherry, 75W	11
			40122
GREEN			
	1896-S	ATHENS Desk	49
	1896-S	ATHENS Desk	-1
	1896-S	ATHENS Desk	1

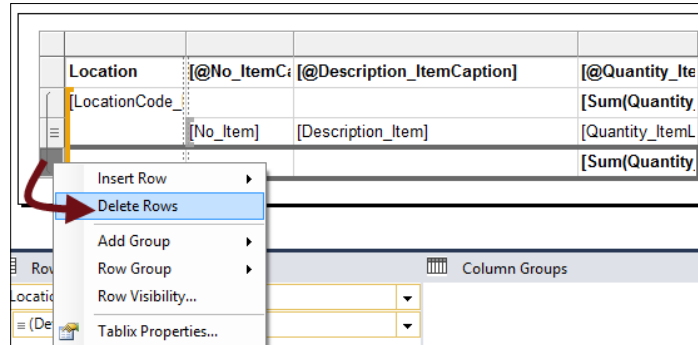
9. Next, I will remove the footer row and put the total in the header. Most managers find it more interesting to start a report with the totals and then the details. To do this, you can copy past the field from the footer row to the corresponding textbox in the header row:

Location	[@No_ItemCaption]	[@Description_ItemCaption]	[@Quantity_ItemCaption]
[LocationCode]			[Sum(Quantity)]
	[No_Item]	[Description_Item]	[Quantity_ItemCaption]
			[Sum(Quantity)]

1 Ctrl + C

2 Ctrl + V

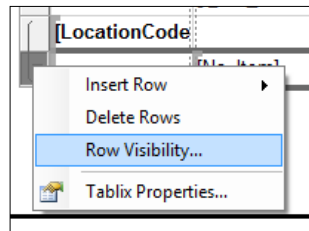
10. Then, select the footer row, right-click, and select **Delete Rows**:



How do I implement expand/collapse?

To make it even more interesting, I will now add an expand/collapse option to the Tablix, so you can show or hide the details. To do this, you use the **Hidden** property.

1. Every textbox has a **Hidden** property which you can either set to **True** or **False**, or to an expression. Instead of setting the **Hidden** property of each individual textbox in the details to **False**, I will do it via the row visibility. Right-click the detail row handle and select **Row Visibility...**:



2. In the row visibility window that opens, select the option, **Show** or **Hide**, based upon an expression. I will use this as an expression:
`=Last(Fields!HideDetails.Value, "DataSet_Result")`

This means that the visibility of the detail row now depends on what the user selects for the **HideDetails** option on the request page.



The reason for the Last function is that the **HideDetails** field is added as the last row in the dataset designer.

- Then, select the **Location Code** field as the toggle item, as in the following screenshot:

A toggle item is the name of a textbox on which you can click to change the **Hidden** property. So, by doing this, we enable the expand/collapse option on our group, allowing the user to show or hide details dynamically when running the report.


- Next, set the `InitialToggleState` property of the textbox that contains the `LocationCode` to:

```
=NOT Last(Fields!HideDetails.Value, "DataSet_Result")
```


This will make sure that, when the user selects `HideDetails` (or not), the `+` or `-` icon is correctly displayed. The `InitialToggleState` property of a textbox determines the initial state of the toggle image. When you run the report, it now looks as follows:

Location Code Item Ledger Entry	No.	Description	Quantity
<input type="checkbox"/> BLUE			40122
<input type="checkbox"/> GREEN			5894
<input type="checkbox"/> OUT. LOG.			29
	1896-S	ATHENS Desk	25
	1936-S	BERLIN Guest Chair, yellow	4
	70001	Base	15
	70001	Base	-15
	70002	Top Panel	3
	70002	Top Panel	-3
	70003	Rear Panel	31
	70003	Rear Panel	-31
<input type="checkbox"/> OWN LOG.			1365
<input type="checkbox"/> RED			528,52480
<input type="checkbox"/> WHITE			848
<input type="checkbox"/> YELLOW			806

In the request page, the user can select the `HideDetails` option to select the default value of the `Hidden` property for the detail row. At runtime the user can click on `Location Code` fields to expand or collapse the row and show or hide the details dynamically.

 An example of the finished report is available in the object: Packt - CH02-4. An example of the starter object for this report is available in the object: Packt - CH02-8.

Adding an adjacent group to a Tablix

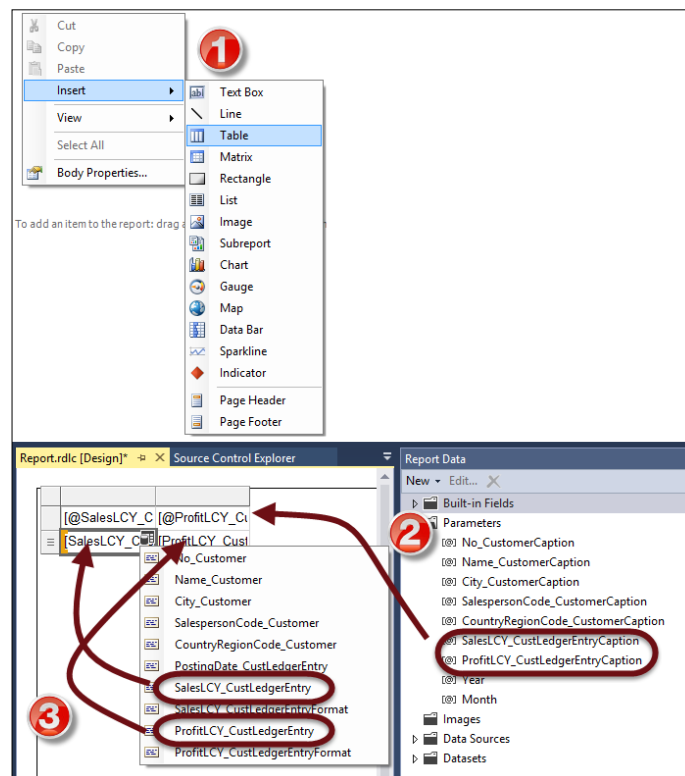
Now, let's have a look at how we can use an adjacent grouping in a report. You can start by importing the object: Packt - CH02-9.

For this new report, I will use the following dataset:

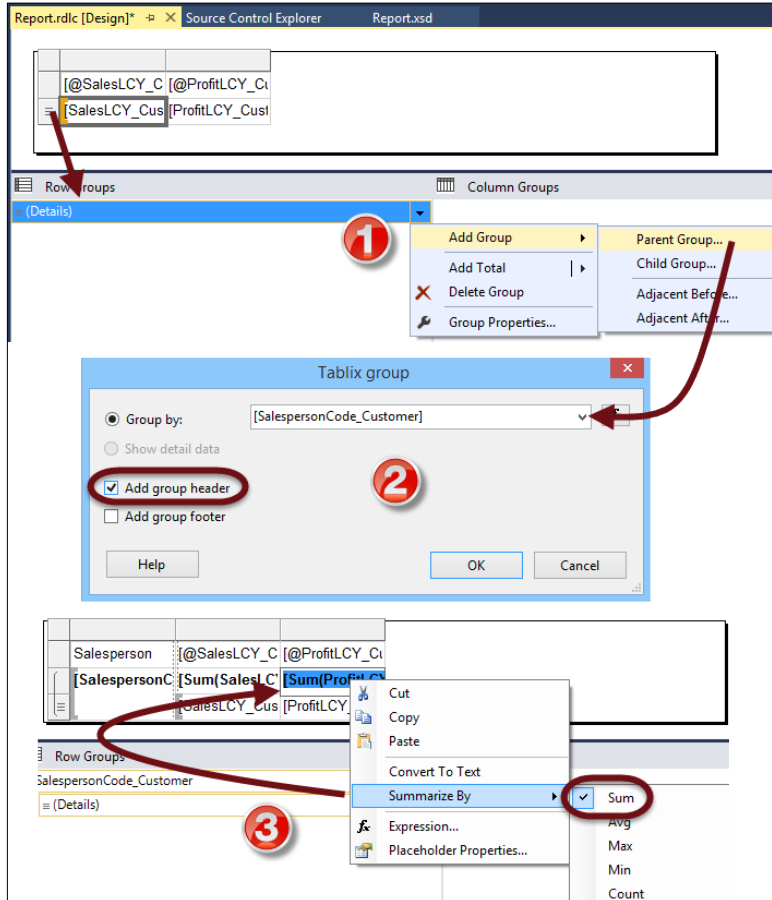
E.. Data Type	Data Source	Name	I...
DataItem	Customer	<Customer>	
Column	Customer."No."	No_Customer	✓
Column	Customer.Name	Name_Customer	✓
Column	Customer.City	City_Customer	✓
Column	Customer."Salesperson Code"	SalespersonCode_Customer	✓
Column	Customer."Country/Region Code"	CountryRegionCode_Customer	✓
DataItem	Cust. Ledger Entry	<Cust. Ledger Entry>	
Column	"Cust. Ledger Entry"."Posting Date"	PostingDate_CustLedgerEntry	
Column	"Cust. Ledger Entry"."Sales (LCY)"	SalesLCY_CustLedgerEntry	✓
Column	"Cust. Ledger Entry"."Profit (LCY)"	ProfitLCY_CustLedgerEntry	✓

To create the layout follow these steps:

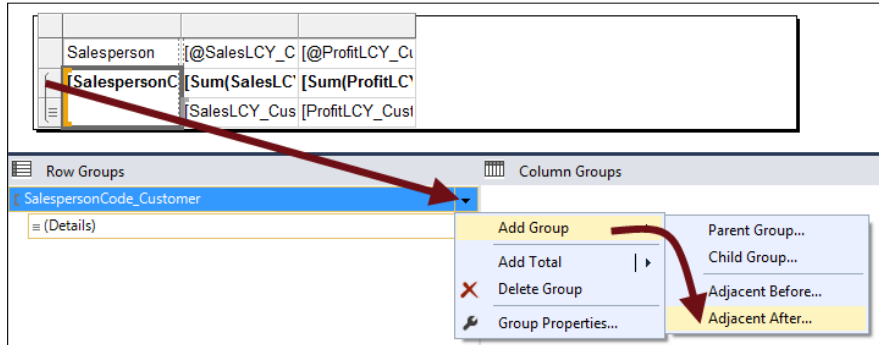
1. In the layout, you start by adding a Tablix and then, in the detail row, you add the Sales LCY and Profit LCY as fields:



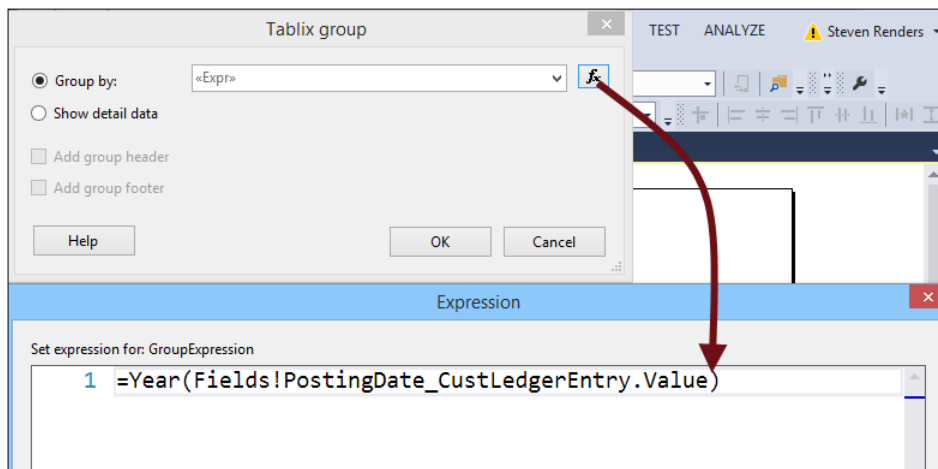
- Next, you add a parent group on [SalespersonCode_Customer], just as we did in the previous report:



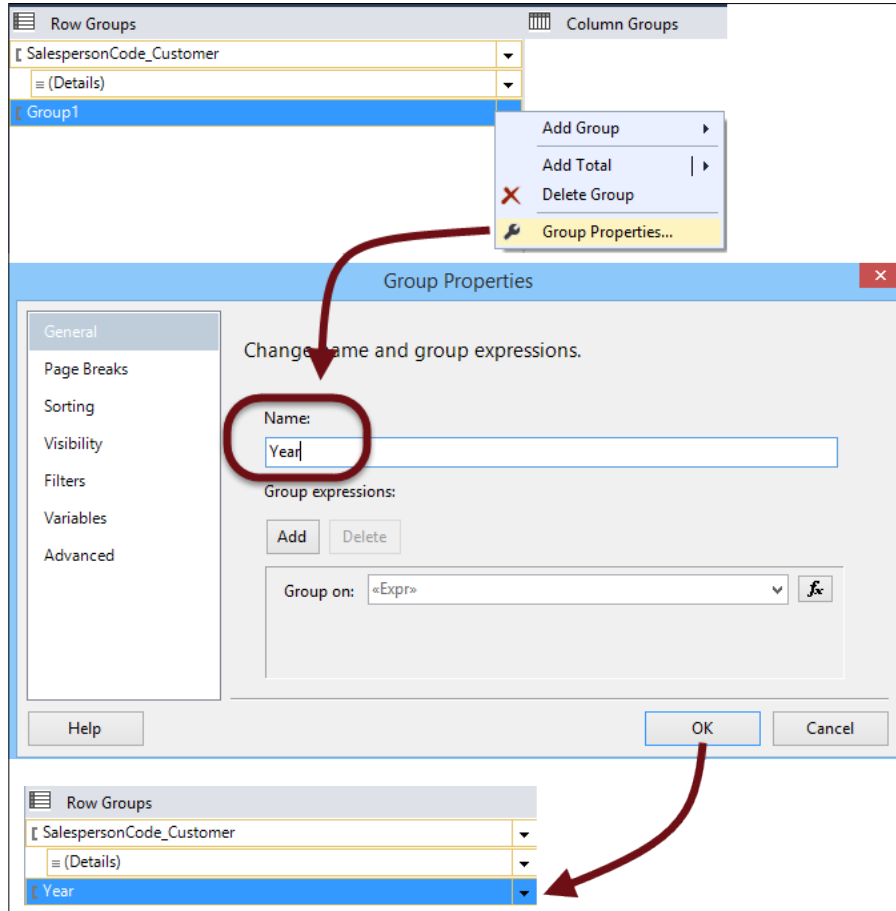
- Then you add a new group but, instead of selecting a parent-child group, you select **Adjacent After...**:



- For the group expression, use the following expression:
`=Year(Fields!PostingDate_CustLedgerEntry.Value)`
- The result of the preceding expression is shown in this screenshot:



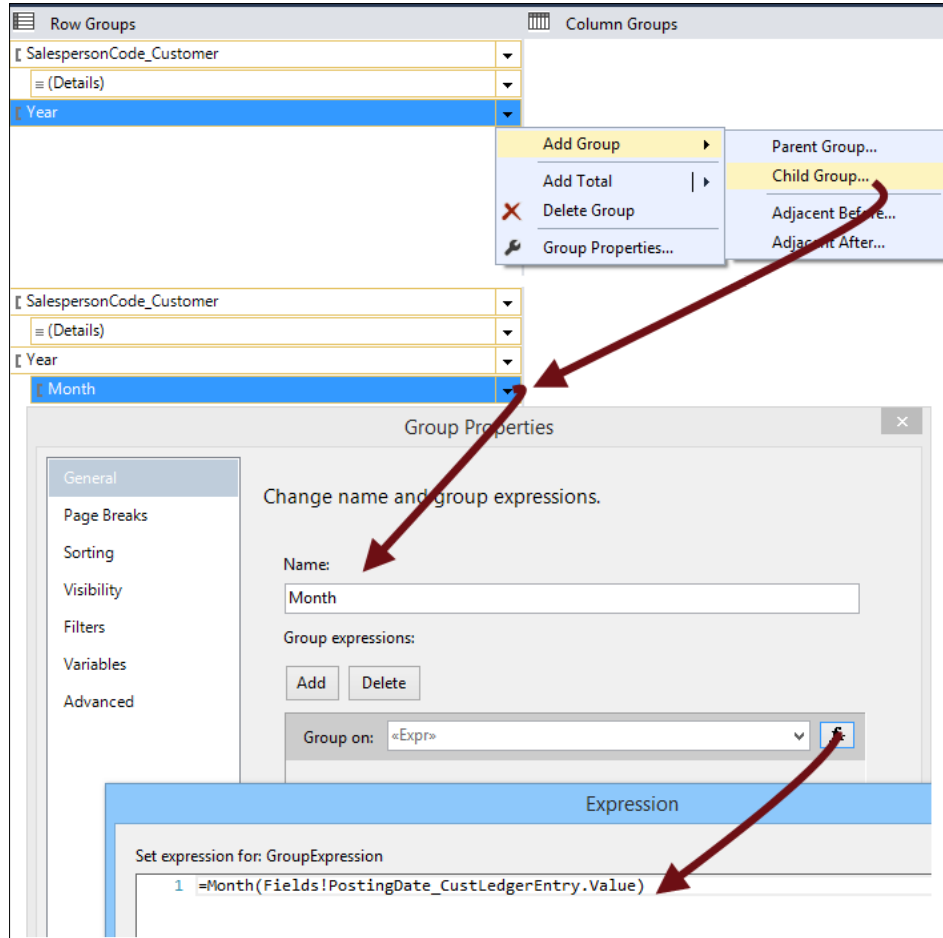
- Then, rename the group to Year:



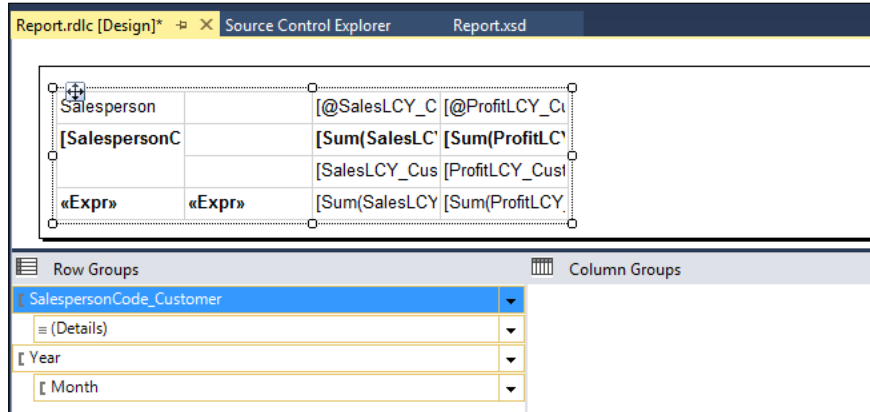
- Now add a child group below the Year group, name it Month, and use the following expression:

`=Month(Fields!PostingDate_CustLedgerEntry.Value)`

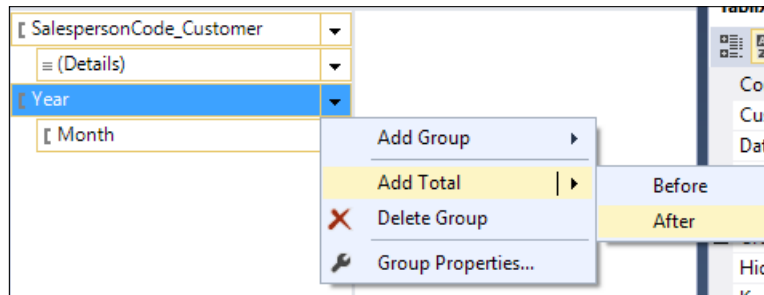
8. The result of the preceding expression is shown in this screenshot:



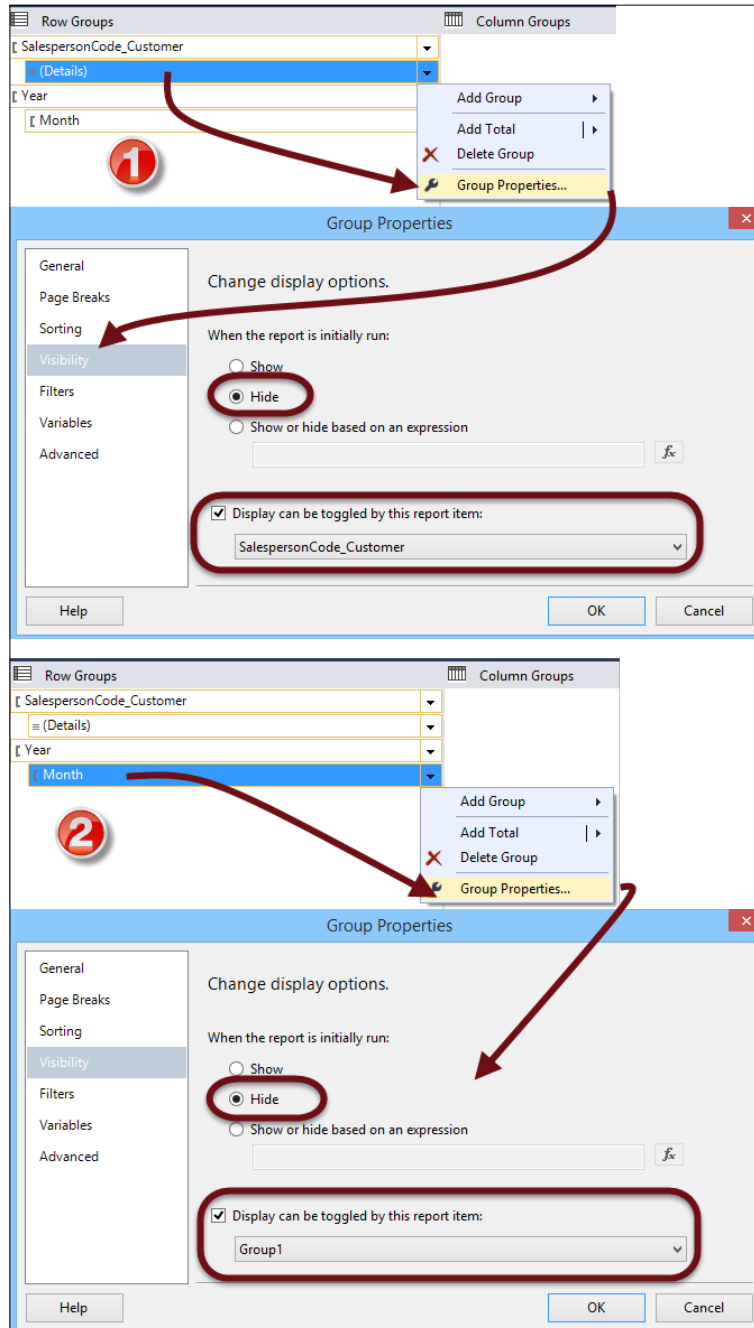
- In the group fields, select the **Sum** aggregation for the Sales LCY and Profit LCY fields. You will now have a report layout that resembles this:



- To add a row at the end of the report showing a total, go into the **Row Groups** and add a total, as in this screenshot:



11. Now, as in the previous report, use the row visibility and toggle properties to create an expand/collapse option for all groupings:



12. Then run the report. It will look like the following screenshot:

Salesperson Code	Customer		Sales (LCY)	Profit (LCY)
IR			60311,61	27553,14
			1499,02	305,12
PS			31351,00	12367,70
			0	0
	2016	12	1596,5	1596,5
	2017	1	90066,11	38324,34

In this report, we now have the details, first grouped by salesperson, and then below it by year and month. This gives the reader a different view of the same data, all in one Tablix. That's the power of using adjacent groupings.



Adjacent groupings also have other advantages. You can use them to offer alternatives in document reports, for example. By combining them with group filters you can use one Tablix to show information from different data items, instead of using multiple tables, as in standard reports.

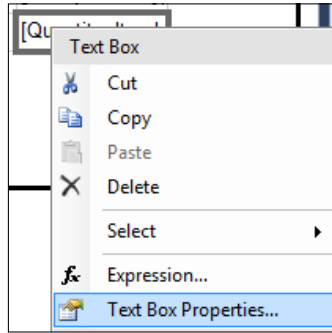
An example of the report with adjacent groupings is available in the object: Packt - CH02-5

Formatting report items

Now we need to have a look at formatting because, as you will have noticed, the amounts or quantities in the report are not formatted in the way we are used to in Dynamics NAV. This is because the dataset that is generated by Dynamics NAV contains the numerical values without formatting. It sends a separate field with a format code that can be used in the format properties of a textbox in the layout. Numerical fields have a `Format` property. This `Format` property is populated by Dynamics NAV and contains, at runtime, an RDL format code that you can use in the `Format` property of a textbox in Visual Studio.

To get started with formatting, perform the following steps:

1. When you right-click on a textbox, a menu appears in which you can select the properties of the textbox, as shown in the following screenshot:



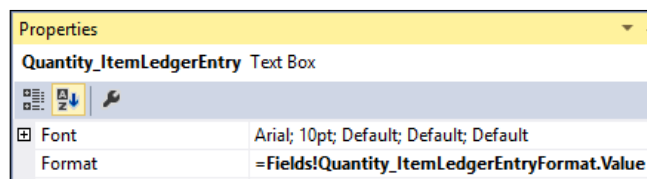
I'm using object: Packt - CH02-4 as the example for formatting but, of course, this can be applied in any report that has numerical fields.

2. In the **Textbox Properties** window, go to **Number** and then select **Custom**. Click on the **Fx** button to open **Expression Designer** and type an expression. The result of the expression will be the value of the property. In this case, our expression should fetch the value from the format field from the **Quantity** field. The expression will be:

```
=Fields!Quantity_ItemLedgerEntryFormat.Value
```

This means that the format of the textbox is fetched from the dataset field: `Quantity_Item`.

3. Instead of using **Expression Designer**, you can also just type this expression directly into the **Format** code textbox or in the **Format** property in the properties window of the textbox, as shown in the following screenshot:



Reporting Services and RDLC use .NET Framework formatting strings for the **Format** property of a textbox. The following is a list of possible format strings:

- C: Currency
- D: Decimal
- E: Scientific
- F: Fixed point
- G: General
- N: Number
- P: Percentage
- R: Round trip
- X: Hexadecimal

4. After the format string, you can provide a number representing the amount of digits that have to be shown to the right of the decimal point.

For example:

F2 means a fixed point with 2 digits: 1.234,00 or 1,234.00

F0 means a fixed point with no digits: 1.234 or 1,234

The thousand and comma separators (. and) that are applied, and the currency symbol, depend on the Language property of the report.



More information about .NET Framework formatting strings can be found here:

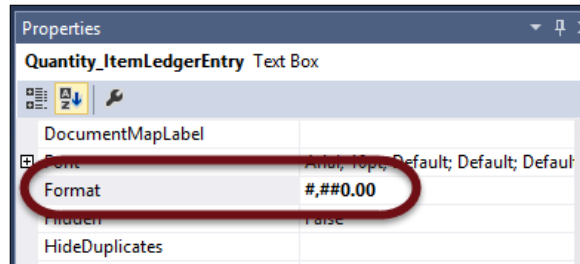
Custom Numeric Format Strings: <http://msdn.microsoft.com/en-us/library/0c899ak8.aspx>.

Standard Date and Time Format Strings: <http://msdn.microsoft.com/en-us/library/az4se3k1.aspx>.

As an alternative, you can use custom format strings to define the format value. This is actually how Dynamics NAV populates the **Format** fields in the dataset. The syntax is:

`#,##0.00`

You can use this to define the precision of a numeric field. The following image provides an example:



Why does the Format property sometimes have no effect?



To apply formatting to a textbox, the textbox must contain an expression, for example `=Fields!LineTotal.Value` or `=1000`.

When the text in the textbox does not begin with the `=` sign, then the text is interpreted as a string and formatting does not apply.

You can also set the format in the report dataset designer, instead of in the layout. You can do this by using the `Format C/AL` function. You can do this directly in the dataset in the `SourceExpression` of any field, or you can do it in the data item triggers, for example the `OnAfterGetRecord()` trigger. But, if you use an expression in the `SourceExpression`, you lose the option to use the `IncludeCaption` property.



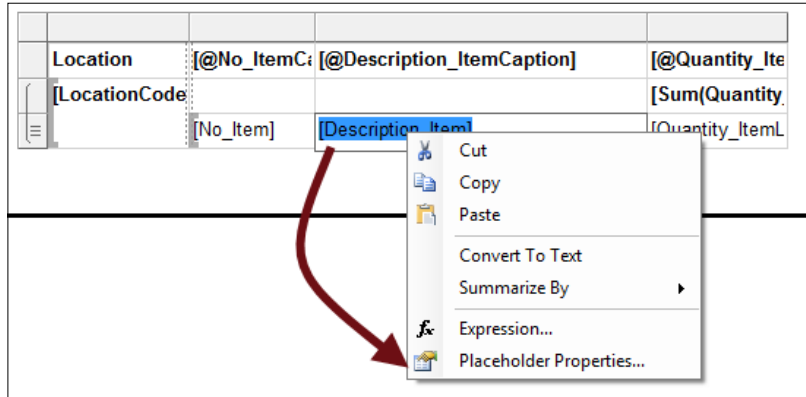
A good example of a textbox format property is available here:


<http://thinkaboutit.be/2015/06/how-do-i-implement-blankzero-or-replacezero-in-a-report>

Using placeholders

If you select a textbox and right-click on it, you open the textbox properties, as you have already seen in this chapter. But, inside the textbox, there's the placeholder. A placeholder is the text, or expression, that becomes the information displayed in the textbox at runtime. And the placeholder also has a set of properties that you can set. So you can consider a placeholder as an entity inside a textbox, with its own set of properties, which are, by default, inherited from its parent, the textbox.

The following screenshot shows that, when you right-click on the text in a textbox, you can then select its placeholder properties:



[ The example report with the placeholders is available in the object: Packt - CH02-6]

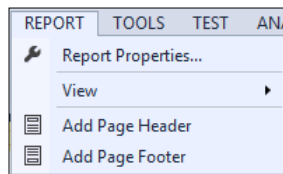
A textbox can contain one or more placeholders. By using multiple placeholders in one textbox, you can display multiple fields in one textbox, and give them different properties.

In the following example, I will add a header to the report, and in the header, I will display the company information.

To add a header (and/or footer) to a report, go to the **Report** menu and select:

- **Add Page Header**
- **Add Page Footer**

The following screenshot shows an example of this:



A report can contain a maximum of one header and one footer.



As an alternative you can right-click anywhere in the body of the report, in the empty space to the left or right of the body, and add a page header or footer.

The page header and page footer are always shown on every page, except if you decide not to show it for the first and/or last page by using the properties:

- **PrintOnFirstPage**
- **PrintOnLastPage**



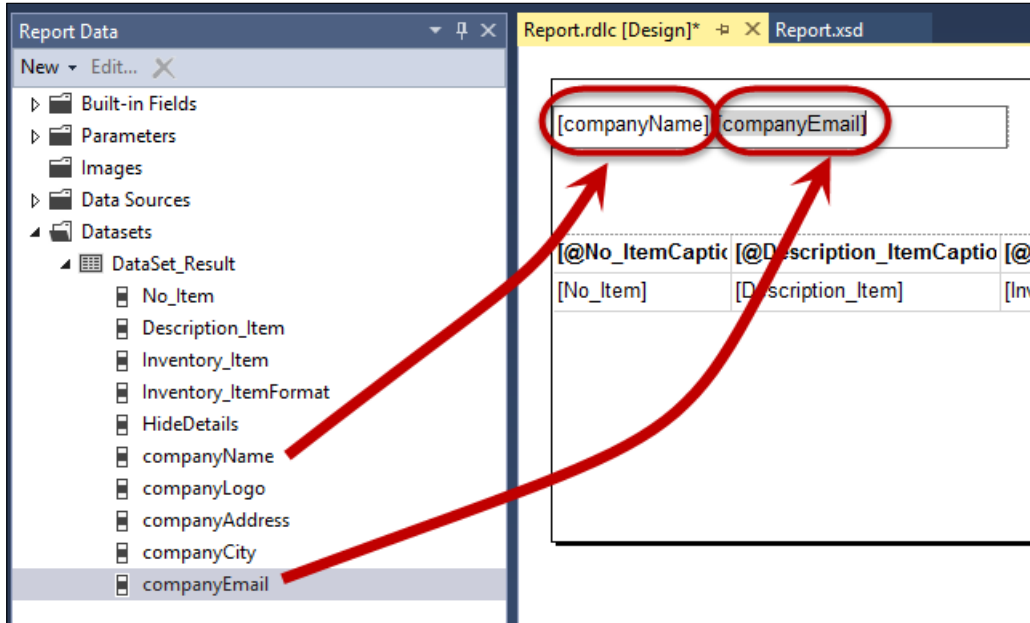
Dynamically hiding a page header/footer

A page header and footer cannot be hidden dynamically. A workaround would be to put a rectangle in the page header and/or footer and use the Hidden property of the rectangle to show or hide the content of the header/footer dynamically. You need to be aware that, even when you hide the content of the page header/footer, the report viewer will preserve the space. This means that the header/footer is still displayed, but will be empty.

A page header or footer cannot contain a data region. The only controls you can add to a page header or footer are:

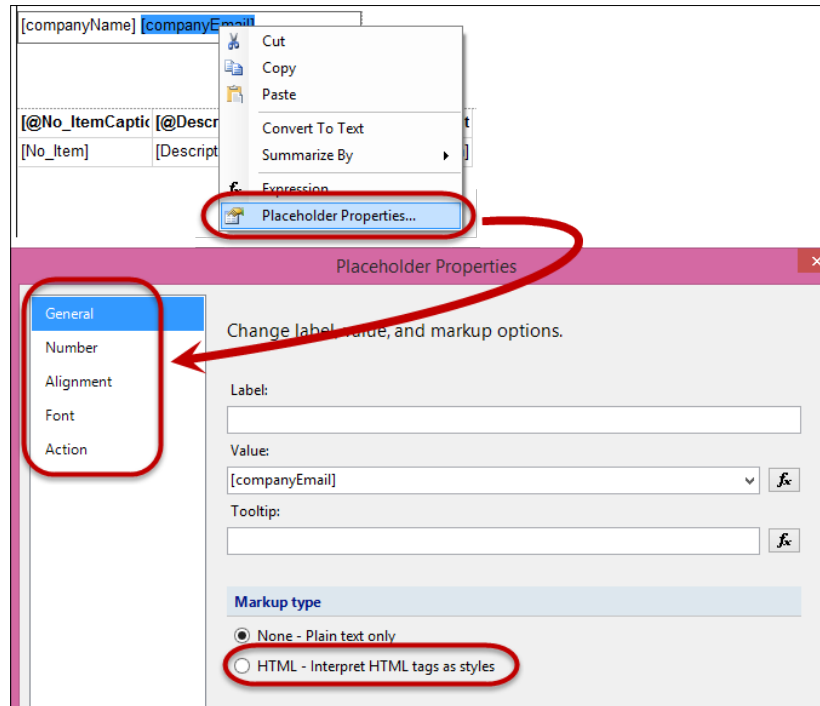
- Textbox
- Line
- Rectangle
- Image

So, in the page header, I will add a textbox with a placeholder, as in the following screenshot:



To do this, add a textbox in the page header. Then, drag a field from the dataset into the textbox. Then, add one or more spaces and drag another field into the same textbox. You will notice the two fields can be selected inside the textbox and when they are, they become gray. If you right-click on the placeholder, you can see its properties.

This is how you can see that it is a placeholder.



It is interesting that the mark-up type for a placeholder can be changed to HTML. This means that, if the placeholder contains HTML, it will be recognized by the report viewer and rendered, as it would be by a browser. The HTML tags that are recognized are the following:

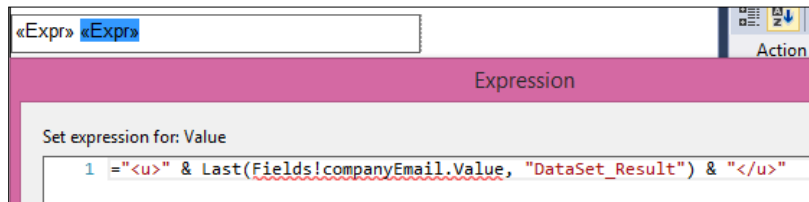
- <A href>
-
- <H{n}>, <DIV>, , <P> ,
- <DIV>, , <HN>
- , <I>, <U>, <S>
- , ,

If you use these HTML tags in a badly organized way then they will be interpreted as text and rendered as such.

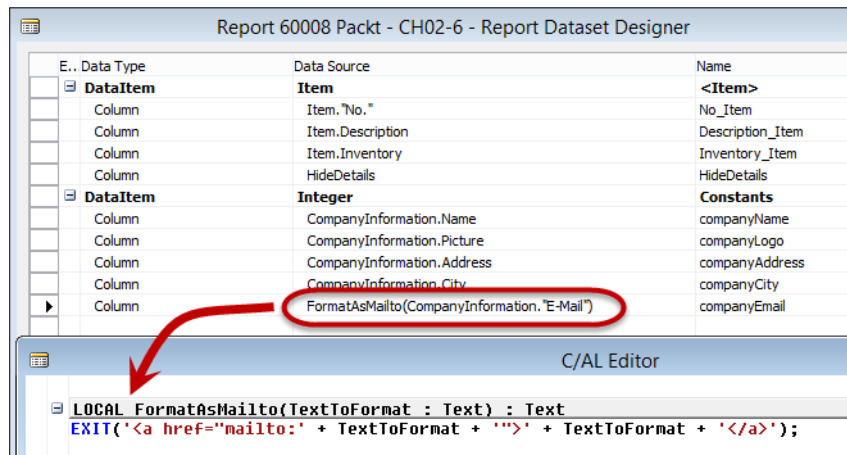


The possibility of using HTML in placeholders creates an opportunity for Dynamics NAV developers. What you can do, for example, is generate the HTML tags in C/AL code and send them to the dataset. By using this approach, you can format text and manage it dynamically via C/AL. You could even use a special setup table in which you let users decide how certain fields should be formatted.

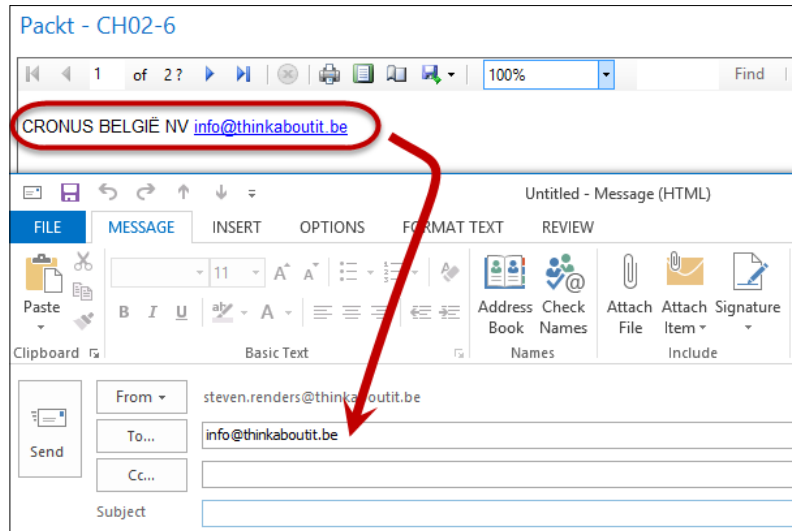
In our example report, I will format the company e-mail address in two ways. First, I will use the placeholder expression to underline the text:



Then, I will go to the C/AL code and create a function that will format the e-mail address using a `mailto` hyperlink:



When you run the report, the result is this:



The e-mail address is underlined and there is also a hyperlink and when you click on it, your e-mail client opens. As you can see, the formatting in the placeholder and the formatting in the C/AL code are combined.



Use a code unit or buffer table

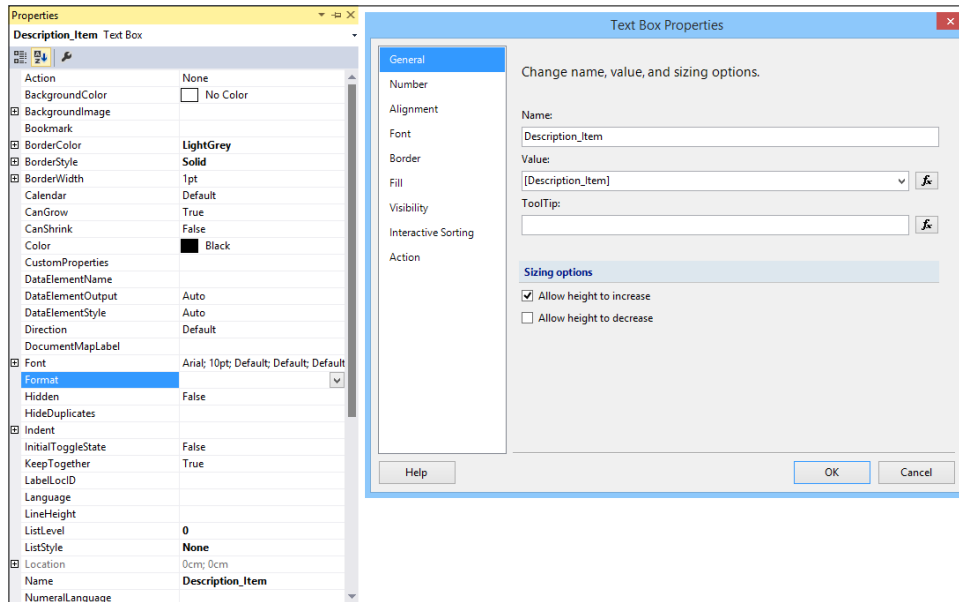
In this example I used a custom function in the report (`FormatAsMailto`). In real life, it is better to create these types of functions in a separate code unit, or buffer table, so you can reuse them in other reports.



The example report with the placeholders is available in the object: Packt - CH02-6

Important properties – CanGrow and CanShrink

A textbox has many properties, as you can see in the following screenshot.



If you right-click a textbox and select the textbox properties, they will open in a separate popup window. In this window, some of the textbox properties are available and they are divided into categories. To see all of the textbox properties, you can use the properties window, which is usually on the right in Visual Studio. Here you can sort the properties or group them using the buttons on top:



The first button groups the properties. The second button sorts the properties and the third button opens the properties popup window.

I am not going to discuss all of the properties, but I would like to draw your attention to CanGrow and CanShrink. These two properties can be set to True or False. If you set CanGrow to True then the height of the textbox will increase if the text, at runtime, is bigger than the width of the textbox. With CanShrink, the height of the textbox may shrink.



I do not recommend these properties, except when really necessary. When a textbox grows, the height increases and it pushes the content down below. This makes it difficult to predict if the content of the report will still fit on the page. Also, the effects of CanGrow and CanShrink are different if you run the report in Preview and export it to PDF, Word, Excel, or if you print the report.

Example – create an item dashboard report

In this example, I am going to create an item dashboard report. Actually, I will create a first version of the dashboard, in later chapters I will come back to this report and enhance it.

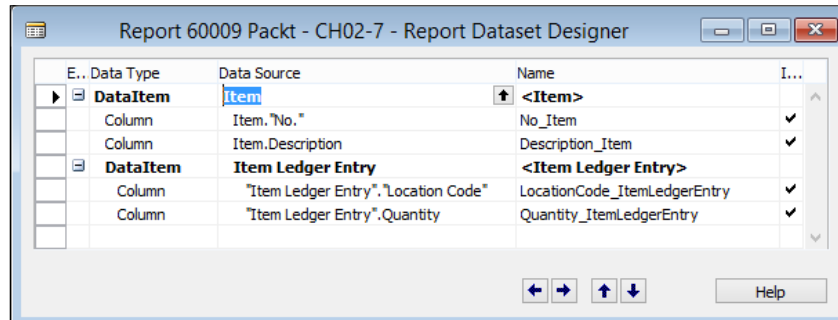
The result of the report looks like the following screenshot:

	BLAUW	GEEL	GROEN	OUT. LOG.	OWN LOG.	ROOD	Total
ALBERTVILLE Whiteboard, groen (1992-W)	6		5			-1	10
AMSTERDAM Lamp (1928-S)	149	97	-19			56	282
ANTWERPEN Vergadertafel (1920-S)	38		66			3	106
ATHENE Mobiel onderstel (1906-S)	70		88		40	56	254
ATHENE Tafel (1896-S)		160	49	25		20	254
ATLANTA Whiteboard, basis (1996-S)	44	116	-1			22	181
BERLIJN Bezoekersstoel, geel (1936-S)	36		46	4	0	50	136
CALGARY Whiteboard, geel (1988-W)		13	8			5	26
CHAMONIX Bergmeubel basisunit (1924-W)	1	15	8			2	26
GRENOBLE Whiteboard, rood (1968-W)		10	4			4	18
INNSBRUCK Kast/G.Deur (1964-W)	21	8	27			-2	54
INNSBRUCK Kast/vv.deur (1976-W)	3	3	-2			-106,4752	-102,4752
LONDEN Draaistoel, blauw (1908-S)	234		57		0	14	305
MEXICO Draaistoel, zwart (1968-S)	236		14			15	265
MOSKOU Draaistoel, rood (1980-S)	66		14			21	100
MÜNCHEN Draaistoel, geel (1972-S)	37	90	-1			-4	122
OSLO Boekenkast (1952-W)	9		-1			7	15
PARIJS Bezoekersstoel, zwart (1900-S)	52	160	41			48	299
ROME Bezoekersstoel, groen (1960-S)	153					24	177
SAPPORO Whiteboard, zwart (1972-W)	4		2			5	11
SARAJEVO Whiteboard, blauw (1984-W)	3		3			4	10
SEOUL Bezoekersstoel, rood (1988-S)	41	43	83				167
ST.MORITZ Ladekast (1928-W)	4	41	23			-1	67
TOKYO Bezoekersstoel, blauw (1964-S)	59		60		25	29	173
Total	1.265	766	573	29	65	267,5248	2.965,5248



The report is available in the object: Packt - CH02-7

What we need to do is to show the inventory of a list of items by location. The report also includes totals and subtotals of the inventory by location, by item and a grand total. To start, you define a dataset, as follows:

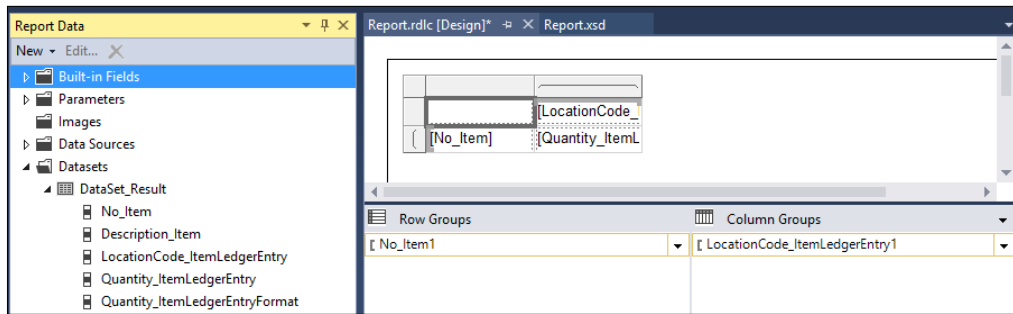


In this dataset, I will start with the item table and, per item, fetch the item ledger entries. The inventory is the sum of the quantities of the item in the item ledger entry table. I have also included a filter, using the `PrintOnlyIfDetail` property of the item data item. This means that, if an item does not have any ledger entries, it will not be shown in the report. Also, I'm using the item ledger entry table to get the location code and quantity fields. In the report layout, I will create a group and calculate the inventory via an aggregate function.



In real life, there might be many items and ledger entries, so this approach is not the best one. It would be better to use a buffer table or query object, and calculate the inventory and filter in the dataset, instead of in the layout. I will show you an example of this approach in a later chapter. At this point, my objective is to demonstrate how you can use a Matrix-Tablix to create a layout that has a dynamic number of rows and columns.

1. Once you have defined the dataset, open the layout and add a matrix control to the report body. In the data cell, use the **Quantity** field, on the row, use the **Item No**, and on the column, use the **Location Code**. This will create the following matrix and groups:

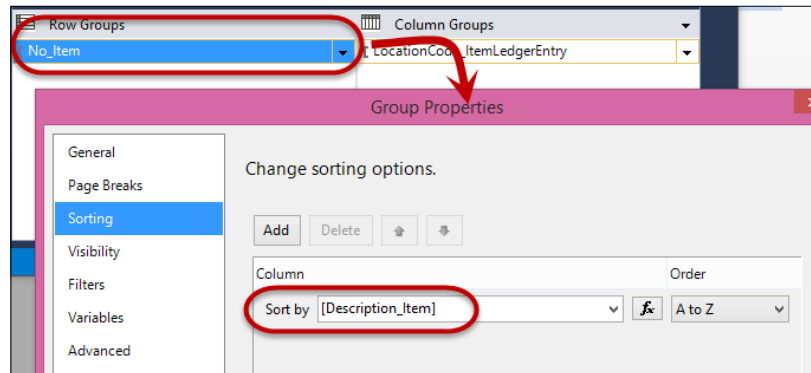


- Next, modify the expression of the textbox that contains the item number, to the following expression:

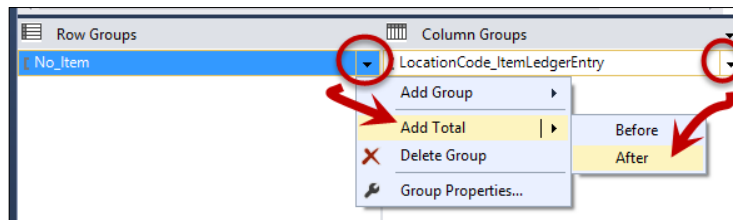
```
=Fields!Description_Item.Value & " (" &  
    Fields!No_Item.Value & ") "
```

This will display the item description and, between brackets, the item number.

- Next, change the sorting of the group by item number to sort on the description:

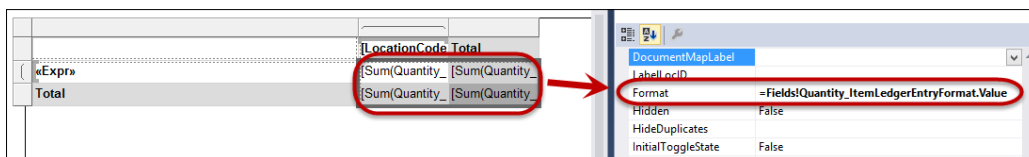


- Next, add totals for the two groups:



This will add an extra column and row to the matrix.


- Select the **Quantity** and then select the **Sum** as an aggregate. Then, select the four textboxes and, in the properties, apply the formatting for the quantity field:



- Next, you can use different background colors for the textboxes in the total rows and resize the description column to resemble the layout in the preceding screenshot. If you save and run the report, you have now created an item dashboard.

Notice how easy it is to use the matrix control to create a dashboard. At runtime, the number of columns depends on the number of locations. The matrix has a dynamic number of columns. There is no detail level, because the ledger entries are grouped on row and on column level.

Colors and background colors

 When using colors in a report, pay attention to how the report is printed. Not all printers are color printers, so you need to make sure that your visualization has an effect. That's why I have used gray colors in this example.

Colors are sometimes also used by developers as a trick to see at runtime, where which textbox is displayed and to test report rendering in different formats. If you do this, remember to remove the colors at the end of the development phase of your report.

Summary

The Tablix is a very versatile control that contains templates to create a List, Table, and Matrix layout. It usually contains groups, which can be created with a parent-child relationship or an adjacent one. We have seen how we can apply sorting and filtering in a report, both in the layout and at the point of the generation of the dataset.

Textboxes have a lot of properties and contain placeholders, so we can format information in many ways, which we also covered in this chapter.

Expressions are frequently used to make report execution dynamic, and in the next chapter, we will dive into the expression editor and see examples of how you can create expressions and the syntax that you need to use to do that.

3

Expressions

In this chapter, we will explore the expression designer. I will explain and demonstrate how you can use expressions to manipulate properties to create dynamic behavior from reports at runtime. You can create simple and complex expressions, and functions that you can reuse when you design reports. Last but not least, I will demonstrate this with some typical real life scenarios and examples.

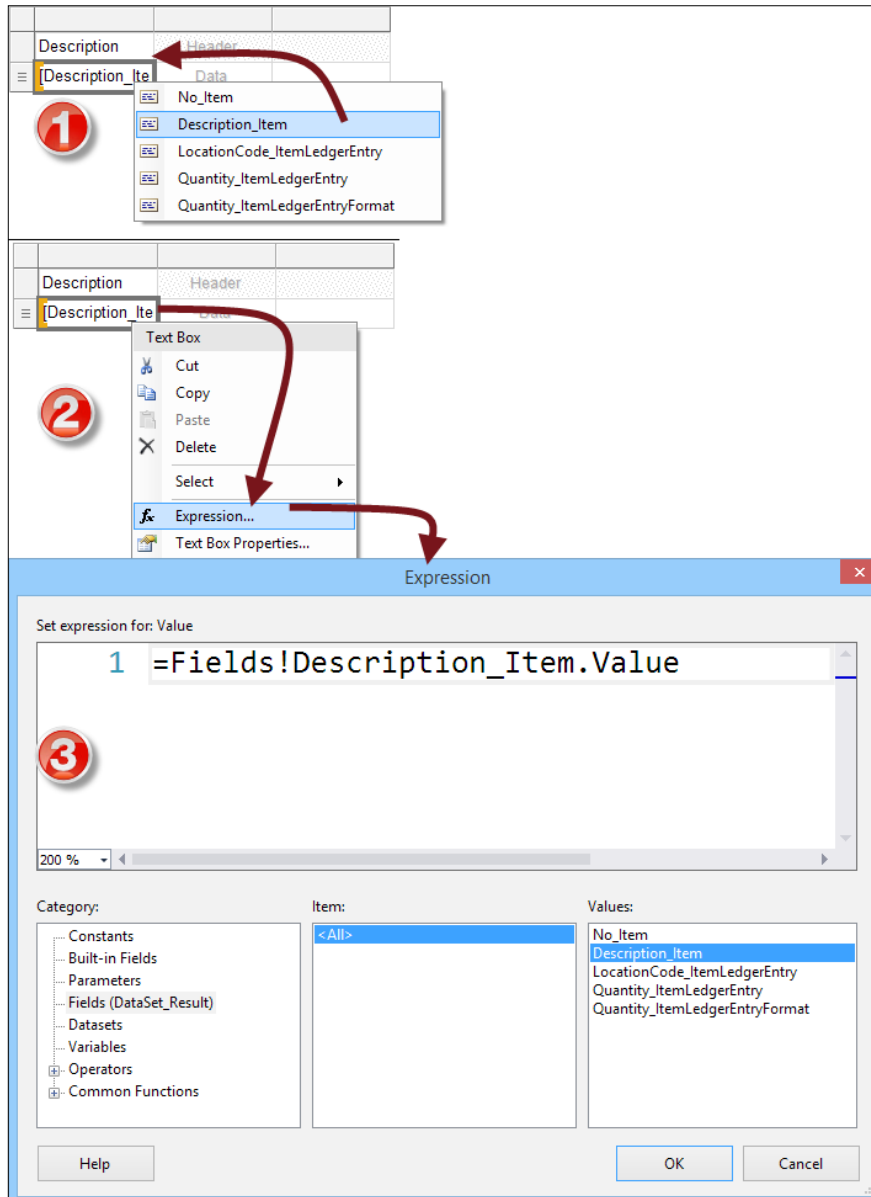
Using expressions for properties

Without realizing it, you might have already used expressions when designing reports. This is because, most of the time, expressions are implicitly created when you design the layout of your report. For example, when you drag and drop a field on a textbox, the system creates the following expression for the `Value` property of the textbox:

```
=Fields!ColumnName.Value
```

The expression gives the instruction to fetch the `Value` property from the dataset `Field` with the name `ColumnName`. `Fields` is actually a collection that holds all of the fields in the dataset. There are other collections that you can use.

For example, if you right-click on a textbox and select **Expression...**, you will end up in the expression designer for the `Value` property of that textbox, as shown in the following screenshot:



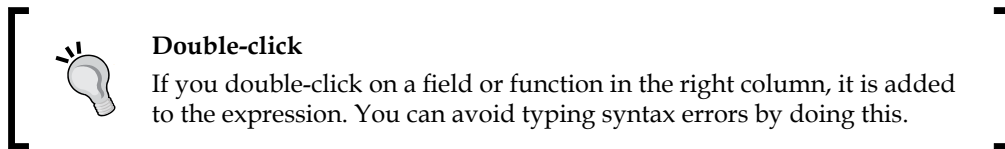
In the preceding screenshot, a field is dragged onto a textbox. In the textbox in the Tablix, it says, [Description_Item]. When you drag or select a field from the dataset into a textbox, the system uses the following notation:

```
[ColumnName]
```

If you open the expression you can see that it actually stands for:

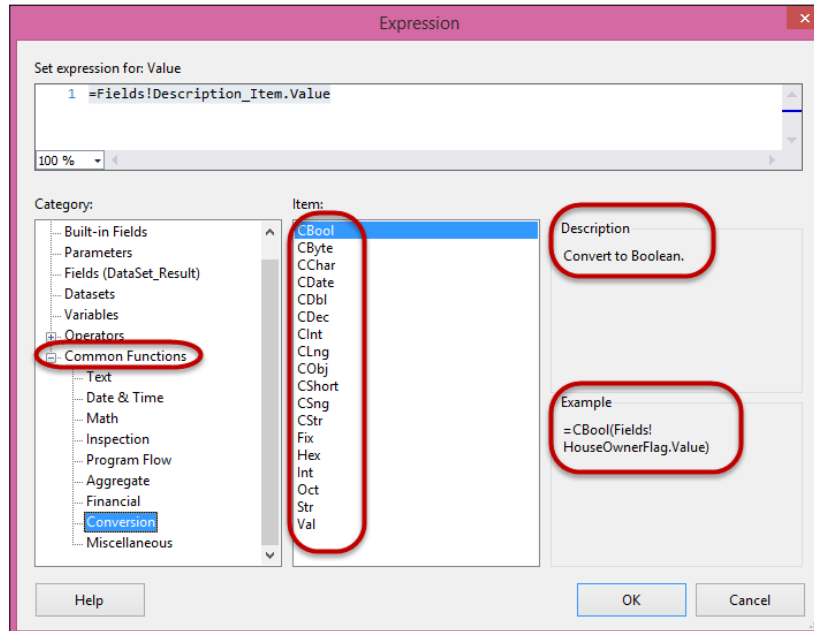
```
=Fields!Description_Item.Value
```


You can also see that, in the expression designer window, we are in the expression for the `Value` property. It's mentioned at the top in **Set expression for: Value** and, if you look in the **Category** list, you will see **Fields (Dataset_Result)**, which contains all of the fields from our dataset.



An expression will return a result and it is the result of the expression that is used, at runtime, for the value of the property. That is why you always need to start your expression with an equal sign (=). The data type of the result of the expression should match the data type of the property, otherwise you will get a runtime error. If the data types are not the same, you can use a conversion function.

For example, if you expand **Common Functions**, at the bottom you will see a list of available conversion functions:



 We have already used a conversion function in a previous chapter when we applied a filter to the Tablix to link it to specific records in the dataset. In that example, we used the `CStr()` function to compare a field from the dataset to the empty string.

Conversion functions are used when we need to convert the result of one expression to another data type so that it can be used in a property or sent to another function that expects a specific data type. Sometimes you don't know what the data type of an expression is. To find it out, you can use the following expression:

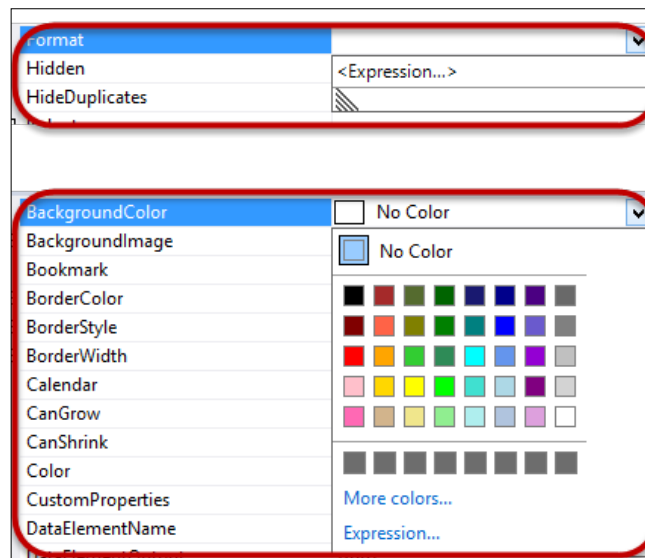
```
=Fields!MyField.Value.GetType().ToString()
```

This will result in the dot net data type being returned, for example, `System.String`, or something similar.

Expressions can be used almost everywhere in the report layout. If you open a property box and you see the **Fx** button, as in the following image, it means that you can use an expression for this property:



When you select a property in the properties window and then click on it, you will see **Expression** open the expression designer for this property in the drop-down list:




If there is no **Fx** button or no **Expression** shortcut, it means that this property cannot be set by an expression.

There are many types of functions, operators and collections and, in the following lessons, I will explain how you can apply them. First, let me introduce and explain the syntax and language you need to apply and learn, and how to write your own expressions.

The expression language

The expression language and syntax is Visual Basic. If you have some experience with writing macros in Excel, then you will pick it up very quickly because it is very similar.

You can consider an expression as a formula for data that is evaluated at runtime by the report viewer. The expression editor verifies the syntax and will underline parts it believes that are incorrect. You will only notice if there's an error when an expression is executed at runtime.

 If you use custom functions that you create yourself, as I will demonstrate later in this chapter, the syntax verifier will not recognize your function and will underline it in red, even though it might be completely correct.

The following are some general rules about the expression language:

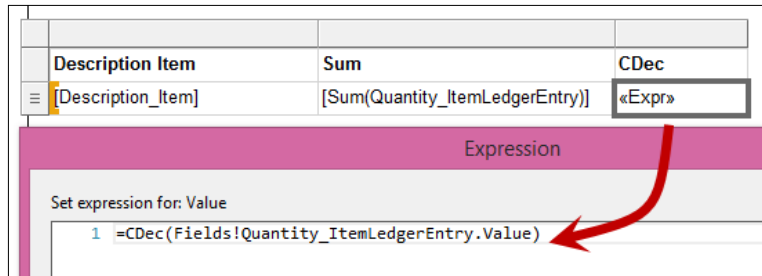
- You can write expressions that use functions from the Visual Basic run-time library, and from the `System.Convert` and `System.Math` namespaces.
- You can add references to functions from other assemblies or custom code.
- You can use classes from the Microsoft .NET Framework, including `System.Text.RegularExpressions`.
- To include a reference to other less commonly used CLR namespaces, you must use a fully qualified reference, for example, `System.Text.StringBuilder`. IntelliSense is not supported in the code pane of the **Expression** dialog box for these less commonly used functions.

 For more information about Visual Basic functions supported in expressions, see "Visual Basic Run-Time Library" at www.msdn.microsoft.com.

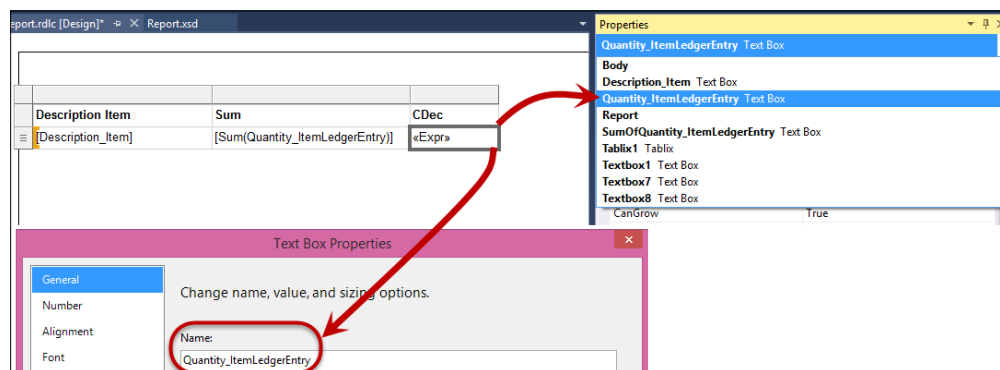
Simple and complex expressions

A simple expression is a reference or a pointer to an item in one of the collections, for example, a field from the dataset. A simple expression is visualized in the textbox using (`[]`) brackets.

A complex expression is a combination of simple expressions and operators or functions. As soon as you use a complex expression for the value property of a textbox, the textbox will contain <<Expr>> when you look at it in the report layout, as you can see in the following example:



For this reason, I recommend that you give every textbox a proper name, so that you can see in its properties what it actually contains, as in the following example:



This is especially important when you are developing a document report. In this type of report there are many fields, and it can get confusing to edit and customize if you are asked to change the context of a textbox. By applying proper names, you can use the property drop-down or document outline to quickly locate the textbox that you are looking for.

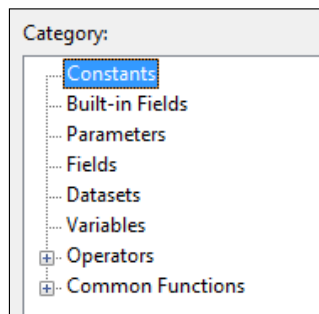
Symbols used in expression placeholders

The following table illustrates the symbols that are used to identify every type of expression in a textbox:

Type	Symbol	Expression
Fields (from the Dataset)	[Quantity]	=Fields!Quantity.Value
	[SUM(Quantity)]	=Sum(Fields!Quantity.Value)
	[FIRST(Quantity)]	=First(Fields!Quantity.Value)
Parameters	[@Parameter]	=Parameters!Parameter.Value
	[@Parameter.Label]	=Parameters!Parameter.Label
Built-in fields	[&PageNumber]	=Globals!PageNumber
Text	\[Quantity\]	[Quantity]

Collections

When you drag a field from the dataset onto a textbox, its expression references the **Fields** collection. There are also other collections you can reference. Some, but not all, of the collections that you can reference in the expression designer are shown here:



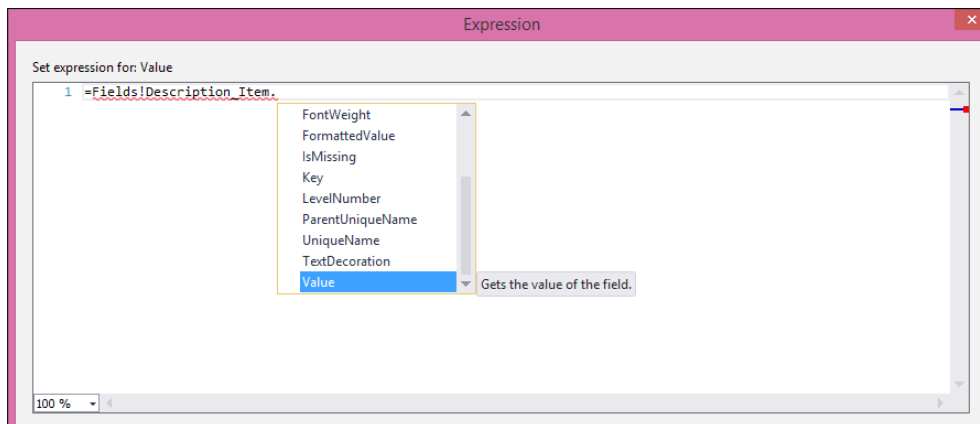
The following is a list of the collections you can reference:

Collection	Description
Globals	Contains global variables. Not all of them can be used in RDLC.
User	Contains information about the user who runs the report. (In some versions of Dynamics NAV, it is the user under which the Service Tier runs and not the actual user who runs the report).

Collection	Description
Fields	Contains all of the Fields collection from the DataSet: DataSet_Result
Parameters	Contains the Parameters , created via Labels and IncludeCaption .
ReportItems	Contains a list of all textboxes in the report. Textboxes are referenced via their Name property.
Datasets	Contains the DataSet: DataSet_Result . Because there's only one dataset in Dynamics NAV reports, there's no point in using this collection.
Variables	Contains Variables you create in the Report properties. (These are not the C/AL globals or locals that you create in the Report Dataset Designer.)

A collection is an object. This means that, in order to access its members, you need to apply the correct syntax. For example, when you use the expression, `Fields!FieldName` in the `Value` property of a textbox, there will be a runtime error, because the object cannot be converted to text. Instead, you should access the value as follows: `Fields!FieldName.Value`.

When you type in the expression editor, there is an inline auto completion function that shows the members of an object. The following screenshot demonstrates this:



In the preceding screenshot, you can see that **Value** returns the actual value of the field.



Not all collections appear in the expression dialog box. Some of them are only available at runtime like, for example, the **ReportItems** collection, which contains the list of textboxes in your report, but that does not mean you can't reference them. You can simply type in these collections manually in the expression designer.

Understanding the scope of an expression

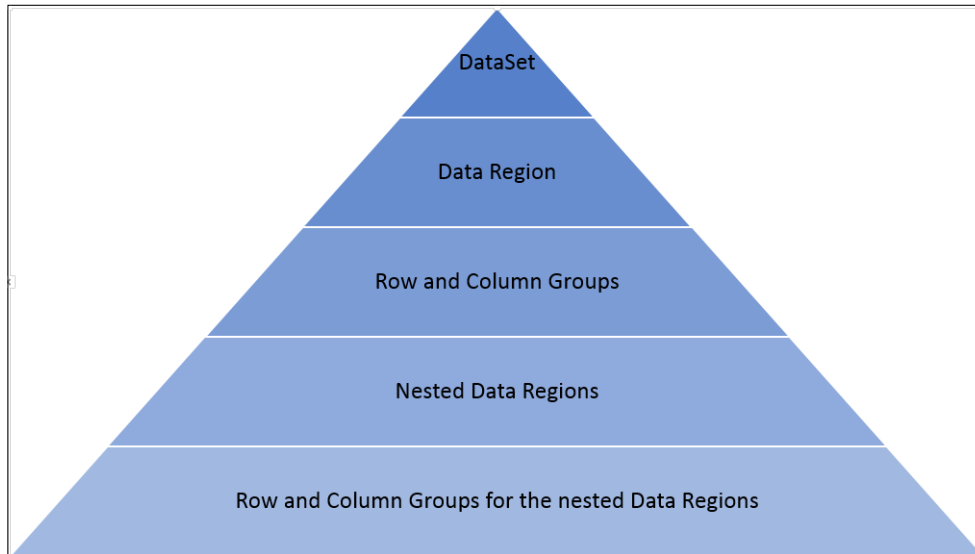
It's important to understand the scope of an expression, because it will determine its result and it might not always be what you expect. The scope is determined by two factors. You can explicitly define a scope in your expression, this is named scope, or the scope is determined implicitly by the system, depending on where the expression is executed or, in other words, where your textbox is located in the report, as in, this is the default scope.



For example the `=Sum(Fields!Quantity_ItemLedgerEntry.Value)` expression does not have a value for its scope parameter so it uses the implicit scope of the object it is in.

The `=Sum(Fields!Quantity_ItemLedgerEntry.Value, "DataSet_Result")` expression has a scope parameter, which is `DataSet_Result`, so it calculates the sum of the `Quantity_ItemLedgerEntry` field for all of the rows in the dataset.

Scope is important when you use aggregations such as, for example, when calculating a sum, an average, and so on. Scope is determined by the container that holds your textbox. A representation of how to understand scope is shown in the following figure:



The default scope is determined by the container, or containers, that hold your textbox.

If you drop a textbox in the report body and use the `Sum()` function, then the implicit scope is the highest, the `DataSet`. If you drop a textbox in a header or footer row of a `Tablix`, then the `Sum()` function gets the implicit scope of the group you are in at runtime.

In the following example, there is a Tablix that contains one group:


Location Code	Description Item	Sum	
[LocationCode]	[Description_Item]	[Quantity_ItemLedgerEntry]	1
		[Sum(Quantity_ItemLedgerEntry)]	2
Total		[Sum(Quantity_ItemLedgerEntry)]	3

Row Groups	Column Groups
[LocationCode_ItemLedgerEntry	
(Details)	

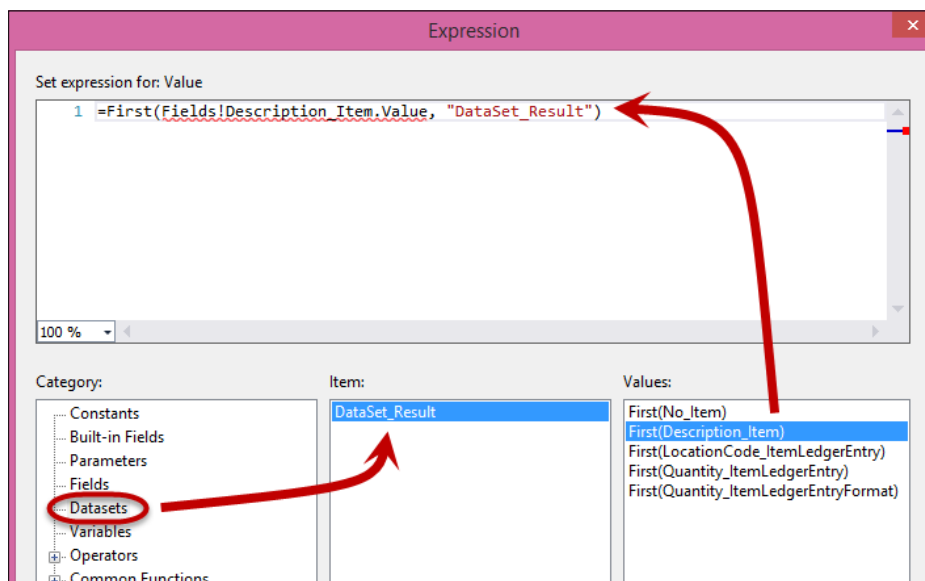
Items are grouped by location code. There is one row for the details (1), one footer row for the location (2) and a footer row for the Tablix (3). On level 2 and level 3, I have used the sum of the quantity. The expressions look exactly the same. But when I run the report, the result looks as follows:

Location Code	Description Item	Sum	
Item Ledger Entry			
BLAUW			
	ATHENE Mobiel onderstel	70	1
		70	2
GROEN			
	ATHENE Mobiel onderstel	108	1
	ATHENE Mobiel onderstel	20	1
	ATHENE Mobiel onderstel	-40	1
		88	2
OWN LOG.			
	ATHENE Mobiel onderstel	40	1
		40	2
ROOD			
	ATHENE Mobiel onderstel	63	1
	ATHENE Mobiel onderstel	-1	1
	ATHENE Mobiel onderstel	-6	1
		56	2
Total		254	3

Because level **3** is in the report footer row, its scope is different to the group footer row and therefore the result of **Sum** is also different. This is an example of default scope depending on the container that contains the textbox.

[ An example of this report is available in the object: Packt - CH03-1]

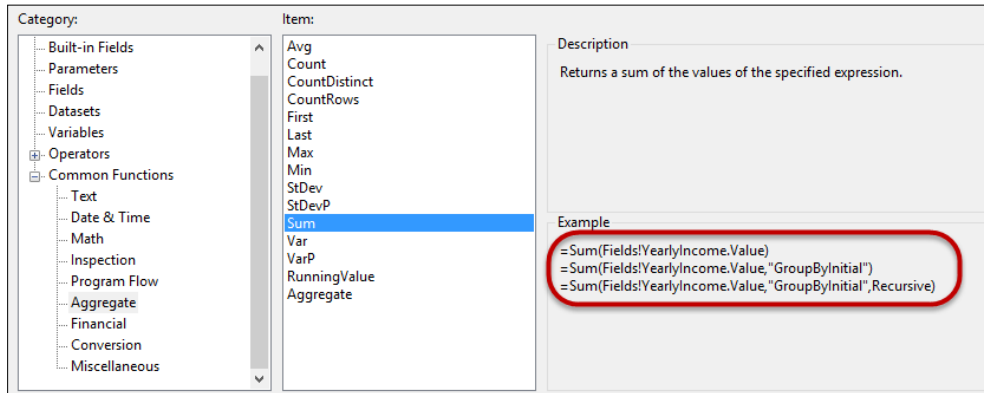
Named scope is when you explicitly set the scope of an expression. An example is when you create a textbox and, in the expression for its value, select a field from the **Datasets** collection, as shown in the following screenshot:



The string `DataSet_Result` is used as the scope for the `First` function, which implies fetching the value from the first row of the dataset with the name `DataSet_Result`. Other aggregate functions such as, for example, `Last()`, `Sum()`, can also accept a scope parameter.

When you select a field in the expression designer from the **Fields** collection, no scope is added, but when you select a field from the **Datasets** collection, a function is used, **First** or **Sum**, and a scope is automatically set to the dataset.

If you have a look in the expression editor when you select a function, at the right bottom, you can see how to use it with different parameters:

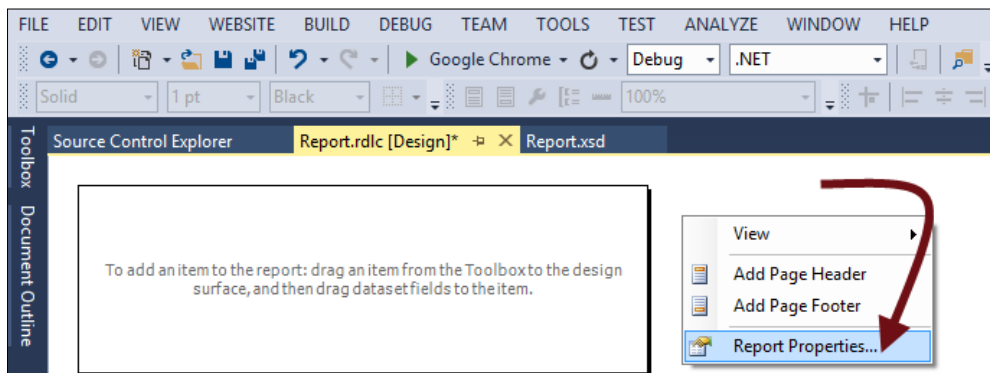


If you do not use a scope, then the default or implicit scope is used.

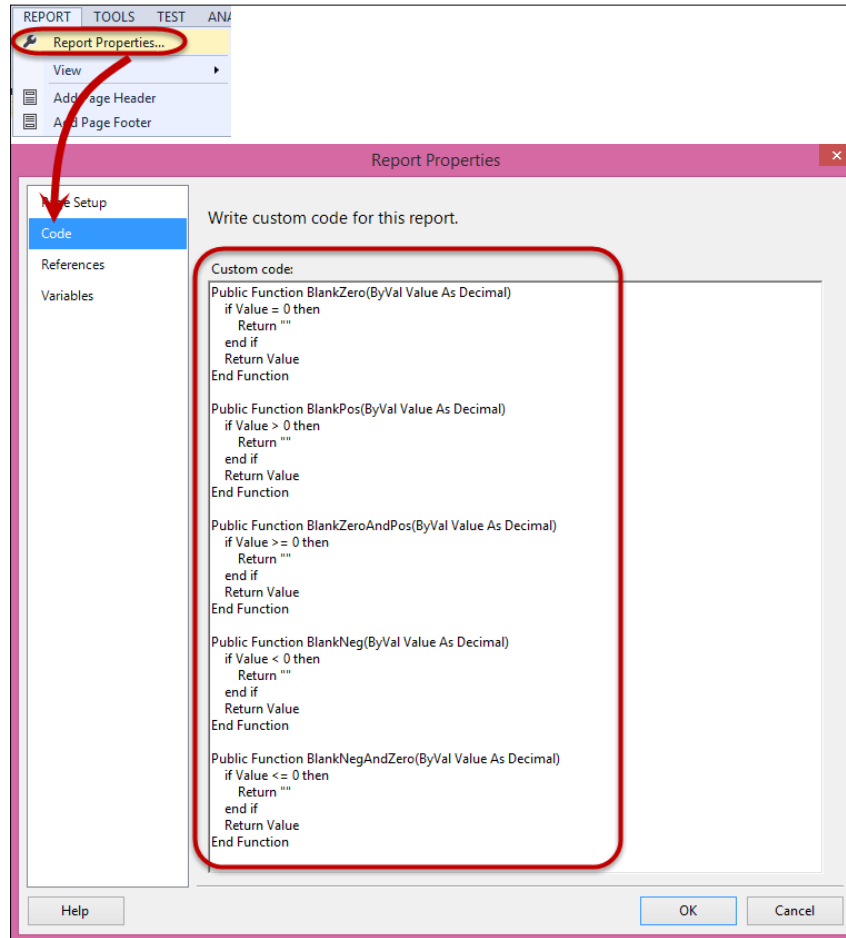
Creating custom functions

Custom functions are functions that you create yourself. You can do this in the report properties. In the menu, open **Report, Properties**.

Sometimes the **Report** menu is not displayed, because it depends on what you have selected in the report. If so, simply click on the report body and the report menu will appear again. Or, you can immediately right-click on the report body and select **Report Properties** from the dropdown, as in the following screenshot:



You will notice a **Code** tab, in which there are already some custom functions defined:



The following functions feature in this report:

- BlankZero
- BlankPos
- BlankZeroAndPos
- BlankNeg
- BlankNegAndZero

To call a custom function from within an expression, you need to use this syntax:

```
=Code.NameOfFunction(Parameters,...)
```



Note that we use a dot (.) instead of an exclamation mark (!). The exclamation mark is referred to as the bang operator. The difference between the two is simply that the dot is early-bound and the bang is late-bound. This has to do with VBA language syntax. Unfortunately, there's almost no documentation from Microsoft on this subject, but you will find some good examples in this blog: <http://bytecomb.com/the-bang-exclamation-operator-in-vba>.

The preceding functions are used to format numbers. For example, if the quantity for an item is zero, you can show an empty value instead of a zero via the following expression:

```
=Code.BlankZero(Fields!Quantity_ItemLedgerEntry.Value)
```



Note that the syntax editor underlines the custom function. This is because it does not recognize the function as a known Visual Basic function. It does not mean that the function does not exist or that there's a syntax error.

Apart from functions, you can also create constants and variables using standard Visual Basic syntax. An example is in **Report 206 Sales Invoice**. Here, in the code tab, you will find the following variables:

- Shared Data1 as Object
- Shared Data2 as Object
- Shared Data3 as Object
- Shared Data4 as Object
- Shared NoOfCopies as integer

In document reports, shared variables are used to store information that needs to be available on all pages of a report in the page header or footer. Since you cannot use data regions in a report header or footer, you add the fields from the dataset onto the body of the report, usually in a list container, and then you store the values in the shared variables using a Set and Get function.

An example of this function is shown in the following code:

```
Public Function SetNoOfCopies(Value as integer)
    NoOfCopies = Value
End Function

Public Function GetNoOfCopies() As integer
    Return NoOfCopies
End Function
```

When you type `=Code.SetNoOfCopies(3)` in an expression, the value 3 is stored in the shared variable `NoOfCopies`. When you type `=Code.GetNoOfCopies`, the expression returns the value 3, stored in the shared variable `NoOfCopies`.



An in depth explanation of these functions and the sales invoice report is discussed in *Chapter 5, Document Reports*.

Most variants of `Get` and `Set` functions use multiple parameters to work on multiple variables. Imagine that you have four shared variables and the following function is used to store information in any of the four variables, using the `Group` parameter:

```
Public Function SetData(NewData as Object,Group as integer)
    If Group = 1 and NewData > "" Then
        Data1 = NewData
    End If

    If Group = 2 and NewData > "" Then
        Data2 = NewData
    End If

    If Group = 3 and NewData > "" Then
        Data3 = NewData
    End If

    If Group = 4 and NewData > "" Then
        Data4 = NewData
    End If

    Return True
End Function
```



Although there are better ways to use collections, arrays, or dictionary objects, most document reports in Dynamics NAV use these `GetData` and `SetData` functions. In *Chapter 5, Document Reports*, I will explain in detail how and why this is done.

Report 111 Customer Top 10 has the following function in its code tab:

```
Shared Pct as Decimal
Public Function CalcPct(Amount1 as Decimal, Amount2 as Decimal) as
Decimal
    if Amount2 <> 0 then
        Pct = Amount1 / Amount2 * 100
    else
        Pct = 0
    end if
    REM Rounding precision = 0.1
    Return ROUND(10*Pct)/10
End Function
```

As you can see, this function calculates a percentage, based upon 2 amounts entered as parameters.

There are many other examples of functions available in standard Dynamics NAV reports. RDL and RDLC technology has been around for many years; if you search online, you will find many other examples of functions.



Although you can do almost anything you want with custom functions, it is better to avoid functions that contain business logic. Business logic does not belong in a report layout. Most functions you create and use in a report layout are formatting and conditional formatting functions.

Reusable custom functions

It is important to remember that custom functions are created in the report layout and are embedded in the report layout. This means that you cannot call them or reuse them from another report. A possible solution is to create a template report that contains all the functions that you have created. When you create a new report, you can then base it on the template, or copy/paste the function from the template into your new report. If you create a new custom function in your new report, remember to add it to the template report.



Another solution is to store functions in a separate assembly that you then reference in the code tab of your report. You can do this in the report properties references tab. Personally, I don't recommend this, because you then need to deploy this .dll to all client machines in order for the report to work for all users. Secondly, if you use external assemblies, then you will also need to enable this in the report properties in the report dataset designer via the property: `EnableExternalAssemblies`. Since reports are executed in the Report Viewer, and the report viewer runs on the client, all expressions in reports are executed on the client and you cannot therefore reference any assemblies that are not available on the client.

Typical expression examples

Now that we know how to create expressions and custom functions, let's have a look at some typical examples of how expressions can help us achieve our goals when creating reports.

Working with dates

The following function can be used to display the current date:

```
=Today()
```

The following function can be used to calculate new dates based upon an existing date:

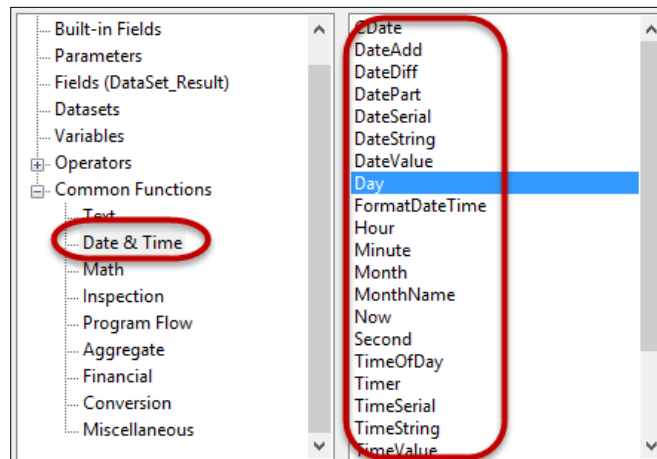
```
=DateAdd(DateInterval.Month, 6, Parameters!ExpirationDate.Value)
```

In this example, starting from the expiration date, you add 6 months to it to calculate a new date. The number 6 can be any number and, when it is negative, you can subtract periods from a date. `DateInterval` contains `Year`, `Month`, `Day`, `Hour`, `Minute`, and so on.

The following function retrieves the year from a date:

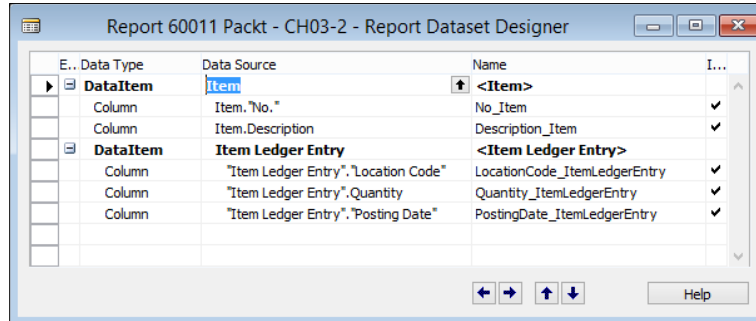
```
Year(Fields!PostingDate.Value)
```

In this example, I have used the posting date, but it could be any date. As in the `Year()` function, you can also use `Month()`, `Day()`, and other functions. To see which functions are available for dates, you can have a look in the expression editor, as illustrated in the following screenshot:



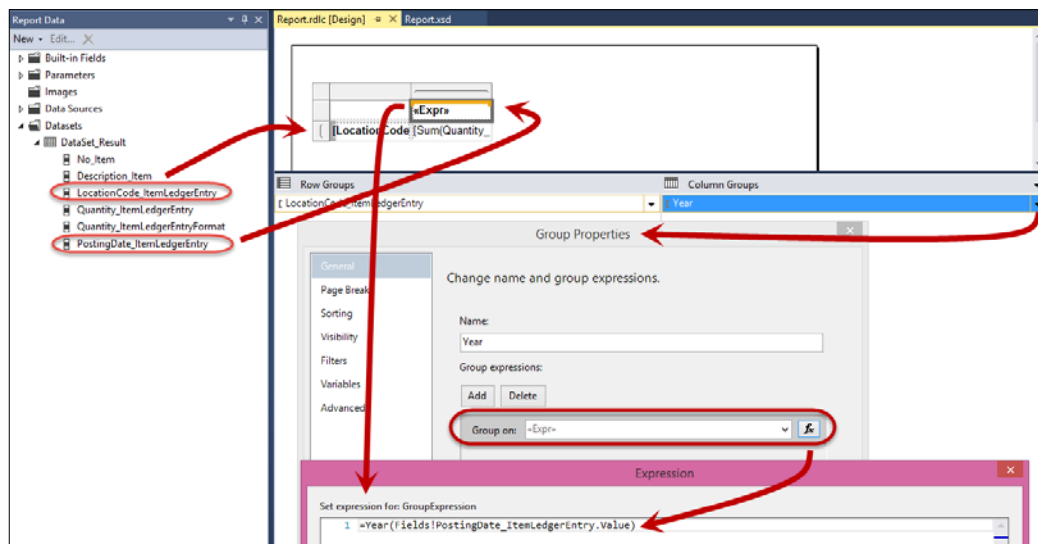
I find the `Year()`, `Quarter()` and `Month()` functions useful when I need to create **Year To Date (YTD)** reports. In these types of reports, users want to see aggregated information grouped by year, month and date. Instead of calculating different dates in C/AL and adding multiple columns in the dataset, you can just have one date field in your dataset and create groups with expressions. In the following example, you can see how to do this.

Start with a new report and create the following dataset:



Then, follow these steps:


1. Add a **Matrix** to the report.
2. Select the **Location Code** field as the row group.
3. Select the **PostingDate** field as the column group.
4. Select the **Quantity** field in the **details** and then select the **Sum** aggregate:



- Next, open the properties of the group expression for the **PostingDate** group and change the expression so as to use the `Year` function, as in the preceding example. Use the same expression in the textbox for the column header.
- Then, when you run the report, you will see the quantity, per location, per year:

	2015	2016
BLAUW	38795	1327
GEEL	533	273
GROEN	604	5290
OUT. LOG.		29
OWN LOG.		1365
ROOD	468	60,52480
WIT		848

Now you can add a child group below the year, where you can group by month, amongst others.


[ An example of this report is available in the object: Packt - CH03-2]

Working with strings

To concatenate (or glue) strings together, you can use the `&` (ampersand) operator, as in the following example, where we glue the `LastName` after the `FirstName` value:

```
=Fields!FirstName.Value & vbCrLf & Fields!LastName.Value
```

The `vbCrLf` is a Visual Basic constant used to create a new line.

[ Although you can also use the plus (+) operator to concatenate strings, I recommend the `&` operator because, when both terms are numerical, the plus operator adds them instead of concatenating them. This can happen unexpectedly when the terms are expressions instead of fields.]

You can use the `Format ()` function to format strings, as shown in the following example:

```
=Format (Fields!PostingDate.Value, "D")
```

In this example, we use the `D` format code to format the `PostingDate` field. The `D` format code specifies a specific data format.



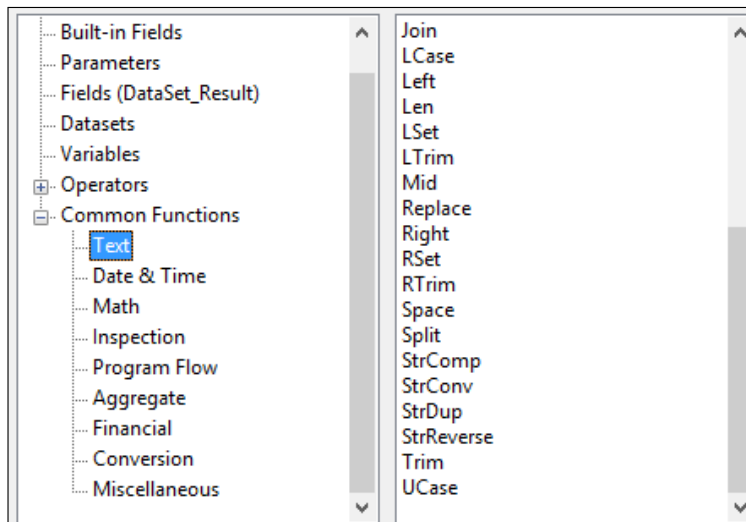
You can find other examples of date format codes here: [https://msdn.microsoft.com/en-US/library/73ctwf33\(v=vs.80\).aspx](https://msdn.microsoft.com/en-US/library/73ctwf33(v=vs.80).aspx)
 You find more information about the `Format ()` function and links to all the format codes you can use, depending on the data type, here: [https://msdn.microsoft.com/en-us/library/59bz1f0h\(v=vs.80\).aspx](https://msdn.microsoft.com/en-us/library/59bz1f0h(v=vs.80).aspx)


In this way, you can do it in the expression for the `Value` expression, instead of having to use the `Format` property.



Pay attention when using the `Format ()` function in the `Value` expression and when setting the `Format` property, as the `Format` property is applied to the result of the `Value` expression.

There are many string (or text) functions that you can use. You will find them in the expression editor, as shown in the following screenshot:



 **Performance warning**
Try not to use string functions for group expressions. This is because string functions can be time and resource (CPU) consuming and will therefore slow down report performance. If this is the case, then I recommend C/AL code instead.

Decision functions

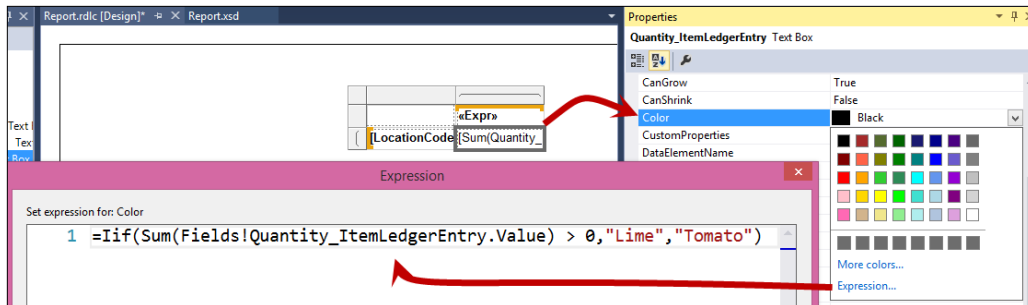
Sometimes, when you write an expression, you want the output to depend on a condition. You can make use of the following decision functions to do that:

Iif()
Switch()

The syntax of the Iif (or Inline If) is as follows:

=Iif (condition, ResultIfTrue, ResultIfFalse)

You can use it to test for a specific condition and, if it's true or not, get a different result. Let me give you an example in the following screenshot:



I have used an expression that sets the color to `Lime` or `Tomato`, in the `Color` property for the textbox that holds the inventory, depending on whether the inventory is positive or not. The result is shown in the following screenshot:

	2015	2016
BLAUW	38795	1327
GEEL	533	273
GROEN	-5052	4277
OUT. LOG.		29
OWN LOG.		1365
ROOD	468	60,52480
WIT		848

You can also nest `Iif` conditions, as in the following example:

```
=Iif(Fields!PctComplete.Value >= .8, "Green",
    Iif(Fields!PctComplete.Value >= .5, "Amber", "Red"))
```

Nesting `Iif()` conditions makes your code more difficult to read and maintain. That's why you can replace it with a `Switch()` condition, as follows:

```
=Switch(Fields!PctComplete.Value >= .8, "Green",
    Fields!PctComplete.Value >= .5, "Amber",
    Fields!PctComplete.Value < .5, "Red")
```



A `Switch()` condition in Visual Basic is similar to a `CASE` statement in C/AL.

You can, of course, also combine a decision statement with a date function. The following example could be used in the `Color` (or `BackColor`) property to give the field a different color, depending on whether the expiration date is within seven days:

```
=IIF(DateDiff("d",Fields!ExpirationDate.Value,
    Now())>7,"Red","Blue")
```

The `RowNumber (scope)` function returns the number of the row, depending on the scope you provide. If you use the word `Nothing` as the scope, then it returns the row number in the Tablix, if you have a group in the Tablix, then you can use its name as the scope. If you combine the `RowNumber` function with the `Iif` function in the `BackgroundColor` property of a row in a Tablix as follows:

```
=Iif(RowNumber(Nothing) Mod 2, "PaleGreen", "White")
```

Then, when you run the report, it will look like this:

C-100	Bekabeling voor LS-100	33
FF-100	Frequentiefilter voor LS-100	42
HS-100	Behuizing LS-100,Eiken 120 Its	56
LS-100	Speaker 100W OakwoodDeluxe	32
LS-10PC	Speakers, Wit voor pc	38
LS-120	Speakers, Zwart, 120W	6

In the preceding example, every even or odd row has a different color. This is called a **green-bar** effect.



If you try this in a matrix, then you will not get the expected result. This is because, in a matrix, the `RowNumber ()` function uses the row and column groups as scope. See the *Example – the green-bar matrix* section in this chapter for an example of how to solve this problem.

A more complex variation using a `Switch ()` expression would be:

```
=Switch(  
    Me.Value > 10000, "DarkOliveGreen",  
    Me.Value > 1000, "OliveDrab",  
    Me.Value > 100, "ForestGreen",  
    Me.Value > 10, "LimeGreen",  
    Me.Value > 0, "GreenYellow",  
    Me.Value = 0, "Yellow",  
    Me.Value < 0, "Goldenrod",  
    Me.Value < -10, "DarkGoldenrod",  
    Me.Value < -100, "Orange",  
    Me.Value < -1000, "DarkOrange",  
    Me.Value < -10000, "Red")
```

As you can see, the `Color` properties of textboxes, rows, and columns are ideal candidates for these kinds of expressions.

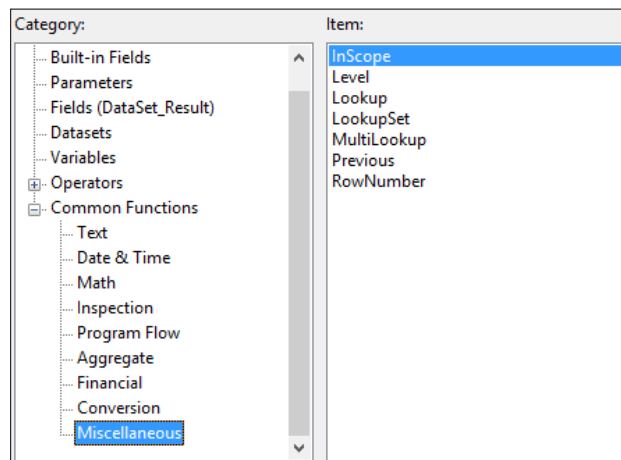
When you are planning to use more complex expressions, or even simple ones, for conditional formatting, I would advise against creating and storing the expressions in the individual properties, but advocate creating custom functions. In this way, you can still call the custom function from wherever you want to inside the report, and you only have to maintain the function in one location.



When to use conditional formatting

Although conditional formatting can increase the readability of a report, it is important not to overdo it. As in the early days of the World Wide Web, when we developed our first web pages, it was very tempting to include a lot of colors, pictures, moving and animated texts, and so on. But these do not necessarily improve the usefulness of a report, rather the contrary. So, I would like to stress that conditional formatting should be used with a purpose, and not just because it's possible.

If you have a look in the miscellaneous functions, you will see more functions that you can use:



Here's another example of an expression for the `BackColor` property:

```
=Iif(RunningValue(Fields!Item__Inventory_Posting_Group_.Value,  
CountDistinct, Nothing) MOD 2 = 1,"LimeGreen", "White")
```

This expression is most useful when you have a grouping in your report, for example the `InventoryPostingGroup` field. Placing the expression in the `BackColor` property creates an effect, so that when a new group starts it will get a different background color, instead of with every line.

Essentially, this expression can be translated to read "If the distinct count of unique `InventoryPostingGroup` field values is 1 (and is therefore an odd number), return the color `LimeGreen`; otherwise, return the color `White`". Remember to set this expression for the detail row and also the group header and/or footer rows.

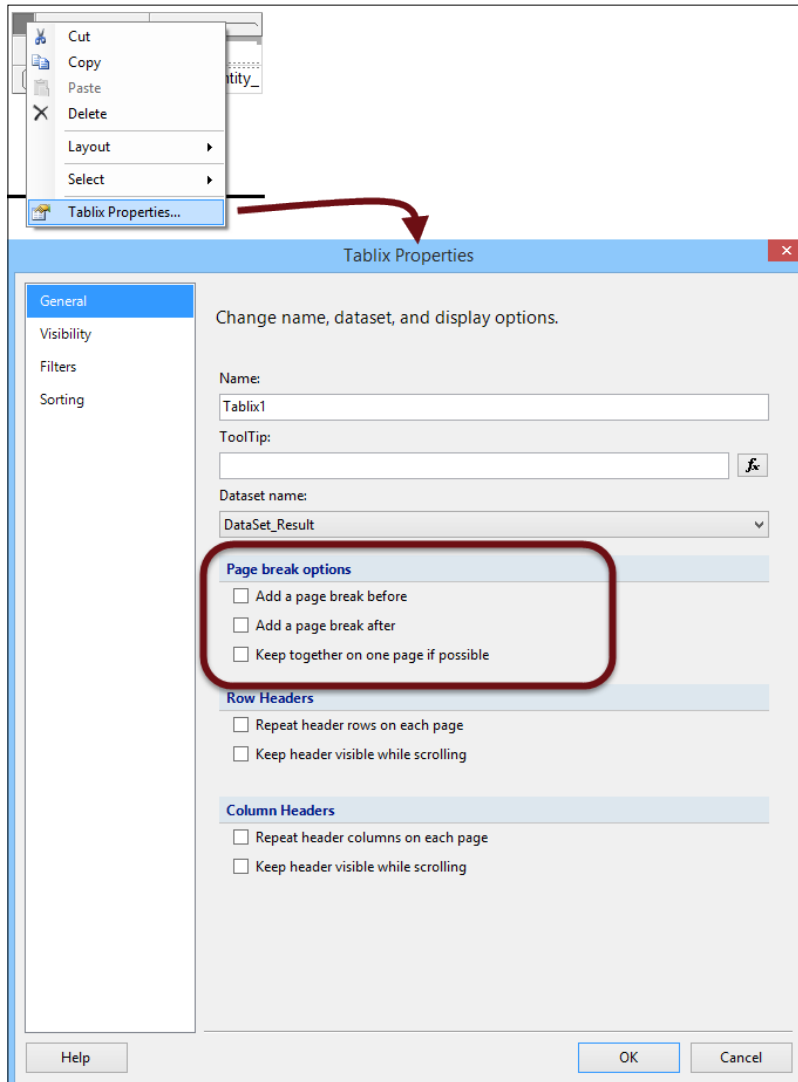
Conditional formatting can also be used to simulate KPIs. When a value is above or below a threshold, for example, a green, orange, or red bullet could be shown, as is the case in many business intelligence reports using traffic light indicators. This could easily be achieved using embedded images and an expression. All you need to do is embed three images in a report and display them at the correct moment.

For example, when creating a report about items and inventory, it could be interesting to visualize when a specific item has to be reordered or is close to its maximum inventory level.

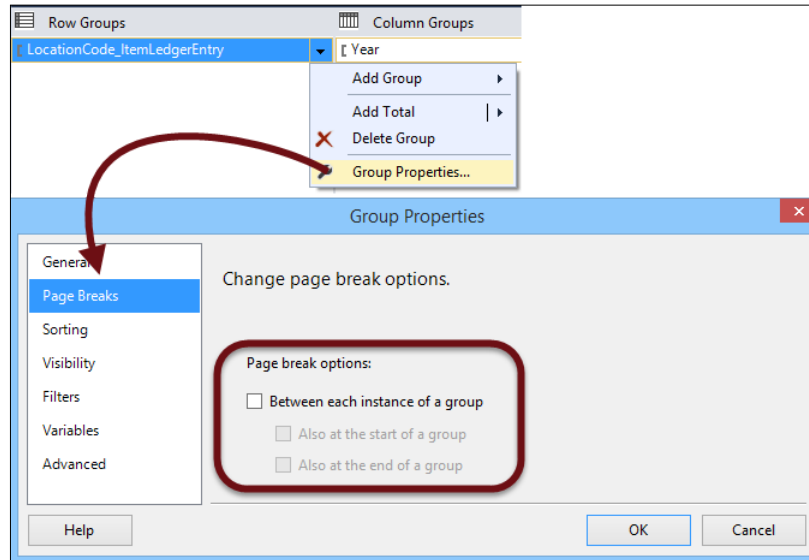
Generating page breaks in code

You can not only create conditional formatting effects with expressions, but you can also determine when a page break should occur.

Page breaks can be generated using Tablix properties, as you can see in the following screenshot:



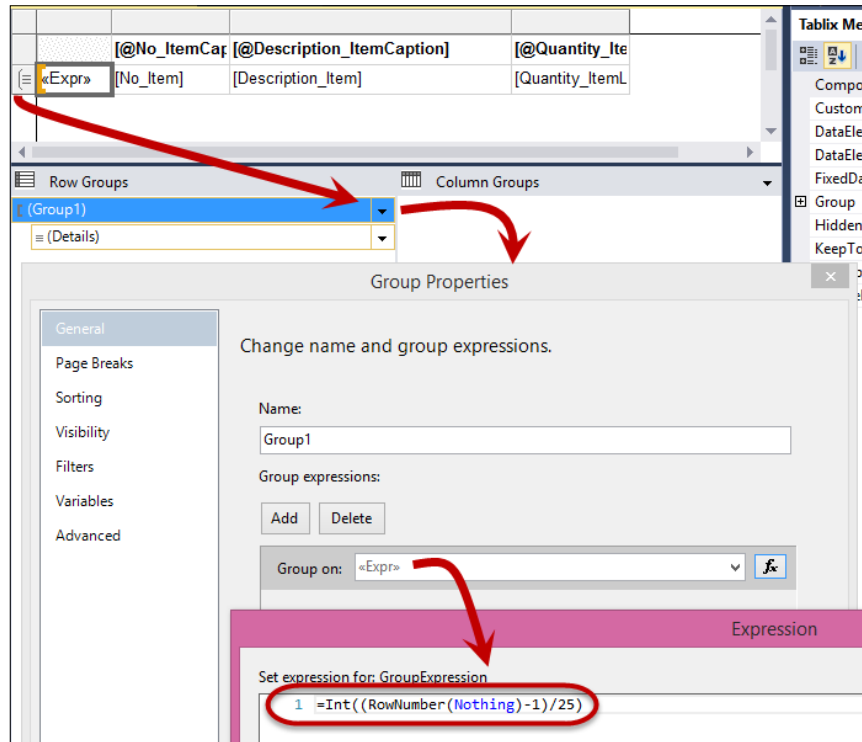
A page break can also be generated using group properties, as you can see in the following screenshot:



You can dynamically determine when a page break needs to happen by using following expressions:

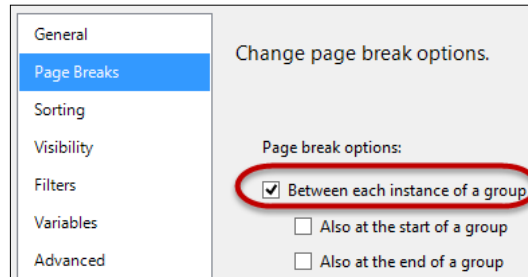
```
=Int ((RowNumber (Nothing) - 1) / 25)
```

The expression will return 1 if the row number can be divided as a whole by 25, otherwise it will return 0. Now you can use this function as the group expression when you add a group to your Tablix, as shown in the following example:



Expressions

Then, in the group properties, in the **Page Breaks** field, you can enable a page break between each instance of the group:



When you run the report, a page break is generated after every 25 lines, as shown in the following image:

No.	Description	Quantity
1	1896-S ATHENE Tafel	160
2	1896-S ATHENE Tafel	52
3	1896-S ATHENE Tafel	49
4	1896-S ATHENE Tafel	-1
5	1896-S ATHENE Tafel	-1
6	1896-S ATHENE Tafel	1
7	1896-S ATHENE Tafel	1
8	1896-S ATHENE Tafel	-6
9	1896-S ATHENE Tafel	-1
10	1896-S ATHENE Tafel	-25
11	1896-S ATHENE Tafel	25
12	1900-S PARUS Bezoekersstoel, zwart	52
13	1900-S PARUS Bezoekersstoel, zwart	46
14	1900-S PARUS Bezoekersstoel, zwart	47
15	1900-S PARUS Bezoekersstoel, zwart	-6
16	1900-S PARUS Bezoekersstoel, zwart	160
17	1906-S ATHENE Mobiel onderstel	70
18	1906-S ATHENE Mobiel onderstel	63
19	1906-S ATHENE Mobiel onderstel	108
20	1906-S ATHENE Mobiel onderstel	-1
21	1906-S ATHENE Mobiel onderstel	-5
22	1906-S ATHENE Mobiel onderstel	20
23	1906-S ATHENE Mobiel onderstel	-40
24	1906-S ATHENE Mobiel onderstel	40
25	1908-S LONDEN Draaistoel, blauw	234

26	1908-S LONDEN Draaistoel, blauw	5
27	1908-S LONDEN Draaistoel, blauw	-4700
28	1908-S LONDEN Draaistoel, blauw	-1
29	1908-S LONDEN Draaistoel, blauw	-10
30	1908-S LONDEN Draaistoel, blauw	10
31	1908-S LONDEN Draaistoel, blauw	-10
32	1908-S LONDEN Draaistoel, blauw	10
33	1908-S LONDEN Draaistoel, blauw	20
34	1920-S ANTWERPEN Vergadertafel	38
35	1920-S ANTWERPEN Vergadertafel	8
36	1920-S ANTWERPEN Vergadertafel	67
37	1920-S ANTWERPEN Vergadertafel	-2
38	1920-S ANTWERPEN Vergadertafel	-2
39	1920-S ANTWERPEN Vergadertafel	-1
40	1920-S ANTWERPEN Vergadertafel	-1
41	1920-S ANTWERPEN Vergadertafel	-1
42	1924-W CHAMONIX Bergmeubel basisunit	1
43	1924-W CHAMONIX Bergmeubel basisunit	2
44	1924-W CHAMONIX Bergmeubel basisunit	3
45	1924-W CHAMONIX Bergmeubel basisunit	5
46	1924-W CHAMONIX Bergmeubel basisunit	15
47	1928-S AMSTERDAM Lamp	97
48	1928-S AMSTERDAM Lamp	57
49	1928-S AMSTERDAM Lamp	149
50	1928-S AMSTERDAM Lamp	-1

This page break effect is useful if your report is always going to be exported to PDF and read on a tablet that has low resolution.



An example of this report is available in the object: Packt - CH03-3

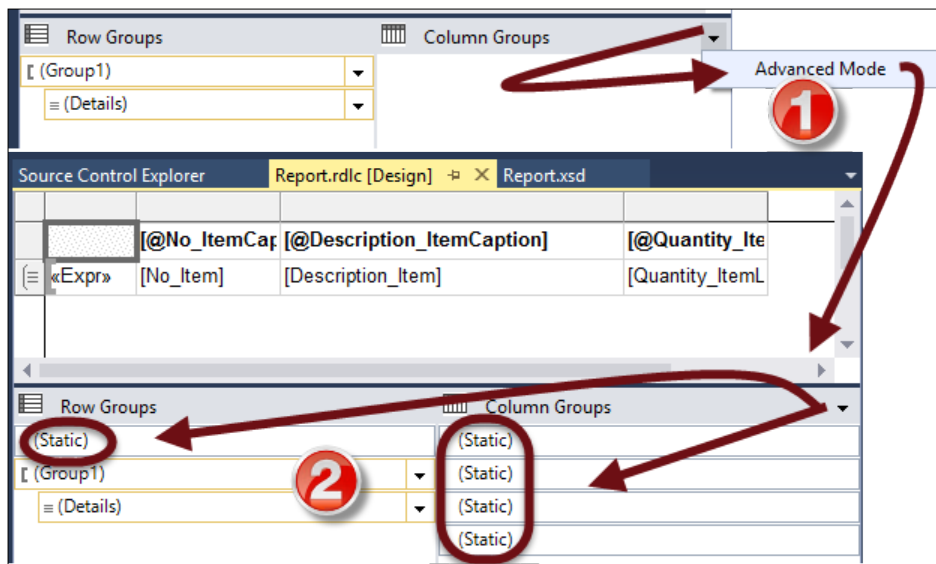
Repeating a column header on every page

In this example, you might have noticed that the column headers are not repeated on every page. In the properties of a Tablix, there are options that you can enable:

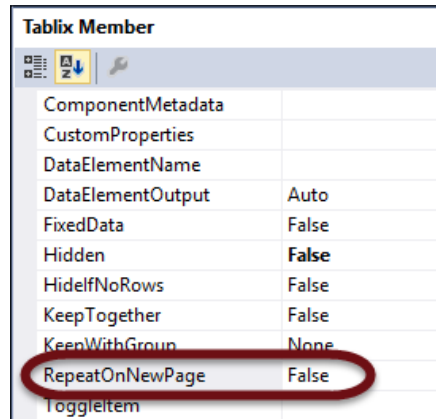
Row Headers	
<input type="checkbox"/>	Repeat header rows on each page
<input type="checkbox"/>	Keep header visible while scrolling
Column Headers	
<input type="checkbox"/>	Repeat header columns on each page
<input type="checkbox"/>	Keep header visible while scrolling

Even when you enable these **Repeat header columns on each page** options, the header row is still not repeated on every page. The reason is unknown. In fact, when you create and run an RDL report, they work, but not in RDLC. To solve this issue, use the following workaround.


First, open the advanced mode of the Tablix, as follows:



In advanced mode, the static rows of the Tablix become visible in the row and column groups pane. There, select the properties of the static row on the top in the row groups, and you will see the **RepeatOnNewPage** property:



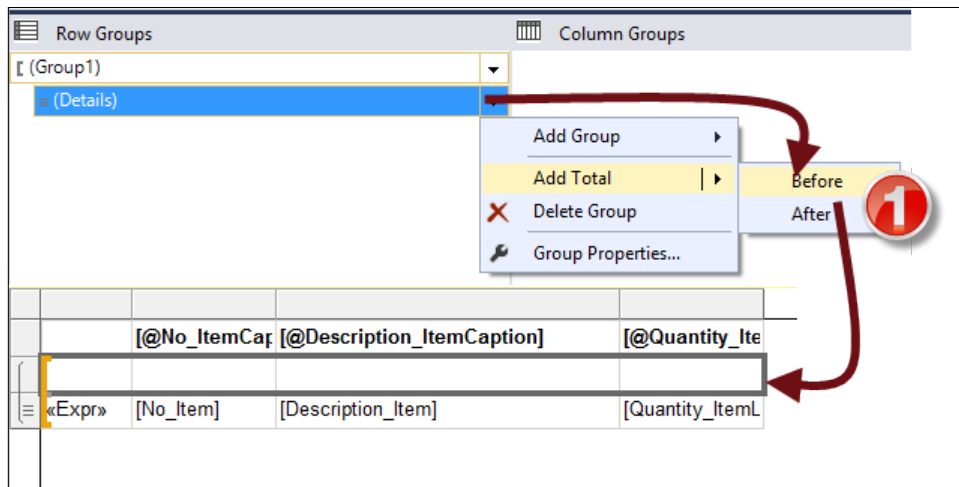
In most cases this works, but not for this report. The problem is that, when you set **RepeatOnNewPage** to **True** for the static members of this Tablix, you get the following error:

[ **Error while validating RDL content:**
The Tablix 'Tablix2' has an invalid TablixMember. All TablixMember elements in a TablixColumnHierarchy must have the RepeatOnNewPage property set to false.]

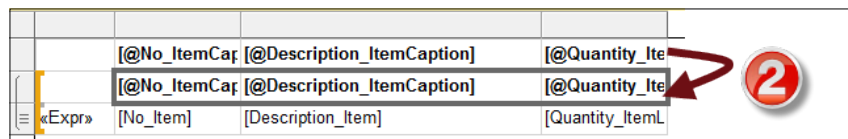
The reason for this error is that, in this Tablix, we have a group without a group header row and so the static member cannot be repeated. If the Tablix had a group header row, then the error would not happen and the header row would be repeated using this property.

To fix the problem you need to add a group header row, move the column headers to the group header row, and repeat this new group header row.

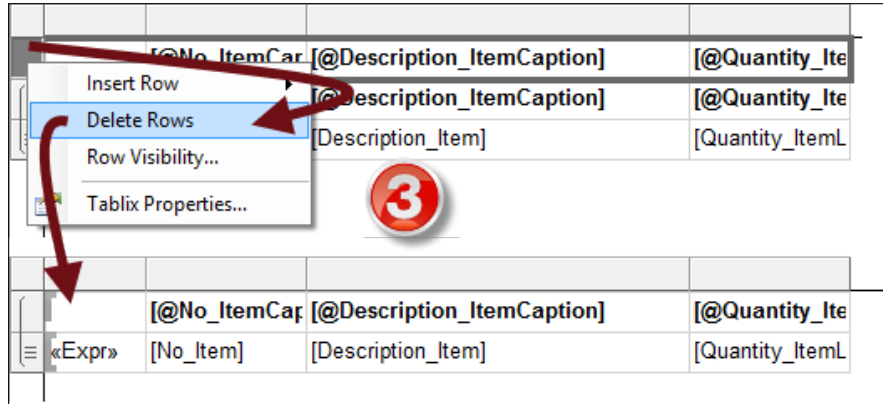
1. First, add a group header row. To do this, right-click on **Details** and select **Add Total, Before**:



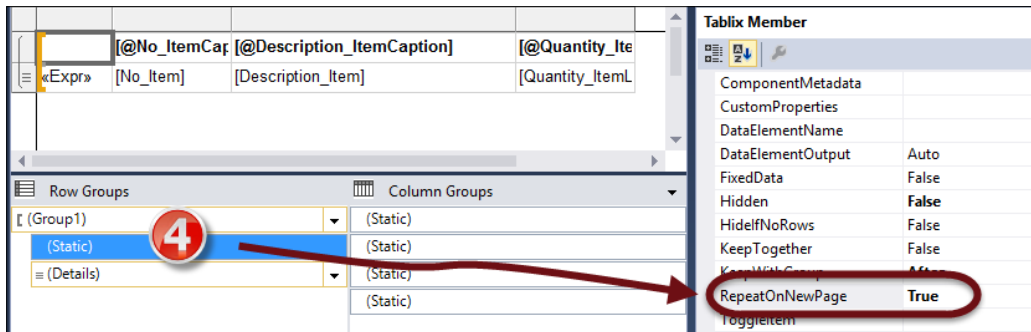
2. Then, copy/paste the columns from the first row onto the newly created group header:




- Next, delete the first row:



- Now, open **Advanced Mode**, select the **Static** member in the **Row Groups** pane, and set **RepeatOnNewPage** to **True**:




- Save and run the report. You will see that the header row is now repeated on every page.

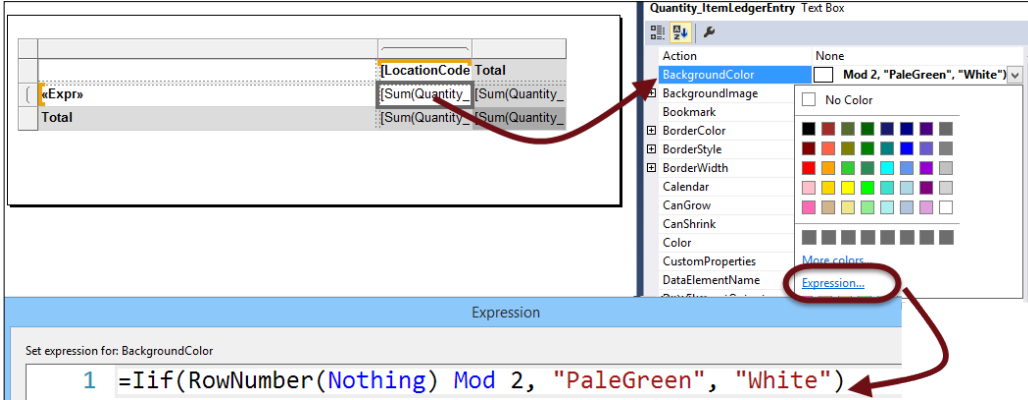
[ An example of this report, with repeating headers, is available in the object: Packt - CH03-4]

Example – the green-bar-matrix

When we applied alternating colors in the Tablix using decision functions, I mentioned that in a matrix, which is a Tablix with row and column groups, the function has an undesired result.

 For this demonstration you can import the object: Packt - CH03-5. In fact, this report continues from where we left off in the object: Packt - CH02-7.

Select the textbox under the location code in the layout of the report and, in the **BackgroundColor** property, set the following expression:



The screenshot shows the report designer interface. On the left, a table is visible with columns for LocationCode, Total, and Sum(Quantity). A red arrow points from the 'Total' column to the 'BackgroundColor' property in the Properties window on the right. The Properties window shows the 'BackgroundColor' property set to 'Mod 2, "PaleGreen", "White"'. Below the Properties window, the 'Expression' field is populated with the following code:

```
1 =Iif(RowNumber(Nothing) Mod 2, "PaleGreen", "White")
```

Now run the report and you will see the following output:

BLUE	GREEN	OUT. LOG.	OWN LOG.	RED	WHITE	YELLOW
6	5			-1		
149	-19			55		97
38	65			3		
	49	25		20		160
70	88		40	56		
44	-1			22		116
2 310		0				15
					28	

As you can see, the effect is not alternating green bars but it is more like a strange checker-board. The reason for this is that the `RowNumber` property for this textbox at runtime is not continuous because it depends on the row and column groups. In order to create a green-bar effect, you need to use another expression.

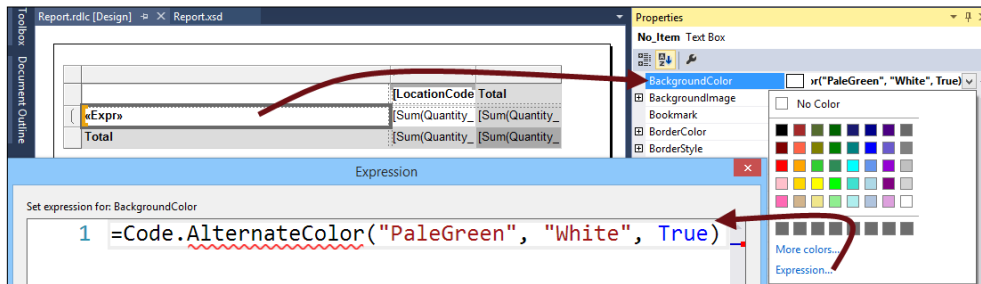
Start by adding the following to the code section of the report:

```
Private bOddRow As Boolean

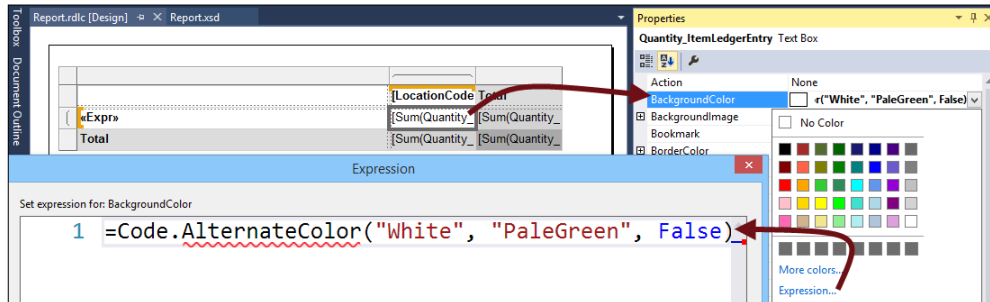
Function AlternateColor (ByVal OddColor As String, ByVal EvenColor
    As String, ByVal Toggle As Boolean) As String
    If Toggle Then
        bOddRow = Not bOddRow
    End If
    If bOddRow Then
        Return OddColor
    Else
        Return EvenColor
    End If
End Function
```

This function returns a color, depending on the third parameter that toggles the colors. When you run this function in the detail and in one of the header cells of a matrix, you can set the colors dynamically.

Now set the following expression in the **BackgroundColor** property of the textbox on the first column, second row:



Then, in the **BackgroundColor** property of the textbox in the second row, second column, set the following expression:



Now run the report, and the effect is as follows:

	BLUE	GREEN	OUT. LOG.	OWN LOG.	RED	WHITE	YELLOW	Total
ALBERTVILLE Whiteboard, green (1992-W)	6	5				-1		10
AMSTERDAM Lamp (1928-S)	149	-19				55	97	282
ANTWERP Conference Table (1920-S)	38	65				3		106
ATHENS Desk (1896-S)		49	25			20	160	254
ATHENS Mobile Pedestal (1906-S)		70	88		40	56		254
ATLANTA Whiteboard, base (1996-S)	44	-1				22	116	181



An example of the green-bar report for a matrix with alternating row colors is available in the object: Packt - CH03-6

If you change the expression for the **BackgroundColor** property for both textboxes to the following:

```
=Code.AlternateColor("PaleGreen", "White", True)
```

Then you get this result:

	BLUE	GREEN	OUT. LOG.	OWN LOG.	RED	WHITE	YELLOW	Total
ALBERTVILLE Whiteboard, green (1992-W)	6	5				-1		10
AMSTERDAM Lamp (1928-S)	149	-19				55	97	282
ANTWERP Conference Table (1920-S)	38	65				3		106
ATHENS Desk (1896-S)		49	25			20	160	254
ATHENS Mobile Pedestal (1906-S)		70	88		40	56		254
ATLANTA Whiteboard, base (1996-S)	44	-1				22	116	181
Base (70001)	2 310		0				15	2 325



An example of the green-bar report for a matrix, with alternating column colors, is available in the object: Packt - CH03-7

There are other solutions for a green-bar matrix, but the advantage of using this function is that, by making a small change in the code, you can switch between alternating row colors and alternating column colors.

Summary

In this chapter, I have explained and demonstrated how we can use and create simple and complex expressions, and use them to generate values for properties. By applying this knowledge, you can create conditional formatting effects in your reports. The scope is an important factor in determining the result of certain expressions. There are many different functions you can use, and also create yourself and, in this chapter, I have looked at a couple of the possibilities.

In the following chapters, we are going to see more examples of how we can use expressions. It is important to remember that expressions should not be too complex due to performance and your report layout should not contain any business logic. Business logic belongs in the report dataset designer, the expressions in the layout should be user interface and formatting logic.

4

Data Visualization Techniques

Creating a report is not difficult, but making it easy to understand so you can spot trends and learn from your data takes some consideration. The main goal of a report is to visualize information clearly and effectively, for example by graphical means. A report needs to create insights by communicating its key points in an intuitive way. In this chapter, you will learn about the different techniques available in Microsoft Dynamics NAV to visualize information.

An introduction to data visualization

Data visualization is a technique used for visual communication. The idea is to present information clearly and in such a way that it can be easily understood and interpreted correctly. A report in which the user does not understand the purpose or business result in a blink of an eye presented in the report is a missed opportunity. By using simple techniques, I will demonstrate in this chapter how to present information more clearly and in an intuitive way.

Recipes to implement top x filtering

A technique commonly applied when developing dashboard reports is top x filtering. You may be asked to create a report about the top five customers.

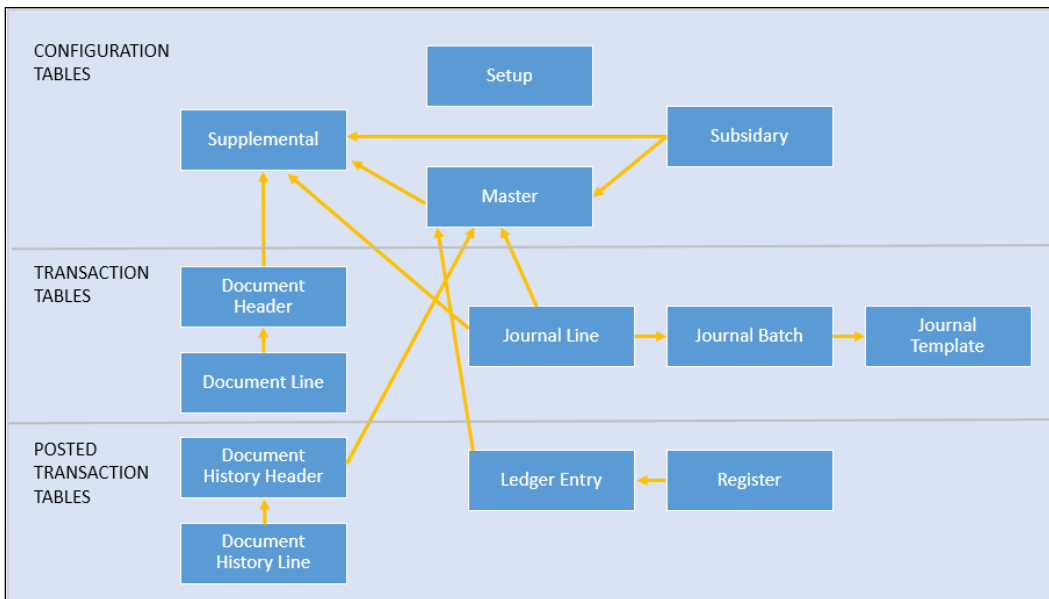


From a performance point of view, filtering should occur as soon as possible, thus minimizing the size of the dataset.

In this lesson, I will demonstrate how easy it is to create a top x filter in the layout of the report. In *Chapter 7, Performance Optimization Techniques*, I will explain and demonstrate how you can do this in order to minimize the dataset by using a query object. In this way, you will have two examples and you can decide when to apply which approach.

Imagine you have been asked to create a report in which you need to show the top five customers according to their sales. The first thing to do is to create a dataset in the report dataset designer to fetch customer and sales information. For the purpose of this demonstration, I will fetch the information from the Customer Ledger Entry table. It holds all the posted transactions for customers and looks like the correct table to use for this report.

A customer is a master record. All master tables in Dynamics NAV have a corresponding Ledger Entry table that holds the posted transactions for the master record. The data model in Dynamics NAV looks like the following figure:



So, in this example, object: Packt - CH04-1, I will use the **Customer** and **Cust. Ledger Entry** tables as data items and build the data model as follows:

E.. Data Type	Data Source	Name	I...
DataItem	Customer	<Customer>	
Column	Customer.No."	No_Customer	
Column	Customer.Name	Name_Customer	
DataItem	Cust. Ledger Entry	<Cust. Ledger Entry>	
Column	"Cust. Ledger Entry", "Posting Date"	PostingDate_CustLedgerEntry	
Column	"Cust. Ledger Entry", "Sales (LCY)"	SalesLCY_CustLedgerEntry	
Column	"Cust. Ledger Entry", "Profit (LCY)"	ProfitLCY_CustLedgerEntry	
Column	"Cust. Ledger Entry", "Salesperson Code"	SalespersonCode_CustLedgerEntry	
Column			...

I will create the layout based on this dataset and add a table. The table contains sales and profit grouped by customer. The result looks like this:

The screenshot shows a report design view with a table. The table has columns for sales and profit, grouped by customer. A red box highlights the expand/collapse control for the 'No_Customer' group, with the following properties:

Visibility	
Hidden	True
ToggleItem	No_Customer

The expand/collapse control is implemented using the following expression:

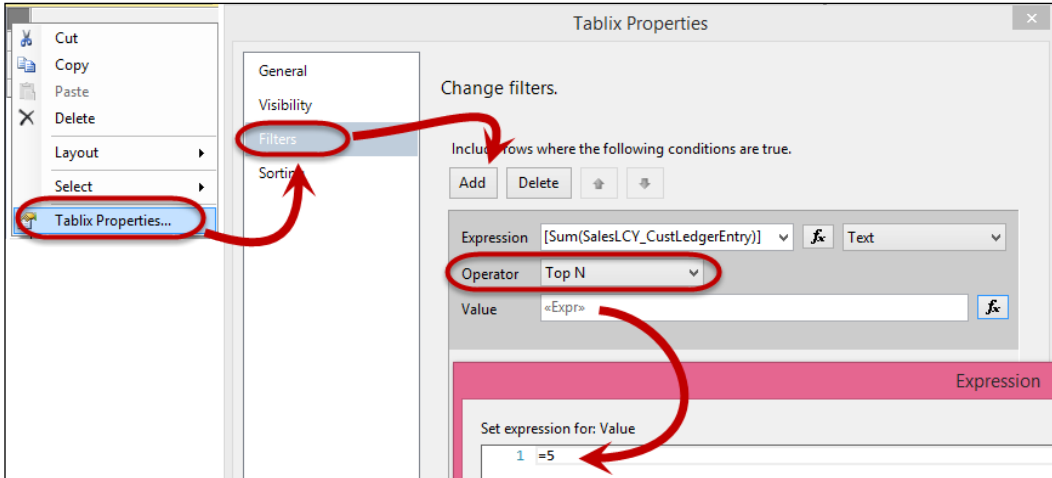
```
=Fields!No_Customer.Value & "-" & Fields!Name_Customer.Value
```


The Row Groups pane shows the 'No_Customer' group expanded to show '(Details)'. Red arrows indicate the flow from the expression box to the visibility properties and then to the row group.

In the table, in the customer number group, I implemented an expand/collapse of the details, as shown in the preceding screenshot.

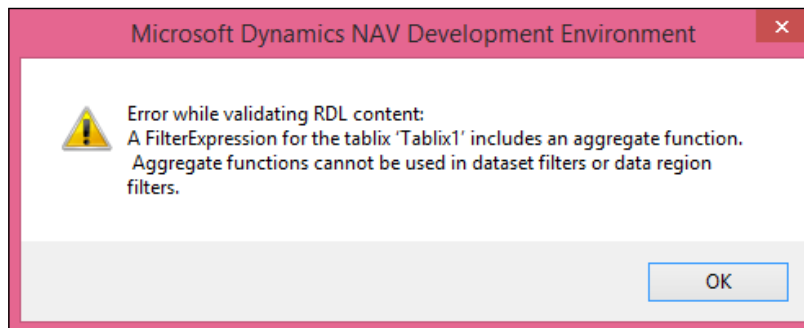
Imagine that you only want the top five customers, according to their sales, to be shown in the report. To implement such a filter, you might consider using the Tablix filter options, but that would not produce the correct results. Let's see why.

Use the **Filters** pane in the Tablix and add a **Top N** filter, as shown in the following example:



[ Remember that the value for the filter is an expression and so it must begin with an equal sign (=).]

When you save the report, the result is this:



The reason for the error is that I used the `Sum` expression on the `Sales (LCY)` field in the Tablix filter. I can remove `Sum`, because it's not allowed, so the report will run, but it will be incorrect. Let me explain why this is wrong.

If I remove Sum and run the report, the result is this:

No Customer	Verkoop (LV)	Winst (LV)
☐ 10001-Steven	21679,09	6089,99
	10245,84	1834,24
	11433,25	4255,75
☐ 20000-Anton Geestig Adviezen	9741,71	3752,21
	9741,71	3752,21
☐ 47563218-Klubben	18232,9	6381,5
	18232,9	6381,5
☐ 49858585-Hotel Pferdesees	10533,23	3248,58
	10533,23	3248,58

The top five filter is applied to the `Sales (LCY)` field, which is in the dataset and actually comes from the ledger entries. The filter works from a technical point of view but, from a financial point of view, it's totally wrong.

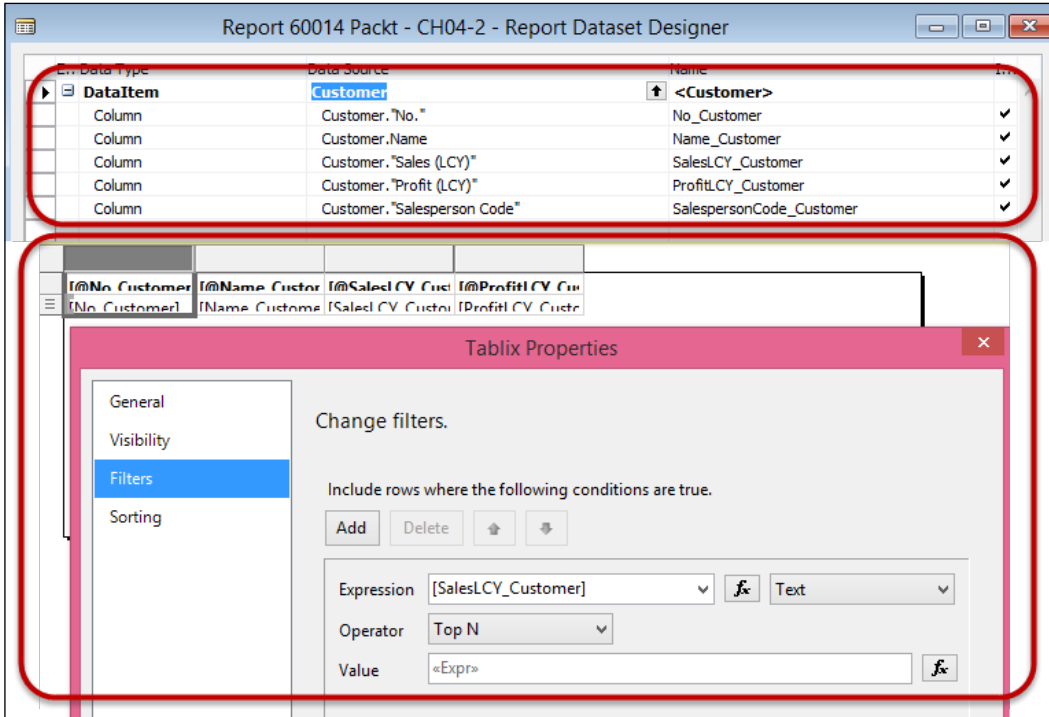
A solution to this problem in this particular report would be to ignore the ledger entries table. I can use the customer table without the `Ledger Entry` table because the customer table also holds `Sales (LCY)` and `Profit (LCY)` as FlowFields, and `Salesperson` has a date filter to filter on a date.

Of course, there are alternatives to consider. Another alternative is to create a query object for this dataset and use that. Both approaches produce better performance and results.



When you use ledger entry tables in your dataset you should pay attention to performance because ledger entry tables can contain thousands (or sometimes even millions) of records. I'm not implying that you should ignore ledger entry tables in your reports, only that you pay attention to performance and the correctness of your report.


In the object: Packt - CH04-2, I have a dataset that contains the customer number, salesperson, and sales and profit FlowFields. The dataset and layout look as follows:



Now, when you run this report, the result looks correct:

Nr.	Naam	Verkoop (LV)	Winst (LV)
10001	Steven	26399,19	8169,89
49858585	Hotel Pferdese	22298,71	7705,2
43687129	Designstudio Gmunden	21264,96	17956,66
47563218	Klubben	18232,9	6381,5
20000	Anton Geestig Adviezen	10009,43	3803,43

This is a typical example of testing your report. When running the report we saw that the figures did not seem correct and the report showed the wrong top five customers. Then, we changed the dataset to correct this problem. Remember, even though your report seems to produce results without errors, it might still be far from the truth.

 The two example reports are available in these objects:
Packt - CH04-1 and Packt - CH04-2

Conditional formatting in a report


Conditional formatting means using expressions to determine how information is formatted. In its simplest form, and in most common examples, it is used in the color (and background color) properties. An example is using the following expression for the color property of a textbox:

```
=Iif(Me.Value > 0, "Blue", "Red")
```

In this example, I'm using the `Me` object, and `Me` refers to the current textbox. Another example is this expression:

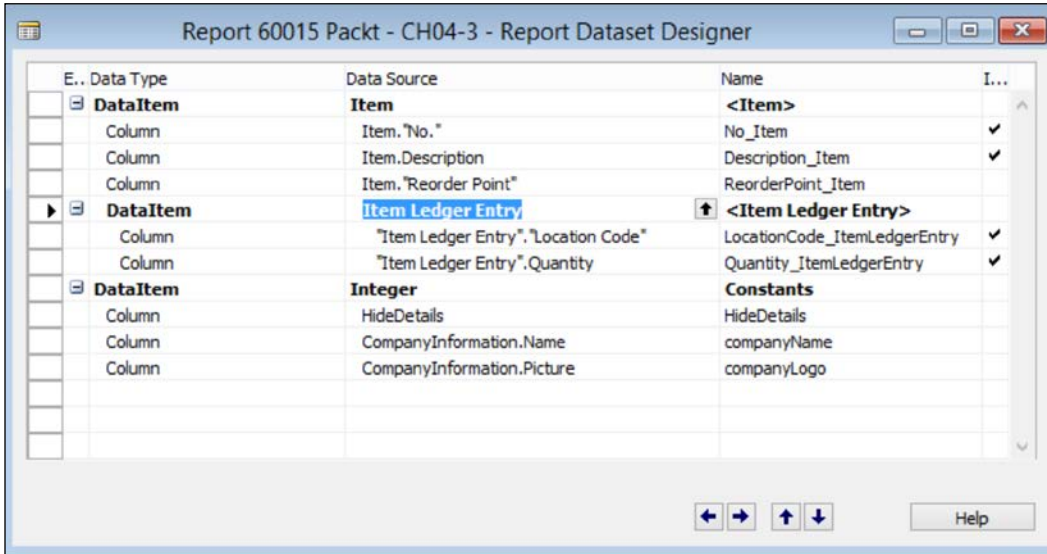
```
=Iif(Rownumber(Nothing) MOD 2, "PaleGreen", "White")
```

If you use it for the background color property of a row then it means that the even rows will become pale green and odd rows will become white.

 Sometimes `Me.Value` is not recognized in the expression builder. It will depend on if you use Report Builder or Visual Studio.

Let's use this now in a report. You can import the object: Packt - CH04-3 to follow the steps.

The dataset of the report is as follows:



I have used a matrix in the layout to show the inventory (Sum of Quantity) by Location:

	No Item	[LocationCode_ItemLedgerEntry]
([No_Item]	[Sum(Quantity_ItemLedgerEntry)]

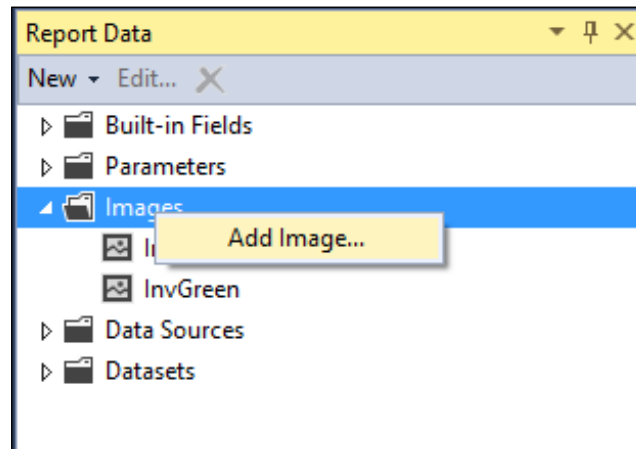
The result looks as follows:

No Item	BLAUW	GEEL	GROEN	OUT. LOG.	OWN LOG.	ROOD
1896-S			160	49	25	20
1900-S		52	160	41		46
1906-S		70		88	40	56
1908-S		234		57	0	14
1920-S		38		65		3
1924-W		1	15	8		2
1928-S		149	97	-19		55
1928-W		4	41	23		-1
1936-S		36		46	4	0

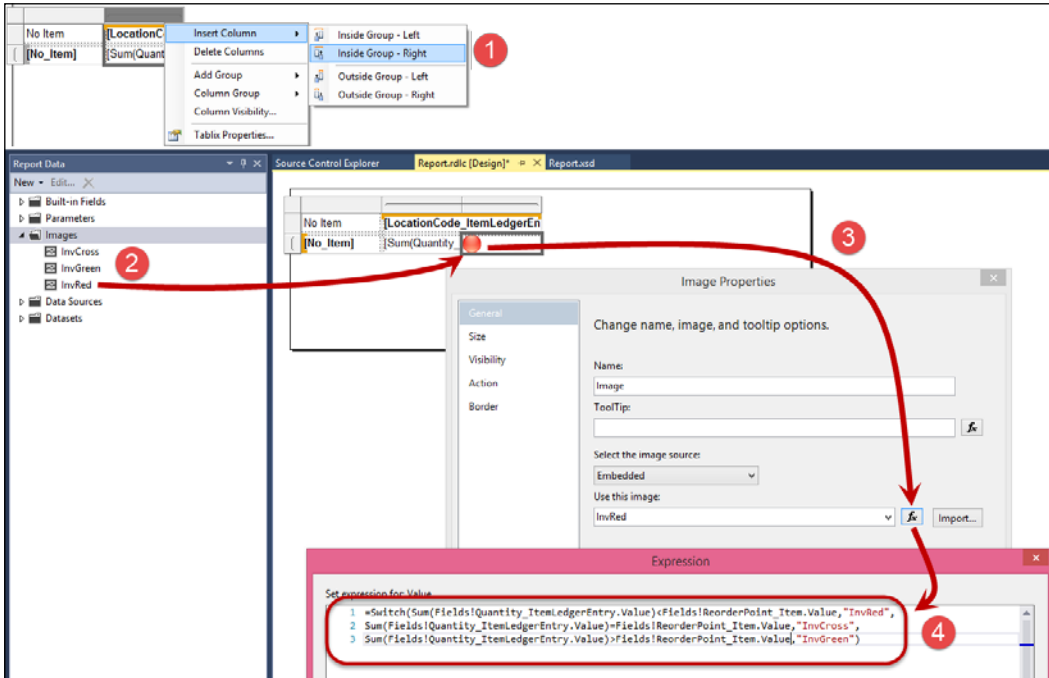
Now let's use the expression, `=Iif (Me.Value > 0, "Blue", "Red")`, for the color of the textbox:

No Item	BLAUW	GEEL	GROEN	OUT. LOG.	OWN LOG.	ROOD
1896-S		160	49	25		20
1900-S	52	160	41			46
1906-S	70		88		40	56
1908-S	234		57		0	14
1920-S	38		65			3
1924-W	1	15	8			2

Next, I will embed three images in the report. To do that, you can right-click on the Images folder in the **Report Data** menu:



Next, I will add one of the images to the Tablix:



As you can see, after adding it to the Tablix, I went into the expression for the image and used a Switch function to determine when to show which image:


```

=Switch(Sum(Fields!Quantity_ItemLedgerEntry.Value)
<Fields!ReorderPoint_Item.Value, "InvRed",
Sum(Fields!Quantity_ItemLedgerEntry.Value)
=Fields!ReorderPoint_Item.Value, "InvCross",
Sum(Fields!Quantity_ItemLedgerEntry.Value)
>Fields!ReorderPoint_Item.Value, "InvGreen")
    
```

When you run the report, the result looks as follows:

No Item	BLAUW	GEEL	GROEN
1896-S		★	160 ● 49 ●
1900-S	52 ●		160 ● 41 ●
1906-S	70 ●		★ 88 ●
1908-S	234 ●		★ -4690 ●
1920-S	38 ●		★ 65 ●

Using expressions to show images is very useful. The user can quickly see and analyze the results.

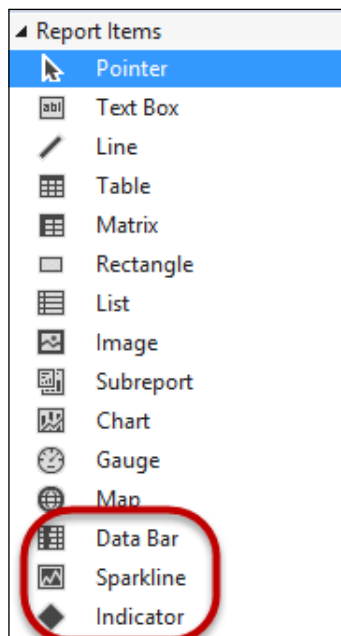
 **Use shapes and colors**
It's important with images that, apart from using different colors, you also use different shapes. Since reports aren't always printed in color and not everyone might be able to see the colors.

Analyzing your data with data bars and indicators

In the previous example, I used images and expressions to visualize the inventory. There are controls in the **Toolbox** which are better suited to perform this type of visualization. They are:

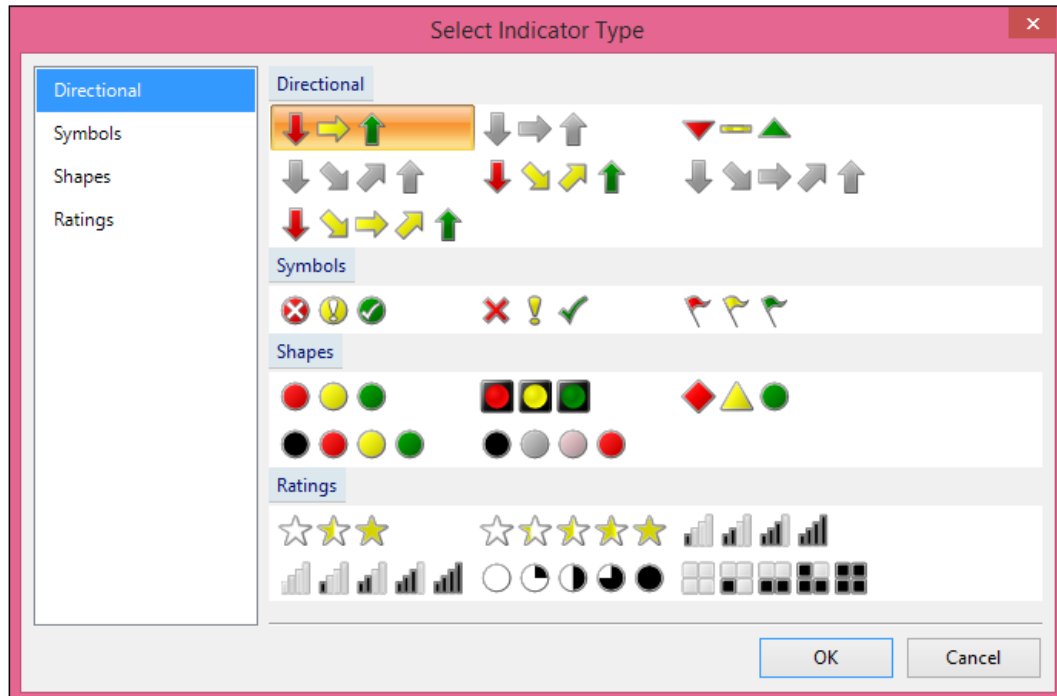
- **Data Bar**
- **Sparkline**
- **Indicator**

The following screenshot shows the controls:



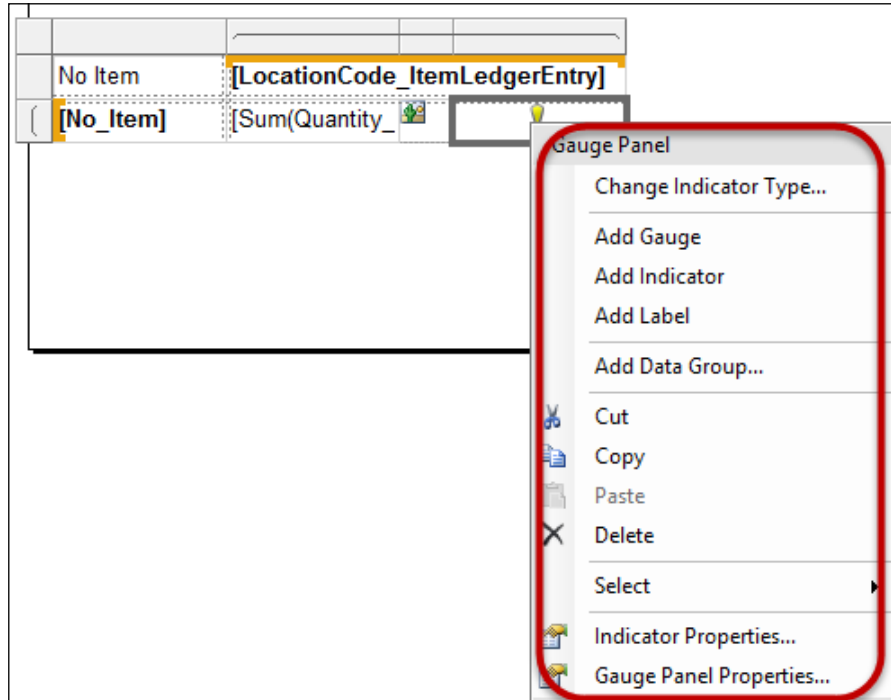
Let's start with the indicator. I will add an extra column to the right of our image in our previous object: Packt - CH04-3, and I will drag an indicator into it.

When you do this, a window opens that allows you to select an **Indicator Type**:

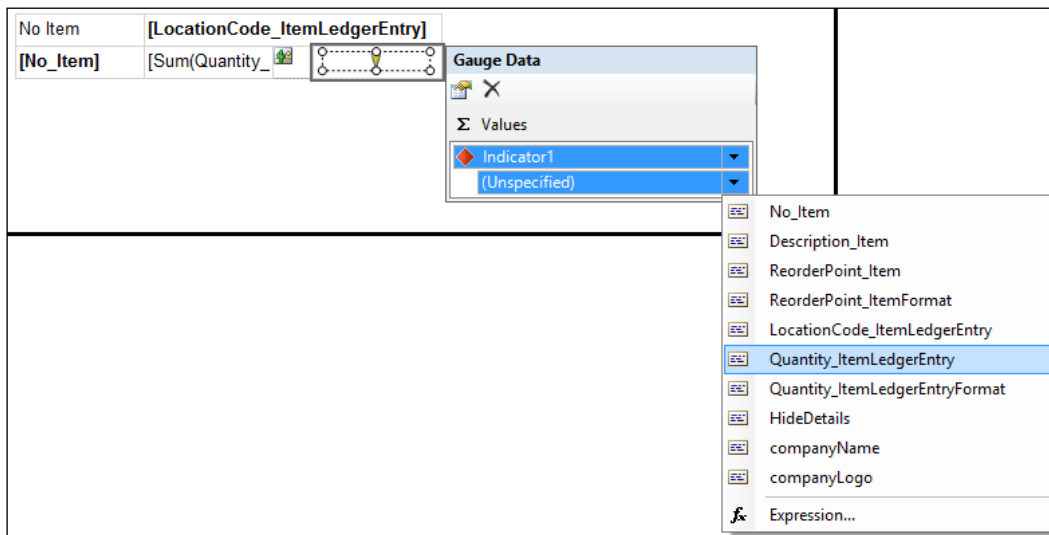


You can choose between **Directional**, **Symbols**, **Shapes**, and **Ratings** indicators. Now, regardless of which one you select, it is still possible to determine how they will be used and also to use your own images.

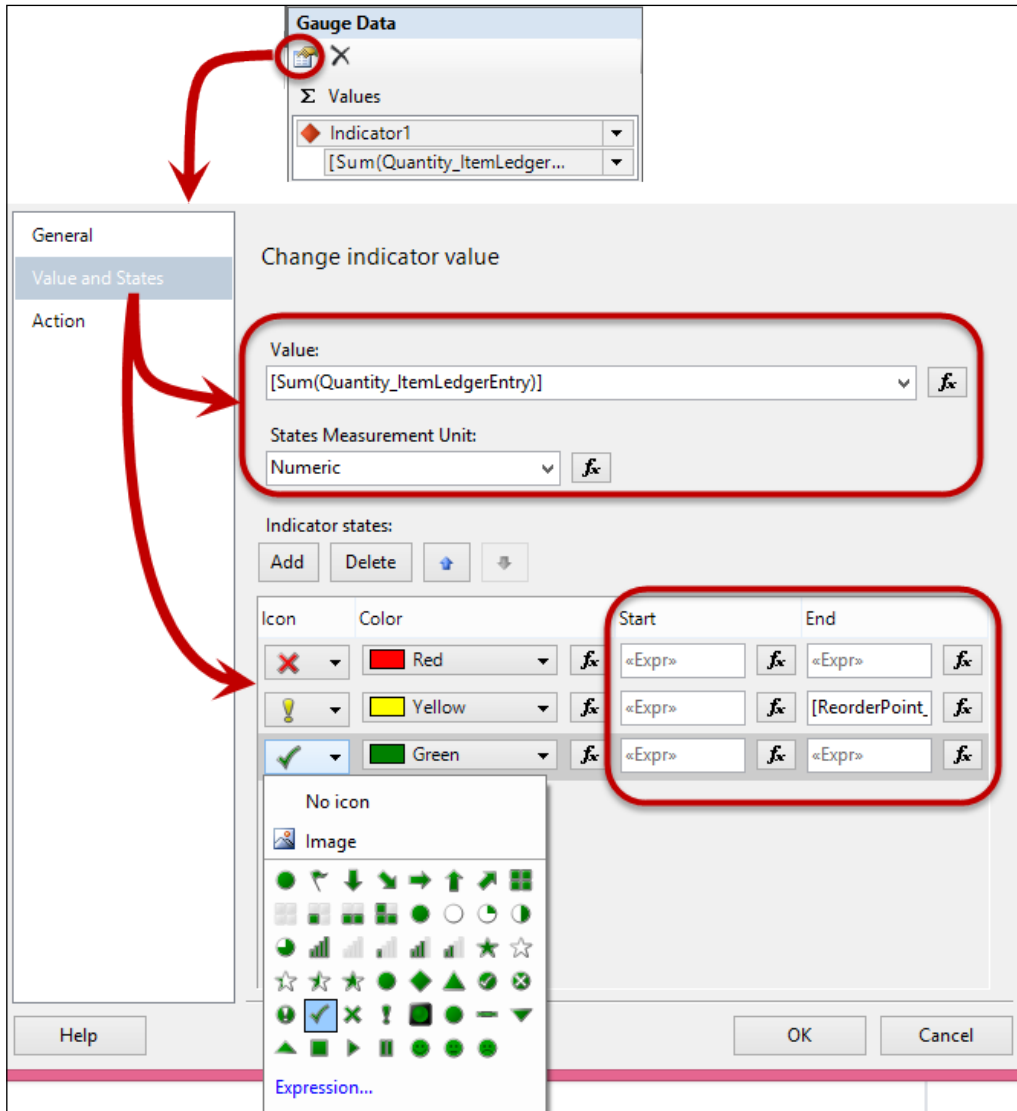
Next, right-click on the indicator to obtain the following menu:



Or left-click on the indicator to obtain the following menu and select the value that determines the indicator:



After you have selected the **Quantity_ItemLedgerEntry** field, click on the properties button to obtain the following window:



First, you need to set the **States Measurement Unit** to **Numeric** and then you can use start and end values or expressions to determine when each icon and color is shown. As you can see in this example, you can also click on an icon and, in the drop-down menu, select another icon or upload an image.

When I run the report, the result is now as follows:

No Item	BLAUW		GEEL	
1896-S		★	160	● ✓
1900-S	52	●	160	● ✓
1906-S	70	●	★	
1908-S	234	●	★	
1920-S	38	●	★	
1924-W	1	●	15	● ✗
1928-S	149	●	97	● ✓
1928-W	4	●	41	● ✗
1936-S	36	●	★	
1952-W	9	●	★	
1960-S	153	●	★	
1964-S	59	●	★	

The indicators don't completely match the image because I used slightly different values in the expressions.



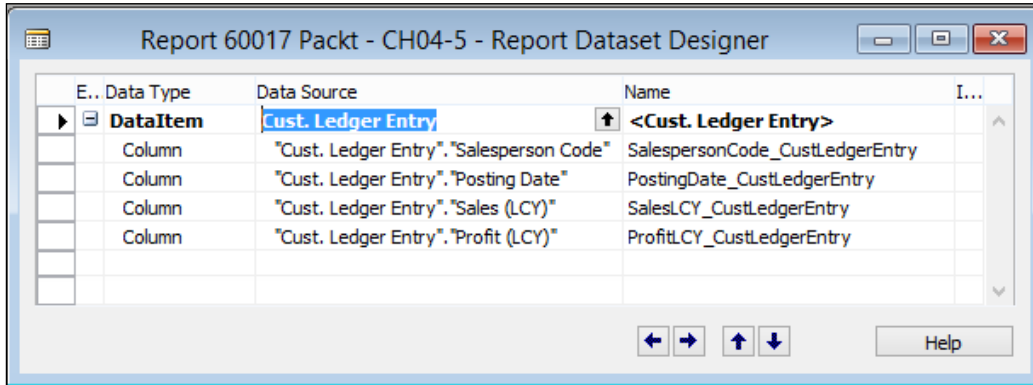
An example of this report is available in the object:
Packt - CH04-4

Data bars are used to indicate the size of a value or, in other words, to compare values visually. Instead of having to read through a list of values and figure out how they compare, you can use a data bar to make it more obvious.

In the following example, object Packt - CH04-5, I will show you how to use data bars to create the following report:

Year	Month	Weekday	HD	KS
2015	november	maandag		826,10
		dinsdag		1 646,30
	december	vrijdag	0,00	0,00
2016	januari	maandag	1 043,54	336,04
		dinsdag	2 322,61	472,81
		woensdag	18 232,90	6 381,50
		donderdag	3 135,13	523,83
		vrijdag	1 358,82	261,32
		zaterdag		
		zondag	2 483,00	547,30

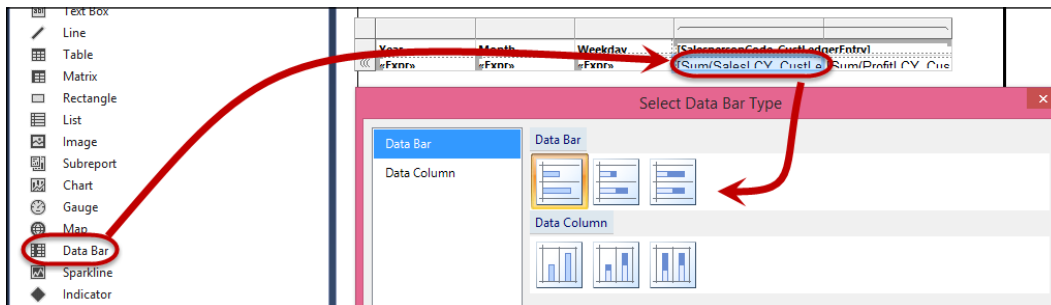
In order to get started, I have used the following dataset:



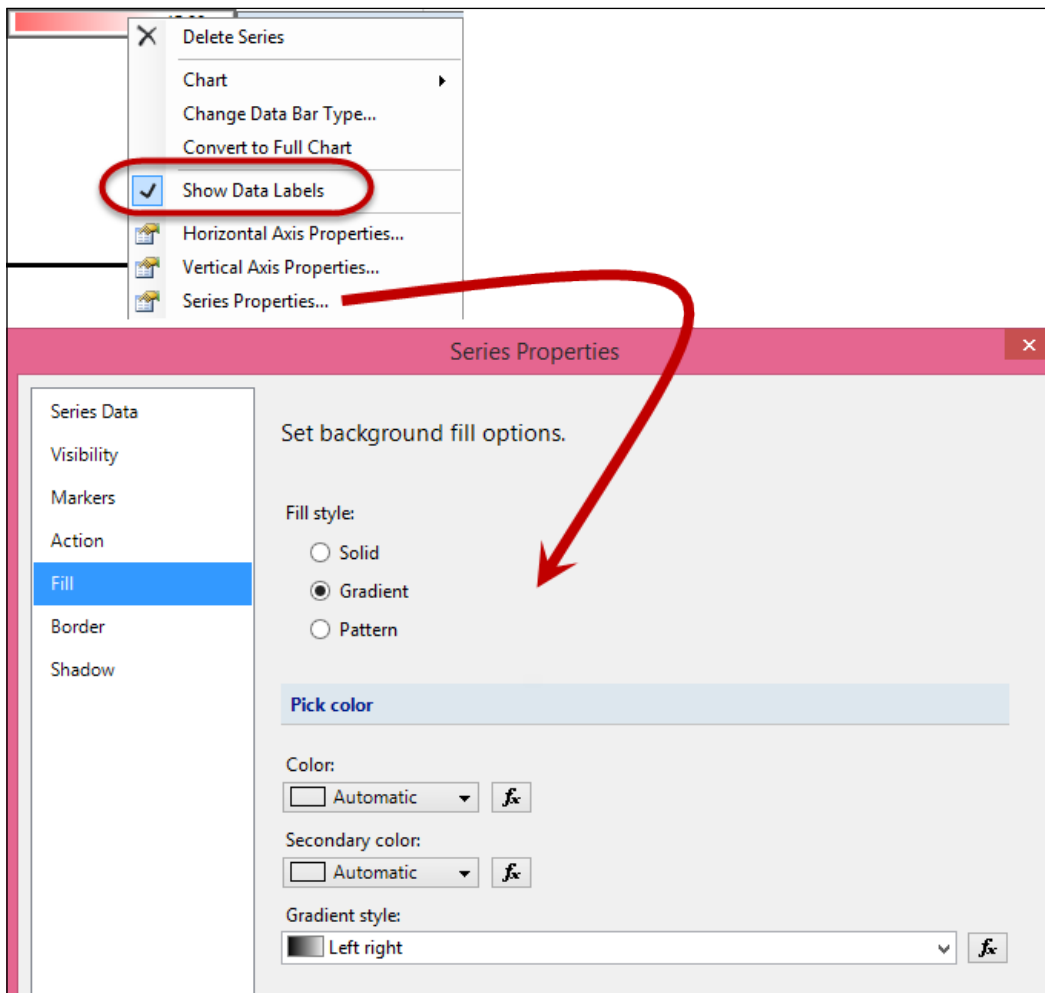
Add a matrix with two data fields to the layout: Sales (LCY) and Profit (LCY), then add Salespersoncode_CustLedgerEntry to the column groups and three groups to the row groups: year, month, and weekday. In order to group by year, month, and weekday, you can use the following expressions, based on the PostingDate field:

```
=Year(Fields!PostingDate_CustLedgerEntry.Value)  
=Month(Fields!PostingDate_CustLedgerEntry.Value)  
=DatePart(DateInterval.Weekday,Fields!PostingDate_CustLedgerEntry.  
Value)
```

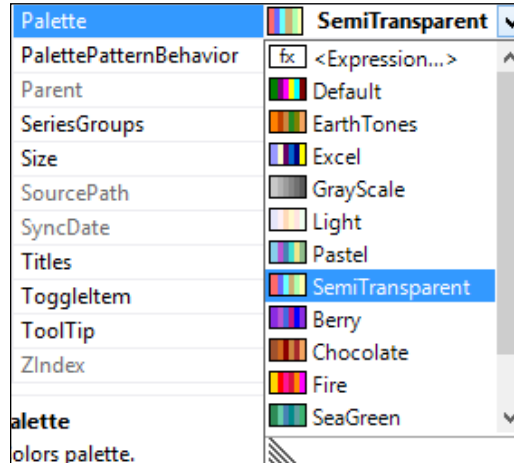
Select **Data Bar** from the toolbox and drag it onto the textbox that contains the Sales (LCY) field:




Next, right-click on **Data Bar** and activate **Show Data Labels**. Then, in **Series Properties**, select **Gradient** and a **Gradient style** of **Left right** in the **Fill** options:



You can also select a **Palette** in the properties window of the data bar, as follows:



Then, if you repeat this process for the `Profit (LCY)` field, your report is ready.

[ An example of this report is available in the object: Packt - CH04-5]

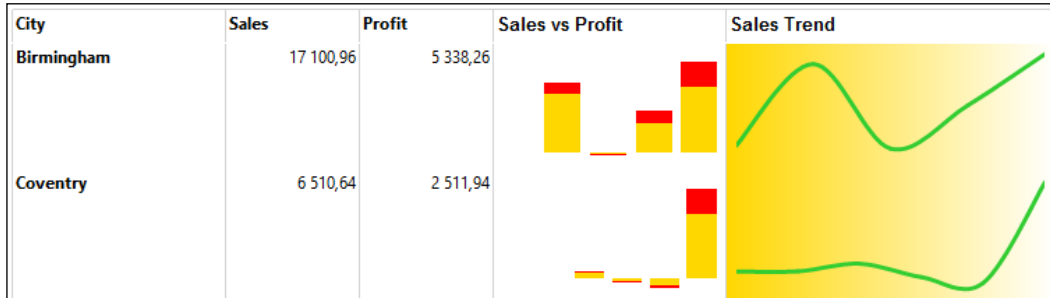
As you can see, data bars make it easier to read your report and compare values. In this example, I grouped and sorted by date and you can see that most sales are on Fridays and Tuesdays. Now, this is the demo database and it's probably a coincidence. If you change the sorting and sort the `Sales (LCY)` field from top to bottom you get a good idea of who the top selling customers are. This is best suited to a "Customer Top X report".

Using Sparklines to visualize trends

Sparklines are used to indicate trends in your data and so provide insight into what might happen in the future. Sparklines also make it easier to spot data that is out of sync or that displays non-normal behavior. In statistical terms, these are referred to as outliers. Sparklines cannot be added to the detail level in a Tablix, you have to add them to a group header or footer row. This is because they work on aggregate values.

You can consider a Sparkline as a mini-chart. I will explain the usage of the chart control later in this chapter.

The following is an example of object: Packt - CH04-6, using Sparklines:



The dataset of the report contains the following information:

Report 60018 Packt - CH04-6 - Report Dataset Designer

E.. Data Type	Data Source	Name	I...
▶ DataItem	Cust. Ledger Entry	<Cust. Ledger Entry>	
Column	"Cust. Ledger Entry"."Salesperson Code"	SalespersonCode_CustLedgerEntry	
Column	"Cust. Ledger Entry"."Posting Date"	PostingDate_CustLedgerEntry	
Column	"Cust. Ledger Entry"."Sales (LCY)"	SalesLCY_CustLedgerEntry	
Column	"Cust. Ledger Entry"."Profit (LCY)"	ProfitLCY_CustLedgerEntry	
Column	Customer.City	City	
Column	Customer."Country/Region Code"	Country	

C/AL Globals

Variables Text Constants Functions

Data Type	Subtype	Name	Length
▶ Record	Customer	Customer	


C/AL Editor

```

Documentation()
OnInitReport()
OnPreReport()
OnPostReport()
Cust. Ledger Entry - OnPreDataItem()
Cust. Ledger Entry - OnAfterGetRecord()
customer.get("Cust. Ledger Entry"."Customer No.");
Cust. Ledger Entry - OnPostDataItem()

```

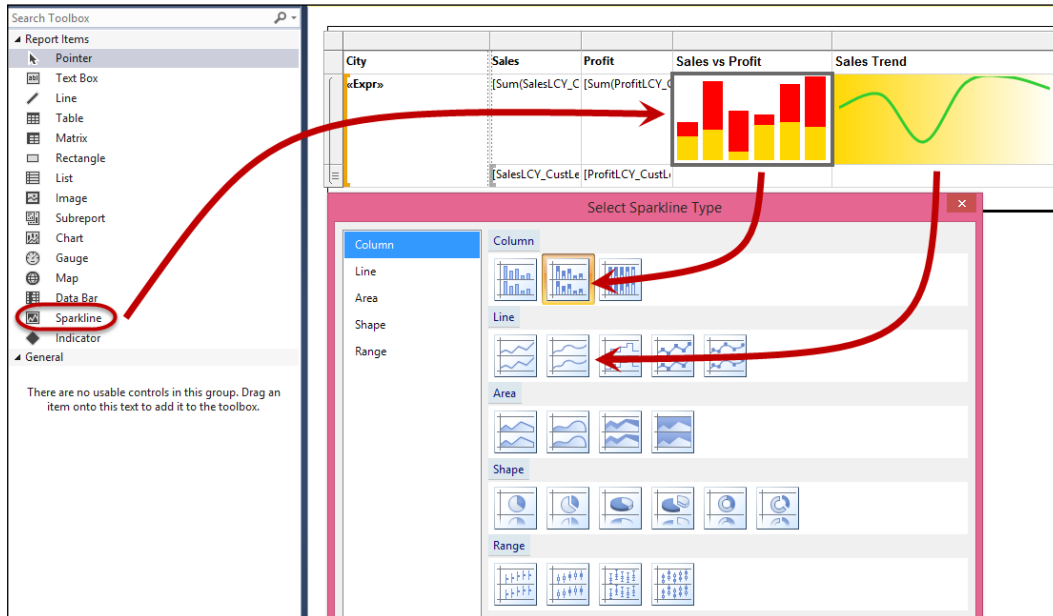
In order to keep the dataset simple, I'm using the customer ledger entry table and, in the `OnAfterGetRecord` trigger of the ledger, I'm fetching the customer information so I can add the city and country to the dataset as extra columns.

 Fetching the customer for every ledger entry is very bad for performance. It is better to start with a customer data item and indent the ledger entry below it. But, for the purpose of this example, I'm doing it the other way. As they say on television, please don't try this at home.

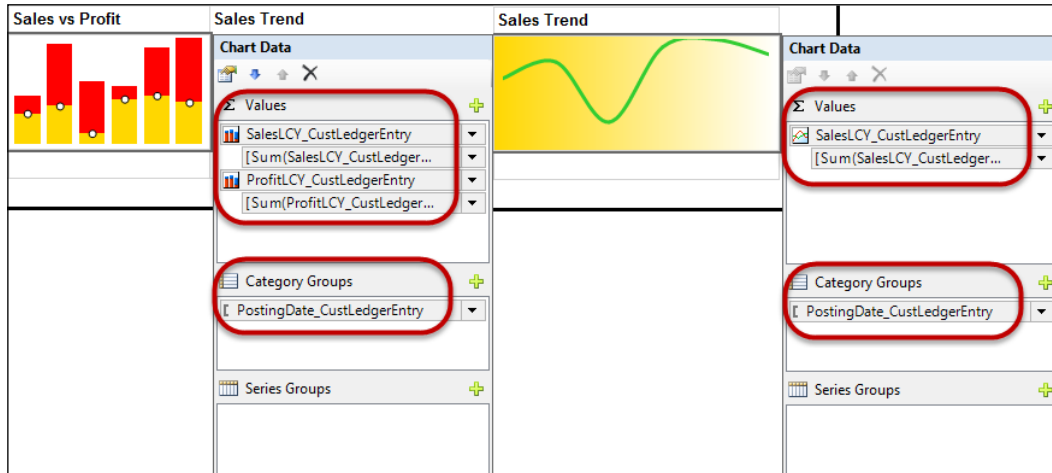
Then I created a table in the layout with a grouping of the `Sales (LCY)` and `Profit (LCY)` by `City`. In order to display the cities in the proper case capitalizing the first letter, I have used the following expression:

```
=StrConv(Fields!City.Value, VbStrConv.ProperCase)
```

I then set the row visibility of the **Details** to **Hidden**. In this report I'm not interested in the details, I only want to see grouped data, because that is where I can add Sparklines. Then, to the right, I added two extra columns into which I dragged a **Sparkline** control from the toolbox.



There are different Sparkline types to choose from. As soon as you add a Sparkline to the Tablix, the popup window opens and you can select your Sparkline type. You can then modify it in the Sparkline's properties. Then, when you click on the Sparkline, the chart data drop-down menu opens, in which you can select the date you would like to see in the mini-chart. I have used the sales and profit by `PostingDate` for the first Sparkline and the sales by `PostingDate` for the second mini-chart:



You can change the properties of each Sparkline control by selecting a color palette and changing the thickness of the lines, display labels, and so on.

[ An example of this report is available in the object: Packt - CH04-6.]

Learning how to visualize information with gauges, maps, and charts

There are other controls in the toolbox that you can use to visualize information:

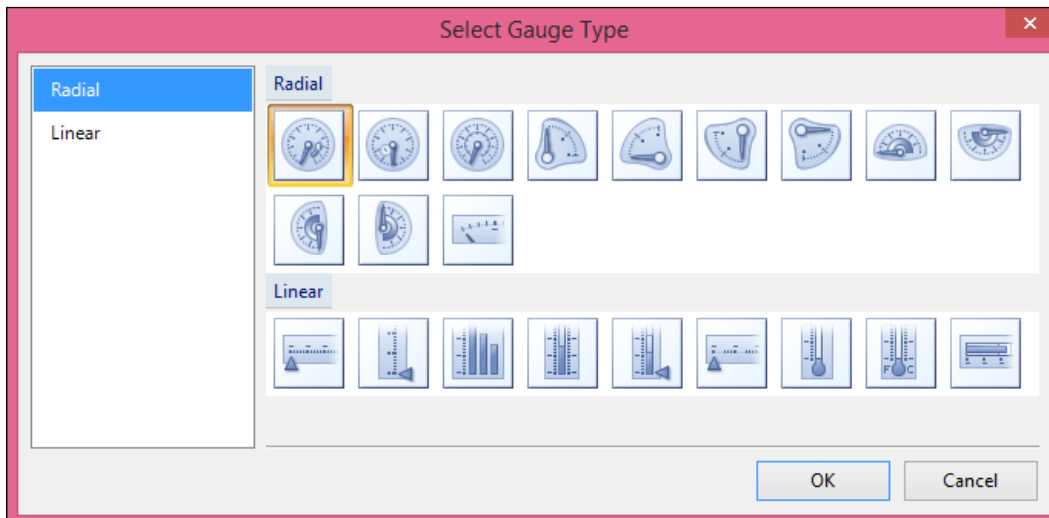
- Gauge
- Map
- Chart

Using gauges

A gauge is used to display a value from your dataset and looks like the panels you see in your car to indicate the speed and oil levels. Basically, there are two types of gauge, the one on the left I call the speedometer, and the one on the right I call the thermometer:



When you select a gauge from the toolbox and add it to the report layout, a window is displayed, in which you can select the kind of gauge:

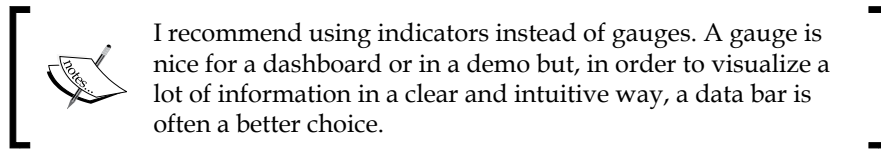


You can further customize the gauge and add elements starting from one of these options. An element could be one of the following:

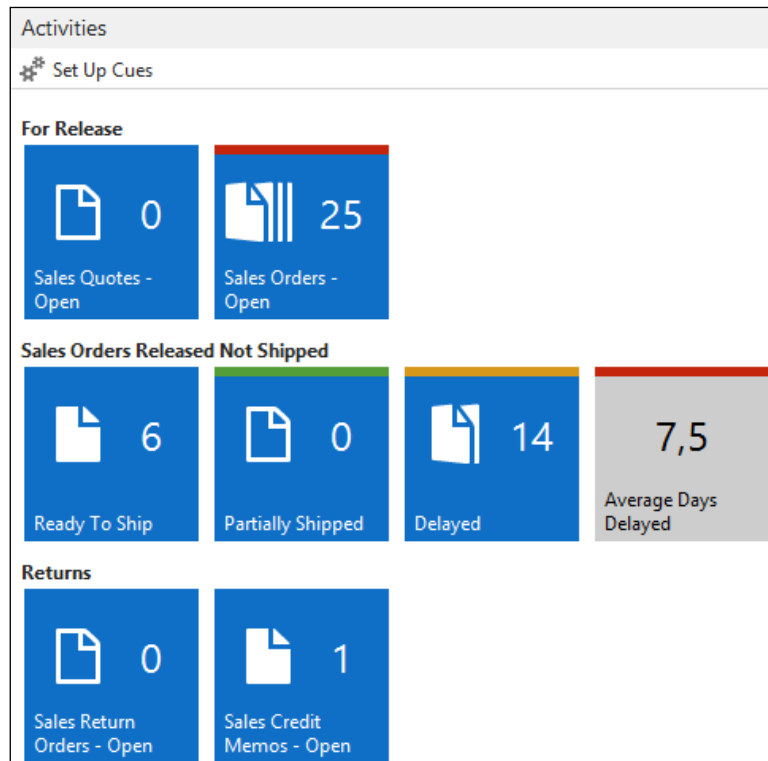
- **Pointer**
- **Range**
- **Scale**

- **Label**
- **Tick Marks**

Each element should be used to display information and make it easier for the user to understand and read the report. Adding too many elements will make your gauge look like a Swiss watch, which may be very beautiful, but it will also make your gauge too complex to understand. The idea when using a gauge in your report should always be: keep it simple. What is the information or the message I want to convey to the user and what are the minimum number of elements required to do so?



A typical example of how I demonstrate gauges when I deliver report trainings is building a report that looks like the activity pane in a role center page. There is always, or always should be, an activity pane in a role center page. The following is an example of the sales order processor activity pane:



This is how you can visualize it in a report, using gauges:



The report is available in the object: Packt - CH04-7

The dataset of the report is as follows:

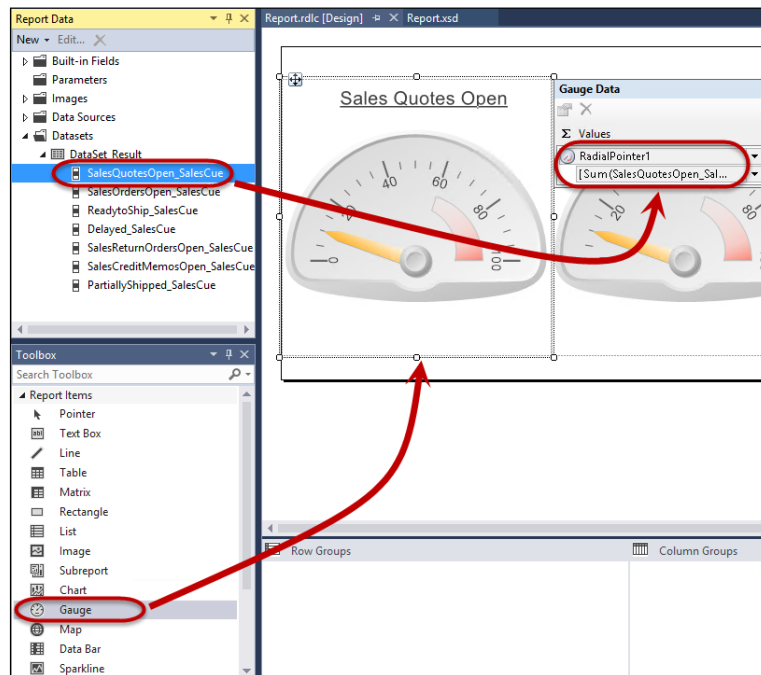
The screenshot shows two windows from SAP. The top window is 'Report 60019 Packt - CH04-7 - Report Dataset Designer'. It contains a table with the following data:

E.. Data Type	Data Source	Name	I...
DataItem	Sales Cue	<Sales Cue>	
Column	"Sales Cue", "Sales Quotes - Open"	SalesQuotesOpen_SalesCue	
Column	"Sales Cue", "Sales Orders - Open"	SalesOrdersOpen_SalesCue	
Column	"Sales Cue", "Ready to Ship"	ReadytoShip_SalesCue	
Column	"Sales Cue".Delayed	Delayed_SalesCue	
Column	"Sales Cue", "Sales Return Orders - Open"	SalesReturnOrdersOpen_SalesCue	
Column	"Sales Cue", "Sales Credit Memos - Open"	SalesCreditMemosOpen_SalesCue	
Column	"Sales Cue", "Partially Shipped"	PartiallyShipped_SalesCue	

The bottom window is 'C/AL Editor' showing the following code:

```
OnInitReport()  
OnPreReport()  
OnPostReport()  
Sales Cue - OnPreDataItem()  
  RESET;  
  IF NOT GET THEN BEGIN  
    INIT;  
    INSERT;  
  END;  
  
  SetRespCenterFilter;  
  SETRANGE("Date Filter",00,WORKDATE - 1);  
  SETFILTER("Date Filter2",>=%1',WORKDATE);  
Sales Cue - OnAfterGetRecord()  
Sales Cue - OnPostDataItem()
```

I have used the Sales Cue table to populate the dataset. I have copy/pasted the code from the OnOpenPage () trigger into the OnPreDataItem () trigger of the Sales Cue table in the report. This code will initialize the table and filter and calculate the FlowFields. Then, in the layout, you can use the gauge to create the layout, as follows:



The gauge has several properties that you can use to apply colors and formatting. You can also format the needle, the scale, and so on. In this example, I have used the gauge to display key performance indicators and single objects in the layout. You can also put the gauge inside a group header or footer, as in the Sparkline control. In real life, the gauge report item isn't used that much. This is because it's limited in visualization and it takes up a lot of space. Also, there are other report items, such as databars, indicators, sparklines, and charts, that have a slicker design than the gauge.



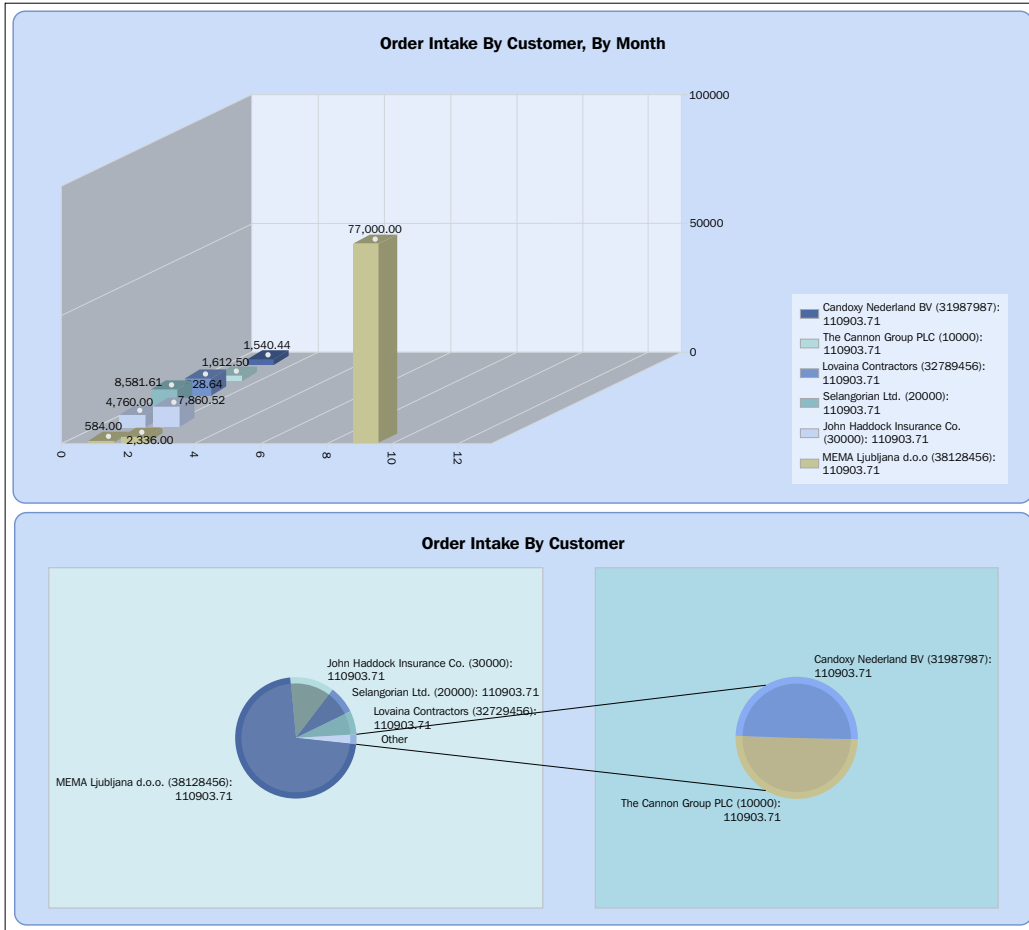
Convert items into a chart

Every time you use a databar, indicator, sparkline, or gauge you will notice that you can convert them into a full chart. The chart control is also available in the toolbox, and you could argue that sparklines, databars, gauges, and indicators behave as templates for charts.


Using charts

Let's have a look at the chart control and see how we can use it to visualize information.

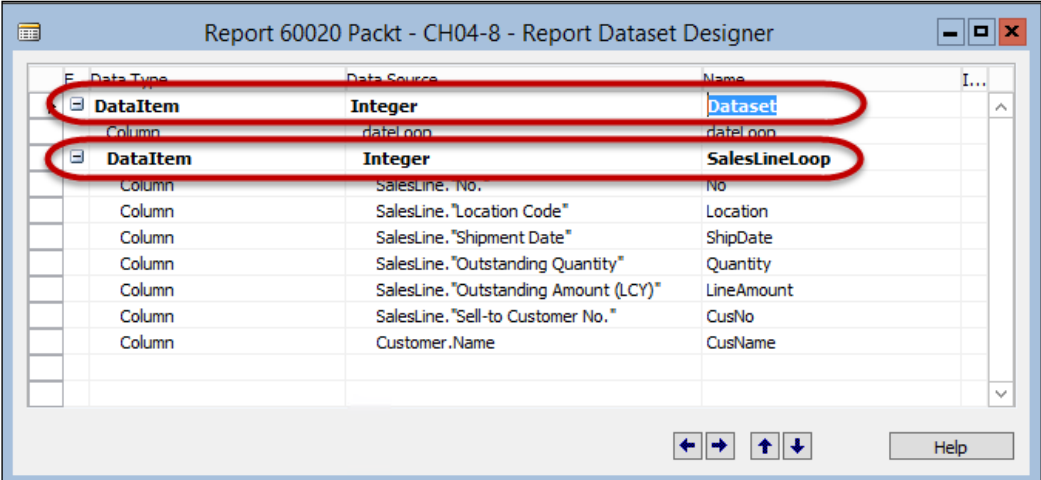
In this example, I have used the chart control to develop a report that visualizes the order intake, and the result looks as follows:



The order intake can be defined in different ways, the definition that I will use is: the amount or quantity of sales that has not yet been posted to the ledgers. In the case of Dynamics NAV, it means that we will fetch the information from the `Sales Line` table. This information needs to be presented on a timeline. The question is, how are we going to do that? You might think, that's easy, just take the `PostingDate` or `ShipmentDate` and use that to create a timeline. Well, if you follow that approach then you will get gaps or jumps in your timeline. For example, if there are sales planned on day x and on day $x+3$, then day $x+2$ will not be shown on the timeline. What you need is to have a value for every date in every month, use that as the timeline, and show a value for the dates on which there are sales. In many systems, for example in the business intelligence world, you can use a date or time dimension to accomplish this. But, in Dynamics NAV, we don't have a date dimension. I will use the integer table instead. The integer table is a virtual table in Dynamics NAV. So, I need to use the integer table, then filter it on a specific period, loop over those days and, for every day in that period, go and find the sales and add them to the dataset. This will require some programming magic in C/AL.

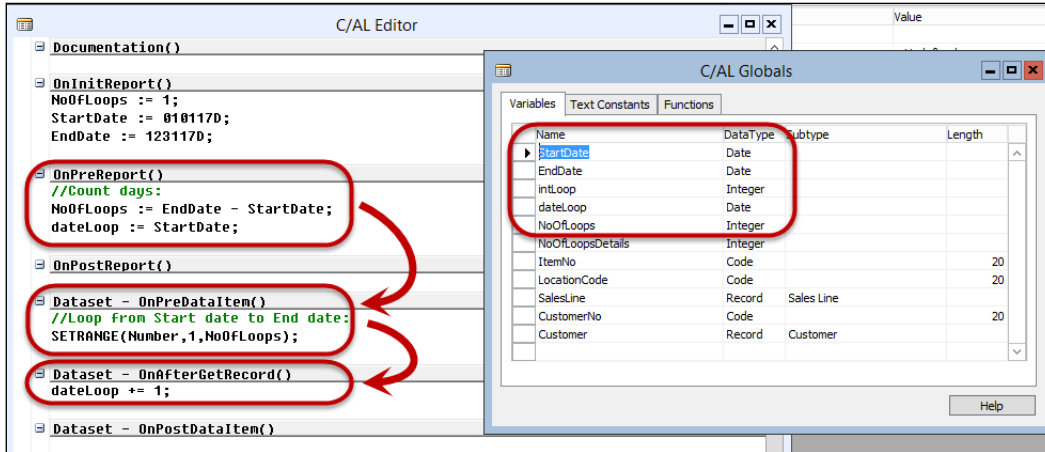

 Another approach would be to use the date table, which is also a virtual table in Dynamics NAV, but I find it easier to use the integer table in this example.

I will start by building my dataset, as follows:

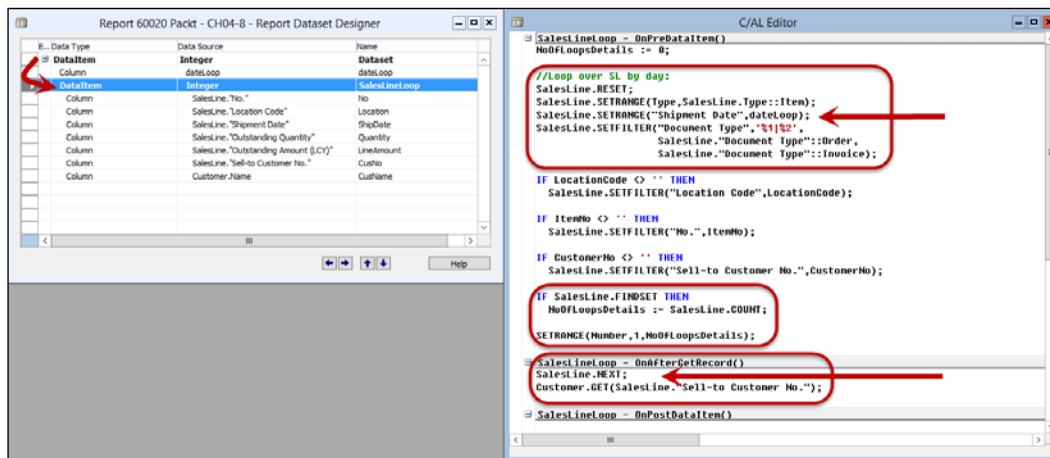


Data Type	Data Source	Name	I...
DataItem	Integer	Dataset	
Column	dateLoop	dateLoop	
DataItem	Integer	SalesLineLoop	
Column	SalesLine.No.	No	
Column	SalesLine."Location Code"	Location	
Column	SalesLine."Shipment Date"	ShipDate	
Column	SalesLine."Outstanding Quantity"	Quantity	
Column	SalesLine."Outstanding Amount (LCY)"	LineAmount	
Column	SalesLine."Sell-to Customer No."	CusNo	
Column	Customer.Name	CusName	

As you can see, I'm using the integer table twice as a data item, once for the dates and once for the sales lines. Then, in the C/AL code, I have built the dataset as follows:



When the report runs, I will initialize the variables, `StartDate` and `EndDate`. I will also add these two fields to the request page, so that the user is able to modify the period. Then, I need to count the number of days in between the start and end date and that will become the `NoOfLoops`. The number of loops value is then used to filter the integer table from 1 to `NoOfLoops`. Next, I need to find the sales for every day, returned by the dataset (integer). I will do that using another integer data item, named `SalesLineLoop`, which I indented below the dataset integer data item, as shown in the following screenshot:



You see in the C/AL code here that I have filtered the Sales Lines on the Shipment Date with the date from the dataset, and I'll count the sales to determine how many times I need to loop in the SalesLineLoop integer data item. Then, in the inner loop, I'll add the Sales Line information to the dataset.

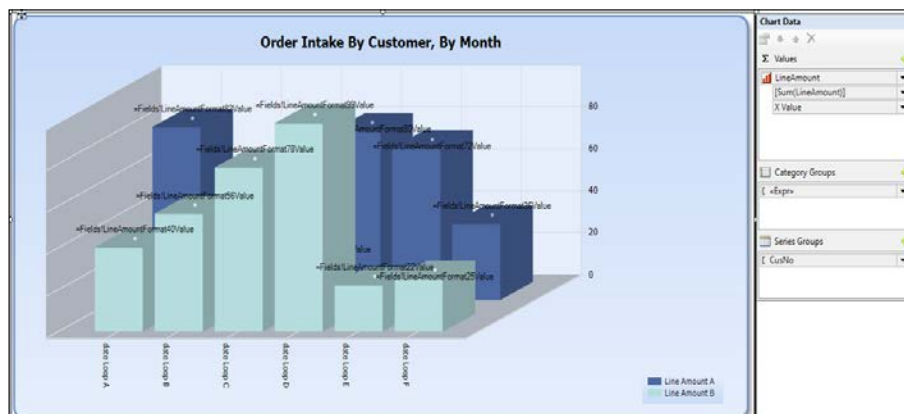
When I run the report, it will produce a dataset that looks as follows:

About This Report: Packt - CH04-8

DateLoop	No	Location	ShipDate	Quantity	QuantityFormat	LineAmount	LineAmountF...	CusNo	CusName
1/20/2017	1964-W	GREEN	1/20/2017	1	###0,####	292	###0.00	38128456	MEMA Ljubjan...
1/21/2017	<>	<>	<>	<>	<>	<>	<>	<>	<>
1/22/2017	<>	<>	<>	<>	<>	<>	<>	<>	<>
1/23/2017	1928-W	RED	1/23/2017	1	###0,####	290.79	###0.00	31987987	Candoxy Neder...
1/23/2017	1976-W	RED	1/23/2017	1	###0,####	217.69	###0.00	31987987	Candoxy Neder...
1/23/2017	1964-W	RED	1/23/2017	1	###0,####	248.2	###0.00	31987987	Candoxy Neder...
1/23/2017	70060	RED	1/23/2017	1	###0,####	10.48	###0.00	31987987	Candoxy Neder...
1/23/2017	1896-S	RED	1/23/2017	1	###0,####	616.93	###0.00	31987987	Candoxy Neder...
1/23/2017	1908-S	RED	1/23/2017	1	###0,####	117.13	###0.00	31987987	Candoxy Neder...
1/23/2017	1928-S	RED	1/23/2017	1	###0,####	33.82	###0.00	31987987	Candoxy Neder...
1/23/2017	70102	RED	1/23/2017	1	###0,####	2.7	###0.00	31987987	Candoxy Neder...
1/23/2017	70102	RED	1/23/2017	1	###0,####	2.7	###0.00	31987987	Candoxy Neder...
1/24/2017	<>	<>	<>	<>	<>	<>	<>	<>	<>
1/25/2017	<>	<>	<>	<>	<>	<>	<>	<>	<>
1/26/2017	1920-S	RED	1/26/2017	0	###0,####	0	###0.00	10000	The Cannon Gr...
1/27/2017	1964-W	GREEN	1/27/2017	1	###0,####	328.5	###0.00	20000	Selangorian Ltd.
1/27/2017	1976-W	GREEN	1/27/2017	1	###0,####	288.11	###0.00	20000	Selangorian Ltd.
1/27/2017	1992-W	RED	1/27/2017	4	###0,####	3314.32	###0.00	32789456	Lovina Contra...
1/27/2017	1992-W	RED	1/27/2017	4	###0,####	3314.32	###0.00	32789456	Lovina Contra...
1/28/2017	<>	<>	<>	<>	<>	<>	<>	<>	<>
1/29/2017	<>	<>	<>	<>	<>	<>	<>	<>	<>

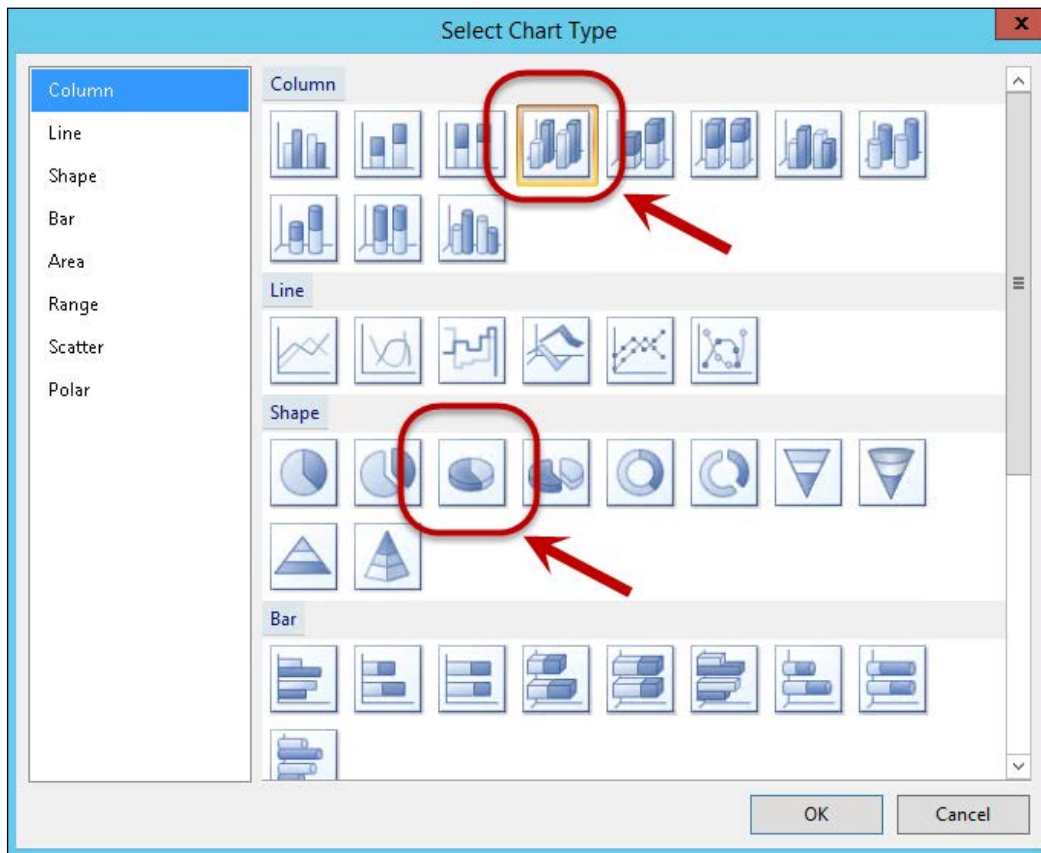
Now we are ready to build the report layout and visualize this information on a chart.

You can select the chart from the toolbox and drag it onto the body. A chart is actually similar to a matrix. As in a matrix, in a chart you can display a numerical field as the value, and group it in series and categories:

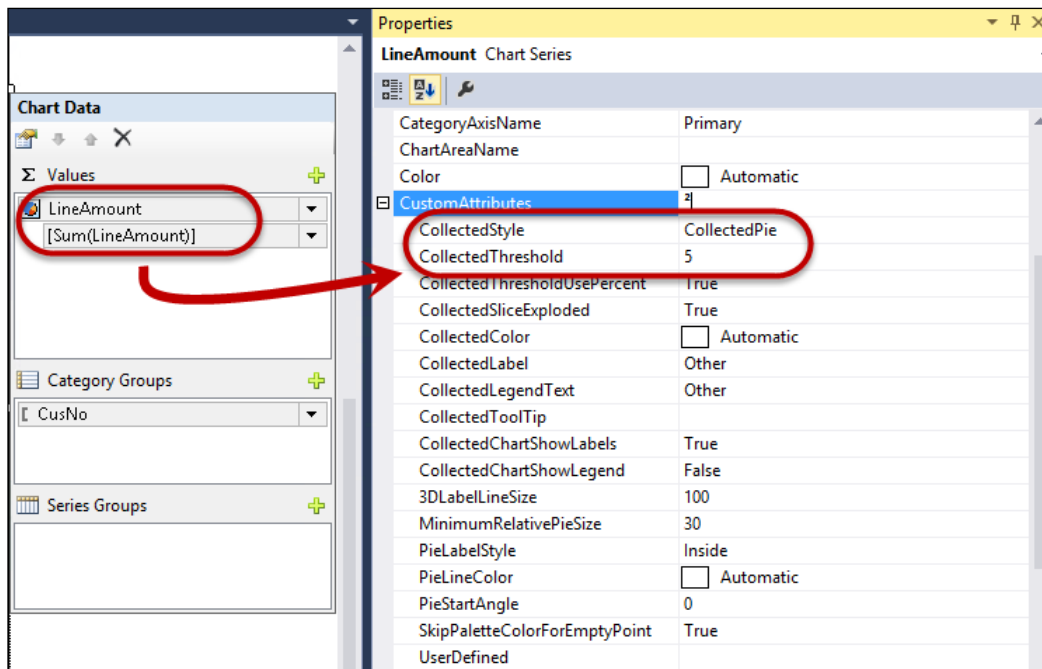


In this chart, I have calculated the sum of the **LineAmount** and grouped on **CusNo** and **MonthName** (=MonthName (Fields!dateLoop.Value)).

A window opens when you add a chart to your report and you can select from different chart types. In this example, I selected a 3D bar chart and, below that, I added a second chart, a 3D pie chart:




I did not add a series group to the second chart and I activated a special feature called **CollectedTreshHold** in the `LineAmount` properties:



This property makes sure that the chart only shows five pieces of pie, the rest of the pieces are visualized in a separate chart, to the right.

[




Optimizing charts

A recommendation when visualizing information in charts is to avoid too many slices or bars. A maximum of four or five is enough, the rest can be collected and shown as "Others", or in a separate visualization. Otherwise, your chart will contain too much information and become unreadable.

]

[

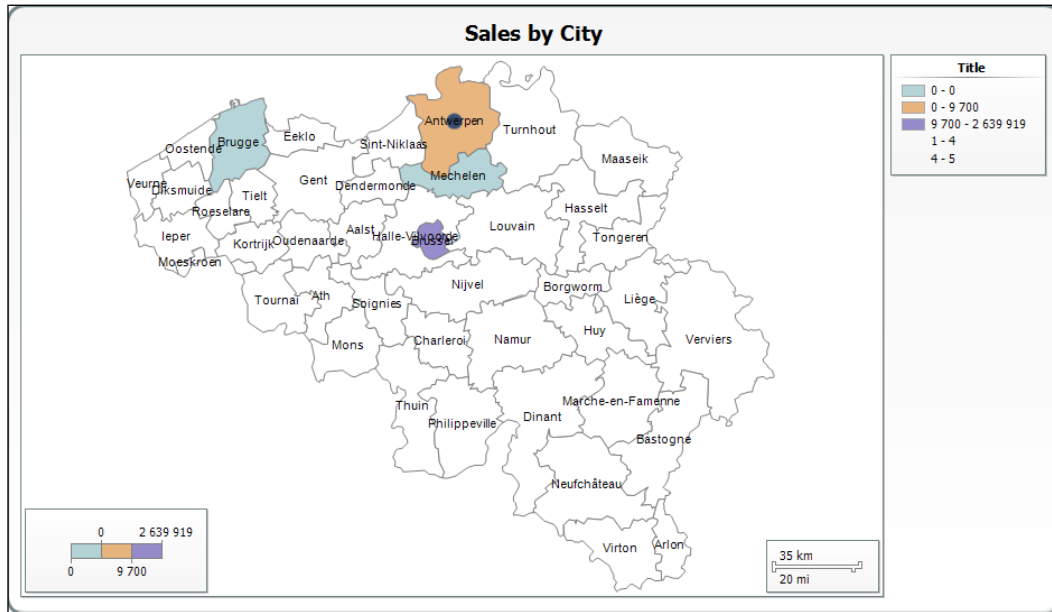


An example of this report is available in the object Packt - CH04-8.

]

Using maps

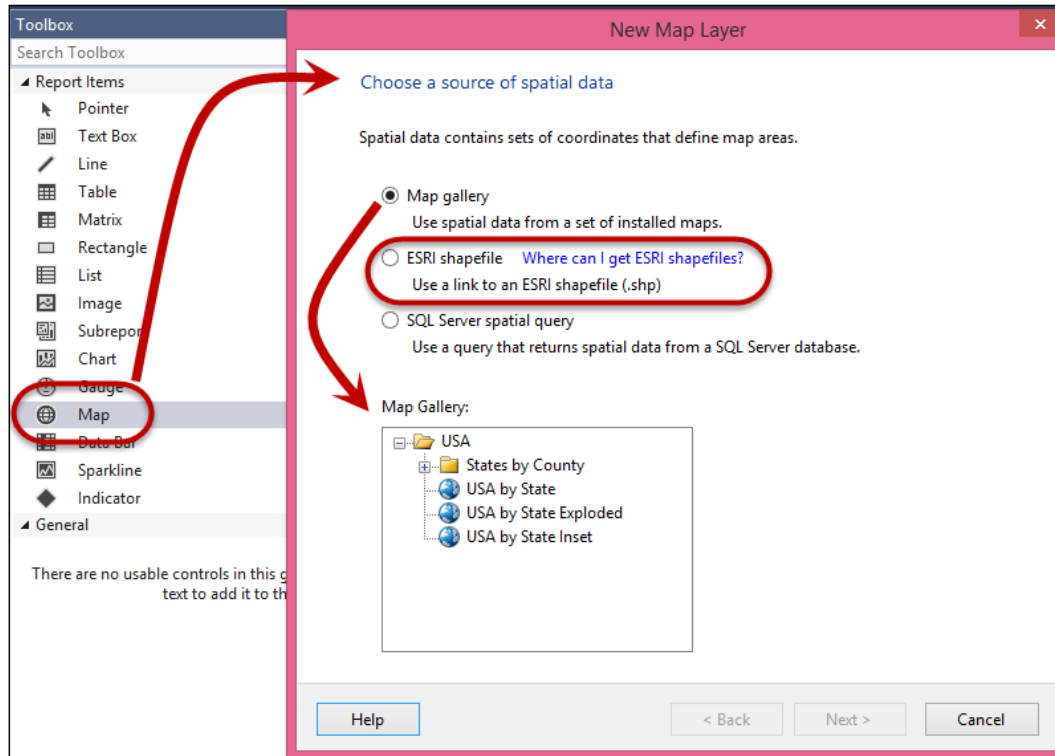
I'm going to display customer sales on a geographical map in the following example. The result of the report is as follows:



The interesting fact about this report is that the information is shown on a map of Belgium. In order to create the layout, you need a dataset that contains geographical information such as cities, postal codes, or countries. I will fetch the information directly from the customer table for this report:

E.. Data Type	Data Source	Name	I...
DataItem	Customer	<Customer>	
Column	Customer.Name	Name_Customer	
Column	Customer.City	City_Customer	
Column	Customer."Country/Region Code"	CountryRegionCode_Customer	
Column	Customer.County	County_Customer	
Column	Customer."Post Code"	PostCode_Customer	
Column	Customer."Sales (LCY)"	SalesLCY_Customer	
Column	Customer."Profit (LCY)"	ProfitLCY_Customer	

Select **Map** from the toolbox in the layout and drag it onto the report body. Then, the following window opens in which you decide the type of map you want to add:



By default, there are various maps available in the map gallery, but they are all from the US. In some or most cases, your data will come from other regions of the world. You can then select an ESRI shape file instead. When you click on the link in the preceding screenshot, you are redirected to a Microsoft website, where you can find links to other sites where you can download ESRI shape files for different countries from all over the world:

Find ESRI Shapefiles for a SQL Server 2008 R2 Reporting Services Map (SSRS)

ESRI shapefiles contain data that complies with the Environmental Systems Research Institute, Inc. (ESRI) shapefile spatial data format. ESRI shapefiles are available on the Web from a variety of sites. The term "shapefile" applies to a set of related files. To add a map to a report, you must have both the .shp file that contains spatial data and the matching .dbf file that contains supporting information.

Note Spatial data and other data that is contained in ESRI shapefiles can be politically sensitive and possibly copyrighted. Check the terms of use and privacy statements for the ESRI shapefiles to understand how you can use spatial data in your report.

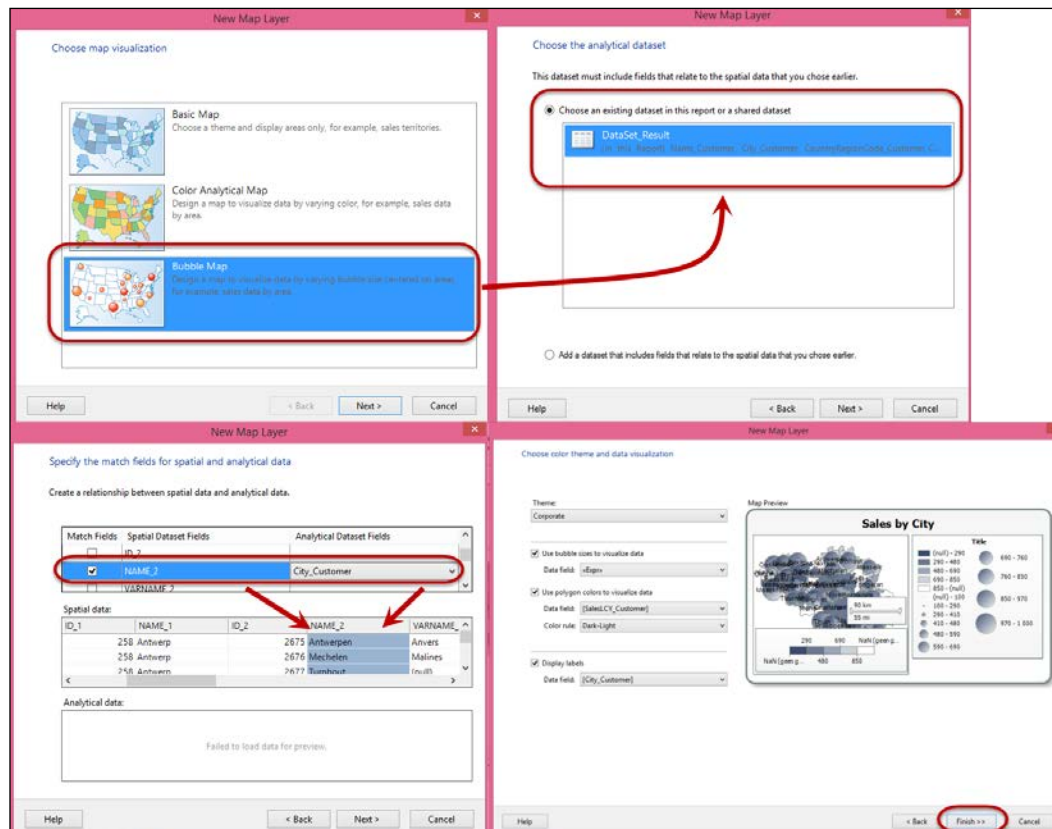
Finding ESRI Shapefiles from Public Domain Sites

You can download ESRI files from public domain data sources on the Web, including government and university sites. For example, the built-in map feature in SQL Server 2008 R2 Reporting Services uses data from TIGER/Line Shapefiles provided courtesy of the U.S. Census Bureau (<http://www.census.gov/>). TIGER/Line Shapefiles are an extract of selected geographic and cartographic information from the Census MAF/TIGER database. TIGER/Line Shapefiles are available without charge from the U.S. Census Bureau. To obtain more information about the TIGER/Line Shapefiles go to <http://www.census.gov/geo/www/tiger>. The boundary information in the TIGER/Line Shapefiles are for statistical data collection and tabulation purposes only; their depiction and designation for statistical purposes does not constitute a determination of jurisdictional authority or rights of ownership or entitlement and they are not legal land descriptions. Census TIGER and TIGER/Line are registered trademarks of the U.S. Bureau of the Census.

Finding Spatial or Other Commercial Data from Windows Azure™ Marketplace DataMarket

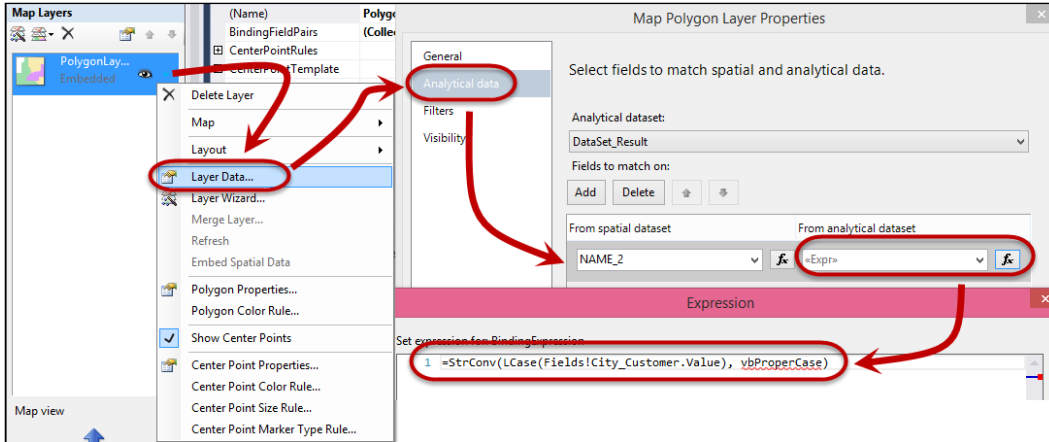
The DataMarket section of Windows Azure Marketplace, formerly known as Codename Dallas, includes data, imagery, and real-time web services from leading commercial data providers and authoritative public data sources. DataMarket includes datasets, imagery, and real-time web services from leading commercial data providers and authoritative public data sources into the Windows® Azure™ Platform. For more information, see <http://www.microsoft.com/windowsazure/sqlazure> and <http://www.microsoft.com/windowsazure/marketplace/default.aspx>.

That is how I was able to find a map of Belgium. You can select the visualization type from the following screen; then select the dataset, match fields and a visualization:





The most important element of this map wizard is linking the geographical data from the map to the geographical data in your dataset. In this example, I used the city names from the dataset to link to the Name_2 field in the ESRI shape file. The problem here is that, in the shape file, the capitalization of the city names does not match the way the names are capitalized in the dataset.

You can fix this after you complete the map wizard by using an expression to produce a better match:



I used an expression to convert the City field from the dataset into the case that is used in the ESRI shape file to get a better link:

```
=StrConv(LCase(Fields!City_Customer.Value), vbProperCase)
```

-  **Selecting an ESRI shape file**
When you select, or create, an ESRI shape file, make sure the size is not too big. The shape file will be imported into the RDLC file and, if it contains too much information, it will take the report viewer more time and resources to render it at runtime.
-  An example of this report is available in the object: Packt - CH04-9.

To display geographical information, I suggest using Power View or Power Map instead of an RDLC report. In *Chapter 9, Power BI*, you will see how to do this and why it has much more potential for these types of reports.

Summary

In this chapter we have seen how data can be visualized to understand and read a report just by looking at it. When you select and implement the appropriate visualization technique then users will not have to spend a lot of time interpreting numerical information but they will be able to see what's going on and get a clearer picture of the KPI in the report. As they say, a picture is worth a thousand words, this is of course also very true in reporting. We have seen how to use images, data bars, and indicators to create KPIs, Sparklines and charts to indicate and spot trends, and used the map to display geographical information.

In the next chapter, I will explore another type of report, the document report.

5

Document Reports

In this chapter, I will introduce one of the most commonly used types of reports: document reports. I will explain how the dataset of this type of report is built in standard Dynamics NAV. Then, I will guide you through the layout of the report and explain the how and why of typical report patterns for documents like the number of copies option and how you can display information in the document header that is linked directly to the record shown in the body of the report.

What is a document?

A typical example of a document report is the **Report 206 Sales - Invoice**. A document report is a report that is printed and mainly used to communicate with third parties. It's the kind of report that you send to your customers or vendors, informing them of a certain transaction that requires their attention. This type of report is also one of the most frequently customized reports in Dynamics NAV, since every company usually applies their own house style on this report.



The **Report 206 Sales - Invoice**, is not used in the North America version of Dynamics NAV; you can import it from the object: Packt - CH05-6

When performing an implementation, make sure you leave enough time to customize the document reports, because depending on the requirements, it can take a lot of time, starting from a couple of hours per report up to one or two days. Some companies, when it comes down to documents such as sales invoices, credit memos, proformas, and shipment or purchase notes, have far more issues than in other types of reports.

You will spend most of the time implementing exceptions or special circumstances that need to be foreseen in the report layout. That is why it is very important to have a clear and full understanding of what is required and to get this agreed.

In real life, I have seen it happen more than once that the developer finishes the development of the report in time, but once the users start testing the report, they notice that certain exceptions were not covered. Because of the complexity of the document report, making changes to it can be time-consuming.

Then, of course, you will also need to guide your customer, because having many exceptions in a process or report is usually an indication that something might be missing in the implemented business processes, and having a very complex document report will not solve those kinds of problems or gaps.

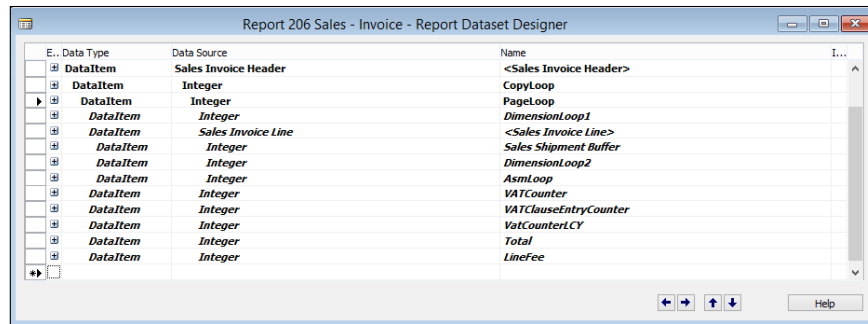
It's important that you manage the expectations with your client and explain that it is very common that document reports take time to develop and usually more than was planned. You will then avoid conflicts.

It's a good idea to spend time creating customizable templates for document reports because document reports are always customized with every new implementation and many of those customizations are repeated every time. You can then spend your development time with advanced measures and add any missing functionality to the standard reports later and reuse it with every implementation. Standard document reports have a very complicated data model, and the layout is also very complex. It could pay off to create a set of templates with simplified datasets, to be used as a starting point for custom developed documents, rather than start with the out-of-the-box overly complex report objects.

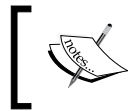
Although this seems like a no-brainer, in real life many companies always try to reinvent the wheel...

The data model

The data model for a typical document report consists of many data items. You will see the `Document Header` and `Document Line` table indented in the following screenshot because a document stores information in a header table that is connected to multiple lines. Because of the workaround to implement the **No Of Copies** option, which I will explain later in this chapter, it also contains two integer tables. Then, depending on the type of document report, you will also find links to other tables like, for example, dimensions for the header, dimensions for the line, VAT information, local and actual currency, shipment information, assembly information, and totaling. An example of the dataset for **Report 206 Sales - Invoice** is shown in the following screenshot:



As you can see, this is a very large and complex dataset. If you open the data items, using the plus sign, you can see the columns. Every column in the dataset designer becomes a column in the runtime dataset and, because of the size, performance needs to be monitored.



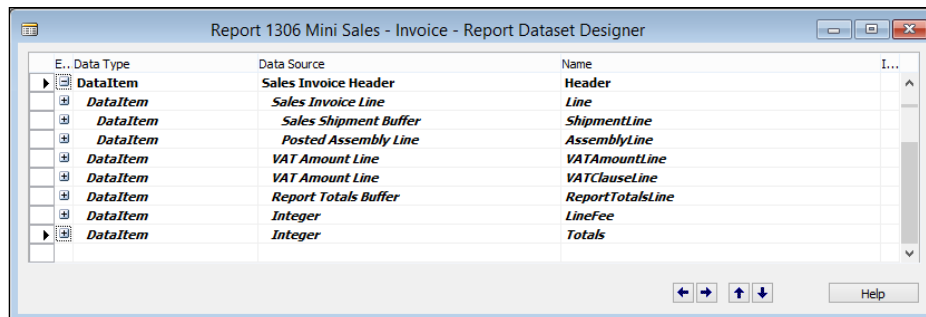
In order to optimize the image to fit on one page, I have collapsed the data items. To see the columns, open **Report 206 Sales - Invoice** and expand the data items.

The dataset of this report is so large because Microsoft has foreseen every possible piece of information in the standard document reports. In real life and most situations this is not needed and you can reduce it. One example is dimensions. The dataset contains the dimensions for the header and for the lines, although, in real life, these are almost never used or required.



This is the dataset of the standard report that comes out-of-the-box in Dynamics NAV. Needless to say, there are a few things you can do to optimize its performance.

Now, if you compare the earlier dataset to the dataset of the **Report 1306 Mini Sales - Invoice**, as shown in the following screenshot, you will notice some differences:



The **Report 1306 Mini Sales - Invoice** gives another way of creating a document report, which I will explain later in this chapter.

The data model of a document report basically consists of two tables, the `Header` and the `Line` table, for example, `Sales Invoice Header` and `Sales Invoice Line`. The lines are indented below the header, because a header can contain multiple lines, and the `No.` of the `Header` table is linked to the `Document No.` of the `Line`. Because a document is linked to many other tables such as, for example, dimensions, shipments, and VAT, these other tables are added as integer data items. Then, using C/AL code, records are added to these integer data items using buffer or temporary tables. The following screenshot shows how the shipment information is collected via C/AL code and added to the `Sales Shipment Buffer` table:

E.. Data Type	Data Source	Name
Column	FIELDCAPTION("VAT Identifier")	VATIdentifier_SalesInvLineCap.. ^
DataItem		Sales Shipment Buffer
Column	FORMAT(SalesShipmentBuffer."Posting Date")	SalesShptBufferPostDate
Column	SalesShipmentBuffer."Quantity"	SalesShptBufferQty
Column	ShipmentCaptionLbl	ShipmentCaption

```

Sales Shipment Buffer - OnPreDataItem()
SalesShipmentBuffer.SETRANGE("Document No.,"Sales Invoice Line"."Document No.");
SalesShipmentBuffer.SETRANGE("Line No.,"Sales Invoice Line"."Line No.");

SETRANGE(Number,1,SalesShipmentBuffer.COUNT);

Sales Shipment Buffer - OnAfterGetRecord()
IF Number = 1 THEN
    SalesShipmentBuffer.FIND('-')
ELSE
    SalesShipmentBuffer.NEXT;

Sales Shipment Buffer - OnPostDataItem()
    
```

The pattern that is used here with an integer (or buffer) table as a data item will be covered in more detail in *Chapter 7, Performance Optimization Techniques*.



Buffer tables are used all over Dynamics NAV and also in reports. The advantage of using buffer tables is performance and flexibility. In the development environment, when you apply the `@*Buffer*` filter to the name field of tables, you will see a list of about 70 tables. Although you can use any table as a buffer table, by making it temporary, these 70 buffer tables contain interesting business logic built in as functions.

To summarize, the data model of document reports is complex, to say the least, and contains a mix of normal and temporary tables. Depending on your location (NA, W1, BE ...) the data model of these reports might differ. For example, in North America there's no VAT, as opposed to European countries such as Belgium and Germany, where their VAT rules are sometimes very challenging.

In the following sections, I will go into detail about specific parts of the data model and explain the how and why of each type of pattern.



When I use the word pattern, I don't mean it's a design pattern, such as in the object-oriented world. What I mean is that the way that it's implemented can and is reused in many other reports, and it can be seen as a building block of a typical document report.

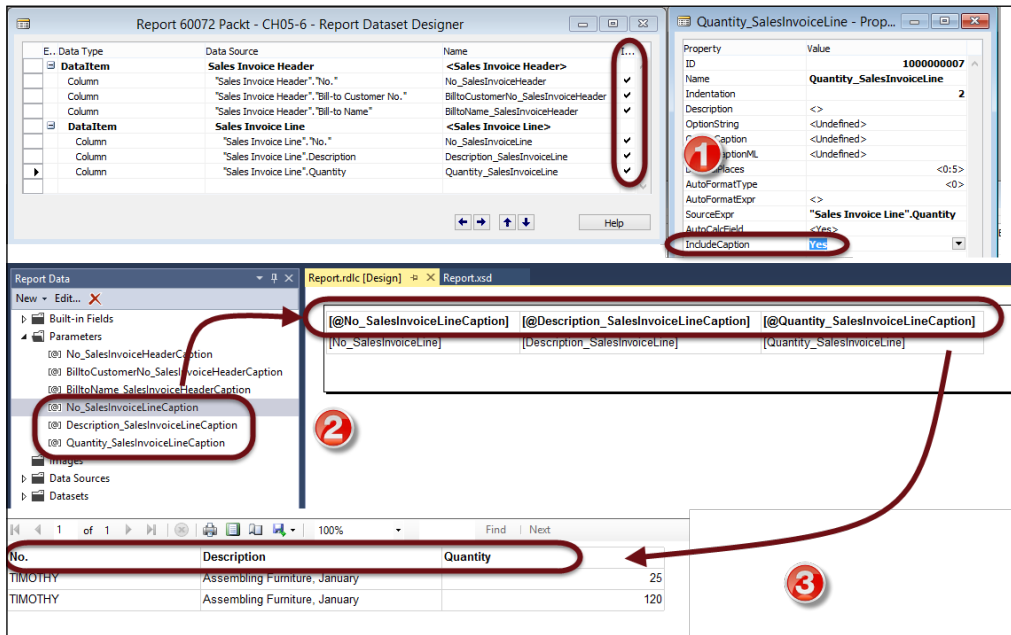
Implementing multilanguage

A document report is exchanged or sent to multiple third parties, so it should be a multilanguage document. Let's investigate how you can make a report multilanguage enabled.

Imagine that we have a report with the following dataset:

E.. Data Type	Data Source	Name	I...
DataItem	Sales Invoice Header	<Sales Invoice Header>	
Column	"Sales Invoice Header".No."	No_SalesInvoiceHeader	
Column	"Sales Invoice Header".Bill-to Customer No."	BilltoCustomerNo_SalesInvoiceHeader	
Column	"Sales Invoice Header".Bill-to Name"	BilltoName_SalesInvoiceHeader	
DataItem	Sales Invoice Line	<Sales Invoice Line>	
Column	"Sales Invoice Line".No."	No_SalesInvoiceLine	
Column	"Sales Invoice Line".Description	Description_SalesInvoiceLine	
Column	"Sales Invoice Line".Quantity	Quantity_SalesInvoiceLine	

The report contains information from the sales header and line table. In order to make this report multilanguage you usually enable the `IncludeCaption` property of every field. Because of this, a caption will be sent to the layout as a parameter, and you can display the column labels in the language of the user. The following screenshot illustrates this concept:



You now have a multilanguage report with captions and labels. The printed language is the language of the user running the application.


The problem with a document, however, is that it is typically printed and sent to a recipient, and the document has to be generated in the language of the recipient.

Until now, we have always used captions (`IncludeCaption`) and labels to implement multilanguage functionality in reports. The problem is that, when you use captions and labels, they will be generated in the language of the user who is running the report, and not the language of the recipient. This is a fundamental difference.

Another problem is that, when printing multiple documents at once, the different documents each need to be printed in the language of the recipient. For example, when you run the **Sales Invoice** report for multiple customers, each customer might use a different language.

So, one report run should generate different labels in one dataset. This means that captions and labels are not a solution, because they end up in the parameters in the layout, and the values of the parameters are the same for every record in the dataset.


What we need are captions in the language of the recipient, and that is why you need to add the captions to the dataset of the report. For every document that is printed (or record in the dataset), you can then decide what the language should be.

 This technique adds extra fields (captions) to the report dataset, which decreases report execution performance. So, make sure you only do this for the captions that are actually used, and not for all fields.

What you need to do is to add the captions to the dataset using the `FIELDCAPTION` function, as shown in the following example:


E.. Data Type	Data Source	Name	I...
DataItem	Sales Invoice Header	<Sales Invoice Header>	
Column	"Sales Invoice Header"."No."	No_SalesInvoiceHeader	
Column	"Sales Invoice Header"."Bill-to Customer No."	BilltoCustomerNo_SalesInvoiceHeader	
Column	"Sales Invoice Header"."Bill-to Name"	BilltoName_SalesInvoiceHeader	
DataItem	Sales Invoice Line	<Sales Invoice Line>	
Column	"Sales Invoice Line"."No."	No_SalesInvoiceLine	
Column	"Sales Invoice Line".Description	Description_SalesInvoiceLine	
Column	"Sales Invoice Line".Quantity	Quantity_SalesInvoiceLine	
Column	"Sales Invoice Line".FIELDCAPTION("No.")	No_SalesInvoiceLineCptn	
Column	"Sales Invoice Line".FIELDCAPTION(Description)	Description_SalesInvoiceLineCptn	
Column	"Sales Invoice Line".FIELDCAPTION(Quantity)	Quantity_SalesInvoiceLineCptn	

Instead of using the `FIELDCAPTION` function, you can use text constants that you define in **Globals**, and add them to the dataset.


 Text constants in document reports usually get an `lbl` suffix in their definition and a `Cptn` or `Caption` suffix in their name in the dataset, as you can see in the preceding screenshot.

Then, in the `OnAfterGetRecord()` trigger of the `Document Header`, you write the following code:

```
CurrReport.LANGUAGE := Language.GetLanguageID("Language Code");
```

 This code uses `Language Code` in the document header to set the language of the report. The language code in the document header is inherited from the customer when it is added to the document.

When a new document is fetched, then the language of the fields added to the dataset will change accordingly.

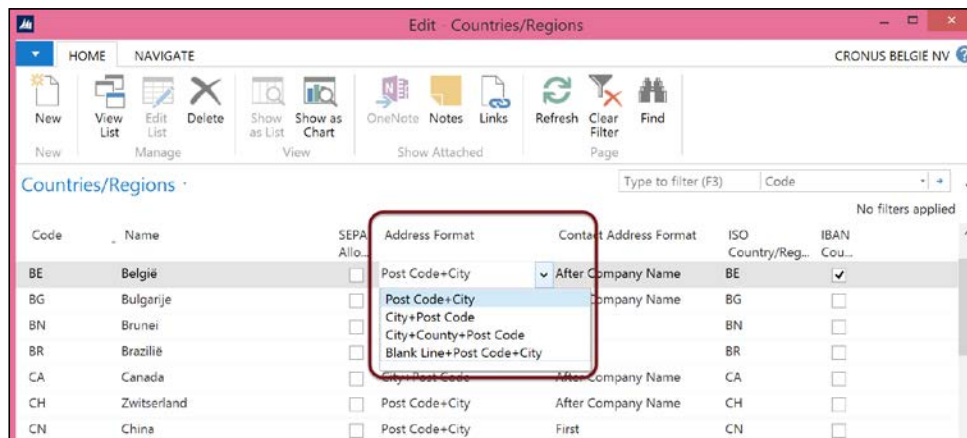
 The choice between using a text constant and the FIELDCAPTION function is obvious. When you add a caption for a field from a table, use the FIELDCAPTION function. The FIELDCAPTION function fetches the captions that are defined for the field in the table designer. If you want to add text to a document, which is not bound to a field in a table, then use a text constant.

An example of the report with the FIELDCAPTION function and the corresponding layout is available in the object: Packt - CH05-7.

Address formatting

Documents also contain address fields, for example the Bill-To and Ship-To addresses. The way that you add address fields in the dataset is an example of a report pattern. Dynamics NAV provides functionality to format an address across an array of eight textboxes, and this functionality is used in all standard document reports.

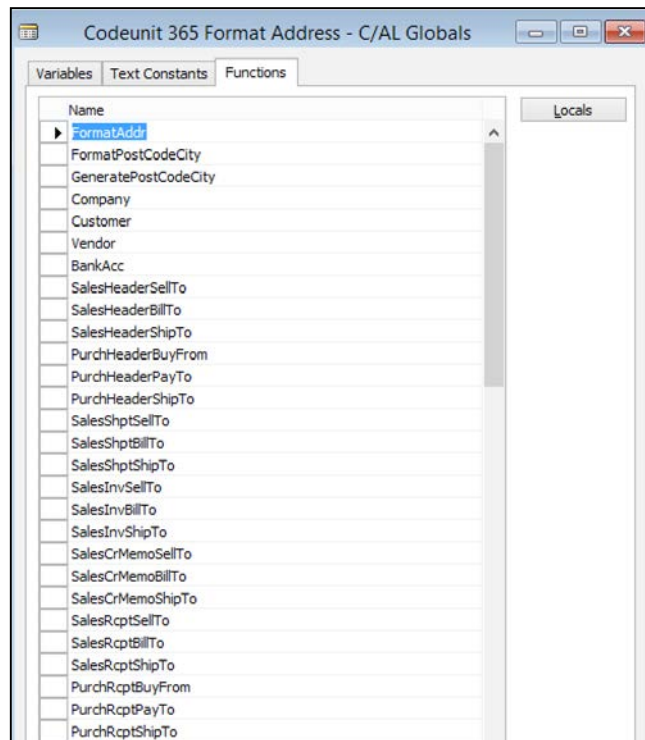
First, you need to know that the setup of Dynamics NAV allows you to define how addresses need to be formatted. You can do this in the **Countries** page, as shown in the following screenshot:



Here, you can decide, for example, if **Post Code** needs to be in front of **City** or not, and if **Contact** needs to be before or after **Company Name**, and so on. When printing an address on a document, you need to follow this protocol.

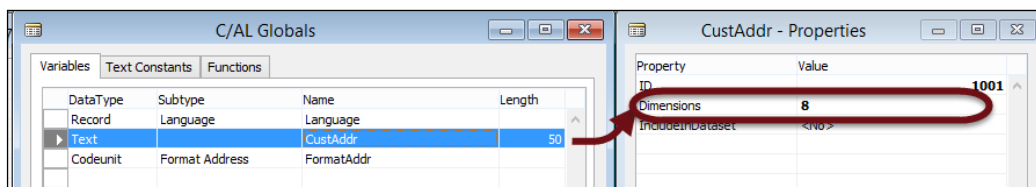
An address in Dynamics NAV can consist of eight fields. We will use one variable that holds all eight fields to put them in the dataset. This variable will be defined as an array with eight elements and, via C/AL code, we will then populate the array with the address fields, in the order defined in the setup.

Since the logic is the same for all reports (and pages), it is contained in a code unit: **Format Address**. This code unit contains several functions, as you can see in the following screenshot:

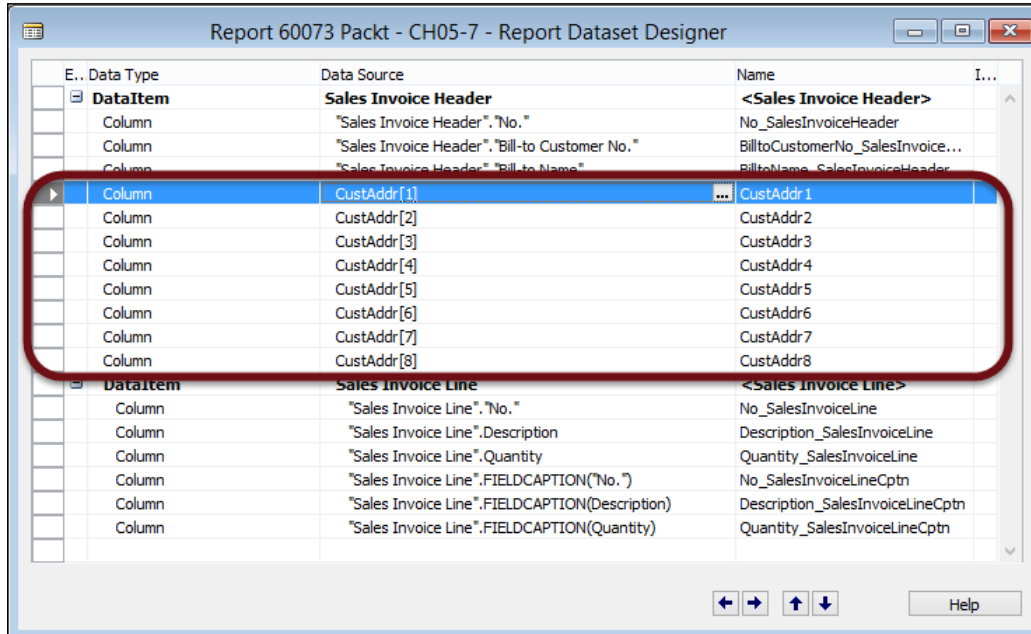


To get started, import object: Packt - CH05-7. This report contains a simple dataset with fields from the Sales Invoice Header and Line table.

Define an address array in the **C/AL Globals** of the report, as follows:



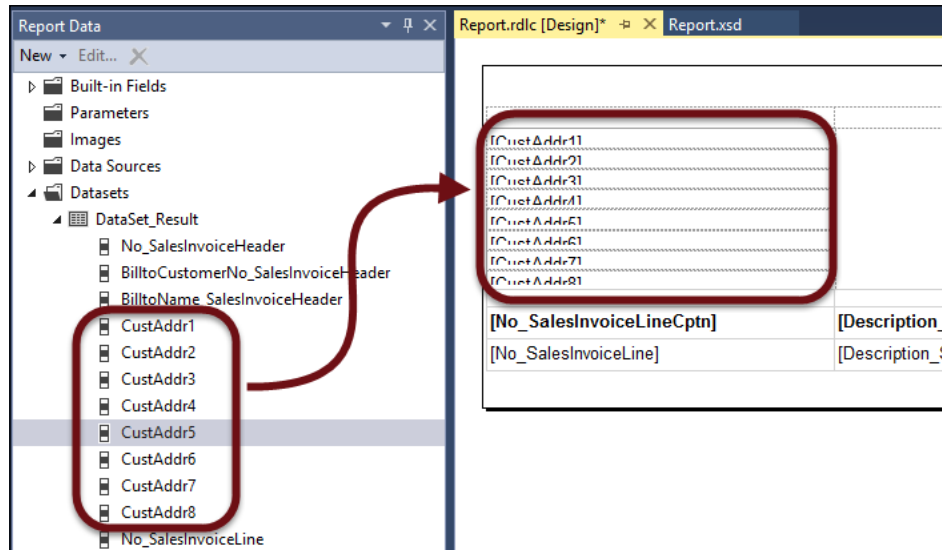
Add each individual field to the dataset as a separate column, using the array notation:




You can use the code unit to populate the address fields in the array in the C/AL code of the report:

```
Sales Invoice Header - OnAfterGetRecord()  
CurrReport.LANGUAGE := Language.GetLanguageID("Language Code");  
FormatAddr.SalesInvBillTo(CustAddr,"Sales Invoice Header");
```

Then, you can add the fields to the report layout:



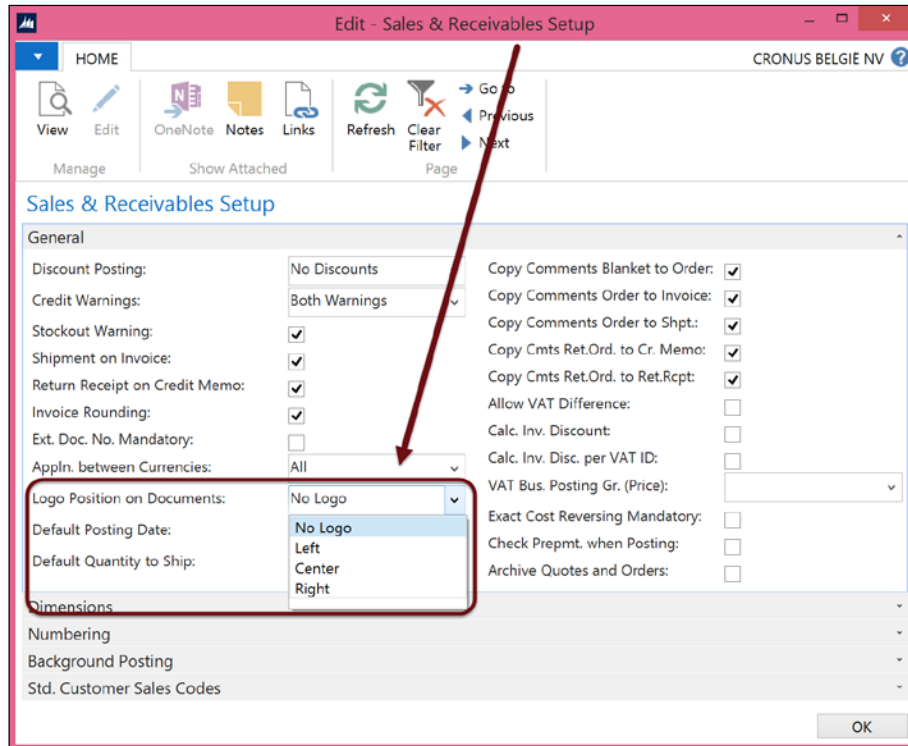
 An example of this report is available in the object: Packt - CH05-8

If you are asked to modify the order in which address fields should be displayed, then you can add a custom function to the **Format Address** code unit and use that in your document reports, without having to modify the report layout.

Other examples of how to use the functions for other addresses are available in **Report 206, Sales Invoice**, **208 Sales Shipment**, and many other reports.

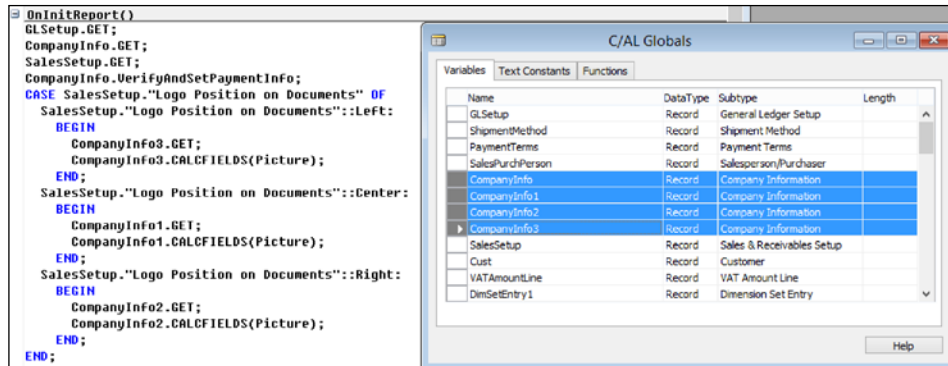
Including logos

In a document report, as in other reports that you exchange with third parties, you typically want to add your company logo and information. You can define how the logo should be positioned in all documents in Dynamics NAV. This can be done in the **Sales & Receivables Setup** window:

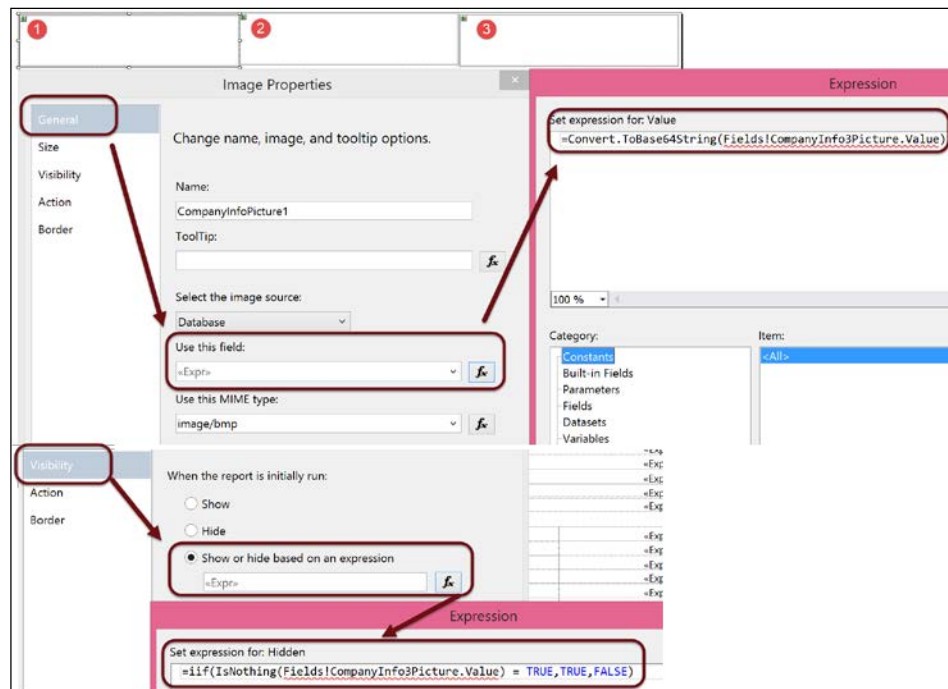


Depending on what you select, a logo will be shown left, center, right, or it is not shown at all.

Four different variables that point to the company information table are defined in the report because you can set up the image to be displayed in different ways. This is the table that holds the logo and company name and address fields, and then there's the following code in the OnInitReport () trigger:



The code in the CALCFIELDS function calculates the image depending on the value of the Logo Position on Documents field in one of the company info variables. Then, in the layout of the report, in the header, there are three image controls, as shown in the following screenshot:



Each of the image controls is bound to one of the company information fields from the dataset and hidden, depending on the value of the `Picture` field.



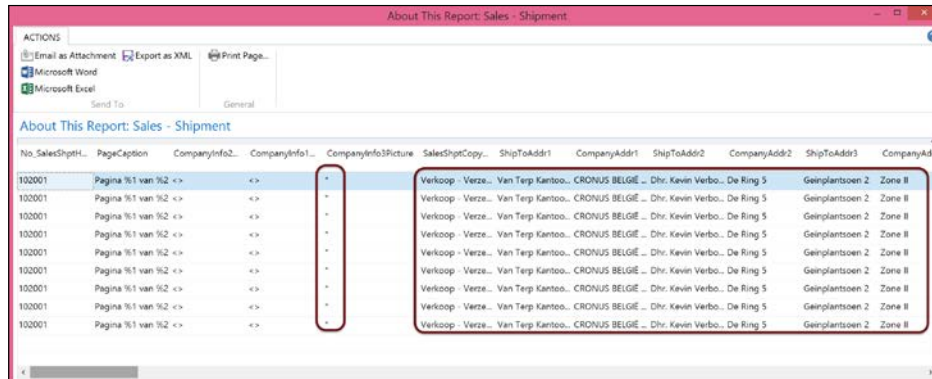
In the expression for the image, the `Convert.FromBase64String()` function is used. This is not required. It is there because, in Dynamics NAV 2009, the picture was stored in the body of the report in a hidden textbox and the BLOB field from the dataset was converted to a string with the `Convert.ToBase64String()` function. As you can see, the expression here is `=Convert.ToBase64String(Fields!CompanyInfo3Picture.Value)`, which means that the picture is from the `Fields` collection (the dataset). You could replace this with `Fields!CompanyInfo3Picture.Value` and the report would still display the image. The reason the `Convert.ToBase64String` function is there is probably because it was upgraded from a previous version.

This is overkill and I would do this using only one variable, instead of four, and I would show or hide the image based on the value of the `Logo Position on Documents` field, instead of the picture field. The dataset would then contain fewer columns. The logo field is added to the dataset as a member of the `Document Header` table because it's repeated on all rows of the dataset. This pattern can slow down performance dramatically because an `image` field, which has a BLOB data type, can potentially hold up to 2 GB of information. I would add the image fields on a separate integer record and so only add them in one row (the first or last) of the dataset.



I once had the experience of setting up Dynamics NAV using the standard **Sales Invoice** report, as described in the preceding section. The company had a logo with a size of about 500 KB. The report took more than a minute to run for a single invoice because this image was repeated on every row of the dataset. I modified the dataset and moved the image to an integer table, so that it was only added in one row, the last one, and after that the report took only a couple of seconds to process.

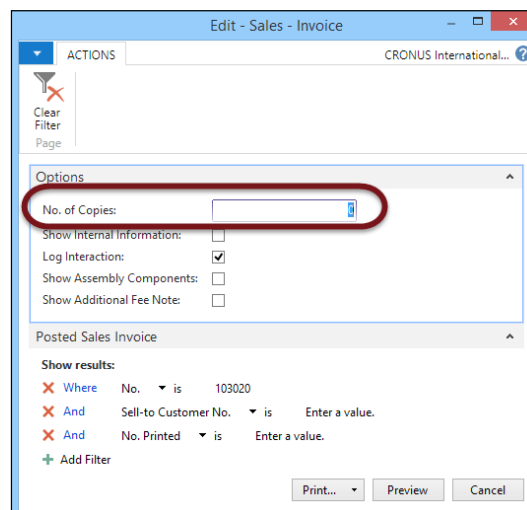
Instead of moving the company picture to another data item, if you find it too cumbersome, an alternative is to clear it after the first record is added to the dataset, using the `CLEAR ()` function. Otherwise, as you can see in the following screenshot, the image will be repeated on every row:



This is not only valid for the company logo, but for all fields that have the same constant value in every record of the dataset. You should only add them to the dataset once. More information about performance optimization techniques, including this one, is available in *Chapter 7, Performance Optimization Techniques*.

The No. of Copies option

When you print a document report in Dynamics NAV, one of the features that has always been available is the **No. of Copies** option on the request page:



You can use it to print one copy, have extra copies printed, or multiple invoices. Most printers also have this option in the printer settings, but when printing an invoice you also require the copies to have a different header. In some countries, such as Belgium, you can only legally print an invoice once. If you need to print a copy, then it should be mentioned on the invoice. The printer settings are then not sufficient and you will need a way to control it from within the report object.

What we need, to be able to do this, is to duplicate the dataset for every copy. If you require two copies, for example, then you need to loop three times over the data items in the report, once for the original and twice for the copies.



There are different ways to implement this feature, and in this chapter, I will explain how it is done in standard document reports. This feature has been available in many versions, back to the days when we used the classic report designer. When the Dynamics NAV team decided to switch over to RDLC reporting, this pattern was simply reused, to make the upgrade process easier.

So, how can you loop multiple times over a data item, and its indented data items?

First, we need to create a dataset containing two data items, a Document Header and a Document Line table, similar to a document report:

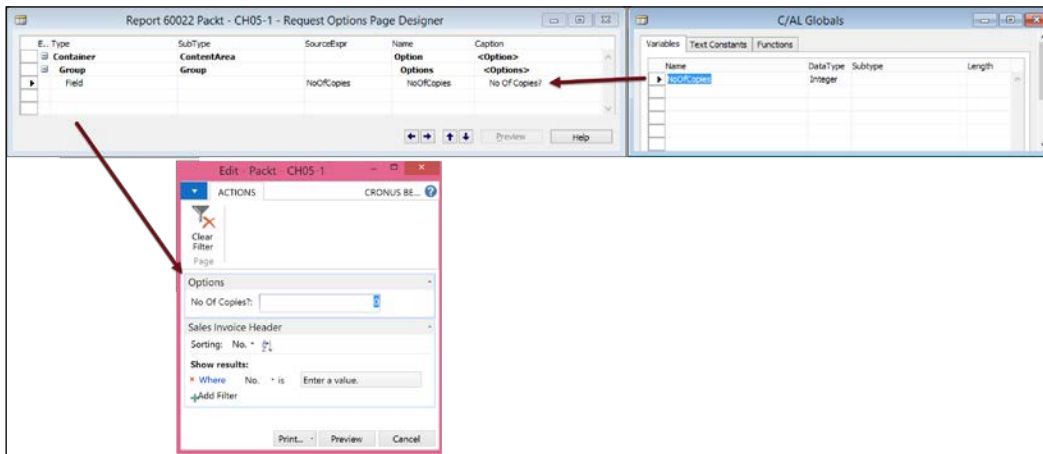
The screenshot shows the 'Report Dataset Designer' window for 'Report 60022 Packt - CH05-1'. It displays two data items: 'Sales Invoice Header' and 'Sales Invoice Line'. The 'Sales Invoice Header' data item has three columns: 'No.', 'Bill-to Customer No.', and 'Bill-to Name'. The 'Sales Invoice Line' data item has three columns: 'No.', 'Description', and 'Quantity'. Below the designer is a preview window titled 'About This Report: Packt - CH05-1' showing a table with two rows of data. The first row is highlighted with a red box.

No_SalesInvoic...	BilltoCustomer...	BilltoName_Sal...	No_SalesI...	Description_Sal...	Quantity_SalesI...	Quantity_SalesI...
103015	10000	Van Terp Kantoo...	1968 S	MEXICO Draaisto...	5	#,##0.#####
103015	10000	Van Terp Kantoo...	1996 S	ATLANTA White...	7	#,##0.#####

In the preceding example, the invoice document has two lines, and they are sent to the dataset.

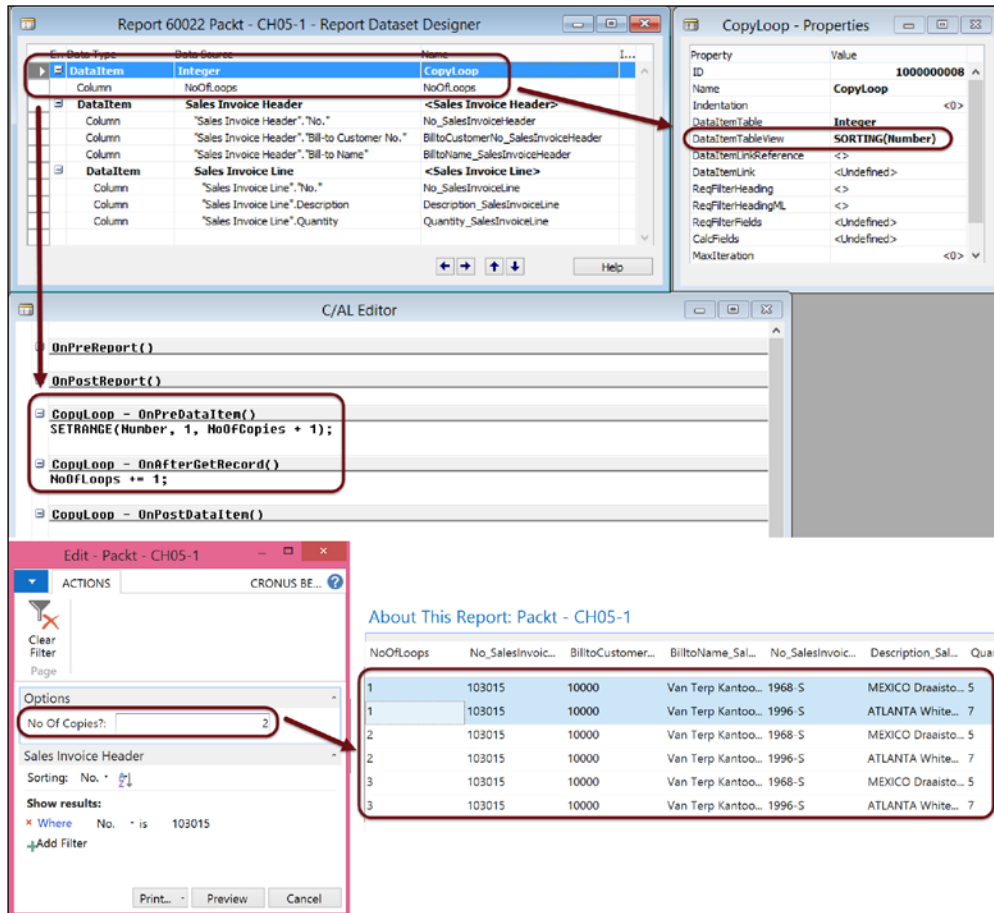
[ To get started you can import the object: Packt - CH05-9]

Now we need to implement a way for the user to ask for copies. In order to do that, I will create a request page with a **No. Of Copies** field:




Now, once the user enters a value, we need to figure out a way to loop $\text{NoOfCopies} + 1$ over our data items. The reason I'm adding 1 is that you also need to print the original. The number of loops is the original plus the number of copies.

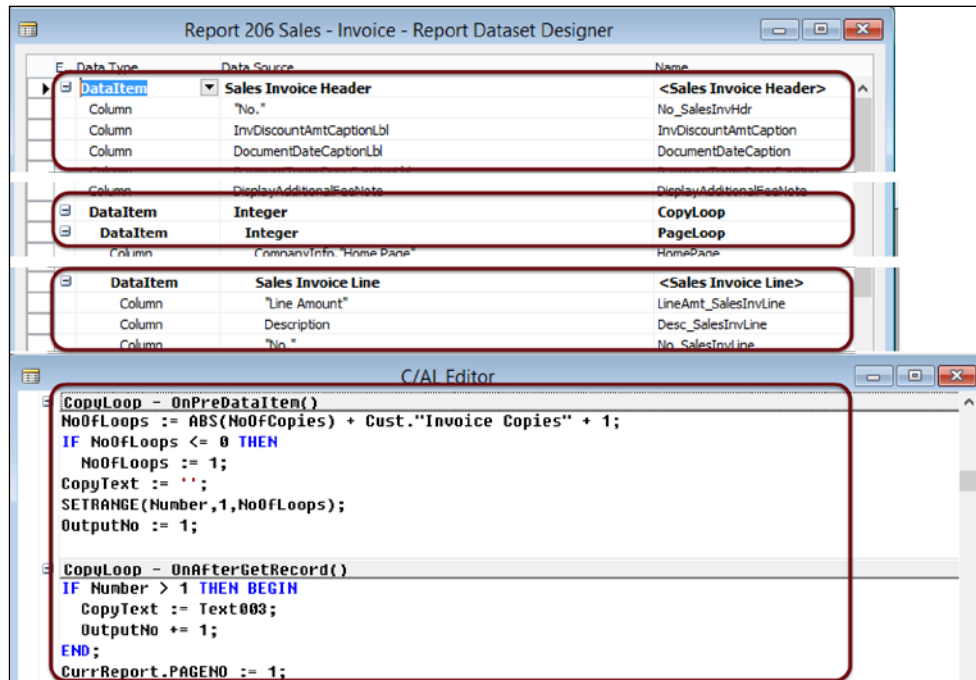
As you might have guessed, I'm going to use an integer data item to do this. I will add it as the first data item, and then indent the other existing data items below it, as in the following example:



I have named the Integer table CopyLoop and, in the OnPreDataItem() trigger, I have used the value that the user enters as NoOfCopies to filter the data item, and so, looped multiple times over the underlying data items, including the Document Header and Document Line tables. In the OnAfterGetRecord() trigger of the CopyLoop, I have incremented an integer NoOfLoops to count the iterations, which I will add to the dataset so that you can see at runtime which are the original lines, with NoOfLoops set to 1, and which are the copies, with NoOfLoops > 1.

 An example of this report is available in the object: Packt - CH05-1

In the actual document reports, for example **Report 206 Sales - Invoice**, Microsoft has implemented this a bit differently, as you can see in the following screenshot:



The screenshot shows two windows. The top window is 'Report 206 Sales - Invoice - Report Dataset Designer'. It displays a table of data items and columns. The bottom window is 'C/AL Editor', showing code for the CopyLoop and PageLoop.

Data Type	Data Source	Name
DataItem	Sales Invoice Header	<Sales Invoice Header>
Column	No.	No_SalesInvHdr
Column	InvDiscountAmtCaptionLbl	InvDiscountAmtCaption
Column	DocumentDateCaptionLbl	DocumentDateCaption
Column	DisplayAdditionalFeeNote	DisplayAdditionalFeeNote
DataItem	Integer	CopyLoop
DataItem	Integer	PageLoop
Column	HomePane	HomePane
DataItem	Sales Invoice Line	<Sales Invoice Line>
Column	Line Amount	LineAmt_SalesInvLine
Column	Description	Desc_SalesInvLine
Column	No.	No_SalesInvLine


```

CopyLoop - OnPreDataItem()
NoOfLoops := ABS(NoOfCopies) + Cust."Invoice Copies" + 1;
IF NoOfLoops <= 0 THEN
    NoOfLoops := 1;
CopyText := '';
SETRANGE(Number,1,NoOfLoops);
OutputNo := 1;

CopyLoop - OnAfterGetRecord()
IF Number > 1 THEN BEGIN
    CopyText := Text003;
    OutputNo += 1;
END;
CurrReport.PAGEN0 := 1;

```

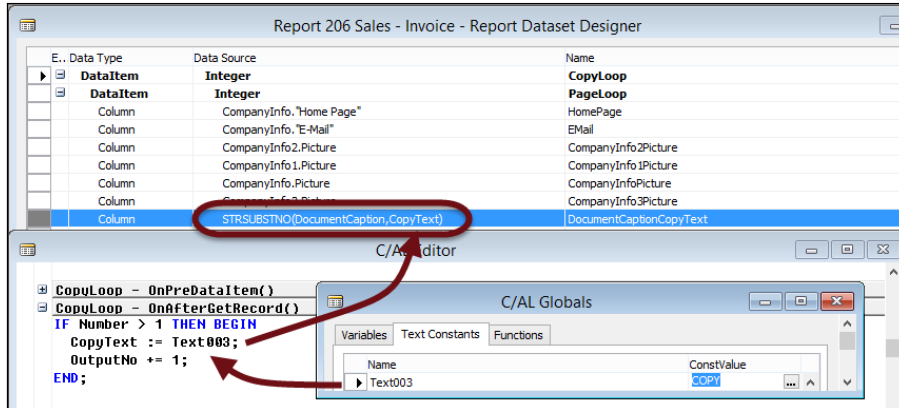
Instead of placing CopyLoop as the first data item, they have put it in between the header and the line, and they have also included a second integer named PageLoop. This PageLoop dates back from the classic document reports in earlier versions and has no added value at all here.

 Also, in the C/AL code the CurrReport . PAGENO is no longer valid in the C/AL code in RDLC because page numbers are calculated in the layout and this function has no effect at all.

You can also define an extra number of copies per customer because it is on the customer card in Dynamics NAV and it is also added to NoOfCopies.

Document Reports

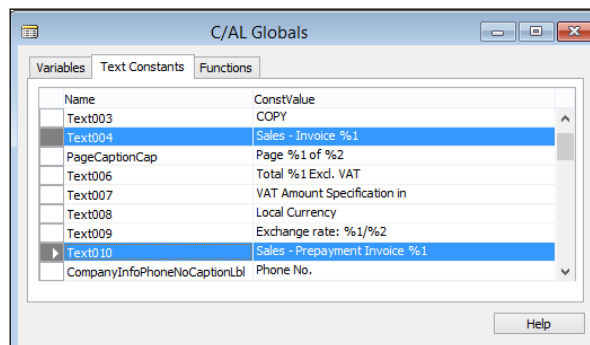
There is a variable, `CopyText`, which is reset for every new document, and contains the text `COPY` for every copy. This text is added to the layout, on the header, to indicate that it is a copy and not the original. This is managed via the function `DocumentCaption()`, as you can see in the following screenshot:



The `DocumentCaption()` function does the following:

```
LOCAL DocumentCaption() : Text[250]
IF "Sales Invoice Header"."Prepayment Invoice" THEN
    EXIT(Text010);
EXIT(Text004);
```

Depending on whether it is a normal or a prepayment invoice, it returns a `Text010` or `Text004` string:



The placeholder in this text string (`%1`) is replaced with what is in the `CopyText` variable.

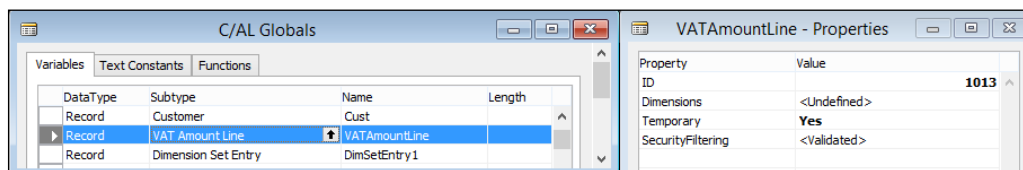
Totaling and VAT

Many years ago, when I was learning about and exploring reports in Dynamics NAV, I was a bit surprised about the way that VAT is calculated in the **Sales Invoice** report. At the time I wasn't familiar with the data model of the application and I expected the VAT information to come from a table where it was simply copied over to the dataset of the report. But this is not the case. In the report, VAT is calculated when the invoice lines are processed.



In the US, there's no VAT and so you will not see this in the US localization. But the way it is done might still be interesting because this is an example of using a buffer table in a report. To follow this section please open (or import) **Report 206 Sales - Invoice**.

The way that this is done is via a temporary (or buffer) table VATAmountLine. The table is defined as a temporary table in the **Globals** of the report:



The idea is to look for all of the VAT lines for the current invoice and store them in this temporary table when processing the sales invoice lines. Then, later in the report, an integer data item is used to loop over all the records stored in this temporary table and add them to the dataset.

It begins with a reset or by clearing the variable, in the `OnPreDataItem()` trigger of the sales invoice line. Then, while the sales lines are processed in the `OnAfterGetRecord()` trigger, the VAT information from the sales lines is added to the `VATAmountLine` table, as you can see in the following screenshot:

```
Sales Invoice Line - OnPreDataItem()
VATAmountLine.DELETEALL;
SalesShipmentBuffer.RESET;
SalesShipmentBuffer.DELETEALL;
FirstValueEntryNo := 0;
MoreLines := FIND('*');
WHILE MoreLines AND (Description = '') AND ('No.' = '') AND (Quantity = 0) AND (Amount = 0) DO
  MoreLines := NEXT(-1) <> 0;
IF NOT MoreLines THEN
  CurrReport.BREAK;
SETRANGE('Line No.',0,'Line No.');
```

```
Sales Invoice Line - OnAfterGetRecord()
PostedShipmentDate := 0D;
IF Quantity <> 0 THEN
  PostedShipmentDate := FindPostedShipmentDate;

IF (Type = Type::'G/L Account') AND (NOT ShowInternalInfo) THEN
  "No." := '';
```

```
VATAmountLine.INIT;
VATAmountLine."UAT Identifier" := "UAT Identifier";
VATAmountLine."UAT Calculation Type" := "UAT Calculation Type";
VATAmountLine."Tax Group Code" := "Tax Group Code";
VATAmountLine."UAT %" := "UAT %";
VATAmountLine."UAT Base" := Amount;
VATAmountLine."Amount Including VAT" := "Amount Including VAT";
VATAmountLine."Line Amount" := "Line Amount";
IF "Allow Invoice Disc." THEN
  VATAmountLine."Inv. Disc. Base Amount" := "Line Amount";
VATAmountLine."Invoice Discount Amount" := "Inv. Discount Amount";
VATAmountLine."UAT Clause Code" := "UAT Clause Code";
VATAmountLine.InsertLine;
```

```
TotalSubTotal += "Line Amount";
TotalInvDiscAmount -= "Inv. Discount Amount";
TotalAmount += Amount;
TotalAmountVAT += "Amount Including VAT" - Amount;
TotalAmountInclVAT += "Amount Including VAT";
TotalPaymentDiscOnVAT += -( "Line Amount" - "Inv. Discount Amount" - "Amount Including VAT");
```

```
Sales Invoice Line - OnPostDataItem()
```

The Total fields (`TotalAmount`, `TotalAmountVAT`, and so on) are incremented after every Invoice Line, and they are added to the sales invoice line data item. This means that the actual total is only available in the last sales invoice line record in the dataset, and should be retrieved as such.



You might wonder why this totaling happens here in the dataset and not in the RDLC layout. You could create columns for these fields in the RDLC layout and let RDLC total the numbers using `Sum()` expressions. Although this would probably work, it's better to do this in C/AL. The layout, and the expressions in the layout, are processed in the Report Viewer application, which runs on every client. Calculating these totals in the layout would mean running it on every client, and for every copy (`NoOfCopies`). That would consume a lot of memory and CPU. Doing the calculations in C/AL means that it will run on the server (Service Tier), which is faster and consumes less resources. Furthermore, in Dynamics NAV 2015, there's a new buffer table, `Report Totals Buffer`, which, when used as a temporary table, improves and simplifies this process even more. It is not used in this report, but it is used in **Report 1306 Mini Sales - Invoice**. This is explained in *Chapter 7, Performance Optimization Techniques*, in the section about using a buffer table.

The information from the sales invoice line and the information from the VAT amount line data items is in different rows in the dataset:

About This Report: Sales - Invoice

Desc_SalesInvLine	No_SalesInvLine	Qty_SalesInvLi...	Qty_SalesInvLi...	UOM_SalesInv...	UnitPrice
<>	<>	<>	<>	<>	<>
MEXICO Swivel Chair, black	1968-S	4	###0.####	Piece	178,74
ROME Guest Chair, green	1960-S	7	###0.####	Piece	181,357
INNSBRUCK Storage Unit/W.Door	1976-W	5	###0.####	Piece	371,267
Glass Door	70011	1	###0.####	Piece	104,813
<>	<>	<>	<>	<>	<>

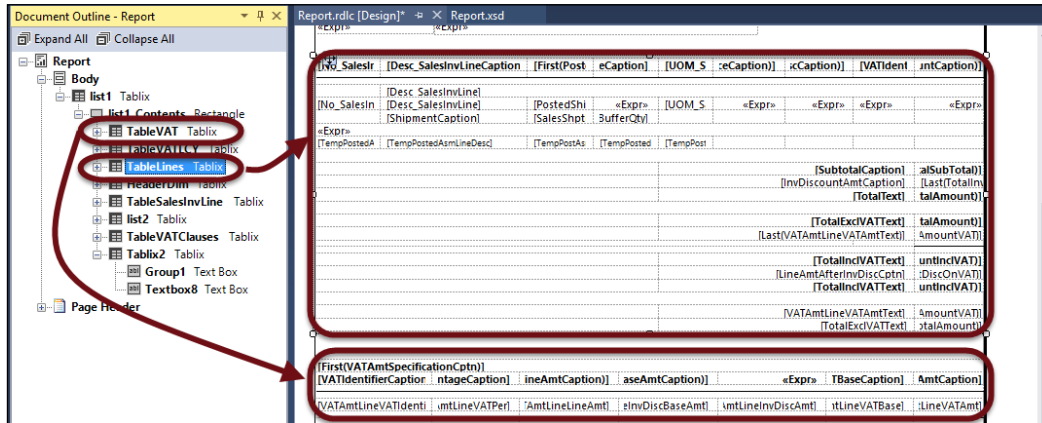
About This Report: Sales - Invoice

VATAmtLineV...	VATAmtLineV...	VATAmtLineV...	VATAmtLineV...	VATAmtLineLi...	VATAmtLineLi...	VATAmt
<>	<>	<>	<>	<>	<>	<>
<>	<>	<>	<>	<>	<>	<>
<>	<>	<>	<>	<>	<>	<>
<>	<>	<>	<>	<>	<>	<>
<>	<>	<>	<>	<>	<>	<>
744,29	###0.00	0	###0.00	3744,29	###0.00	3744,29

Document Reports

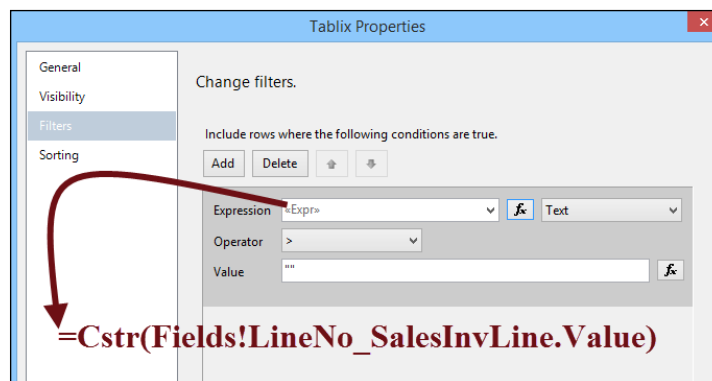
You can see the dataset in the preceding screenshot. In the top part, I selected the rows that contain the sales invoice line information. In the bottom part, I selected the line that contains the VAT amount line information.

There's one table (`TableLines`) in the layout that needs to display the information from the sales invoice lines, and there's another table (`TableVAT`) that displays the records from the VAT amount line table:

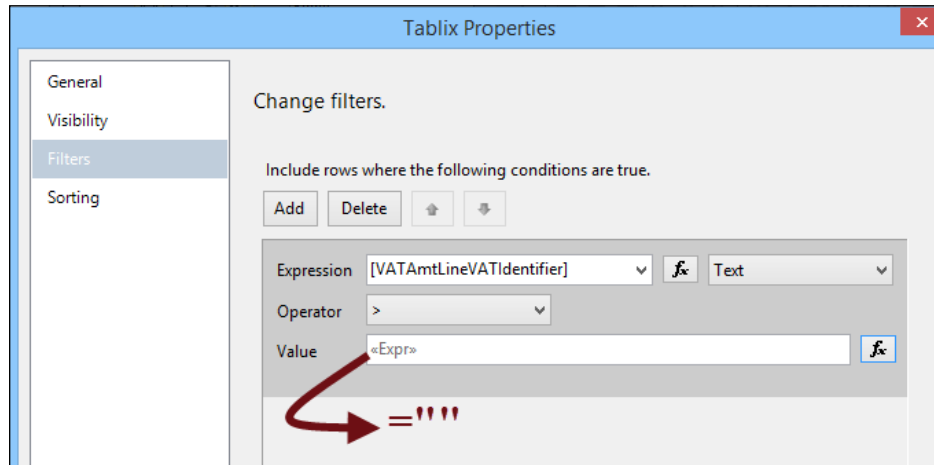


You need to apply filters in the layout because the dataset is sent as a whole, one flat dataset, to the layout. The `TableLines` Tablix needs to apply a filter to the dataset to filter out those lines that contain the sales invoice line records and the `VATTable` Tablix needs to have a filter so that it filters out the lines from the VAT amount line records.

The following screenshot shows the filter that is applied in the `TableLines` Tablix:



The following screenshot shows the filter that is applied in the `TableVAT` Tablix:



This is a typical example of how you can filter a Tablix to select a subset of records from a dataset that contains rows from multiple data items. It is also an example of how you can use a buffer (temporary) table in a report.

The VAT is put in the dataset in a separate data item that loops over the `VATAmountLine` buffer table. The `VATAmtLineVATIdentifier` field is then used in the report layout, in a Tablix, to filter out and display the VAT lines.

The Tablix that displays the VAT has an expression in its hidden property to show or hide the Tablix because VAT is not required for certain customers or invoices:

```
=(Fields!VATAmtLineVATIdentifier.Value = "")
```



There is a similar pattern used to add VAT clauses, and the VAT in local currency (LCY), to the dataset and layout.

Logging and No. Printed

Whenever an invoice is sent to a customer, Dynamics NAV logs this as an interaction with the customer or related contact. At the end of the sales invoice header `OnAfterGetRecord()` trigger, there's code that does this, if the option `LogInteraction` is enabled. This option can be set in the request page of the report.

Every time you print an invoice, it is also logged. There's a field in the Sales Invoice Header table, as in most document header tables, No. Printed, that is incremented, using the following code:

```
CopyLoop - OnPostDataItem()
IF NOT CurrReport.PREVIEW THEN
    SalesInvCountPrinted.RUN("Sales Invoice Header");


Codeunit 315 Sales Inv.-Printed - AL Editor

Documentation()
OnRun(VAR Rec : Record "Sales Invoice Header")
    FIND;
    "No. Printed" := "No. Printed" + 1;
    MODIFY;
    COMMIT;
```

When you open the sales invoice list page, you can see it, as shown in the following screenshot:

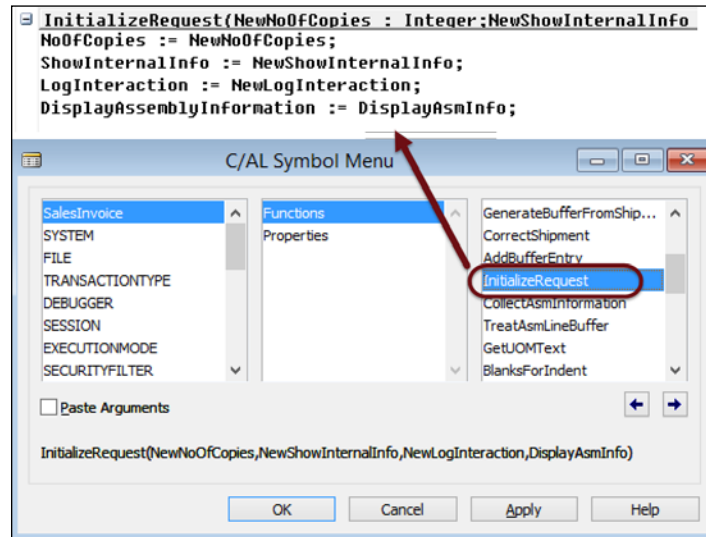
No.	Sell-to Customer...	Sell-to Customer Name	Currency Code	Amount	Amount Including VAT	Location Code	No. Printed
103001	10000	Van Terp Kantoorinrichting		11 433,25	13 786,21	BLAUW	0
103002	20000	Anton Geestig Adviezen		9 741,71	11 787,47		0
103003	30000	Koopmans Verzekeringen B.V.		8 383,00	10 143,43		0
103005	49525252	Beef House		1 549,01	1 549,01	GROEN	0
103006	49525252	Beef House		7 745,02	7 745,02	GROEN	0

Now, if you look closely at the code, the counter is not incremented if you preview the report, which is logical. You can use the function `currReport.PREVIEW` to detect if a report has been previewed, printed or exported.

 If you use the `currReport.PREVIEW` function in a report, then the print button is removed from the report viewer toolbar. Otherwise, you would be able to print a report without it being logged.

InitializeRequest

Reports are not always run directly. What I mean is that you can run a report via C/AL code, without any user interaction. In that case, you might want to set some variables, which are shown in the request page. In order to do that, you have to create a non-local function to access those variables. In most document reports there's a function `InitializeRequest`, to do just that:



As you can see in the preceding example, when you create a variable of type report and subtype 206, then you can access the `InitializeRequest` function to set `NoOfCopies` and other variables.



This is a report design pattern that you can apply every time you want to access, get, or set values of report variables, outside of the report object.

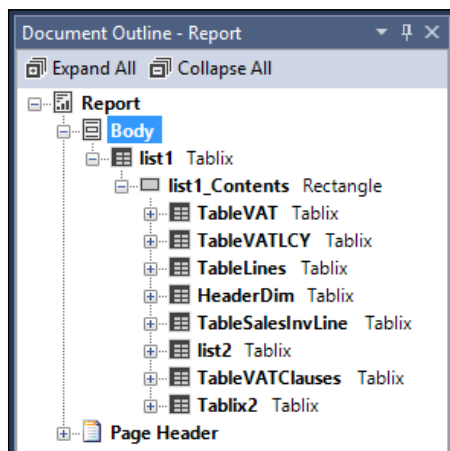
The layout

Now that we have seen how the dataset of a typical document report is generated, let's have a look at how we can create the layout.

Filtering the dataset

Since a document report contains several data items that contain information that needs to be displayed in different sections of the layout, you use separate tables (Tablix) to do that. But, by default, if you access a dataset field from a Tablix, the Tablix will loop over and show all the dataset records. That is why you need to assign a filter to each table, so that only those records that come from a specific data item are shown in the Tablix.

Open the layout for **Report 206 Sales - Invoice** and then open **Document Outline**, to see an overview:



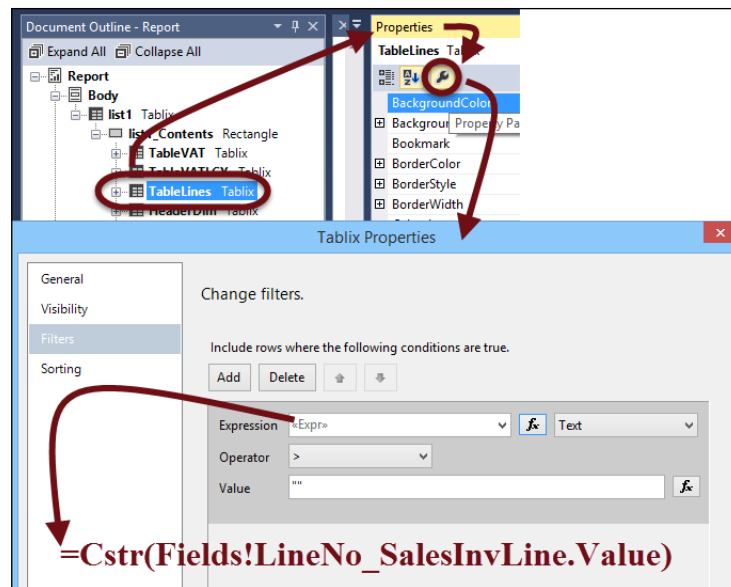
You can see similar information in the report properties drop-down list but in the document outline, everything is outlined in a hierarchy.



Document Outline

The document outline can be activated from the menu: **View, Other Windows**, or via the shortcut: *Ctrl + Alt + T*. It is a really nice tool that can help you to get a better understanding of how a report layout is constructed, for example when you need to reverse engineer a report. It is one of those hidden features of Visual Studio. The document outline is not available in Report Builder.

Each of the tables in the layout of this report contains filters. An example can be seen in the following screenshot:

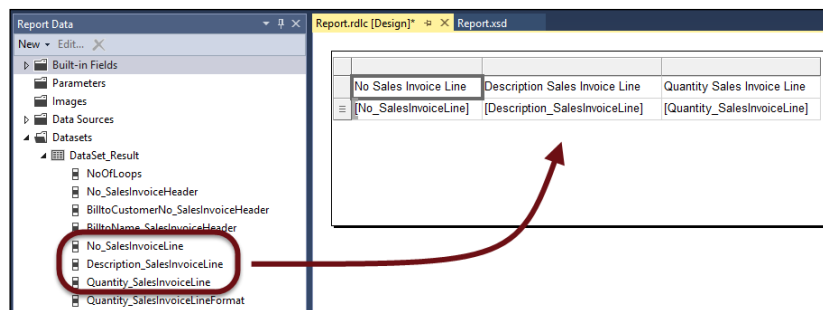


When applying filters in the report layout, it is important to understand that data types in C/AL are not the same as data types in VB.NET. To make sure the filter works, you need to convert the field in the dataset to a string via the `Cstr()` conversion function.

Working with headers and footers

Let's go back to the example report that I used to implement the `NoOfCopies` option. To follow the steps, import the object: Packt - CH05-10.

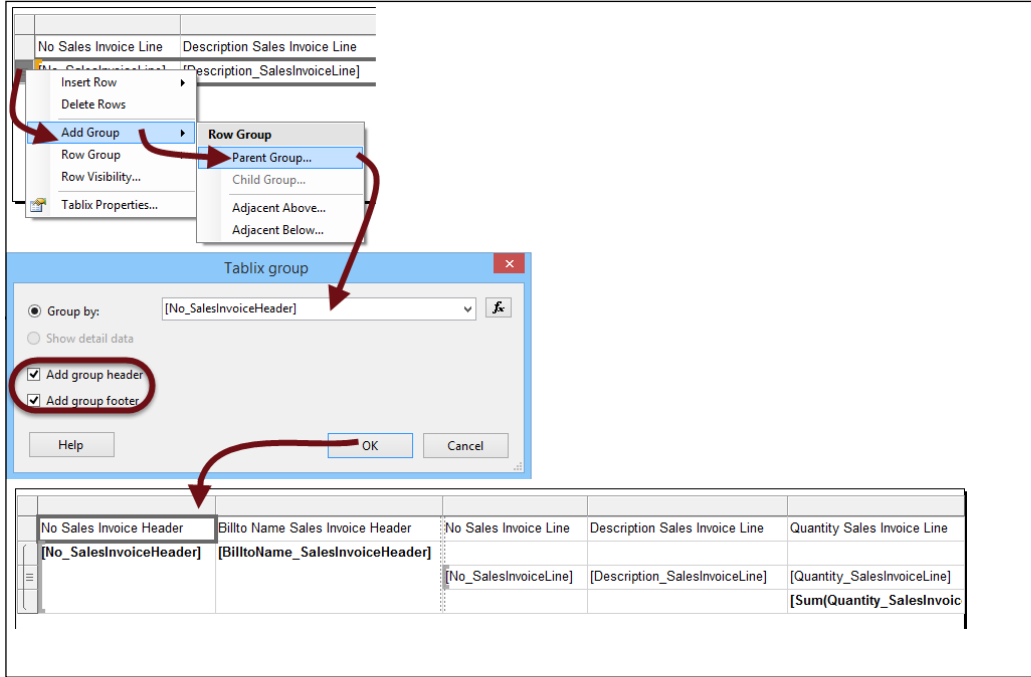
Imagine that I am building the report layout using a Tablix to show the fields from the line table. This layout would look like the following screenshot:



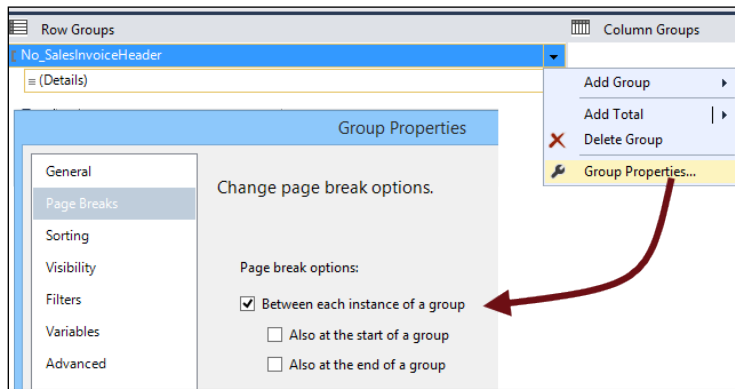
The layout contains a Tablix with the fields from the line table on the detail rows.

Document Reports

Now, add a group on the document number and, in the group header, add the fields from the header table and, in the group footer, add the totals:



Enable the page break option in **Group Properties...**, to have every document printed on a new page.



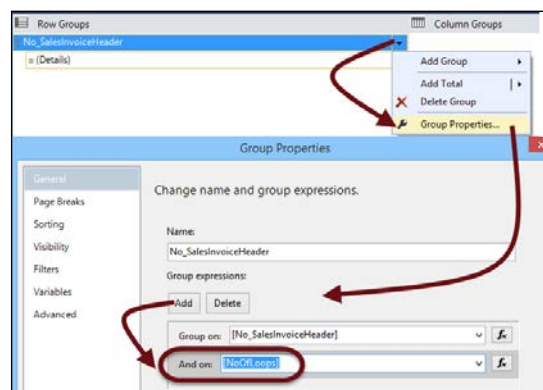
The problem with this layout is the `NoOfCopies` option. When it is more than 0, for example 1, then, when you run the report, the layout looks as follows:

No Sales Invoice Header	Billto Name Sales Invoice Header	No Sales Invoice Line	Description Sales Invoice Line	Quantity Sales Invoice Line
103001	The Cannon Group PLC	TIMOTHY	Assembling Furniture, January	25
		TIMOTHY	Assembling Furniture, January	120
		TIMOTHY	Assembling Furniture, January	25
		TIMOTHY	Assembling Furniture, January	120
				290

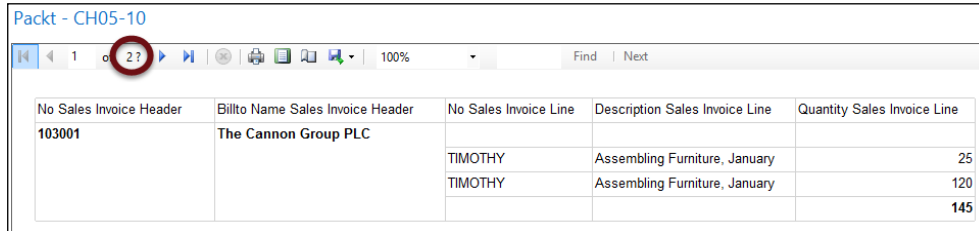
NoOfLoops	No_SalesInvoi...	BilltoCustome...	BilltoName_Sa...	No_SalesInvoi...	Description_SalesInvoiceLine	Quantity_Sale...	Quantity_Sale...
1	103001	10000	The Cannon Gr...	TIMOTHY	Assembling Furniture, January	25	##0,####
1	103001	10000	The Cannon Gr...	TIMOTHY	Assembling Furniture, January	120	##0,####
2	103001	10000	The Cannon Gr...	TIMOTHY	Assembling Furniture, January	25	##0,####
2	103001	10000	The Cannon Gr...	TIMOTHY	Assembling Furniture, January	120	##0,####

The copies are put into the table because the table is a one-on-one mapping of the dataset, grouped by document number. You need to have the copies printed as a separate document, on a new page. How can you modify this report to achieve this goal?

Well, when you think about it, it's actually very simple. The report already shows one new page per document number, and we want to extend that so it also shows a new page per copy. Why not put the copy (or `NoOfLoops` variable) inside the group? Let's see what this does to our layout:



As you can see, when we print the report, the copy is shown on a new page:



No Sales Invoice Header	Billto Name Sales Invoice Header	No Sales Invoice Line	Description Sales Invoice Line	Quantity Sales Invoice Line
103001	The Cannon Group PLC			
		TIMOTHY	Assembling Furniture, January	25
		TIMOTHY	Assembling Furniture, January	120
				145

So our problem is solved, or is it?

Well, if the report had a simple layout like the one shown here, then our problem would be solved. But, in an actual document report, you have more than one table. Usually, VAT (and other) information is printed in other tables, which are also in the body of the report and when you add another table to the report layout, then it will also need to be printed for every document. If I put a table below the current table, then it would be printed after the current table, and so it would not be printed on every page.

You might think, no problem, I will also group the second table on the `NoOfLoops` variable to solve this problem. Well, if you do that, the problem is still there. If the first table contains several rows, so that the table spans over multiple pages, then the second (and third, fourth...) table will be pushed down and will not be shown on the correct invoice when you print multiple invoices at the same time.

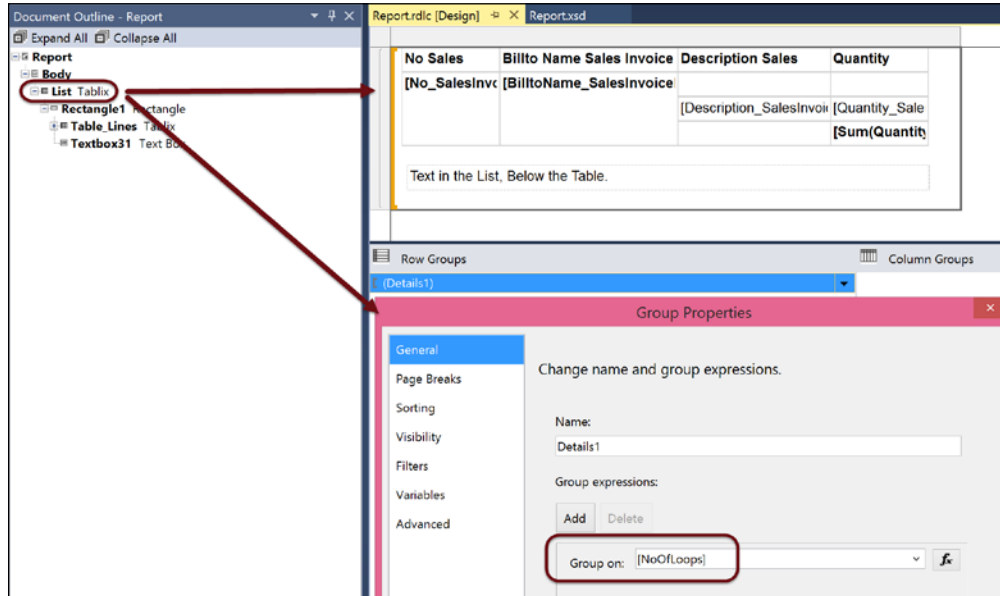
So we need another solution. We need to find a way to print every document and every copy for every table we put onto the report body. Ideally, if we could use a property that allowed us to group the body itself, that would solve our problem, but this is not possible. On the other hand, we can simulate this. How can we simulate a group on the entire body of the report and include all the tables it contains?

The solution is actually quite simple. If we can't group the body itself, then we add something to the body that we can group. You can use data regions in the toolbox because only data regions support grouping. The data region that I'm going to use is the list. I could also use a table or a matrix, but a list has a simple layout that will not even show at runtime. Then, I will put all of the tables that normally go into the body of the report inside the list control and add the groups to the list itself. This will allow you to simulate the `NoOfCopies` in the report layout because the list is grouped and it contains the entire body content.

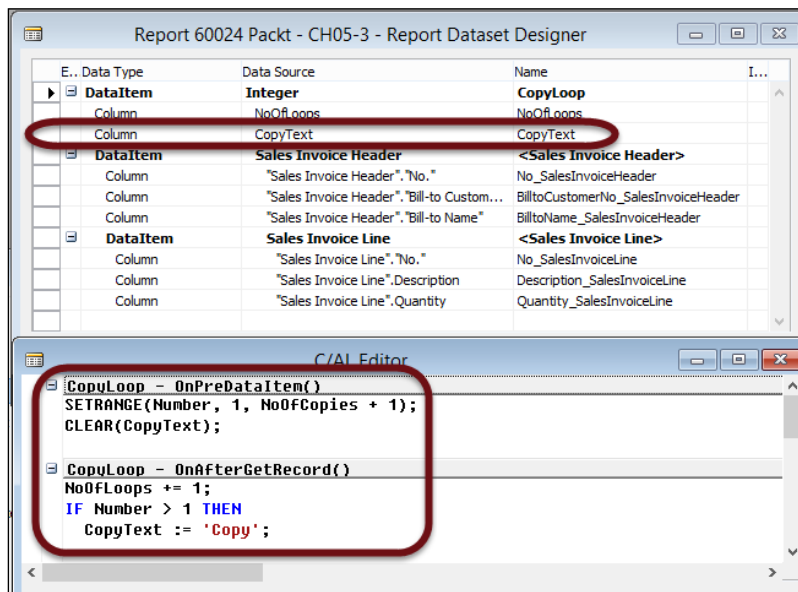


An example of this report is available in the object: Packt - CH05-3

Let's look at an example of this solution:



When you run the report, every page looks the same, and contains the table and the textbox below it. The textbox below the table contains text COPY if the page is a copy and not the original document. In order to do this, you can use the following code:



The following screenshot displays the result when you run the report:


The screenshot shows a report viewer with three instances of a report titled "Packt - CH05-3". Each instance displays a table with the following data:

No Sales Invoice Header	Billto Name Sales Invoice Header	Description Sales Invoice Line	Quantity Sales Invoice Line
103001	Van Terp Kantoorinrichting	Montage meubels, januari	25
		Montage meubels, januari	120
			145

The dialog box on the left shows the following options:

- Options: No Of Copies?: 3
- Sales Invoice Header
- Sorting: No.
- Show results: Where No. is 103001
- Buttons: Print, Preview, Cancel

Red arrows indicate the flow from the "No Of Copies?" field in the dialog box to the page numbers (1, 2, and 3) in the report headers.

[ An example of this report is available in the object: Packt - CH05-3]

Remember that, if you now add another Tablix below the first one, you need to move the group on No_SalesInvoiceHeader from the Tablix to the list.

GetData and SetData explained

The next problem you will need to fix arises when you add a header or footer to the report. You may have noticed that, when you drag and drop a field from the dataset onto the header (or footer) of a report, the system uses an aggregate function, usually `First(FieldName.Value, "Dataset_Result")`.

This expression fetches the value from `FieldName` from the very first row in the dataset. In most reports, that is just fine.

But, in a document report, imagine that you are printing invoices for multiple customers. You need to put the bill-to or send-to information of the customer in the header of the report. This information is linked to the lines that you are invoicing, which are in the report body. If you print two invoices for two different customers and simply drag and drop the bill-to or any other field from the dataset onto the report header, then the system will use the `First()` function and, in doing so at runtime, every invoice will display the information from the first customer in the header. This is definitely not what you want.

The information in the header of the report should be correct and it should be linked to the record displayed in the body of the report. So, as a consequence, you cannot simply drag and drop fields from the dataset onto the report header (or footer).

So, how can you solve this problem? Well, what you do is you put the data that needs to be available on the header of every page into a global variable that's accessible from every page, and contains the header information linked to the record shown in the body of the report. To do that, you need to do three things:

- Declare a global variable
- Create a function to store a value in the global variable
- Create a function to retrieve a value from the global variable



Open object: Packt - CH05-4 to see how this solution is implemented.

Declaring the global variable and functions

This can be done in the **Code** tab of the report properties, as follows:

```
Shared GlobalVariable as Object

Public Function SetGlobalVariable(Value as Object) As Boolean
    If Value > "" Then
        GlobalVariable = Value
    End If
    Return True
End Function

Public Function GetGlobalVariable() As Object
    Return GlobalVariable
End Function
```

As you can see, we have two functions: `Set` and `Get` for `GlobalVariable`. This is a simplified example that shows you how you can do it for one field. Once you have defined the variable and functions, then you need to add a textbox to the header of the report and use the following expression for it:

```
=Code.GetGlobalVariable
```

Before we can get the value from the variable, we first need to set it. This can only be done from within the body of the report, because there we have a link to the current record in the dataset.

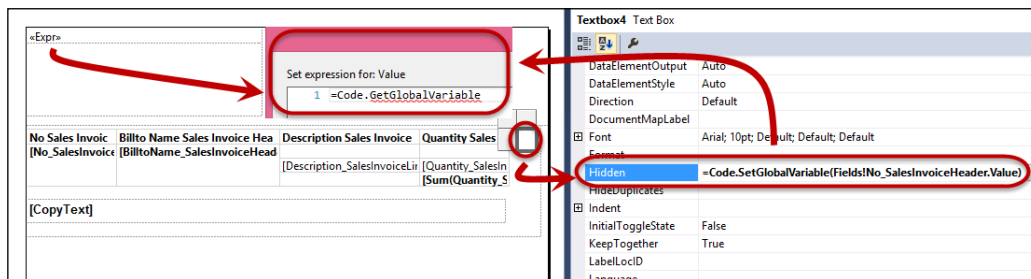
You therefore add a textbox to the body, and in this textbox you call the `Set` function. In order for this textbox not to show up, you also hide it, and this is where we have a problem. In the latest version of RDLC, when a textbox is hidden, then the expression for its `Value` property is no longer executed. So we can't call the `Set` function in the `Value` expression of the textbox.

The only property for which we are sure that it is evaluated is the `Hidden` property. So we will call the `Set` function in the `Hidden` property of the textbox.

In order to do that, you also need to make sure the `Set` function returns a `True` value, because that will be used as the `Value` for the `Hidden` property of the textbox.

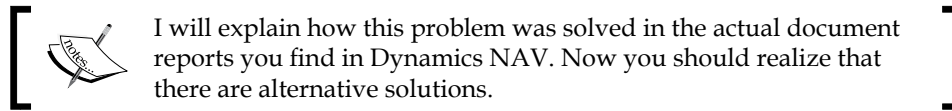
Implementing the Get and Set functions

Let's have a look an example which demonstrates how to implement this pattern:



An example of this report is available in the object: Packt - CH05-4

In real life, you need to add multiple fields to the report header or footer, and you are not going to create a separate global variable, and a `Get` and `Set` function for each field. So you need to make the function more generic. The variable needs to be able to store more than one value.



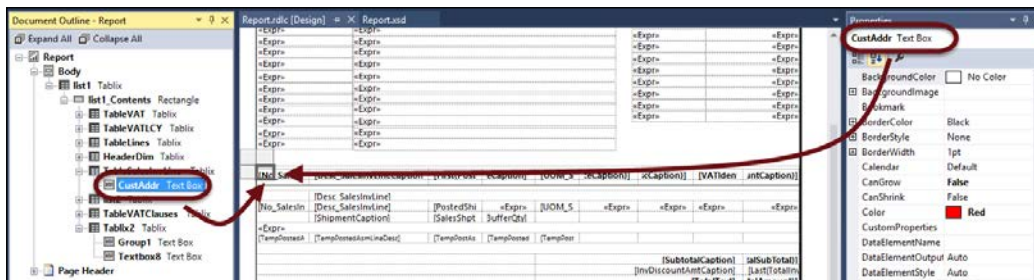
In order to store multiple values in one variable, Microsoft decided to concatenate all fields in the one variable. To do that, you need to do two things:

- Convert all values to strings
- Use a delimiter to concatenate the strings

Converting to strings can be done with the `CStr()` function and, as a delimiter, you need to choose something which you are certain is not a part of the strings that you are concatenating. The character that was selected for this is `Chr(177)`:

`Chr(177)` is actually the `+/-` sign.

The `SetData` function, in report 206 Sales Invoice, is in the textbox named `CustAddr`. To find it, use the properties window or the document outline. The textbox is hidden behind the `TableSalesInvLine`:



If you have a look in the `Hidden` property of the `CustAddr` textbox in **Report 206 Sales - Invoice**, you will see the following:

```
=Code.SetData(Cstr(Fields!CustAddr1.Value) + Chr(177) +
Cstr(Fields!CustAddr2.Value) + Chr(177) +
Cstr(Fields!CustAddr3.Value) + Chr(177) +
Cstr(Fields!CustAddr4.Value) + Chr(177) +
Cstr(Fields!CustAddr5.Value) + Chr(177) +
Cstr(Fields!CustAddr6.Value) + Chr(177) +
Cstr(Fields!CustAddr7.Value) + Chr(177) +
Cstr(Fields!CustAddr8.Value) + Chr(177) +
Cstr(Fields!BilltoCustNo_SalesInvHdr.Value) + Chr(177) +
Cstr(Fields!NoText.Value) + Chr(177) +
```

```
Cstr(Fields!YourReference_SalesInvHdr.Value) + Chr(177) +
Cstr(Fields!No_SalesInvHdr.Value) + Chr(177) +
Cstr(Fields!HdrOrderNo_SalesInvHdr.Value) + Chr(177) +
Cstr(Fields!PostingDate_SalesInvHdr.Value) + Chr(177) +
Cstr(Fields!DueDate_SalesInvHdr.Value) + Chr(177) +
Cstr(Cstr(Fields!PricesInclVATYesNo_SalesInvHdr.Value)) + Chr(177)
+
Cstr(First(Fields!DocDate_SalesInvHdr.Value)) + Chr(177) +
Cstr(Fields!SalesPurchPersonName.Value) + Chr(177) +
Cstr(Fields!DocumentCaptionCopyText.Value) + Chr(177) +
Cstr(Fields!PaymentTermsDesc.Value) + Chr(177) +
Cstr(Fields!ShipmentMethodDesc.Value) + Chr(177) +
Cstr(Fields!CompanyAddr1.Value) + Chr(177) +
Cstr(Fields!CompanyAddr2.Value) + Chr(177) +
Cstr(Fields!CompanyAddr3.Value) + Chr(177) +
Cstr(Fields!CompanyAddr4.Value) + Chr(177) +
Cstr(Fields!CompanyAddr5.Value) + Chr(177) +
Cstr(Fields!CompanyAddr6.Value) + Chr(177) +
Cstr(Fields!CompanyInfoPhoneNo.Value) + Chr(177) +
Cstr(Fields!OrderNoText.Value) + Chr(177) +
Cstr(Fields!EMail.Value) + Chr(177) +
Cstr(Fields!HomePage.Value) + Chr(177) +
Cstr(Fields!CompanyInfoEnterpriseNo.Value) + Chr(177) +
Cstr(Fields!CompanyInfoGiroNo.Value) + Chr(177) +
Cstr(Fields!CompanyInfoBankName.Value) + Chr(177) +
Cstr(Fields!CompanyInfoBankAccNo.Value) + Chr(177) +
Cstr(Fields!PricesInclVAT_SalesInvHdrCaption.Value) + Chr(177) +
Cstr(Fields!InvNoCaption.Value) + Chr(177) +
Cstr(Fields!SalesInvPostingDateCptn.Value) + Chr(177) +
Cstr(Fields!BilltoCustNo_SalesInvHdrCaption.Value) + Chr(177) +
Cstr(Fields!CompanyInfoBankAccNoCptn.Value) + Chr(177) +
Cstr(Fields!CompanyInfoBankNameCptn.Value) + Chr(177) +
Cstr(Fields!CompanyInfoGiroNoCaption.Value) + Chr(177) +
Cstr(Fields!CompanyInfoEnterpriseNoCptn.Value) + Chr(177) +
Cstr(Fields!CompanyInfoPhoneNoCaption.Value) + Chr(177) +
Cstr(Fields!PageCaption.Value) + Chr(177) +
Cstr(Fields!ReferenceText.Value) + Chr(177) +
Cstr(Fields!SalesPersonText.Value) + Chr(177) +
Cstr(Fields!VATNoText.Value) + Chr(177) +
Cstr(Fields!SalesInvDueDateCaption.Value) + Chr(177) +
Cstr(Fields!DocumentDateCaption.Value) + Chr(177) +
Cstr(Fields!PaymentTermsDescCaption.Value) + Chr(177) +
```

```

Cstr(Fields!ShptMethodDescCaption.Value) + Chr(177) +
Cstr(Fields!EMailCaption.Value) + Chr(177) +
Cstr(Fields!HomePageCaption.Value) + Chr(177) +
Cstr(Fields!LegalNoticeText.Value) + Chr(177) +
Cstr(Fields!ShipmentDate.Value) + Chr(177) +
Cstr(Fields!ShipmentDateText.Value)
, 1)

```

And, if you have a look at the Value expressions of the header textboxes, you will see this:

```
=Code.GetData(x,y)
```

In this report, there are four global variables:

- Data1
- Data2
- Data3
- Data4

Each of the variables is accessible via the Get function and each variable can contain multiple values. That is why the Get function uses two parameters, X and Y. One indicates the variable and the other indicates the field stored inside the variable:

```

Public Function GetData(Num as Integer, Group as integer) as
Object
if Group = 1 then
Return Cstr(Choose(Num, Split(Cstr(Data1),Chr(177))))
End If

if Group = 2 then
Return Cstr(Choose(Num, Split(Cstr(Data2),Chr(177))))
End If

if Group = 3 then
Return Cstr(Choose(Num, Split(Cstr(Data3),Chr(177))))
End If

if Group = 4 then
Return Cstr(Choose(Num, Split(Cstr(Data4),Chr(177))))
End If
End Function

```

The Set function also works with two parameters, one to indicate the value to store and another to indicate the variable in which to store it:

```
Public Function SetData(NewData as Object,Group as integer)
  If Group = 1 and NewData > "" Then
    Data1 = NewData
  End If

  If Group = 2 and NewData > "" Then
    Data2 = NewData
  End If

  If Group = 3 and NewData > "" Then
    Data3 = NewData
  End If

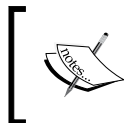
  If Group = 4 and NewData > "" Then
    Data4 = NewData
  End If

  Return True
End Function
```

When you customize the report and make changes, it is very important that the report stores multiple values in one variable in a specific order, so that:

- You always add new fields at the end of the SetData function
- You do not remove any fields from the function

If you added a new field at the beginning or somewhere in the middle, then all the fields below would get a higher number in the position of the concatenation, and so the header would start to retrieve the wrong fields and you would need to manually update all the GetData functions. If you remove a field from the SetData function, the same thing would happen.



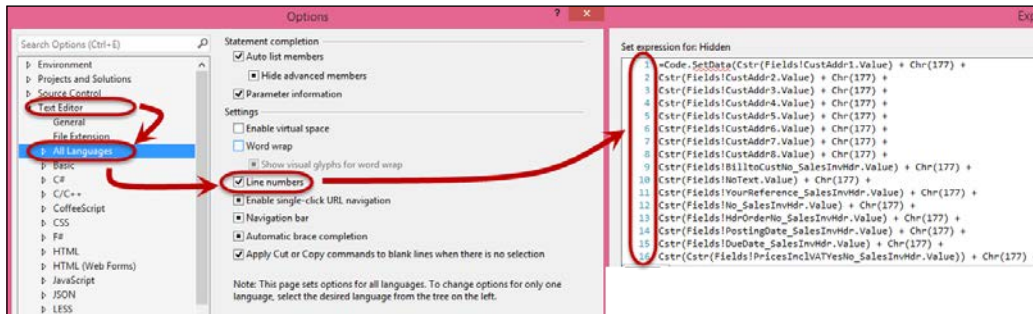
How do I remove a field from SetData?

In order to remove a field from the Setdata function, simply replace it with an empty string: "" + Chr(177) +.



Enable row numbers in the Visual Studio expression designer

In order to quickly see the number for a field in the `GetData` function, you can enable **Row Number** in the expression designer in Visual Studio. To do that, follow these steps:



In Visual Studio, open **Tools, Options**. Then, on the left-hand side, select **Text Editor**, then **All languages**. There, enable **Line Numbers**.

Alternative solutions – the mini-document

An alternative solution to create a simpler document report is available in the mini-document. **Report 1306 Mini Sales - Invoice** is an example of such a report.

In many cases, you don't need to implement the number of copies option, and if that's the case, you should have a look at this report. The data model is relatively simple and contains the `Document Header` and `Line` tables and also tables with related information.

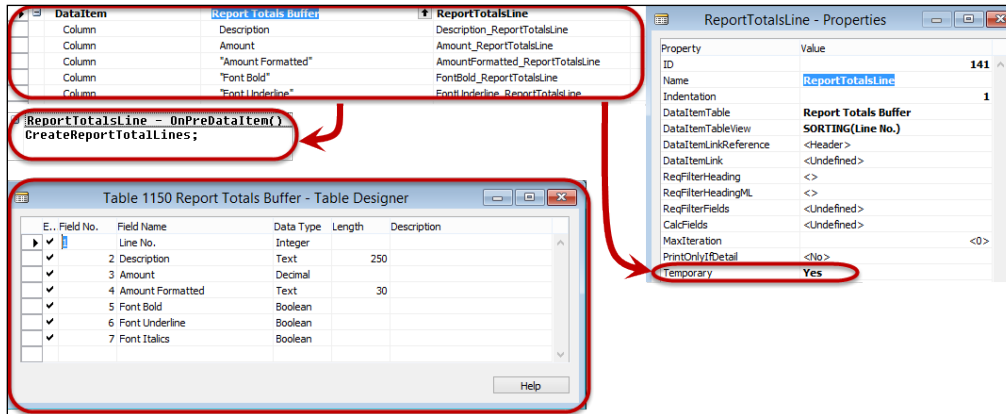
The problem that I described with the image that was repeated on every record of the dataset is solved by using a variable `FirstLineHasBeenOutput` and by clearing the image with the following code in the `OnAfterGetRecord()` trigger of the `Line` table:

```
IF FirstLineHasBeenOutput THEN
    CLEAR (CompanyInfo.Picture);
    FirstLineHasBeenOutput := TRUE;
```

To calculate report totals, a new data item is used, called `ReportTotalsLine`. This data item refers to a new buffer table called **Report Totals Buffer** and via the data item `Temporary` property. This data item is used as a temporary table.

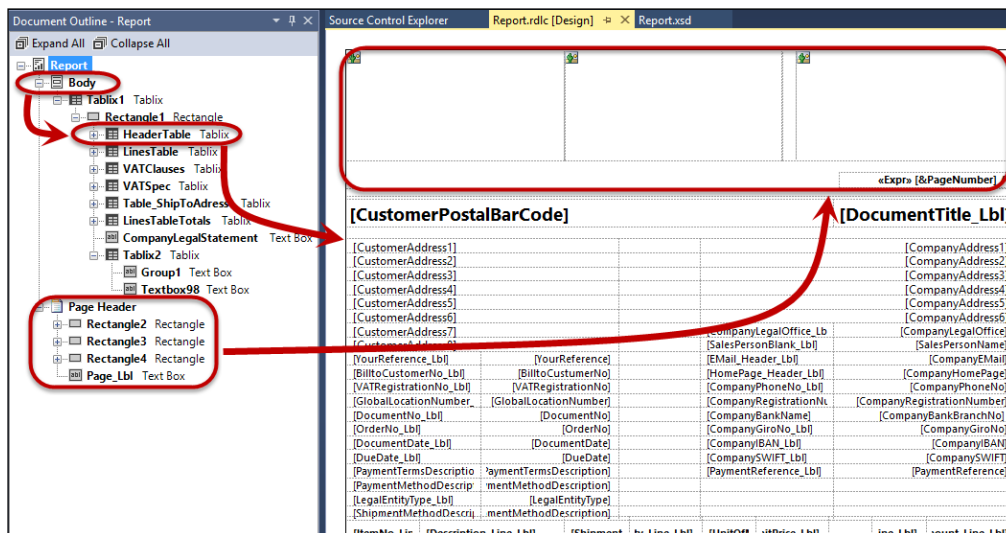
Document Reports

In the `OnPreDataItem()` trigger, you will see a call to the function `CreateReportTotalLines`, which will calculate the totals:



As you can see, the data model, data item, and column names are very clear. The names are kept simple and easy to understand. The reason is that this report is typically used as a basis for a Word layout. The Word layout is managed and designed by end users without much technical knowledge of Dynamics NAV. So the data item and column names need to clearly indicate what they can be used for and what information they contain. In general this is a principle that you should always apply. More information about the Word layout is covered in *Chapter 8, Word Report Layouts*.

This report also contains an RDLC layout and there you will notice something special:



Did you see it? The report header only contains the images and page numbers, all the rest has moved to the body of the report. This means that there is no need for the `Get` and `Set` functions. This really simplifies the development of the report layout and it also improves the readability. Every textbox clearly displays what it contains.

This layout simplification can be achieved by removing the `Copy` and `Page` loop data items and indenting all data items below the header. As you can imagine, it will also decrease design, development and maintenance time for this type of report.



What's missing from **Report 1306 Mini Sales - Invoice** is the number of copies option. But, as a solution, you could create a codeunit or report that runs this report and creates a loop to run it multiple times. You can then simulate the number of copies by printing the report multiple times. In order to have the word `COPY` on every copy you can then implement a parameter, similar to the `InitializeRequest` function in report 206, Sales Invoice, to pass along a Boolean variable that will, depending if it's `true` or `false`, print the word `COPY` in the header.

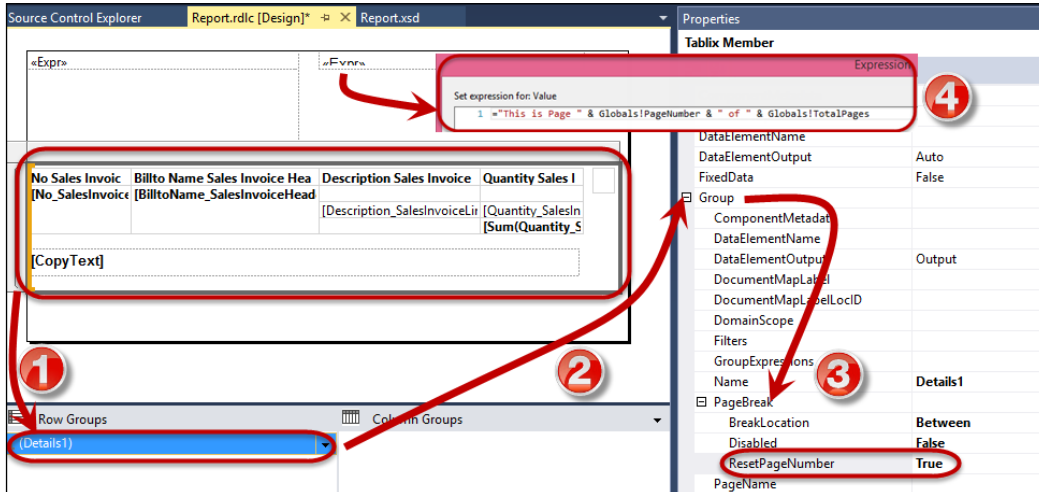
How do I implement page x of y?

Implementing page x of y in a document report used to be very complicated. What you need to do is display the current page number and total number of pages of the document but, every time a copy starts, when `NoOfCopies` is used, then you need to reset the current page number and, of course, recalculate the total number of pages. When the user prints multiple documents at the same time and some of them span over multiple pages with different tables being shown or hidden, this was a very complex task.

Because of its complexity, many partners opted for another solution. That solution usually involved adding the page number to the dataset of the report using some kind of counter. Other solutions first printed the report, or exported it to HTML, then counted the number of pages to use in the report, and then printed the report again including the page numbers and total pages. As you can imagine, this is very complex and not so good for the performance of the report.

But now, in Dynamics NAV 2013 R2 or higher, we can make use of a new property that has become available in RDLC. Whenever you create a group in a Tablix (`List`, `Table`, and `Matrix`), you can reset the page number every time a new group is reached in the group properties, and that list is grouped on the `OutputNo` and document number because, in a document report, all the information in the body resides in a list, and all you need to do is activate this property in the grouping of the list. To find this property, you will need to open the correct property window.

Let me demonstrate how you can do this:



To do this, follow these steps:


1. Select the **List** control in the body of your document report.
2. Select **Group** in the **Row Groups** window.
3. Open the **Properties** of the **Group**.
4. Enable `ResetPageNumber`.

Now, in the actual document reports in Dynamics NAV, the text that is used to display the page numbers contains placeholders. In C/AL, you can simply use a `STRSUBSTNO()` function to update the placeholders but, in VB.NET, you can use the `Replace` function, as shown in the following screenshot:

The image shows three overlapping windows from SAP Report Designer:

- Expression Editor:** Shows the expression `=Replace(Replace(Code.GetData(45,1), "%1", Globals!PageNumber), "%2", Globals!TotalPages)`. Below it, a list of fields is shown, with `Cstr(Fields!PageCaption.Value) + Chr(177) +` selected.
- Report Dataset Designer:** Shows a table with columns: `Column`, `Data Source`, `Name`, and `I...`. The `PageCaptionCap` column is highlighted.
- C/AL Globals Dialog:** Shows a table with `Name` and `ConstValue` columns. `PageCaptionCap` is selected with a value of `Page %1 of %2`. A **Multilanguage Editor** is also visible, showing translations for `Engels (Verenigde Staten)`, `Frans (België)`, and `Nederlands (België)`.

Red arrows indicate the flow of data and configuration from the Globals dialog to the Dataset Designer, and from the Dataset Designer to the Expression Editor.

[ An example of this report is available in object: Packt - CH05-5]

Summary

In this chapter, we have learned how the data model and layout of a typical document report is created. A document report contains several patterns that you can reuse when you create other types of reports. These patterns include implementing multilanguage functions, address formatting, company logos, totaling, filtering the dataset, and working with headers and footers. We also had a look at an alternative report for documents called the mini document.

In the next chapter, I will demonstrate some tips and tricks like working with hyperlinks, displaying header or footer information on specific pages, and Trans headers and footers. In *Chapter 6, Tips and Tricks*, I will demonstrate some common real world problems and issues, and how you can solve them.

6

Tips and Tricks

In this chapter I will cover some well-known time-savers and workarounds; and how you can solve problems. Sometimes a real solution is not available and then, I will demonstrate a workaround. The chapter is rather a collection of the tips, tricks and challenges you will encounter when you are charged with the task of implementing report layouts.

Report pagination

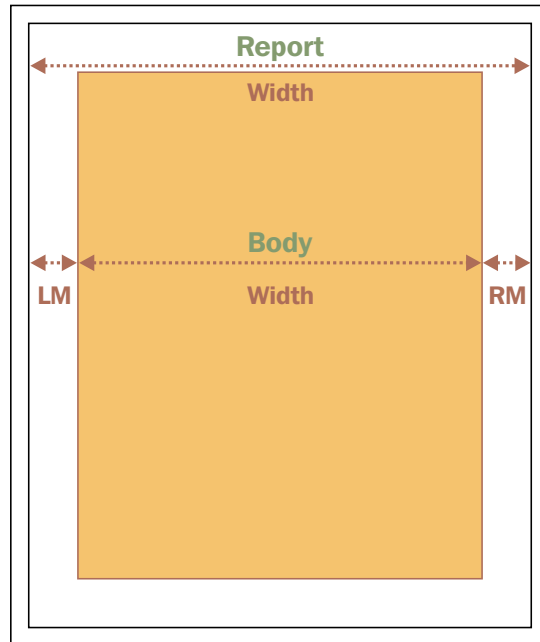
Have you ever had the problem that, when you run a report in preview mode, everything looks just fine, but, when you print the report or export it to PDF, there are blank pages in between the pages with information? If you have, then rest assured, this is a typical pagination problem that can be solved by selecting the correct width properties. One of the problems with RDLC reports is that the preview and printed versions might appear different.

First let's see what is exactly causing this blank page effect.

Basically, it's very simple. Whenever the report renderer, which is responsible for printing a report or generating the PDF version, encounters information that does not fit on the current page, it will print it on the next page, even if it's blank information.

When does this happen? It happens when the width of the report body is larger (meaning greater than or equal to) than the width of the report, minus the left and right margins.

The following diagram illustrates the margins:

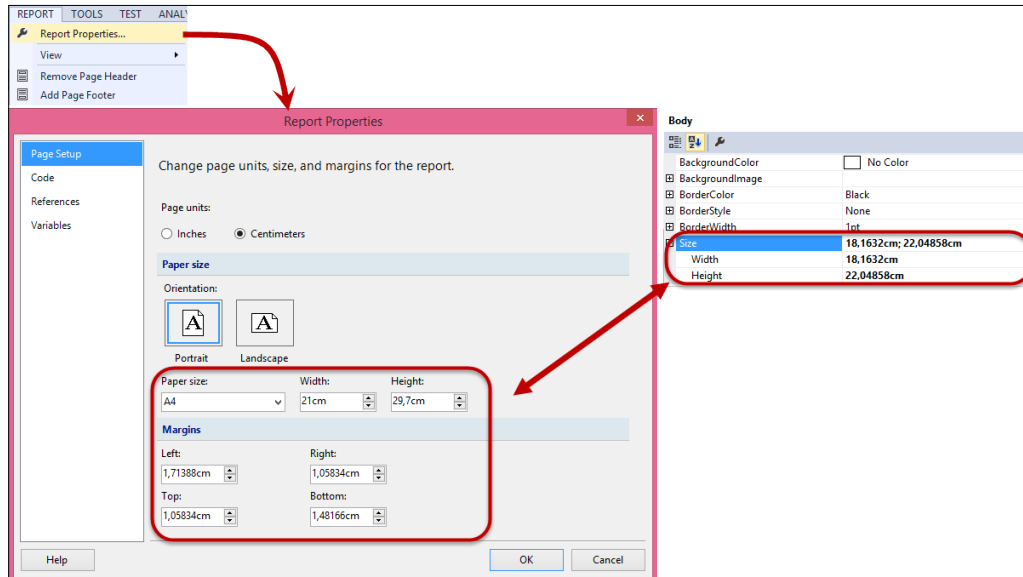


Import the object: Packt - CH06-9 to see the margins, and run the report in **Print Layout** mode.

You need to make sure the width properties of the body and report are set correctly to avoid the issue of having blank pages,

For example, if the width of the body is 20 cm, and the paper size is set to A4, this means that the width of the report is set to 21 cm. When you then set the left margin to 1 cm and the right margin to 1 cm, then the body width (20 cm) plus the left margin (1 cm) plus the right margin (1 cm) equals 22 cm, which is greater than the report width of 21 cm. This extra space is then printed on a new page.

You can do this in the report and body properties window:



Of course, sometimes the width of the body can grow, depending on the information it is displaying. This can happen when you use a matrix with a variable number of columns. Another time this might happen is when controls push other controls to the right. Typically, the scenario occurs when you set the `Hidden` property of certain controls using an expression, so sometimes the control is visible, and sometimes it's not.



Use Rectangles to keep objects together

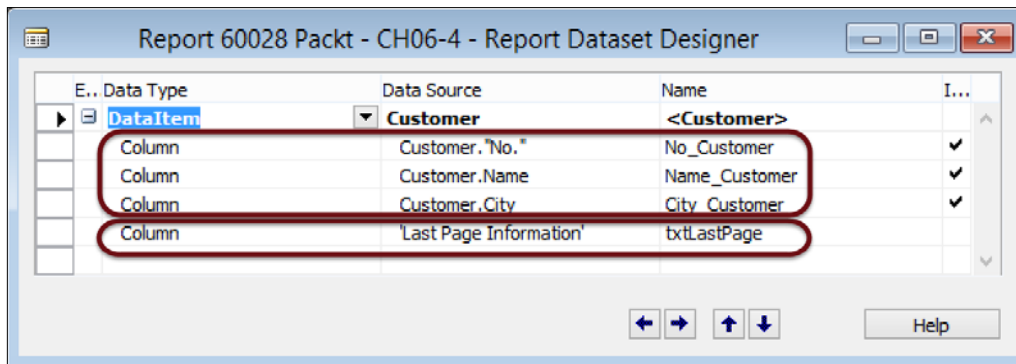
To avoid controls pushing each other from the page I recommend using Rectangle containers. Normally, when you put a control into a Rectangle and the control grows, it cannot grow bigger than the rectangle that contains it. Make sure you leave some whitespace to the right of the rectangle.

Always make sure you test your report in different rendering formats, Word, Excel, PDF, and Print, because each of them will render the report layout differently and that might cause this blank page effect to occur.

Show a footer or header on the last page

A question I get asked a lot is, how can you display information on certain pages and not on others? I will explain in this section how you can show or hide information on the last page of a report, and then use the same method for any page.

Let's start with a report that has a simple dataset with so many records that when you put them in a table, the table is printed on multiple pages:

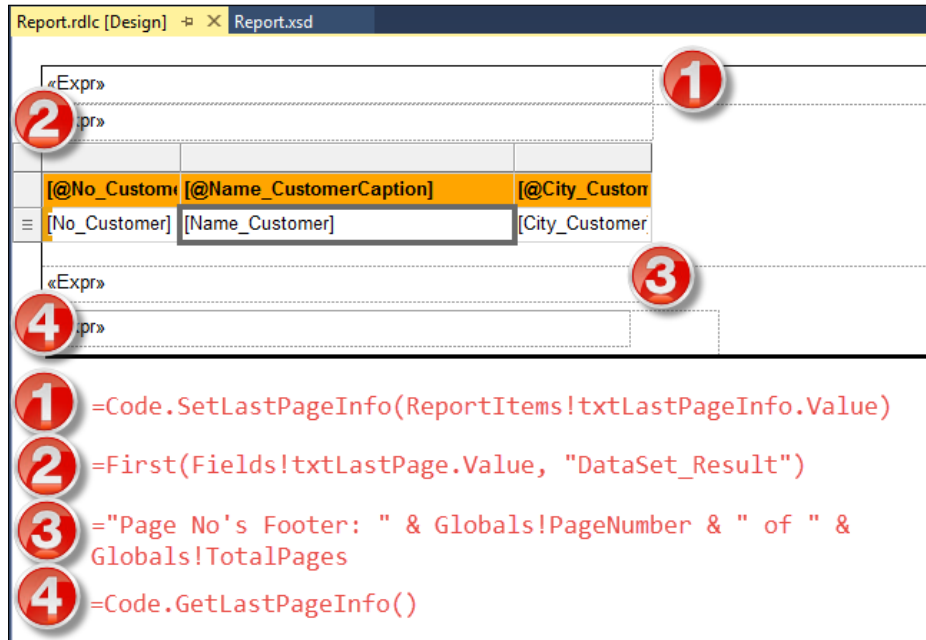


An example of this report is available in the object: Packt - CH06-4

I have added a field to this dataset named `txtLastPage` and, as its name suggests, this information only needs to be visible on the last page.

- Build a layout that contains a table with the fields from the dataset and, above the table and inside the body, add a textbox
- Add a page header in which you add a textbox

- Add a page footer in which you add two textboxes, as shown in the following screenshot:



You use expressions in each of the textboxes, as displayed on the right of the image.

Now, as you might have noticed, the first textbox executes a function `=Code.SetLastPageInfo(ReportItems!txtLastPageInfo.Value)`, which uses the content of the textbox `txtLastPageInfo`, which is the second textbox from the top and contains the following expression: `=First(Fields!txtLastPage.Value, "DataSet_Result")`. This expression fetches a value from the dataset and passes it as a parameter to the `SetLastPageInfo` function, and this stores the value in the global variable: `LastPageInfo`. The textbox at the bottom of the report, in the footer, uses the function `GetLastPageInfo()` to retrieve whatever is stored in the variable `LastPageInfo`.

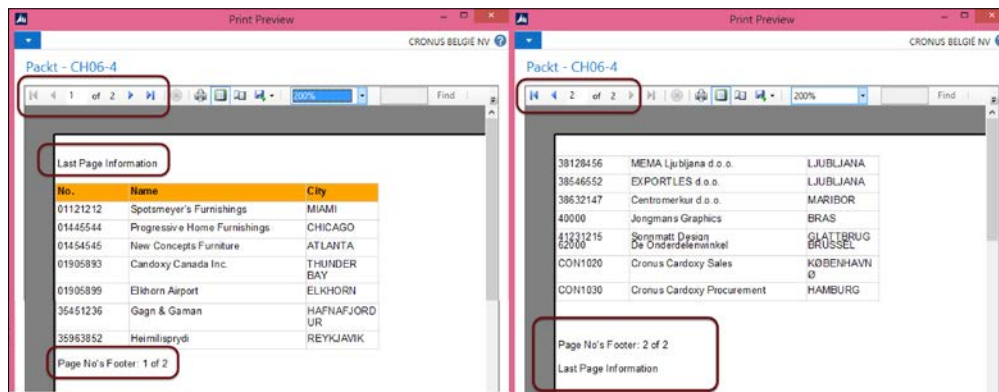
For this to work, you need to define the following in the **Code** tab of the report:

```
Shared LastPageInfo as Object
Public Function GetLastPageInfo() as Object
    Return Cstr>LastPageInfo)
End Function
Public Function SetLastPageInfo(NewData as Object)
    If NewData > "" Then
        LastPageInfo = NewData
    End If
End Function
```



This works exactly like the Get and Set functions in a document report.

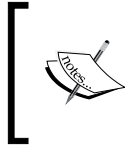
Now, when you run the report, the table is split over two pages and they are displayed, as shown in the following screenshot:



The footer displays the text **Last Page Information** on the last page only. Why is that? Well, it's because I have put that last textbox inside a Rectangle and in the Hidden property of that rectangle I used the following expression:

```
=Iif (Globals!TotalPages = Globals!PageNumber, False, True)
```

This means that the `Hidden` property of the rectangle is false only on the last page, because that's the page where the number is equal to the total number of pages.



I recommend using Rectangles for Hidden Expressions because they can contain multiple textboxes and then you only need to set and maintain the expression on the rectangle, and not on each individual textbox.

This example demonstrates how to hide a textbox on the last page, but it's very easy to do on another page by changing the expression. You can also do it for textboxes on the header of the page.

This example uses a function and global variable to store the information. That's not really required for this report but, when you implement this in document reports that contain several tables in the body, it's the only way to make it work properly.

Place at the bottom

In certain reports, it is necessary to place information at the bottom of the page, even when there's still room available above it. RDLC does not have a property with this effect. The only thing that will make sure it is placed at the bottom of the page is the page footer.

A disadvantage of putting information in the page footer is that, when you hide, for example, a textbox or multiple textboxes in a rectangle, the space is still reserved in the page footer, and you end up with a lot of white space at the bottom of your report.

A typical example of a place in bottom request is in the sales invoice report, where you have the table containing the invoice details, header and lines, and below it the table with the VAT information. A company usually wants the table with the VAT information to be placed at the bottom of the page, even when the table that displays the details only contains one, or a few, rows of information.

I must admit, there's not a good solution for this problem, or in other words, I haven't found a good working solution. Instead, I can explain a workaround.

The workaround consists of a way of using an expression which pushes the table that contains the footer information down the page, until it reaches the bottom of the page.

You see an example of the effect of this workaround in the following screenshot:

62000	The Device Shop	London	62000	The Device Shop	London
IC1020	Cronus Cardoxy Sales	København Ø	IC1020	Cronus Cardoxy Sales	København Ø
IC1030	Cronus Cardoxy Procurement	Hamburg	IC1030	Cronus Cardoxy Procurement	Hamburg
This is a textbox that contains text, to be displayed at the bottom of the page.					

The orange textbox in the first example is placed directly under the table, while in the second example it is placed at the bottom of the page.

It comes down to counting, and determining how many rows the table that contains the details requires or contains, to position the table below it towards the bottom of the page. Then, when the table contains fewer rows, you will generate empty rows that will push everything that comes after it to the bottom of the page.

To do this, you need a way to count the rows currently displayed in a table, for which you can use the `CountRows()` function. You need a function that is able to generate empty rows, or line breaks. The easiest way to generate a line break is the `vbCrLf` function, or the Visual Basic carriage return line feed.

You can use this function in the following way:

```
Public Function GenerateVbCrLf(ByVal Count as integer)
    dim Value as String
    dim i as integer
    For i = 1 To Count Step 1
        Value = Value & " " & vbCrLf
    Next i
    Return Value
End Function
```

This function uses a parameter and, depending on the value of the parameter, it generates a number of line breaks.

Imagine that you have counted, via a process of trial and error, that the number of rows on a page is x , then used the following expression to determine how many line breaks were required:

```
=Code.GenerateVbCrLf(X - cint(CountRows("Table1")))
```

This function allows you to generate any number of new lines dynamically.

The problem becomes even more complicated when there are multiple tables above the table that needs to be placed at the bottom of the page. Then, you need to count the rows in each of them and subtract them from the magic number (x). If so, I would use a global variable that holds the total of rows in all of those tables:

```
Shared TotalRows as Integer
Public Function SetCountRow(RowsToAdd as integer)
    TotalRows = TotalRows + RowsToAdd
End Function
Public Function GetCountRow() as Integer
    Return TotalRows
End Function
```


Next, you need to fill the `TotalRows` variable with the actual number of rows in each of the tables:

```
=Code.SetCountRow(cint(countrows("DimensionLoop1")))
=Code.SetCountRow(cint(countrows("SalesInvoiceLine")))
```

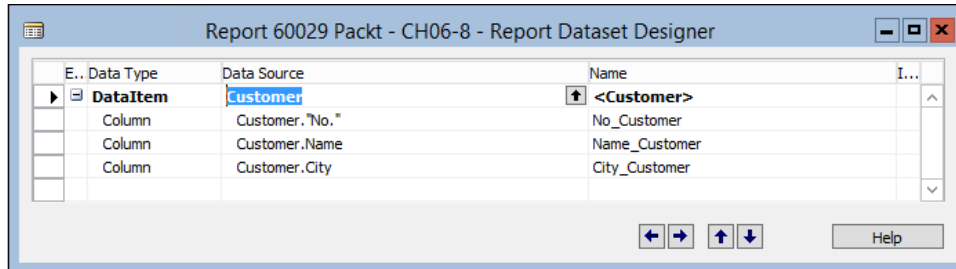
The function then generates the line breaks as follows:

```
=Code.GenerateVbCrLf(42 - Code.GetCountRow)
```

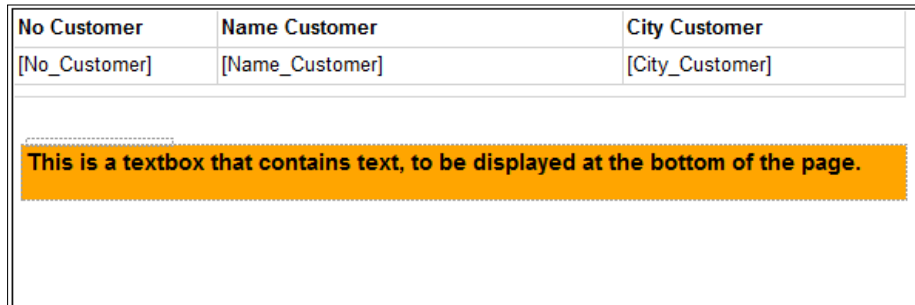
Let's look at an example.

[ An example of this report is available in the object: Packt - CH06-8]

I will start the report with a simple dataset, as follows:




I will then add a table to the layout to display the customer information, and a textbox below the table with the information that needs to be shown at the bottom of the page:



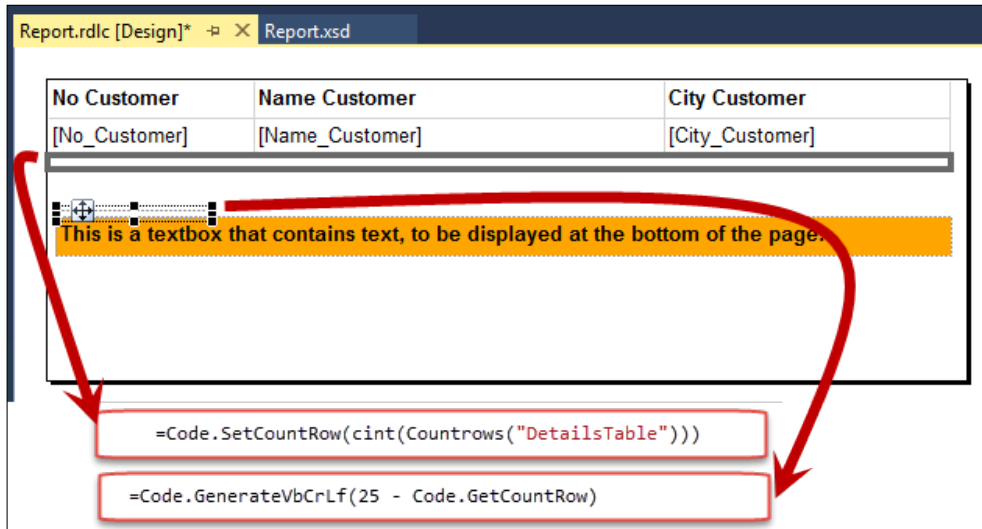
If you run the report, it should look as follows:

62000	The Device Shop	London
IC1020	Cronus Cardoxy Sales	København Ø
IC1030	Cronus Cardoxy Procurement	Hamburg

This is a textbox that contains text, to be displayed at the bottom of the page.



As you can see, the textbox is clearly not at the bottom of the page. I will now implement the counting rows function, as shown in the following screenshot:



I added a footer row to the table to execute the `SetCountRow` function. This needs to happen in the table, because the scope of the `CountRows` function must be an object that is within the scope of the textbox or, in other words, the textbox needs to be contained in the same object. The `rowcount` of the table needs to happen within the scope of the table.

Then I added a textbox above the orange textbox to generate the line feeds. This is a little hidden textbox that pushes down the elements at the bottom of the page.

The table ends with the last row in the dataset. To make sure the orange textbox is placed at the bottom of the page, I added the other textbox before it. Then we fill that last textbox with line feeds to expand the height of the textbox, which will push down the orange textbox.

Now that you have seen the example, you can see that it is only a workaround and not a perfect solution. The workaround depends on counting rows and is not very stable. In some cases it might not work or produce the desired results.

The only way to be one hundred percent sure that something is put at the bottom of the page is to put it into the page footer. In the page footer, you will have to work with textboxes that hold the information, and probably also with the `SetData` and `GetData` functions to retrieve the information from the dataset linked to the record currently displayed in the body. If you do this it is a good idea to put the information that needs to be positioned at the bottom of the page in a separate data item. Then you create a small textbox, within a table, in the report body, filter it on the specific data item and then use the `SetData` function to store the information in a separate variable. You use the `GetData` function from the page footer to retrieve those fields.



There are better ways to solve this problem. One way would be to use the information in the next section, *A Fixed number of rows*, in which you are going to create a table that always displays a fixed number of rows. When you put another table below it you are sure it will always be positioned at the same location in the page.

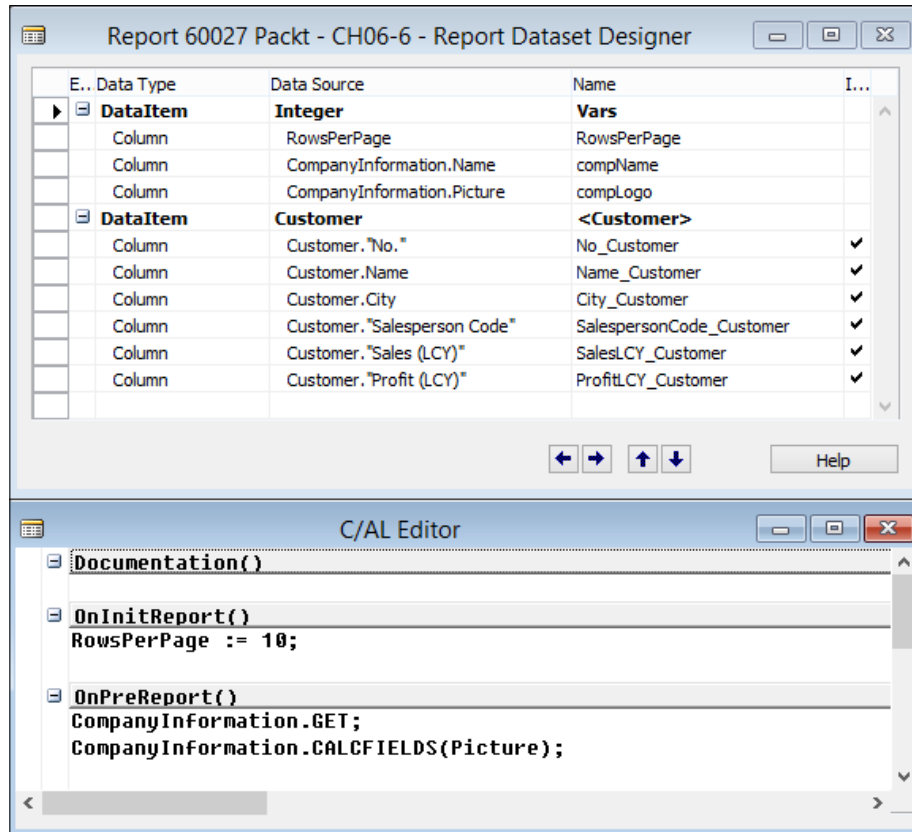
A fixed number of rows

This section describes a trick that you can apply when you need a table to always have the same number of rows, on every page. Normally, when you use a table and the information is printed over multiple pages, then, on the last page, the table stops when it runs out of data. You can avoid that with this trick, and position another element below the table, at the bottom of the page.



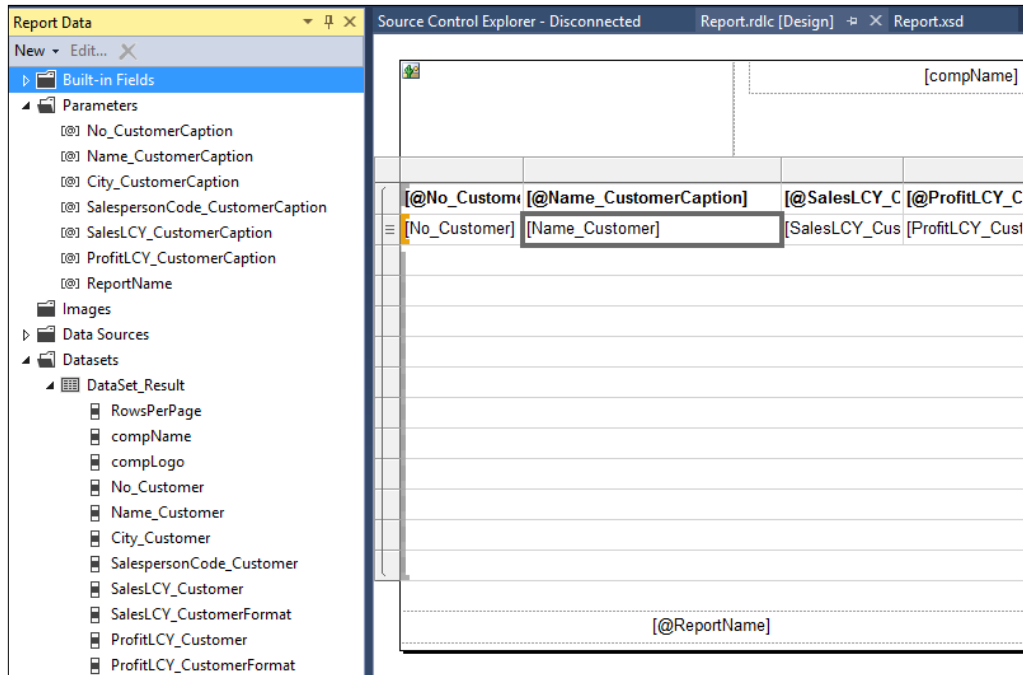
An example of this report is available in the object: Packt - CH06-6

In this example, I will start with the following dataset:



There's a customer data item to fetch records from the corresponding table, an integer data item to add one record to the dataset that contains a RowsPerPage variable, and some fields from the company information table.

I have added a table to the body in the layout of the report. I used the caption fields on the header row and the data fields in the detail row. I then added a report header on which I will display the company name and logo, and a report footer on which I display the report name, which comes from a label that I have added:



As you can see in the preceding screenshot, the table also contains several footer rows, which is done on purpose. The footer rows will not always be shown.

Imagine that you want 10 rows per page, and each footer row is a placeholder for when the last page has one, or two, or three, ... or nine rows. If there is one row left, then you want all nine footers to show. If there are two rows left, you want eight footers to be visible, and so on.

I have added a group to the table to generate a page break after every ten rows, which uses the following expression:

```
=Ceiling(Rownumber(Nothing) / 10)
```



The Ceiling function returns the smallest integer greater than or equal to the specified numeric expression. An alternative is the Floor function, which returns the largest integer less than or equal to the specified numeric expression. For example, when considering a numeric expression of 12.9273, Ceiling returns 13 and Floor - returns 12. The return value of both Floor and Ceiling is the same data type as the input numeric expression.

The RowNumber function returns the number of rows in an object. Nothing specifies the outermost context, usually the report dataset.

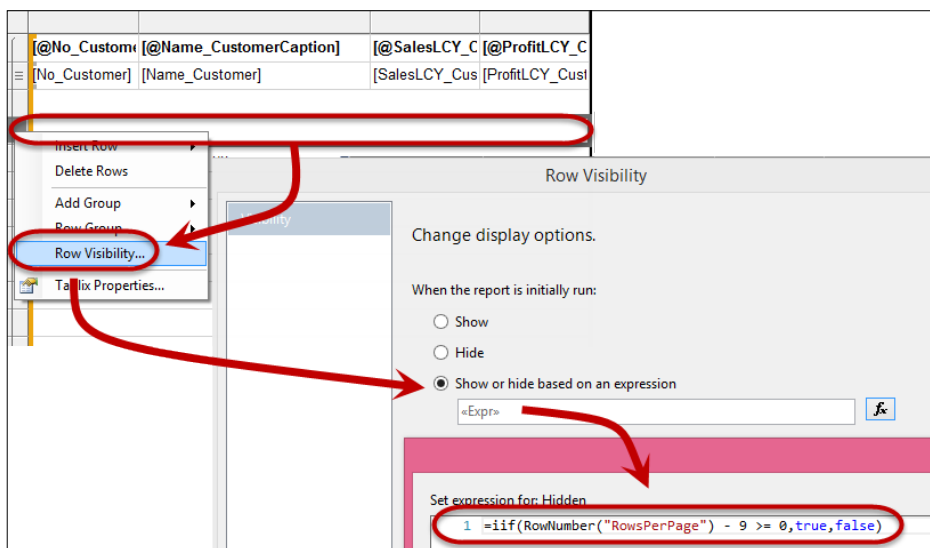
The name of the group is RowsPerPage, and I have removed the column that is normally added when you add a group to a table. I have enabled page breaks between each instance of the group in the properties of the group.

This report has so far been like any other report that you create when you want to have a page break after every tenth row.

I have used an expression that will evaluate to true or false in the Hidden property of each footer row to hide or show the row. The expression on every footer row is similar, but not the same. In the row visibility of the first footer row, you can see the following expression:

```
=iif(RowNumber("RowsPerPage") - 10 >= 0,true,false)
```

This means that the expression is true if the rownumber is bigger than 10 and, then, the footer row is hidden. I have added the following expression to the next footer:



This means that, if there are less than nine rows in the table, the footer will be shown. I will use similar expressions for all the footer rows until the last footer row, where the expression is `=iif (RowNumber ("RowsPerPage") >= 0, true, false)`.

When you run the report and look at the last page, the result is, as shown in the following screenshot:

No.	Name	Sales (LCY)	Profit (LCY)	No.	Name	Sales (LCY)	Profit (LCY)	No.	Name	Sales (LCY)	Profit (LCY)
20000	Anton Giestig Adhazen	10 000.43	3 803.43	01905902	London Candyox Storage Campus	0.00	0.00	K00020		0.00	0.00
62000	De Oinderdeleminkel	0.00	0.00					K00030		0.00	0.00
								K00060		0.00	0.00

There are two customers in the first print and only one in the second but, in both cases, the table displays the same number of rows. All the customers are there in the third print and there is the same number of rows on every page, including on the last page.

This means that the table is always going to have the same height and, if you position another table below it, it will start from a fixed position, independent of the number of rows in the first table.


Trans headers and footers

In Dynamics NAV, before version 2009, in the classic client, the report layout was developed in what was called the **Section Designer**. This was a layout designer that was optimized to create simple ERP types of reports without using data visualizations, though it contained some nice features that we use a lot in ERP reports. Two of these features are **transheader** and **transfooter**, sometimes also referred to as transport or running totals.

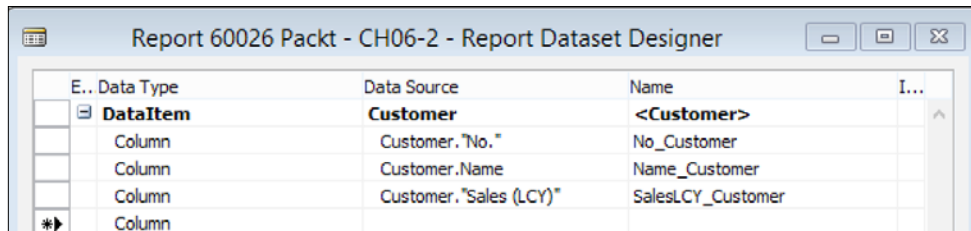
The idea behind it is that, when report information spans over multiple pages, you add a row (or a textbox or another report item) to the top of the next page that contains a total of the information that was shown in the previous page. Similarly, at the bottom of the page, you can also display a page total. You need to use an expression in the `rdlc` report layout, because there are no transheader or transfooter report items in the toolbox.

What you need is some kind of variable that can contain a page total (one or multiple) for every page. You could use an array because you have access to the .NET framework in the `rdlc` report layout. An array is a variable that can contain multiple values. The variable needs to support an index that can be used to indicate the page number for which it is holding a value, and it also needs to contain a value for each index.

Although an array provides this, in .NET a hash table might be more interesting. A `hashtable` is a variable that you can use to hold multiple values based upon a key/value pair. A `hashtable` also uses an internal hash key to speed up lookups in the key/value pairs, which is why it is faster to use, when compared with an array. Another possibility is a .NET generic Dictionary type, which is even faster than a `hashtable`. I will use a `hashtable` in this example.


[ An example of this report is available in the object: Packt - CH06-2]

I will start with a report that has a simple dataset, as shown in the following example:



E.. Data Type	Data Source	Name	I...
<input checked="" type="checkbox"/> DataItem	Customer	<Customer>	
<input type="checkbox"/> Column	Customer.No."	No_Customer	
<input type="checkbox"/> Column	Customer.Name	Name_Customer	
<input type="checkbox"/> Column	Customer."Sales (LCY)"	SalesLCY_Customer	
<input checked="" type="checkbox"/> Column			

The idea here is to select a table that contains enough rows to span over multiple pages. You then go into the report layout, and add a Tablix to the body that contains the dataset fields. Then, add a page header and a page footer, and put a textbox in them.

[ I used an olive and blue `BackgroundColor` in the textboxes in this example so you can spot them more quickly at runtime.]

The next thing to do is to create the `HashTable` variable, and the functions to work with the table. You can use the following code and add it to the report **Code**:

```
Shared RunningTotals As New System.Collections.Hashtable

Public Function GetRunningTotal (ByVal CurrentPageNumber)
    Return IIF (CurrentPageNumber > 0,
        RunningTotals (CurrentPageNumber), 0)
End Function

Public Function SetRunningTotal (ByVal CurrentPageTotal, ByVal
    CurrentPageNumber)
    RunningTotals (CurrentPageNumber) = CurrentPageTotal +
    GetRunningTotal (CurrentPageNumber - 1)
    Return RunningTotals (CurrentPageNumber)
End Function
```

The `RunningTotals` variable is the hashtable that holds the page totals. The `GetRunningTotal` function retrieves a total from the hashtable, based on the page number you provide. The `SetRunningTotal` function stores and adds a page total to the hashtable for the page number that you specify. The hashtable variable then holds a key/value pair at runtime that becomes the page number (key) and page total (value).

You now need to use the variable. There's a total in our table, in the footer row, for the `Sales (LCY)` field. This total is shown on the last page and is a grand total of all the rows. The textbox on the page footer holds the page total, displaying the total for the `Sales (LCY)` displayed on the current page. This total is simply the sum of the values shown in the textbox on the detail row that contains the `Sales (LCY)` field. The name of this textbox is `SalesLCY_Customer`.

You can then use the footer textbox in the following expression:

```
= "Transfooter subtotal = " &
    Code.SetRunningTotal (Sum (ReportItems!SalesLCY_Customer.Value),
        Globals!PageNumber)
```

This expression uses the `ReportItems` collection to refer to the `SalesLCY_Customer` textbox and calculates the `Sum` or page total. The result, or `Sum`, is then sent as a parameter to the `SetRunningTotal` function and linked to the `PageNumber`. The function also returns this value as the total for the current page.

You can then use the following expression for the textbox on the page header:

```
"Transheader subtotal = " &  
Code.GetRunningTotal (Globals!PageNumber-1)
```

This will retrieve the total for the previous page, which is the page where the page number is equal to the current page number minus one.

This is how you can show page totals or transheader and transfooters in the report rdlc layout.



Page header/footer required

The PageNumber and TotalPages, from the Globals collection, can only be accessed from within a page header or footer. That is why you need to add a page header and page footer to the report layout for this solution to work.

Creating links

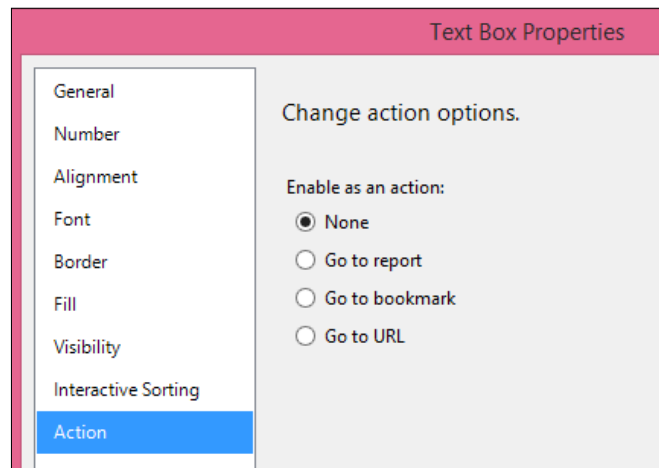
Sometimes, as a user, when you run a report and analyze the information that is displayed on it, you might require more in-depth information about related subjects. The report may show a document number, or an item or customer number and you want to provide the user with a way to drill through to the document report or item card or customer list page. You can do this by creating links. A link can be created in several ways. First we will have a look at how it works in the layout and then at the kind of links that can be used.

Links always start from inside a textbox in the layout of a report.



Remember that a chart also contains textboxes, the difference is that a chart renders them differently. This can also be applied to parts of a chart.

When you right-click on a textbox and go to its properties, have a look in the **Action** pane:



Here is where you can define an action or, in other words, a link or a hyperlink:

- The **Go to report** option is not available in Dynamics NAV. It is meant for Reporting Services reports that run on the SQL Server Report Server
- The **Go to bookmark** is meant for internal navigation in a report
- The **Go to URL** is meant to navigate or link to an external object which could be a Dynamics NAV object, a network drive, or a file or website, and so on

Enable hyperlinks



In order to compile a report that uses links in its layout, you need to activate the report `EnableHyperlinks` property. Otherwise you will get the following error: Report RDLC control 'XXX' contains an external link. The `EnableHyperlinks` property has not been set. This property is in the dataset designer in C/SIDE, not in Visual Studio.

You need something to link to before you can create a hyperlink in the layout. There are three ways to create a link in Dynamics NAV:


- Using the `GETURL` function
- Using a bookmark
- Using a filter

Using a filter

I will start with the third option, using a filter. Although this is simple and light on resources, I prefer not to use it, because you don't need to add anything to the dataset. I prefer to avoid it because there's not much documentation on how to set it up, and it does not work in all the versions of Dynamics NAV.

The idea is to build a URL that looks as follows:

- `DynamicsNAV://server/instance/company/runpage?page=xx`
- `DynamicsNAV://server/instance/company/runreport?report=xx`
- `DynamicsNAV://server/instance/company/runquery?query=xx`
- `DynamicsNAV://server/instance/company/runxmlport?xmlport=xx`
- `DynamicsNAV://server/instance/company/runcodeunit?codeunit=xx`
- `DynamicsNAV://server/instance/company/RunTable?Table=xx`

	<p>xx: stands for the ID of the object you would like to link to server: is the server on which the service tier is running instance: is the name of the service tier company: is the name of the company</p>]
---	---	---

You can simply add this URL to the `Action` property as a text string via the **Go To URL** option. When you create a hyperlink you normally want to link to a specific record or filter the object that you are opening. This can be achieved by adding extra parameters to the link. If you want to link to another report, for example, you can append the following string:

```
&filter=<table>.<field>:<value>
```

So the complete URL would be:

```
DynamicsNAV://server/instance/company/runreport?  
report=xx&filter=<table>.<field>:<value>
```

The explanation of the terms used in the URL is as follows:

- **Table:** is the name of the data item of the report you are linking to
- **Field:** is the data item field you are linking to, to which you want to apply a filter
- **Value:** is the filter you want to apply to the data item field of the report you are linking to

If the field name contains special characters such as, for example, spaces or dots, then you must enclose it in quotation marks by using an escape character, %22

The value that you specify can also contain wildcards, as in:

- > (greater than)
- >= (greater than or equal to)
- < (less than)
- <= (less than or equal to)
- <> (not equal to)
- * (anything that begins with this)

You can also filter on multiple fields. Concatenate (or glue) the filters together in the URL, as in the following example:

```
DynamicsNAV:////runreport?report=xxx&filter=Customer.  
City:A*&filter=Customer.Name:B*
```

We also have access to the dataset because we are building these URLs in the `rdlc` layout, so the values on which we are filtering will come from the report dataset.

Tips and Tricks

The following screenshot shows an example of implementing this type of link and filtering the link from one report to another:

The screenshot illustrates the implementation of a hyperlink in a report design and its effect in the print preview. The top part shows the report design interface with a table containing a hyperlink field. The 'Text Box Properties' window is open, showing the 'Action' tab where 'Go to URL' is selected. The 'Expression' window shows the URL: '=DynamicsNAV:////runreport?report=112&filter=Customer.%22No.%22:' & Fields!No_Customer.Value'. The bottom part shows the print preview of two reports: 'Customer - Top 10 List' and 'Sales Statistics'. A red arrow points from the 'Sales (LCY)' column in the first report to the 'Sales Statistics' report in the second, where the 'Customer: No.: 10000' filter is visible.

No.	Name	Sales (LCY)	Balance (LCY)
10000	The Cannon Group PLC	17,100.96	168,364.41
43687129	Designstudio Gmunden	14,498.04	14,498.04
49858585	Hotel Pferdese	14,450.00	14,450.00
47563218	Klubben	11,772.20	11,772.20
20000	Selangorian Ltd.	6,510.64	96,049.99
30000	John Haddock Insurance Co.	6,142.90	349,615.40
49525252	BeefHouse	6,000.00	12,346.16
49633663	Autohaus Mielberg KG	4,331.40	8,188.80

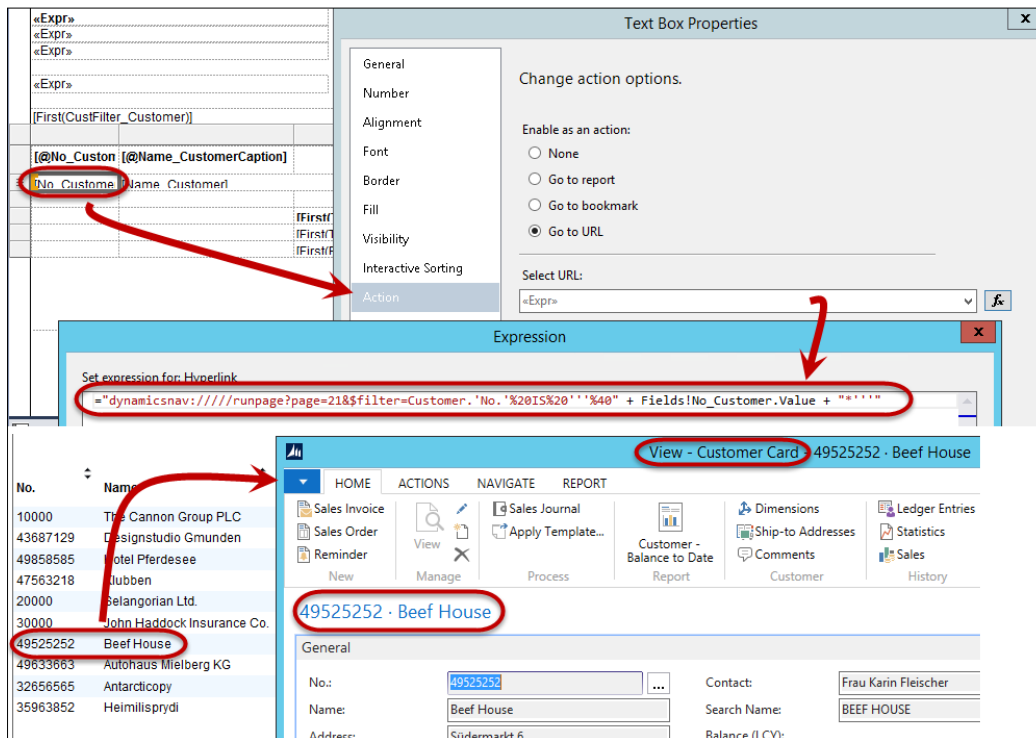
No.	Name	...	before	01/26/17	02/25/17
10000	The Cannon Group PLC	17,100.96		0.00	
	Sales (LCY)	11,762.70		0.00	
	Original Costs (LCY)	5,338.26		0.00	
	Original Profit (LCY)	31.2		0.0	
	Adjusted Costs (LCY)	7,137.20		0.00	
	Adjusted Profit (LCY)	9,963.76		0.00	
	Adjusted Profit %	58.3		0.0	
	Cost Adjmt. Amounts (LCY)	-4,625.50		0.00	
	Inv. Discounts (LCY)	727.34		0.00	
	Pmt. Discounts (LCY)	0.00		0.00	
	Pmt. Disc Tol. (LCY)	0.00		0.00	
	Pmt. Tolerances (LCY)	0.00		0.00	

In the preceding screenshot, the hyperlink was implemented in the **Customer - Top 10** report (111). When you click on **Sales (LCY)**, the report 112 is opened, filtered on the number of the customer that was clicked.

As you can see, this is a very simple way to provide a link to a report, allowing you to filter the report you are linking to.

When you want to link to a page and filter the page you are linking to, as in the previous example, the syntax of the URL is a little different as DynamicsNAV:////runpage?page=<page id>&\$filter='<field>'%20IS%20'<value>' [%20AND%20'<field>'%20IS%20'<value>'].

I have implemented this in the same report as an example, behind the customer number, as follows:



So, to summarize, you are able to create a link, containing a filter, to a report or page by using the `&filter` and `&$filter` options. This was not possible in previous versions of Dynamics NAV. The `&filter` option could not be used on page objects, only on reports.



The difference between `&filter` and `&$filter` is when you want to filter a report or a page. The filter to a page is similar to a filter for an ODATA web service, which requires the \$ sign. More information about these types of filters is available at <https://msdn.microsoft.com/en-us/library/dd338628.aspx> and [https://msdn.microsoft.com/en-us/library/hh169248\(v=nav.80\).aspx](https://msdn.microsoft.com/en-us/library/hh169248(v=nav.80).aspx).

You need to provide a bookmark to link and apply a filter to a page object. This is actually still an option in the current version of Dynamics NAV.

Using a bookmark

But first you need to calculate or get the bookmark and this requires some C/AL magic. A bookmark is no more or less than the ID of the record you want to link to, and this ID can be retrieved with the `RecordId()` function. This function is only available in a variable of type `recordref`, so you will need to define one and then use the `GetPosition()` and `SetPosition()` functions to connect it to the current record. You then need to format it using the `Format()` function.



An example of this report is available in the object: Customer - Top 10 List

The following example will shed more light on this procedure.

First, I added the following code to **Report 111 Customer - Top 10 List**:

The screenshot shows the 'Report Dataset Designer' window for 'Report 111 Customer - Top 10 List'. A table lists data sources and columns. The 'CustomerBookMark' column is highlighted with a red circle, showing the formula `FORMAT(RecRef.RECORDID,0,10)`.

Below, the 'C/AL Editor' shows two code snippets. The first snippet, `Integer - OnPreDataItem()`, includes the following code (highlighted with a red circle):

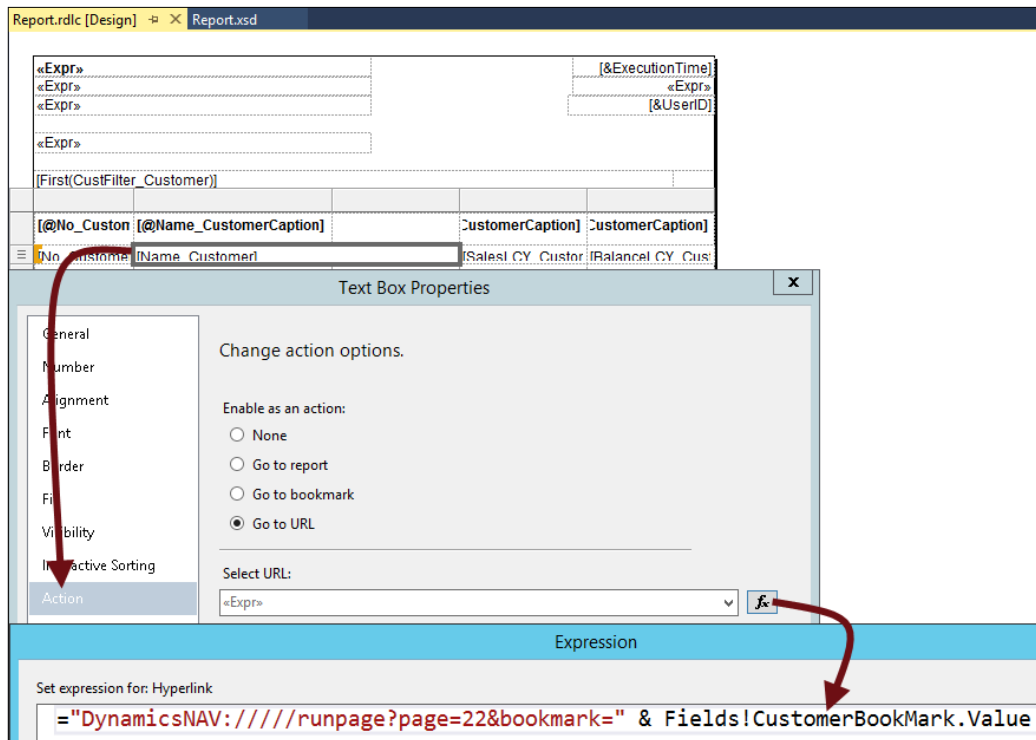
```
Integer - OnPreDataItem()  
CustSalesLCY := Customer."Sales (LCY)";  
CustBalanceLCY := Customer."Balance (LCY)";  
Window.CLOSE;  
CurrReport.CREATETOTALS(Customer."Sales (LCY)", Customer."Balance (LCY)");  
//***  
RecRef.OPEN(18);  
//***;
```

The second snippet, `Integer - OnAfterGetRecord()`, includes the following code (highlighted with a red circle):

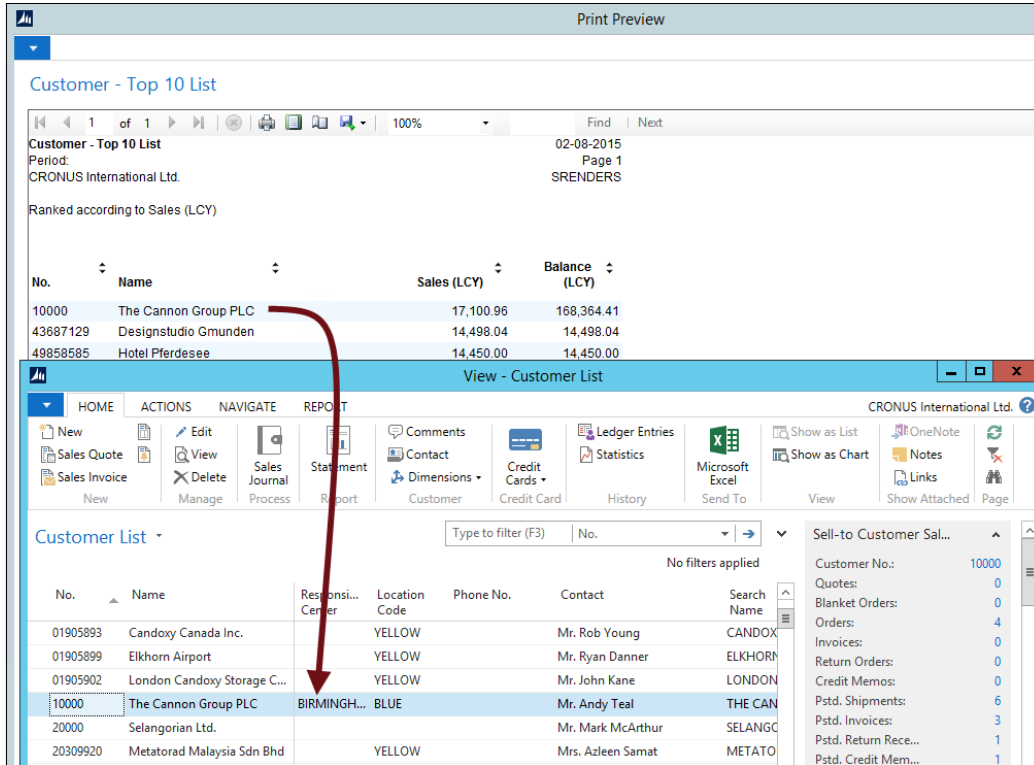
```
Integer - OnAfterGetRecord()  
IF Number = 1 THEN BEGIN  
IF NOT CustAmount.FIND('-') THEN  
CurrReport.BREAK;  
END ELSE  
IF CustAmount.NEXT = 0 THEN  
CurrReport.BREAK;  
CustAmount."Amount (LCY)" := -CustAmount."Amount (LCY)";  
Customer.GET(CustAmount."Customer No.");  
Customer.CALCFIELDS("Sales (LCY)", "Balance (LCY)");  
IF MaxAmount = 0 THEN  
MaxAmount := CustAmount."Amount (LCY)";  
IF (MaxAmount > 0) AND (CustAmount."Amount (LCY)" > 0) THEN  
BarText := PADSTR(' ', ROUND(CustAmount."Amount (LCY)" / MaxAmount * 45, 1), '*');  
ELSE  
BarText := ' ';  
CustAmount."Amount (LCY)" := -CustAmount."Amount (LCY)";  
//***  
RecRef.SETPOSITION(Customer.GETPOSITION());  
//***
```

This allows you to add the `FORMAT(RecRef.RECORDID, 0, 10)` function to the dataset add column, `CustomerBookMark`.

You then add the link to the layout in Visual Studio, in the **Action** property of the textbox that holds the customer name:



Then, when you run the report and click on the name, the corresponding page opens:



The advantage of using a bookmark is that it works in all versions of Dynamics NAV (that support RDLC). The disadvantage is that you need to use a *recordref* variable, and add the bookmark to the report dataset, which might have an impact on performance.

Using the GETURL() function

The third approach to generate URLs is by using the `GETURL` function. This function was introduced in Dynamics NAV 2015 and offers a new way to generate a URL. Furthermore, the generated URL can be linked to a specific type of client. The syntax of the `GETURL()` function is as follows:

```
[String :=] GETURL(ClientType[, Company][, Object Type][, Object Id][, Record])
```

The `ClientType` is also an option and can be set as `Current`, `Default`, `Windows`, `Web`, `SOAP`, or `OData`. It is the type of client that you want the link to be opened in. The other parameters speak for themselves. In the following example, the `GETURL()` function is used to generate a URL, based on an option provided in the request page. Then, when you click on a customer in the chart, the corresponding page is opened in the requested client:


The image shows three windows from the Report Designer:

- Report 111 Customer - Top 10 List - Report Dataset Designer:** Shows a table with columns: `TotalSalesCaptionLbl`, `PercentofTotalSalesCaptionLbl`, `CustomerURL`, and `CustomerBookMark`.
- C/AL Editor:** Contains the following code snippet, with a red box highlighting the `Client` case:


```
Integer - OnAfterGetRecord()
IF Number = 1 THEN BEGIN
  IF NOT CustAmount.FIND('-') THEN
    CurrReport.BREAK;
  END ELSE
  IF CustAmount.NEXT = 0 THEN
    CurrReport.BREAK;
  CustAmount."Amount (LCY)" := -CustAmount."Amount (LCY)";
  Customer.GET(CustAmount."Customer No.");
  Customer.CALCFIELDS("Sales (LCY)", "Balance (LCY)");
  IF MaxAmount = 0 THEN
    MaxAmount := CustAmount."Amount (LCY)";
  IF (MaxAmount > 0) AND (CustAmount."Amount (LCY)" > 0) THEN
    BarText := PADSTR('', ROUND(CustAmount."Amount (LCY)" / MaxAmount * 45, 1), '*');
  ELSE
    BarText := '';
  CustAmount."Amount (LCY)" := -CustAmount."Amount (LCY)";


  /**
  CASE Client OF
  Client::Windows:
    CustomerURL := GETURL(CLIENTTYPE::Windows, COMPANYNAME, OBJECTTYPE::Page, 21, Customer);
  Client::Web:
    CustomerURL := GETURL(CLIENTTYPE::Web, COMPANYNAME, OBJECTTYPE::Page, 21, Customer);
  Client::Current:
    CustomerURL := GETURL(CLIENTTYPE::Current, COMPANYNAME, OBJECTTYPE::Page, 21, Customer);
  END;
  /**
```
- Chart Data and Series Properties:** Shows a bar chart with two bars labeled "No Customer A" and "No Customer B". The **Series Properties** dialog is open, showing the **Action** tab. The **Enable as an action:** section has **Go to URL** selected. The **Select URL:** field contains `[CustomerURL]`.

When you run the report in a Windows, Web, or Tablet client, the links will work and you can link to the Windows, Web, or Tablet client.

 The Tablet client does not have a report viewer, but you can export a report to Excel and then use the hyperlinks in Excel.

Personally, I prefer to work with the `GETURL` function, as opposed to the `&Filter` or `&BookMark` options. The link is very simple in the `RDL` layout and does not require a complex expression. You can also link to different client types and, if you export a report to Excel, the links still work. A disadvantage is that the URL is added to the dataset and so consumes resources. I sometimes also include an extra option in the options tab so that the user can enable/disable the links.

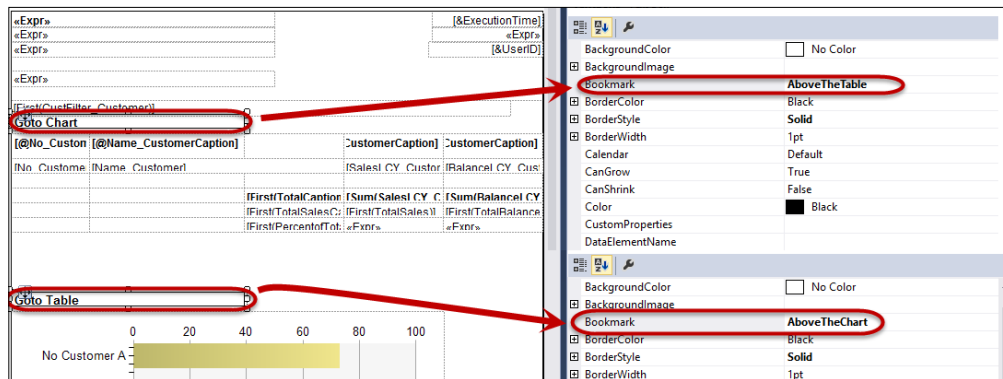
When you include links in reports remember that, although the user can click on the textbox or chart, the link itself is not underlined or in blue, as in a browser. You might therefore want to underline the text and make them blue, to make it clearer to the user where they can click.

 An example of working with the three types of URLs is available in the object: Customer - Top 10 List

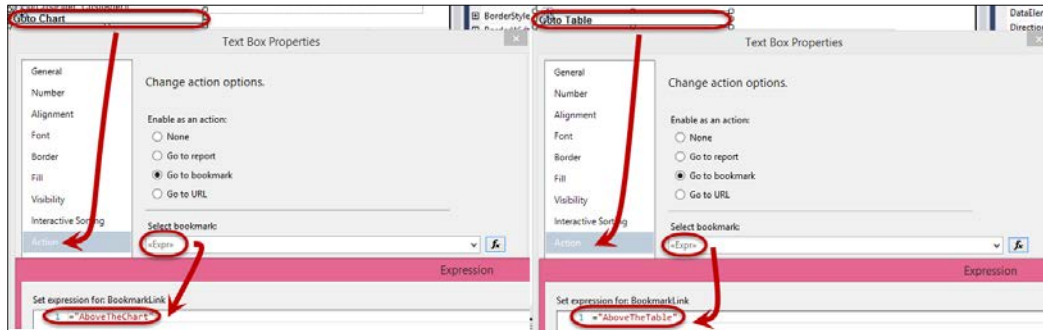
Using internal bookmarks

Another feature is internal bookmarks. You can use internal bookmarks to create a link that points to another location in the same report.

I have added two textboxes to the Customer-Top 10 report, 111 in the following example:



I have entered some text in the **Bookmark** property of the two textboxes: `AboveTheTable` and `AboveTheChart`. You can activate a `Goto Bookmark` link and link to the bookmarks that you have just created in the **Action** properties of the two textboxes, as follows:




Then, when you run the report and click on one of the textboxes, you will be redirected to the other textbox and vice versa.

Bookmark and **Select bookmark** are strings or hardcoded text, but they can also be expressions. For example, if you had two tables in a report, and would like to link the detail level of one table to the detail level of another table, you could use expressions to accomplish this.

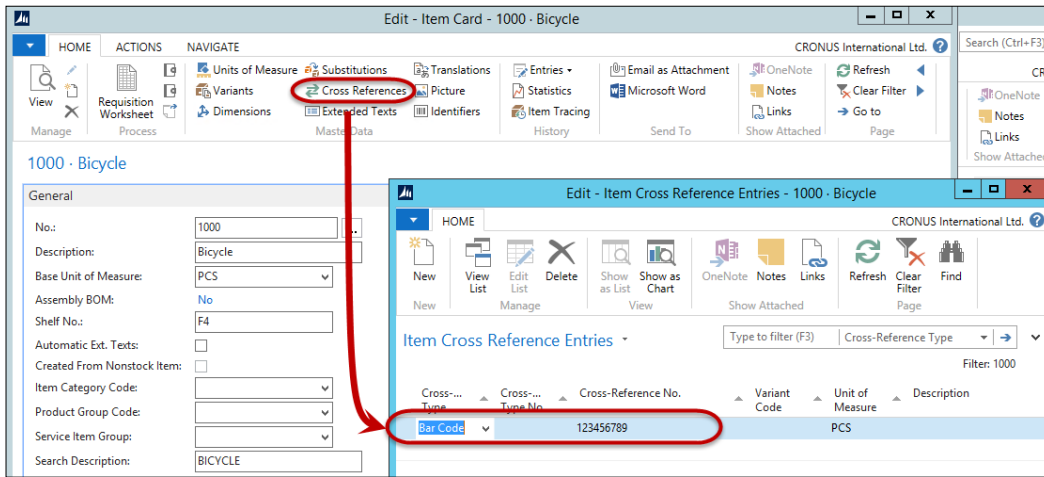
Printing barcodes

You might need to print bar codes when you are designing reports that are used in warehouse management, for example. Printing a barcode is actually very simple, depending on the type of barcode, of course. In its most simple form, a barcode is no more or less than a type of font. A barcode can be used to visualize numeric or alpha-numeric characters using vertical stripes of different widths. This information can then be scanned with barcode scanners.

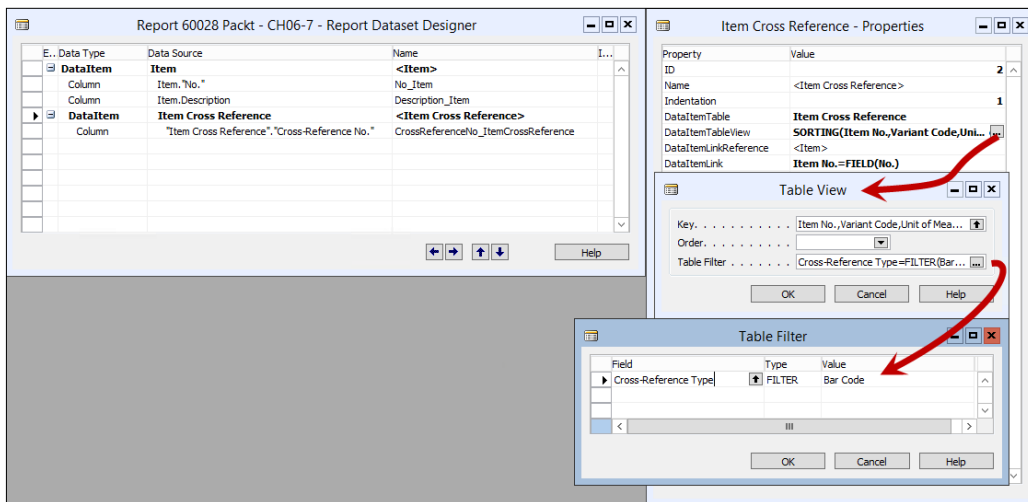
This method of processing information can improve performance. For example, when you are a warehouse employee and you need to pick items from different bins in the warehouse to prepare shipments, then you also need to enter this information in your ERP system, in our case Dynamics NAV. An item number can be made up of many characters, and a lot of time is lost typing those numbers. You can simply scan the number with a barcode scanner and it will be recognized by the system.

[ Did you know that it's the whitespace in between the vertical stripes that represents the barcode data and not the vertical stripes themselves, as most people believe?]

Let's have a look at an example. In Dynamics NAV you can register barcodes for items as cross-references. In an item card, select the cross reference action, as shown in the following screenshot:



Next, I'm going to create a new report to display items with barcodes. You can use the following dataset to get started:



I will print the barcodes in the layout as labels. You can then print them and get more information on the page, and also print them on sticky paper so that you can use the stickers to put on your bins. You need to go to **Report** properties and use the **Columns** setting:

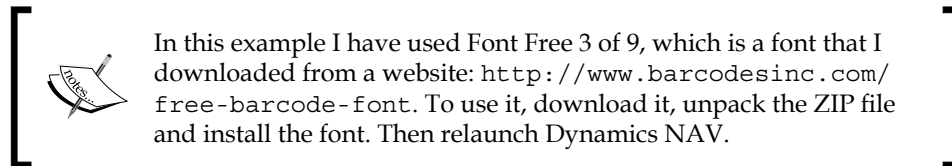


This will add three columns to the report. Only the first column is enabled in the body. At runtime, the information will continue on the other columns.

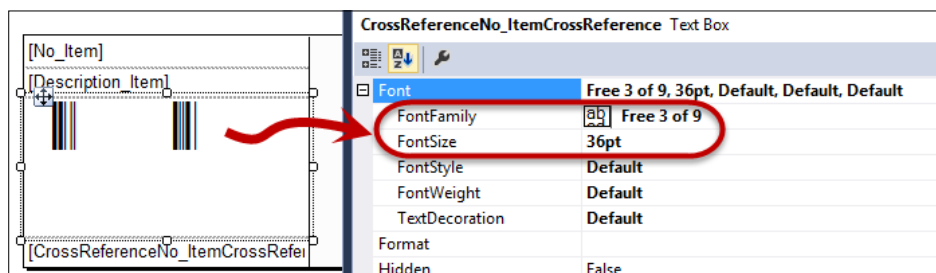
I then added a list to the body and added four textboxes to it. I will display the following descriptions in the textboxes:

- Item No
- Item Description
- Barcode (visually)
- Barcode (numerically)

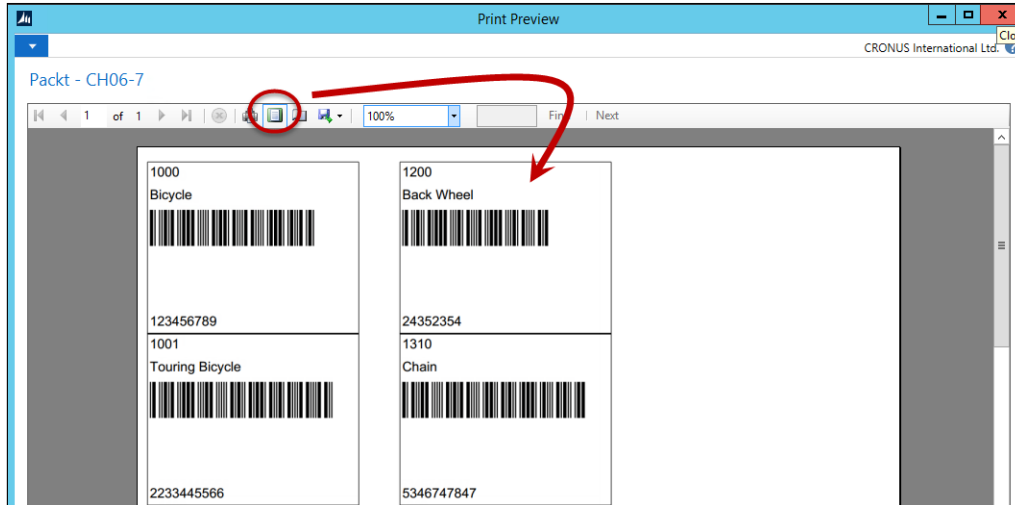
A barcode is actually a font that you need to select in the textbox properties.




Once you have downloaded the font, you need to install it to make it available in the list of fonts.



Now, when you run the report, it will look as follows:



Note that the columns are not visible when you run the report in preview mode, you need to select **Print Layout**, or export the report to PDF, Word, Excel, or Print for the columns to be visible. You can only see two columns in this example, although I defined three columns at design time. This is because there are not enough items in my database that have barcodes.

[ **Barcode fonts** You can download or purchase barcode fonts from several vendors online. Simply perform an online search and I'm sure you will find the font you require.]

Report templates

When you are charged with building a lot of reports it typically involves a lot of work that you always need to repeat. Having a template available that you can reuse can drastically cut down development time. Dynamics NAV and Visual Studio both have options available that you can use to create and reuse templates. Templates can be report parts or even complete pre-built reports.

One option, perhaps the best option, is to create a dummy report that contains the look and feel of the reports that you need to develop, because it is in the database, and so not bound to the filesystem or workstation that you are working on. For example, the header and footer of most document reports always contain the same information from the same tables, such as company information, including addresses and logos, and/or the number of copies option. You can create one report with a simple dataset, for example with an integer data item, and then build the report layout based on that dataset. Then, save the report as an object in the object designer. Then, every time you need to create a report with this basic layout, you can start from the dummy report, design it and save it under another number and name. After that, you only need to add the specific tables, bound to the current report, to the dataset and finish the layout.

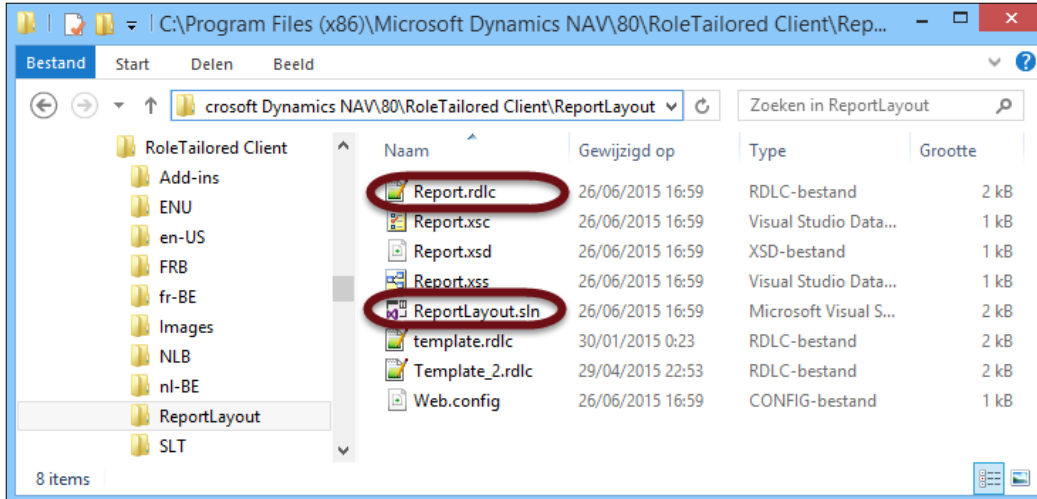


Although this is a very simple way to create a template report, unfortunately I don't see a lot of companies that use this approach. For some reason, most developers always use the start-from-scratch method. I recommend spending a little bit of time in creating one, or more, of these template reports and then reusing them, as it will save time, and the time you save you can spend on the specific things you need to create in your reports.

Another approach is to work with the report layout folder. Every time you create a new layout for a report, Dynamics NAV uses a template for the `report.rdlc` file. Why not modify this `report.rdlc` file and add all the necessary functions and layout parts to it? The only downside to this approach is that this template does not contain a dataset, so you can only use non-data bound fields or report parts.

The template is available in the folder where the Role Tailored Client and Development Environment are installed. So, remember that this template is bound to the workstation on which you are developing, and is not saved in the database.

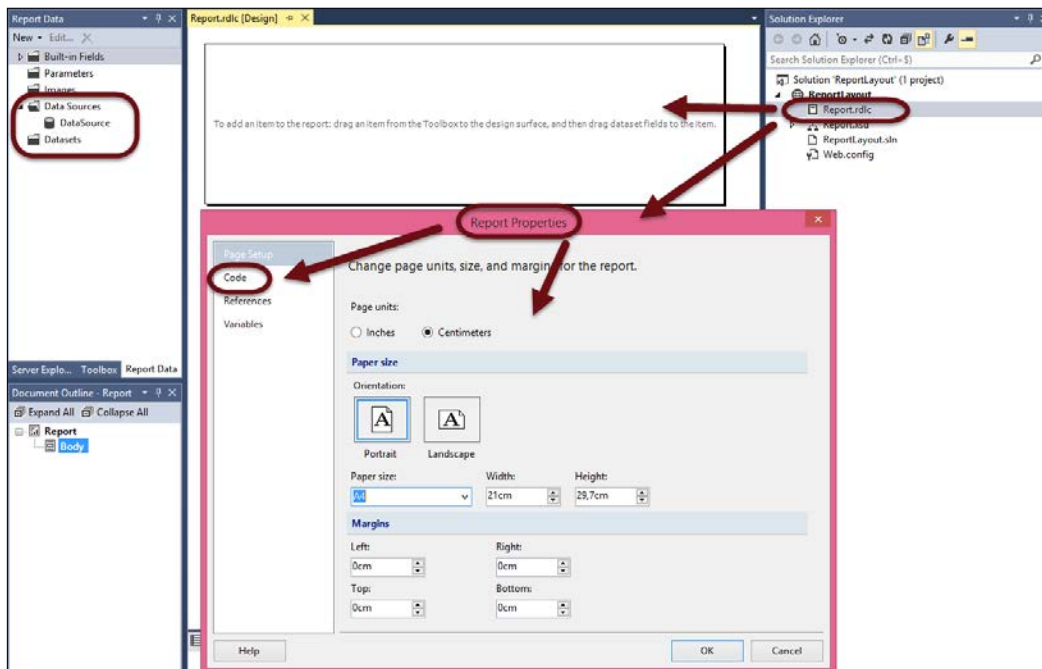
If you modify the template, you will need to manually deploy it to all the other developers in your company if you want them to be able to use it.



You can see the template in the preceding screenshot. The folder is: C:\Program Files (x86)\Microsoft Dynamics NAV\80\RoleTailored Client\ReportLayout and the template is the report.rdlc file. To modify it, you can open the ReportLayout.sln file in Visual Studio, which is the solution file.



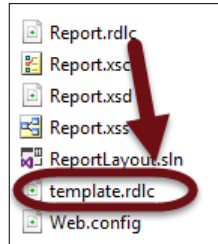
Be careful to use the correct version of Visual Studio. If you use a newer version of Visual Studio, it may ask you to upgrade. In that case, don't do it, because it will cause errors whenever you need to create a new report layout in Dynamics NAV. I therefore recommend backing up this folder, so that you can revert back to a stable version if something goes wrong.



When you open the template in Visual Studio, you can go to **Report Setup** and change the default settings for pagination such as height, width, margins, page size, and so on. You can also open report **Code** tab and add your own functions to it. It's even possible to add a page header and footer, with embedded images. You can access the toolbox in the body and add Tablixes and other controls to it. The only thing you don't have is a dataset. So, basically, you can prepare a default report layout using unbound report controls and properties.

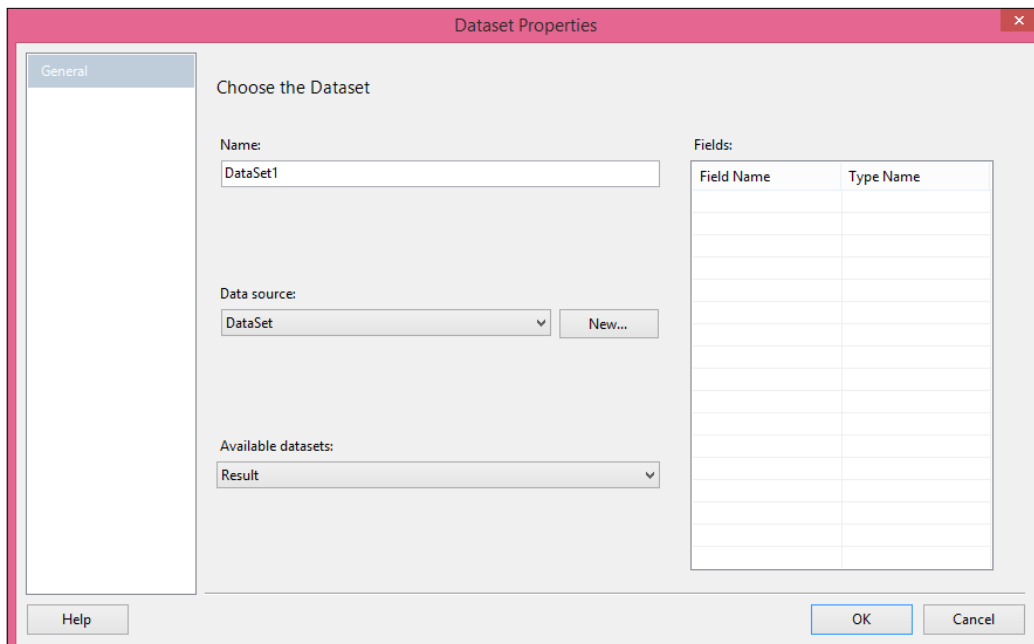
There's a third way to reuse report parts or create templates. It is actually similar to the above mentioned method but, instead of modifying the `report.rdlc` file, you need to copy/paste the file and rename it as, for example, `template.rdlc`. The next time you open the layout for a report, you will then see the `template.rdlc` file in the solution explorer.

So, first you copy, paste and rename the `report.rdlc` file as `template.rdlc`:

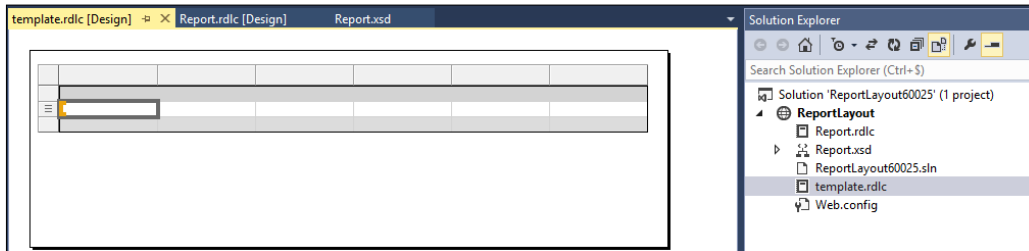


Then you create a new report, or open an existing report and open the layout in Visual Studio. Then, open **Solution Explorer** and open the `Template.rdlc` file.

When you try to add a Tablix to it, you will see the following window, in which you can bind the Tablix to the dataset:



Then, you can add rows, columns and apply formatting to the Tablix, as shown in the following example:



You can add multiple tables to the report body and apply formatting properties. For example, you can pre-set the font, font weight, font size, textbox or row height and width, the padding (left, right, top, and bottom), and so on. Setting these properties always takes a lot of time, so predefining them in the template will reduce report development time.

You can also pre-set the values for dynamic properties, using expressions like changing the background color of even and odd rows, using the following expression:

```
=Iif(LineNumber(Nothing) MOD 2,"even color","odd color").
```

You can then define the `GetData` and `SetData` functions, variables, and so on, for the document reports. When you are ready, save the file using `Ctrl + S` or the **Save** button.

You will then see `Template.rdlc` in **Solution Explorer** when you are in the layout of any report, using Visual Studio. You can then open the `Template.rdlc` file, copy the Tablix and paste it into the `Report.rdlc` file. Now you can reuse report parts that contain preformatted properties. You can even create multiple `Template.rdlc` files and add them to the `Report Layout` folder.

Your template report can also contain a report header and or footer in which you can predefine controls to hold the company logo, address and page numbers. I recommend putting these fields into a Rectangle, so that, when you want to copy it, you only need to copy and paste the rectangle.



There are many websites and blogs about Dynamics NAV where you can download similar templates. This is Claus Lundstrum's blog: <http://mibuso.com/blogs/clausl>.

Using a report setup table

Sometimes, when you create a report, you also create a request page, so that the user can set up the report layout according to options they specify in the **Options** tab of the request page. Similarly, you can also create a report setup table that contains information about the report layout. You can then create a page so that users can change the fields in this table, because the information is stored in a table. Then, when you create a report, you can query this table to fetch the information that you will use in the report layout.

There's actually already an example of this principle used in document reports. You might remember the company logo that can be hidden or displayed on the left, middle or top of a document. The field is stored in the `Sales and Receivables Setup` table and is named `Logo Position on Documents`. The logo is placed in the dataset and displayed on the left, middle or right of the report header depending on its value, in the `OnPreReport` trigger of most document reports, such as **Report 206 Sales - Invoice**.

The idea is to assemble all of these kinds of fields into a single report setup table and reuse it when you create a report I have also mentioned creating and using report templates in this chapter. The information you define in this report setup table is perfect to add to your report templates.

You can even take this idea to the next level and combine a template report with a report setup table. Think about the report header in most document reports. It almost always contains the same information such as the company information and logo and address fields, and information about the document. You might create a dummy table that contains all of those fields. For the purpose of this example let's name the table `Document Header`. Add some fields to the document header table such as a document number field and eight address fields. Then, create a report that uses the `Document Header` table as a data item and create a layout for the report that displays those fields, as in the header of a document report. Then, in the `C/AL` code of the report, all you need to do is write code that fetches the data from the real table and copies it over in our dummy document header table. After that, the layout does not require any modification.

Then, imagine, instead of using only a document header table, you also include a document line table, and so on. The layout of the report is built upon the dummy tables. The data is put into those tables via C/AL code. Whenever you need to create or run a document report, you only need to modify the C/AL code, not the layout.

Then, to make the report even more flexible you might build in some options. You can let the user decide which field needs to be displayed where in the report setup table. You need to take into account the information stored in the report setup table in the code that fills the buffer table. This is similar to the logo position on documents, but for other fields. You can use C/AL code or even expressions in the report layout to hide or show fields, rows, and tables, and so on, dynamically.

This might seem like a far-fetched idea and a lot of work, but you only need to do this once and then you can reuse this document template report for all your documents and all your customers. If you are a partner selling and implementing Dynamics NAV this will have a huge return on investment. In fact, I have come across some partners in Belgium and Denmark who have already created this type of document template.

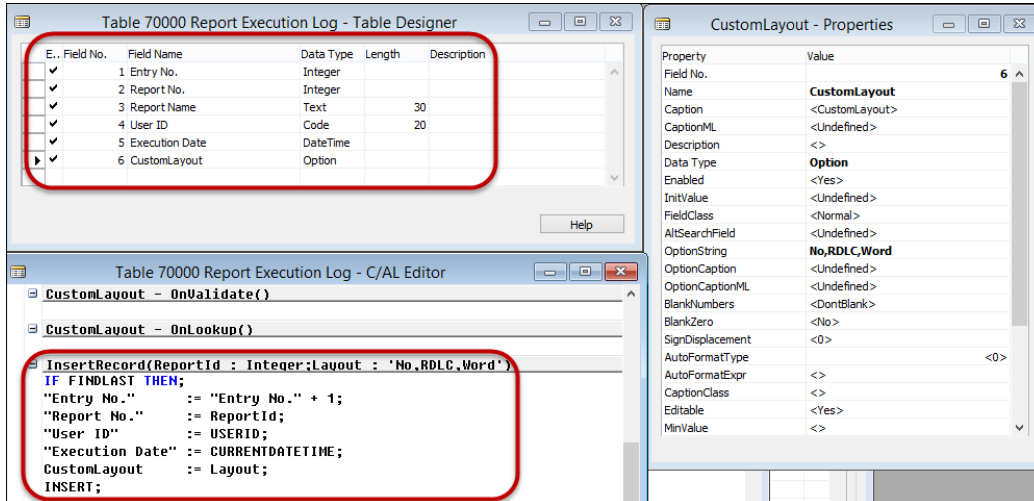


We will return to this principle of using a dummy table in more detail using an example in the next chapter, where we will use it as a performance improvement. This type of table is also referred to as a buffer table.

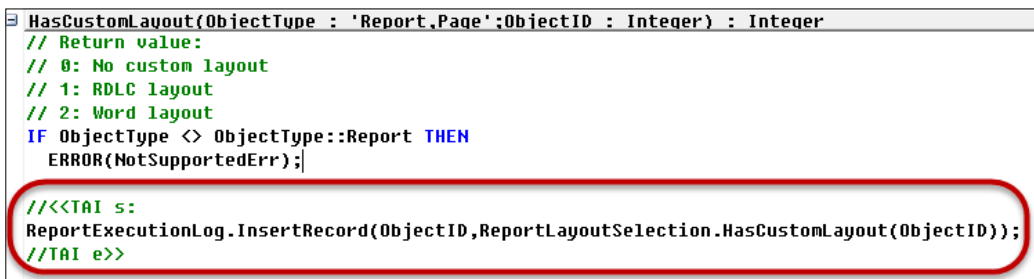
Report logging

Did you know that, every time you run a report, the system executes a function in code unit 1, application management to determine whether the report has a custom layout or only a built-in layout? You can use this to your advantage to implement report usage logging. In the following example, I will demonstrate how you can implement a log table that will keep track of who is running which report at what moment, and also which layout was used to render the report.

First, let's create a log table. The table is named **Report Execution Log**:



It contains fields which log who is running which report, at what moment, and the type of layout. The table also contains an `InsertRecord()` function that you can call to insert a record. Next, you need to open Codeunit 1, Application Management and look for the function named `HasCustomLayout()`. You can add the following code there:



Every time a report is executed it will be logged in this table, as you can see in the following screenshot:

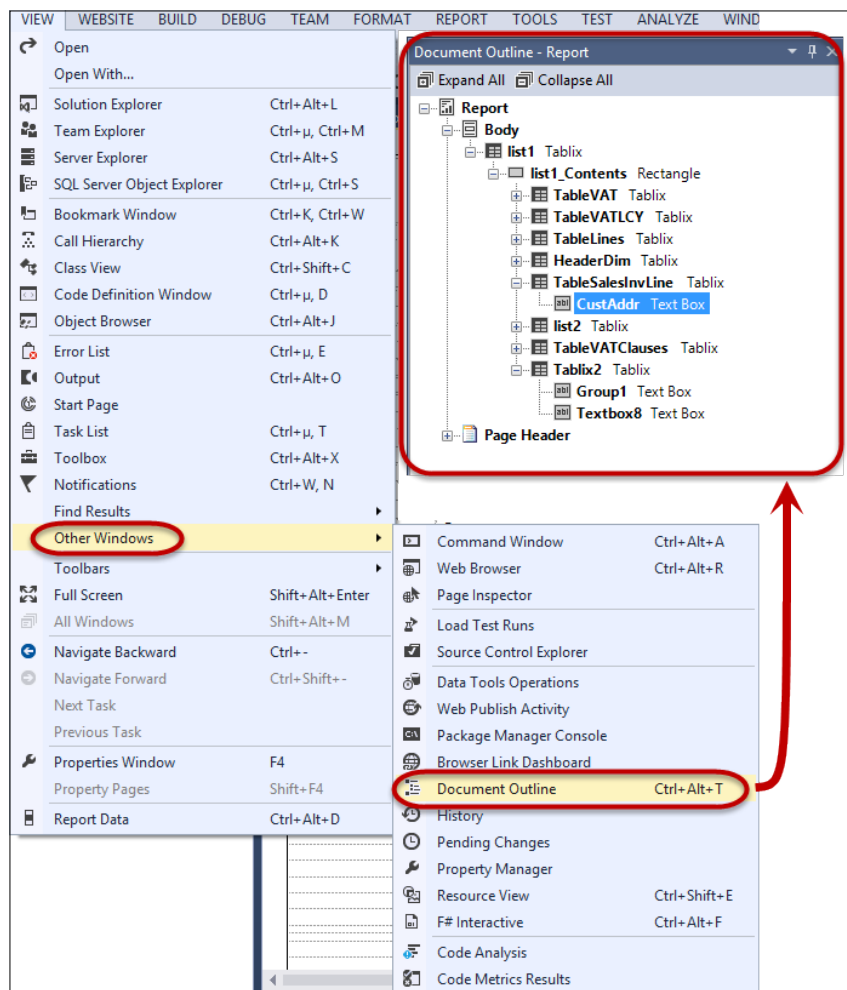
Report Execution Log						
Type to filter (F3)					Entry No.	→
No filters applied						
Entry No.	Report No.	Report Name	User ID	Execution Date	CustomLayout	
1	1306	Mini Sales - Invoice	STEVENR\STEVEN	8/04/2015 10:34	No	
2	1306	Mini Sales - Invoice	STEVENR\STEVEN	8/04/2015 10:38	No	
3	111	Customer - Top 10 List	STEVENR\STEVEN	8/04/2015 10:38	No	
4	1306	Mini Sales - Invoice	STEVENR\STEVEN	8/04/2015 10:48	No	
5	111	Customer - Top 10 List	STEVENR\STEVEN	8/04/2015 10:49	No	
6	111	Customer - Top 10 List	STEVENR\STEVEN	8/04/2015 13:33	No	
7	111	Customer - Top 10 List	STEVENR\STEVEN	8/04/2015 13:34	No	
8	111	Customer - Top 10 List	STEVENR\STEVEN	8/04/2015 13:35	No	
9	111	Customer - Top 10 List	STEVENR\STEVEN	8/04/2015 13:36	No	
10	60025	Packt - CH05-4	STEVENR\STEVEN	13/04/2015 8:50	No	
11	60025	Packt - CH05-4	STEVENR\STEVEN	13/04/2015 8:51	No	
12	60025	Packt - CH05-4	STEVENR\STEVEN	13/04/2015 8:52	No	
13	60025	Packt - CH05-4	STEVENR\STEVEN	13/04/2015 8:52	No	
14	1306	Mini Sales - Invoice	STEVENR\STEVEN	13/04/2015 11:35	No	
15	1306	Mini Sales - Invoice	STEVENR\STEVEN	13/04/2015 11:36	No	
16	60026	Packt - CH05-5	STEVENR\STEVEN	13/04/2015 12:11	No	
17	60026	Packt - CH05-5	STEVENR\STEVEN	13/04/2015 12:17	No	
18	60026	Packt - CH05-5	STEVENR\STEVEN	13/04/2015 12:18	No	

This is proof of concept implementation, but you can imagine how you could modify this to log even more information. You can also build a report using this table as a data source, and display the reports that are executed the most. This kind of information is very useful when you are confronted with an upgrade and need to determine which reports should be upgraded first.

The fixed header problem

I have seen this a couple of times and therefore wanted to mention it here briefly. When you print a report, for example a document report, the report might be printed for multiple records (documents, customers, and so on) at a time. If you see that all the headers contain the same information, it could be down to two causes.

First, you might not be using the `setData` function correctly. A different version of RDLC is used depending on the version of Dynamics NAV. In RDLC 2010, which is, at the time of writing, the latest version, if a textbox is hidden, then the expression for the value property is not executed. If that's the case then your header will be blank. If the header is not blank but always contains the same information then check if the textbox that executes the `setData` function is in the correct container. The list usually contains the groupings on the document number and output number. You can use the document outline tool to do this. Select **View, Other Windows** in the menu, and then **Document Outline**. The document outline will give you a tree view look at the layout of the report:



If the `custaddr` textbox or, in this case, `TableSalesInvLine` is moved outside `List1`, then the `SetDate` function is only executed for the first record and so the header will always contain static information. In the case of invoices this means they will all be sent to the same customer and that's not a good idea.

Now, even if you do everything right, then it will go wrong if you export the report to Word. Word does not support dynamic headers. The report header in the `rdlc` layout is transformed into a header in the Word document that is generated. This Word page header can only display static text and it will show the first page's information on every page:

The screenshot shows a software interface for a 'Posted Sales Invoice' report. A filter dialog box is open, showing a 'Where' clause: 'No. is 103001|103002'. The print menu is open, showing options for 'Print...', 'PDF', and 'Microsoft Word'. Below the interface is a preview of the generated Word document header, which is a static snapshot of the first record's data.

No.	Sell-to Custom...
103001	10001
103002	20000
103003	30000
103004	50000
103005	49525252
103006	49525252
103007	49858585
103008	49858585
103009	49858585
103010	49633663
103011	43687129

Filter dialog box:

Show Assembly Components:
 Show Additional Fee Note:
 Posted Sales Invoice
 Show results:
 X Where No. is 103001|103002
 X And Sell-to Customer No. is Enter a value.
 X And No. Printed is Enter a value.
 + Add Filter

Print menu:

Print...
 PDF
 Microsoft Word

Word document header:

Verkoop - Factuur
 Pagina 1 van 1
 CRONUS BELGIË NV
 De Ring 5
 Zone II
 2800 MECHELEN

Van Ierp Kantoorinrichting
 Dhr. Kevin Verboort
 Geinplantsoen 2
 2800 MECHELEN
 België

Factureren aan: 10001
 Ondernemingsnr.: 0996000057

Factuurnr.: 103001

Boekingsdatum: 25. januari 2016
 Vervaldatum: 25. februari 2016
 Documentdatum: 25. januari 2016
 Betalingsvoorwaarden: 1 maand/2% 8 dagen
 Verzendwijze: Af magazijn
 Verzenddatum: 25/01/2016
 Inclusief btw: Nee

Telefoon: +32 (0)2 424 64 60
 E-mail: info@thinkaboutit.be
 Startpagina: 0058.315.707
 Ondernemingsnr.: 888-9999
 Giro nr.: Rabobank
 Bank: 99-99-888
 Rekeningnr.: Koos Splinter
 Verkoper:

This is the case when you run a report that has an `rdlc` layout and then export it to Word.



I'm not sure if this is a bug or if it's by design, but it has been the case since Dynamics NAV 2009 and it's still so in version 2015.

Summary

In this chapter we have learned some tips and tricks, such as working with hyperlinks, displaying header or footer information on specific pages, and transheaders and footers. I will explain how you can keep the performance of a report as fast as possible in *Chapter 7, Performance Optimization Techniques*.

The next chapter covers very important concepts and recommendations on how you should build your dataset and layout, so that your reports remain at optimum performance at all times.

7

Performance Optimization Techniques

This chapter is all about performance optimization techniques. I will start with some recommendations about how you can build the dataset and layout of a report so that it will remain as efficient as possible. Then I will introduce you to techniques such as using a buffer table and query when building the dataset of your report. It will involve some preparation but, in the end, you will benefit from the advantages that this approach brings.

Performance recommendations

When you create a report, there are always two phases, the construction of the dataset and the creation of the layout. I will provide some recommendations for the report development process with performance in mind.

Although you can probably achieve your goal without following these guidelines, you will notice that, after a while, when the database grows and more users run the system, things get slower. That is why it is important to always have performance at the forefront. The idea is to prevent problems occurring, instead of having to solve them when they eventually do.

The dataset

When you create your dataset, you can make changes without major consequences. If your report has a layout already then modifying the dataset will have a big impact and, as a result, you will probably also need to spend some time updating the layout accordingly.

There are a couple of techniques you will need to use:

- Optimize the C/AL code for the report. For example, use `currreport.skip`, `CALCSUMS`, and use variables instead of data items
- Reduce the number of columns
- Reduce the number of rows
- Apply filters in the request page
- Use the job queue to generate the report on the server
- Use the report inbox/scheduling to generate the report on the server
- Use the `startsession` function in a separate code unit to generate the report in a separate session on the server, and then export it to Excel, PDF, or Word
- Use a code unit to run the report and save it to PDF, Excel, or Word

Captions and labels

Using the include caption property and the labels; functionality is an easy way to add multi-language functionality to your report.

If multi-language means generating the report in the language of the user, then use captions and labels. If it means generating the report in the language of the recipient, then you need to add the captions to the dataset. (Multi-language was discussed in *Chapter 5, Document Reports*.)



If a field does not need to be translated or if the caption is not used in the report layout, then don't add it. I often see reports where all fields have the include caption property enabled, just in case. Of course, although minimal, this will have a negative impact on performance because the report will require more resources.

When a report is upgraded from a previous version of Dynamics NAV, then all captions, by default, are in the dataset. You should take the time to move those captions to the parameters of the report (using include captions and labels). This will remove the caption fields from the dataset at runtime and so improve performance.

An example of such a report is **Report 1 Chart of Accounts**. Although the version number of the report, *NAVW17.10*, implies it has been developed for version 7.1, which is 2013 R2, it does not use captions or labels, but sends the captions to the dataset.



There are many more reports similar to this, In order to find them, put a filter in the object designer, in the version list column and look for: @*NAVW17.10*.

Also in the C/AL code of the report there's isn't any code that changes the language, as with document reports:

```
CurrReport.LANGUAGE := Language.GetLanguageID("Language Code");
```

This is an example of the preceding code:

E.. Data Type	Data Source	Name
DataItem	G/L Account	<G/L Account>
Column	COMPANYNAME	COMPANYNAME
Column	CurrReport.PAGENO	CurrReport_PAGENO
Column	TABLECAPTION+' '+GLFilter	G_L_Account__TABLECAPTION____GLFilter
Column	GLFilter	GLFilter
Column	Chart_of_AccountsCaptionLbl	Chart_of_AccountsCaption
Column	CurrReport_PAGENOCaptionLbl	CurrReport_PAGENOCaption
Column	FIELDCAPTION("No.")	G_L_Account__No__Caption
Column	PADSTR____G_L_Account__Indentation__2__G_L_Account...	PADSTR____G_L_Account__Indentation__2__G_L_Account...
Column	G_L_Account__Income_Balance_CaptionLbl	G_L_Account__Income_Balance_Caption
Column	G_L_Account__Account_Type_CaptionLbl	G_L_Account__Account_Type_Caption
Column	G_L_Account__TotalingCaptionLbl	G_L_Account__TotalingCaption
Column	G_L_Account__Gen__Posting_Type_CaptionLbl	G_L_Account__Gen__Posting_Type_Caption
Column	G_L_Account__Gen__Bus__Posting_Group_CaptionLbl	G_L_Account__Gen__Bus__Posting_Group_Caption
Column	G_L_Account__Gen__Prod__Posting_Group_CaptionLbl	G_L_Account__Gen__Prod__Posting_Group_Caption
Column	G_L_Account__Direct_Posting_CaptionLbl	G_L_Account__Direct_Posting_Caption
Column	G_L_Account__Consol__Translation_Method_CaptionLbl	G_L_Account__Consol__Translation_Method_Caption

Variables	Text Constants	Functions
	Chart of AccountsCaptionLbl	Chart of Accounts
	CurrReport_PAGENOCaptionLbl	Page
	PADSTR____G_L_Account__Indentation__2__G_L_Account...	Name
	G_L_Account__Income_Balance_CaptionLbl	Income/Balance
	G_L_Account__Account_Type_CaptionLbl	Account Type
	G_L_Account__TotalingCaptionLbl	Totaling
	G_L_Account__Gen__Posting_Type_CaptionLbl	Gen. Posting Type
	G_L_Account__Gen__Bus__Posting_Group_CaptionLbl	Gen. Bus. Posting Group
	G_L_Account__Gen__Prod__Posting_Group_CaptionLbl	Gen. Prod. Posting Group
	G_L_Account__Direct_Posting_CaptionLbl	Direct Posting
	G_L_Account__Consol__Translation_Method_CaptionLbl	Consol. Translation Method

As you can see, the report contains text constants as columns in the dataset.



The column names contain underscores, this is an indicator that it is an upgraded report because, column names contain underscores by default in Dynamics NAV 2009.

When the report runs, the dataset contains values for these captions on every line, and these are repeated on every record:

COMPANYNA	Cur.	G_L_Account_...	G...	Chart_of_Accounts...	CurRe...	G_L_A...	PADSTR...	G_L_Account_...	G_L_Account_...	G_L_Acc...	G_L_Account_...	G_L_Account_...	G_L_Account_...	G_L_Account_...	G_L_Account_...	PADSTR_...
RONUS Intern...	1	G/L Account	Chart of Accounts	Page	No.	Name	Income/Balanc	Account Type	Totalling	Gen. Posting T...	Gen. Bus. Post...	Gen. Prod. Post...	Direct Posting	Consol. Transla...	<>	<>
RONUS Intern...	1	G/L Account	Chart of Accounts	Page	No.	Name	Income/Balanc	Account Type	Totalling	Gen. Posting T...	Gen. Bus. Post...	Gen. Prod. Post...	Direct Posting	Consol. Transla...	1000	BALANCE SHEET
RONUS Intern...	1	G/L Account	Chart of Accounts	Page	No.	Name	Income/Balanc	Account Type	Totalling	Gen. Posting T...	Gen. Bus. Post...	Gen. Prod. Post...	Direct Posting	Consol. Transla...	<>	<>
RONUS Intern...	1	G/L Account	Chart of Accounts	Page	No.	Name	Income/Balanc	Account Type	Totalling	Gen. Posting T...	Gen. Bus. Post...	Gen. Prod. Post...	Direct Posting	Consol. Transla...	1002	ASSETS
RONUS Intern...	1	G/L Account	Chart of Accounts	Page	No.	Name	Income/Balanc	Account Type	Totalling	Gen. Posting T...	Gen. Bus. Post...	Gen. Prod. Post...	Direct Posting	Consol. Transla...	<>	<>
RONUS Intern...	1	G/L Account	Chart of Accounts	Page	No.	Name	Income/Balanc	Account Type	Totalling	Gen. Posting T...	Gen. Bus. Post...	Gen. Prod. Post...	Direct Posting	Consol. Transla...	1003	Fixed Assets
RONUS Intern...	1	G/L Account	Chart of Accounts	Page	No.	Name	Income/Balanc	Account Type	Totalling	Gen. Posting T...	Gen. Bus. Post...	Gen. Prod. Post...	Direct Posting	Consol. Transla...	<>	<>
RONUS Intern...	1	G/L Account	Chart of Accounts	Page	No.	Name	Income/Balanc	Account Type	Totalling	Gen. Posting T...	Gen. Bus. Post...	Gen. Prod. Post...	Direct Posting	Consol. Transla...	1005	Tangible Fix...
RONUS Intern...	1	G/L Account	Chart of Accounts	Page	No.	Name	Income/Balanc	Account Type	Totalling	Gen. Posting T...	Gen. Bus. Post...	Gen. Prod. Post...	Direct Posting	Consol. Transla...	<>	<>
RONUS Intern...	1	G/L Account	Chart of Accounts	Page	No.	Name	Income/Balanc	Account Type	Totalling	Gen. Posting T...	Gen. Bus. Post...	Gen. Prod. Post...	Direct Posting	Consol. Transla...	1100	Land and B...
RONUS Intern...	1	G/L Account	Chart of Accounts	Page	No.	Name	Income/Balanc	Account Type	Totalling	Gen. Posting T...	Gen. Bus. Post...	Gen. Prod. Post...	Direct Posting	Consol. Transla...	<>	<>
RONUS Intern...	1	G/L Account	Chart of Accounts	Page	No.	Name	Income/Balanc	Account Type	Totalling	Gen. Posting T...	Gen. Bus. Post...	Gen. Prod. Post...	Direct Posting	Consol. Transla...	1110	Land and ...
RONUS Intern...	1	G/L Account	Chart of Accounts	Page	No.	Name	Income/Balanc	Account Type	Totalling	Gen. Posting T...	Gen. Bus. Post...	Gen. Prod. Post...	Direct Posting	Consol. Transla...	<>	<>

You can see the following in the RDLC layout of the report:

{First(G_L_Account__TABLECAPTION____GLFilter)}															
{First(G_L_...	{First(PADSTR____G_L_Account__Indentation	{Fir	{First(G_L_...	{Fir	{First(G_L_Account__To	{First	{First	{First(G_...	{First(G_L_Accou...						

All textboxes use a First() expression in the table header, where the captions are shown, for example: =First(Fields!G_L_Account__No__Caption.Value). This means that only the captions of the first record in the dataset are used.

You can optimize this report in different ways:

- Replace the captions using labels or the include caption property
- Move the captions to a new, separate, integer data item, which is filtered to add only one row to the dataset

By doing so, you will remove a lot of information from the runtime dataset and, as a consequence, the report will use fewer resources and will be faster.

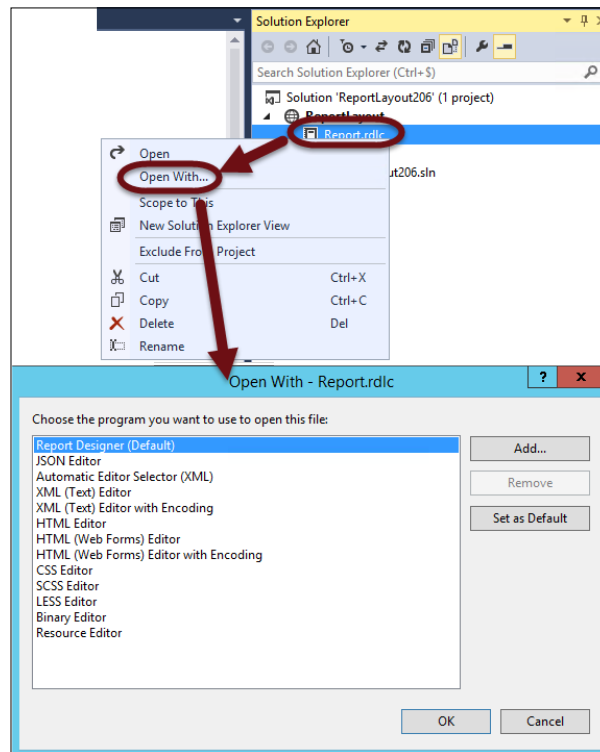
Remove unused columns

You always need to be aware of the columns, and only include the ones that are displayed in the report.

An example of this is that, when you look at a report upgraded from Dynamics NAV 2009 to a later version, you can see that there are a lot of columns in the dataset that aren't used anywhere in the layout of the report. These fields are in the dataset because they were used in the classic layout of the old report, but not in the RDLC layout.

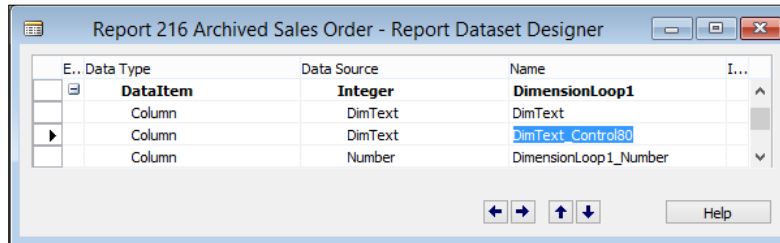
You can open the layout of the report in another editor to see if a field in the dataset is used or not.

The way to do this is to use **Solution Explorer** in Visual Studio. When you right-click on the `report.rdlc` file, you can open it in an editor of your choice, as shown in the following screenshot:

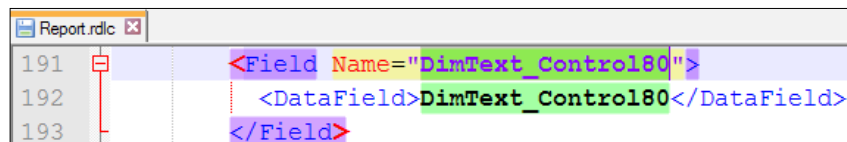


Then, in the editor, you can use the search and replace functions to change references to fields that have been removed or renamed in the dataset.


For example, in **Report 216 Archived Sales Order**, there's a field in the dataset named `DimText_Control180`, as you can see in the following screenshot:



When I open the RDLC layout in Notepad, and search for the field `DimText_Control180`, I see the following:



This indicates that the field, as expected, is a part of the dataset of the layout. But there's no other reference to this field in the RDLC file anywhere. This indicates that this field is not used in the RDLC layout, and thus should not be in the dataset, so I should simply remove it. This is a simple example of how you can figure out if a field in the dataset is referenced anywhere in the layout.

[ You can also use Notepad, or Notepad++, which is a free alternative with color recognition from <http://notepad-plus-plus.org>.]


As a precaution, I recommend making a backup of your report (or report layout) before you make modifications to the `rdlc` file with an external editor. If you make a mistake in the `rdlc` file you will no longer be able to edit it with Visual Studio.

Optimize the usage of BLOB fields

When you add a database image to the dataset of a report, then the data type of the field is a **Binary Large Object (BLOB)**. A BLOB datatype can potentially hold up to 2 GB of information. By default, a BLOB field that is added to every record in the dataset can inflate the size of the dataset dramatically. It should therefore not be available on every row of the dataset and you need to make sure you only add it once.

There are two ways of implementing a BLOB field, either you add it in the dataset and write C/AL code to clear it, or you add/move the field to a separate integer data item, at the beginning or the end, which only holds one row of data.

Once, when I was creating a document report for a customer, they were using a company logo with a size of about 500 KB. You might think that this is not so big, but when I ran the report it took about 30 seconds to render. Then I added some code to the dataset that cleared the image field after the first record and the report rendered in a few seconds. When you print one invoice, 30 seconds might seem acceptable, but when you are printing multiple invoices, it is not.

 Performance is all about perception. Once the users believe that the application is slow, you can implement performance improvements to speed up the application, but the perception will be that it is still slow. Getting it right from the beginning is the best way to avoid negative perceptions.

Let's look at an example of how you can optimize the usage of a BLOB field.

The idea is to have the logo, or BLOB field, only in one row in the dataset. There are two ways of doing this. The first is to use a separate integer data item, filtered on one row. If you are applying this technique in an existing report, however, it will involve retesting the layout and that can take some time, especially in document reports. Another technique is to leave the dataset as it is and add code to the triggers to clear the BLOB field after it has been added to the dataset in the first iteration. You can use the `CLEAR()` function to clear a field.

A nice example of this is available in **Report 1306 Mini Sales Invoice**. You will see the following code in the `OnPreDataItem()` trigger of the header data item:

```
FirstLineHasBeenOutput := FALSE;
```

Then, in the `OnPreDataItem()` trigger of the line data item, there's this code:

```
FirstLineHasBeenOutput := FALSE;
CompanyInfo.CALCFIELDS(Picture);
```


So here the BLOB field and picture, is calculated. Next, in the `OnAfterGetRecord()` trigger of the line data item you will notice the following code:

```
IF FirstLineHasBeenOutput THEN
    CLEAR(CompanyInfo.Picture);
FirstLineHasBeenOutput := TRUE;
```


The first time a line is added to the dataset the variable is false. From the second line on, the variable is set to true and the BLOB field is cleared. The column `CompanyInfo.Picture` is in the header data item of the report but, if you run the report and open the runtime dataset, you will see it is cleared when it needs to be and so the performance impact is reduced:

CompanyAdd...	CompanyAdd...	CompanyAdd...	CompanyAdd...	CompanyAdd...	CompanyAdd...	CompanyHo...	CompanyEMa...	CompanyPicture
CRONUS Intern...	5 The Ring	Westminster	W2 8HG London					*
CRONUS Intern...	5 The Ring	Westminster	W2 8HG London					
CRONUS Intern...	5 The Ring	Westminster	W2 8HG London					
CRONUS Intern...	5 The Ring	Westminster	W2 8HG London					
CRONUS Intern...	5 The Ring	Westminster	W2 8HG London					
CRONUS Intern...	5 The Ring	Westminster	W2 8HG London					
CRONUS Intern...	5 The Ring	Westminster	W2 8HG London					
CRONUS Intern...	5 The Ring	Westminster	W2 8HG London					
CRONUS Intern...	5 The Ring	Westminster	W2 8HG London					

As you can see in the preceding screenshot, the field is in the dataset and contains data for every header but not for every line. You can remove the `Picture` field from the header data item and put it on a separate integer data item to further optimize the dataset, filtered on one row.

 The asterisk is used in the dataset to indicate that a BLOB field contains a value.

Although this technique is available in some Dynamics NAV reports, it is not applied in all reports or document reports. For example, the **Reports 206 Sales Invoice** or **Report 208 Sales - Shipment** still use the *old* and *expensive* way of handling logos. I strongly recommend implementing the new technique in all document reports.

 **Moving the BLOB field to separate tables**
BLOB fields should not be created in normal tables, such as Customer, Vendor, and so on. It's better to move BLOB fields to separate tables so they will only be queried when required. For example, when you run the customer card/list page or report, even when you don't put the BLOB field on the page or in the dataset of the report, the database drive generates a `select *` statement which might cause the field to be retrieved from the database, especially when `AutoCalcField` properties or functions are used in the page or report object. Moving BLOB fields to separate tables avoids this performance penalty. This problem might be fixed in the latest version of Dynamics NAV but, in previous versions, I have seen this causing performance issues.

Variables and setup information

As with BLOB fields, you should move all variables and setup fields to an integer data item filtered on one row. Variables and fields that hold the same value for every record in the dataset should not be repeated for every record.

As an example, import object: Packt - CH07-2. Here is that dataset:

No_Customer	Name_Customer	Name_CompanyInformati...	Picture_Comp...	LogoPosition...	HideDetails
01121212	Spotsmeyer's Furnishings	CRONUS International Ltd.	*	<>	<>
01121212	Spotsmeyer's Furnishings	<>	<>	Left	No
01445544	Progressive Home Furnishings	CRONUS International Ltd.	*	<>	<>
01445544	Progressive Home Furnishings	<>	<>	Left	No
01454545	New Concepts Furniture	CRONUS International Ltd.	*	<>	<>
01454545	New Concepts Furniture	<>	<>	Left	No

As you can see, the Company Information, Sales Setup and HideDetails columns are repeated for every customer record. You can optimize this dataset as follows:

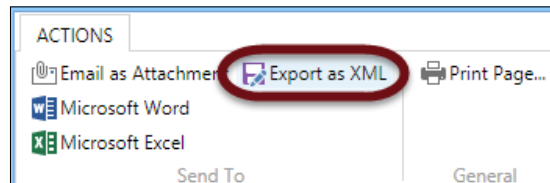
The screenshot illustrates the optimization process in the Report Dataset Designer. A red box highlights the Integer data item in the dataset designer, which is linked to the Integer Properties dialog. The Integer Properties dialog shows the Name set to 'SORTING(NUMBER) WHERE...' and the Table Filter set to 'Number=CONST(1)'. The C/AL Editor shows the OnPreReport() function with code to get the Company Information, Picture, and Sales Receivables Setup fields. The C/AL Globals dialog shows the Integer data item with the Name 'HideDetails'.

As an example of the optimized dataset, you can import object: Packt - CH07-3.

Here, I replaced the data items for Sales & Receivables and Company Information with variables and added them to an integer data item. The dataset of this report looks as follows:

Name_Compa...	Picture_Comp...	LogoPosition...	HideDetails	No_Customer	Name_Customer
CRONUS Intern...	*	Left	No	<>	<>
<>	<>	<>	<>	01121212	Spotsmeyer's Furnishings
<>	<>	<>	<>	01445544	Progressive Home Furnishings
<>	<>	<>	<>	01454545	New Concepts Furniture

This dataset contains all the required information, but nothing is repeated on multiple rows, and the size of this dataset is smaller. You can see the size of a dataset by exporting it to an XML file. In the **About This Report** feature of a report there's this button:



This exports the runtime dataset to an XML file. Simply do this before and after an optimization and compare the file size.

Avoid unnecessary rows

When you have removed all unused columns and optimized BLOB fields then it's time to remove all the rows that are not required in the dataset.

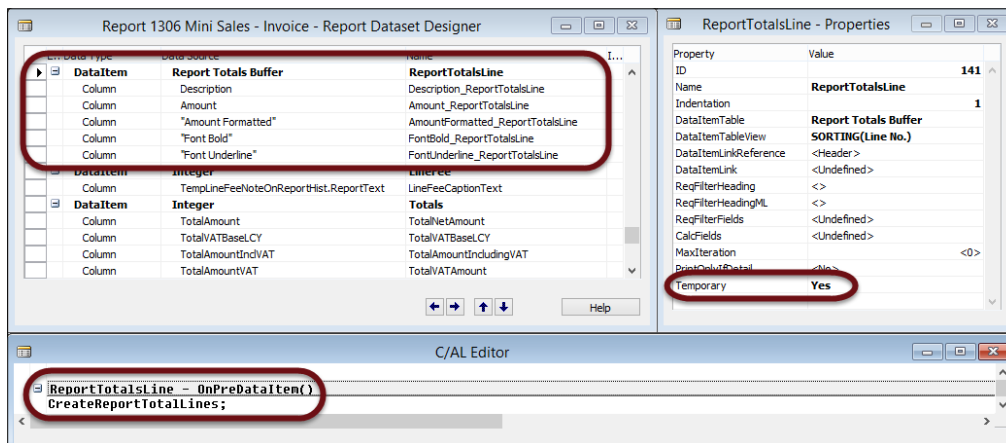
Then, for example in the case of a top X report, you need to apply the appropriate filters. As you know, you can create and apply filters in the RDLC layout, but only if that's really necessary. Filtering should be done in the report dataset designer and you only need to send those rows to the layout that are actually required.

In order to do this, you might also experiment with your data items, the way that they are indented and in what order you put them in the report dataset designer. Sometimes, by rearranging the data items, you will end up with a smaller dataset. For example, if you compare **Sales-Invoice (206)** with **Mini Sales-Invoice (1306)** you will see that the same information is available in both datasets, but the data items are in a different order.

Report totals

You can improve report performance and decrease the size and time it takes to generate the dataset by using buffer (or temporary) tables. An example of this is the usage of the report totals buffer table in the **Report 1306 Mini Sales - Invoice**.

In the dataset of this report there's the following data item named Report Totals Buffer:



The `Temporary` property is enabled in the properties of the data item. This means that the table is used as a temporary table, in RAM memory. Then, in the `OnPreDataItem` trigger, a function is called: `CreateReportTotalLines`. This function contains the following code:

```

ReportTotalsLine.DELETEALL;
IF (TotalInvDiscAmount <> 0) OR (TotalAmountVAT <> 0) THEN
    ReportTotalsLine.Add(SubtotalLbl, TotalSubTotal, TRUE, FALSE,
        FALSE);
IF TotalInvDiscAmount <> 0 THEN BEGIN

```

```
ReportTotalsLine.Add(InvDiscountAmtLbl, TotalInvDiscAmount,
FALSE, FALSE, FALSE);
IF TotalAmountVAT <> 0 THEN
    ReportTotalsLine.Add(TotalExclVATText, TotalAmount, TRUE, FALSE,
FALSE);
END;
IF TotalAmountVAT <> 0 THEN
    ReportTotalsLine.Add(VATAmountLine.VATAmountText, TotalAmountVAT,
FALSE, TRUE, FALSE);
```

The code simply adds a line to the temp table for every total that needs to be displayed in the report. You will see the totals in the dataset, as follows:

The screenshot shows a report window titled "About This Report: Sales - Invoice". It contains a table with the following columns: Description_ReportT..., Amount_Repo..., Amount_Repo..., AmountForm..., FontBold_Rep..., and FontUnderline... The table has five rows of totals highlighted by a red rounded rectangle:

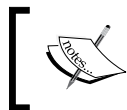
Description_ReportT...	Amount_Repo...	Amount_Repo...	AmountForm...	FontBold_Rep...	FontUnderline...
<>	<>	<>	<>	<>	<>
<>	<>	<>	<>	<>	<>
<>	<>	<>	<>	<>	<>
<>	<>	<>	<>	<>	<>
Subtotal	7830	#,##0.00	7 830,00	Yes	No
Invoice Discount	-391,5	#,##0.00	-391,50	No	No
Total GBP Excl. VAT	7438,5	#,##0.00	7 438,50	Yes	No
10% VAT	743,85	#,##0.00	743,85	No	Yes
<>	<>	<>	<>	<>	<>

The totals are grouped together, contain six columns and, in this example, five rows. If you need to use these totals in the layout, you can reference them in the dataset.

Similarly, there's a data item named Totals that contains total fields (variables) that are calculated when the lines and VAT are processed. The fields are set as columns on a separate integer data item that add only one row at the end of the dataset, and so the dataset is optimized for these fields.

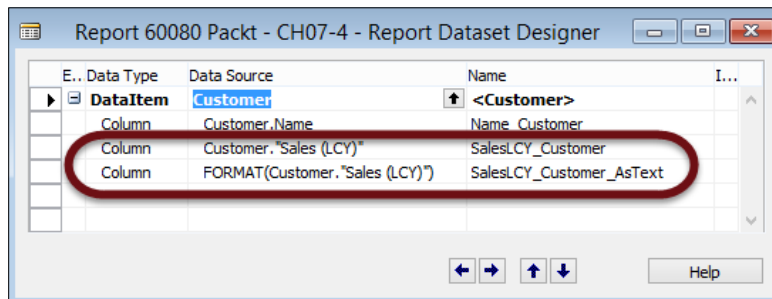
Number formatting

Every time you add a decimal field to the dataset of a report then, in the `rdlc` layout, you get an extra field that contains the format string. This format string is also added to the dataset at runtime as a column, and it always contains the same value. This is a typical example of adding information to the dataset that is not really required and there's a simple way to avoid this. When you add a decimal field to the dataset, you can simply use the `FORMAT` function to convert the field to a string. Then, no format code is generated or added to the dataset, and you will save resources at runtime. If you want to have a format code available in the `rdlc` layout, simply add it manually in a separate data item. Usually, all decimal fields, amounts and unit amounts use the same format. So you can add one unit amount format code and one amount format code to the dataset and use that in the format property in the layout.



When you apply this trick, don't forget to convert the string back into a decimal field in the layout. Simply use the `CDec()` function in order to do that.

Here's an example of how you can hide a number format. Import the object: Packt - CH07-4 and you will see the following dataset:



In the preceding screenshot you can see the difference between the `SalesLCY_Customer` and `SalesLCY_Customer_AsText` fields. One was added to the dataset as a decimal field, the other with a `Format()` function.

When you run the report and open the dataset, you get this:

Name_Customer	SalesLCY_Customer	SalesLCY_CustomerFormat	SalesLCY_Customer_AsText
The Cannon Group PLC	17100,96	##0.00	17 100,96
Selangorian Ltd.	6510,64	##0.00	6 510,64
Metatorad Malavsia Sdn Bhd	0	##0.00	0

Both fields (`SalesLCY_Customer` and `SalesLCY_Customer_AsText`) contain the same values. There's a value and a format string for the `SalesLCY_Customer` field in the column named `SalesLCY_CustomerFormat`. For the field `SalesLCY_Customer_AsText`, there's only a value. The `Format()` function returns a text data type, so the value of the field `SalesLCY_Customer_AsText` is in a text data type.

This means that, when you use it in the RDLC layout, you need to convert it back to a decimal data type. You use the `CDec()` function to do that. If you do not convert it to a decimal then, in the report layout, you can't use the `Sum()` function on this field, for example when you want to display a footer row containing the total sales.

Name Customer	Sales LCY Customer	Sales LCY Customer As Text
[Name_Customer]	[SalesLCY_Customer]	«Expr»
Total:	[Sum(Sales_LCY_Customer)]	«Expr»

=CDec(Fields!SalesLCY_Customer_AsText.Value)

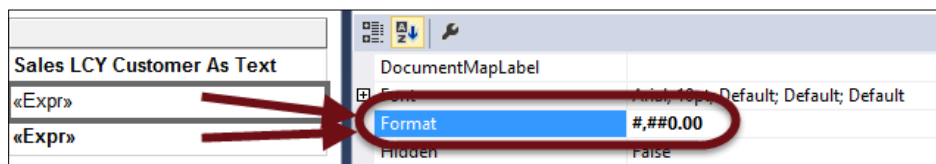
=Sum(CDec(Fields!SalesLCY_Customer_AsText.Value))

When you run the report, there's no difference between the two columns:

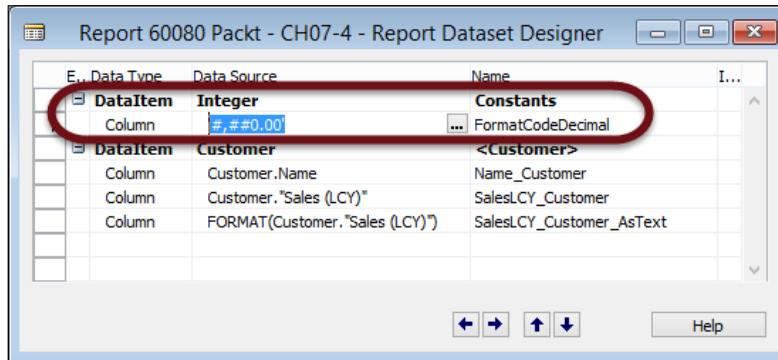
Name Customer	Sales LCY Customer	Sales LCY Customer As Text
Progressive Home Furnishings	1499,02	1499,02
The Cannon Group PLC	17100,96	17100,96
Selangorian Ltd.	6510,64	6510,64
John Haddock Insurance Co.	6142,9	6142,9
Antarctcopy	2582,81	2582,81
Gagn & Gaman	877,32	877,32
Heimilisprydi	2024,21	2024,21
Deerfield Graphics Company	1063,1	1063,1
BYT-KOMPLET s.r.o.	1602,9	1602,9
Designstudio Gmunden	14498,04	14498,04
Englunds Kontorsmöbler AB	673,71	673,71
Klubben	11772,2	11772,2
Beef House	6000	6000
Autohaus Mielberg KG	4331,4	4331,4
Hotel Pferdesee	14450	14450
Guildford Water Department	533,4	533,4
Total:	91662,61	91662,61

If you remove the `SalesLCY_Customer` column from the dataset designer, there will be two fewer columns then in the runtime dataset, `SalesLCY_Customer` and `SalesLCY_CustomerFormat`. You can use different options to format the `SalesLCY_Customer_AsText` field.

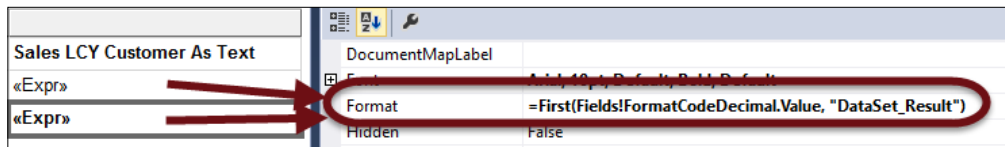
Option one is to use a format constant, for example `n2` or `#,##0.00` in the format property of the textbox:



Option two is when you don't want to hard code the `format` property, but you want it to come from the dataset. In this case, you add a `format` constant to the dataset, as follows:



You use the expression `=First(Fields!FormatCodeDecimal.Value, "DataSet_Result")` in the RDLC layout in the `Format` property, as you can see in this screenshot:



I recommend the second approach because you can determine, and change, the format code via C/AL, and you can reuse the field `FormatCodeDecimal` in all the textboxes that need to be formatted as a decimal.

Applying the correct filters

Making sure the dataset is not too big is the most important factor in optimizing report performance. The dataset and the report layout are the two files that the report viewer uses to render the report:

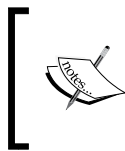
- Report.rdlc
- Dataset.xml

The report viewer application runs on the client, not on the server. Dynamics NAV is a three tier application and all C/AL code is executed on the service tier, on the server. This means that, when a report is rendered, the dataset will be streamed from the server to the client. The data is fetched from the database, the SQL server, which generates a complete result set. This result set is then sent to the Dynamics NAV service tier over the network. When the service tier receives the information from the SQL server, it sends it in chunks to the report viewer (on the client). Once a packet or chunk is received by the client, it is cleared from the memory of the service tier.

If you open the Windows task manager on the server and on the client while you are executing a report with a big dataset, you will see that the memory consumption on the server is pretty constant, while the memory consumption of the client keeps on growing and growing, until it runs out of memory.

Depending on the number of columns and the data types of the columns, this will typically happen as soon as the dataset reaches about 1 GB. At that moment, an out of memory exception will be thrown.

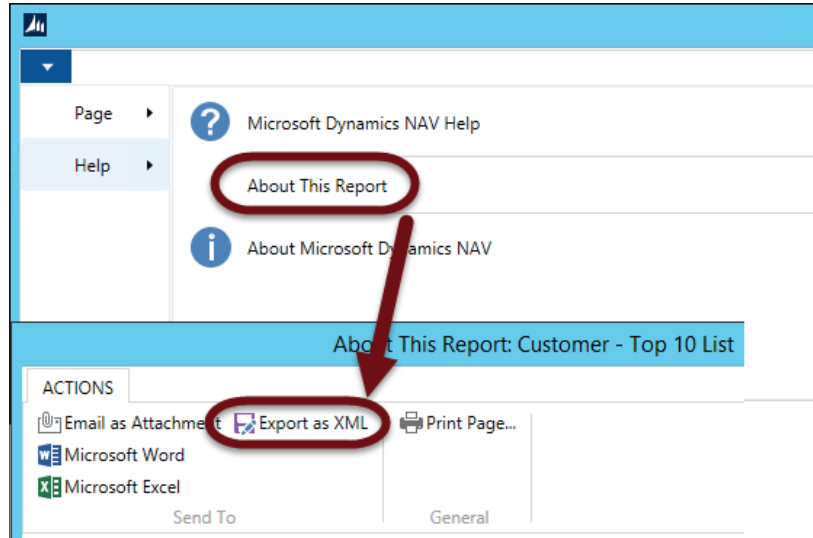
To avoid this happening you need to consider reducing the size of the dataset.



Running a report on the server (the service tier) is faster because the Dynamics NAV service tier is a 64-bit application and can therefore consume much more RAM memory.

If you optimize the dataset of your report, you can also test the results to see the difference. Before you apply the modifications, run your report and, in the report viewer, export the dataset to an XML file. You can use the **About This Report** feature to do this to open the runtime dataset and then use the **Export as XML** file option.

Then, after you have implemented the performance improvements, run your report again, export the dataset again to an XML file and simply compare the size of the two files:



The page file



Microsoft released a recommendation on the Dynamics NAV Team blog a while ago, which can be seen here: <http://blogs.msdn.com/b/nav/archive/2011/03/29/designing-reports-for-better-performance-on-rtc-ii.aspx> This article describes a method for solving or avoiding out of memory exceptions and basically consists of making your swap file bigger. I definitely recommend avoiding this. It is not a solution to the problem and only fixes the symptoms and merely delays the error and, most importantly, increases the size of the swap file which can make your Windows operating system unstable.

Recommendations according to the version of Dynamics NAV

When you use C/AL code, as you do in reports, there are several functions and methods that you can apply when working with tables and record functions. Depending on the version of Dynamics NAV, the recommendations on when to use which `FIND` statement, are different. You can find an explanation of the differences between Dynamics NAV versions in this article: <http://blogs.msdn.com/b/nav/archive/2011/05/12/microsoft-dynamics-nav-changes-by-version.aspx>.

The layout

Once you have optimized the dataset of a report, it's time to have a look at the layout. In this section I have made recommendations about how you can optimize the report layout. This is important when you are upgrading a report. There are some workarounds for document reports that are implemented differently depending on the version of RDLC that is used.

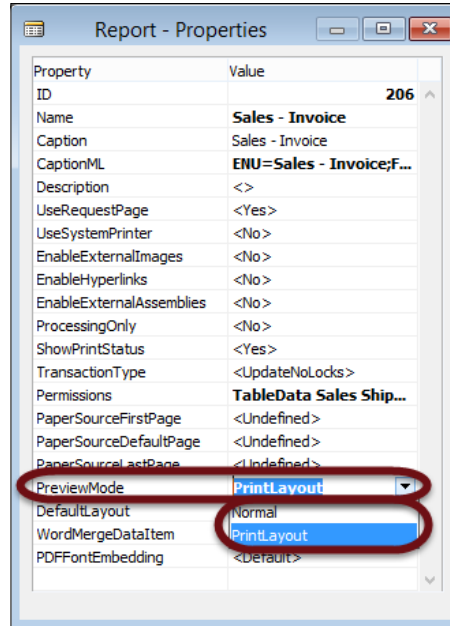


When you are implementing performance optimizations, I recommend including the user. The user must understand what is going on, and feel that they have a say in what is done about it. I'd recommend that partners create a document that explains the pros and cons of these changes, and so make sure that users are informed about it. An informed customer is often very reasonable in deciding what is done about performance problems. One customer may "REALLY" want the page x of y functionality while another may "REALLY" need the interactive nature of print layout/preview.

Print layout versus print preview

When you run a report, the report viewer starts in interactive mode by default (print preview). Interactive mode features such as expand/collapse, interactive sorting, and so on, are enabled. These features can consume extra memory. If these features are not always required, you can make sure the report runs in print layout mode by default, instead of print preview.

This can be achieved using the report property:



Once you run the report and you want to switch back to interactive mode, you can use the report viewer menu:



Avoid conditional visibility on a big dataset

Report items such as textboxes, columns, rows, tables, and so on, all have visibility properties. You can conditionally show or hide a report item using the `Hidden` and `ToggleItem` properties. At runtime, every time you click the toggle item, the report viewer performs calculations and consumes resources, like RAM memory. The bigger the report item that you toggle, the more memory it will consume, until the report viewer runs out of RAM memory.

As an alternative, you can implement drill-through. This means using hyperlinks to link to a report that contains the detail data, as when you click on a `FlowField` in a page.

Another alternative is to provide an option in the request page of the report, as in report **Customer - Top 10 List**. There the user can select between a bar or pie chart. The user still has a choice then, and the report viewer processing will be faster. You can give the user a choice before the report is rendered, instead of a toggle after the report is rendered, which consumes more RAM memory.

Furthermore, options in the request page can also be set using a `Set` function, to expose the property outside the report object and, when you render the report via C/AL code, for example via a code unit, you can determine which visualization will be used.

Best practices when visualizing information

If your report contains a lot of textboxes or instances of textboxes, then you should avoid `CanGrow` and `CanShrink` or set `TextAlign` to `General`. These properties require processing depending on the content of the textbox and so will use more resources. Make sure you only use them when they are absolutely necessary.

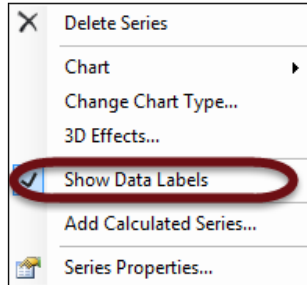
If you use a lot of images in reports, then don't set the `AutoSize` property to `Fit`. Using the `Fit` option increases rendering time because the report viewer needs to resize the images.

If you use the `KeepTogether` property on a `Tablix`, the report viewer requires additional processing when it encounters a page break.

If you are using a chart or gauge control, think about the amount of data that is required to render it. The effectiveness of a chart decreases when you display several pieces of pie or bars. Ideally you should show between three and eight pieces, and if more is required, display them in a different part.

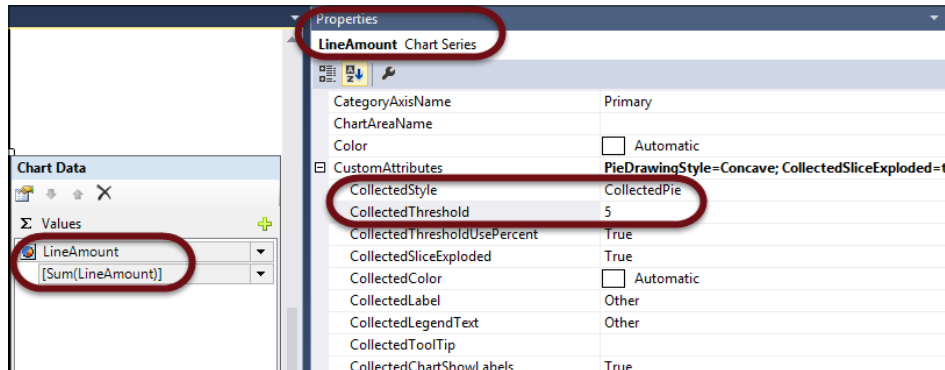
If you want to display a lot of data points in a chart, it is better to use a scatter chart instead of a pie chart.

If you have a lot of data points, then don't display data labels. If you right-click on a chart in Visual Studio, the following options display:

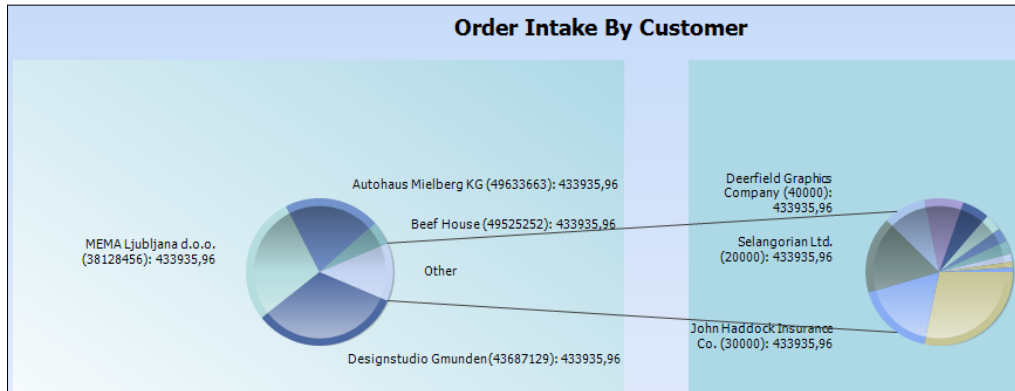


If your chart has a lot of values, you can disable **Show Data Labels**.

An example of displaying only five pieces of pie is available in the object: Packt - CH07-5. If you open the RDLC layout and have a look at the properties of the following chart, you will see this:

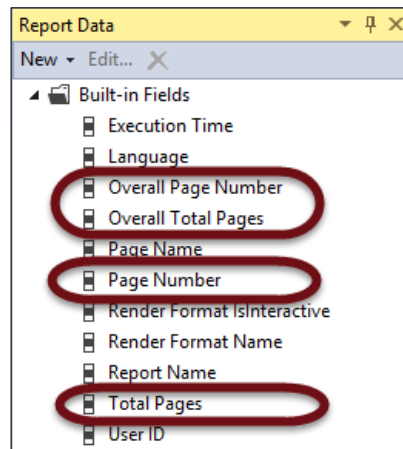


The property `CollectedThreshold` is set to 5 so that any other piece of pie is rendered in a separate chart named `other`:



Expressions in the page header or footer

If you use one of the following globals in the page header or footer of a report, the report viewer has to process the complete report to calculate the pagination:



The more pages there are in the report, the longer it takes to process the dataset, and the longer it takes before it opens.

You can see when this happens in the toolbar of the report, where the navigation buttons are shown. If the report has not yet been processed completely, a question mark is shown next to the total number of pages:



If there's no reference to the page number then the report can be rendered after the first page has been processed, without processing the rest of the report.

Of course, when you then print or export the report, the report viewer will reprocess the complete report.

Complex grouping and aggregate functions

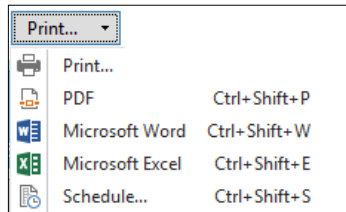
If you have a Tablix that contains multiple levels of adjacent or parent child groupings, it can take a lot of resources to process, especially when you are using complex expressions for the groups. When the expression is dependent on the sorting of the information in the group it becomes more expensive. For example, the `Sum` function does not depend on the sorting, while the `Previous`, `First` and `Last` functions depend on the sorting, and are evaluated after the sorting, grouping and filtering have been applied.

As a solution, you can use a query object that contains groupings, instead of grouping in the layout. You can also reduce the data in the report to reduce the performance impact.

Optimization for the chosen rendering format

Reports are rendered differently in different rendering formats, such as Excel, Word, and PDF. For example, depending on the version of Excel, one page can contain 65,000 or one million rows. You might then have to implement page breaks so that the information is broken up between multiple Excel sheets. The same is valid when you render a report in PDF.

Pagination and report margins might be rendered differently, so implementing page breaks could render a report in PDF better.



Report design guidelines

Microsoft has created a document that defines the design guidelines for report items in the report layout. Following these guidelines will make sure that the reports you develop have a consistent and clean layout. Of course, you don't have to follow these guidelines and can create your own. Based on the guidelines that you want to follow, you can create templates to implement reusability. It might be interesting to implement values for properties in your guidelines to improve report performance.



More information about the Microsoft report and user interface design guidelines can be found at [https://msdn.microsoft.com/en-us/library/jj651616\(v=nav.70\).aspx](https://msdn.microsoft.com/en-us/library/jj651616(v=nav.70).aspx).

Implementing hotfixes and rollup updates

Although it might seem obvious, I rarely see hotfixes being implemented in real life. In the past, implementing a hotfix usually required a manual fix on all clients, and on the server. This is a time-consuming process because you usually have to visit the customer site to do the installation. There are also a lot of hotfixes to implement, depending on the version of Dynamics NAV that you use.

Nowadays, with PowerShell and Click Once technology it's easier to automate this process. Microsoft releases rollup updates about once a month. So, if you experience performance issues, have a look at the Microsoft knowledge base to see if there's an update that might solve the problem.



A good alternative to the Microsoft knowledge base is Waldo's blog, which you can find at <http://www.waldo.be>. He gives an overview of all released platform updates on a regular basis, and I find it easier to see the wood for the trees on his blog.

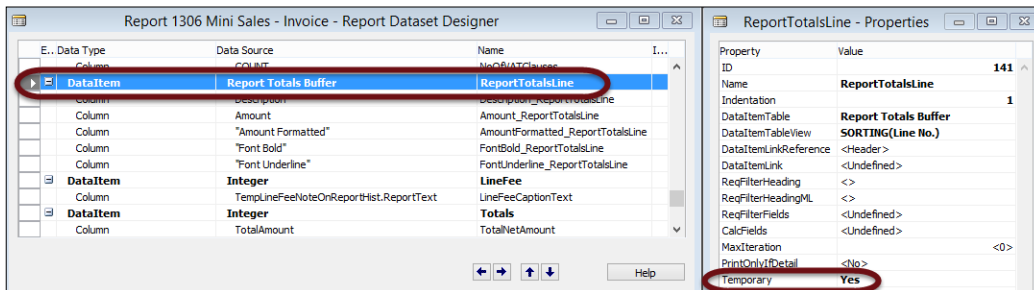
Alternatives for building a faster dataset

I have always demonstrated building the dataset using data items that reference database tables or the integer data item. I will now demonstrate two techniques that you can apply to complex reports to improve report performance and also make the report easier to maintain.

Using a temporary table

The first technique is using a temporary table. This technique is used a lot in document reports and I might have mentioned it already very briefly in previous chapters.

A buffer table is a fancy name for a temporary table, or a table that is used as a temporary table. In Dynamics NAV, when you want to use a temporary table, you need to declare a variable that references an existing table and then set its temporary property to *yes*. You can also do that in a report, and up to version 2013 it was the only way to use temporary tables in reports. In version 2015, it's still possible, but there's an alternative. A data item has a new property named *Temporary*. You can then use any data item as a temporary table, or buffer table, as in a page object. An example is available in **Report 1306 Mini Sales - Invoice**:



The `Report Totals Buffer` table is an actual table that was created specifically to hold temporary data for reports. The data is collected on the server side, saved in this temporary record, and then rendered in the report viewer. This minimizes the dataset in a physical structure, which is very useful to simplify report development.

The advantage of using a buffer table is that you can fill it with data from different sources and then simply loop over the results.



The disadvantage is that you need to have the table available or create it in the database, and table objects are not free. So, if you create a lot of buffer tables, you may run out of tables in your license file. When you develop reports at a customer's site, they might not have tables available in their license file, and it's better to know this in advance instead of informing them afterwards.



Temporary tables and the license file

You can always reference any table as a temporary table, even when the table is not included in your license file.

Another technique, which does not involve using a table, is to use variables and put them into an integer data item. I prefer to work with buffer tables because they allow me to use record variable functions.

A common element of both techniques, integer or buffer tables, is that you are first going to loop over the source tables and then put the results either in the buffer table or in the variables of the integer data item. The advantage is that the source tables are created as separate data items and so can be put into the request page, and the user can apply filters on them.

Another technique is to create a function in the source tables, which generates a dataset that you then put in the report. When you use an integer data item, this function is called on the `OnPreDataItem()` trigger, and you can use the number of records it returns to filter the integer data item.

So, to summarize, the technique consists of two steps:

1. First you generate a temporary dataset.
2. Then you iterate over this dataset, using an integer data item.

Let's look at an example to make it clearer. I will use **Report 113 Customer/Item Sales** as an example because this technique is applied in many of the existing reports in Dynamics NAV.

When I run the report the result looks like this:

Item No.	Description	Invoiced Quantity	Unit of Measure	Amount	Discount Amount	Profit	Profit %
Customer: No.: 10000..20000							
10000	Van Terp Kantoorinrichting Phone No.						
1920-S	ANTWERPEN Vergadert	0	STUKS	0,00	0,00	0,00	0,00
1984-W	INNSBRUCK Kast/G.D	10	STUKS	4 523,00	0,00	1 871,00	41,40
1988-S	MEXICO Draaistoel. zwa	3	STUKS	544,35	28,85	97,65	17,90
1998-S	ATLANTA Whiteboard, b	7	STUKS	9 338,59	491,51	1 671,49	17,90
70011	Glasdeur	5	STUKS	580,00	0,00	274,00	48,90
	Van Terp Kantoorinrichting			14 965,94	520,16	3 914,14	26,20
20000	Anton Geestig Adviezen Phone No.						
1898-S	ATHENE Tafel	0	STUKS	0,00	0,00	0,00	0,00
1928-S	AMSTERDAM Lamp	5	STUKS	287,72	8,28	52,22	19,50
788BC-C	CONTOSO Opslagsyste	0	STUKS	0,00	0,00	0,00	0,00
	Anton Geestig Adviezen			267,72	8,28	52,22	19,50
Total				15 233,66	528,44	3 966,36	26,00

You might wonder why there are only two customers in my report.

Well, the report is filtered on a subset of customers because I applied a filter on the request page:

Show results:

✗ **Where** No. ▼ is 10000..20000

✗ **And** Search Name ▼ is Enter a value.

✗ **And** Customer Posting Group ▼ is Enter a value.


+ **Add Filter**

From the point of view of the user this is very interesting, because they can use the request page to filter on any field from the *Customer* table, and also the *Item* table. The goal of the report is to display an overview of which items a customer has bought from the company, including information about the invoiced quantity, amounts, discounts, and so on. This is reflected in the dataset of the report, as there are two data items: *Customer* and *Item Ledger Entry*. These two data items show up in the request page. The following is a screenshot of the dataset of the report:

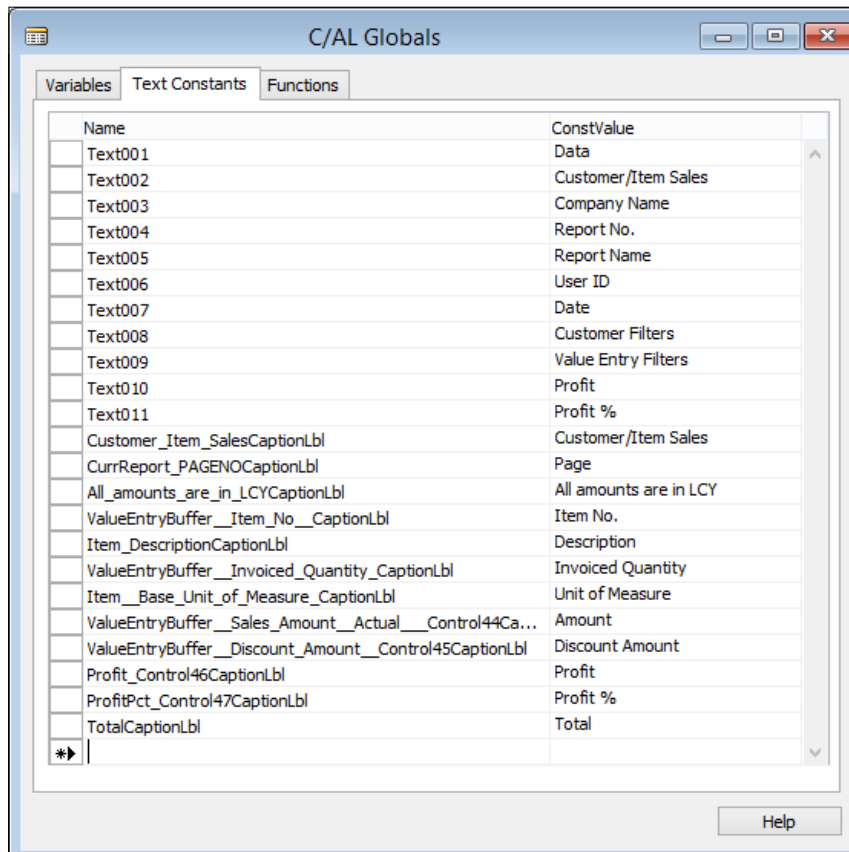
Data Type	Data Source	Name
DataItem	Customer	<Customer>
Column	STRSUBSTNO(Text000,PeriodText)	STRSUBSTNO_Text000_PeriodText_
Column	CurrReport.PAGENO	CurrReport_PAGENO
Column	COMPANYNAME	COMPANYNAME
Column	PrintOnlyOnePerPage	PrintOnlyOnePerPage
Column	TABLECAPTION + ':' + CustFilter	Customer_TABLECAPTION_____CustFilter
Column	CustFilter	CustFilter
Column	'Item Ledger Entry'.TABLECAPTION + ':' + ItemLe...	Value_Entry__TABLECAPTION_____ItemLedg...
Column	ItemLedgEntryFilter	ItemLedgEntryFilter
Column	"No."	Customer_No__
Column	Name	Customer_Name
Column	"Phone No."	Customer__Phone_No__
Column	ValueEntryBuffer."Sales Amount (Actual)"	ValueEntryBuffer__Sales_Amount__Actual__
Column	-ValueEntryBuffer."Discount Amount"	ValueEntryBuffer__Discount_Amount__
Column	Profit	Profit
Column	ProfitPct	ProfitPct
Column	PrintToExcel	PrintToExcel
Column	Customer_Item_SalesCaptionLbl	Customer_Item_SalesCaption
Column	CurrReport_PAGENOCaptionLbl	CurrReport_PAGENOCaption
Column	All_amounts_are_in_LCYCaptionLbl	All_amounts_are_in_LCYCaption
Column	ValueEntryBuffer__Item_No__CaptionLbl	ValueEntryBuffer__Item_No__Caption
Column	Item_DescriptionCaptionLbl	Item_DescriptionCaption
Column	ValueEntryBuffer__Invoiced_Quantity__CaptionLbl	ValueEntryBuffer__Invoiced_Quantity__Caption
Column	Item_Base_Unit_of_Measure_CaptionLbl	Item_Base_Unit_of_Measure_Caption
Column	ValueEntryBuffer__Sales_Amount__Actual__Contr...	ValueEntryBuffer__Sales_Amount__Actual__Contro...
Column	ValueEntryBuffer__Discount_Amount__Control45C...	ValueEntryBuffer__Discount_Amount__Control45Ca...
Column	Profit_Control46CaptionLbl	Profit_Control46Caption
Column	ProfitPct_Control47CaptionLbl	ProfitPct_Control47Caption
Column	FIELDCAPTION("Phone No.")	Customer__Phone_No__Caption
Column	TotalCaptionLbl	TotalCaption
DataItem	Item Ledger Entry	<Item Ledger Entry>
DataItem	Integer	<Integer>
Column	ValueEntryBuffer."Item No."	ValueEntryBuffer__Item_No__
Column	Item.Description	Item_Description
Column	-ValueEntryBuffer."Invoiced Quantity"	ValueEntryBuffer__Invoiced_Quantity__
Column	ValueEntryBuffer."Sales Amount (Actual)"	ValueEntryBuffer__Sales_Amount__Actual__Contro...
Column	-ValueEntryBuffer."Discount Amount"	ValueEntryBuffer__Discount_Amount__Control45...
Column	Profit	Profit_Control46
Column	ProfitPct	ProfitPct_Control47
Column	Item."Base Unit of Measure"	Item_Base_Unit_of_Measure__

By using the `GetFilters()` function in the `OnPreReport()` trigger, the filters that the user enters are added to the dataset so that they can be shown in the layout:


```
CustFilter := Customer.GETFILTERS;  
ItemLedgEntryFilter := "Item Ledger Entry".GETFILTERS;  
PeriodText := "Item Ledger Entry".GETFILTER("Posting Date");
```

 You can use the `record.GETFILTER(Fieldname)` or `record.GETFILTERS()` functions to retrieve the filter criteria set on a record variable.

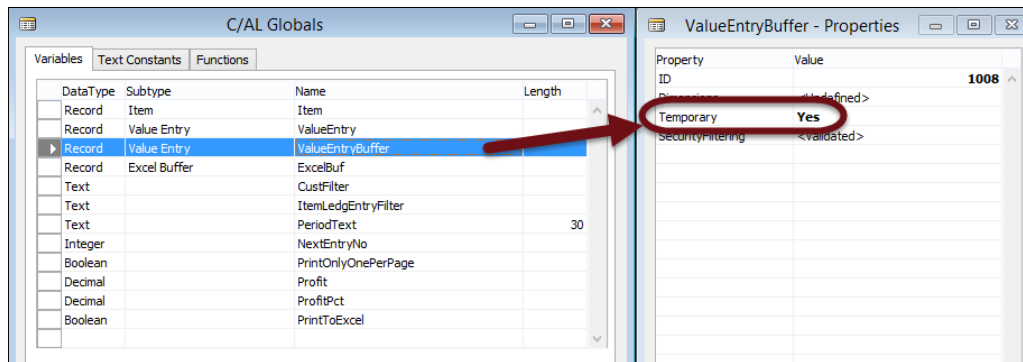
As you can see, the dataset of the report also contains captions/text constants. This is an example of a report that was upgraded from a previous version of Dynamics NAV.



Name	ConstValue
Text001	Data
Text002	Customer/Item Sales
Text003	Company Name
Text004	Report No.
Text005	Report Name
Text006	User ID
Text007	Date
Text008	Customer Filters
Text009	Value Entry Filters
Text010	Profit
Text011	Profit %
Customer_Item_SalesCaptionLbl	Customer/Item Sales
CurrReport_PAGENOCaptionLbl	Page
All_amounts_are_in_LCYCaptionLbl	All amounts are in LCY
ValueEntryBuffer__Item_No__CaptionLbl	Item No.
Item_DescriptionCaptionLbl	Description
ValueEntryBuffer__Invoiced_Quantity__CaptionLbl	Invoiced Quantity
Item_Base_Unit_of_Measure_CaptionLbl	Unit of Measure
ValueEntryBuffer__Sales_Amount_Actual__Control44Ca...	Amount
ValueEntryBuffer__Discount_Amount__Control45CaptionLbl	Discount Amount
Profit_Control46CaptionLbl	Profit
ProfitPct_Control47CaptionLbl	Profit %
TotalCaptionLbl	Total

 These text constants should be replaced by labels to optimize the dataset.

There's an integer data item at the end of the dataset. It mostly contains fields from the ValueEntryBuffer table. The record variable ValueEntryBuffer references the Value Entry table, but is used as a temporary table with its Temporary property. This means that we will be using a copy of the definition of the actual Value Entry table as a buffer table.



What happens in this report is that there's a loop over all customers and every item ledger entry per customer. The Item Ledger Entry table is linked to the Customer table via `Source No.=FIELD(No.)`, `Posting Date=FIELD(Date Filter)`, `Global Dimension 1 Code=FIELD(Global Dimension 1 Filter)`, `Global Dimension 2 Code=FIELD(Global Dimension 2 Filter)`.

The buffer table is cleared in the `OnPreDataItem()` trigger of the Item Ledger Entry table and a counter (`NextEntryNo`) is initiated.

```
Item Ledger Entry - OnPreDataItem()
ValueEntryBuffer.RESET;
ValueEntryBuffer.DELETEALL;

NextEntryNo := 1;
```

You can see the following code in the `OnAfterGetRecord()` trigger:

```
Item Ledger Entry - OnAfterGetRecord()
ValueEntryBuffer.SETRANGE("Item No.", "Item No.");

IF NOT ValueEntryBuffer.FIND('-') THEN BEGIN
    ValueEntryBuffer.INIT;
    ValueEntryBuffer."Entry No." := NextEntryNo;
    ValueEntryBuffer."Item No." := "Item No.";
    ValueEntryBuffer.INSERT;

    NextEntryNo := NextEntryNo + 1;
END;
CALCFIELDS("Sales Amount (Actual)", "Cost Amount (Actual)", "Cost Amount (Non-Invtbl.)");
ValueEntryBuffer."Invoiced Quantity" := ValueEntryBuffer."Invoiced Quantity" + "Invoiced Quantity";
ValueEntryBuffer."Sales Amount (Actual)" := ValueEntryBuffer."Sales Amount (Actual)" + "Sales Amount (Actual)";
ValueEntryBuffer."Cost Amount (Actual)" := ValueEntryBuffer."Cost Amount (Actual)" + "Cost Amount (Actual)";

ValueEntry.SETCURRENTKEY("Item Ledger Entry No.");
ValueEntry.SETRANGE("Item Ledger Entry No.", "Entry No.");
IF ValueEntry.FINDSET THEN
    REPEAT
        ValueEntryBuffer."Discount Amount" := ValueEntryBuffer."Discount Amount" + ValueEntry."Discount Amount";
    UNTIL ValueEntry.NEXT = 0;

ValueEntryBuffer."Cost Amount (Non-Invtbl.)" := ValueEntryBuffer."Cost Amount (Non-Invtbl.)" + "Cost Amount (Non-Invtbl.)";
ValueEntryBuffer.MODIFY;
```

The code inserts an entry in the buffer table if it's the first ledger entry for a customer if not the code will look for the entry using the `Item No` and will then update the entry in the buffer table. This is because one customer record is probably linked to multiple item ledger records. The fields in the buffer table are filled with information from the item ledger entry table, except the `Discount Amount`, which comes from the `Value Entry` table using a separate loop.

After all customers and related item ledgers are processed, the number of records in the buffer table is used to filter the integer data item in the `OnPreDataItem` of the `Integer` table:

```
Integer - OnPreDataItem()
CurrReport.CREATETOTALS(
    ValueEntryBuffer."Sales Amount (Actual)",
    ValueEntryBuffer."Discount Amount",
    Profit);

ValueEntryBuffer.RESET;
SETRANGE(Number, 1, ValueEntryBuffer.COUNT);
```

The first time you execute the `FIND()` function and the other iterations you need to fetch the next record in the `OnAfterGetRecord()` trigger:

```
Integer - OnAfterGetRecord()
IF Number = 1 THEN
    ValueEntryBuffer.FIND('-')
ELSE
    ValueEntryBuffer.NEXT;
```

Now, the integer data item loops over the resulting dataset in the buffer table and becomes the dataset of the report.

This is an example of how you can create a dataset using a buffer table.



In the example, you can see the `CREATETOTALS()` function. This has no purpose in RDLC reporting, since totals will be calculated in the layout.

Other examples of using a buffer table can be found in the following reports:

- **Report 204 Sales - Quote**
- **Report 205 Order - Confirmation**
- **Report 206 Sales - Invoice**
- **Report 208 Sales - Shipment**

There are many other reports where buffer tables are used. Another example, which I mentioned in previous chapters, is the "Customer Top 10 report".



You can avoid complex processing or filtering in the report layout by using a buffer table and moving this to the dataset. The report will run faster because the dataset and all C/AL code run on the server and the layout, which is generated on the client, requires less resources.

Using a query object for the dataset

The `query` object was first introduced in version 2013 and its purpose was to have an effective way to retrieve information from the database, which is an alternative for record variables.



The primary purpose of query objects is to provide a way for the developer to use SQL JOIN expressions and other T-SQL attributes that are not available in C/AL manipulations. The other purpose is to provide the ability to specify columns in queries so that you don't always have SELECT *, and you can reduce the amount of data in the result set.

The query object in Dynamics NAV is built based on a typical SQL statement. The following table summarizes the elements and properties of query objects, and how they relate to T-SQL:


SQL statement	Query object feature
SELECT	Row of type Column in the query designer
FROM	Row of type DataItem in the query designer
JOIN Type	DataItemLinkType and SQLJoinType query properties
ON	DataItemLink data item property
WHERE	DataItemTableFilter data item property ColumnFilter property of columns and filters row of type filter
HAVING	ColumnFilter property of columns and filters, when aggregation is used
GROUP BY	Automatically switched on for each row of type column, when aggregation is used
ORDER BY	OrderBy query property
TOP	TopNumberOfRows query property

You can fetch data from one or more tables with a query object, which you can join in different ways. A query object supports grouping and aggregation methods. This is an advantage for reports because you can move grouping and aggregation from the layout to the dataset and so the report viewer only has to display the results instead of spending resources performing groupings and aggregations.

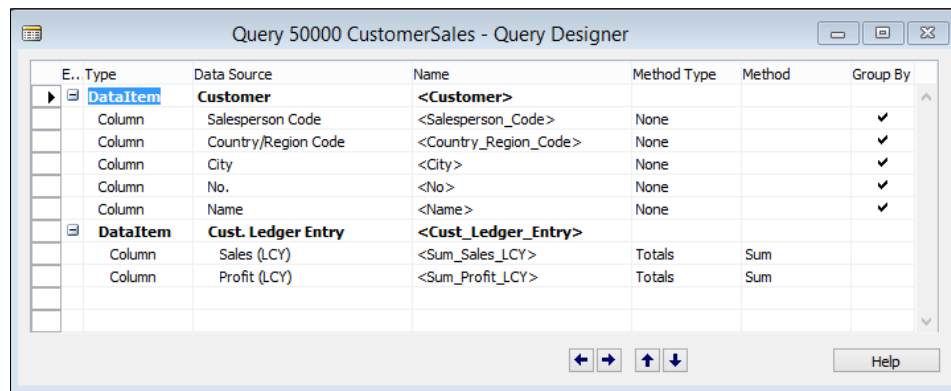


A query object is also reusable in multiple reports, which is another advantage.

The SQL server is also optimized to execute SQL statements generated by query objects quickly, which results in optimal performance.


 Query objects can also be published as ODATA web services. These can then be used by other applications such as Excel, PowerPivot, and so on, as a data source. But this is the subject of the chapter that deals with Power BI.

Let's start by having a look at how you can create a query object. I have created a query that fetches the total Sales (LCY) and Profit (LCY) from the Ledger Entry table in the following example, grouped by the Salesperson, Country, City, No., and Name from the customer table:



E.. Type	Data Source	Name	Method Type	Method	Group By
▶ DataItem	Customer	<Customer>			
Column	Salesperson Code	<Salesperson_Code>	None		✓
Column	Country/Region Code	<Country_Region_Code>	None		✓
Column	City	<City>	None		✓
Column	No.	<No>	None		✓
Column	Name	<Name>	None		✓
▶ DataItem	Cust. Ledger Entry	<Cust_Ledger_Entry>			
Column	Sales (LCY)	<Sum_Sales_LCY>	Totals	Sum	
Column	Profit (LCY)	<Sum_Profit_LCY>	Totals	Sum	

I have set the property `TopNumberOfRows` to 10 in the query properties, which means this query will fetch only the top 10, sorted by `Sum_Sales_LCY`.

 This is only a default value at runtime. When you execute a query with C/AL code you can modify it via the C/AL function `TOPNUMBEROFROWS ()`.

OrderBy	Sum_Sales_LCY=Descending
TopNumberOfRows	10

The result set of the query looks as follows:

About This Query: Packt - CH07-1

Salesperson_C...	Country_Regi...	City	No	Name	Sum_Sales_LCY	Sum_Profit_LCY
PS	GB	Birmingham	10000	The Cannon Gr...	17100,96	5338,26
JR	AT	Gmunden	43687129	Designstudio G...	14498,04	12361,44
JR	DE	Frankfurt/Main	49858585	Hotel Pferdesee	14450	5025,47
JR	NO	Haslum	47563218	Klubben	11772,2	4120,2
PS	GB	Coventry	20000	Selangorian Ltd.	6510,64	2511,94
PS	GB	Manchester	30000	John Haddock I...	6142,9	2921
JR	DE	Düsseldorf	49525252	Beef House	6000	2471,69
JR	DE	Hamburg 36	49633663	Autohaus Miel...	4331,4	1465,67
JR	BE	Antwerpen	32656565	Antarcticopy	2582,81	726,01
JR	IS	Reykjavik	35963852	Heimilispyrdi	2024,21	338,31

Now I will use this result set as the dataset in a report. First I will create a new report and add the following variables:

C/AL Globals

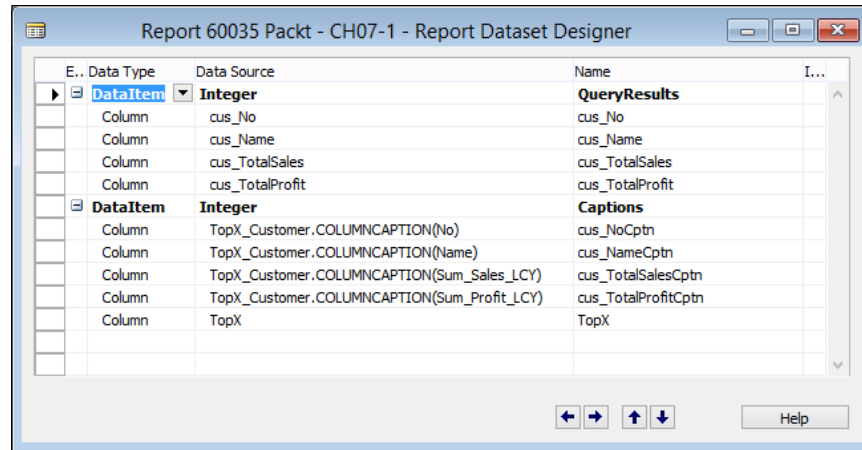
DataType	Subtype	Name	Length
Query	Packt - CH07-1	TopX_Customer	
Code		cus_No	20
Text		cus_Name	
Decimal		cus_TotalSales	
Decimal		cus_TotalProfit	
Integer		TopX	

There's one variable that references the query object, and one variable per column in the result set of the query. Next I have included a `TopX` variable which I will use to enable the user to specify a top x value in the request page of the report, which looks as follows:


Report 50005 Report Query - Request Options Page Designer

E.. Type	SubType	SourceExpr	Name	Caption
Container	ContentArea		Options1	<Options1>
Group	Group		Options	<Options>
Field		TopX	TopX	TopX?

Then I create the report dataset, using two integer data items:



I will use the first data item, named *QueryResults*, to iterate over the result set of the query object. I will use the second data item, *Captions*, to add the captions of the query columns to my dataset, in one row, using the `COLUMNCAPTION` function.

 I have used the value `SORTING (Number)` `WHERE (Number=CONST (1))` in the `DataItemTableView` property of the `Caption` data item so that it's filtered on one row.

Now I need to add code to the report triggers that will run the query. I will do this in the `OnPreDataItem()` trigger of the `QueryResults` data item, as follows:

```
QueryResults - OnPreDataItem()
SETRANGE(Number,1,TopX);
CLEAR(TopX_Customer);

TopX_Customer.TOPNUMBEROFROWS(TopX);
TopX_Customer.OPEN;
```

I start by filtering the data item entered by the user in the request page from 1 to `TopX`. Then I set the `TopNumberOfRows` property of the query to `TopX` using the `TOPNUMBEROFROWS()` function. Next I use the `OPEN()` function to execute the query.



The `Open()` function runs the query and leaves the dataset on the service tier.

Next I will retrieve every row from the result set of the query using the `READ()` function, as follows:

```

QueryResults - OnAfterGetRecord()

IF TopX_Customer.READ THEN
BEGIN
    cus_No           := TopX_Customer.No;
    cus_Name         := TopX_Customer.Name;
    cus_TotalSales   := TopX_Customer.Sum_Sales_LCY;
    cus_TotalProfit  := TopX_Customer.Sum_Profit_LCY;
END
ELSE
    CurrReport.SKIP;
    
```

If the query returns less than `TopX` rows, I use the `currReport.SKIP` function so nothing is added to the dataset.

Then, I use the `Close` function in the `OnPostDataItem()` trigger to close the query object:


```

QueryResults - OnPostDataItem()
TopX_Customer.CLOSE;
    
```


Now you are ready. The report runs the query, adds the result set to the report dataset and you have the captions in the last row, as you can see in the following example:

About This Report: Packt - CH07-1										
cus_No	cus_Name	cus_TotalSales	cus_TotalSales...	cus_TotalProfit	cus_TotalProfi...	cus_NoCptn	cus_NameCptn	cus_TotalSales...	cus_TotalProfi...	TopX
10000	The Cannon Gr...	17100,96	###0.00	5338,26	###0.00	<>	<>	<>	<>	<>
43687129	Designstudio G...	14498,04	###0.00	12361,44	###0.00	<>	<>	<>	<>	<>
49858585	Hotel Pferdesee	14450	###0.00	5025,47	###0.00	<>	<>	<>	<>	<>
47563218	Klubben	11772,2	###0.00	4120,2	###0.00	<>	<>	<>	<>	<>
20000	Selangorian Ltd.	6510,64	###0.00	2511,94	###0.00	<>	<>	<>	<>	<>
30000	John Haddock L...	6142,9	###0.00	2921	###0.00	<>	<>	<>	<>	<>
49525252	Beef House	6000	###0.00	2471,69	###0.00	<>	<>	<>	<>	<>
49633663	Autohaus Miel...	4331,4	###0.00	1465,67	###0.00	<>	<>	<>	<>	<>
32656565	Antarcticopy	2582,81	###0.00	726,01	###0.00	<>	<>	<>	<>	<>
35963852	Heimilispyrdi	2024,21	###0.00	338,31	###0.00	<>	<>	<>	<>	<>
<>	<>	<>	<>	<>	<>	No.	Name	Sum_Sales_LCY	Sum_Profit_LCY	10

Now you can create a layout, based on the columns available in the dataset, as with any other report.

 An example of this report is available in the object: Packt - CH07-1

This demonstrates how you can use a query object to create a report dataset. Once you know how to do this, it's very simple to create a template report that you can reuse and modify, to link to any query that you create.

 It would be better if Microsoft decided to extend the report dataset designer to reference a query object as a data item directly, as with tables. But because there's a working solution or workaround, as described previously, I think this is not likely to happen in the near future.

Summary

In this chapter we have learned performance optimization techniques so that running a report consumes fewer resources and out of memory errors can be avoided. It's very important to optimize the dataset and layout of your report and avoid sending unnecessary information to the dataset or layout. You can do this by eliminating rows and columns, applying filters, formatting and sometimes by using C/AL code to process resource-intensive tasks on the server instead of in the report viewer, which runs on the client.

In the next chapter, I will discuss Word report layouts. You can now create a layout in Word in Dynamics NAV as an alternative to RDLC layouts. This makes it possible for end or non-technical users to create or maintain custom report layouts using Microsoft Office Word for small and simple reports.

8

Word Report Layouts

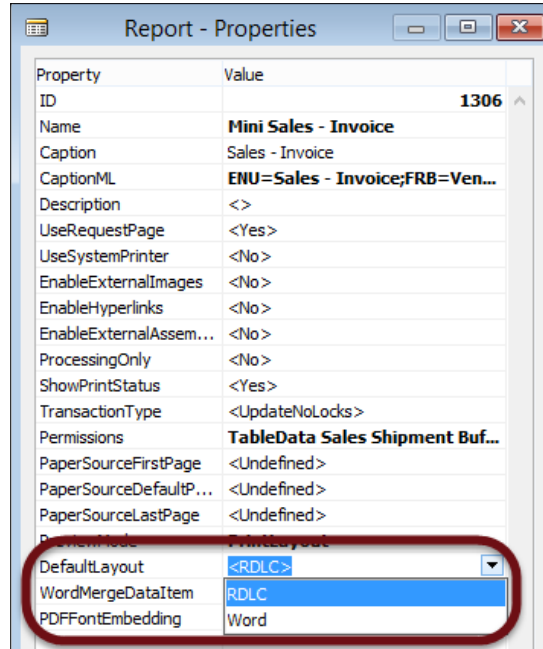
In this chapter, you will learn how to create a report layout using Microsoft Office Word. This is an alternative to the RDLC layouts you have created until now. Although the Word layout is designed for quite simple reports, it's a handy alternative and allows non-technical users to create and design custom layouts quickly. This chapter also discusses the management of report layouts and, last but not least, how to schedule report execution. How you create a Word report dataset makes it easier to create and maintain RDLC reports.

Introducing the Word report layout

There are two types of layouts that you can create for a report in Dynamics NAV: RDLC and Word. You can do this in two environments: the development environment and the Dynamics NAV RoleTailored Client.

The layouts that you create in the development environment are built-in layouts. You can create one built-in RDLC and one Word layout per report. One of these two layouts is then defined as the default layout for the report.

The default report layout is defined using the report property, **DefaultLayout**:



A user can create one or more custom layouts without using the development environment. These can be RDLC and/or Word layouts.

You need to install Report Builder to create an RDLC layout. You need Microsoft Word 2013 to create a Word layout.

One report can contain two built-in layouts (one RDLC and one Word) and multiple custom layouts. The custom report layouts are stored in the database and, when you run a report, the application uses code to determine which layout has to be rendered.

I will explain how to decide which report layout should be executed when you run a specific report in the section about managing report layouts. In this section, I will explain how you can create a built-in Word report layout. Once you understand how to build a built-in Word report layout, the process for creating a custom layout is very similar. The only difference is where you start from.

The advantage of Word report layouts is that users can create these custom layouts very quickly, without much technical knowledge, using the application and Microsoft Office Word.



There are some limitations in the Word report layout, when compared to the RDLC layout. For example, in RDLC you can use expressions to generate values for properties. You can also use the toolbox to create tables, lists, matrixes, charts, data visualizations, and so on. This is not possible in the Word layout. That is why a Word report requires more preparation and you will have to use C/AL code and reorder the data items, so it becomes clearer how to use the dataset in Word.

The advantage of using the Word layout is that you can use Word formatting toolbars, templates, color schemas, and so on. I recommend the Word layout as an alternative to the RDLC layout when you need a simple, straightforward layout that is bound directly to the dataset.

Creating a Word report layout

Let's start by creating a simple Word report layout from within the development environment.

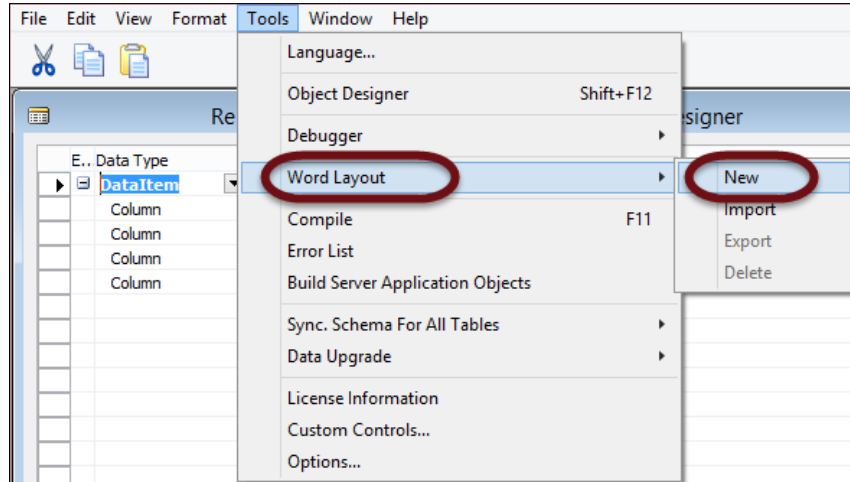
1. We will start with the following simple dataset:

...	Data Type	Data Source	Name	I...
▶	DataItem	Item	<Item>	
	Column	Item."No."	No_Item	
	Column	Item.Description	Description_Item	
	Column	Item."Unit Price"	UnitPrice_Item	
	Column	Item.Inventory	Inventory_Item	

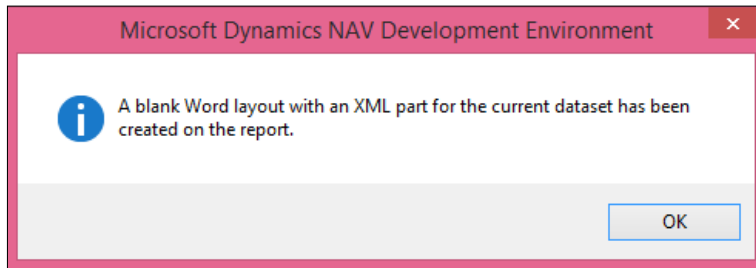


An example of this report, with only the dataset, is available in the object: Packt - CH08-3

2. We then use the **Tools, Word Layout** menu to create a new layout:



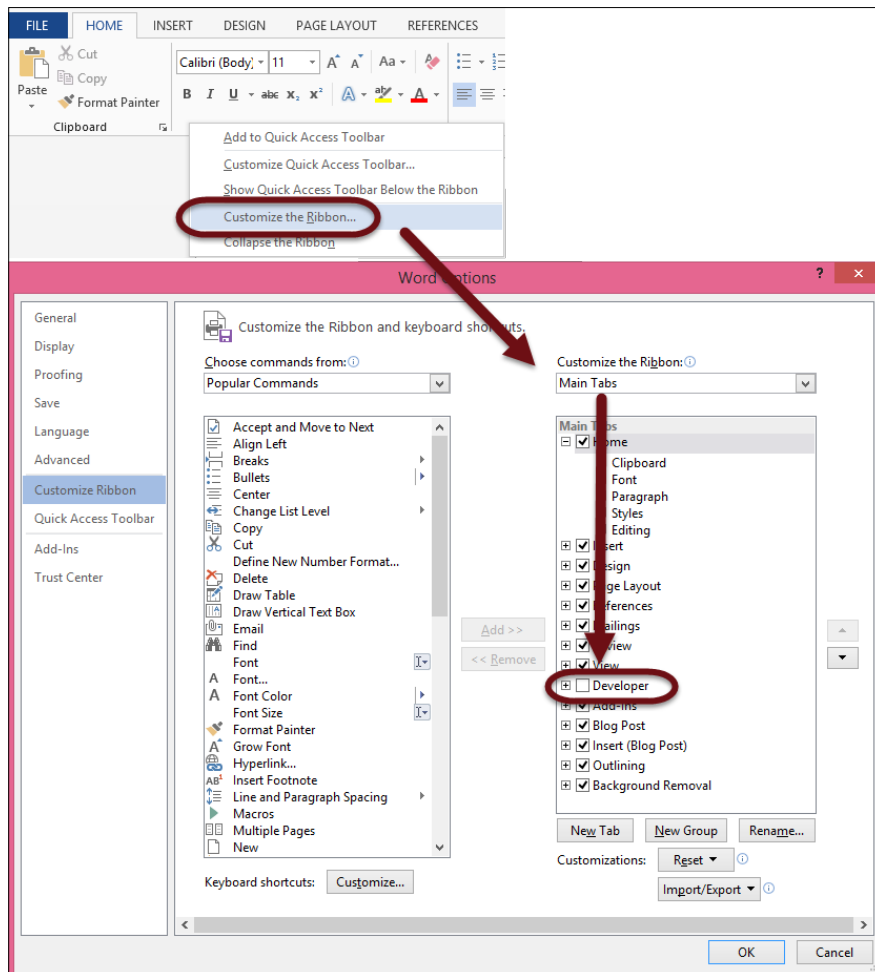
3. When you do this, the following message confirms the creation of the new layout:



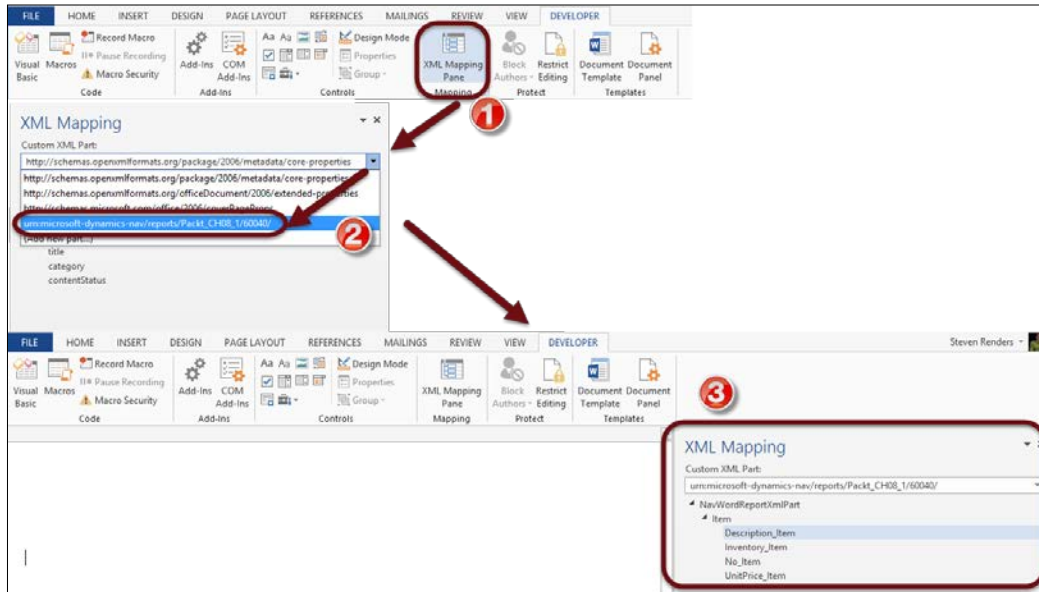
A Word document has thus been created, or should I say exported, from Dynamics NAV. You can now open it in Word, modify it, and then import it back again into the report object via the Dynamics NAV development environment. So we will open the Word document in Word.

When you open the Word document, it will be empty. You have to create a layout. This is similar to creating an RDLC layout. In Word, you use the fields from the **Report Dataset Designer** to create a layout. You can visualize the dataset with the developers tab in the ribbon in Word. If the developers tab is not visible, you will need to activate it by customizing the ribbon as follows:

1. Right-click on the ribbon and select **Customize the Ribbon....** Then, enable the **Developer** tab:



- This will display the **Developer** tab in the ribbon. There you need to select the **XML Mapping** pane to display the dataset, as follows:



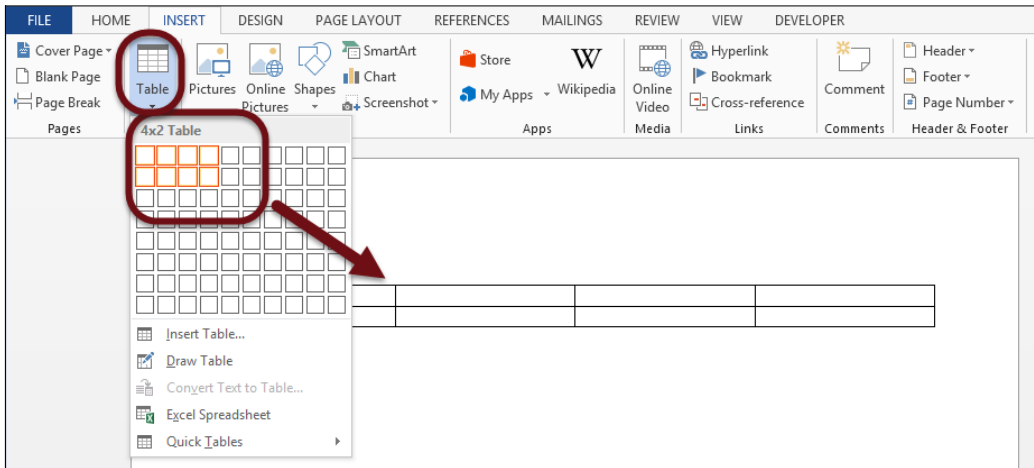
- Select the line that contains `urn:Microsoft-dynamics-nav/reports/reportname/reportnumber` from the drop-down menu in the **XML Mapping** pane.
- The **NavWordReportXmlPart** then displays the report dataset. Every data item is visible, with the name you used in the report dataset designer. When you click on a data item, in this example **Item**, it opens and displays the columns. Dynamics NAV translates a dataset in the report dataset designer into a Word XML format.

Jet Reports Express for Microsoft Word for Microsoft Dynamics NAV

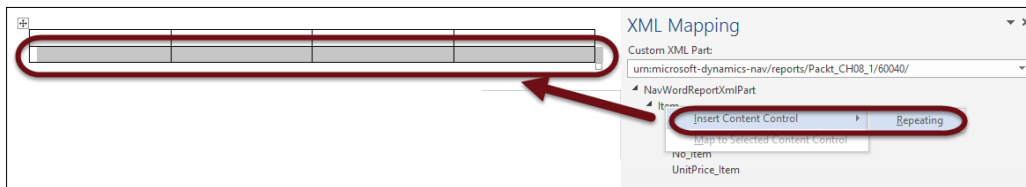


You can also use **Jet Reports Express** for Microsoft Word as an alternative to Microsoft Dynamics NAV. It's easier and quicker to see the dataset in Word using this add-on. The add-on also contains some predefined templates you can use and has a search bar, allowing you to find a field in the dataset more quickly. You can download it here: <http://www2.jetreports.com/1/3692/2014-08-28/366nvf>

- Next, we will add a layout to the Word document. We want to create a list of items that displays the columns I added to the dataset. To do that, we first need to add a table in the Word document. We will add a table with two rows, one for the header and one for the details, and a column for each field in the dataset:

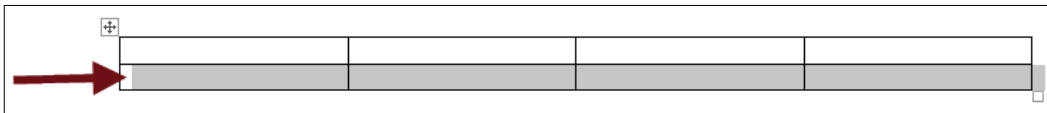


- We will type the names of the columns in the first row. Then, in the second row, we need to repeat the fields that are available in the dataset. In order to do that, I first need to add a repeater to it. As in the RDLC layout, if we drag the fields from the dataset directly into the table, they will not repeat for every record.
- We need to select the whole row and then add a repeating section, as shown in the following screenshot:



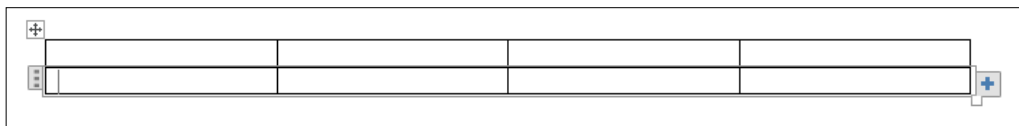
The steps to follow in order to create a repeating section are as follows:

1. First, double-click in the cell, which is in the first column, second row. Double-click multiple times until the complete row is selected, as in the following screenshot:



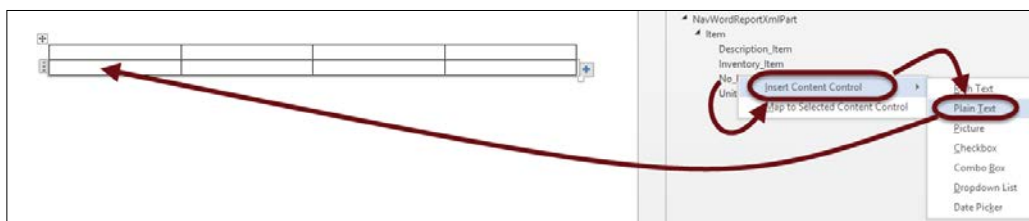
When the row is selected when it becomes gray.

2. Then, with the row selected, go to the **XML Mapping** pane and right-click on **Item**.
3. Then, select **Insert Content Control, Repeating**. The row in the table converts into a repeater, as shown in the following screenshot:



Now I can add fields from the dataset into the different columns of the repeater, as follows:

1. Click in the first textbox (first column, second row). This places the cursor inside the cell.
2. Then, right-click on the **No_Item** field in the **XML Mapping** pane and select **Insert Content Control, Plain Text**.



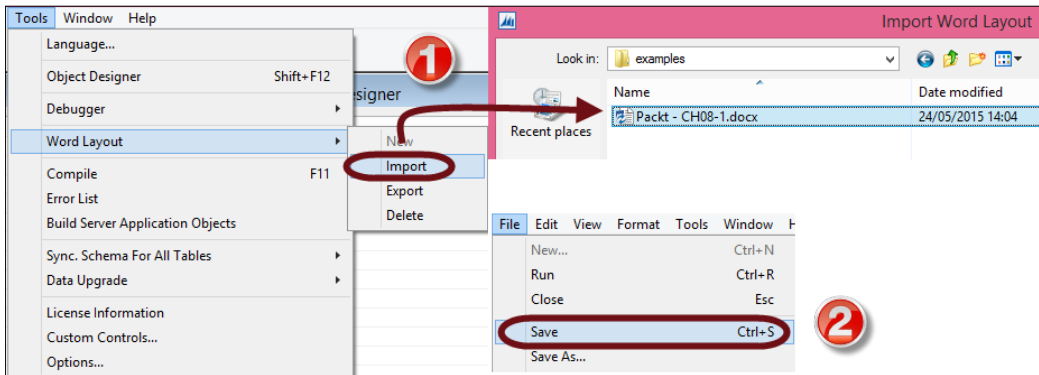


Note that you do not need to drag and drop the fields from the **XML Mapping** pane into the table. Always use the right-click button and the drop-down menu option **Insert Content Control**.

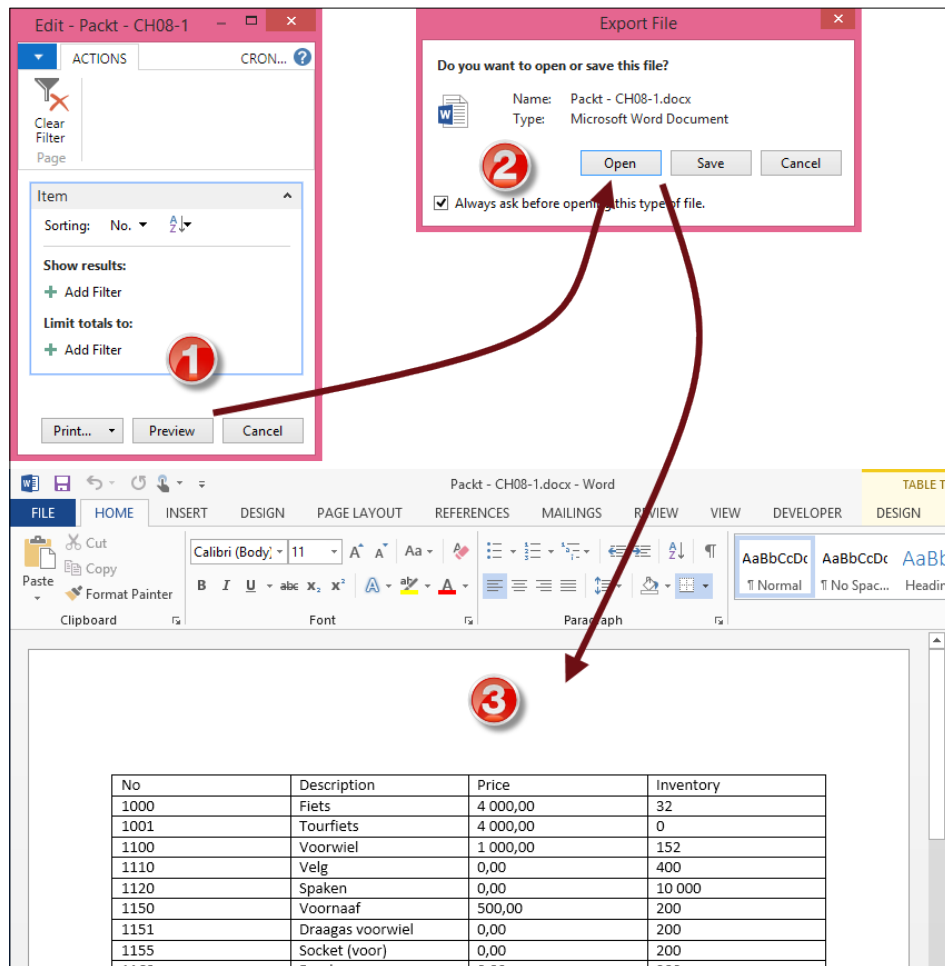
- I then repeat the process for the other fields and, in the header row, I simply type in the names of the columns. The result should look as follows:

No	Description	Price	Inventory
No_Item	Description_Item	UnitPrice_Item	Inventory_Item

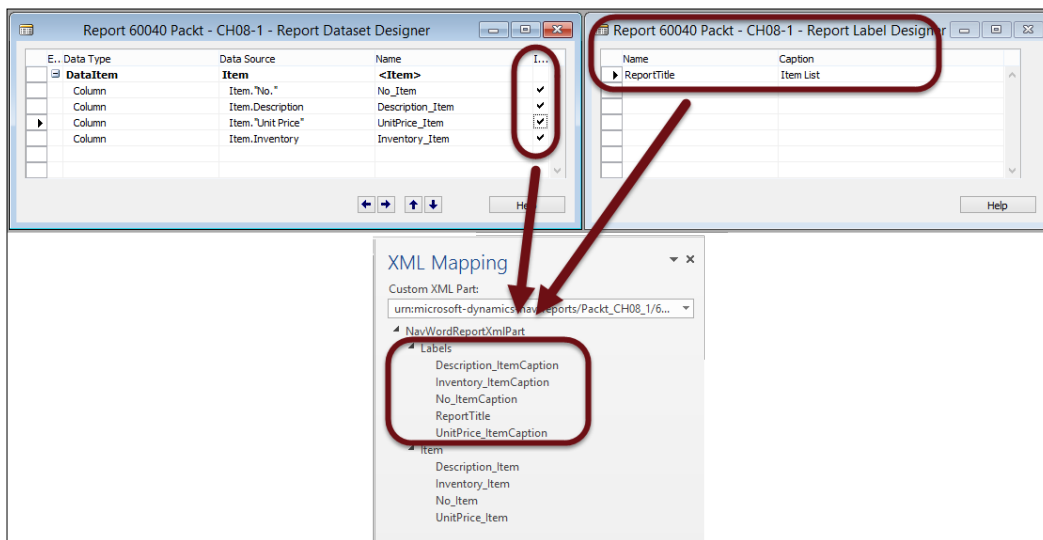
- I then save the Word document and close it. I will import the layout using the **Tools, Word Layout, Import** option in the development environment, from the **Report Dataset Designer**:



5. I then set the `DefaultLayout` property to **Word**, and save the report object. Now, when I run the report, the following happens:



6. The request page opens, as it does with an RDLC report. We can enter filters and/or select a key for sorting. A popup window asks if we want to **Open** or **Save** the layout and, when we select **Open**, the Word file opens and displays a list of items.
7. The column captions are not multi-language because we typed them in manually. Let's see how we can replace them with captions from the database.
8. We will use the **IncludeCaption** property on the columns in the dataset and include a **Label** (via **View, Labels**) that contains the report name. When you do this, you will have to:
 1. Save the report object.
 2. Export the Word layout.
 3. Open the Word layout.
 4. Open the XML mapping pane.
9. You should see a **Labels** section at the top of the **XML Mapping** pane:



10. Next, we will perform the following steps:

1. Replace the column headers with the labels in the dataset.
2. Save the Word layout.
3. Import it back again into the report dataset designer.
4. Save the report and run it.

This is what you will see after performing these steps:

The screenshot shows the XML Mapping window on the right, which maps XML elements to report fields. The mappings are:

- ReportTitle (XML) to ReportTitle (Report Field)
- No_ItemCaption (XML) to No_ItemCaption (Report Field)
- Description_ItemCaption (XML) to Description_ItemCaption (Report Field)
- UnitPrice_ItemCaption (XML) to UnitPrice_ItemCaption (Report Field)
- Inventory_ItemCaption (XML) to Inventory_ItemCaption (Report Field)

The report preview on the left shows the following table:

No.	Description	Unit Price	Inventory
1000	Prijs	4 000,00	0
1001	Tourfiets	4 000,00	0
1100	Voorwiel	1 000,00	152
1110	Velg	0,00	400
1120	Spaken	0,00	10 000
1150	Voornaaf	500,00	200
1151	Draagas voorwiel	0,00	200
1155	Socket (voor)	0,00	200
1160	Band	0,00	200
1170	Binnenband	0,00	200
1200	Achterwiel	1 200,00	152
1250	Achternaaf	1 100,00	200

It now displays the report title and column headers in the language of the user with the captions that are defined in the table.



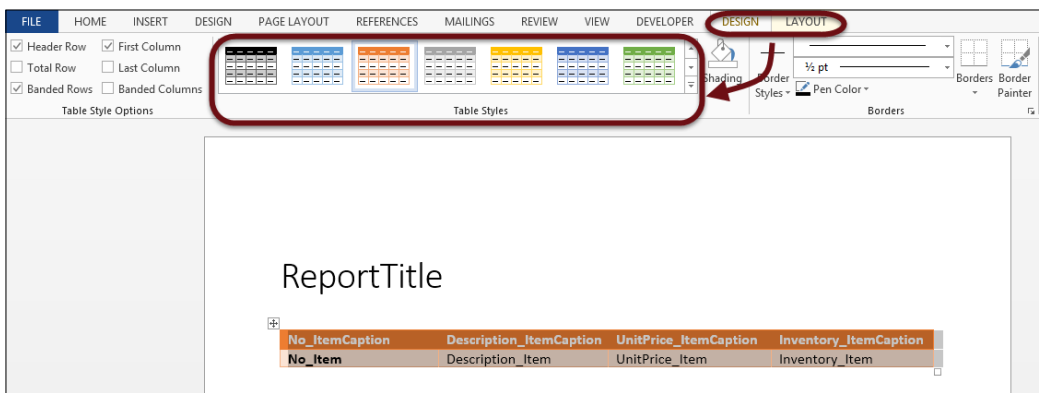
An example of this report is available in the object: Packt - CH08-1

Formatting the Word report layout

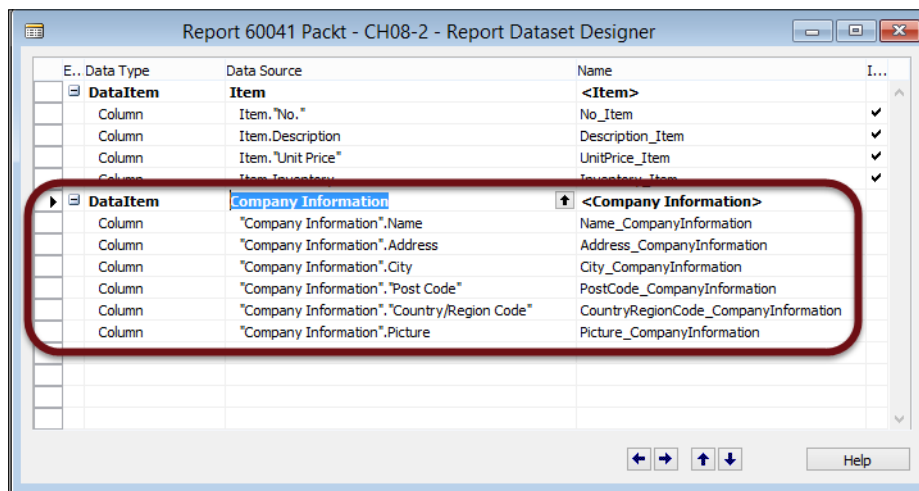
Now I'm ready to implement some advanced formatting in my report, so I will use the options available in Microsoft Word.

I have used a Word table because this is a report with a list or table layout and, I can also use the **Word Table Formatting** options.

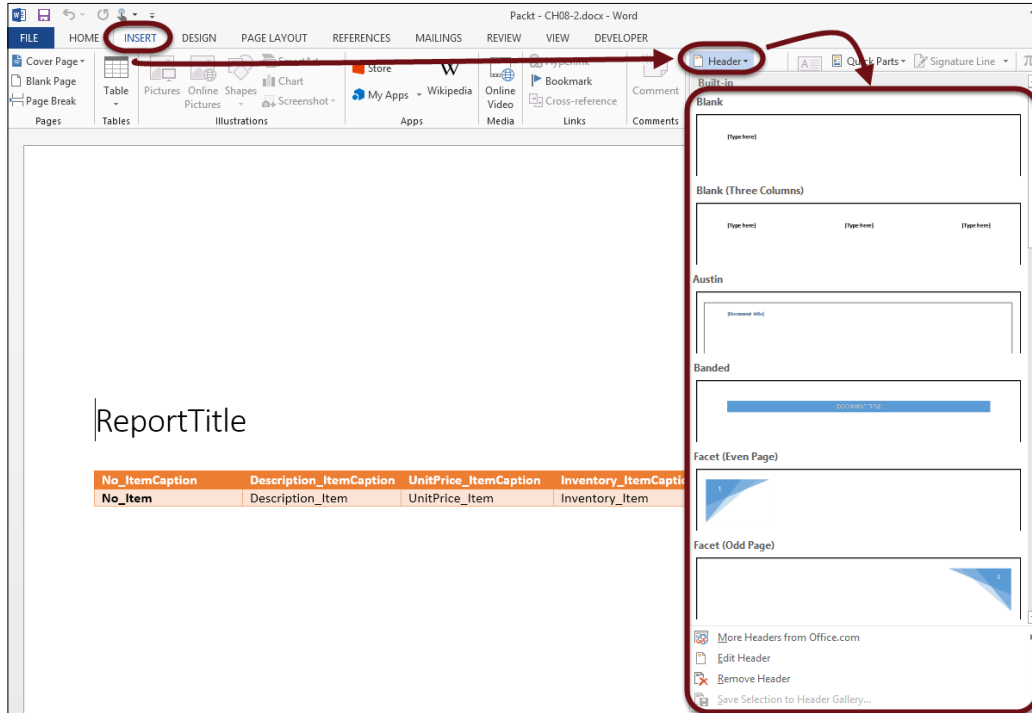
I will reopen the Word layout and select the table. You can see the **Design** and **Layout** tab in the ribbon, you can select a **Table Style** in the **Design** tab and apply it as follows:



I want to include company information and a logo, so I will first add the company information to the dataset as an extra row:

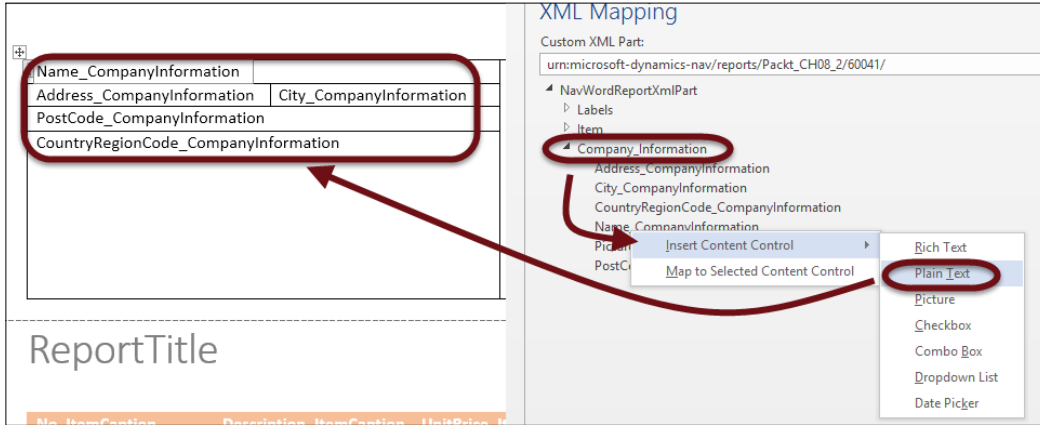


Then, I will add a page **Header** to the Word layout:

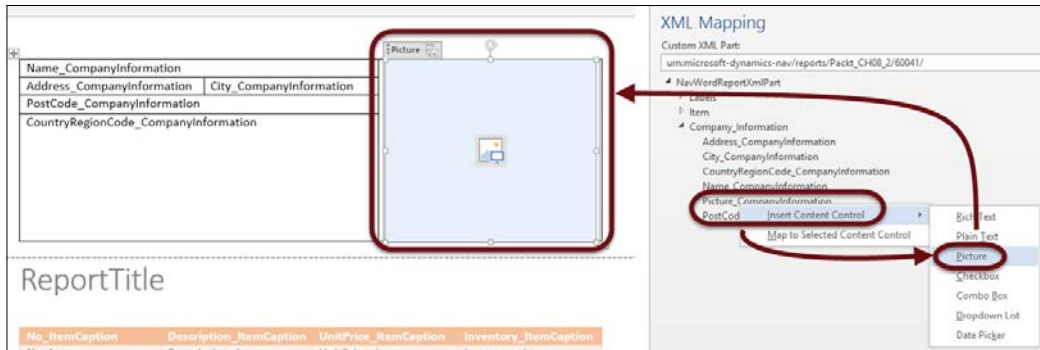


As you can see, Word offers different layouts for the header and you can access an online library that contains many more examples with the **More Headers from Office.com** option. Next, I will add a table to the header section. The table will hold the address fields for the company.


I then add **Plain Text** controls for the fields in the Company Information data item to the table fields in the header, as follows:



I right-click the field in the dataset and select the **Picture** option to add the company logo:




Now my layout is ready and I import it back again into the report dataset designer and save the report object. Then, when I run it, I see the following result:

CRONUS International Ltd.		
5 The Ring	London	
W2 8HG		
GB		


Item List

No.	Description	Unit Price	Inventory
1000	Bicycle	4 000,00	32
1001	Touring Bicycle	4 000,00	0
1100	Front Wheel	1 000,00	152
1110	Rim	0,00	400
1120	Spokes	0,00	10 000
1150	Front Hub	500,00	200
1151	Axle Front Wheel	0,00	200
1155	Socket Front	0,00	200
1160	Tire	0,00	200
1170	Tube	0,00	200
1200	Back Wheel	1 200,00	152
1250	Back Hub	1 100,00	200
1251	Axle Back Wheel	0,00	10 000
1255	Socket Back	0,00	200
1300	Chain Assy	800,00	152
1310	Chain	0,00	100
1320	Chain Wheel Front	0,00	100
1330	Chain Wheel Back	0,00	100
1400	Mudguard front	0,00	152
1450	Mudguard back	0,00	152
1500	Lamp	0,00	152
1600	Bell	0,00	152
1700	Brake	600,00	152
1710	Hand rear wheel Brake	0,00	200
1720	Hand front wheel Brake	0,00	200
1800	Handlebars	0,00	152
1850	Saddle	0,00	152
1896-S	ATHENS Desk	649,40	254
1900	Frame	0,00	152
1900-S	PARIS Guest Chair, black	125,10	299
1906-S	ATHENS Mobile Pedestal	281,40	254
1908-S	LONDON Swivel Chair, blue	123,30	305
1920-S	ANTWERP Conference Table	420,40	96
1924-W	CHAMONIX Base Storage Unit	136,40	26
1928-S	AMSTERDAM Lamp	35,60	272

When you print the report depicted in the preceding screenshot, all pages contain the header, and the header contains the company information, including the company logo.

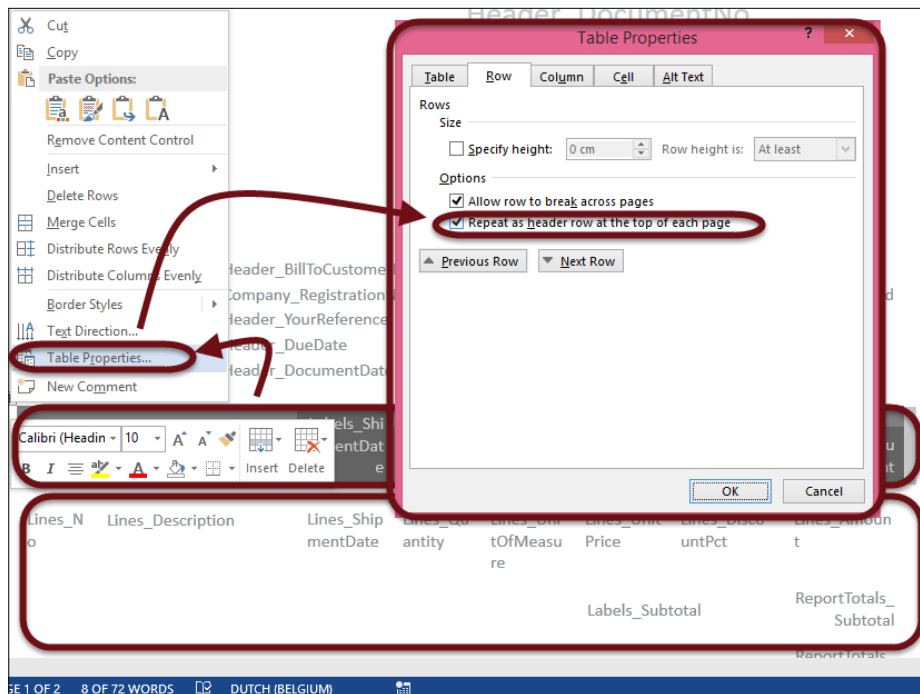
 The Dynamics NAV image types need to be supported by Microsoft Word. Word supports the following image types: .bmp, .jpeg, and .png.

You can now go a step further and use other Word options, for example, page numbers, quick parts, smart-arts, and so on.

 An example of this report is available in the object: Packt - CH08-2


Repeating a table header

Word also has a table property that enables the repeating of the header row, which is not possible in RDLC. When you edit a Word layout that contains a table, you can select the header row and open its property, as shown in the following screenshot:



When you enable **Repeat as header row at the top of each page**, it does what it says. In the following screenshot, you can see that the header row is repeated on the second page:

1920-S	ANTWERP Conference Table	420,40	96
1924-W	CHAMONIX Base Storage Unit	136,40	26
1928-S	AMSTERDAM Lamp	35,60	272

CRONUS International Ltd.		
5 The Ring	London	
W2 8HG		
GB		

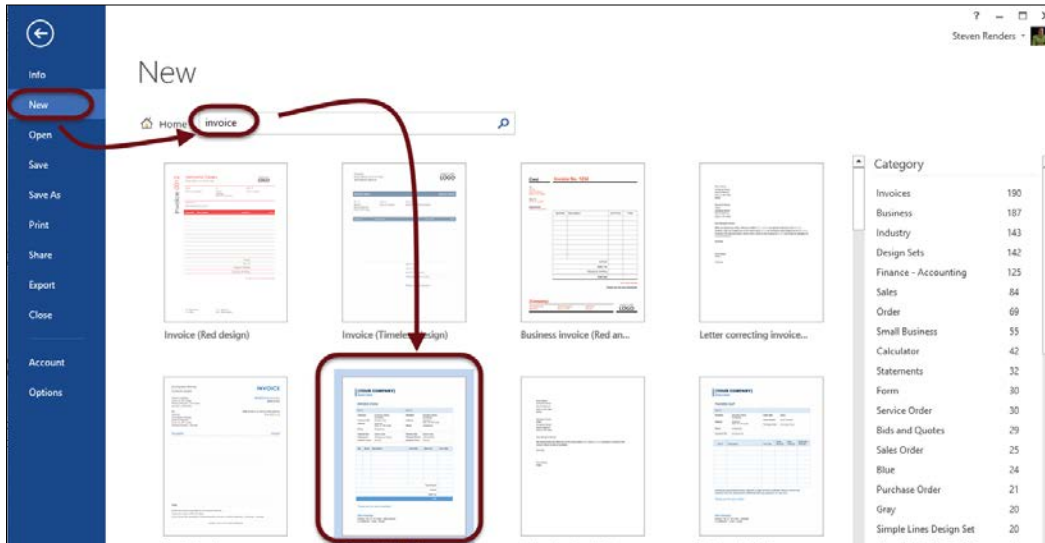
No.	Description	Unit Price	Inventory
1928-W	ST.MORITZ Storage Unit/Drawers	342,10	67
1936-S	BERLIN Guest Chair, yellow	125,10	136
1952-W	OSLO Storage Unit/Shelf	158,50	15

Using Word templates

When designing a layout, reusing an existing template can be a great time-saver. When you create a new Word document, Word has built-in templates that you can use and you can even download extra templates from the online gallery.

I will first create a Word document and import that into the custom layout that I created in Dynamics NAV.

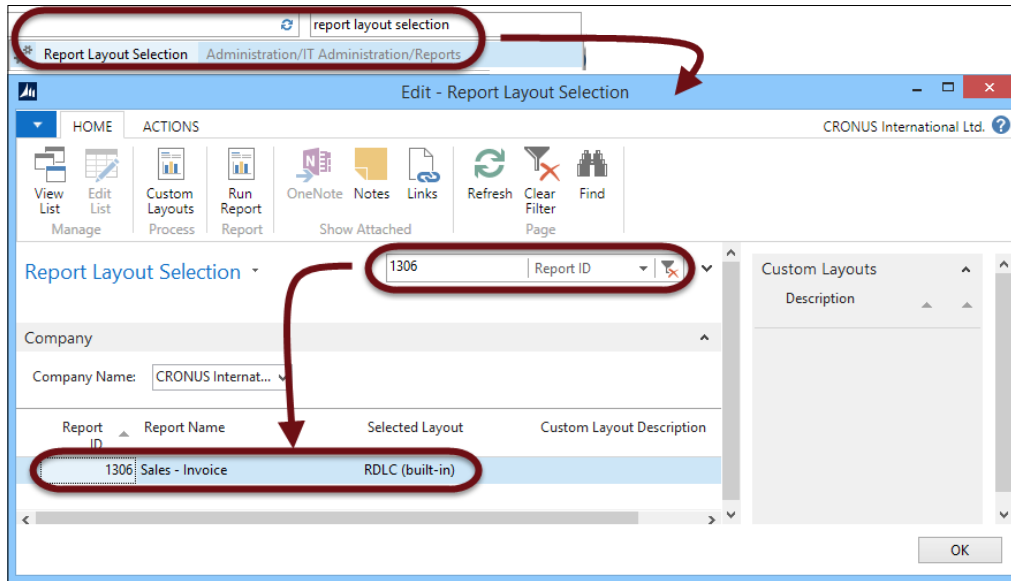
To create a new Word document, I will open Word manually and then select **New**, to use an existing template, as follows:



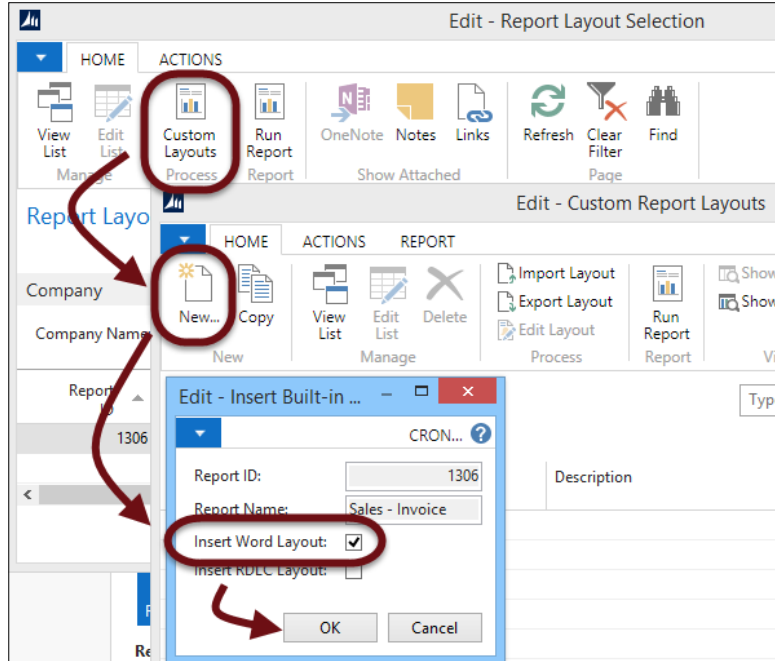
I then save the Word document and close it. I have named it InvoiceTemplate.docx.

This Word document does not contain the dataset from report 1306. To add it, I will now import the Word document into Dynamics NAV using the **Import Layout** button. Dynamics NAV then adds the dataset to the Word document. I will then export it again so that I can edit it in Word and map the elements from the dataset. I will start from a report in Dynamics NAV that already has a very nice dataset, report 1306, the mini sales invoice.

A user can go to **Report Layout Selection** and filter the page on ID 1306:



I will select the **Custom Layouts** action in the ribbon, followed by **New**:

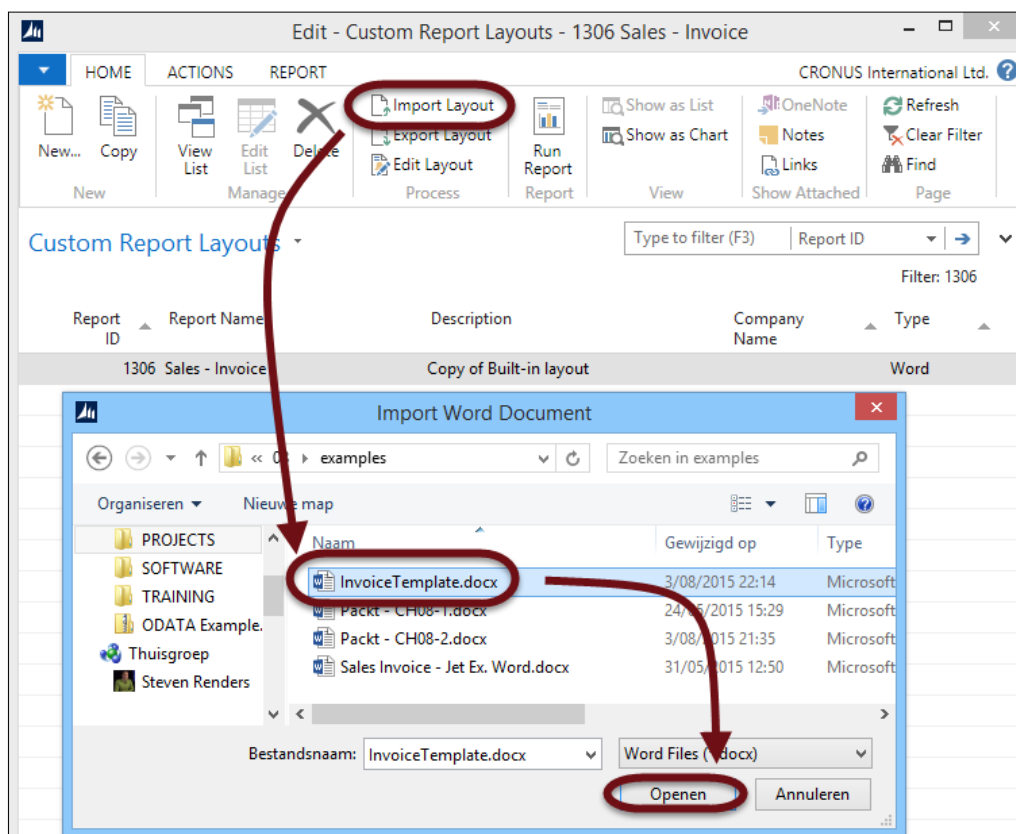


Word Report Layouts

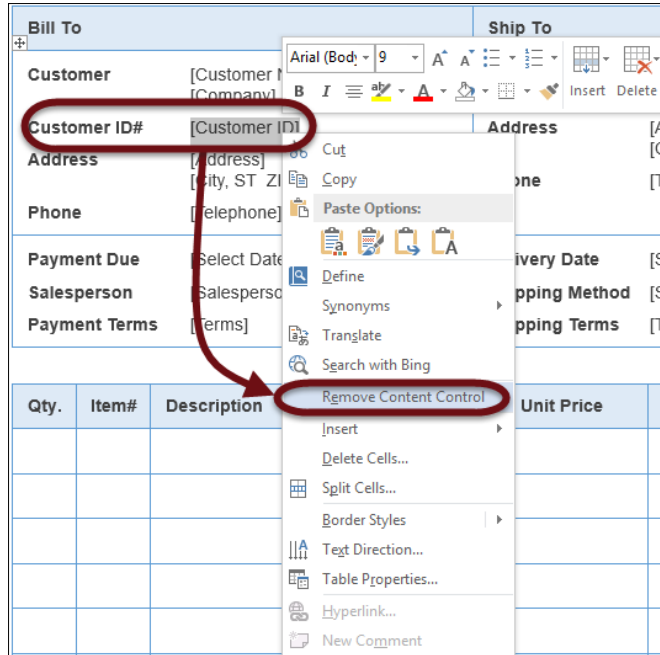
I have now created a new custom layout in Dynamics NAV, which you can see in the list here:

Report ID	Report Name	Description	Company Name	Type
1306	Sales - Invoice	Copy of Built-in layout		Word

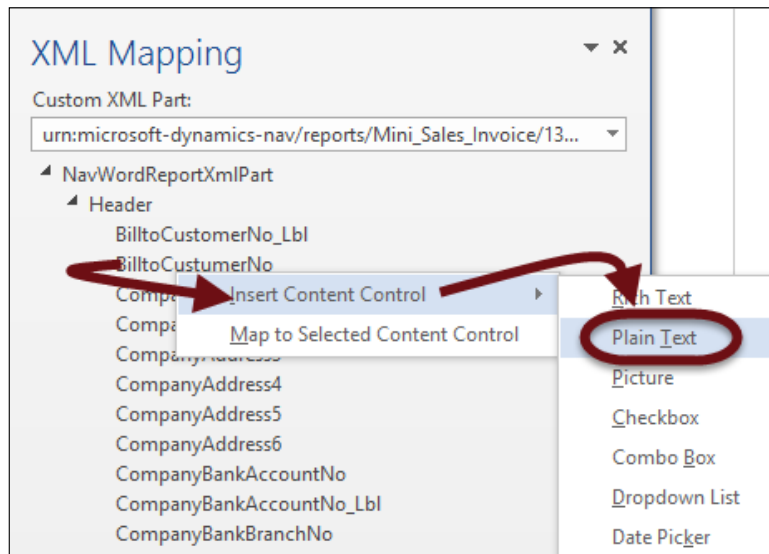
Now I need to import the Word document, with the invoice layout, into Dynamics NAV. I will use the **Import Layout** button to import the Word document that I just created into report 1306:



Remove the current content control from the layout, as follows:



Then, insert a new content control in the XML Mapping pane:



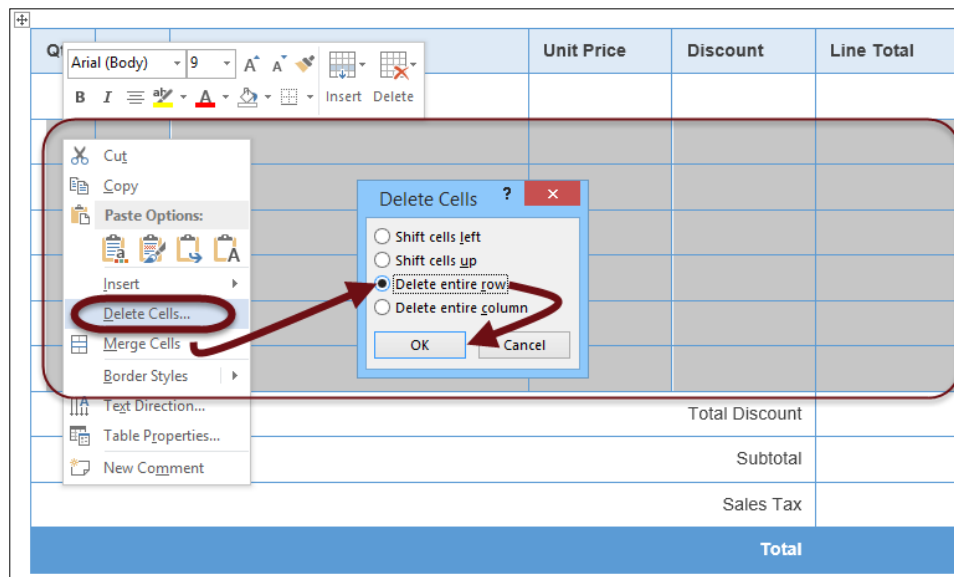
The result will look as follows:

Bill To	
Customer	[Customer Name] [Company]
Customer ID#	BilltoCustomerNo
Address	[Address] [City, ST ZIP Code]
Phone	[Telephone]

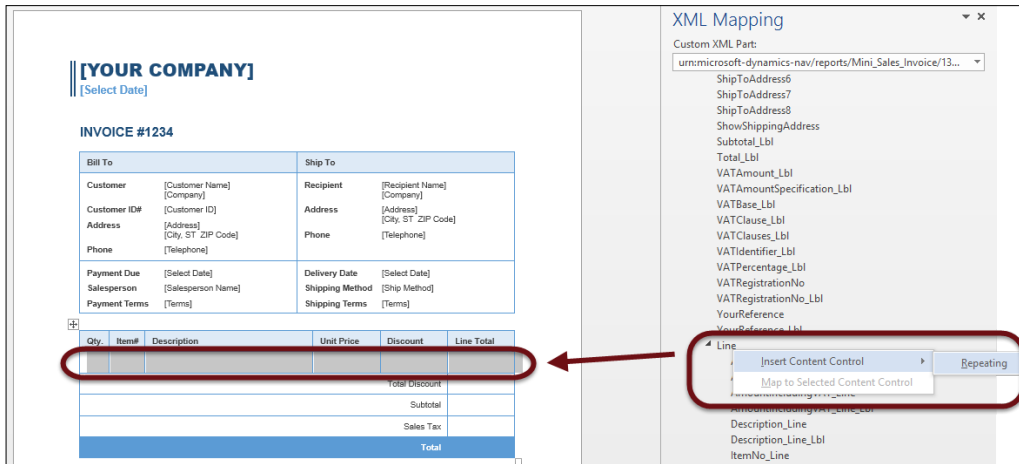
You now need to repeat these steps for all of the fields in the header and the first table in the Word document.

Now, go to the Lines data item and insert a repeating section, as follows:

First, select and delete all rows from the detail table, except one:

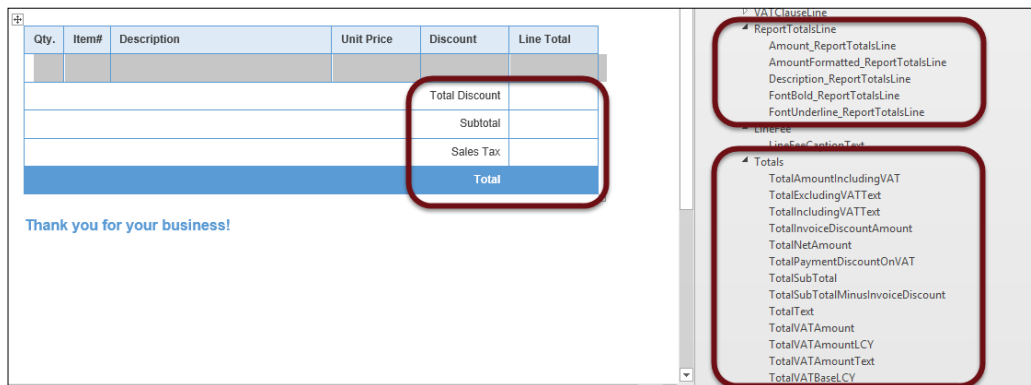


I'm going to convert this remaining row into a repeating content control by selecting the row (double-click until it is selected) and then selecting **Insert Content Control, Repeating** in the **XML Mapping** pane, from the **Line** data item:



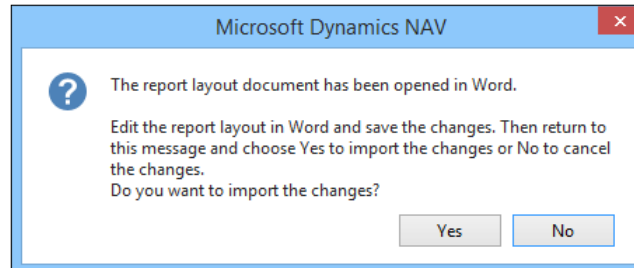
I can then add the fields to the repeater from the line data item as **Plain Text Controls**, as I did at the beginning of this chapter.

There are also totals below the **Lines** data item, in the **XML Mapping** pane, as you can see here:




These are special data items that contain the totals and were added to report 1306, so that you can find them very quickly and use them in the Word layout.

You can then close Word. Dynamics NAV detects this and asks the following question:



Click **Yes** and your new layout will be imported into Dynamics NAV and becomes your new custom layout for report 1306.

As you can see, there is a wide choice of simple and advanced templates in the online gallery. It's very easy to link a template to a Dynamics NAV report and then use the report dataset fields in your template.

 If you want users to use Word templates, then I suggest spending some time on creating similar datasets for your reports, based on how it's done in report 1306. Give the tables and fields proper and easy to understand names and create a similar structure of not too many data items. For most document reports one header, lines and totals should be enough. Keep it simple and your users will also find it easy to use.

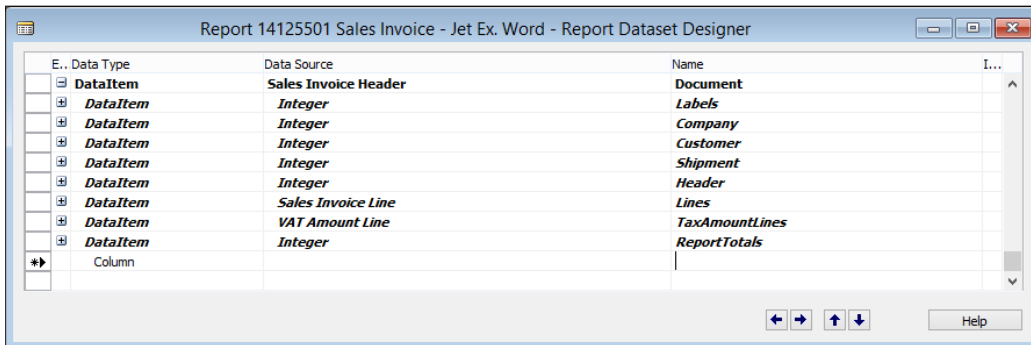
Optimizing your dataset for Word reports

Creating a report with a Word layout is usually divided into two steps. A developer creates the report object and the dataset in the development environment, and a user creates the Word layout using the application. As a developer, you will need to take performance into account, as is always the case when you create a dataset, but even more important is that you focus on making it easy for a user to create or modify the Word layout.

This means that you need to apply some simple guidelines to create a user-friendly and simple dataset and, in this section, I will provide some recommendations on how to do that.

As a developer, you should not always reinvent the wheel, and there are many examples of Word reports available for you to use and study. One of those stunning examples comes from Jet Reports. You can download .rpt and Word files, based on the Dynamics NAV demo database. In the following example, I will be using the **Report 14125501 Sales Invoice - Jet Ex**, which I downloaded from the Jet Reports website, which you can find here: <http://jetreports.com>.

When I open the report, I can see the following dataset:



As you can see, there are two levels of indentation: the Sales Invoice Header, which has been named Document, and the other data items. Each of the data items has clear and intuitive names:


- Document
- Labels
- Company
- Customer
- Shipment
- Header
- Lines
- TaxAmountLines
- ReportTotals

This makes it clear what to expect in each data item. When you expand the data items, the fields from the document header table are divided over several data items. This is a very good technique for grouping fields that belong together. A user can then see them together in the Word dataset.


Labels contain all of the labels of the report. Each label has an easy to understand name. Similarly, the company data item contains the fields from the company, which are usually put onto the header of a report. The same is valid for Customer and Shipto information.

It is important in a Word layout report that the lowest level of iteration of the data items matches the report layout. In this example the lowest level is level 2 and it contains the lines, VAT, and so on.

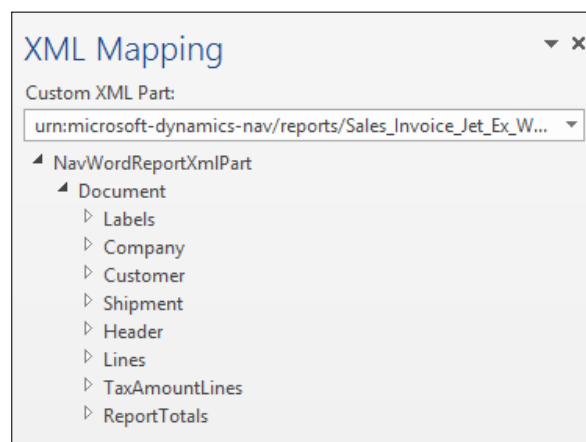
The totals are all in the same data item. Unlike RDLC, Word is not good at calculating totals. So you need to do that in the dataset.

 Although Word layouts create great flexibility for an end user to create and manage report layouts, it lacks the advantage of being able to use expressions, as in RDLC. In order to work around this limitation, you need to make sure you prepare these things in the dataset, via the C/AL code.

This is done here in the `ReportTotals` data item. The variables in the report totals are calculated in the triggers of the other data items, as you can see when you open the C/AL code. As an alternative, you could also use the `Report Total Buffer` table as a temporary table, as in **Report 1306 Mini Sales - Invoice**.

 The integer data items in this report are used to group fields together. To make sure they only add one line to the dataset, the data item property `MaxIteration` is set to 1.

So, to summarize, thinking about how you group and name data items and fields produces an easy to understand dataset in Word, as you can see in this screenshot:



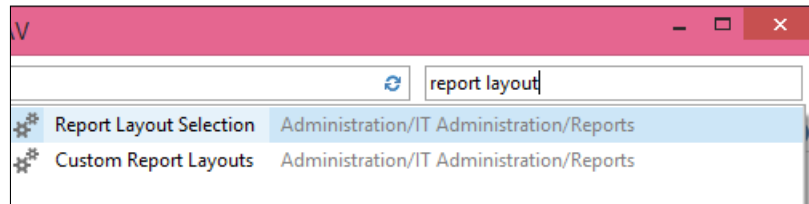
This dataset is self-explanatory, and that's how it should be for all reports that you want a non-technical user to edit.



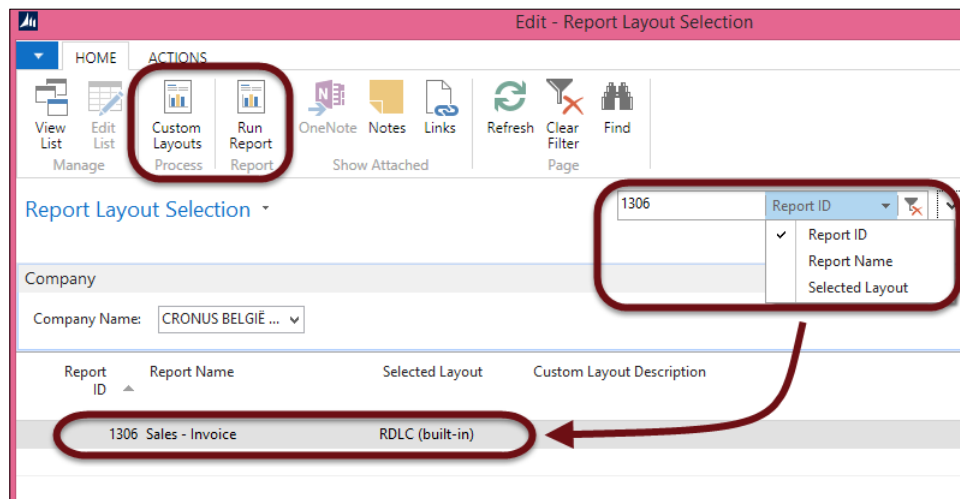
Designing the dataset of a report such as the one in the above example will not only make it easier in the Word layout, but also in the RDLC layout. Using this best practice will make report design easier and also make your reports much easier to maintain.

Managing report layouts

Managing Word report layouts can be done in the Dynamics NAV application. You use the **Report Layout Selection** page, which you can find using the search box or in **Administration, IT Administration, Reports**:



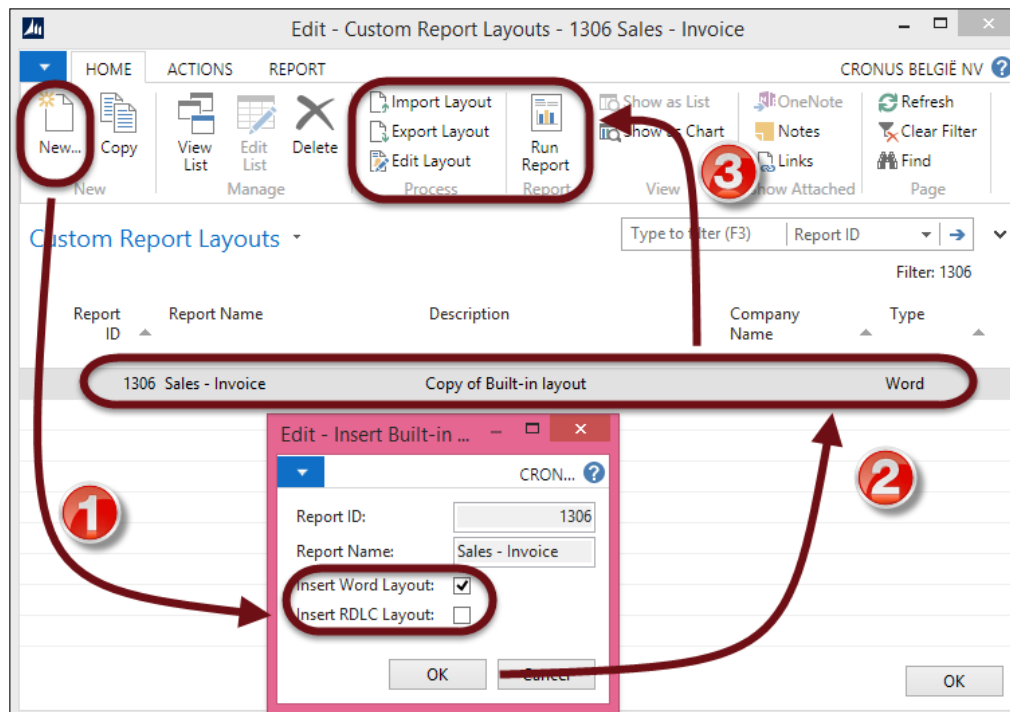
When this window opens, it first loads a list of all the reports available in the application. You can filter the list on the top, as in the following example from **Report ID 1306**:




You can create or edit a custom layout for the selected report with the **Custom Layouts** and **Run Report** buttons.

Custom layouts

When I select the **Custom Layouts** button, the **Edit - Custom Report Layouts** window opens:

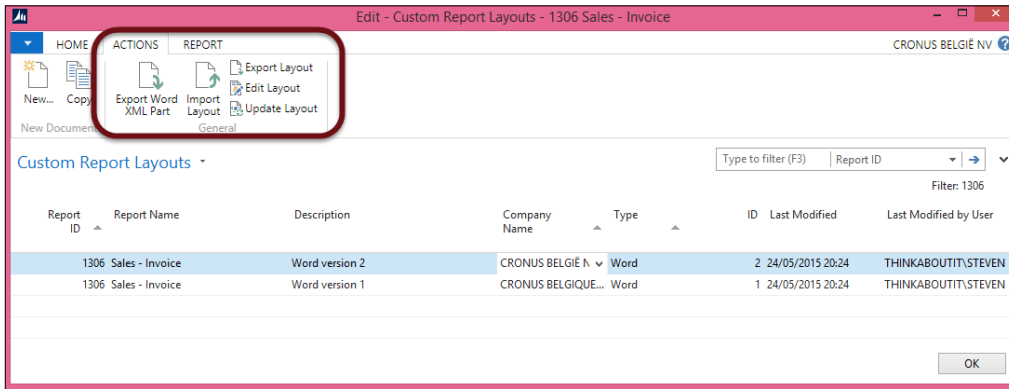


As you can see, when you select the **New...** button, you can create a new Word and/or RDLC layout for the report. When you do this, a new line is added. You can then select the line and use the **Import Layout**, **Export Layout**, and **Edit Layout** buttons to manage the new layout.

[ In the previous section, *Using Word templates*, I used this window to create, import and edit the *Invoice* template.]


After you have created or updated the layout, you can then use the **Run Report** button to run the report with the new layout. There's no restriction on the number of layouts you can create in a report. This means, of course, that one report could potentially have several RDLC and several Word layouts.

If you select the **Actions** menu, you will see the following actions:



There are two actions in the **Actions** group, which are not, by default, in the **Home** group: **Export Word XML Part** and **Update Layout**.

The **Update Layout** action is used when someone has changed the report dataset in the report dataset designer. If that happens and you try to run the report, an error message is displayed. You can then select the update layout option, or you can use the update layout action from this window. It will try to fix the error in the report layout, so you don't have to correct it manually.

 You can use the Company Name field, to restrict the report layout to a specific company. One database can contain multiple companies and so you can have a different report layout for every company. If you leave the Company field blank, then the layout is available for all companies in the database.

When you select the **Export Word XML Part** button, the Word report layout dataset is exported. You can open it in Notepad:

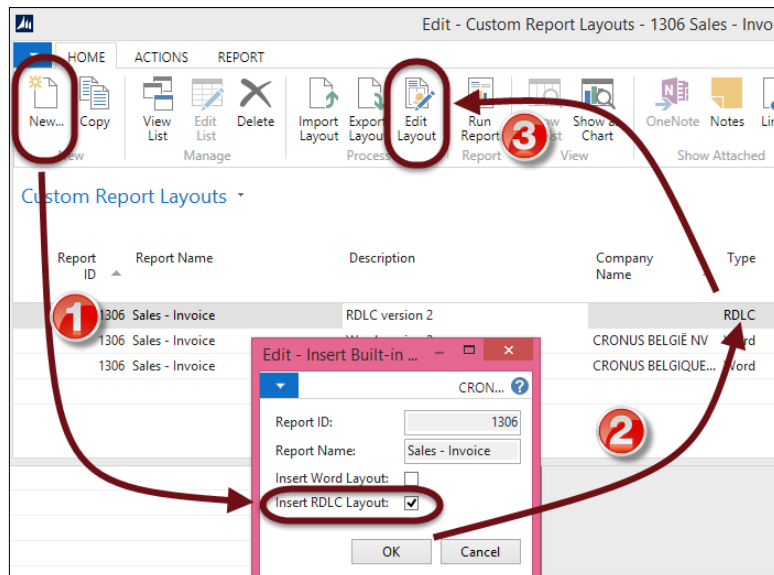
```

Defaultv2.xml - Notepad
File Edit Format View Help
<?xml version="1.0" encoding="utf-16"?>
<NavWordReportXmlPart xmlns="urn:microsoft-dynamics-nav/reports/Mini_Sales_Invoice/1306/">
  <Header>
    <BilltoCustomerNo_Lbl>BilltoCustomerNo_Lbl</BilltoCustomerNo_Lbl>
    <BilltoCustomerNo>BilltoCustomerNo</BilltoCustomerNo>
    <CompanyAddress1>CompanyAddress1</CompanyAddress1>
    <CompanyAddress2>CompanyAddress2</CompanyAddress2>
    <CompanyAddress3>CompanyAddress3</CompanyAddress3>
    <CompanyAddress4>CompanyAddress4</CompanyAddress4>
    <CompanyAddress5>CompanyAddress5</CompanyAddress5>
    <CompanyAddress6>CompanyAddress6</CompanyAddress6>
    <CompanyBankAccountNo>CompanyBankAccountNo</CompanyBankAccountNo>
    <CompanyBankAccountNo_Lbl>CompanyBankAccountNo_Lbl</CompanyBankAccountNo_Lbl>
    <CompanyBankBranchNo>CompanyBankBranchNo</CompanyBankBranchNo>
    <CompanyBankBranchNo_Lbl>CompanyBankBranchNo_Lbl</CompanyBankBranchNo_Lbl>
    <CompanyBankName>CompanyBankName</CompanyBankName>
    <CompanyBankName_Lbl>CompanyBankName_Lbl</CompanyBankName_Lbl>
    <CompanyCustomGiro>CompanyCustomGiro</CompanyCustomGiro>
    <CompanyCustomGiro_Lbl>CompanyCustomGiro_Lbl</CompanyCustomGiro_Lbl>
    <CompanyEMail>CompanyEMail</CompanyEMail>
    <CompanyGiroNo>CompanyGiroNo</CompanyGiroNo>
    <CompanyGiroNo_Lbl>CompanyGiroNo_Lbl</CompanyGiroNo_Lbl>
    <CompanyHomePage>CompanyHomePage</CompanyHomePage>
    <CompanyIBAN>CompanyIBAN</CompanyIBAN>
  </Header>
</NavWordReportXmlPart>

```

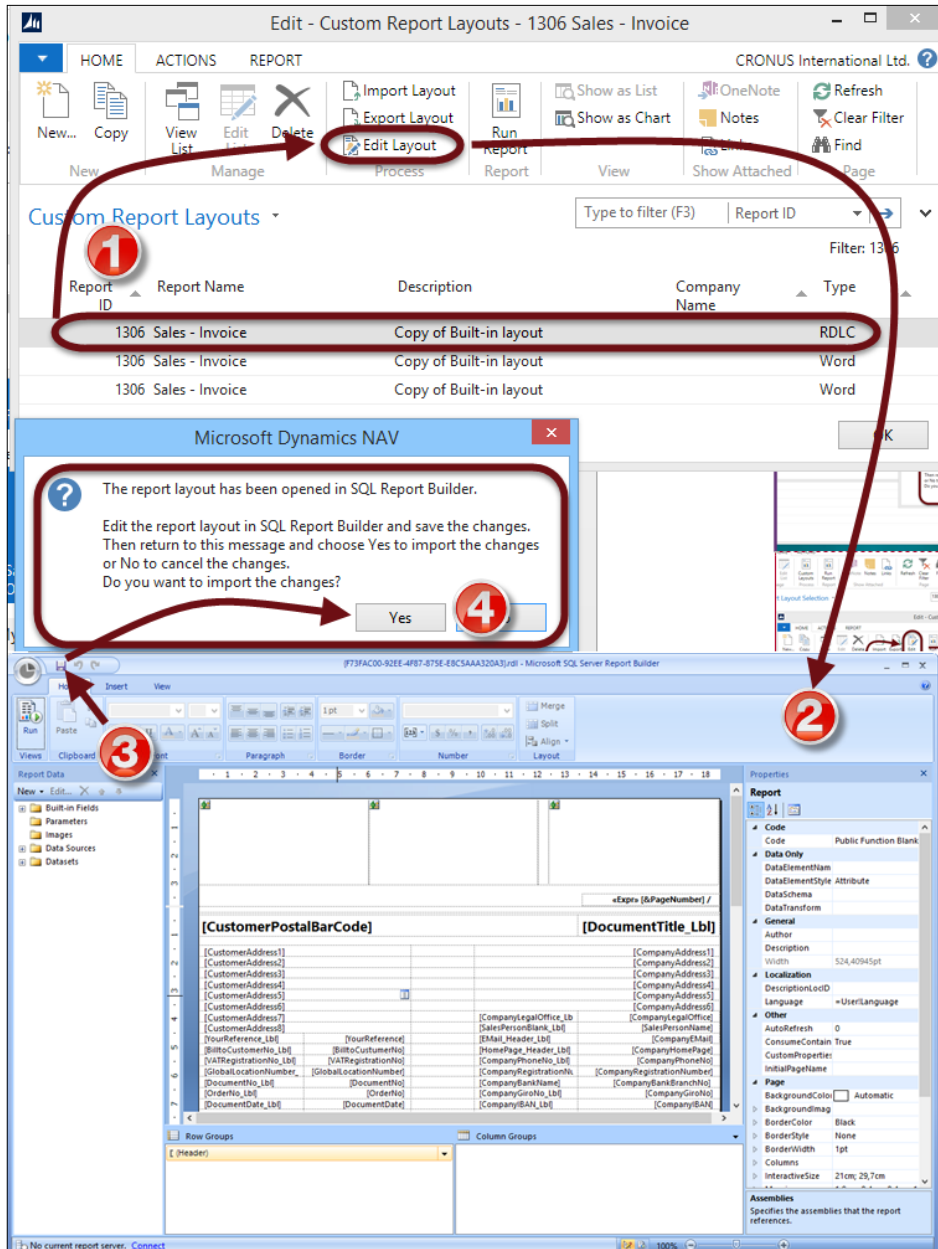
Editing a Custom RDLC layout

You can create a new RDLC layout for a report in the **Custom Report Layouts** window, using the **New** button:



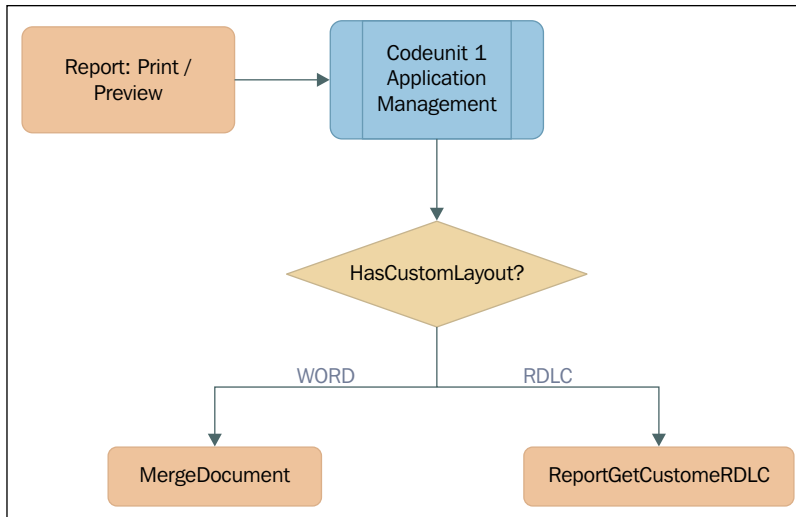
Word Report Layouts

Once the new line has been added, you can select the **Edit Layout** button for the RDLC layout. This will trigger Report Builder so that you can edit the layout. You then need to save the layout in Report Builder and close it. Next, you need to click on **Yes** to import the updated RDLC layout:



The report execution flow

The application needs to determine when to run which layout because a report can have built-in layouts (RDLC and/or Word) and custom layouts (RDLC and Word). This is handled via several codeunits that follow a report execution flow. The following figure visualizes this flow:

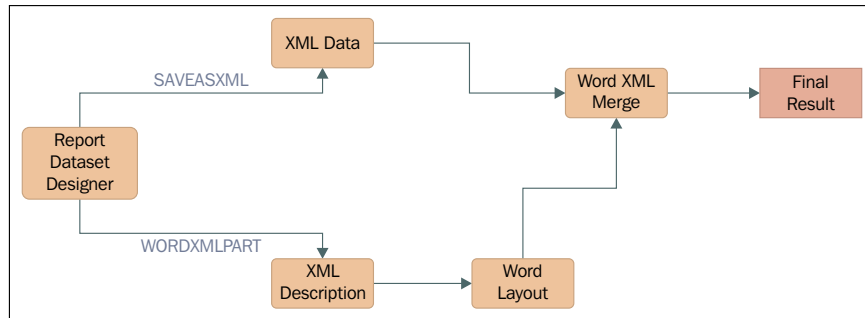


This flow is executed when you run a report from the application and select **Print, Preview** or when you run a report via C/AL code:

- First, the codeunit uses the `HasCustomLayout()` function to determine if the report has been set up with a custom RDLC or Word layout. If not, the default built-in layout is used.
- If the report has a custom or built-in Word layout, the `MergeDocument` function manages the report execution.
- If the report has been set up to use a custom RDLC layout, the `ReportGetCustomRDLC` function returns the custom RDLC layout as an XML string and that is used to render the report.

The Word report execution flow

When the report uses a Word layout, the execution flow is separated into two parts: the design time and the runtime flow.



At design time

When a user creates a Word report layout in the application, the dataset of the report needs to be sent to Microsoft Word. This is done via an XML file, a schema file. This XML schema is generated by the `REPORT.WORDXMLPART` function. This is done automatically by Dynamics NAV, but you could create custom functionality based on this logic using this function.

At runtime

When a report is executed, the report data output (the XML file) is merged with the Word layout to produce the final result. The report data output is in the XML format, which is in the same form as the output of the `REPORT.SAVEASXML` function. In the **Codeunit 1 Application Management**, there's a `MergeWordLayout` function that handles the merge, depending on the action that the user has selected: PDF, Word, Excel, or print.

Managing layouts in code

When I introduced the report execution flow, I mentioned that the application code determines which layout needs to be used when. This flow starts from codeunit 1, application management, and uses several other tables and codeunits. If you spend a little time investigating this codeunit, then you will discover how you can use it to your advantage when you are confronted with managing report layouts so that you don't reinvent the wheel.

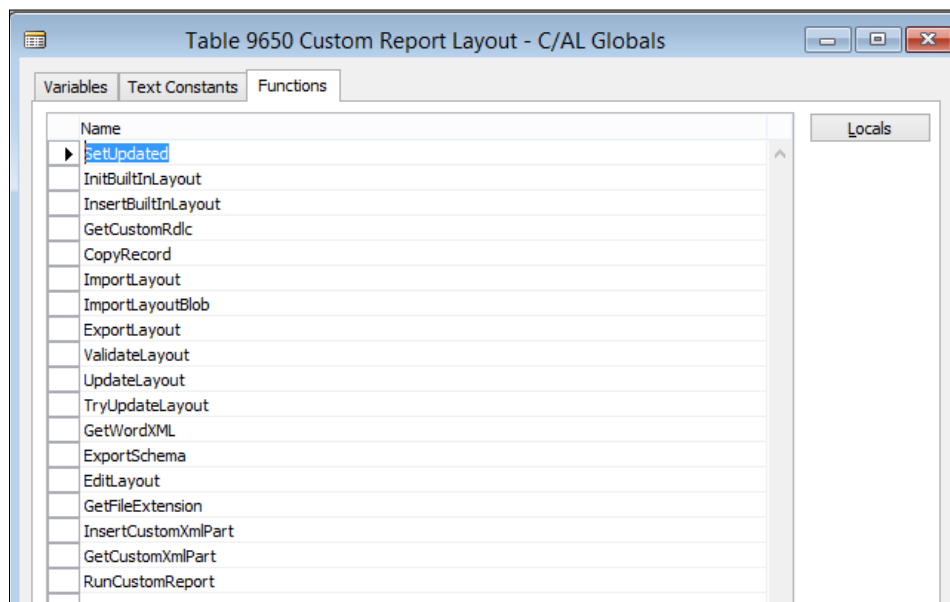
In codeunit 1, application management, and in the Custom Report Layout and Report Layout Selection tables, you will find many functions for managing reports and report layouts.

In codeunit 1, application management, there's a HasCustomLayout function, that determines if a report has a custom layout or not, and if so, whether it's Word or RDLC:

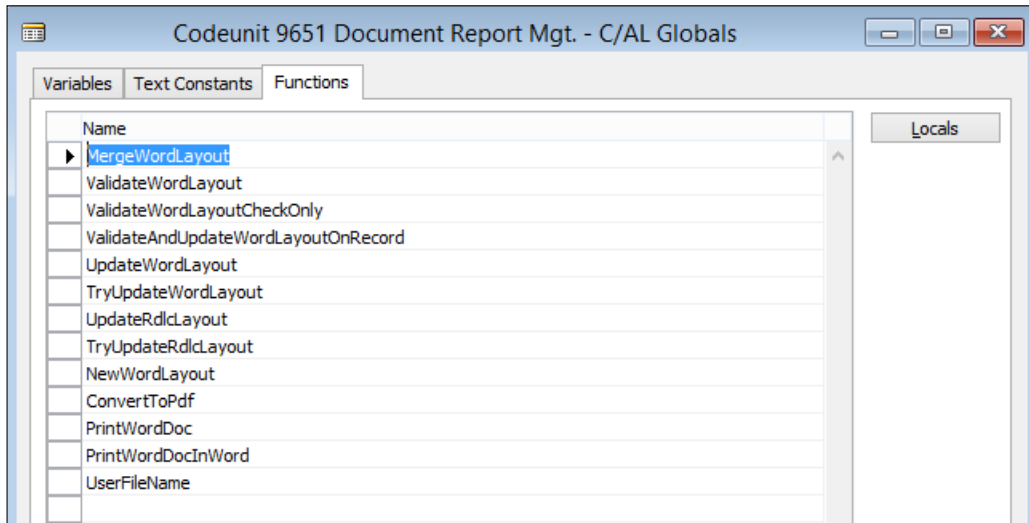
```
// Return value:
// 0: No custom layout
// 1: RDLC layout
// 2: Word layout
IF ObjectType <> ObjectType::Report THEN
    ERROR(NotSupportedErr);

EXIT(ReportLayoutSelection.HasCustomLayout(ObjectID));
```

It calls a HasCustomLayout function, which is from the table 9651, Report Layout Selection. If a report has a custom layout, it's stored in that table, so that's where you can find it. There's a reference to the **Table 9650 Custom Report Layout** in this function. When you open that table and have a look at its functions, you will see the following:



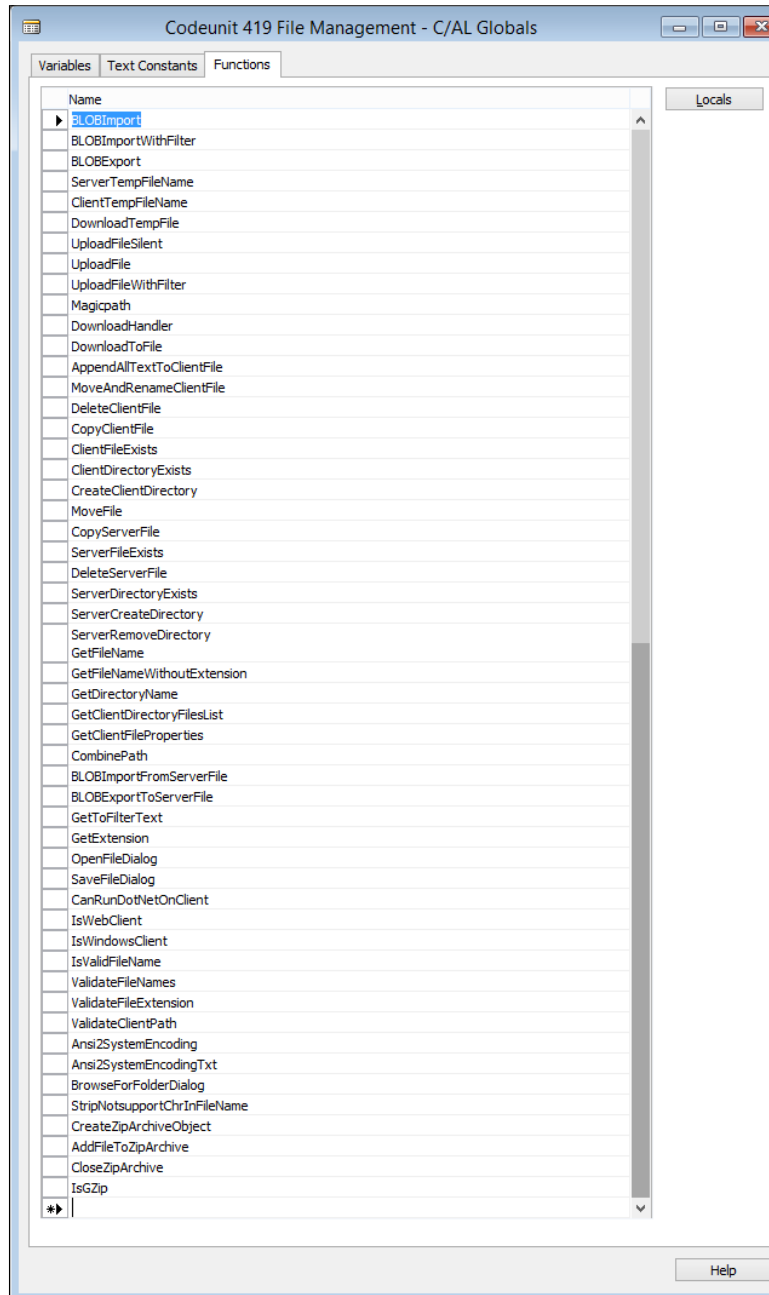
As you can see, there are functions that can import, export and edit layouts. Some of these functions reference **Codeunit 9651 Document Report Mgt.** This codeunit contains the following functions:



You can print a report or export it to PDF with these functions. The `ConvertToPdf` function uses a dot net (.NET) variable that references the assembly:

```
Microsoft.Dynamics.Nav.PdfWriter.WordToPdf.'Microsoft.Dynamics.  
Nav.PdfWriter, Version=8.0.0.0, Culture=neutral,  
PublicKeyToken=31bf3856ad364e35'
```

This codeunit then also references **Codeunit 419 File Management**. The file management codeunit contains functions to handle files, for example, how to send a file from the server to the client and vice versa, how to create an archive, and so on.



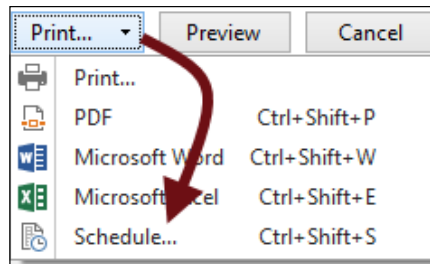
This codeunit contains functions that you can use to do almost anything with files.



It has been redesigned in Dynamics NAV 2015, and I believe it's important to know that it exists, so you don't reinvent the wheel when confronted with file management.

Scheduling reports

As of version 2015 of Dynamics NAV, you can schedule the execution of a report via the request page. When you run any report, you will see the following button:



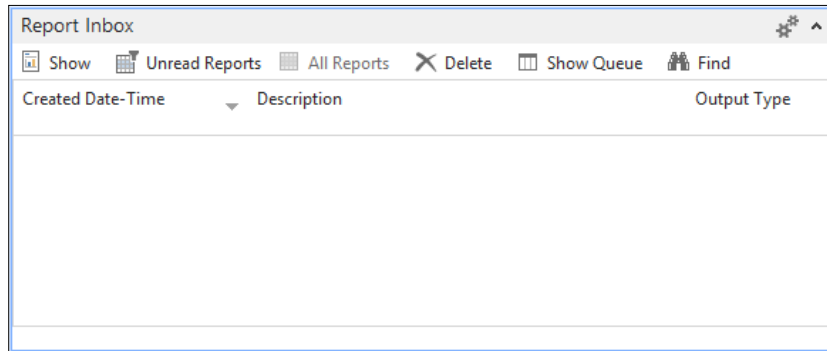
When you select **Schedule...**, you can determine when the report needs to be generated and in what format:

There is no running job queue for scheduled reports.

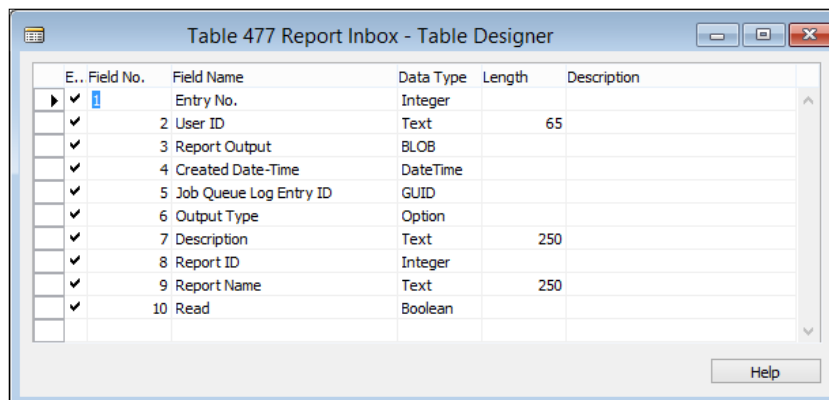
Report ID:	111 ▾
Report Name:	Customer - Top 10 List
Description:	Customer - Top 10 List
Report Output Type:	PDF ▾
Printer Name:	
Earliest Start Date/Time:	▾
Expiration Date/Time:	▾

OK Cancel

In order for this to work, you first need to set up the job queue. When the report is executed, via the schedule, it will then show up in the **Report Inbox** of the user that scheduled it. This inbox is usually on the role center page, as shown in the following screenshot:



The information is actually stored in the **Table 477 Report Inbox**:

A screenshot of a "Table Designer" window for "Table 477 Report Inbox". The window displays a table with the following columns: "E.. Field No.", "Field Name", "Data Type", "Length", and "Description". The table contains 10 rows of data.

E.. Field No.	Field Name	Data Type	Length	Description
	Entry No.	Integer		
✓ 2	User ID	Text	65	
✓ 3	Report Output	BLOB		
✓ 4	Created Date-Time	DateTime		
✓ 5	Job Queue Log Entry ID	GUID		
✓ 6	Output Type	Option		
✓ 7	Description	Text	250	
✓ 8	Report ID	Integer		
✓ 9	Report Name	Text	250	
✓ 10	Read	Boolean		

As you can see, the report is generated on a per user basis and the report inbox is filtered on the user ID.



Run and run modal

When a report is executed via code, you can use the `RUN` or `RUNMODAL` functions. Reports that are executed by the `RUNMODAL` function cannot be scheduled. You can only schedule reports that are executed by the `RUN` function.

Some of the document reports in the application use the report selection feature, and this uses the `RUNMODAL` function to run reports in sequence. That is why, when you print a sales invoice, the user will not see a schedule option on the request page. However, if you run the report directly from the object designer, then the schedule option will be visible.

Summary

In this chapter, I introduced you to the Word layout features of report design in Dynamics NAV. You have learned how to create and maintain a Word layout as an alternative to the more complex RDLC layouts.

In the next chapter, I will explain Power BI. I will demonstrate how you can harness the power of Excel, PowerPivot, Power View, and the other Power BI tools to build and share interactive and rich reports, starting from ODATA web services in Dynamics NAV.

9 Power BI

In this chapter I will introduce you to the world of Power BI. Dynamics NAV can publish pages and queries as ODATA web services so that external tools such as Microsoft Office Excel and Power BI can use them as data sources. I will provide an overview of the different tools available in the Power BI suite and how you can leverage their power using Dynamics NAV.

Dynamics NAV web services

You have the option to publish page, query, and codeunit objects as web services in Dynamics NAV when you navigate to the web services page.



Pages and queries are published as ODATA web services and pages and codeunits are published as SOAP web services.

What is a web service?

A web service, in terms of Dynamics NAV, or a SOAP web service, is a program that publishes business logic to the outside world and an ODATA web service can be considered as a data source so that the outside world can access your data in a uniform and self-described manner.

SOAP and ODATA are the protocols that are used for different types of web services.



More information about Dynamics NAV and web services is available here: [https://msdn.microsoft.com/en-us/library/dd355398\(v=nav.80\).aspx](https://msdn.microsoft.com/en-us/library/dd355398(v=nav.80).aspx)

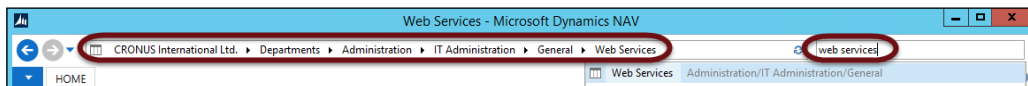
You publish Dynamics NAV business logic to the outside world with SOAP web services so that you can reuse it in other development environments.

For example, when you are developing an application using Visual Studio and .NET or a website using PHP, you can consult Dynamics NAV information via SOAP web servers and create, modify or delete it, using Dynamics NAV authentication, and business logic. Although very interesting, SOAP web services are not covered in this book.

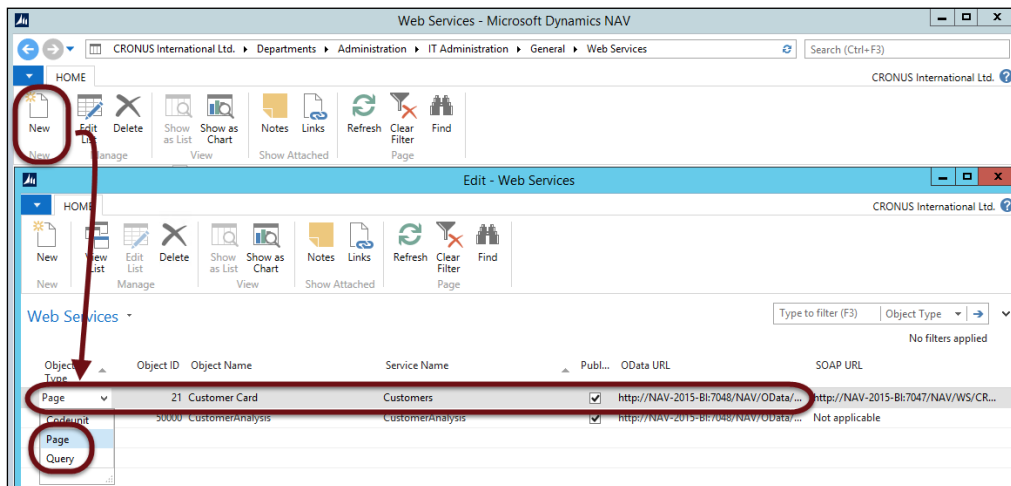
You publish page and query objects to the outside world with ODATA web services so that they can be accessed as data sources. Query web services are read-only, while you can read and also modify information with page ODATA web services. I will focus on the read-only ODATA page and query web services because the purpose of business intelligence is not to modify information from its data source.

So how do you publish a page or query as a web service in Dynamics NAV?

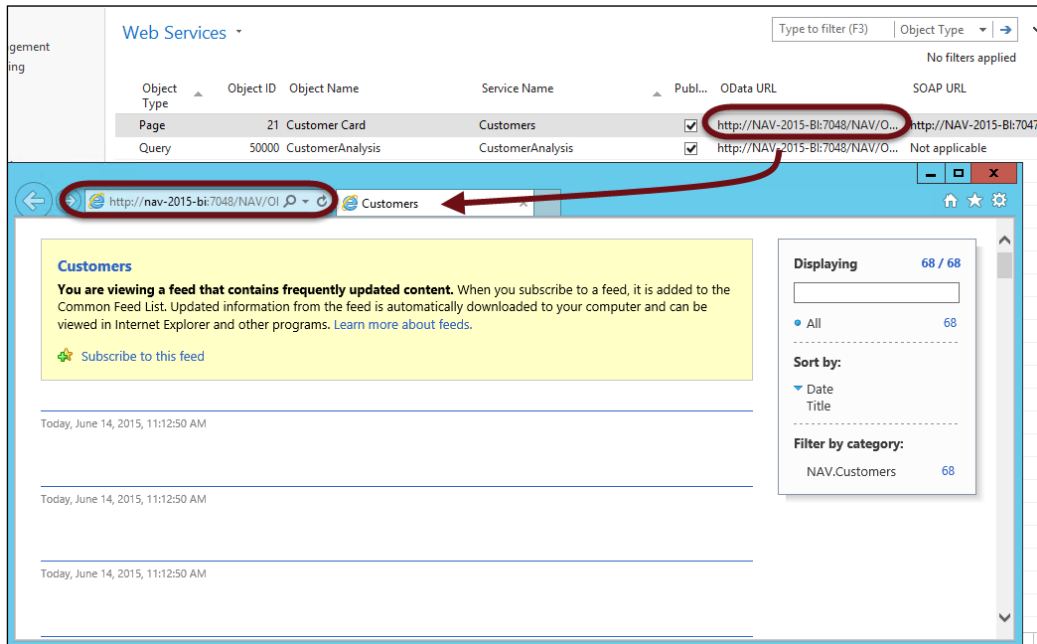
All you need to do is go to the web services page. You can type in web services in the application search box, or you can go here:



Then you use the **New** button to create a new web service, as shown in the following screenshot:



To publish it, simply select the **Publish** checkbox. Then, in the URL column, you can see where the object is available as an ODATA web service. To verify that it works, you can simply click on the URL, which opens in your browser:



In my example, the URL is `http://nav-2015-bi:7048/NAV/OData/Company('CRONUS%20International%20Ltd.']/Customers`. In your case this will be different, depending on your settings and the configuration of the service tier.

The generic URL is as follows:

`http://<Server>:<WebServicePort>/<ServerInstance>/OData`

This URL gives you an overview of all ODATA web services published by the instance of Dynamics NAV. To connect to a specific web service, after /OData, simply enter the name of the web service.

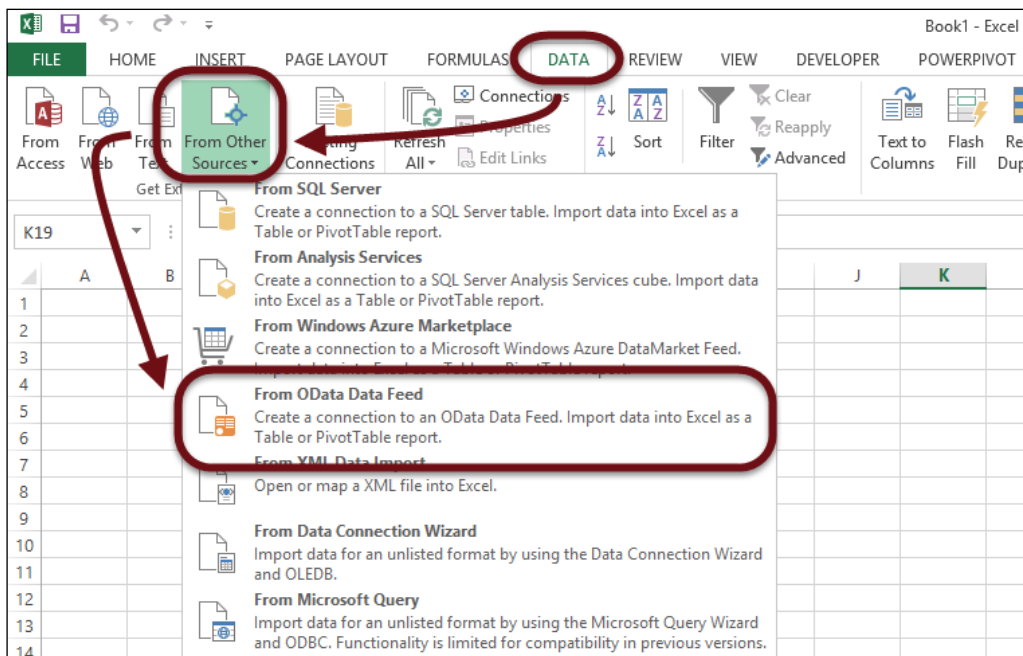
The browser displays the result of the ODATA web services, which is data from your page or query, but it renders it in a format that's not really interesting for a user. Let's open it in Excel and display it as a table.

Using Excel

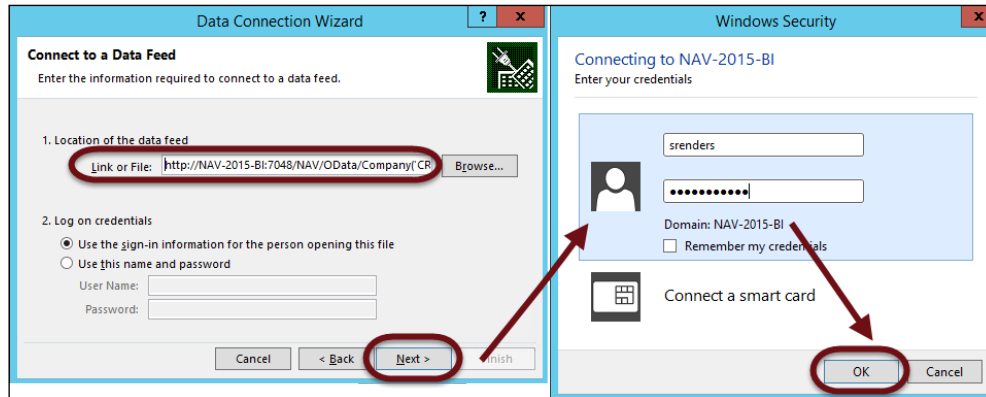
You can import data from different sources using Microsoft Office Excel, one of which, as from version 2010, is ODATA.

[ I'm using Excel version 2013 in the demonstrations and examples. Most of it is also possible in version 2010, but I recommend the 2013 version.]

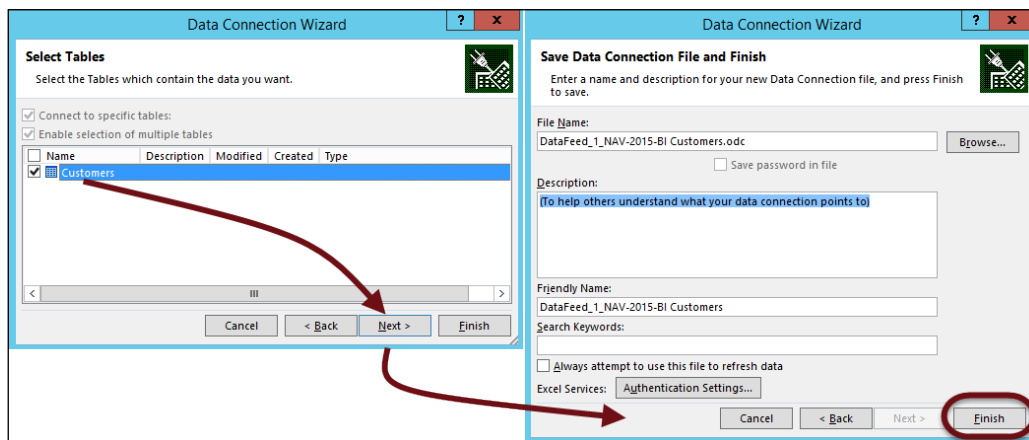
Start Microsoft Office Excel, and create a blank workbook. Then, in the ribbon, look for **DATA** and select **From Other Sources**, then select the **From OData Data Feed**:



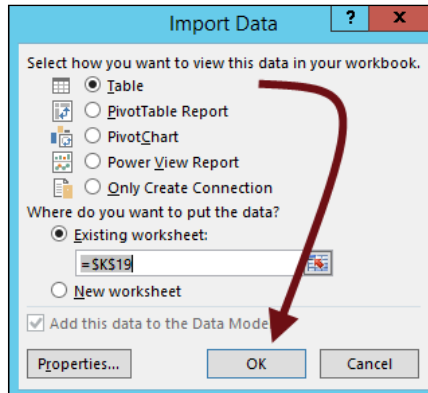
You need to enter the OData web service URL and the credentials used to access Dynamics NAV in the popup window that opens:



In the next window you will see the **Customer Card** web service. Select it and click on **Next**. Then you will be asked to save the connection so that Excel remembers it.

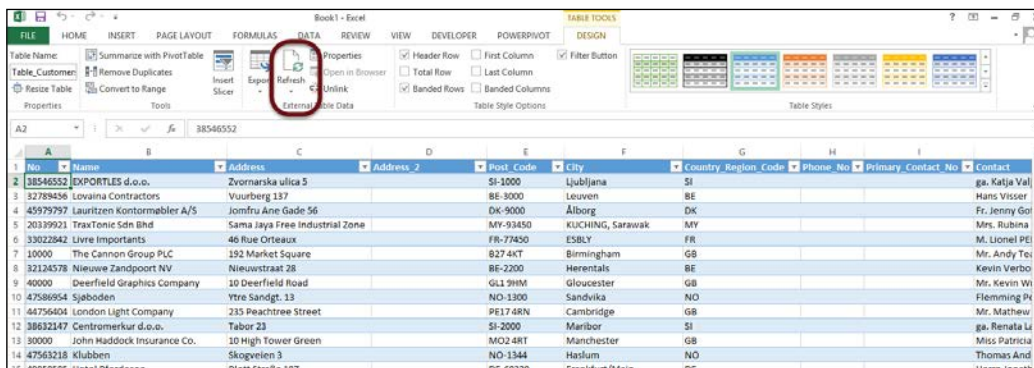


You can then decide if you want the data to be available in a table, a pivot table or a pivot chart:



If you have **Power View** enabled in Excel, the **Power View** option is also available. I will come back to this in the section about *Power View*.

In this example I selected the **Table** option and then clicked on **OK**. The result now looks like this:



The data from the web service is imported into Excel and displayed as a table. You can reload the data from Dynamics NAV with the **Refresh** button in the ribbon.

Now that you have your data in Excel, you can repeat this process to import data from other page or query objects and build your report in Excel.

Power Pivot

Power Pivot is an Excel plugin which allows you to create a data model. The data model can use several types of data sources, including OData web services, Windows Azure Marketplace, XML files, Excel files, databases, and so on.

I will introduce you to Power Pivot and demonstrate how you can use it to combine several Dynamics NAV OData web services and web services from Windows Azure Market Place into a data model. You can then use this data model to enrich your data and use it when you create reports using pivot tables, Power Pivot reports, Power View, and Power Map.

When you publish an object, such as a page or a query object, as an OData web service you can use it directly to create a report in Excel. But, if you want to create multiple reports then it's more interesting to create a data model in which you combine the different OData web services into a more complete and rich dataset that you can reuse.

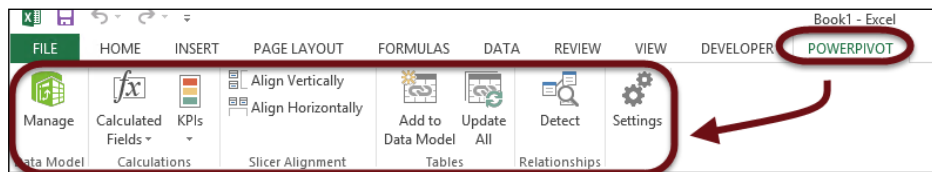
Power Pivot allows you to do this. Power Pivot has been developed specially as an Excel plugin in to perform calculations and manipulations on large datasets. Depending on your version of Excel, there are limitations on the number of rows it can use and Excel can become slow when it has to perform more complex calculations on your data. Power Pivot has a built-in scripting language and it's very fast with calculations, formatting and transformations on large datasets.

You can look on Power Pivot as a tool that helps you combine your Dynamics NAV OData web services with a related data model and you can enhance it with external data from Microsoft Azure Marketplace.

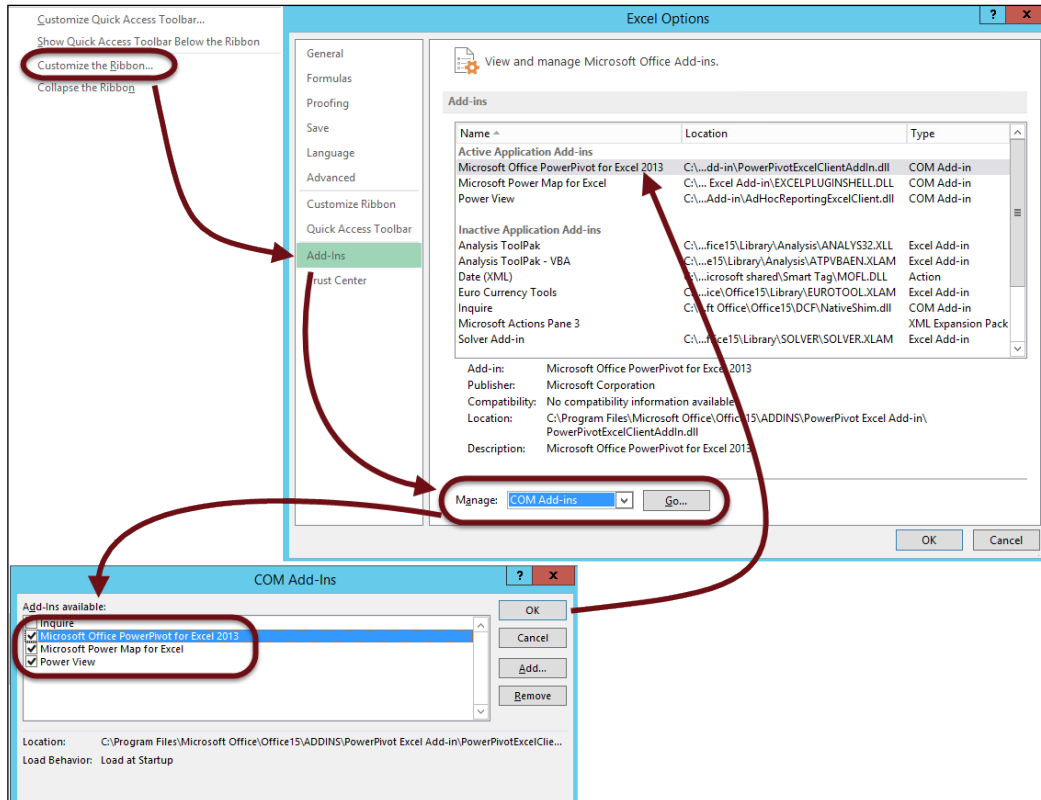
To get started in Power Pivot, you first need to activate it in Excel. Then We will publish several pages as web services and combine them in Power Pivot. Finally, we will add external data from Azure Marketplace.

Activating Power Pivot in Excel

In Excel, Power Pivot is available in the ribbon, as a separate tab, as shown here:



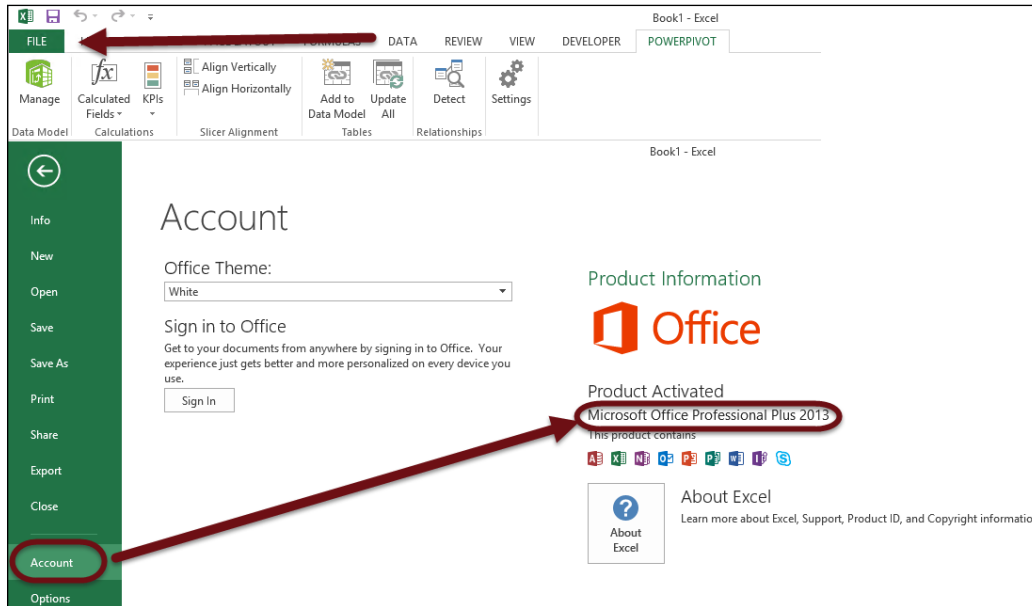
If this is not visible, use the **Customize the Ribbon...** feature to activate the Power Pivot plugin, as follows:



As you can see, the same process can be used to activate Power Map and Power View. If the options are not available in the list of available plugins, it means you are using a version of Excel that is not supported.

[ The minimum version required is Microsoft Office Professional Plus 2013. In Excel 2010, Power Pivot can be downloaded, for free, from the Microsoft website.]

To see which version of Excel you are using, open the application menu and go to **Account**:




Now that you know how to activate or enable Power Pivot, let's use it to create a data model.

Building a Power Pivot data model

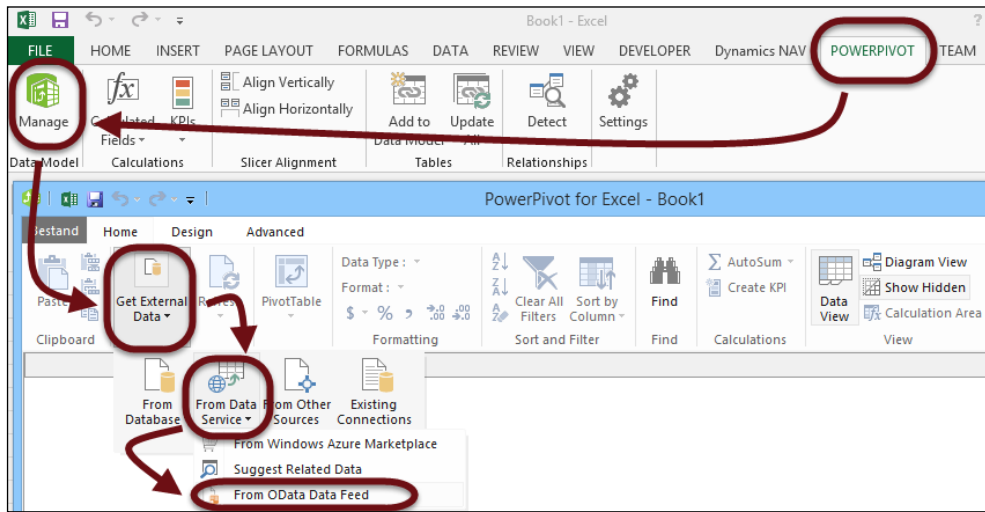
First, I will publish some page and query objects as OData web services in Dynamics NAV. The process is exactly the same as before, using the web services page in the application.

In this example, I have published the **Customer Card** page and a **Customer Analysis query**.

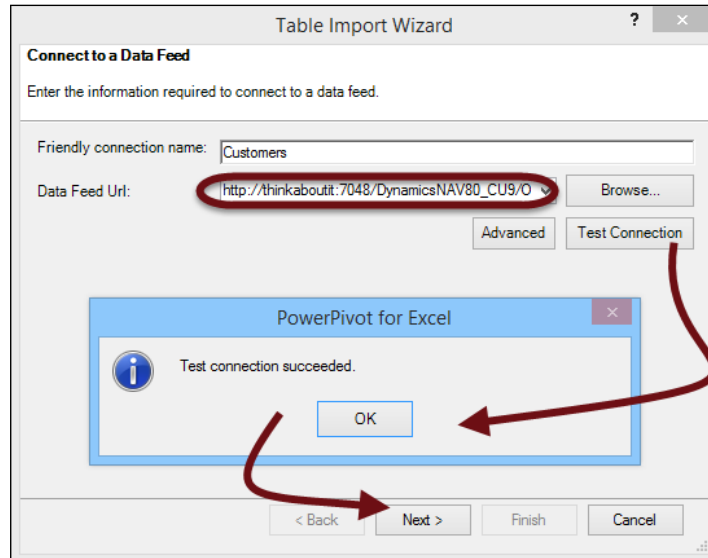
[ The **query Customer Analysis** is available in the object: **CustomerAnalysis**]

Importing data into Power Pivot

Create a new workbook in Excel, go to the **PowerPivot** tab on the ribbon, and click the **Manage** button. This opens up Power Pivot. Then, select **Get External Data, From Data Service, From OData Data Feed**:

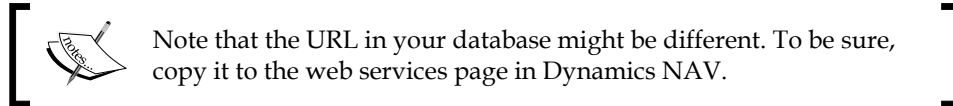


I enter `http://nav-2015-bi:7048/NAV/OData/Company('CRONUS%20International%20Ltd.')` in the **Data Feed Url**, then click on **Next** and **Finish**, as shown in the following screenshot:

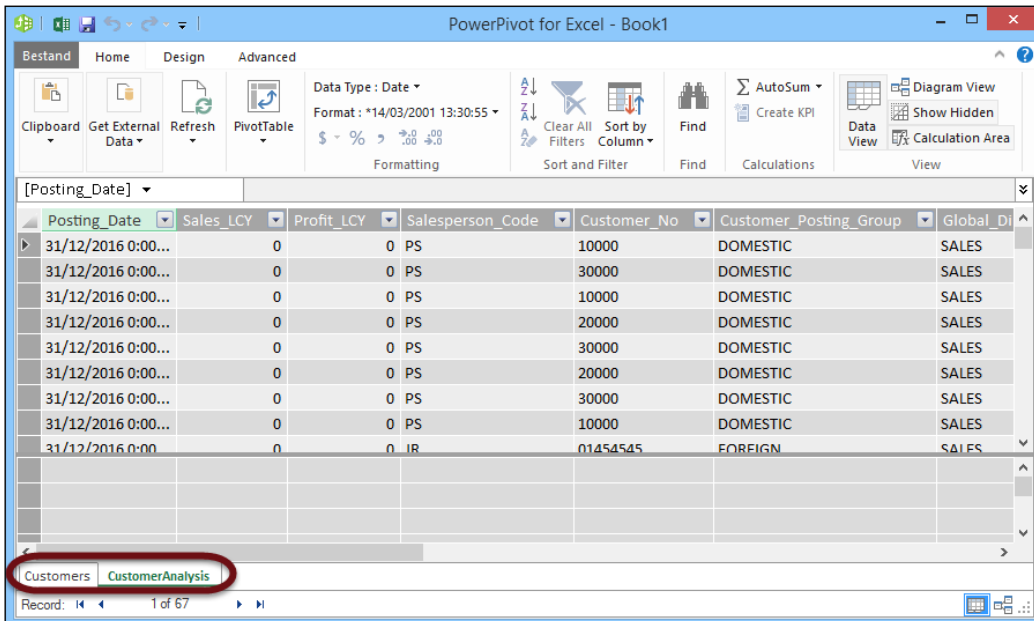


You can also give your connection a friendly name in the connection name field Friendly. In this example, I named it Customers, so I could refer back to it later.

I repeated the same steps to fetch the data feed for the Query object using the URL `http://NAV-2015-BI:7048/NAV/OData/Company('CRONUS%20International%20Ltd.']/CustomerAnalysis.`



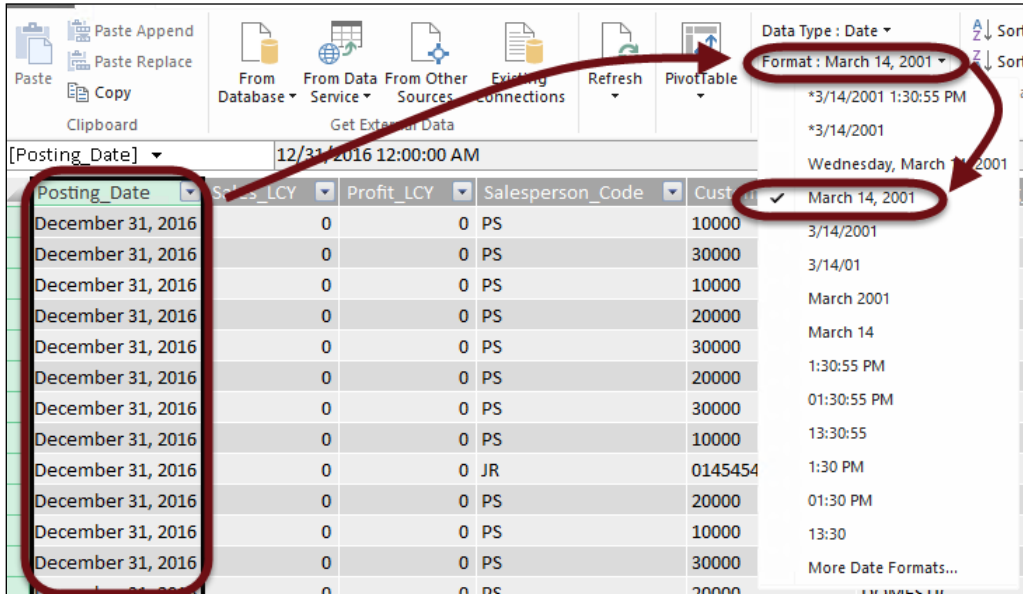
Now you have two data sources imported into Power Pivot, and the data sources show up as tabs just like in Excel, as you can see at the bottom of the window:




Posting_Date	Sales_LCY	Profit_LCY	Salesperson_Code	Customer_No	Customer_Posting_Group	Global_Di
31/12/2016 0:00...	0	0	PS	10000	DOMESTIC	SALES
31/12/2016 0:00...	0	0	PS	30000	DOMESTIC	SALES
31/12/2016 0:00...	0	0	PS	10000	DOMESTIC	SALES
31/12/2016 0:00...	0	0	PS	20000	DOMESTIC	SALES
31/12/2016 0:00...	0	0	PS	30000	DOMESTIC	SALES
31/12/2016 0:00...	0	0	PS	20000	DOMESTIC	SALES
31/12/2016 0:00...	0	0	PS	30000	DOMESTIC	SALES
31/12/2016 0:00...	0	0	PS	10000	DOMESTIC	SALES
31/12/2016 0:00...	0	0	IR	01454545	FOREIGN	SALES

Record: 1 of 67

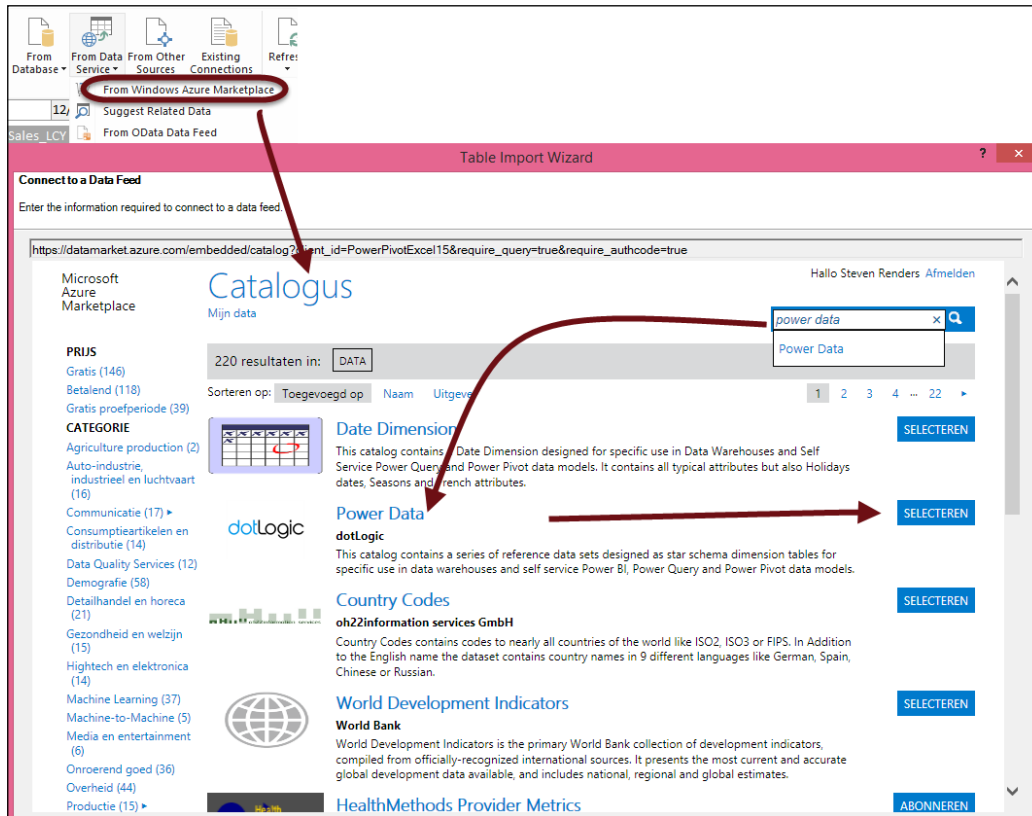
The data from the web services is loaded into Power Pivot and, with the buttons in the ribbon **Formatting**, and then **Sort** and then **Filter**, you can apply filters, formatting and sorting to the OData feeds:



I would like to present the customer analysis query information on a timeline, using the posting date. But, if I used the posting date as a time axis, it would not be linear because there are no postings on all of the dates. I will therefore import a date dimension from Azure Marketplace instead.

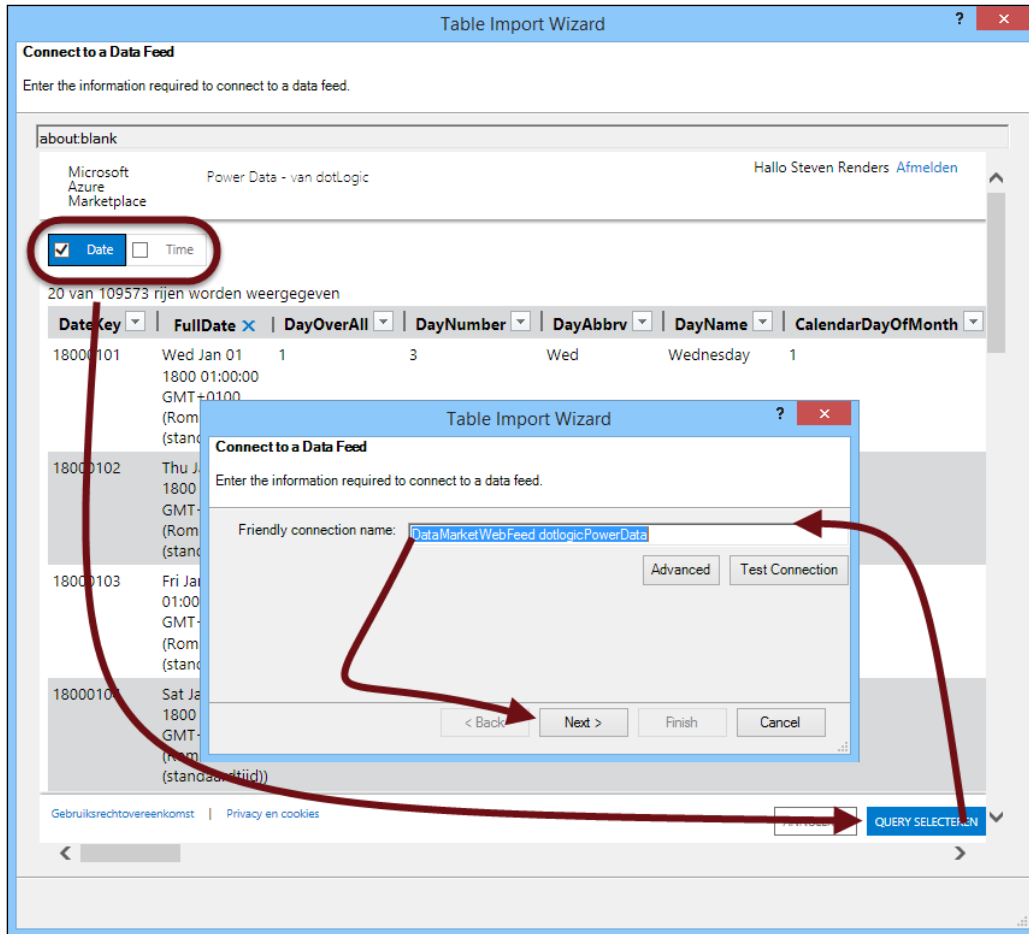
[ In order to connect to Windows Azure Marketplace you need an account. Go to <http://azure.microsoft.com/en-us/marketplace> to create a free account.]

Select **From Data Service**, and then select **From Windows Azure Marketplace**:

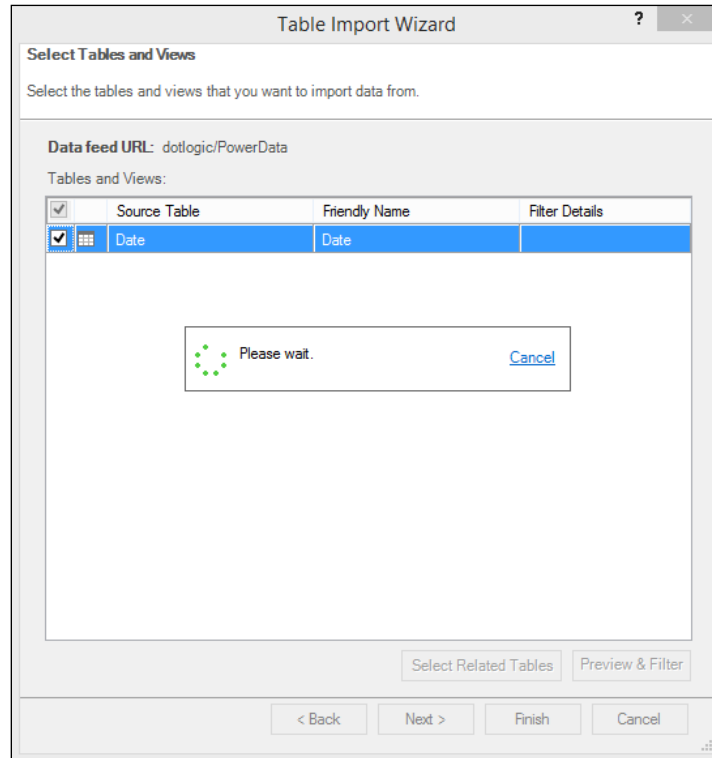


In the preceding window, on the left side I selected the **Free** category and in the search bar I typed `power data`. I then selected **Power Data** from the dropdown. This opens the next window, where you need to provide your live credentials. The service is free but you need to authenticate with a Windows live ID.

After you enter your ID, choose the **Select Query** option, and click **Next**, as shown in the following window:



Then click **Finish**. This will start to load the data from Azure Marketplace. It can take a few minutes, depending on the query that you are importing.

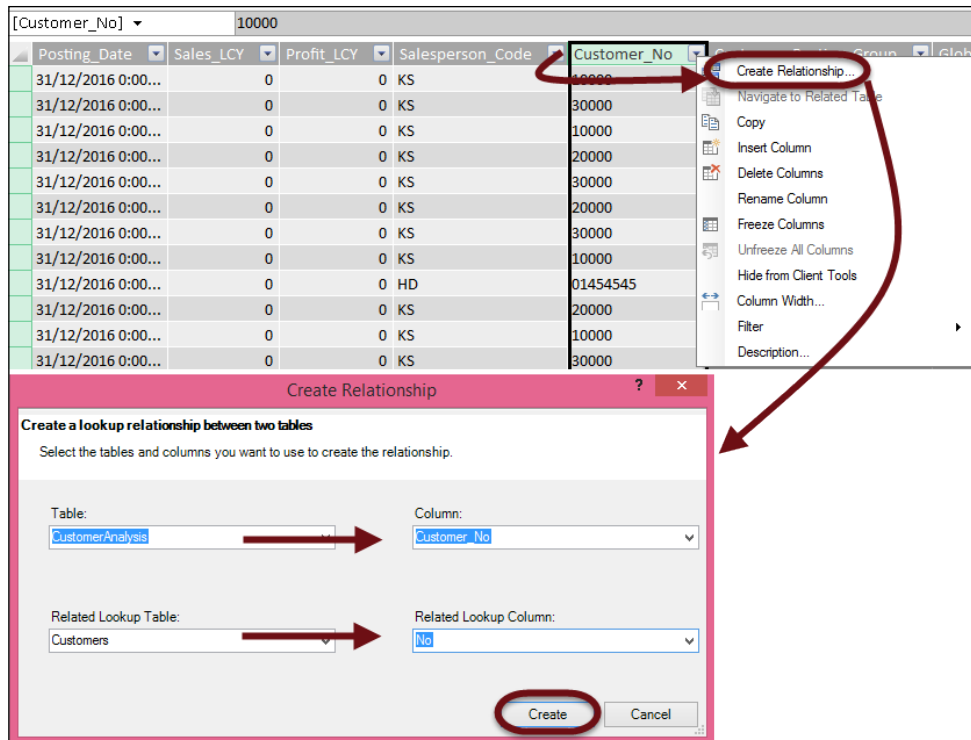


The date dimension contains a lot of information, it will import thousands of rows, that's why it takes some time to import. To speed up the data retrieval, you can deselect columns that you don't require, before starting the import process.

Creating relations in the Power Pivot data model

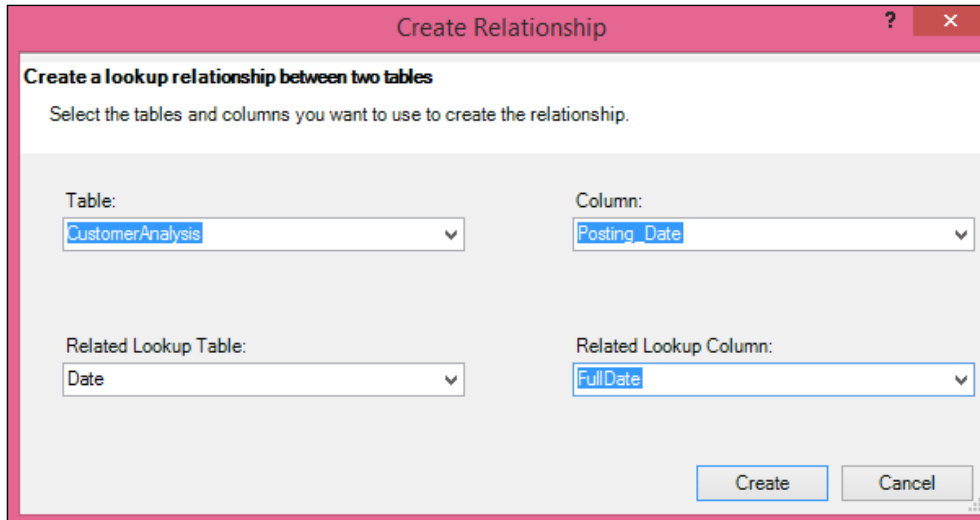
Now that we have imported our data from Azure Marketplace and from our two OData web services it's time to connect them by creating relations in our data model.

In the **CustomerAnalysis** tab, you can right-click the **Customer_No** column, and then select **Create Relationship...**:

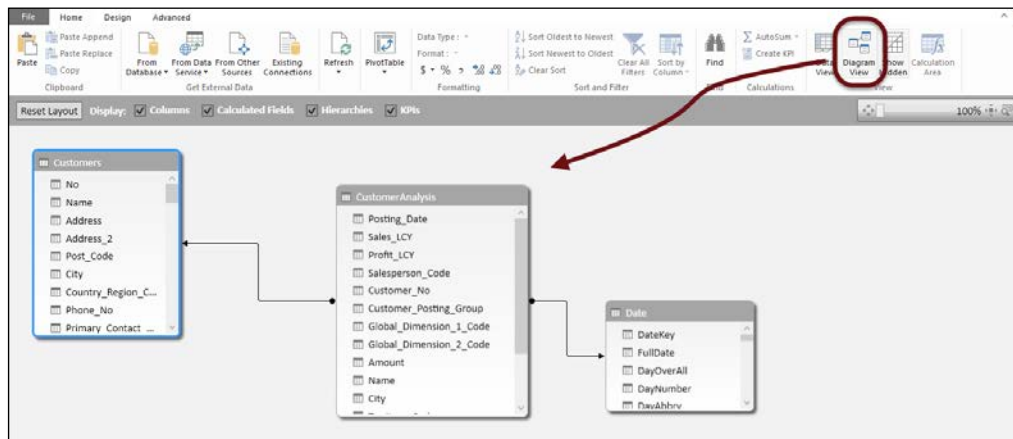


You can select the **Customer_No** field from **CustomerAnalysis** in the popup window to connect to the **No** of the **Customers** datasets and click **Create**.

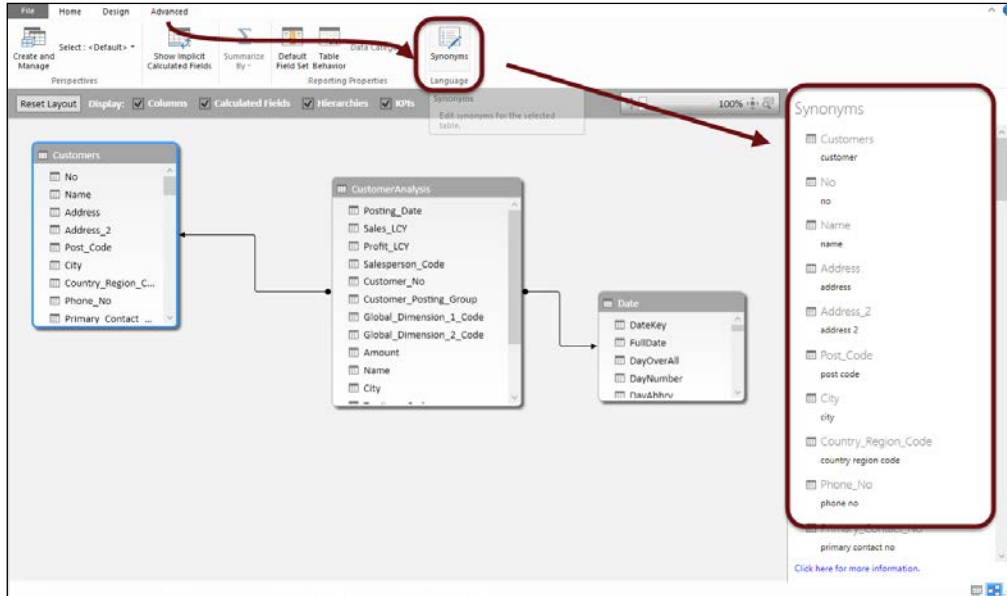
Then you repeat this process to connect the **Posting Date** field from the **CustomerAnalysis** dataset to the **FullDate** field from the **Date** dimension:




Then you can select the **Diagram View** button in the ribbon to visualize the relations you have just created:




If you want to go a step further, you can also provide synonyms for the table names and fields here:



This will help you when you upload your data model later in Office 365 with Power BI enabled. You can then use the Q&A feature, which uses synonyms to query your data model.

[ More information about how to *Add synonyms to a Power Pivot Excel data model* is available here:
<https://support.office.com/en-us/article/Add-synonyms-to-a-Power-Pivot-Excel-data-model-345f4f5b-5ec2-4998-bc46-a26bdc0810b6?CorrelationId=9f6ae99f-75e9-4b71-ae11-06b6d8f8c6b6&ui=en-US&rs=en-US&ad=US>]

You have now built your first, simple, data model in Power Pivot. Now it's time to use this model and create a dashboard report using Power View. Of course, you can also create a Pivot table in the same way we did at the beginning of the chapter instead of Power View.

[ Before you continue, I recommend closing Power Pivot and then saving your Excel workbook.]

Power View

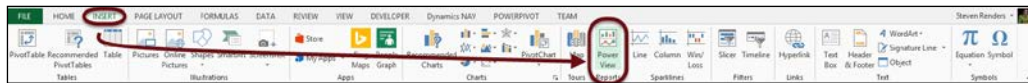
Power View is another Excel plugin which allows you to visualize information and create rich, stunning, interactive reports and a dashboard in Microsoft Office Excel. Power View was built by the SQL Server team, with the end user in mind. The technology is based on reporting services but the idea is that any user, without much technical knowledge, should be able to understand and use Power View intuitively to create rich, interactive and stunning reports, just like you always wanted.



A couple of years ago, I was watching a presentation on TED (www.ted.com) from Hans Rosling, who is the director of Gapminder in Sweden. He specializes in Global Health and statistics and presented information about the different countries in the world, mortality rates, income, and so on. The way that he presented the information, back in 2006, was mind-blowing. He was able to display information on bubble charts, including a timeline, in which he would click on play and you could see data evolving over the last twenty years. At that time, I was thinking, I hope one day we can do the same with our data in Dynamics NAV. When I took my first look at Power View I understood this was the tool that was going to make that possible.

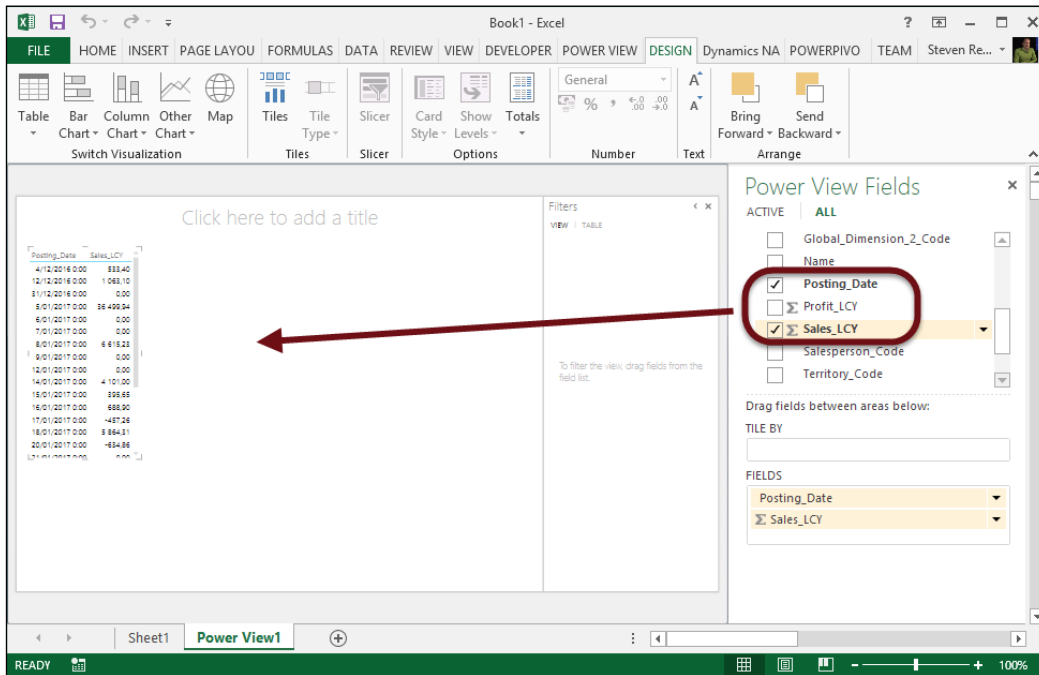
In other words, Power View will make your data come to life.

To get started, reopen the Excel workbook where you created the Power Pivot data model. Then select **Insert, Power View**:

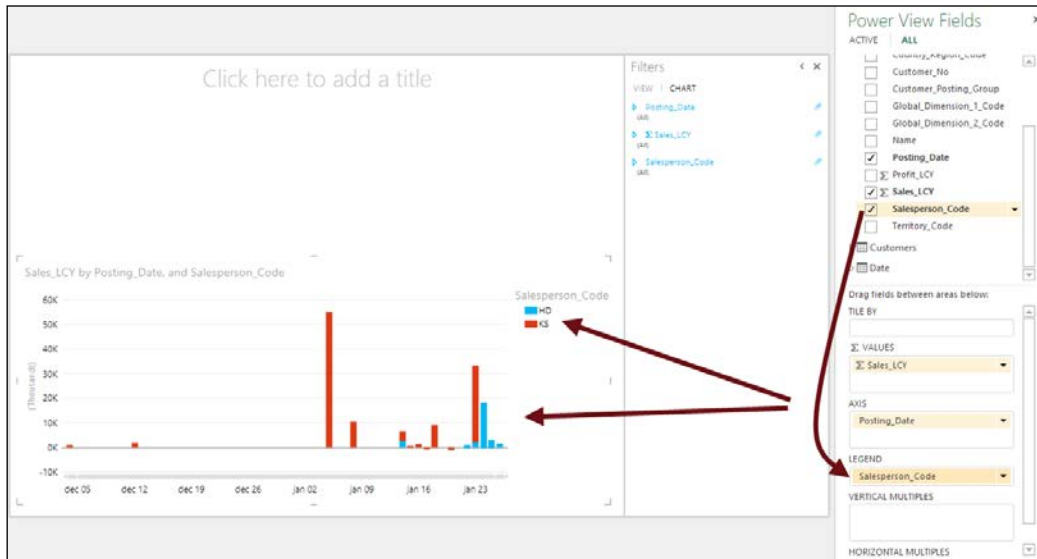


Power View requires Silverlight. If you don't have that installed, it will propose the installation. After installation you need to select the **enable** button and then restart Power View.

In the Power View window that opens, at the right side you will see **Power View Fields**. That contains the three tables from the Power Pivot data model you created earlier. When you open **CustomerAnalysis** and select the **Posting Date** and **Sales LCY** fields, they are shown by default in a table layout. Then, select **Column Chart**, **Stacked Column** in the ribbon:



This transforms the table containing numerical information into a chart which you can move around and resize. Then you can select the **Salesperson Code** field and drag it to the bottom of the **Legend** field:

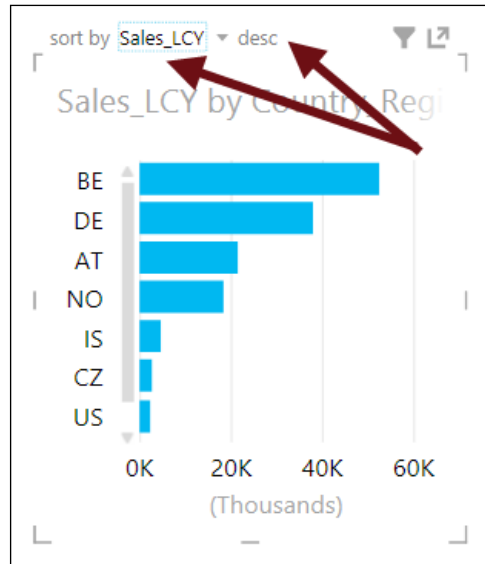


As you can see, this converts the chart into a real stacked chart where you can see the sales, on a timeline, by salesperson.

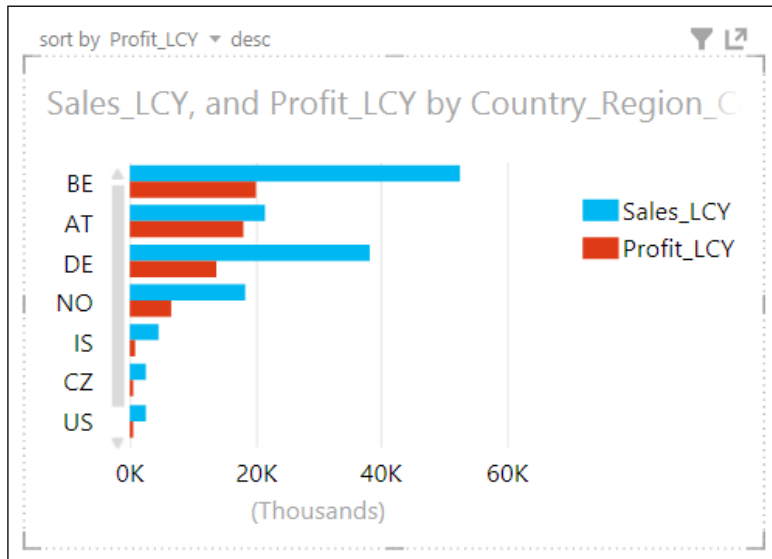
We are now going to add another chart to this report. To do this, first click somewhere in the report, outside the chart, on the white space. This makes sure nothing is selected. Then you select fields from the dataset to be added as a separate table, which you can then convert into another chart or visualization. If you don't click outside your chart to deselect it, and then select more fields from the dataset, then these fields are added to the current chart, instead of becoming a new chart.

Next, select **Sales_LCY** from the **CustomerAnalysis** dataset, and **Country_Region_Code** from the **City** dataset. In the ribbon, select **Bar Chart, Clustered Bar**. Then, select **Sort by Sales_LCY, desc** in the bar chart that is generated, on the top.

You can do this by simply clicking on the field that is shown until it becomes **Sales_LCY**:




You can also select the **Profit_LCY** field to make it a clustered bar:

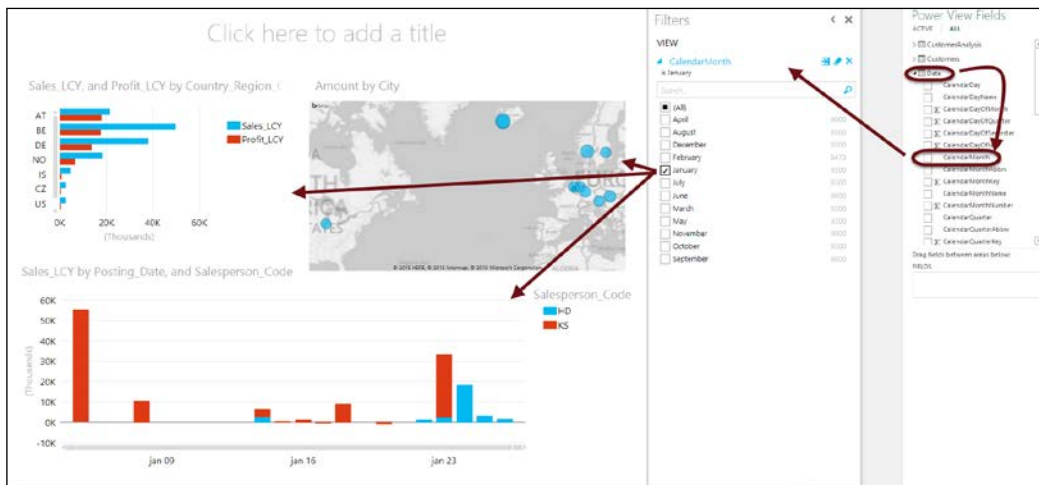


Next, select **City** and **Amount** from the **CustomerAnalysis** dataset. Then, select **Map** in the ribbon. This displays a map control. Resize it and then click the **Enable Content** button, which appears right below the ribbon. This enables the map and immediately loads the content:



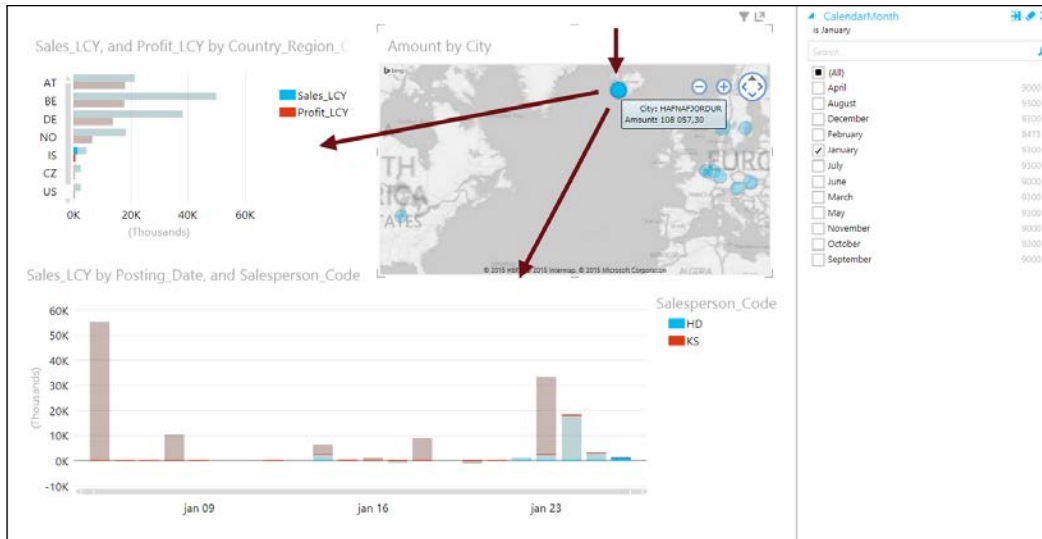
 You can drag the map around with your mouse and zoom in and out with the mouse wheel. When you hover over a circle, a tooltip is shown with the related data.

Open the **Date** dataset and drag **Calendarmonth** onto the **Filters** pane:



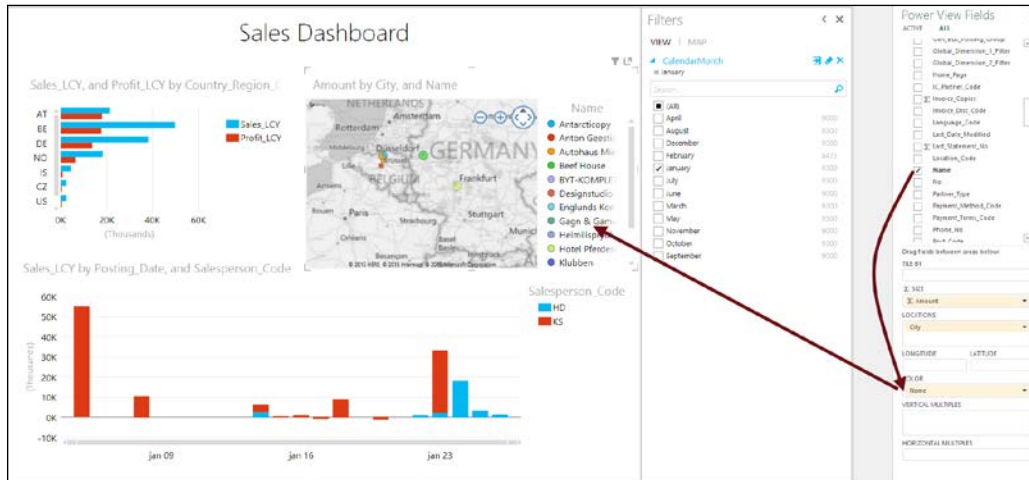
Now you can select one or more months to filter the dashboard. When you select a month, the three charts are updated immediately.

The dashboard is completely interactive. For example, when you click on a country on the map (Iceland), then some parts of the other charts are greyed out:



You can then see which parts originated from the country that you selected. The same is valid if you click on a piece of one of the other charts in the dashboard.

You can further enhance the charts, for example, when you click on the map, and then drag the customer's **Name** on the **Color** field, the map is updated accordingly:



You have now built your first interactive dashboard, in Power View. As you have seen, it's very easy to do. There are many other visualizations to choose from in the ribbon, so you can look at the information from the dataset in different ways. All of the different parts of the dashboard are connected, via the data model. The data model is using OData web services and, when you click the refresh button in the ribbon, live data is fetched from your Dynamics NAV database.

Another demonstration on creating a similar data model and dashboard is available here:




- *How Do I: Build a Power BI Dashboard in Office 365 with Microsoft Dynamics NAV 2015 Part I* (<https://www.youtube.com/watch?v=l-kEKkzjgUw&list=PL5B63EF419A3B59C8&index=96>)
- *How Do I: Build a Power BI Dashboard in Office 365 with Microsoft Dynamics NAV 2015 Part II* (<https://www.youtube.com/watch?v=tZYIIOxcc1o&list=PL5B63EF419A3B59C8&index=95>)

These are two of the many videos that I have created for Microsoft in the *How Do I* series for Dynamics NAV.

You have now visualized information, including sales, on a timeline. But the timeline cannot be played. In order to do that, you can use Power Map.

Power Map

you can present information on a geographical map in different ways with Power Map. Power Map is a little different to the other Power BI tools. Power Map is a tool that is meant as an alternative to PowerPoint, for example, when you need to present information. Instead of using a classic PowerPoint presentation, you can use Power Map to display the information on a map with animations, and analyze and visualize it from different angles. The result of a Power Map project is usually a video file.

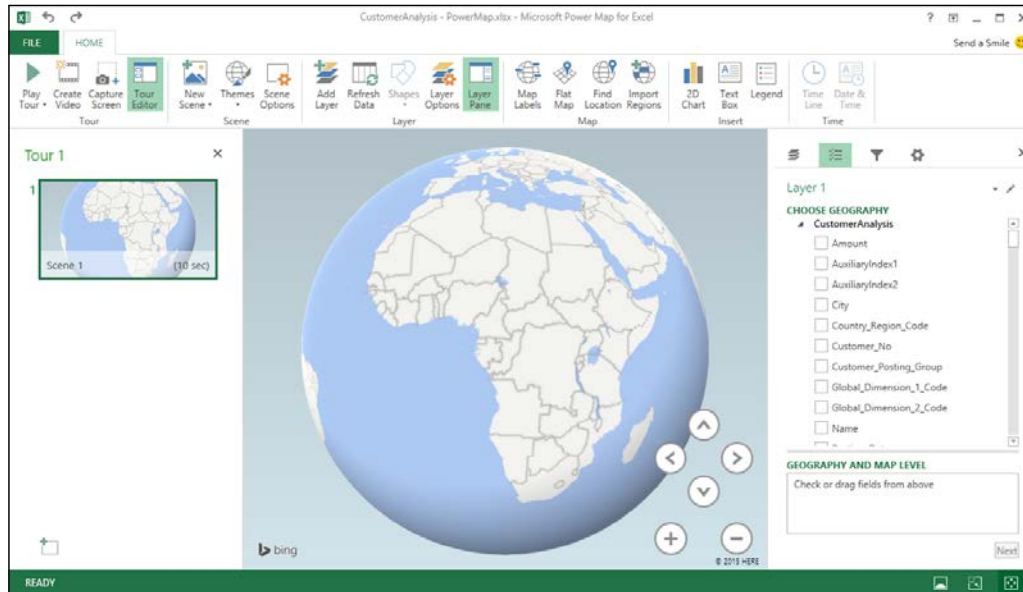
 Power Map, just like Power Pivot and Power View, is an Excel plugin. When you create a Power Map, you can base it on data you already have in Excel, or on different kinds of data sources. But, behind the scenes, Power Map uses Power Pivot to manage its data, so a data model in Power Pivot is, in my opinion, the best way to get started.

A Power Map consists of tours, scenes and layers. A tour contains one or more scenes, and a scene contains one or more layers. A scene is similar to a slide in a PowerPoint presentation, and it displays a map. A layer is the actual geographical mapping presented with a chosen visualization. If your data model contains a date field, then you can link it in your layer to create a timeline, as you would create animations in a PowerPoint slide.

Let's get started. Select **Insert, Map, Launch Power Map** in Excel, in the ribbon:

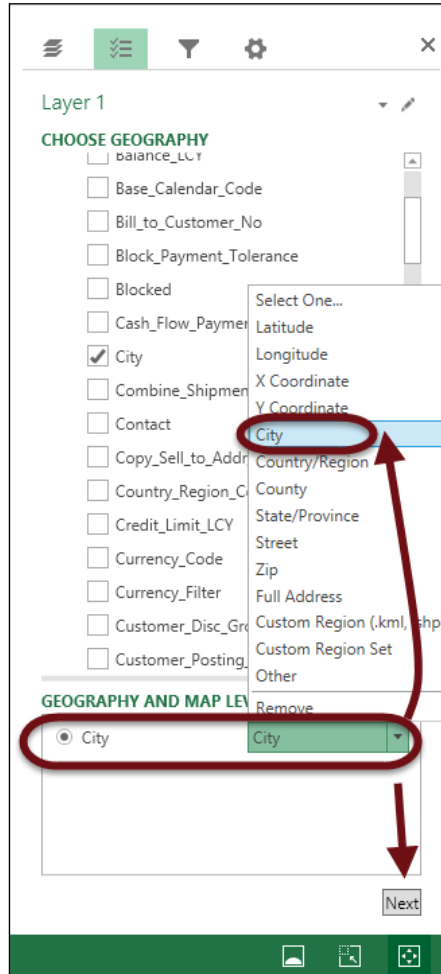


Power Map opens, and displays a map of the world. You can see the data model on the right side:



The first thing to do is choose a geography. That means a field that can be linked to coordinates on the map. Choose a geography where you have to select a field from the dataset at the right side, in **Layer 1**. I will use the **City** field from the customer data source.

You can do this at the bottom of the geography pane, as follows:



Once selected, the data point loads immediately on the map. Then select **Next**. The window updates and you can see the success rate of cities at the top of the pane, or the mapping confidence. When you click on the percentage, you can see the details:

The screenshot shows a software interface with a layer pane on the left and a 'Mapping Confidence' dialog box on the right. The layer pane shows 'GEOGRAPHY' and 'Map by City (City)' with a link to '81%'. The 'Mapping Confidence' dialog box contains the following text and table:

We plotted 81% of locations on Layer 1 with high confidence. Below is the list of locations we weren't sure about.

City	Result
AGDAL-RABAT	⚠ Gare Rabat Agdal, Morocco
ÅLBORG	⚠ Ålborg, Denmark
AMSTERDAM	⚠ Amsterdam, Netherlands
ASKER	⚠ Asker, Norway
ATLANTA	⚠ Atlanta, GA
BARCELONA	⚠ Barcelona, Catalonia, Spain
BLOEMFONTEIN	⚠ Mangaung, South Africa
BRAS	⚠ Brás, Brazil
BRUGGE	⚠ Bruges, Belgium
CAMBRIDGE	⚠ Cambridge, MA
CARLETONVILLE	⚠ Carletonville, South Africa
CASABLANCA	⚠ White House, DC
CHICAGO	⚠ Chicago, IL
DUDLEY	⚠ Dudley, United Kingdom

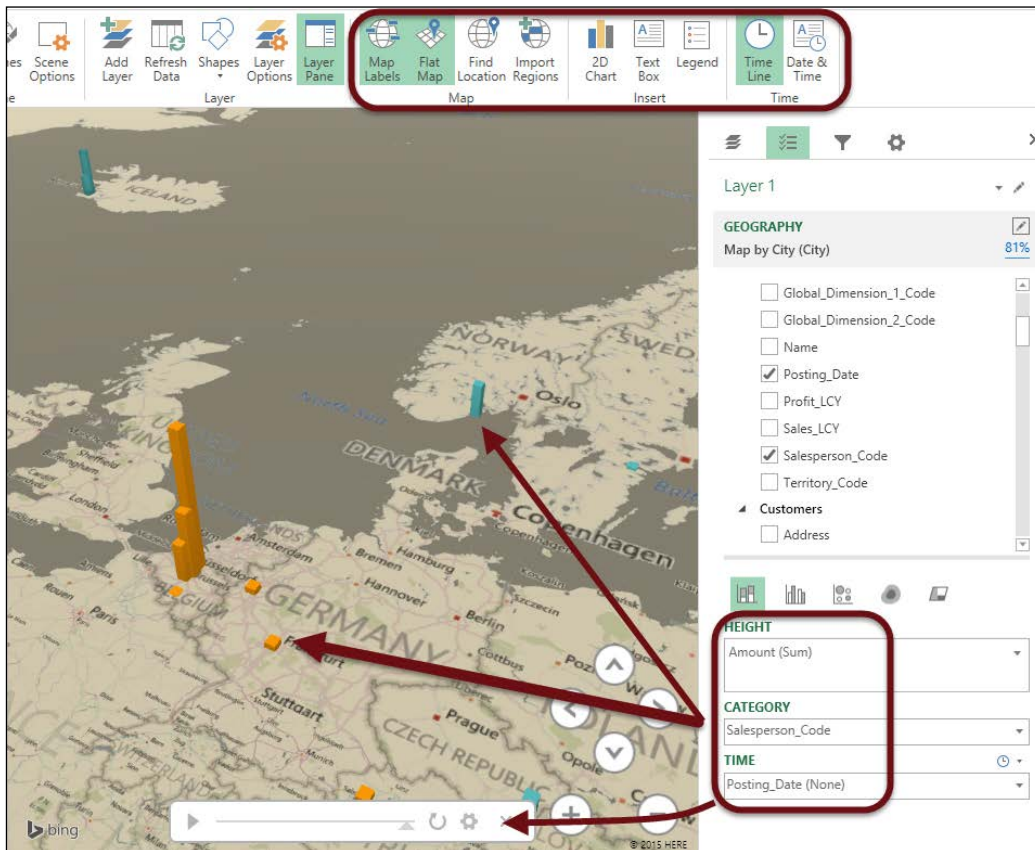
Include more geography columns in your data to improve accuracy.

Page 1 of 1

OK

Power Map will perform an educated guess even when the name of the city on Dynamics NAV does not match exactly. For the cities that do not match, you can try to update the data model.

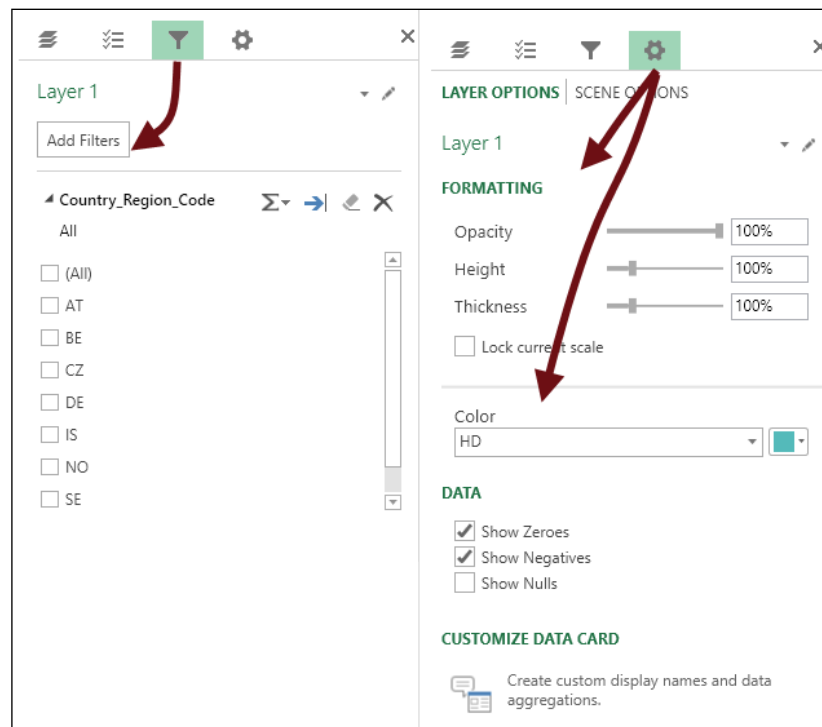
Select the **Amount** field in the data model. It will be used as the height of the bars on the map. Then, drag the salesperson to the category box so that different salespeople display different colors. Select the **Posting Date** field and drag it to the **Time** box. This will make a timeline appear. When you click on it, the data will play, and you can see, over time, the bars growing.



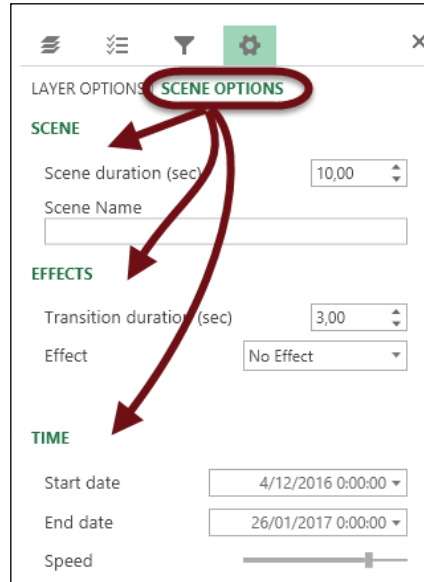
You can choose themes in the **Power Map** ribbon to get different colours. You can change a 3D into a 2D map, display map labels, show a legend, and more, by using the following buttons:



You can change the bars into clustered bars, bubbles, heat indicators or regions. You can add filters and define more layer options with the buttons on top of the layer pane:

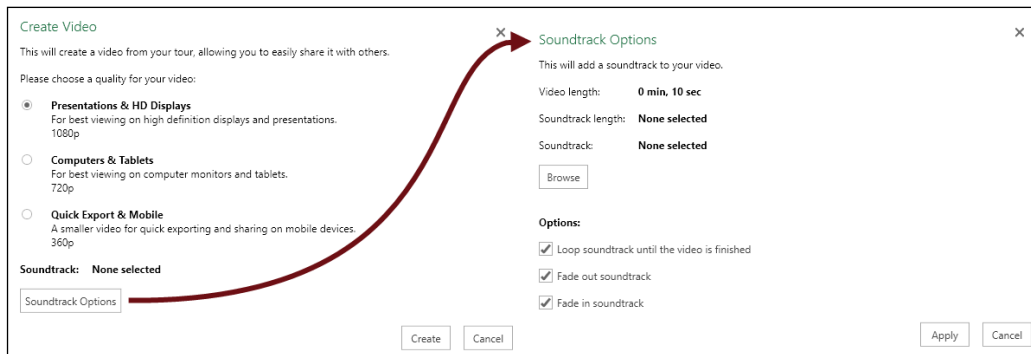


If you click on **SCENE OPTIONS**, you can define the settings for the video effects:



You are still working in **Tour1, Layer 1, Scene 1**. You can add more layers and tours if you want. For the purpose of this example I will keep it simple and produce my video. To do that, I will use the **Create video** button in the ribbon.

This will open the following window:



You can now select the quality settings, import a sound track and then render the video. You require a video card to do this.



Another demonstration of how to create a similar dashboard is available here:

How Do I: Use Power Map to Visualize Data in Microsoft Dynamics NAV 2015 at https://www.youtube.com/watch?v=ZcyG8Y_xY0

Power Query

Power Query is a tool that you can use to create and manage different data sources. It has been able to do this since the release of Power BI Designer and I recommend Power BI Designer over Power Query.

Power BI Designer

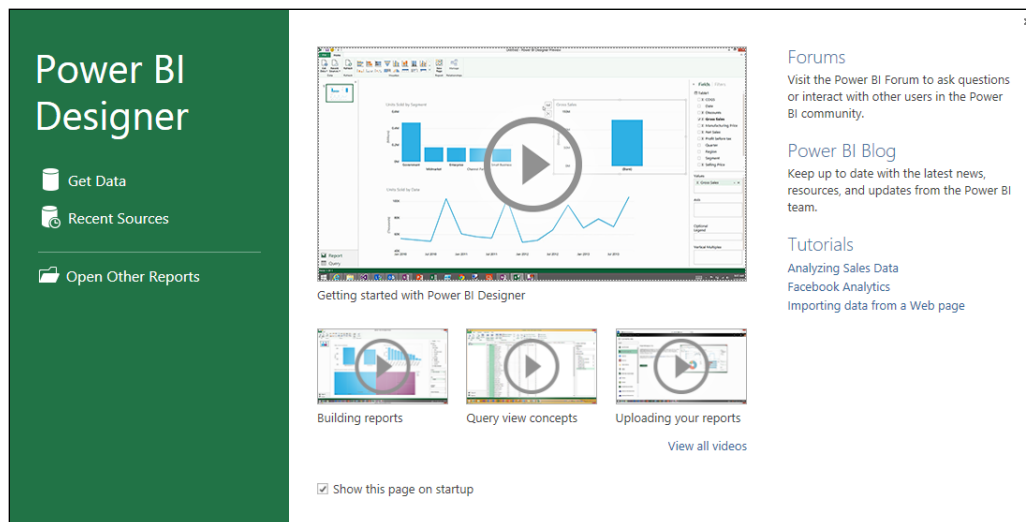
Power BI Designer is a relatively new tool. It incorporates Power Pivot, Power View, Power Query, and Power Map. You can download it for free (at the time of writing), and it allows you to create and manage your Power BI dashboards, even when you don't have Excel installed on your machine.



First, you need to download and install Power BI Designer, and you can get it here:

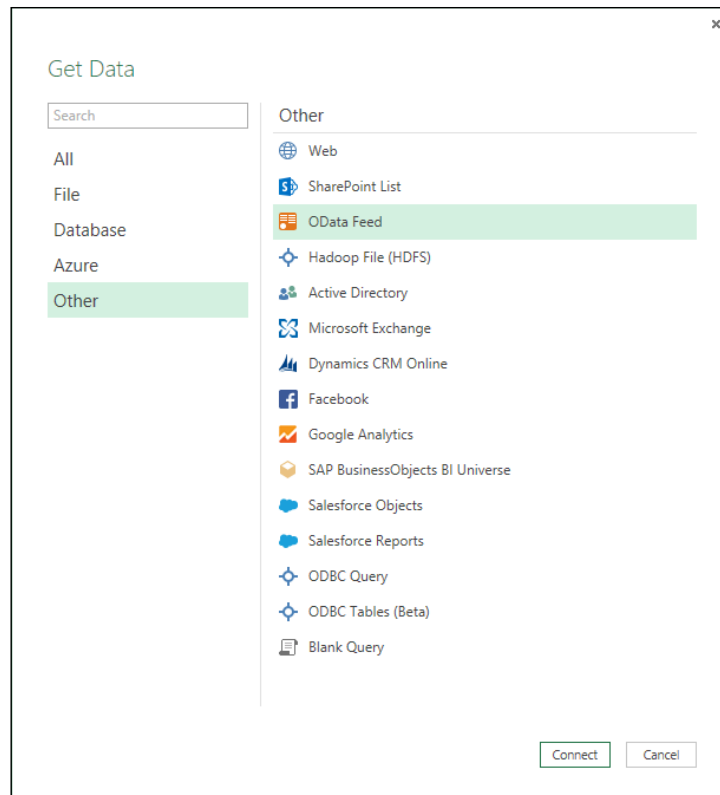
<https://powerbi.microsoft.com/designer>

When you launch it, you are presented with the following splash screen:

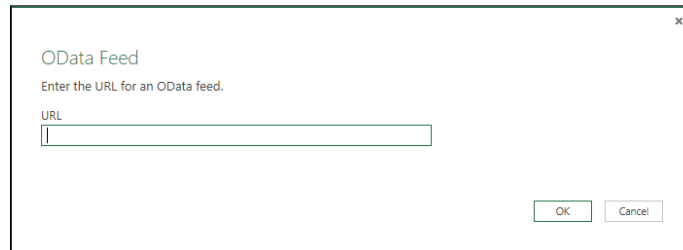


If you are new to Power BI Designer, I recommend having a look at the videos. They provide a good introduction to what you can do with the designer.

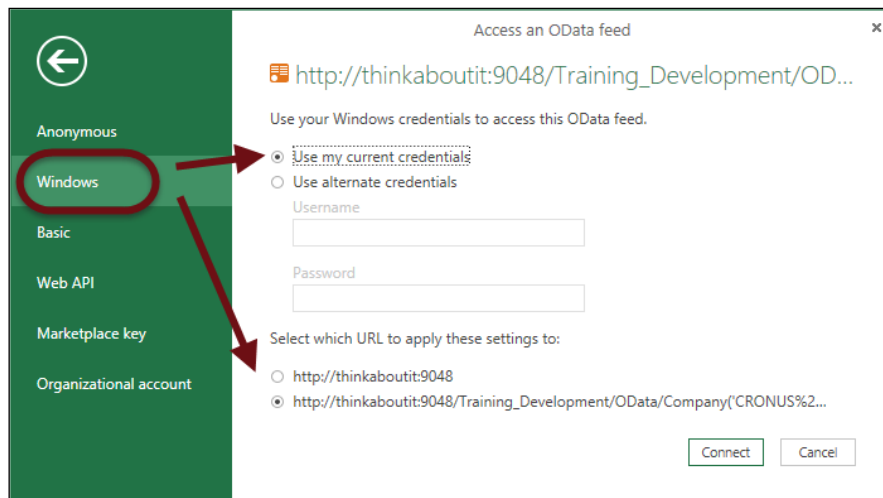
The first thing I will do is determine which data source or data sources I will use. To do that, select **Get Data**.



As you can see, there are many different data sources to choose from. I will use Dynamics NAV OData web services. When you select **Connect**, you have to type or paste the URL of the OData source in the next window:



You might get an error in the next window saying that you are not authorized, depending on the URL that you have entered. This is because Power BI Designer tries to connect without authenticating by default. To fix this problem, select **Windows** and then the user that has access to Dynamics NAV:



Once connected to the web service, the designer can open and display the fields on the right side:

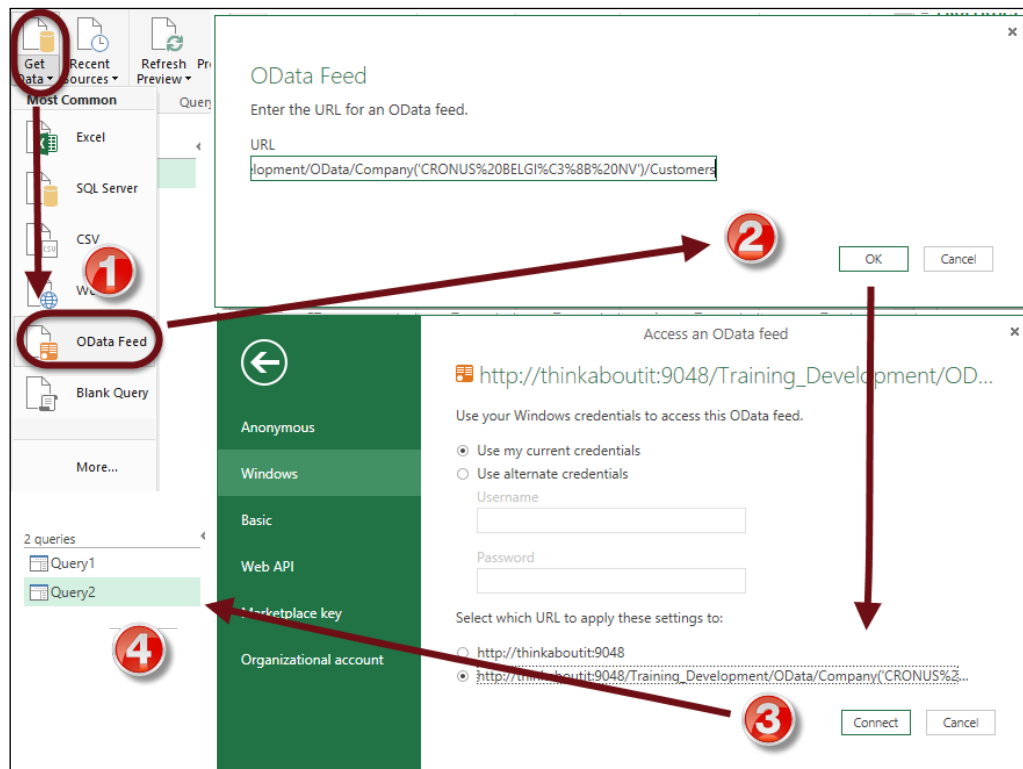
The screenshot displays the Power BI Designer interface. At the top left, a 'Load' dialog box is open, showing 'Query1' (32.5 KB from thinkaboutit) with a red circle '1' and a 'Cancel' button. To the right, the 'Fields' pane lists various columns from 'Query1', with a red circle '2' next to the list. The main workspace shows a data table with columns: Posting_Date, Sales_LCY, Profit_LCY, Salesperson_Code, Customer_No, and Customer_Posting_Group. The table contains 21 rows of data. A red arrow points from the 'Query1' entry in the table to the 'Name' field in the 'Query Settings' pane, which is circled in red. A red circle '3' is placed over the 'Query' button in the bottom left corner.

	Posting_Date	Sales_LCY	Profit_LCY	Salesperson_Code	Customer_No	Customer_Posting_Group
1	31/12/2016 0:00:00	0	0	KS	10000	BIN
2	31/12/2016 0:00:00	0	0	KS	30000	BIN
3	31/12/2016 0:00:00	0	0	KS	10000	BIN
4	31/12/2016 0:00:00	0	0	KS	20000	BIN
5	31/12/2016 0:00:00	0	0	KS	30000	BIN
6	31/12/2016 0:00:00	0	0	KS	20000	BIN
7	31/12/2016 0:00:00	0	0	KS	30000	BIN
8	31/12/2016 0:00:00	0	0	KS	10000	BIN
9	31/12/2016 0:00:00	0	0	HD	01454545	BUJ
10	31/12/2016 0:00:00	0	0	KS	20000	BIN
11	31/12/2016 0:00:00	0	0	KS	10000	BIN
12	31/12/2016 0:00:00	0	0	KS	30000	BIN
13	31/12/2016 0:00:00	0	0	KS	20000	BIN
14	31/12/2016 0:00:00	0	0	KS	10000	BIN
15	31/12/2016 0:00:00	0	0	KS	30000	BIN
16	31/12/2016 0:00:00	0	0	KS	20000	BIN
17	31/12/2016 0:00:00	0	0	KS	10000	BIN
18	5/01/2017 0:00:00	1549,01	638,43	KS	49525252	EU
19	5/01/2017 0:00:00	7745,02	3192,11	KS	49525252	EU
20	5/01/2017 0:00:00	5266,62	1624,29	KS	49858585	EU
21	5/01/2017 0:00:00	5266,62	1624,29	KS	49858585	EU
22	5/01/2017 0:00:00	10533,23	3248,58	KS	49858585	EU
23	5/01/2017 0:00:00	10533,23	3248,58	KS	49858585	EU

Select **Query** on the left hand side, at the bottom of the screen. This opens the query, where you can rename it and, if required, you can transform, sort, and format your data.

Every action that you perform on your data here is recorded as a step in the applied steps window. You can then see which steps you applied and in what order. The next time the designer connects to the data source it will repeat those steps.

Now I will get the other web service, as follows:

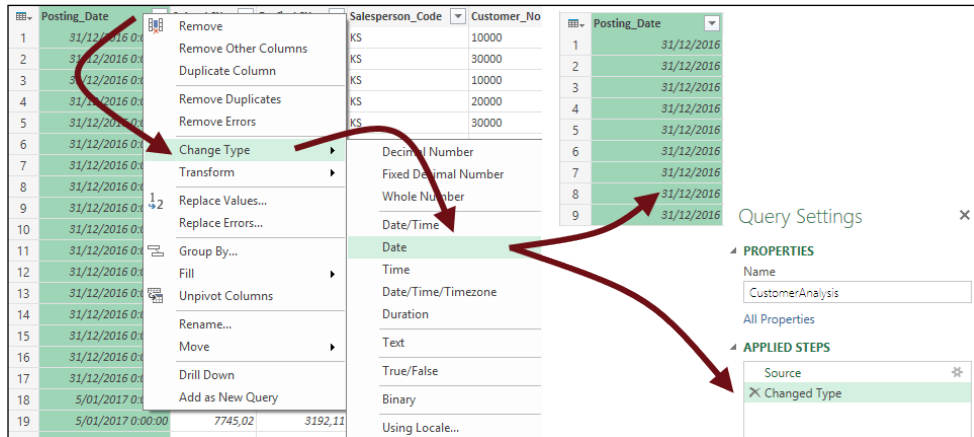


I have connected my two OData web services as follows:

- `http://thinkaboutit:9048/Training_Development/OData/Company('CRONUS%20BELGI%C3%8B%20NV')/CustomerAnalysis`
- `http://thinkaboutit:9048/Training_Development/OData/Company('CRONUS%20BELGI%C3%8B%20NV')/Customers`

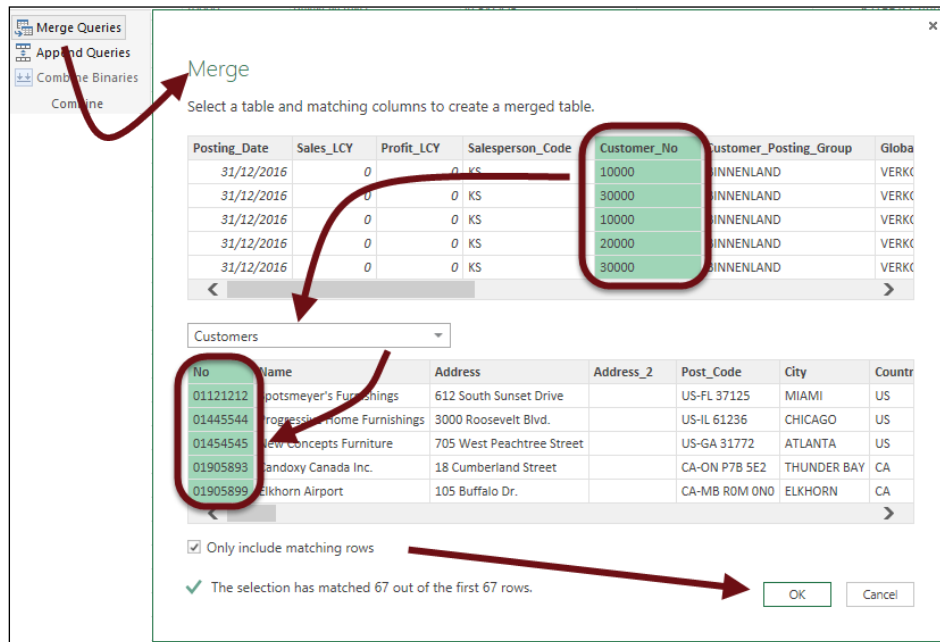
I have renamed Query1 as CustomerAnalysis and Query2 as Customers. I will then apply some transformations and connect the two.

First I right-click on the **Posting Date** column in the **CustomerAnalysis** query. Then I select **Change Type, Date**:

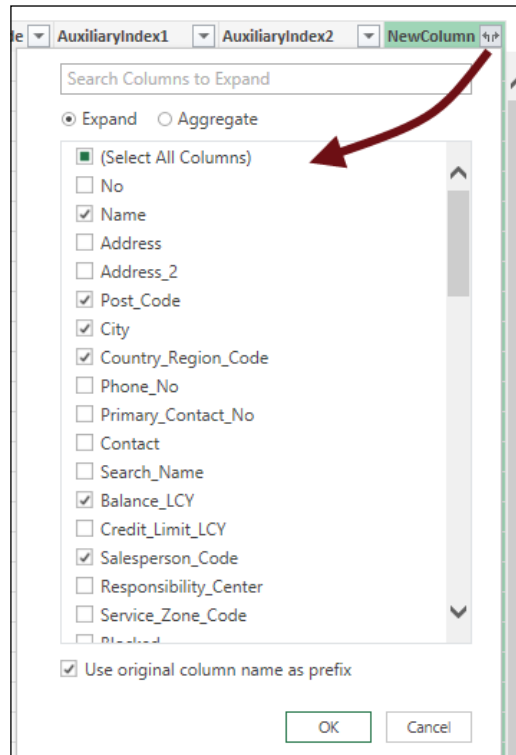


You should see that the format of the data has changed, and the applied steps now contain a **Changed Type** entry. You can undo this by selecting the **X** left to it in the applied steps column.

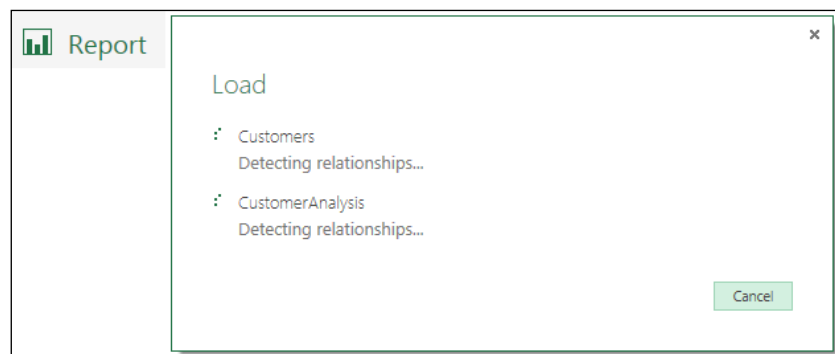
Now I will merge the two queries. You start by selecting **Merge Queries** in the ribbon of the **CustomerAnalysis** query:



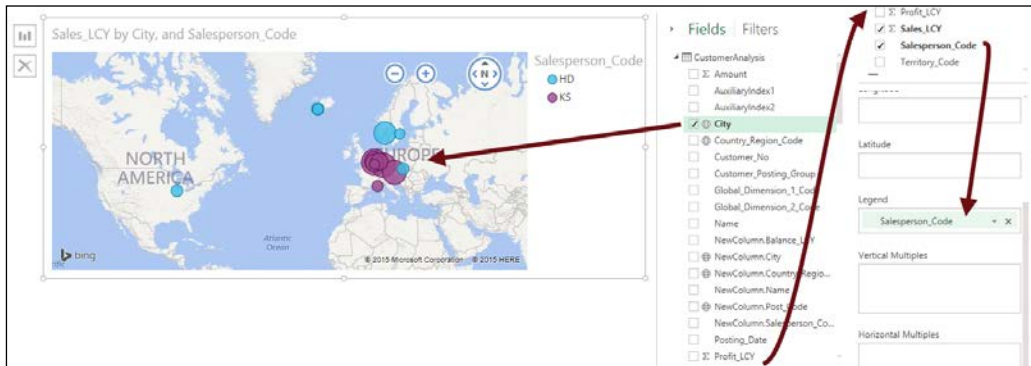
When you click on **OK**, you will see that a new column has been added to the right. You can select the icon and which columns from the **Customer** query you want to include:



Now I'm ready to create a report based on the **CustomerAnalysis** query. To do that, select **Report** at the left bottom:

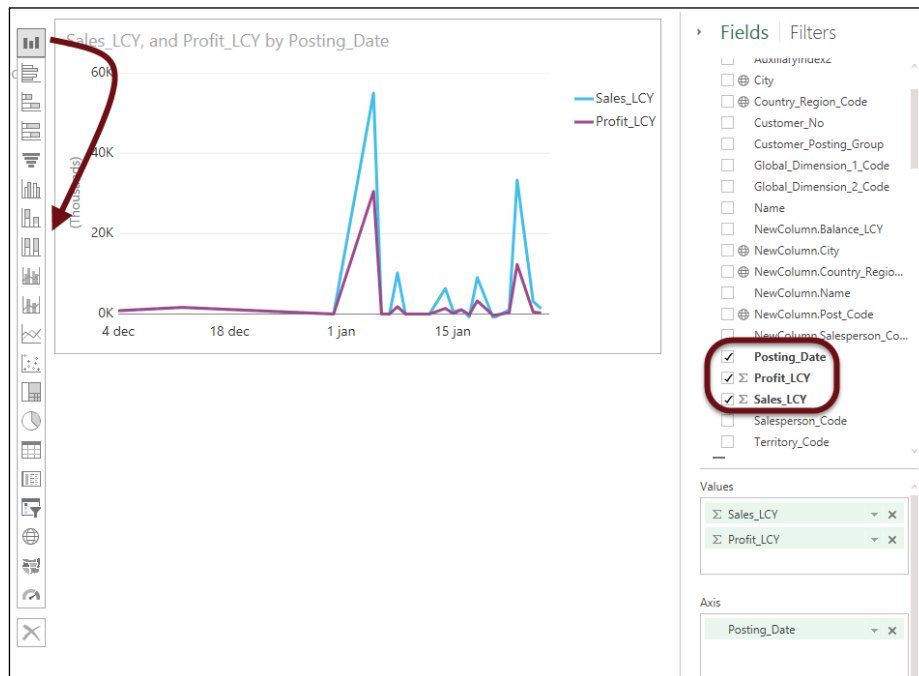


I select the **City** field in the dataset on the right. You can see that it has a small globe icon next to it. This is because it has been recognized and, when you select it, it's displayed using a map:



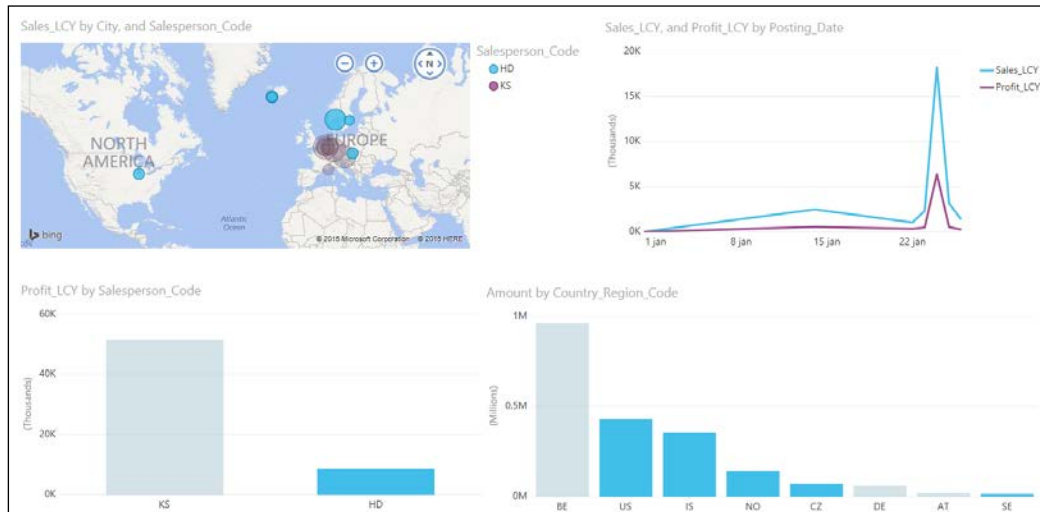
Then I drag **Salesperson_Code** to the **Legend** field, and the map is updated with different columns for different salespersons. By default, **Sales_LCY** was selected as the measure to display on the map.

Then, I select **Posting Date** in the list of fields, and a graph chart is automatically added. Next, I select the **Profit_LCY** field, which is also added to it:



There's a button that you can click at the left top. It allows you to change the current visualization to another.

Then I add two more charts:



They are **Profit_LCY by Salesperson_Code** and **Amount by Country_Region_Code**, as you can see in the preceding screenshot.

The charts are interactive as in Power View. When you select a **Salesperson**, the other charts update.

Next I save the dashboard, via **File, Save as**.

You have now created your first dashboard using Power BI Designer. Next I will use my Office 365 account, in which I have added a Power BI subscription, and I will upload the dashboard we have created with Power View and Power BI Designer.

PowerBI.com

Imagine that you work in a company and the CFO, or someone in management, asks you the following questions. Could you create me a report that shows:

- The sales and profit by customer and salesperson
- The top ten cities or countries, by sales
- Current year sales, on a timeline

Then they mention the fact that the report should use data from the live Dynamics NAV database, it should be able to refresh on the fly, and it should be easy to analyze the data and, if possible make it flexible so that they can create more views and analysis on the same data.

Well, with PowerBI.com, you will be able to do that, very quickly and without much technical knowledge.

If you go to www.powerbi.com, you can subscribe, for free, to the preview version of PowerBI.com:

Power BI Mobile Solutions Pricing Support Downloads [Sign In](#)

Microsoft Power BI Support

Get help and support for Power BI

Get started with Power BI Preview

— Getting Started

Microsoft Power BI helps you stay up to date with the information that matters to you. With Power BI, **dashboards** help you keep a finger on the pulse of your business. Your dashboards display **tiles** that you can click to explore further with **reports**. Connect to multiple **datasets** to bring all of the relevant data together in one place.

Need help understanding the building blocks that make up Power BI Preview? See [Power BI Preview - Basic Concepts](#).

Note: This article is about the new Power BI Preview experience, rather than the current experience, Power BI for Office 365. Read more about [which Power BI experience is right for me](#).

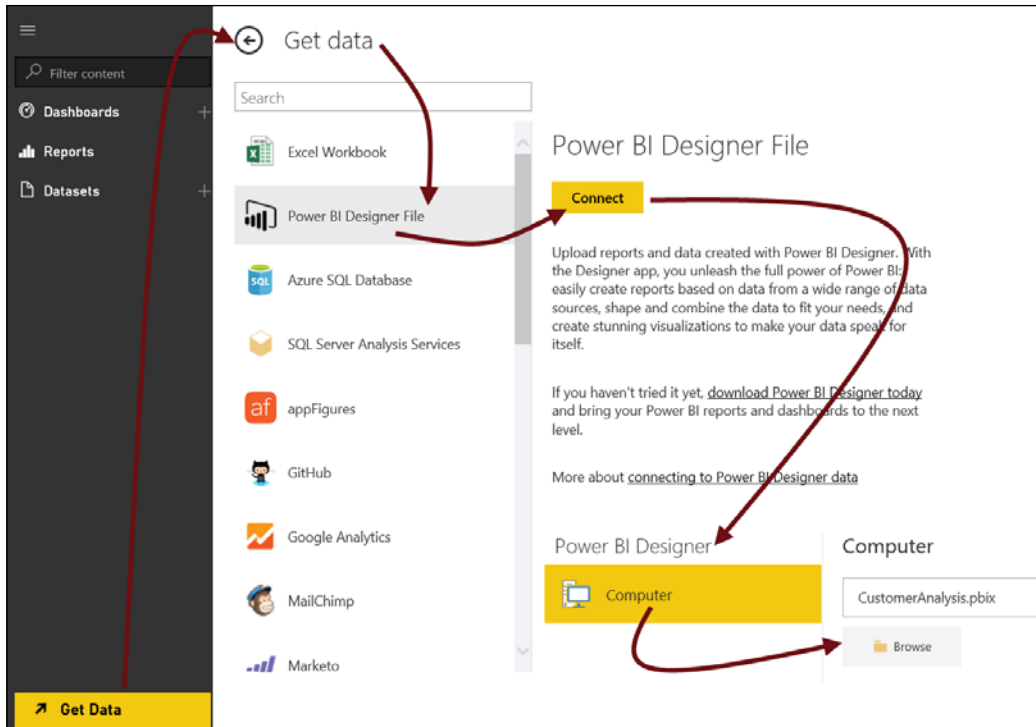
Sign up for Power BI

Get started using Power BI today and begin creating insightful, interactive dashboards within minutes.

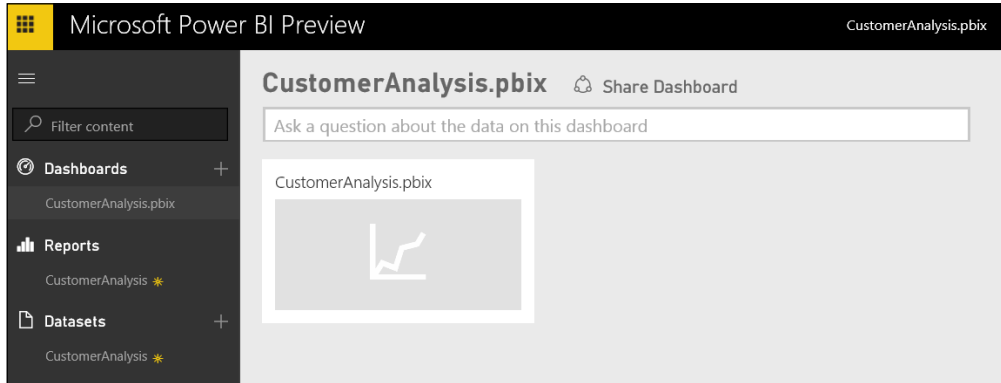
[Sign up](#)

New and returning users may sign in to Power BI Support

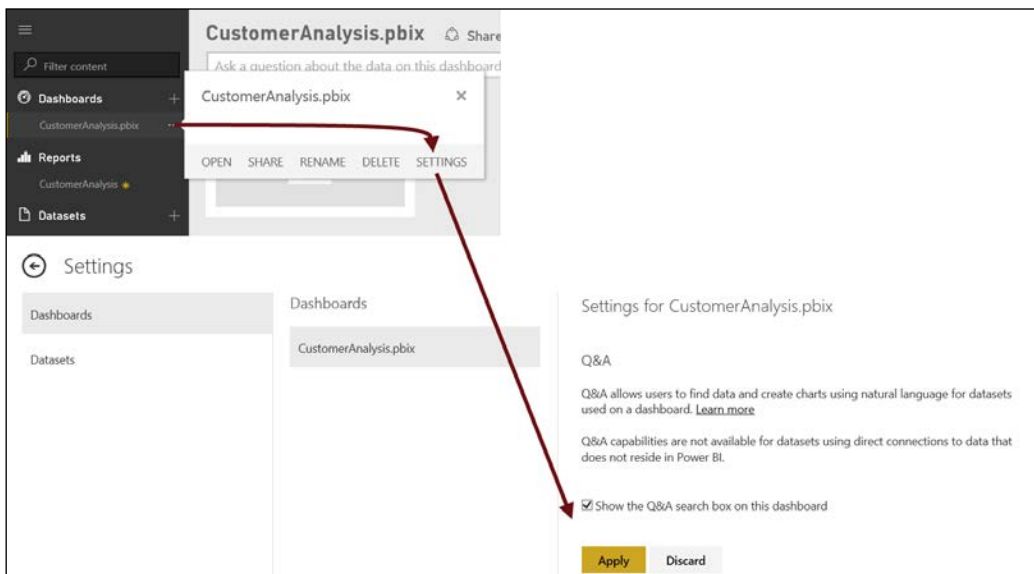
Once you sign up, you can use the website, as an alternative to Power BI Designer, to create reports, dashboards, queries, and so on. You can also upload Excel files, other types of files, and previously created Power BI Designer files, as in the following example, where I will upload the dashboard you created with Power BI Designer:



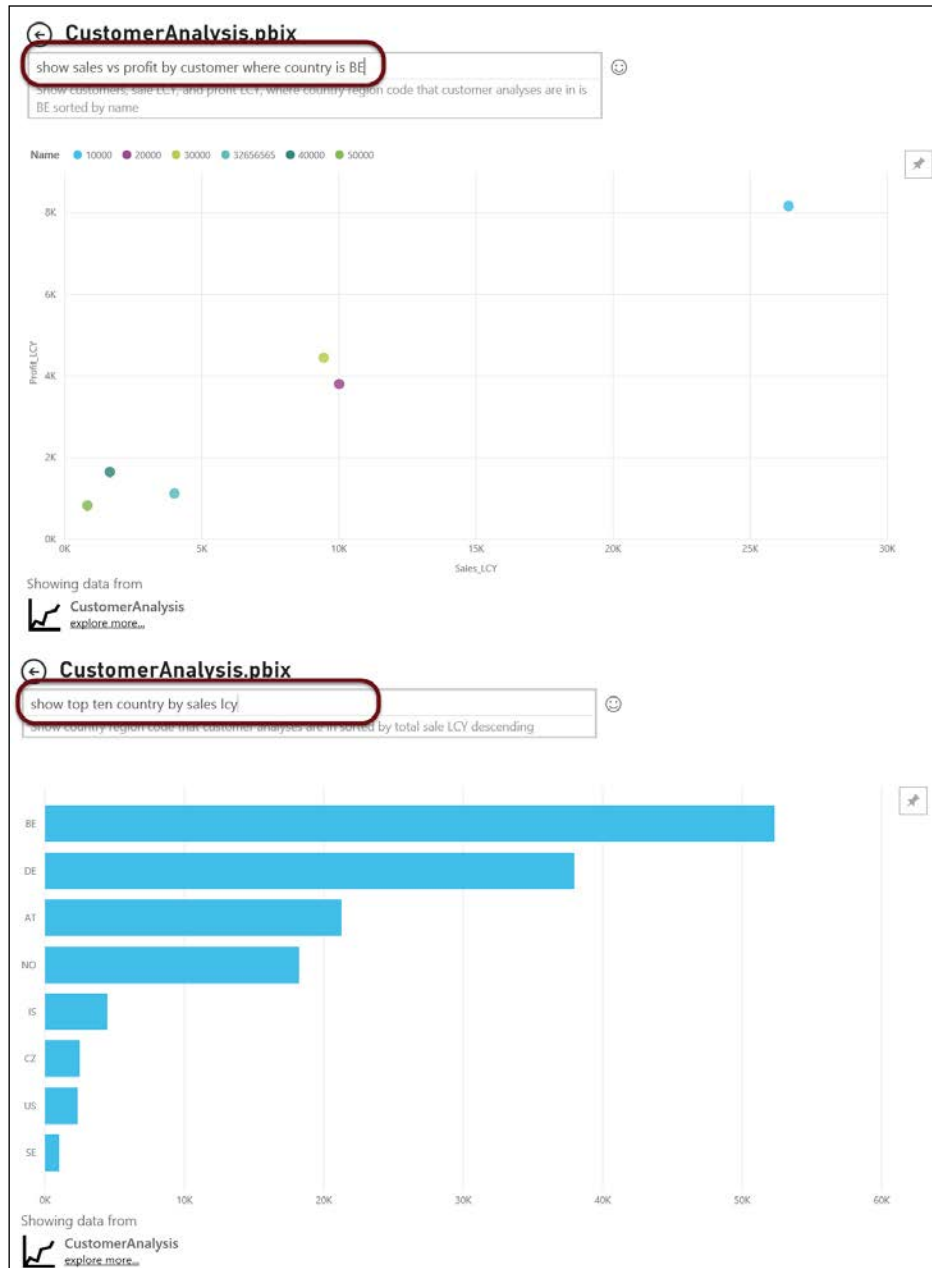
Once the report is uploaded, it becomes available as a dashboard:

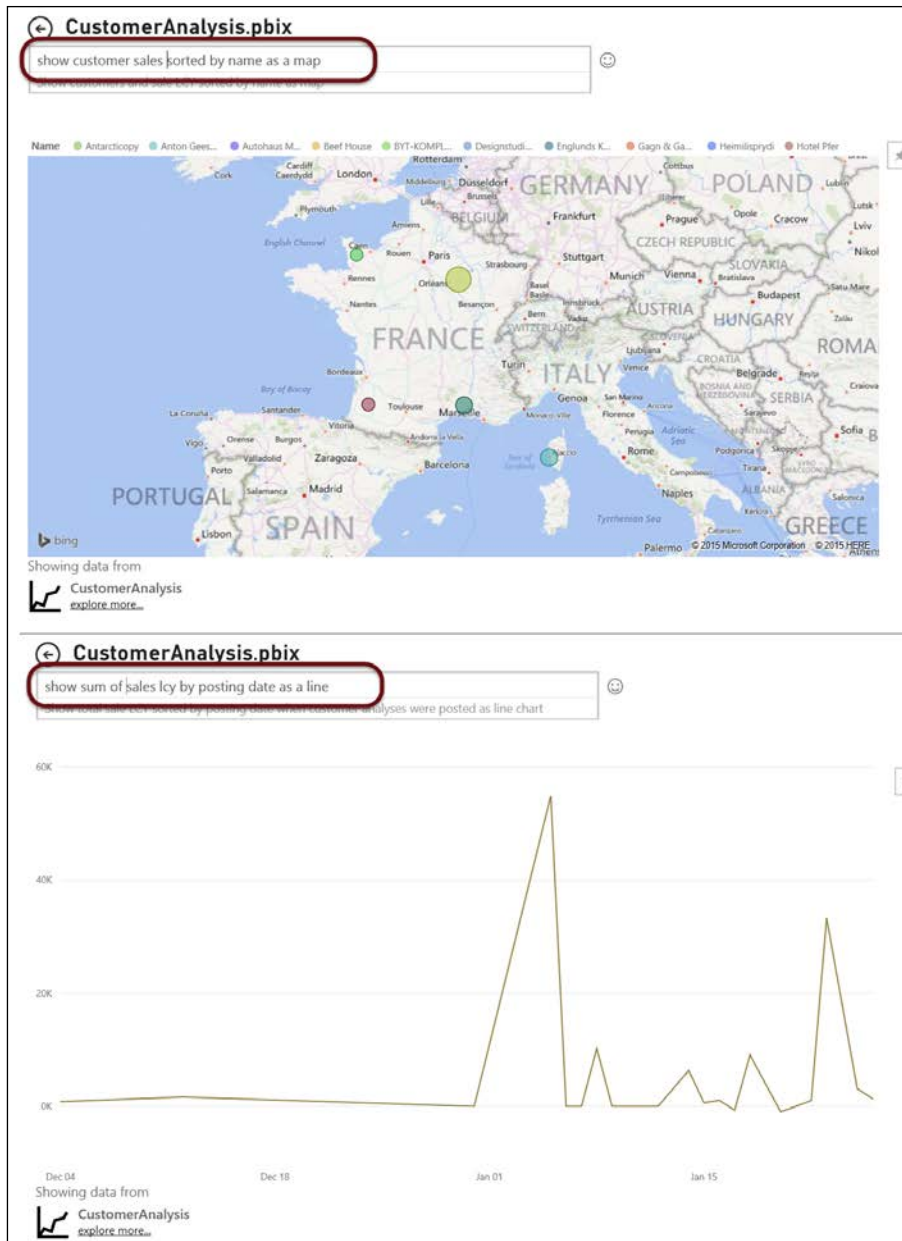


Now I will enable Q&A for this dashboard, as follows:



You can then start typing in a question in natural language, at the top, in the search bar. You can see four charts in the following screenshot, that were created on the fly when typing in the questions marked in red:





Now you have a situation where the CFO of your company, or any other user, gets what they always dreamed of. A dashboard where they can simply ask a question and the chart is generated immediately. How cool is that!



Of course, the accuracy and user friendliness of this feature depends on how you name your fields in the dataset. You might remember, in the section about *Power Pivot*, I showed you the user-friendly names and synonyms section. Well, this is where that has its effect.

You can do much more with Power BI than what I have shown you here in this chapter. This chapter is meant as an introduction, where I show you some of the basic features and how you can get started.



More information about Power BI and Q&A is available here:

Power BI Q&A in Office 365: Searching and Querying with natural language:
<https://support.office.com/en-za/article/Power-BI-Q-A-in-Office-365-Searching-and-Querying-with-natural-language-709ef848-660b-4610-9b40-9395392c38af?ui=en-US&rs=en-ZA&ad=ZA>

Summary

In this chapter I have introduced you to the world of Power BI. You have seen how you can use Power Pivot to create a data model using Dynamics NAV web services. You then created interactive visualizations and dashboards with Power View and Power Map. Power BI Designer combines these tools and enables you to manage and create reports.

In the next chapter, I will introduce you to reporting services and you will learn how you can use it to create reports based on data from Dynamics NAV.

10

Reporting Services

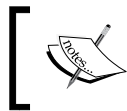
In this chapter, I will introduce you to **Reporting Services**. When RDLC was introduced to Dynamics NAV, the technology was not new. In fact, it came from SQL Server Reporting Services, but implemented in such a way that no report server was required. Now Reporting Services is still available as a, free, and very good reporting tool. You can use it to create interactive reports based on Dynamics NAV data and in this chapter, I will show you how you can get started with it.

What are Reporting Services?

When SQL Server 2000 was released, it contained a new reporting feature named Reporting Services. Since then, it has gained enormous popularity and its technology is currently used in Microsoft Dynamics and other applications. Dynamics NAV, AX, GP, and CRM all use reporting engines based upon Reporting Services technology.

It's important to know what you can do with it because Reporting Services has much more to offer than the built-in reporting tools so you can make an educated choice when you decide to use an external reporting tool and compare it with other third-party applications.

First of all, it's completely free; it comes with SQL Server. When you make the investment in SQL Server, you get the reporting service (and analysis services) as part of the package. The only investment needed is the time to develop reports.




Of course, you need to have SQL Server Edition. Since every Dynamics NAV database (since the RoleTailored Client) runs on SQL Server, you will already have SQL Server installed.

Secondly, you can create reports with Reporting Services that visualize information from a wide range of data sources. Apart from SQL Server, Reporting Services can connect to OData web services, or any database that supports ODBC.

You can also use XML, Excel and text files as data sources and, if required, these data sources can be combined. You can also create dynamic reports that display or aggregate information from multiple databases or companies because the queries you use to access the databases can be parameterized.

Installation and configuration

To get started with Reporting Services, you first need to install and configure a report server. The report server executes and runs the reports while you, as a user, access and run your reports from your browser, in an application named **Report Manager**.

 As an alternative, you can also set up Reporting Services to integrate with SharePoint. This is an interesting approach, especially if you already use SharePoint in your company. But it is beyond the scope of this chapter.

Since Reporting Services (SSRS) are a part of SQL Server, you need to start with the installation media for SQL Server and deploy the SSRS report server. During installation, you will get the choice between installing the report server and installing the report server and automatically configuring it:

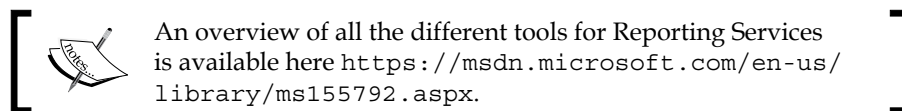




As a minimum, you require the SQL Server:

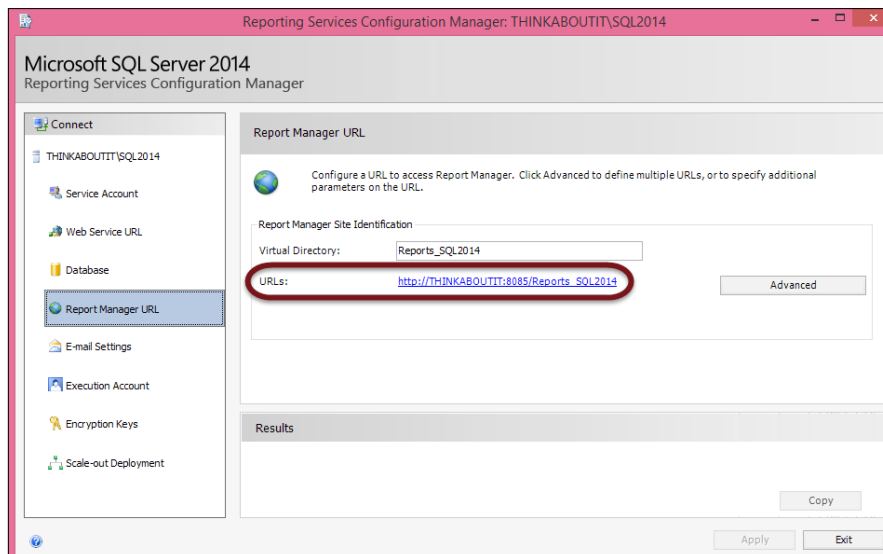
- Database Services
- Reporting Services

I recommend the Management Tools (basic or complete), so that you can then manage databases, security, backups, and so on.

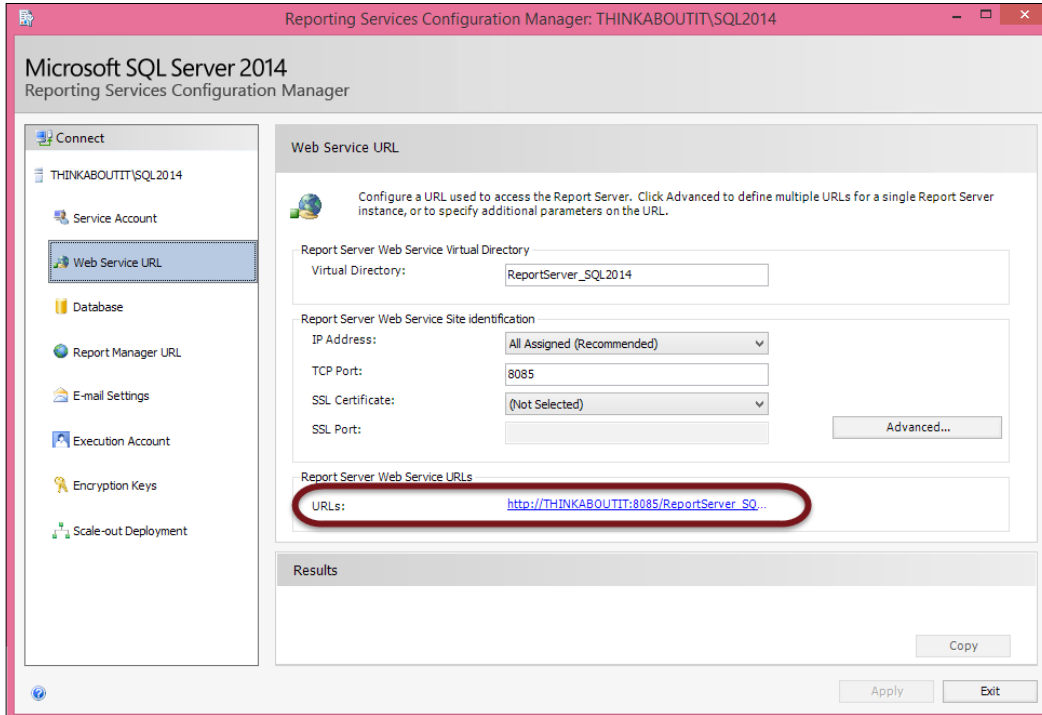


When you select **Install and Configure**, the installation procedure creates a report server database automatically. This is the database that will store your Reporting Services configuration, published reports and security. You can reconfigure the settings and also have a look at the URL that was used to access the Report Manager and report server using the **Reporting Services Configuration Manager**.

The following is a screenshot of the **Reporting Services Configuration Manager** window:

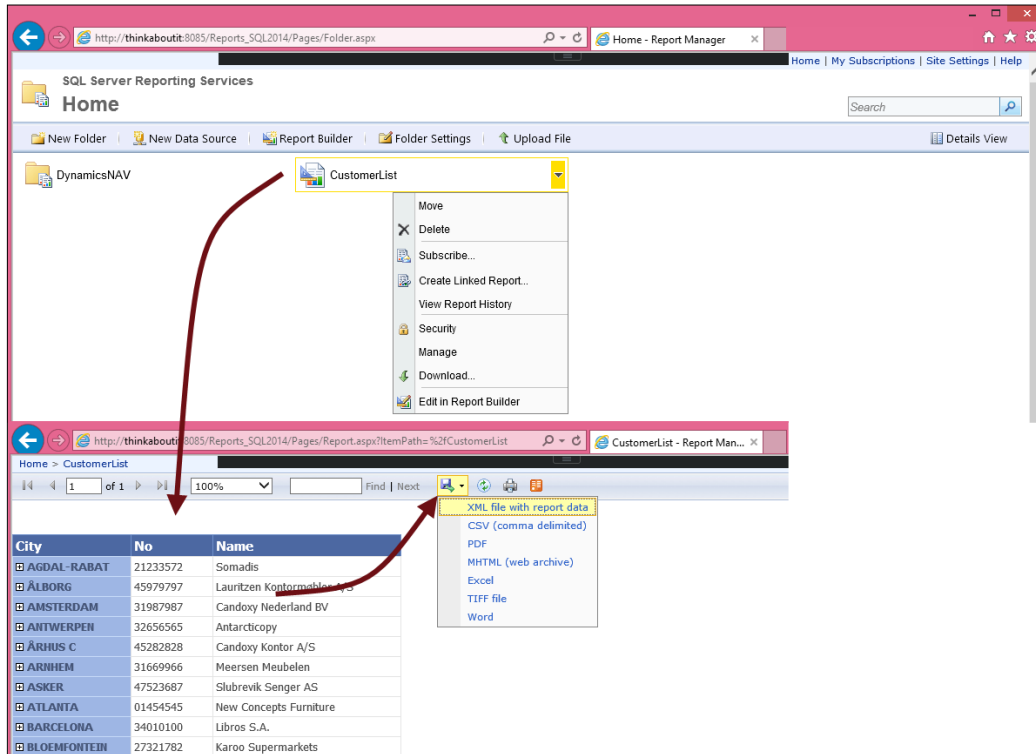


You can find the **Report Server Web Service URL** in the **Web Service URL** section:



A user will use the Report Manager to consult the Reporting Services reports. A developer will publish reports to the report server, after which they will become available in the Report Manager application.

You can see and run the reports published to the report server in the Report Manager:



For a detailed description of how to use the Report Manager, have a look at <https://technet.microsoft.com/en-us/library/ms157147.aspx>.


For a detailed description of the installation and configuration of the Reporting Services, have a look at <https://technet.microsoft.com/en-us/library/ms159106.aspx>.

Creating a report in SSRS

To create a report with SSRS, you can use either:

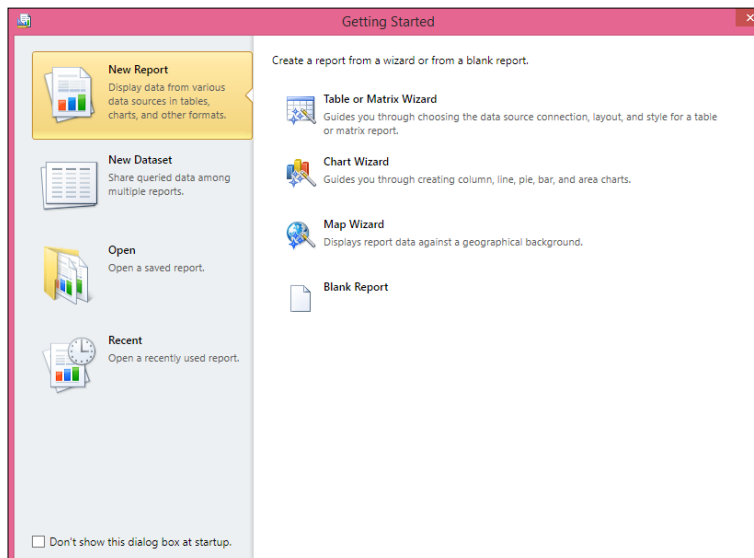
- Report Builder
- SQL Server Data Tools (SSDT-BI)

Both of these tools can be downloaded, free from the Microsoft website. The difference between the two is that with Report Builder you are working on one report. With Data Tools you can create and manage multiple reports in one project or solution.

 SQL Server data tools for Reporting Services is abbreviated to SSDT-BI. (Online you will also find SSDT, which only contains templates for SQL Server project, and not the Reporting Services.) You can download it from here: <http://www.microsoft.com/en-us/download/details.aspx?id=36843>

Now, let's get started on our first report in SSRS. I would like to start with a simple list of customers. I will create it using the Report Builder, and then use Visual Studio data tools to further enhance the report and add it to a solution.

1. First, launch the Report Builder application. When it starts, it immediately opens a popup window:

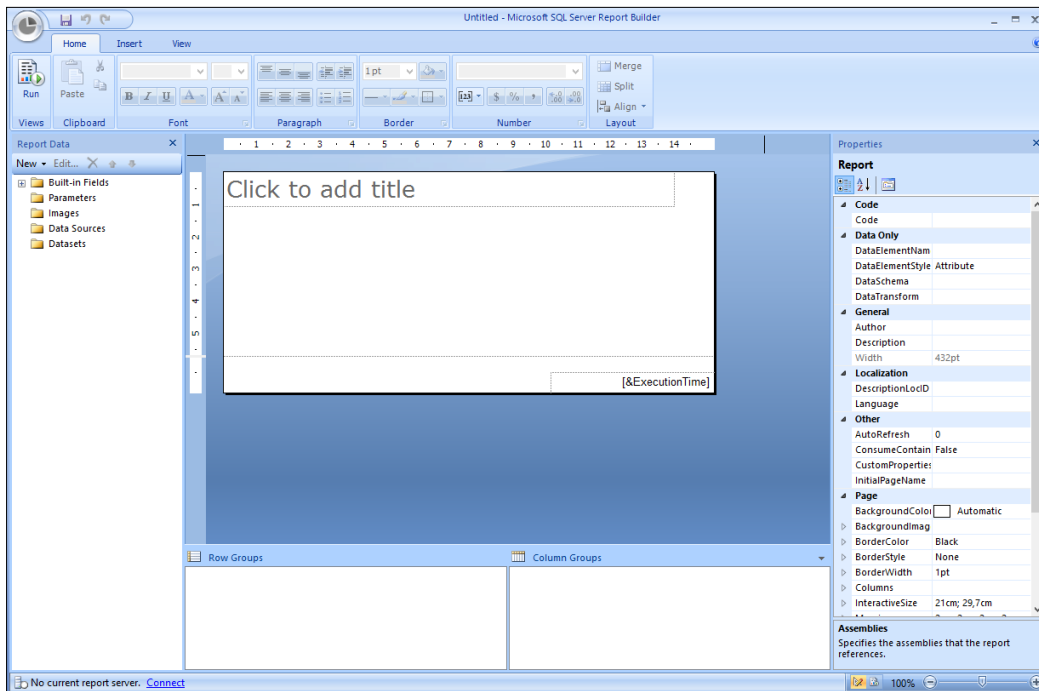


- Here, I will select **Blank Report**, because it's the first report we will create.



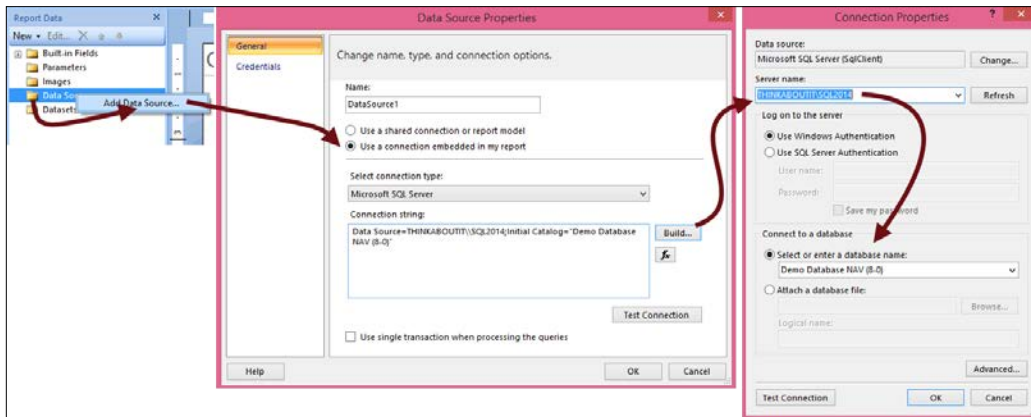
If there are reports already available, I might select one of the other options, so that I can reuse some of the parts of the report I have already created, or parts that might be available on the report server. But I will come back to this later in this chapter.


- When you select **Blank Report**, the following window opens:




This will look familiar to you if you have used the Report Builder to create the RDLC layout for a Dynamics NAV report object. The difference here is that the data source and dataset is empty. First, you will have to create a data source, which is a reference to the Dynamics NAV database on SQL Server, and a dataset, which is an SQL query to fetch the customers from that database.

1. To create a data source, right-click on the **Data Sources** folder and select **Add Data Source...**. This opens the **Data Source Properties** window, in which you can create a link to the Dynamics NAV database on SQL Server:

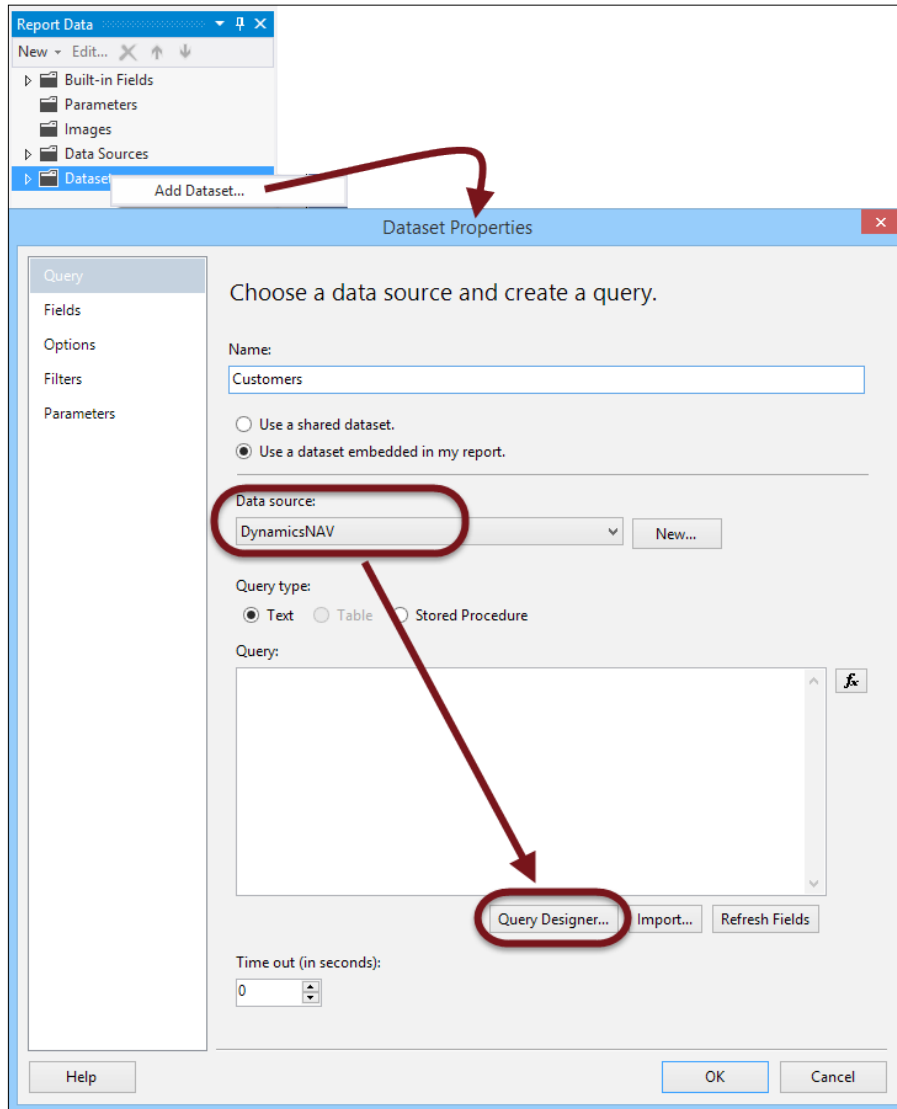


 In your case, the connection string to your database is probably different. It contains the name of your server (or local machine, if installed locally), SQL instance and database. Later, we will use a reference that we will store on the report server. It's easier to maintain that way.

The name of the data source is `Datasource1` by default, but I recommend renaming it `DynamicsNAV`.

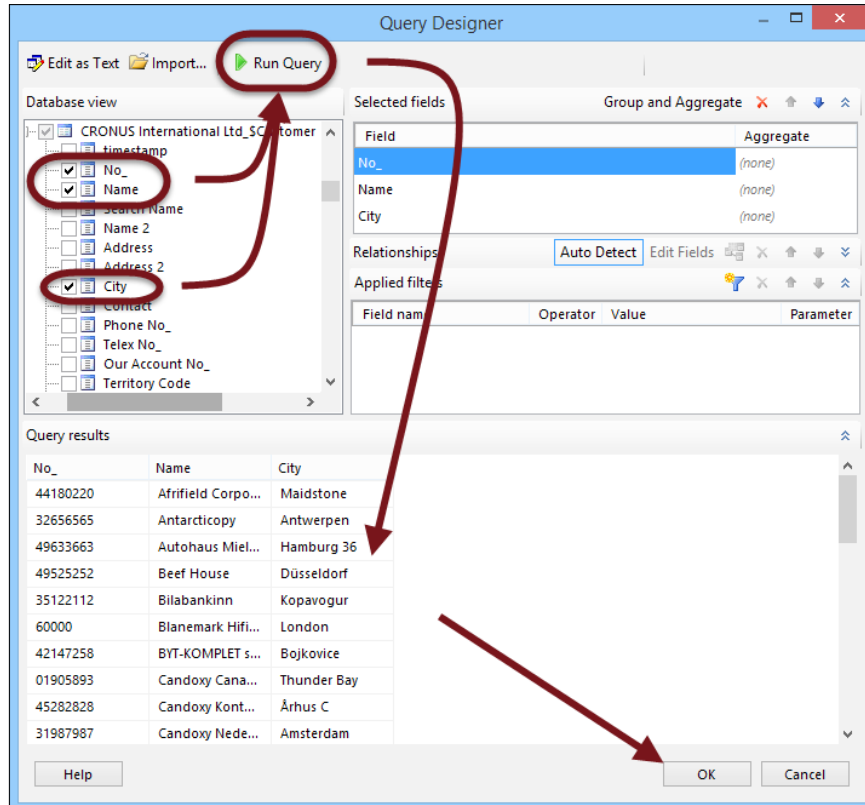
 **Use a reporting user**
Windows authentication is used by default in the connection string. This means that a user has to be created or rights have to be assigned to every user that wants to run this report on SQL Server. I recommend only one dedicated database user. You can give it the name `ReportingUser` and only give it read rights on the tables of the Dynamics NAV database. Security about who has which rights or roles in **Reporting Manager** can be managed in the **Report Manager** settings, and is completely separated from the user that is used to authenticate on the source database.

- Now that you have created a data source, you can create a dataset. Right-click **Datasets** in the **Report Data** pane and select **Add Dataset....**:



- Type **Customers** in the **Name** field of the dataset.

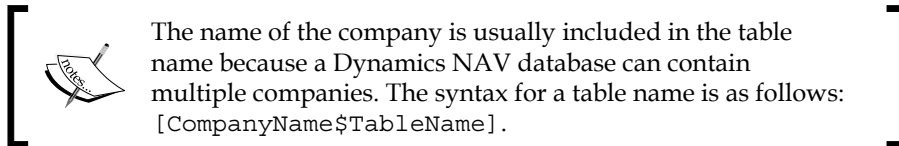
4. Select the **DynamicsNAV** data source you created earlier in the **Data Source** field.
5. Click on the **Query Designer...** button. This opens a popup window. There, in the list of tables on the left side, expand the tables and look for the customer table.
6. Expand the customer table and select the fields: **No_**, **Name** and **City**.



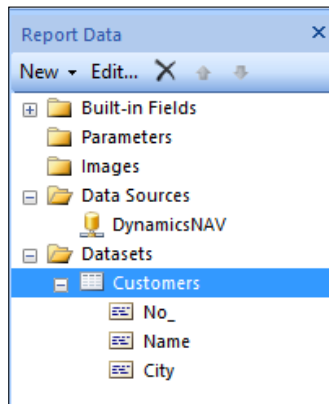
7. Then click on **Run Query** to see the results. Click on **Ok** in the **Query Designer** window and create the following query:

```
SELECT
  [CRONUS International Ltd_$Customer].No_
  , [CRONUS International Ltd_$Customer].Name
  , [CRONUS International Ltd_$Customer].City
FROM
  [CRONUS International Ltd_$Customer]
```

As you can see, it selects the **No_**, **Name** and **City** fields from the customer table in the Dynamics NAV database.



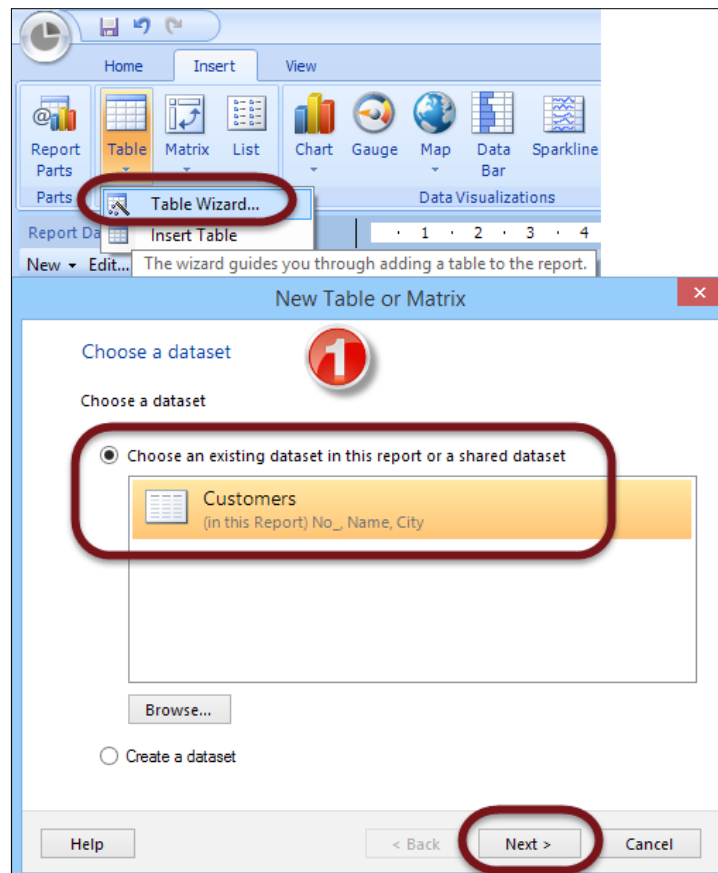
After you close the dataset designer, the dataset is updated in the Report Builder of **Report Data** pane, and now contains the fields from the customer table:



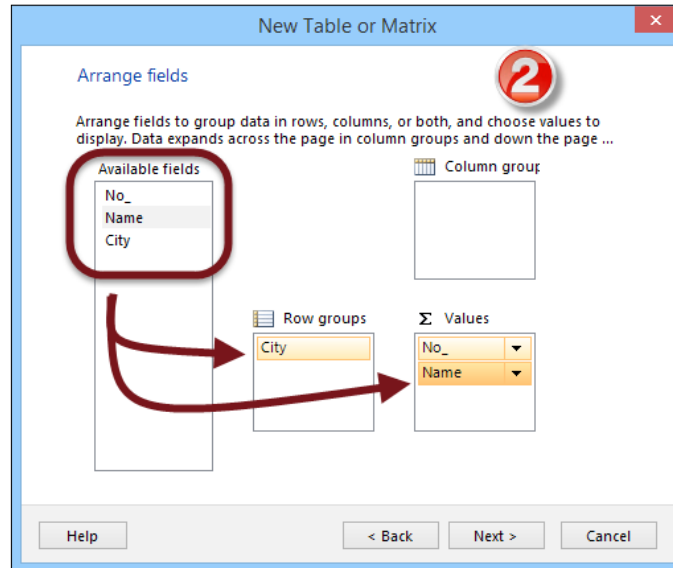
From now on, building the report layout will be very similar to building the layout of an RDLC report.

I will use a wizard to add a table to the report body that will list the fields from the dataset, as follows:

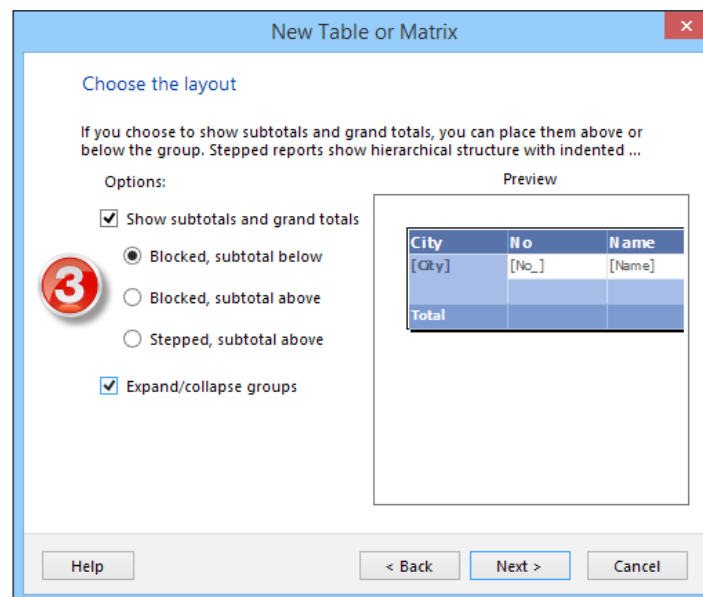
1. First, select **Table, Table Wizard...** to start the wizard. Then, select the **Customers** dataset and click **Next**:



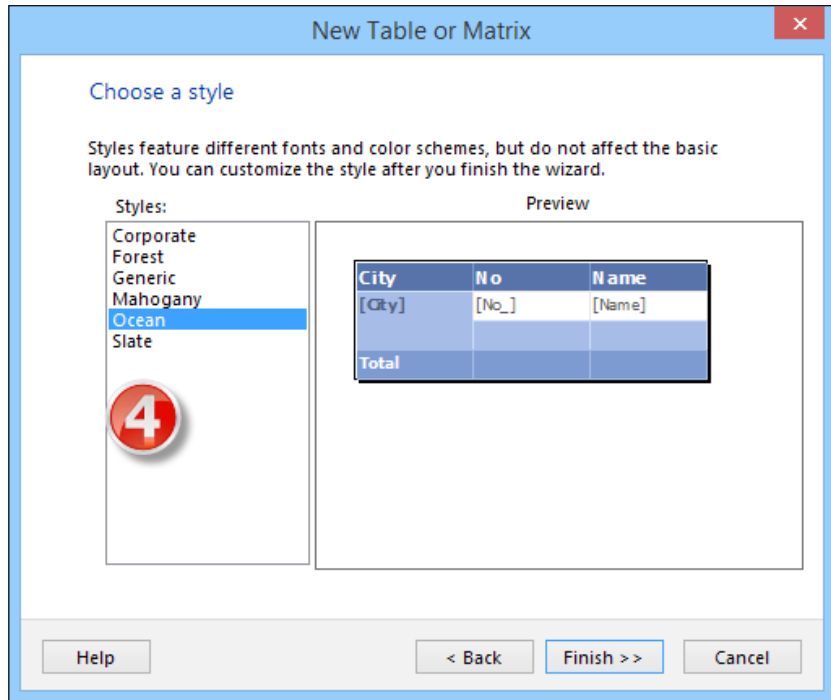
- Now drag and drop the **City** field on the **Row groups** and the **No_** and **Name** fields on **Values**, and click **Next**:



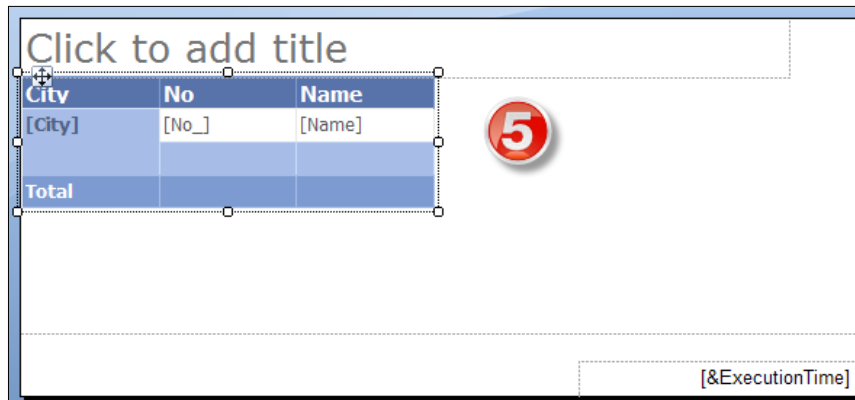
- In the next window, you can enable or disable the **Expand/collapse** options and the option to see totals and subtotals, and then click **Next**:



4. You can select a **Style** in the following window:



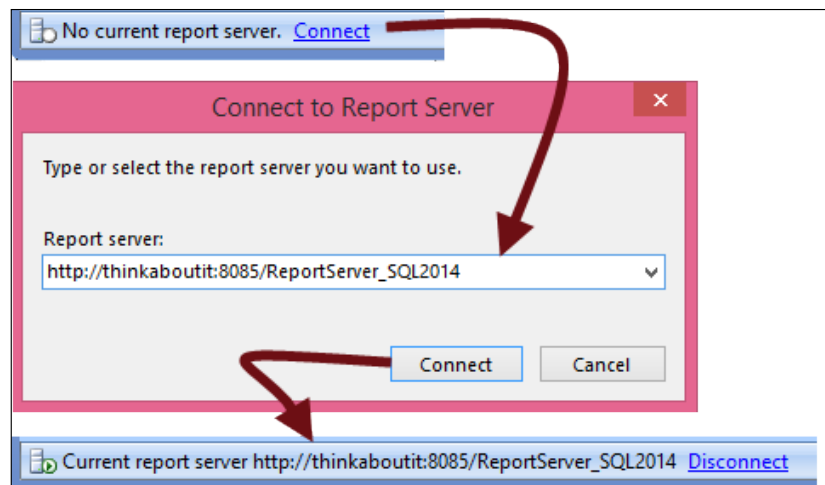
5. When you click **Finish**, the table is added to the report body:



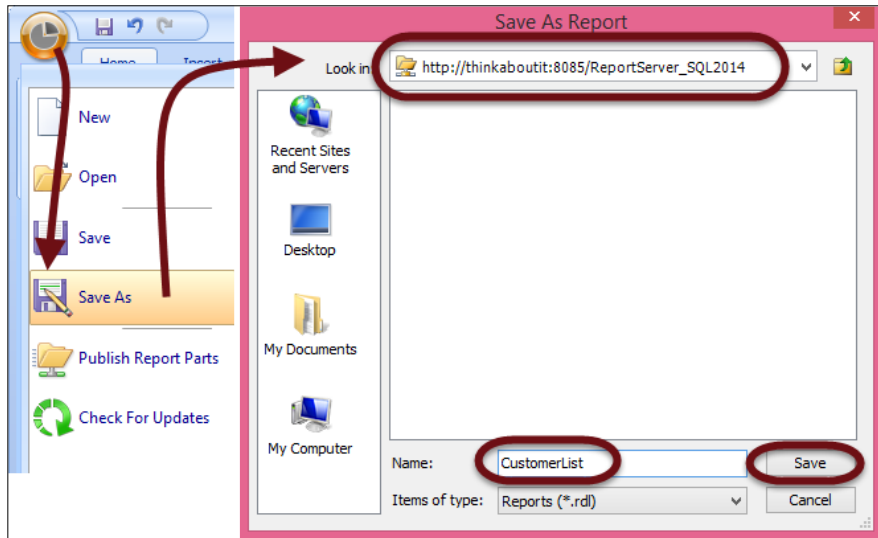
- After you follow the wizard, the table is added to the report body. You can then run and preview the report with the **Run** button at the left top, in the ribbon:



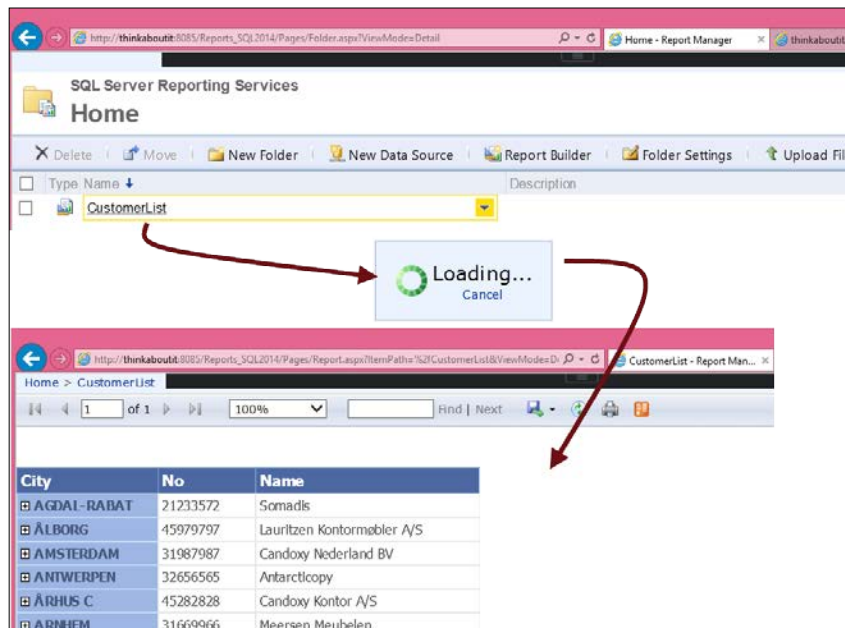
- Now I will publish (or save) my report on the report server. You can see the report server and Report Manager URLs in the Reporting Services configuration manager:
 - Report Server: `http://thinkaboutit:8085/ReportServer_SQL2014`
 - Report Manager: `http://thinkaboutit:8085/Reports_SQL2014`
- You can connect to the report server at the left bottom of the Report Builder, as follows:



- Once connected, you can use the **Save** or **Save As** options of the button in the ribbon to save your report on the report server. Select the **Recent Sites and Servers** option if it's not available in the dropdown at the top:



- Then, go to the report viewer, in your browser, look for the report, and run it:



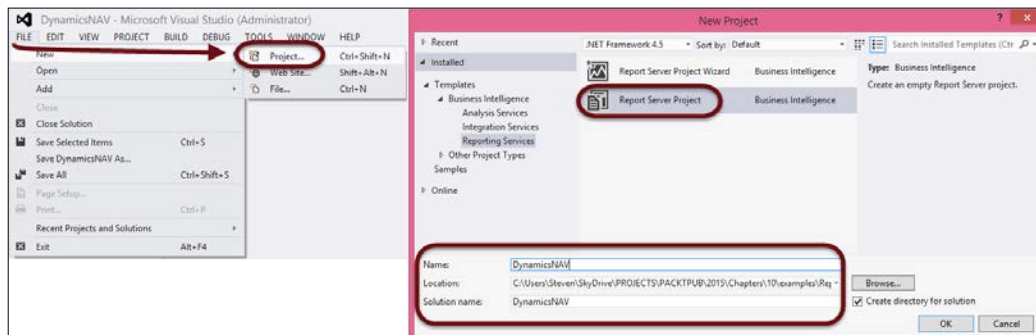
You have now created and published your first Reporting Services report.



If you receive an error (**You don't have sufficient permissions to perform this action.**) while running the Report Manager or trying to save the report to the report server, it's usually because of **User Access Control (UAC)** settings in Windows. Relaunch your browser and the Report Builder and run it as an administrator to fix the problem.

Using SQL Server Data Tools

Now we are going to create a new reporting project in Visual Studio, using SSDT. Go to the **File** menu after you launch SSDT and create a new project as follows:



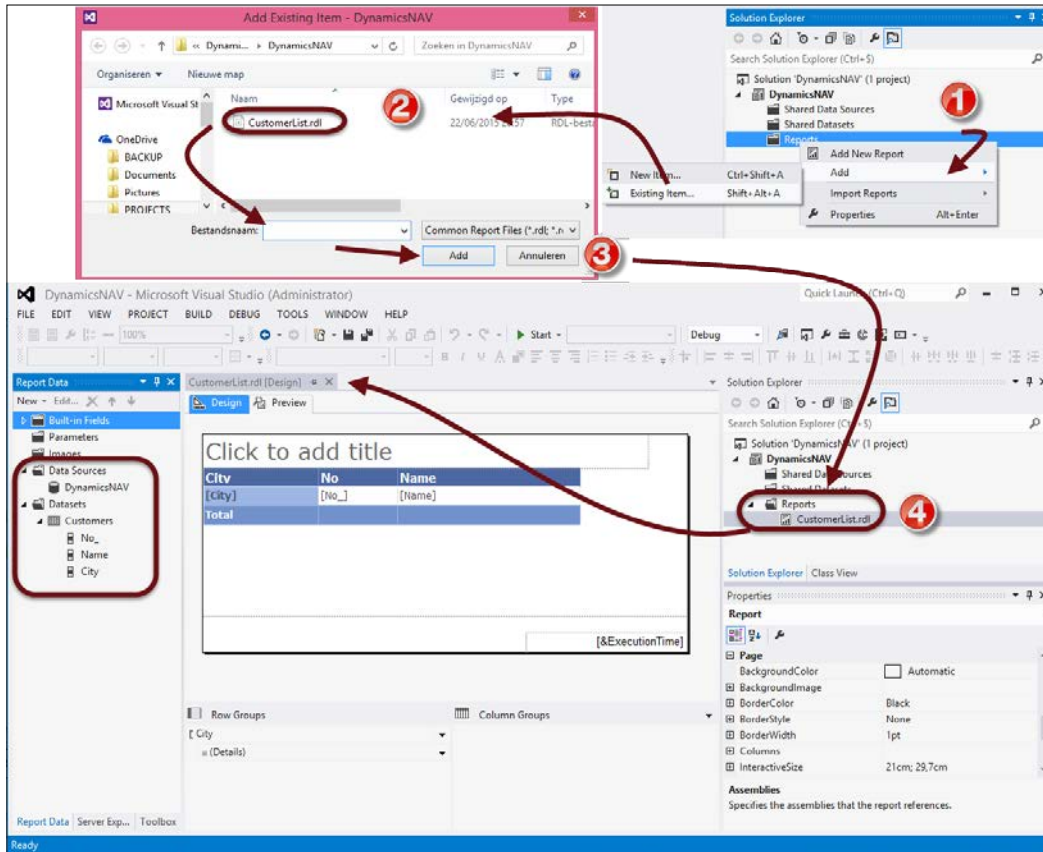
Go to **Business Intelligence** in the **templates** section and select **Report Server Project**.



If you don't see the **Report Server Project Template** here, it is probably because you are not using the correct version of Visual Studio or don't have SSDT installed.

You can download it from here: <http://www.microsoft.com/en-us/download/details.aspx?id=36843>

This creates a new empty solution. Then, I will import the report I created earlier, `customerlist.rdl`, into this solution:

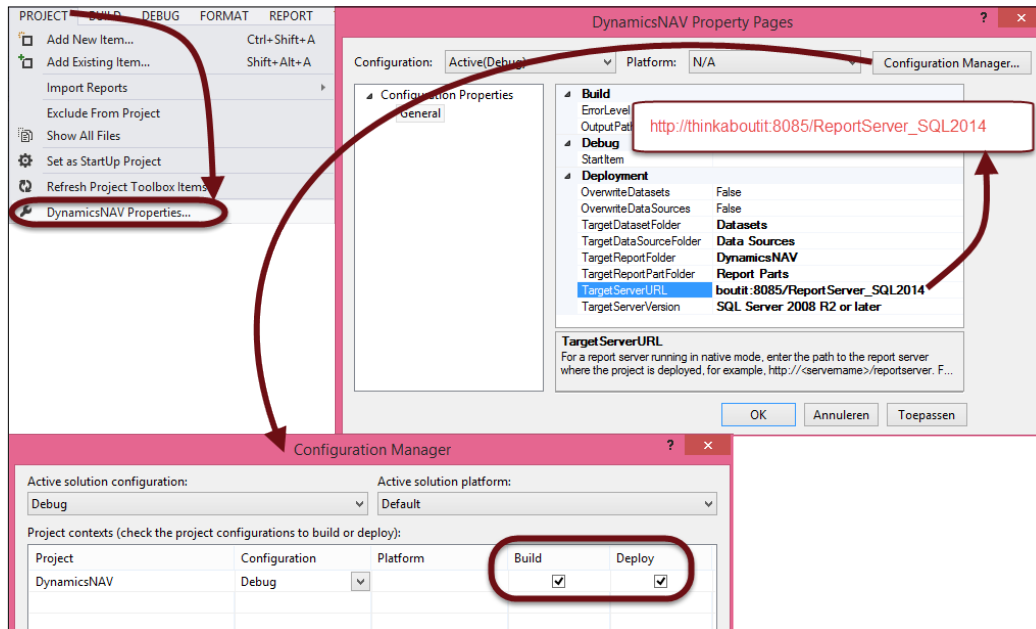


When you double-click the **CustomerList** report in the **Reports** folder, it opens in Visual Studio. The report body is displayed in the middle. The data sources and datasets are displayed on the left, in the **Report Data** pane.

You can now create new reports in **Solution Explorer**, in the **reports** folder. All of them become part of this solution, so you can manage and deploy them to the report server together.

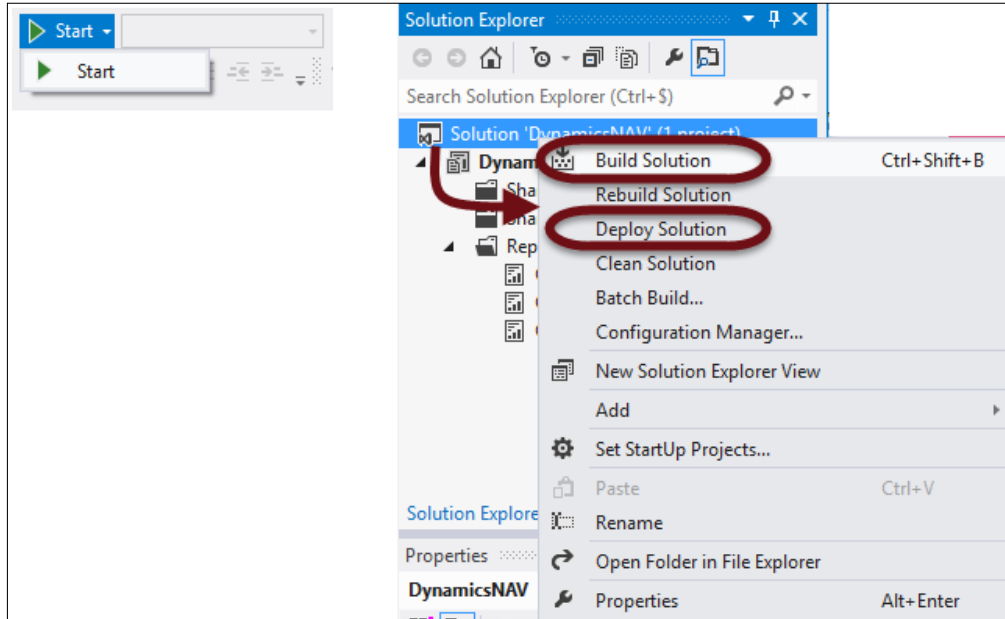
Publishing a report project

You first have to go to the report properties in SSDT and indicate where the report server is located, so that you can publish your reports to it:

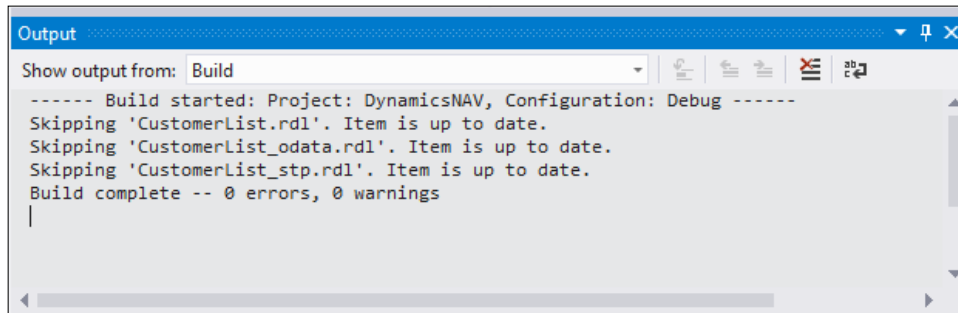


Here, you enter the address of the report server, which you can find in the Reporting Services configuration manager, in **TargetServerURL**. Then, you enable Build and Deploy using **Configuration Manager**....

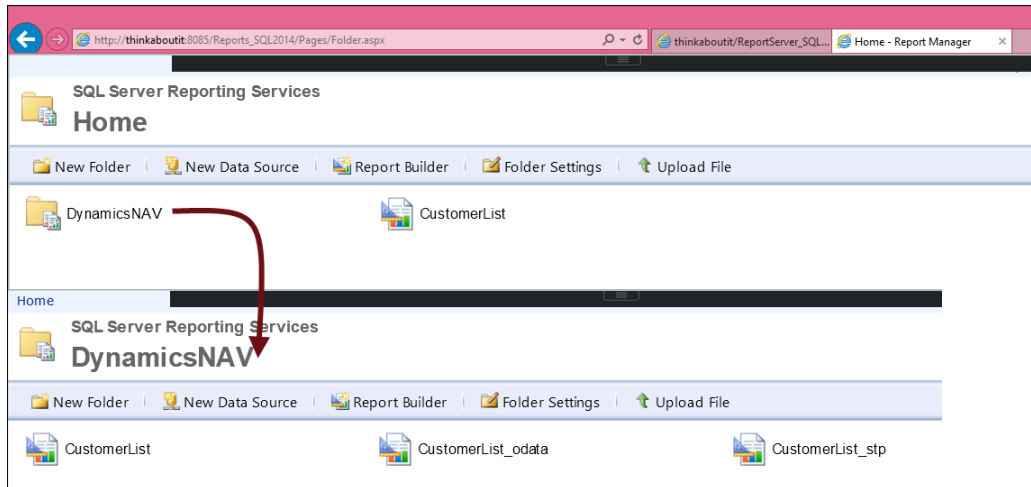
To publish your project, use the Play button, or right-click your solution and select **Build Solution**, then **Deploy Solution**:



The output window at the bottom of Visual Studio indicates whether or not it worked while deploying your solution:



After deploying, it will launch your browser and navigate to the report server. You can navigate to the Report Manager website manually, where you will see the deployed reports. If you select them, they will execute:



You can specify security, for example, who can run or manage reports in the **Report Manager Site Settings** and in the properties of each folder and report.


Implementing reusability

When you create reports with one or more developers, there are features available in Reporting Services that can be reused in other places. For example, you can share data sources, datasets, and report parts. Furthermore, when you create queries for your reports, it is interesting to use views, functions, and stored procedures, and reuse them in different reports or projects.

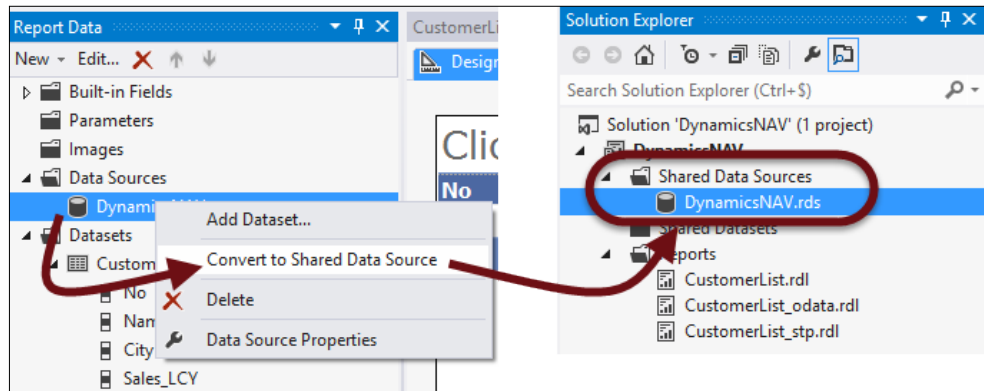
Shared data sources and datasets

Every time you create a report, you can either store the data source reference in the report itself or store it in your project and share it among all the solution reports. You can then publish it separately to the report server. The advantage of this is that all reports use the same reference to the database. So, if you change the reference and point it to another database, all reports are redirected together.

The same is valid for datasets. If you have different reports that use the same datasets, it's better to create the dataset just once, and then share and reuse it in the other reports.

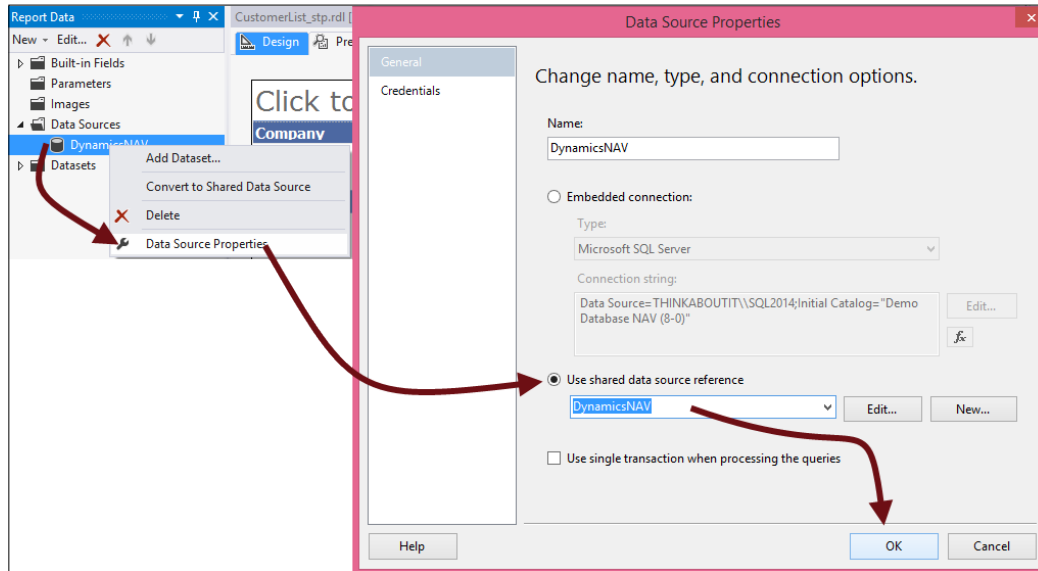
 Although sharing datasets sounds like a good idea, when multiple report developers work together, it can become messy very quickly. That's why you should agree upon a methodology for dataset names and where they are published. You should also document which report uses which shared dataset. If you need to make a modification to a shared dataset afterwards, it will impact on all the reports that use it.

Start from inside your report to convert a data source into a shared one. Go to the **Report Data** pane and right-click on its data source. Then, select the option **Convert to Shared Data Source**:



After that, it will become available in your project under **Shared Data Sources**, and your report will point to it. Now you need to go into the other project reports to make sure they are using the shared data source. You can do that with the data source properties.

You need to change from the embedded connection to a data source reference, as shown in the following screenshot:

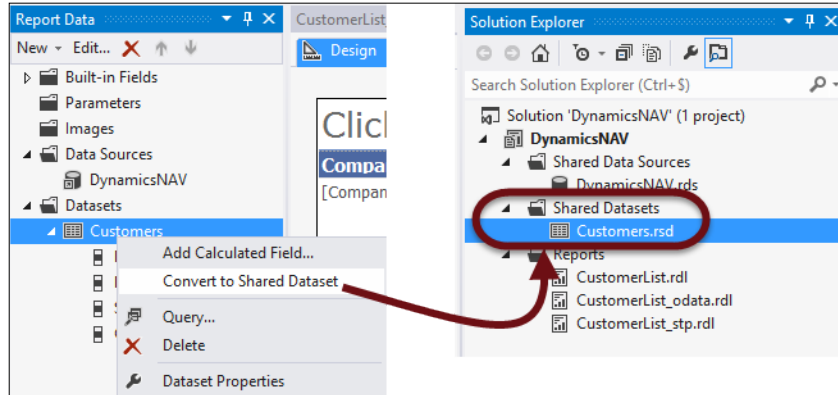


The next time that you publish your project, the shared data source will also be published, and the reports on the server will be updated. You can access and change the shared data source in the Report Manager.

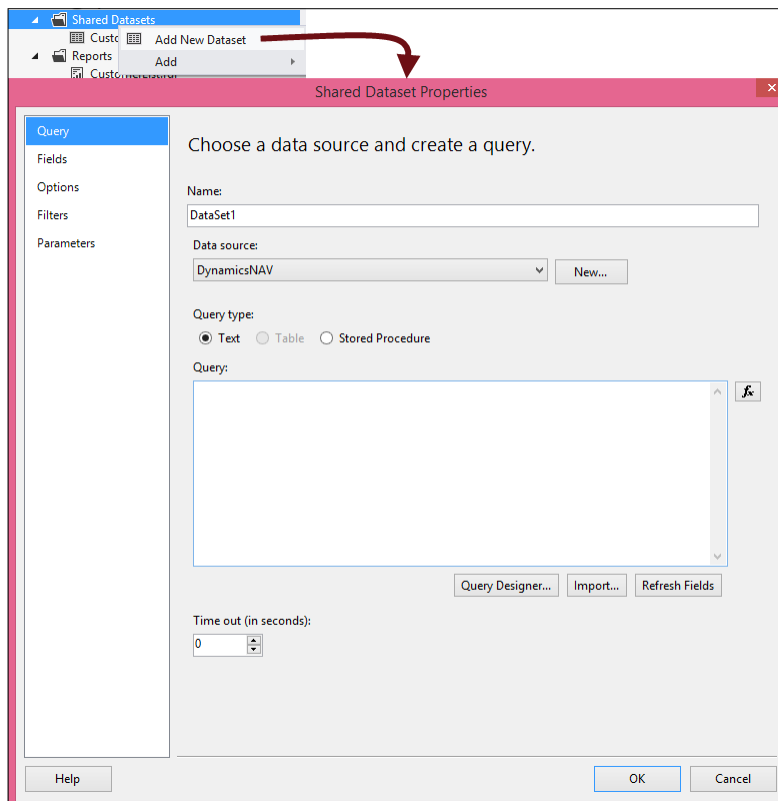
This is interesting because, when you are developing and testing your reports, you can let them reference a development or test database. Then, when development is completed and all tests are successful, you can update the data source on the server and let it reference the live database, without needing to change any of the reports.

You can have one or more datasets in a report. Datasets are the queries to your data source, the database. Some reports use the same datasets. Instead of maintaining all of these datasets in the report definition, you can create a shared dataset, similar to a shared data source.

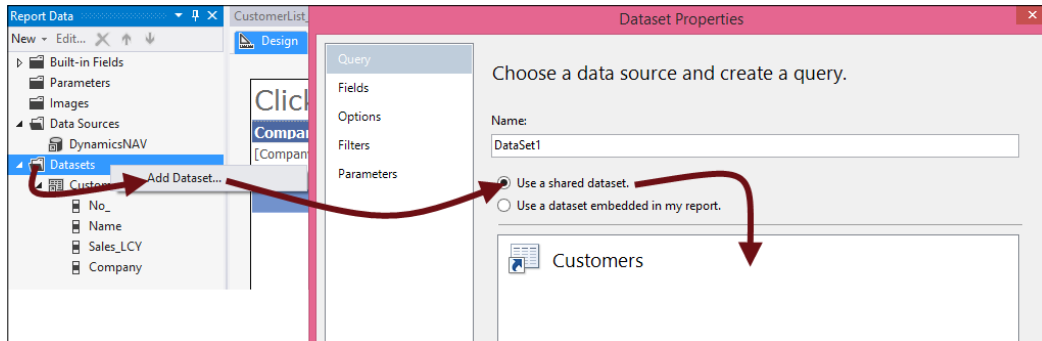
The following screenshot shows how you can convert a dataset from a report into a shared dataset:



You can also start by creating a new shared dataset. You do this with a right-click in **Shared Datasets**:



Once it is ready, you can save it by selecting the **OK** button. Then, when you create a new report, you can add a new dataset to it that references the shared dataset, as follows:



You can create reusable reports components by sharing data sources and datasets. You can even divide the work. For example, someone who is skilled in creating queries can create the datasets and share them, and someone else can simply use those shared datasets when they create a new report.

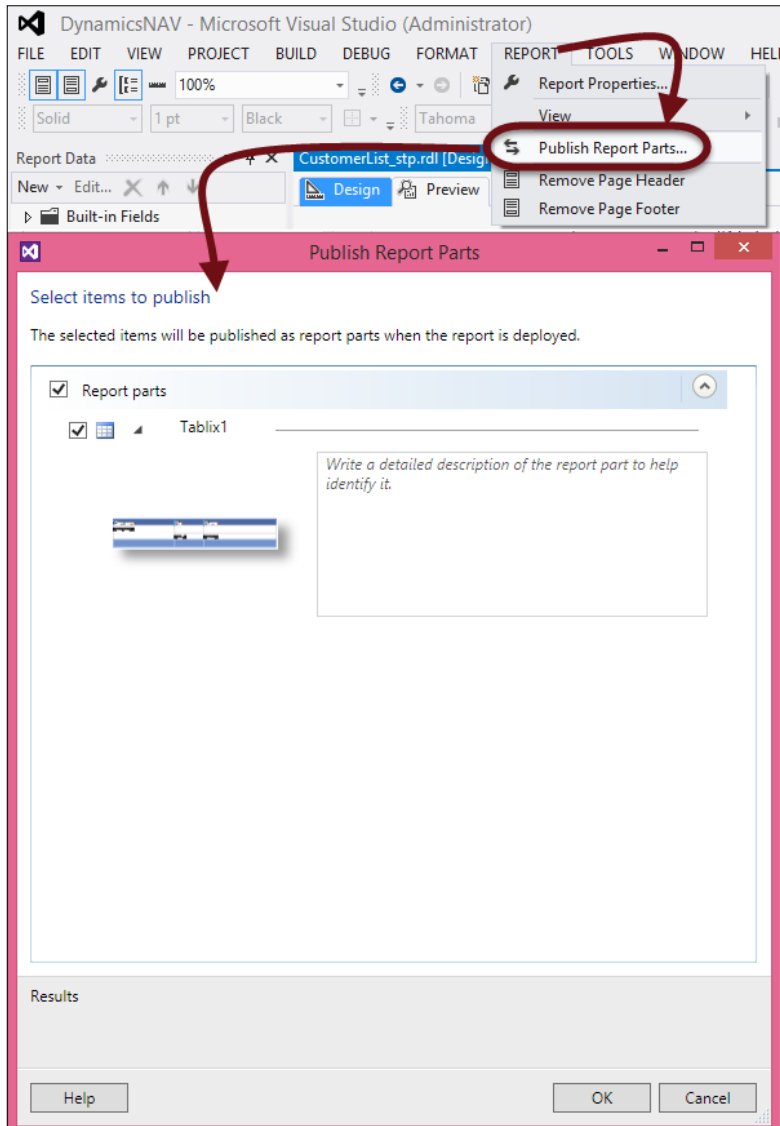
Shared report parts

Apart from sharing data sources and datasets, you can also publish certain parts of a report to the report server. Then, when you create a new report, you can simply reuse them.

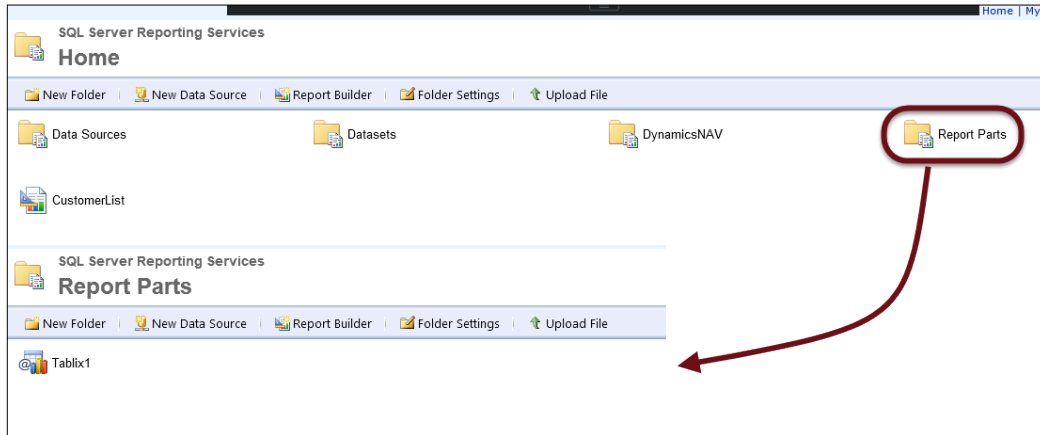
The following are report items that you can publish as report parts:

- Charts
- Gauges
- Images and embedded images
- Maps
- Parameters
- Rectangles
- Tables
- Matrices
- Lists

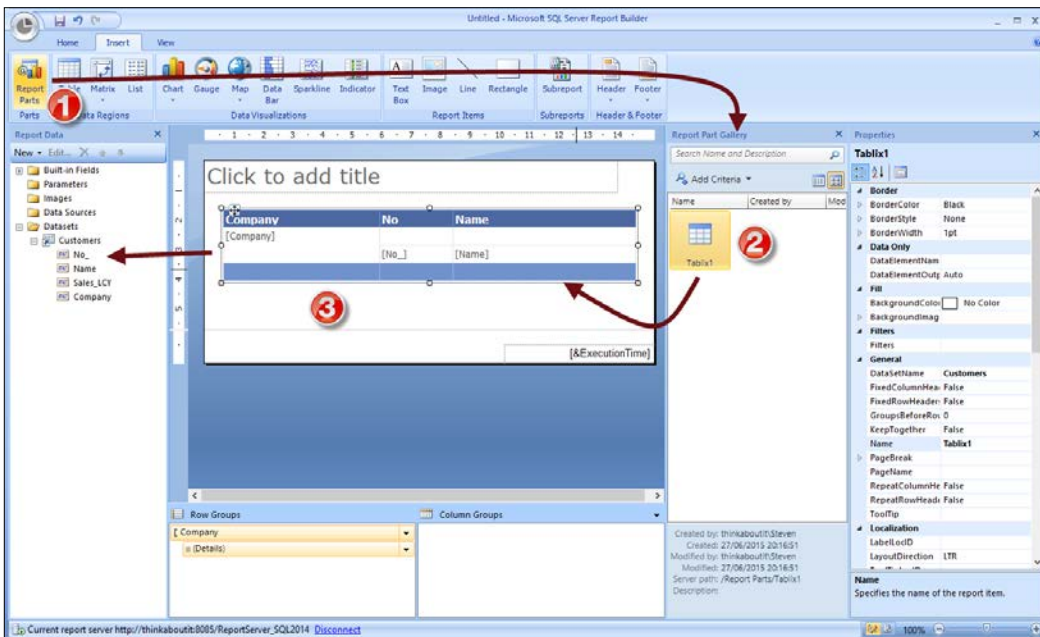
Report parts can be published from SSDT or Report Builder, but only Report Builder supports the functionality to look for published report parts when you create a new report. The following screenshot shows how you can create a report part:



When you publish or deploy the report, the report parts are published to a specific folder on the report server:



There's a **Report Parts** button in the ribbon in Report Builder. When you click on it, a window opens where you can search for report parts on the report server. By selecting and double-clicking on them, they are added to the report, and the shared dataset is also added:



This is a very interesting feature. A user without any technical knowledge can thus connect to the report server and create a report by simply reusing parts. It's a bit like using Lego building blocks, but for reports.

Creating functions

FlowFields in Dynamics NAV are calculated fields. When you create an RDLC report, you reference the table definition directly via the data items, and so, you can use FlowFields. You query the SQL Server database in Reporting Services as FlowFields aren't stored in the database. This means that FlowFields have to be calculated in the query. Let's look at an example, the Sales (LCY) field in the Customer table. In the FlowField calculation formula in Dynamics NAV, the field is calculated as follows:

```
Sum("Cust. Ledger Entry"."Sales (LCY)" WHERE (Customer
  No.=FIELD(No.),Global Dimension 1 Code=FIELD(Global Dimension 1
  Filter),Global Dimension 2 Code=FIELD(Global Dimension 2
  Filter),Posting Date=FIELD(Date Filter),Currency
  Code=FIELD(Currency Filter)))
```

To translate this into an SQL statement that you can use in a customer list, you have to combine the Customer and Customer Ledger Entry tables using the No. field of the Customer table and the Customer No. field of the Customer Ledger Entry table, and then calculate the sum of the field Sales (LCY) for each customer. The SQL query should look like this:

```
SELECT
  CUS.No_,
  CUS.Name,
  SUM(CLE.[Sales (LCY)]) as Sales_LCY
FROM [CRONUS International Ltd_$Customer] CUS
  INNER JOIN [CRONUS International Ltd_$Cust_ Ledger Entry] CLE
  ON CUS.No_ = CLE.[Customer No_]
GROUP BY
  CUS.No_,
  CUS.Name
```

If you have to calculate this FlowField regularly, it might be better to create a function in your SQL database that takes the customer number as a parameter and returns the Sales_LCY value. Then you can call and reuse the function, fGetCustomerSalesLCY, whenever you need it.

Use the following code to create this function:

```

CREATE FUNCTION [dbo].[fGetCustomerSalesLCY]
(
  -- Add the parameters for the function here
  @CusNo nvarchar(20)
)
RETURNS decimal(10,2)
AS
BEGIN
  -- Declare the return variable here
  DECLARE @ResultVar decimal(10,2)

  -- Add the T-SQL statements to compute the return value here
  SELECT @ResultVar =
  (
    SELECT
      isnull(SUM(CLE.[Sales (LCY)]),0) as Sales_LCY
    FROM [CRONUS International Ltd_$Customer] CUS
      INNER JOIN [CRONUS International Ltd_$Cust_ Ledger Entry]
        CLE
      ON CUS.No_ = CLE.[Customer No_]
    WHERE
      CUS.No_ = @CusNo
  )

  -- Return the result of the function
  RETURN @ResultVar

ENDGO

```

Then you could change the SQL statement to fetch the list of customers with the Sales LCY included to:

```

SELECT
  CUS.No_,
  CUS.Name,
  dbo.fGetCustomerSalesLCY(CUS.No_) as Sales_LCY
FROM [CRONUS BELGIË NV$Customer] CUS

```

Now, when you need to calculate this field, in any report, you can simply call this function.



An example of this function is available in the object: `fGetCustomerSalesLCY.sql`. You might need to change the table name in your database.

You can also create functions to perform specific calculations, for example, if you need to look up a dimension for a record, or calculate values over multiple tables.



What this means in practice is that you are rebuilding some of the Dynamics NAV business logic in SQL Server. Try to keep it simple. The more business logic you create, the more you need to maintain. As we will see later, you can use OData web services as data sources in Reporting Services. This is a better alternative for more complex business logic.

Option fields are also very handy option fields. Option fields in Dynamics NAV, for example, the document type in the sales header table, are stored as numerical values in SQL Server. So, you need to convert the numerical value back into a string when you use them in reports. A solution for this is to create a function.

Another solution, and probably a better one, is to create a table in SQL Server where you store all the option fields with their option captions. You will have to create this table yourself, and then a code unit in Dynamics NAV to populate the table.

Using stored procedures

A view in SQL Server is an SQL query that is stored in the database, so you can call it by name and run it. When you query the view, the query behind it is executed. So a view is no more than a virtual table in SQL Server. There's no performance improvement. The advantage is that, if it's a complex query, you can save it as a view and use the name, instead of having to retype the whole query over and over again.

A stored procedure is like a view but it allows you to use variables and parameters. You can then script the behavior and how the query is executed, thus making it very flexible and powerful. Furthermore, when you store a stored procedure on SQL Server, it's not only compiled, but SQL Server also prepares a **Query Execution Plan (QEP)** for it. The STP then runs faster because the QEP is stored with the STP (stored procedure).

The reason I usually implement stored procedures in Reporting Services for Dynamics NAV is to work with multiple companies. You can make the company a parameter, where a user can select one or more and then the report will present its results with the data from multiple companies, or even databases.

Look at the following stored procedure:

```

CREATE PROCEDURE [dbo].[stpCustomers]
  -- Add the parameters for the stored procedure here
AS
BEGIN
  -- SET NOCOUNT ON added to prevent extra result sets from
  -- interfering with SELECT statements.
  SET NOCOUNT ON;

  -- Insert statements for procedure here
  DECLARE @Company as NVARCHAR(30)
  DECLARE @SQLQuery as NVARCHAR(max)
  SET @SQLQuery = ''

  DECLARE db_cursor CURSOR FOR
    select replace(Name, '.', '_') from Company
  OPEN db_cursor
  FETCH NEXT FROM db_cursor INTO @Company
  WHILE @@FETCH_STATUS = 0
  BEGIN
    SET @SQLQuery = @SQLQuery +
      ' SELECT ' +
      '     CUS.No_, ' +
      '     CUS.Name, ' +
      '     dbo.fGetCustomerSalesLCY(CUS.No_) as Sales_LCY, ' +
      '''' + @Company + '''' as Company' +
      '     FROM [' + @Company + '$Customer] CUS ' +
      ' UNION ALL '

    --     PRINT 'xxx'
    FETCH NEXT FROM db_cursor INTO @Company
  END

  CLOSE db_cursor
  DEALLOCATE db_cursor

```


Reporting Services

```
-- ' UNION ALL '
SET @SQLQuery = SUBSTRING ( @SQLQuery ,1, len(@SQLQuery) - 09 )
--SET @SQLQuery = @SQLQuery +

PRINT @SQLQuery
exec sp_executesql @SQLQuery

END
```

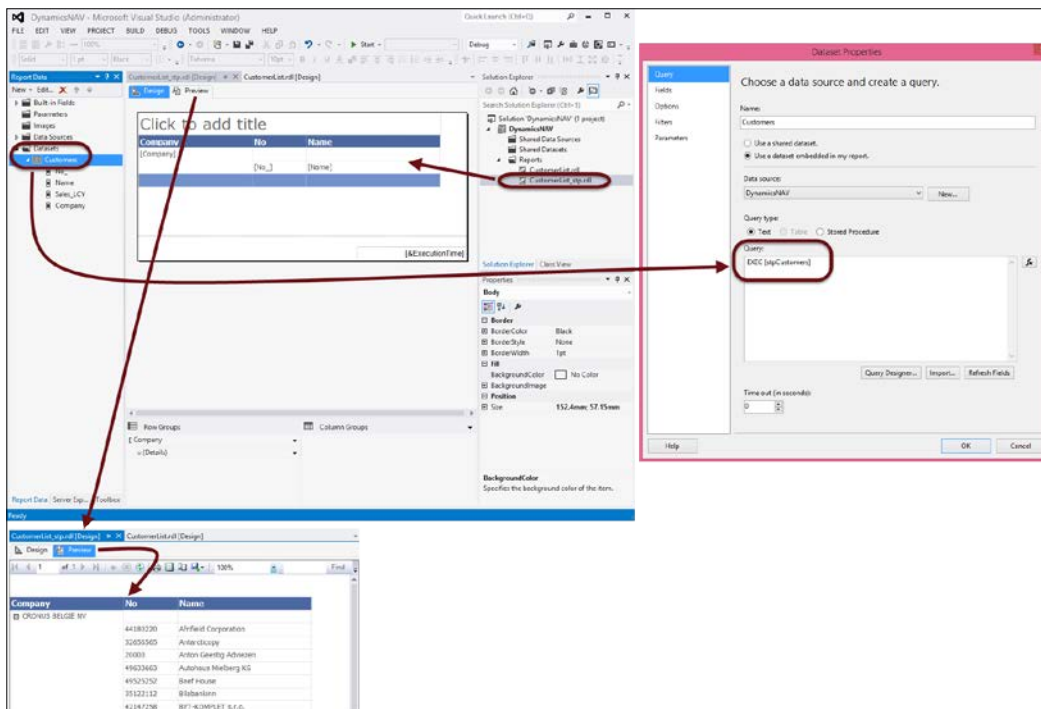
It takes no parameters, looks in the Company table for all company names in the database, and then fetches the customers and glues them together.

[ An example of this stored procedure is available in the object: `stpCustomers.sql`]

To execute this stored procedure, you could simply write:


```
exec [stpCustomers]
```

So, in our report, you could also call this stored procedure instead of the SQL query we created before, as follows:




The screenshot displays the Microsoft Visual Studio interface for configuring a report data source. The main window shows the 'CustomizedList.rdl' report in Design view, with a table containing columns 'Company', 'No.', and 'Name'. The 'Data Sources' pane on the left shows the 'Customers' data source selected. The 'Database Properties' dialog box is open, showing the configuration for the 'Customers' data source. The 'Query type' is set to 'Stored Procedure', and the 'Query' field contains the text 'EXEC [stpCustomers]'. The 'Query Designer' button is visible at the bottom of the dialog. The 'Properties' pane on the right shows the 'Background Color' property set to 'Black'.

The advantage of this approach is that, if you need the customers in multiple reports, all you need to do is call the stored procedure and the SQL query is maintained in the stored procedure. Furthermore, you can make the company a parameter, so instead of gluing (union) the results, you can make the user select for which company they want to see the data.

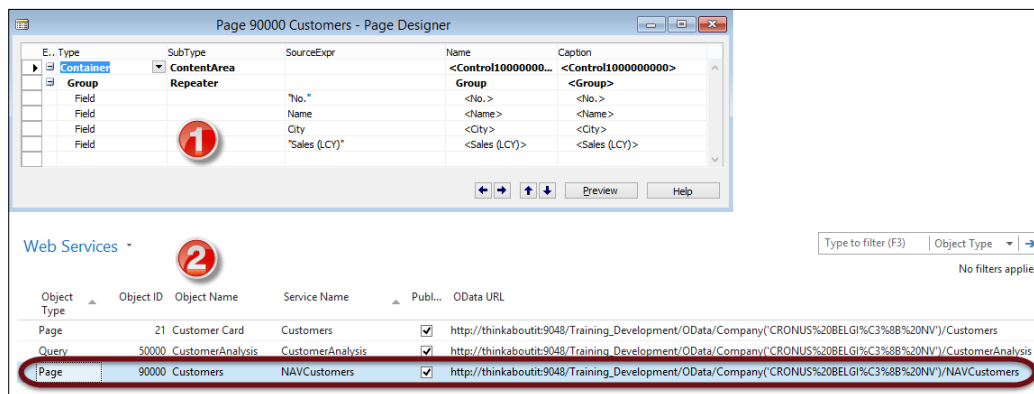
 An example of this report is available in the object: CustomerList_stp.rdl

Calling a Dynamics NAV OData web service

As you already know, you can publish page and query objects and OData web services in Dynamics NAV. Reporting Services reports can also use OData web services as data sources.

 In *Chapter 9, Power BI*, there are examples of how you can publish a query or page as a web service.

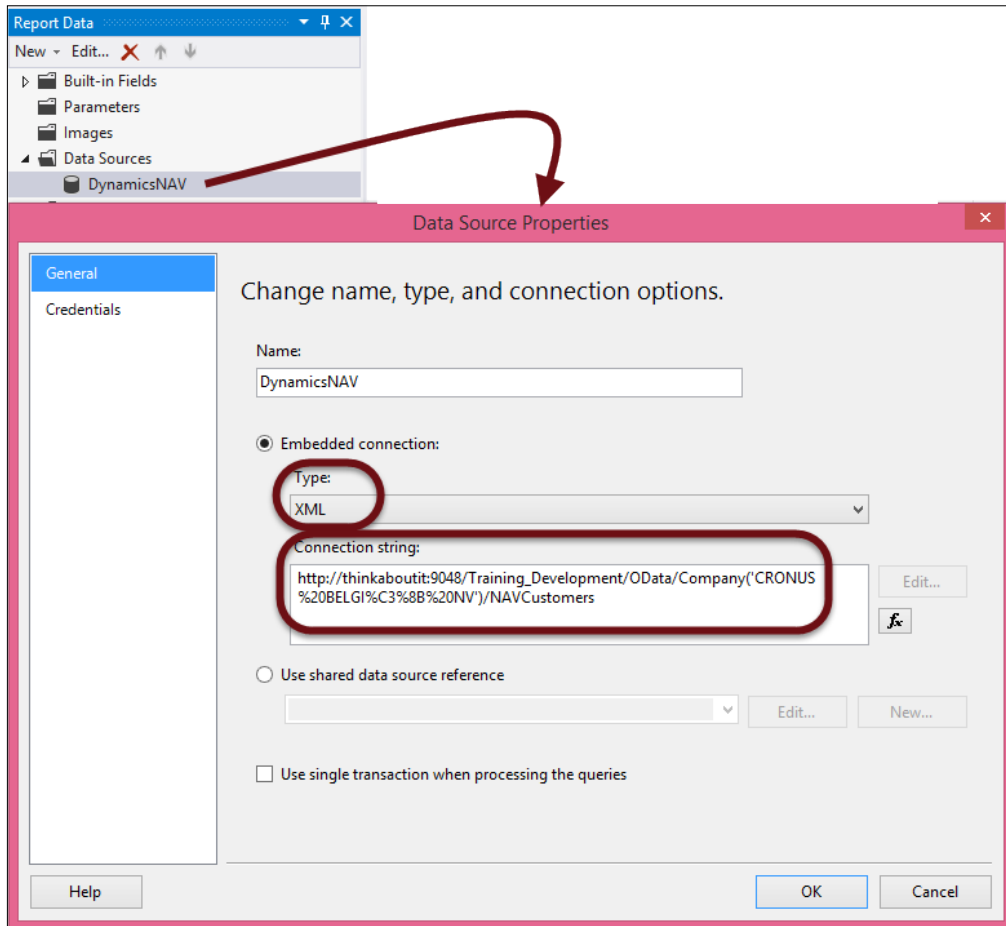
I will create the same report again, a list of customers, but this time based on an OData web service. I will first create a new **Page 90000 Customers**, in which I will add the field from the `Customer` table that I want to use in my report. Then, I will publish this page as a web service:



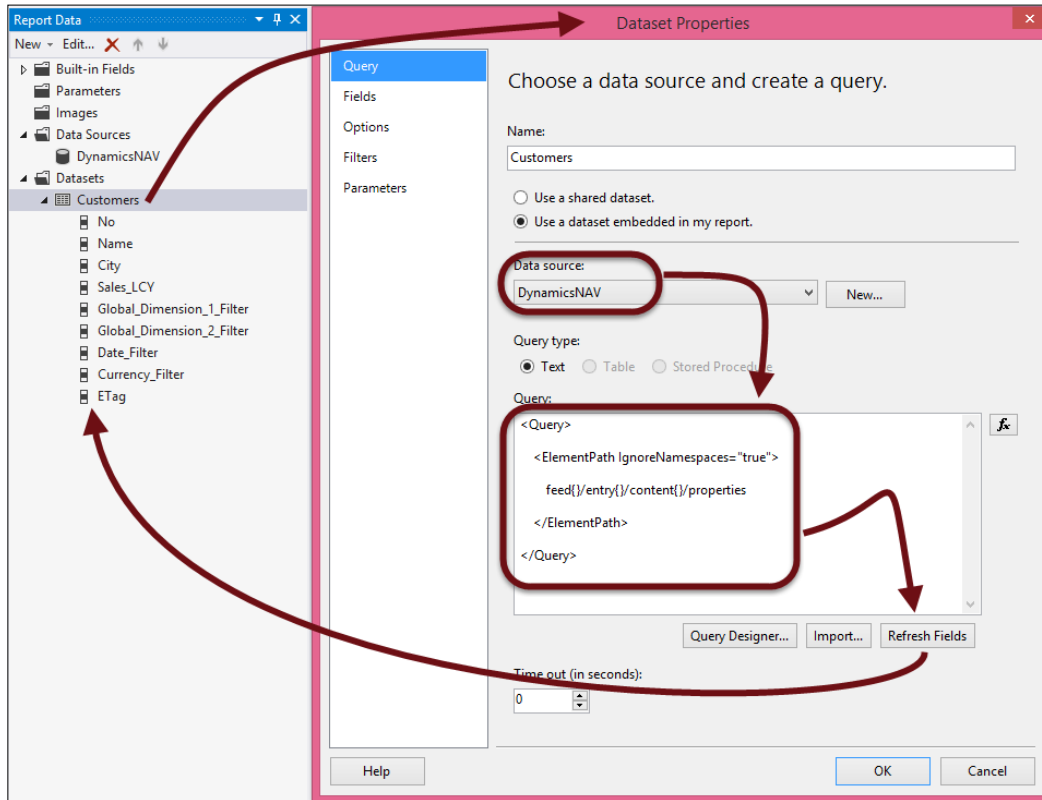
The screenshot shows the Dynamics NAV interface. The top window is 'Page 90000 Customers - Page Designer'. It displays a table with columns: E.. Type, SubType, SourceExpr, Name, and Caption. A red circle with the number '1' highlights the 'Field' row under the 'Repeater' sub-type, where the 'SourceExpr' is set to '"Sales (LCY)"'. Below this, the 'Web Services' list is visible. A red circle with the number '2' highlights the 'Page' row for '90000 Customers'. The 'Web Services' list has the following data:

Object Type	Object ID	Object Name	Service Name	Publ...	OData URL
Page	21	Customer Card	Customers	<input checked="" type="checkbox"/>	http://thinkaboutit:9048/Training_Development/OData/Company('CRONUS%20BELGI%C3%8B%20NV')/Customers
Query	50000	CustomerAnalysis	CustomerAnalysis	<input checked="" type="checkbox"/>	http://thinkaboutit:9048/Training_Development/OData/Company('CRONUS%20BELGI%C3%8B%20NV')/CustomerAnalysis
Page	90000	Customers	NAVCustomers	<input checked="" type="checkbox"/>	http://thinkaboutit:9048/Training_Development/OData/Company('CRONUS%20BELGI%C3%8B%20NV')/NAVCustomers

Then, I will create a new report in SSDT. I will use a new data source in this report. I'm not going to fetch the data directly from SQL Server, but from the OData web service. In order to do that, the data source needs to be of the type XML:




The connection string is the URL of the OData web service from Dynamics NAV. I will then create a dataset named `Customers`, as follows:



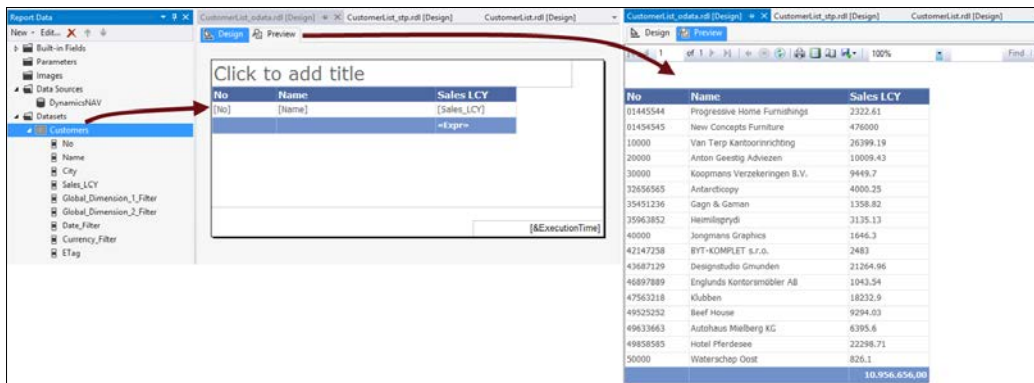
In summary, as a Query to an OData web service you can always use:

```
<Query>
  <ElementPath IgnoreNamespaces="true">
    feed{/entry{/content{/properties
  </ElementPath>
</Query>
```




 There are other ways to query a web service, or to apply filters and sorting. More information about querying XML data sources is available here: [https://technet.microsoft.com/en-us/library/aa964129\(v=sql.90\).aspx](https://technet.microsoft.com/en-us/library/aa964129(v=sql.90).aspx)

When you click the **Refresh Fields** button, the fields of the dataset are refreshed and appear on the left.

Now, all you need to do is build the layout of the report, and run it:



As you can see, this is very simple and straightforward. The advantage is that the business logic is inside Dynamics NAV. In our report, we only call the web service, and we don't store complex queries. Instead of a Page object, you can use a Query object in Dynamics NAV, so you can combine and aggregate information from multiple tables.


 An example of this report is available in the object: CustomerList_odata.rdl

The next step

So far, we have created reports that query the database directly. This is not always recommended, or possible. The most important reason for this is performance. If you have a lot of users running reports, it creates a significant load on the database server, which might even result in loss of performance for the users working in Dynamics NAV. There are several techniques to avoid this.

Caching

You can enable report caching mechanisms using the Report Manager application. When a user runs a report, the information is retrieved from the source database. This information can then be saved in the Reporting Services database so that the next time a user runs the same report, the information is retrieved from the cache instead of the source database. You then determine how long the information remains in the cache.

When you select a report in the Report Manager, you can examine and manage its caching properties, as follows:

SQL Server Reporting Services
CustomerList

Properties
Data Sources
Subscriptions
Processing Options
Cache Refresh Options
Report History
Snapshot Options
Security

Always run this report with the most recent data
 Do not cache temporary copies of this report
 Cache a temporary copy of the report. Expire copy of report after a number of minutes:
 Cache a temporary copy of the report. Expire copy of report on the following schedule:
 Report-specific schedule
 At 8:00 every Mon of every week, starting 27/06/2015
 Shared schedule
 Render this report from a report snapshot
 Use the following schedule to create report snapshots:
 Report-specific schedule
 At 8:00 every Mon of every week, starting 27/06/2015
 Shared schedule
 Create a report snapshot when you click the Apply button on this page

Report Timeout
 Use the system default setting
 Do not timeout report
 Limit report processing to the following number of seconds:

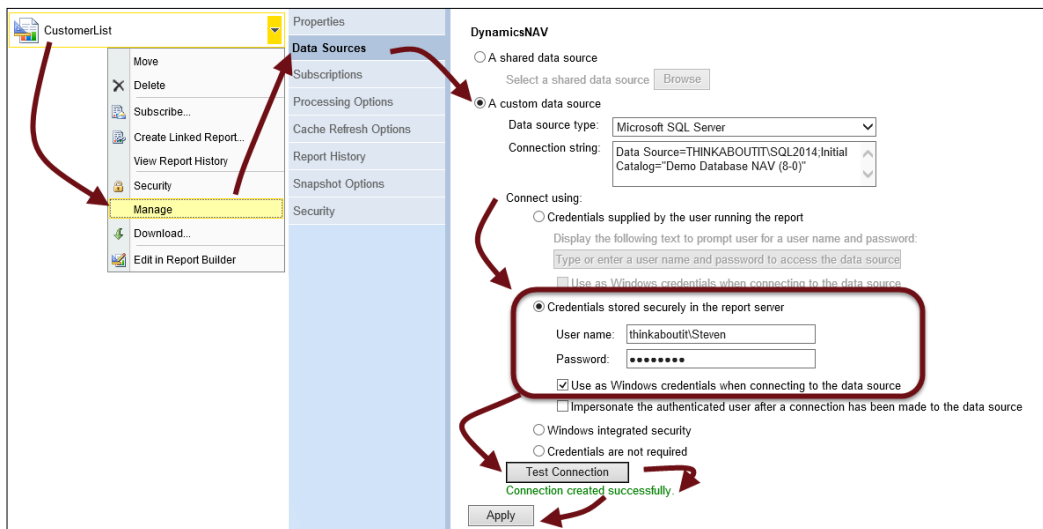
You then have a copy of the results saved in the report server database when someone runs the report. The next time the report is executed the data is fetched from the cache instead of the live database, depending on the expiration options that you select.

You can use are also mechanisms called **snapshots**. A snapshot is a cached version of a report that you create behind the scenes with a schedule. For long running reports, you can generate the snapshots outside of business hours.

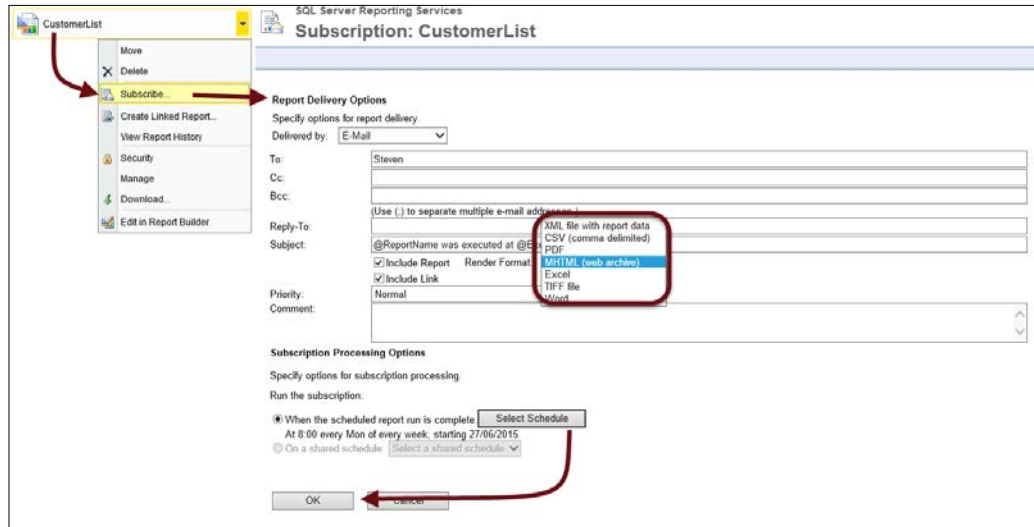
Subscribing or scheduling

Once you have reports published and available on the report server, you can run and manage them via the Report Manager. Report Manager also allows you to subscribe to a report. You can create a subscription to a report, which means that you can have the report generated automatically in a format that you want, and with a schedule that you create or select.

This is only possible if the credentials that the report uses are stored in the report itself. You can do this as follows:



To create a subscription, look at the following example:



You can have the report sent via email or have it created in a shared folder on the server. Another option is to create a subscription based on information in a subscription table. The report scheduler fetches the people that want to receive the report from the database. You then only have to create one subscription in Report Manager and manage it in the subscription table.

Summary

In this chapter, I gave a brief introduction to the use and of Reporting Services. One advantage is that it is a web-based reporting solution that does not require the presence of Dynamics NAV on the local machine. You can harvest and present information from multiple data sources, databases, companies, and so on. Reporting services is completely free: no license cost is involved.

Before you spend a big budget on third-party BI or reporting tools, I recommend a look at what Reporting Services can offer you, out of the box.

In the next chapter, I will present different ways to create charts in the Dynamics NAV application.

11

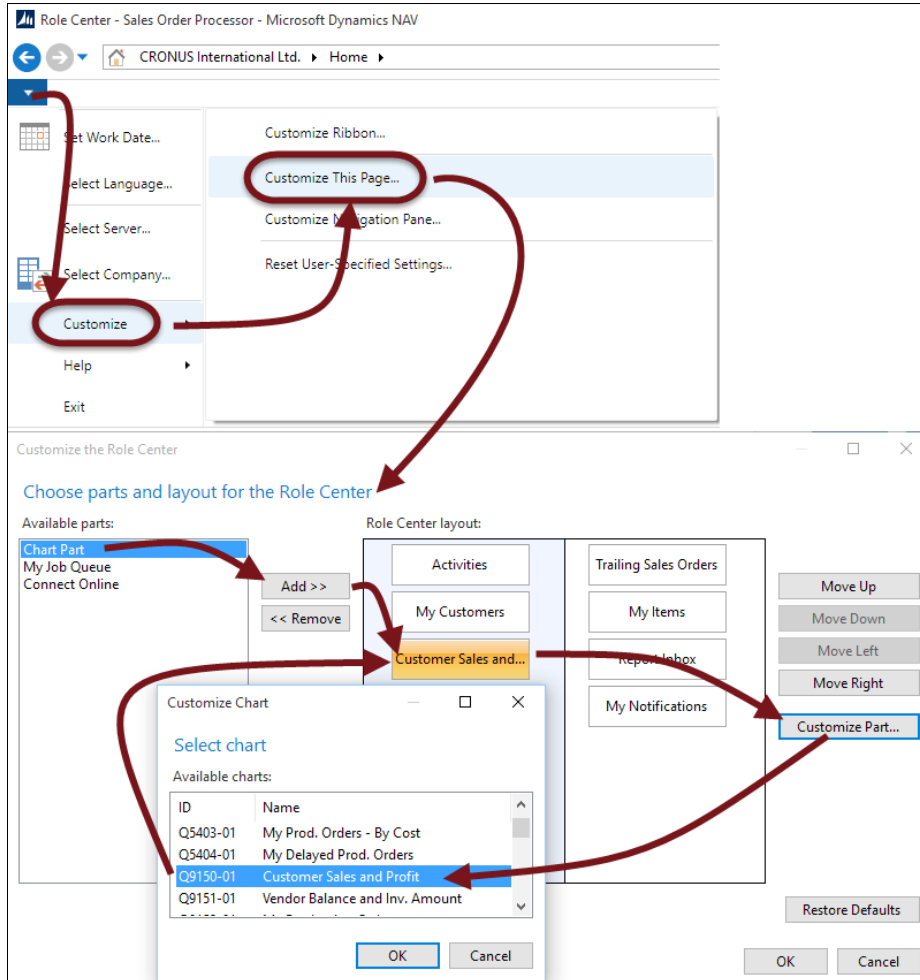
Charts in Dynamics NAV

In this chapter, I will explain and demonstrate the ways in which you can implement charts in Dynamics NAV. You can use the generic chart designer to create your own, simple charts and add them to any list page or role center. Then there are the business charts that are managed via the C/AL code so you can make them more complex in the way that you present information and manage how this information is displayed at runtime. Last but not least, I will demonstrate how you can create key performance indicators on activity pages in a role center. This is an ideal way to see how your business is performing, since it is the first page you see when you run the application.

The generic chart designer

Let's start by having a look at the **Chart Part**. You can go to the Dynamics NAV application menu at launch and select the **Customize** button and **Customize This Page...** on the **Role Center** page.

You can add a **Chart Part** to the **Role Center** page as follows:

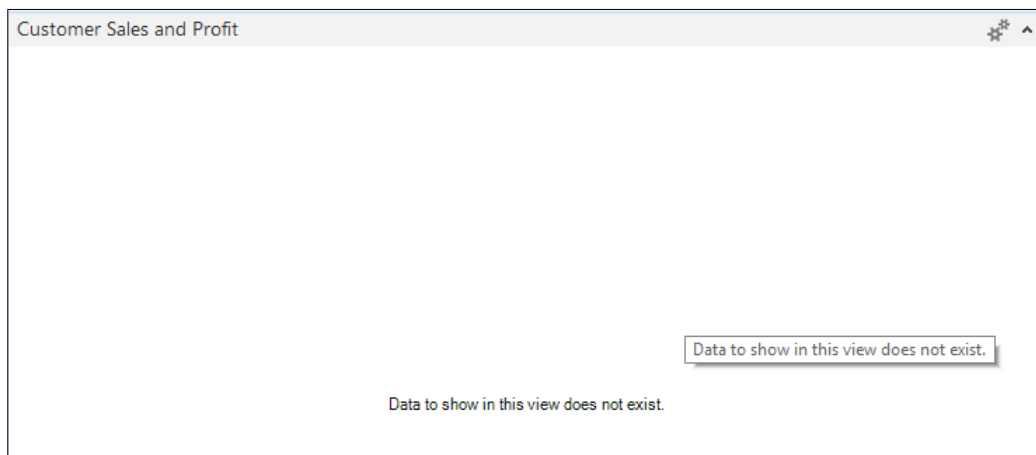


You can select one of the predefined charts from the chart table to display when you run the page with the **Customize** button on the chart part, as shown here.

To summarize, use the following steps:

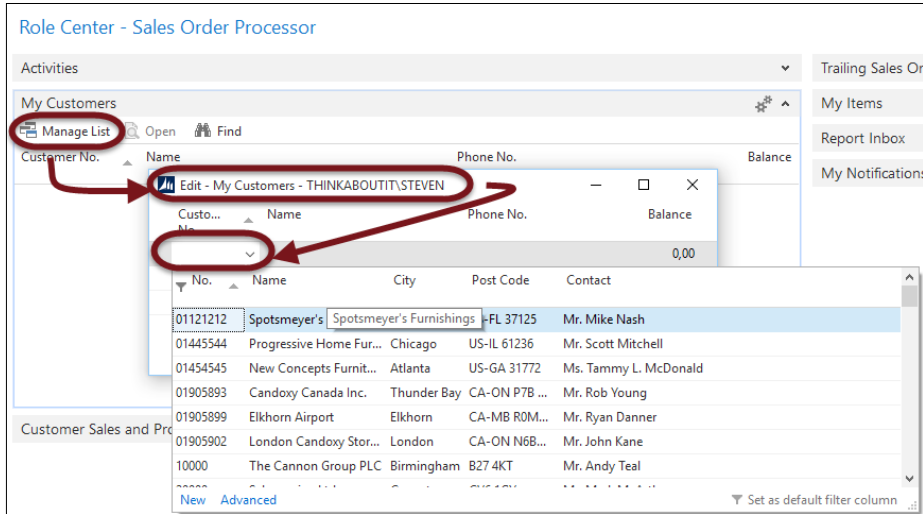
1. Click **Chart Part** on the left.
2. Click **Add >>**, which adds a blank chart to the **Role Center layout** in the middle.
3. Click **Customize Part....**
4. Select the generic chart from the list.

Then, when you select the **OK** button and go back to your role center, you will see the following:



As you can see, the chart is empty. This is because I selected the **Customer Sales and Profit** chart, which is based on the `My Customer` table. So, you need to add customers to your **My Customers** page, which will then be displayed in the chart, where you can see their sales and profit figures.

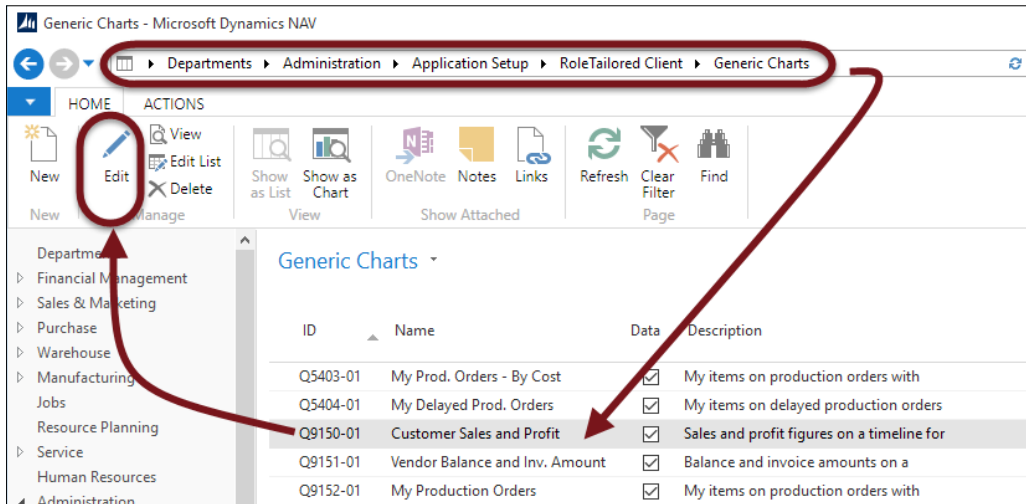
To do this, add customers to **My Customers**, as follows:



Then refresh the **Role Center** page and the chart will be updated with the customers you added to **My Customers**.

How is this possible? Well, let's have a look at the chart itself, and backwards engineer it to see its building blocks.

Go to the **Generic Charts** page, and the applications search box:



When you select the chart **Q9150-01** and then **Edit**, the following window opens:

Q9150-01

General

ID: Name:

Data Source

Source Type: Source Name:
 Source ID: Filters:

Measures (Y-Axis)

	Data Column	Aggregation	Graph Type	Data Point Label	Y-Axis Title:
Required Measure:	Sum_Sales_LCY	Sum	Column	Sum Sales LCY	<input type="text"/>
Optional Measure:	Sum_Profit_L...	Sum	Column	Sum Profit LCY	<input checked="" type="checkbox"/>
Optional Measure:		None	Column		
Optional Measure:		None	Column		
Optional Measure:		None	Column		
Optional Measure:		None	Column		

Dimensions (X- and Z-Axes)

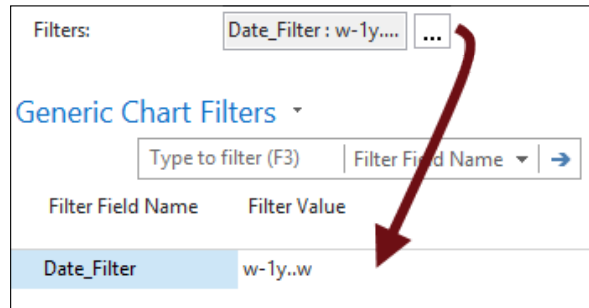
Chart Description

Preview Part

OK

Give the chart an **ID** and a **Name** in the **General** tab. The IDs that Microsoft has chosen for the default charts start with an **Q** or a **T** and then a number. This is because, when you create a chart, you can use either a **Query** or a **Table** as the data source. The number references the number of the **Query** or **Table** so that you can see the data source in the name. The advantage of using a **Query** is, of course, that you can combine and aggregate values from different tables, so that you can display them in the chart.

Select the **Source Type** (**Table** or **Query**) in the **Data Source** tab. Here you can also define a filter. In this example, when you open **Filters**, you will see the following filter:



As you can see, this is a dynamic filter, passed to the **Date_Filter** field, used in the FlowFields calculation displayed in the chart. This is a very clever way to filter a chart. Instead of fetching all of the values from the underlying query, only the relevant information from the last year is fetched.

You have to select at least one measure in the **Measures** tab. A measure is a numeric field from your data source, displayed in the chart. You can then apply an aggregation to the measure and select a graph type.


You then select the fields to use on the X and Z axis in the **Dimensions** tab. In this example, **Customer No.** is used, so the chart displays the sum of **Sales** and **Profit** by customer. If you only select one measure, you can also select a field for the Z axis and create a three dimensional chart. But this is not possible when you have more than one measure.

You can type in a description to explain what the chart is all about in the **Chart Description** tab. You can translate this description into other languages using the assist edit button next to it.


At the bottom, in the **Preview Part** tab, you can get an idea of the look and feel of the chart you are creating. This is a preview that is not based on any actual data. It is merely there to see how the information will be visualized.

There are **Export** and **Import** buttons in the ribbon. You can use them to export a chart definition to an XML file, so that you can import it into another database.

Once you have created a chart with the generic chart designer, it is saved in the chart table in your database. You can then display it on a page using a **Chart Page**. Typically, role center pages are used for this, although you could do it on any page or page part.


 When a developer adds a chart part to a page at design time, a user can add more of them to the page via the **Customize This Page** feature at runtime.

Now you know how to use the built-in generic chart designer to create your own charts in the application, without the need of a developer. You can base your charts on tables or queries.


 I recommend query objects for charts because they offer better performance, and because they can combine information from multiple tables. Furthermore, a query has a `TOPNUMBEROFROWS` property, so you can use them to create `TOPX` charts.

Text management

There are some shortcuts that you can use when you want to apply a filter in Dynamics NAV. For example, the `%MYCUSTOMERS` string is replaced with the list of IDs of the customers you add to the `My Customers` table, as you can see in the following example:

Customers ▾

Show results:

Where No. ▾ is `%MYCUSTOMERS` ▾

+ Add Filter

Customers ▾

Show results:

Where No. ▾ is `01121212|01445544|01454545|01905893|01905899|01905902|10000|20000|21233572`

+ Add Filter

No.	Name	Responsi... Center	Location Code	Phone No.	Contact
01121212	Spotsmeyer's Furnishings		GEEL		Mr. Mike Nash
01445544	Progressive Home Furnishi...		GEEL		Mr. Scott Mitchell
01454545	New Concepts Furniture		GEEL		Ms. Tammy L. McDonald
01905893	Candoxy Canada Inc.		GEEL		Mr. Rob Young
01905899	Elkhorn Airport		GEEL		Mr. Ryan Danner
01905902	London Candoxy Storage C...		GEEL		Mr. John Kane
10000	Van Terp Kantoorinrichting	DILBEEK	BLAUW		Dhr. Kevin Verboort
20000	Anton Geestig Adviezen				Dhr. Luc van Vugt
21233572	Somadis		GEEL		M. Syed ABBAS

In this example, I used the customer list page and applied a filter, but you can also use this as a filter in a chart. This is also possible for other entities. There's a **Codeunit 41 TextManagement**, in which there's a function named `MakeTextFilter`. The following snippet shows the code of this function:

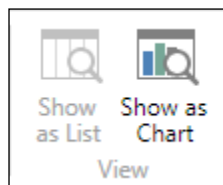
```
Position := 1;
Length := STRLEN(TextFilterText);
ReadCharacter(' ',TextFilterText,Position,Length);
IF FindText(PartOfText,TextFilterText,Position,Length) THEN
CASE PartOfText OF
    COPYSTR('ME',1,STRLEN(PartOfText)),COPYSTR(MeText,1,
    STRLEN(PartOfText)):
        BEGIN
            Position := Position + STRLEN(PartOfText);
            TextFilterText := USERID;
        END;
    COPYSTR('USER',1,STRLEN(PartOfText)),COPYSTR(UserText,1,
    STRLEN(PartOfText)):
        BEGIN
            Position := Position + STRLEN(PartOfText);
            TextFilterText := USERID;
        END;
    COPYSTR('COMPANY',1,STRLEN(PartOfText)),COPYSTR(CompanyText,1,
    STRLEN(PartOfText)):
        BEGIN
            Position := Position + STRLEN(PartOfText);
            TextFilterText := COMPANYNAME;
        END;
    COPYSTR('MYCUSTOMERS',1,STRLEN(PartOfText)),
    COPYSTR(MyCustomersText,1,STRLEN(PartOfText)):
        BEGIN
            Position := Position + STRLEN(PartOfText);
            GetMyFilterText(TextFilterText,DATABASE:"My Customer");
        END;
    COPYSTR('MYITEMS',1,STRLEN(PartOfText)),
    COPYSTR(MyItemsText,1,STRLEN(PartOfText)):
        BEGIN
            Position := Position + STRLEN(PartOfText);
            GetMyFilterText(TextFilterText,DATABASE:"My Item");
        END;
    COPYSTR('MYVENDORS',1,STRLEN(PartOfText)),
    COPYSTR(MyVendorsText,1,STRLEN(PartOfText)):
        BEGIN
            Position := Position + STRLEN(PartOfText);
            GetMyFilterText(TextFilterText,DATABASE:"My Vendor");
        END;
ELSE
    EXIT(Position);
```

```
END;  
EXIT(0);
```

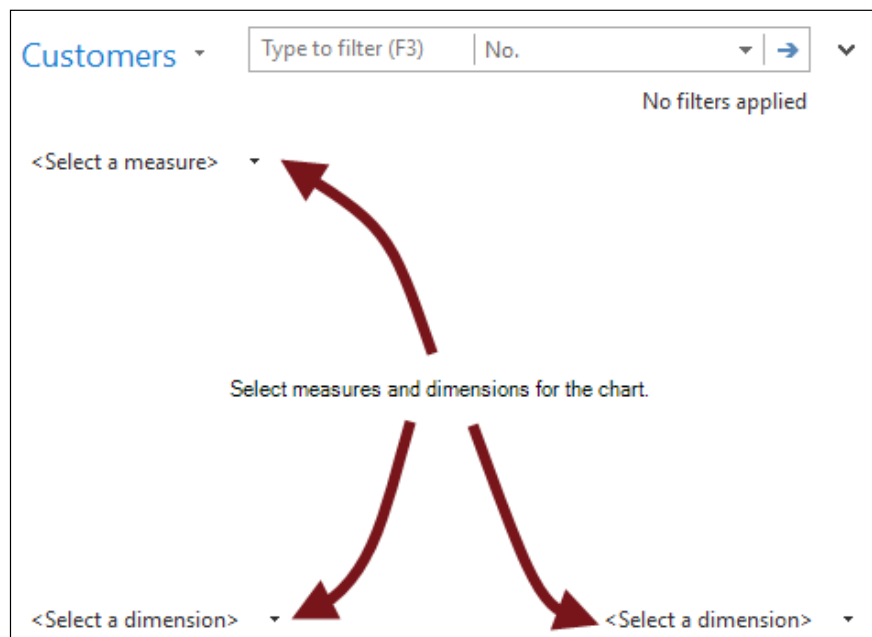
As you can see, it contains code for how filters like %mycustomers, %myitems, %myvendors, %company, %user, and %me are converted into filters with information from the corresponding tables. You can also customize this function to include other types of filter shortcuts, for example, %ImportantCustomers, and so on. This is a very powerful feature, often overlooked.

Show any list as a chart

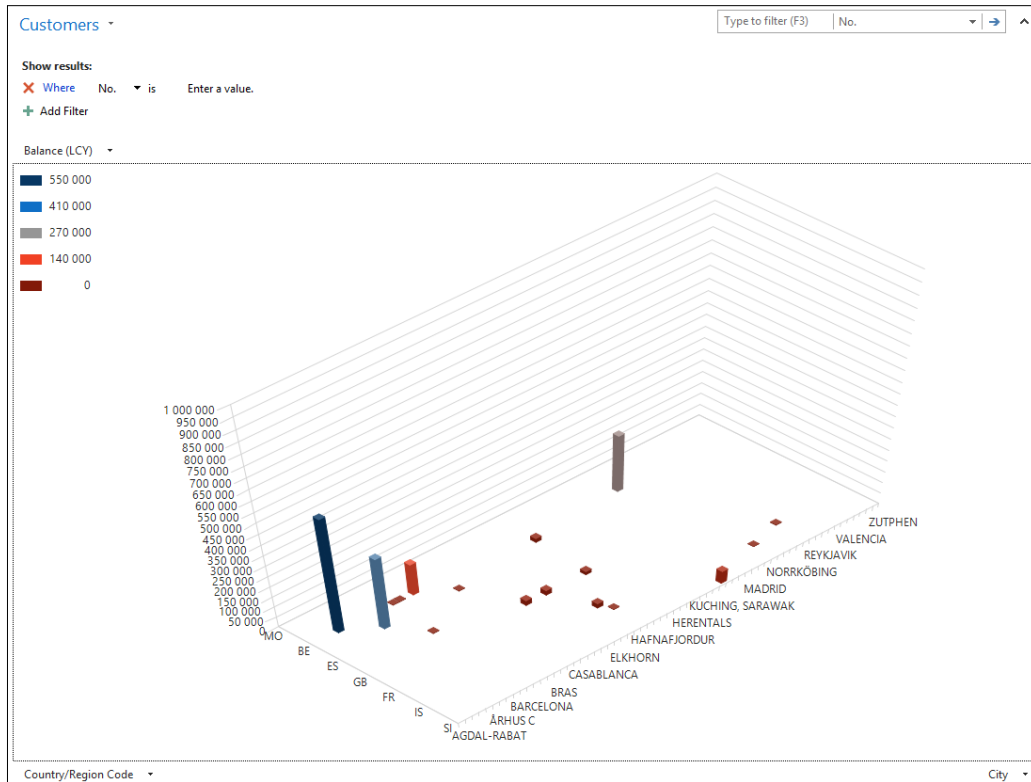
When you are on a list page in the application, such as the **Customer List** or the **Item List**, there's a button in the ribbon, **Show as Chart**, that looks like this:



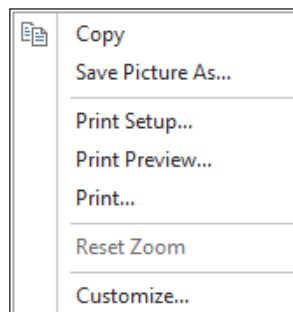
When you select it, a blank canvas is shown, in which you can design a quick chart:



You then need to select a measure, on the left top, and a dimension, at the right bottom, for the chart to display. Optionally, you can also select a second dimension, at the left bottom. Such a chart would look like this:



The measure and dimension fields are fields from the source table of the list page you are on. You can use your mouse to zoom in and out or to rotate the chart. Right-click on it and you get the following menu:



This allows you to print or save the chart as an image. When you select the **Customize...** option, the chart is opened in the generic chart designer, where you can customize it further, to apply filters or select another visualization. This is a quick way to create a simple chart. Once you have created a chart in this way, the system will remember it and every time you select the **Show as Chart** option again, it will show the chart that you have created.

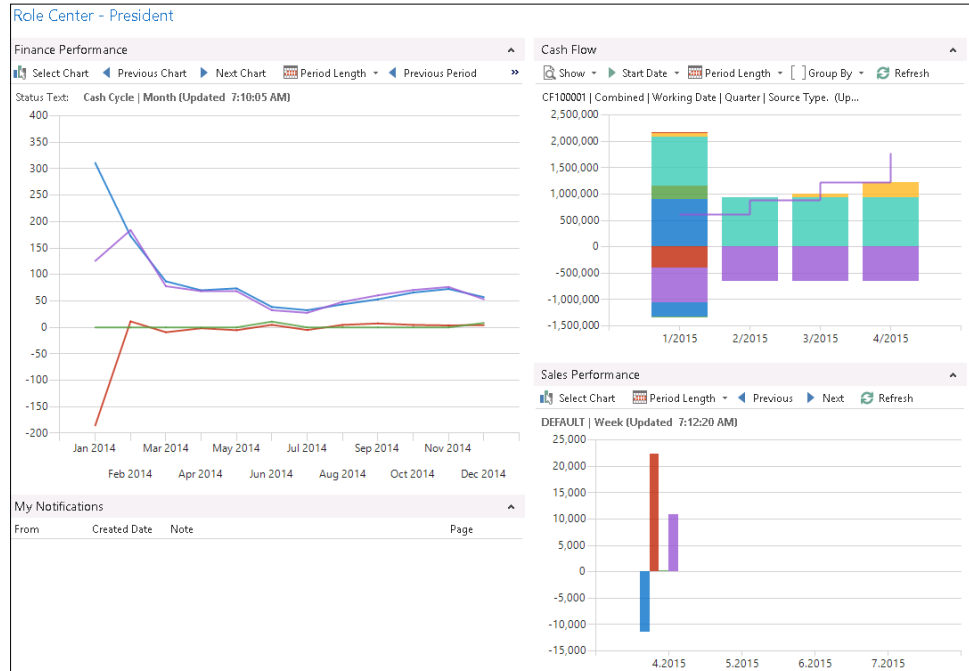
Business charts

Charts are built directly on top of data from either a table or a query object. If you want to create dynamic business charts with flexible options and different visualizations based on business logic, then you can use what is called the business chart.



Another advantage of the business chart is that it works with both a web and tablet client. It uses a JavaScript control in the web client, and a .Net control to render in the Windows client. There are only minor differences in how it is displayed in the different clients, such as the line and chart heights and the way that the legend is presented.


An example of a business chart is shown in the following diagram:



You can see these charts if you log on to Dynamics NAV with the profile of the **President**.

To do this, follow these steps:

1. Go to **User Personalisation**, **CRONUS International Ltd. / Departments/Administration/Application Setup/RoleTailored Client/User Personalization**, and click **New**.
2. Select your user **ID** in the **User Personalization Card**, and enter **PRESIDENT - SMALL BUSINESS** in the **Profile ID** field.
3. Restart the application.

 More information about *Roles* and *Profiles* is available here:
[https://msdn.microsoft.com/en-us/library/hh174139\(v=nav.80\).aspx](https://msdn.microsoft.com/en-us/library/hh174139(v=nav.80).aspx)

As you can see, these charts have better visualizations and more options to customize on the fly.

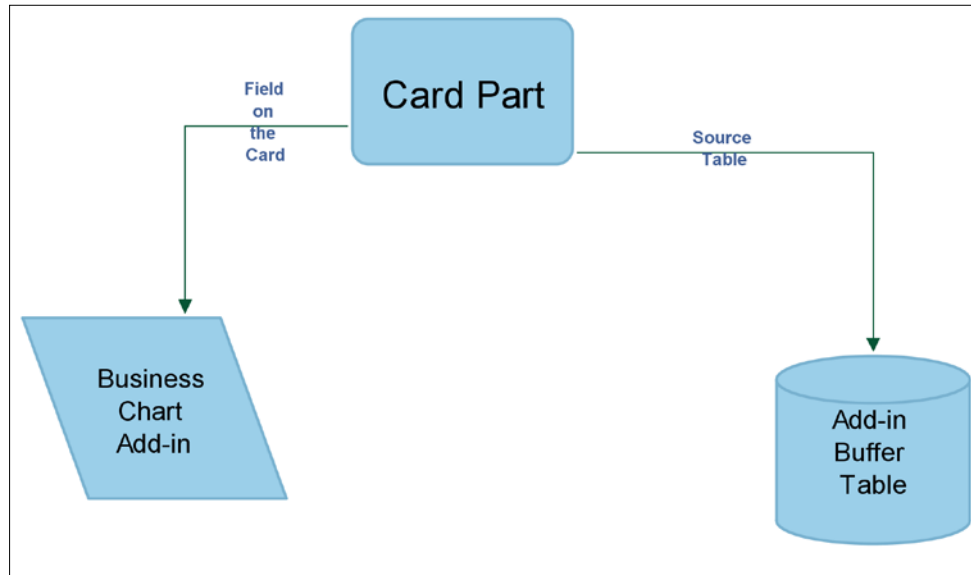
 When you log in as the president for the first time, the **Cash Flow** chart and the **Sales Performance** chart are both blank. You have to select measures to show something in there.

Let me guide you through the process of how to get started in creating your own business chart.

Creating a business chart

Business charts are based upon a .NET add-on. This add-on is added to a page, in the content area. The page itself could be a role center or a factbox page. The data is shown from a table: `Business Chart Buffer (485)`. This is a buffer or temporary table that you need to populate with data, which is then displayed in the chart add-on.

In summary, the following diagram shows you an overview of the objects:



The explanation of the preceding diagram is as follows:

- **Card:**
 - Implements events from the add-on
 - Implements actions
- **Business Chart Add-in:**
 - Events:
 - AddInReady: This indicates if the page is done rendering
 - DataPointClicked: This is a single-click on an element on the chart
 - DataPointDoubleClicked: This is a double-click on an element of the chart
 - Functions:
 - Update: This is used to update or refresh the chart

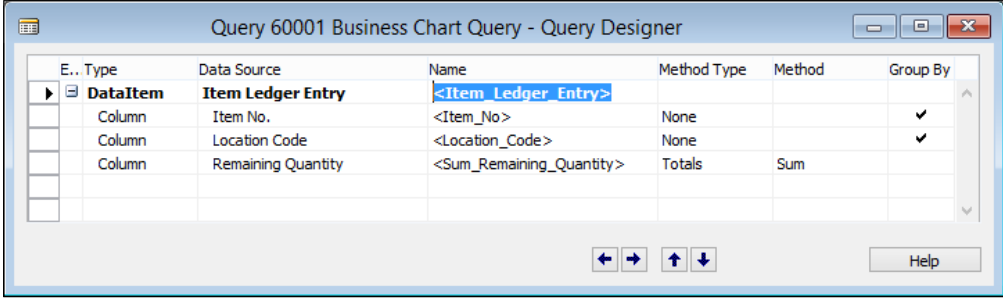
- **Add-in Buffer Table**
 - This is responsible for handling the data from the add-in
 - This handles captions and multilanguage support
 - This stores chart values and conversion from .NET to C/AL
 - This supports drilldown to handle click events
- Contains helper methods and other functionality regarding data

To get started with your first business chart, you need to create three objects:



- A page
- A codeunit
- A query

 The query object is optional and I'm going to use it to fetch the data in a more optimized fashion. 

I will start by creating the following query object:



E..	Type	Data Source	Name	Method Type	Method	Group By
▶	DataItem	Item Ledger Entry	<Item Ledger Entry>			
	Column	Item No.	<Item_No>	None		✓
	Column	Location Code	<Location_Code>	None		✓
	Column	Remaining Quantity	<Sum_Remaining_Quantity>	Totals	Sum	

 Instead of creating the query, you can import the object: Query Business Chart. 

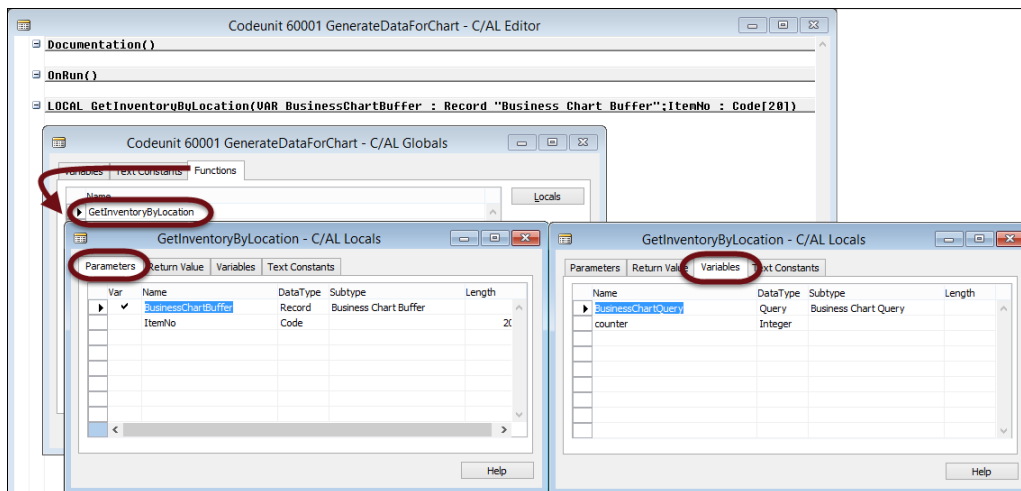
In this query, I'm fetching the items and location codes from the Item Ledger Entry table and the sum of the remaining quantity. The query then shows what's in stock for every item, by location.

Now I will create a codeunit that will run the query and store the results in the Business Chart Buffer table.



You can import the object: CodeUnit GenerateDataForChart instead of creating the codeunit.

I will create a GetInventoryByLocation function in the codeunit with the following parameters and variables:



Write the following code in the GetInventoryByLocation function:

```

WITH BusinessChartBuffer DO
BEGIN
    Initialize;
    AddMeasure('Inventory', 1, "Data Type"::Decimal, "Chart
    Type"::Column);
    SetXAxis('Location', "Data Type"::String);
    BusinessChartQuery.SETRANGE(Item_No, ItemNo);

    CLEAR(counter);
    BusinessChartQuery.OPEN;
    WHILE BusinessChartQuery.READ DO
    BEGIN
        counter += 1;
    
```

```
        AddColumn(BusinessChartQuery.Location_Code);
        SetValue('Inventory',counter -
        1,BusinessChartQuery.Sum_Remaining_Quantity);
    END;
    BusinessChartQuery.CLOSE;
END;
```

The `Initialize` function initializes the .NET add-on. You then define the measure that needs to be displayed on the Y axis (name, data type and chart type).

The available chart types are:

- Point
- Bubble
- Line
- Step Line
- Column
- Stacked Column
- Stacked Column 100
- Area
- Stacked Area
- Stacked Area 100
- Pie
- Doughnut
- Range
- Radar
- Funnel

The `SetXAxis` function defines the measure for the X axis (name and data type).

You then apply the item number from the `codeunit` parameter as a filter for the query object, run the query, and loop over the results. The location code is used for the X axis values and the `SumOfRemainingQuantity` is used for the Y axis values.

Be sure that the name you provide in the `SetValue` function matches what you defined as the name in the `AddMeasure` function. The name is required because the chart add-on allows you to use and display multiple measures.



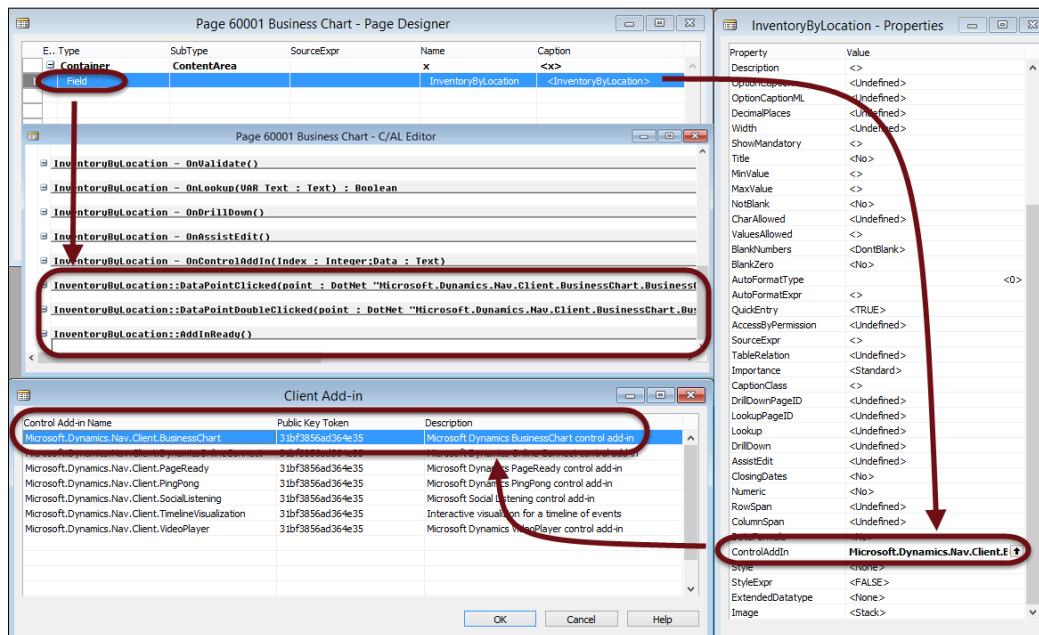
More information about these objects is available at [https://msdn.microsoft.com/en-us/library/hh167009\(v=nav.80\).aspx](https://msdn.microsoft.com/en-us/library/hh167009(v=nav.80).aspx) and [https://msdn.microsoft.com/en-us/library/hh169415\(v=nav.80\).aspx](https://msdn.microsoft.com/en-us/library/hh169415(v=nav.80).aspx).

Now that you have created the query and codeunit, it's time to create the page object that will display the chart.



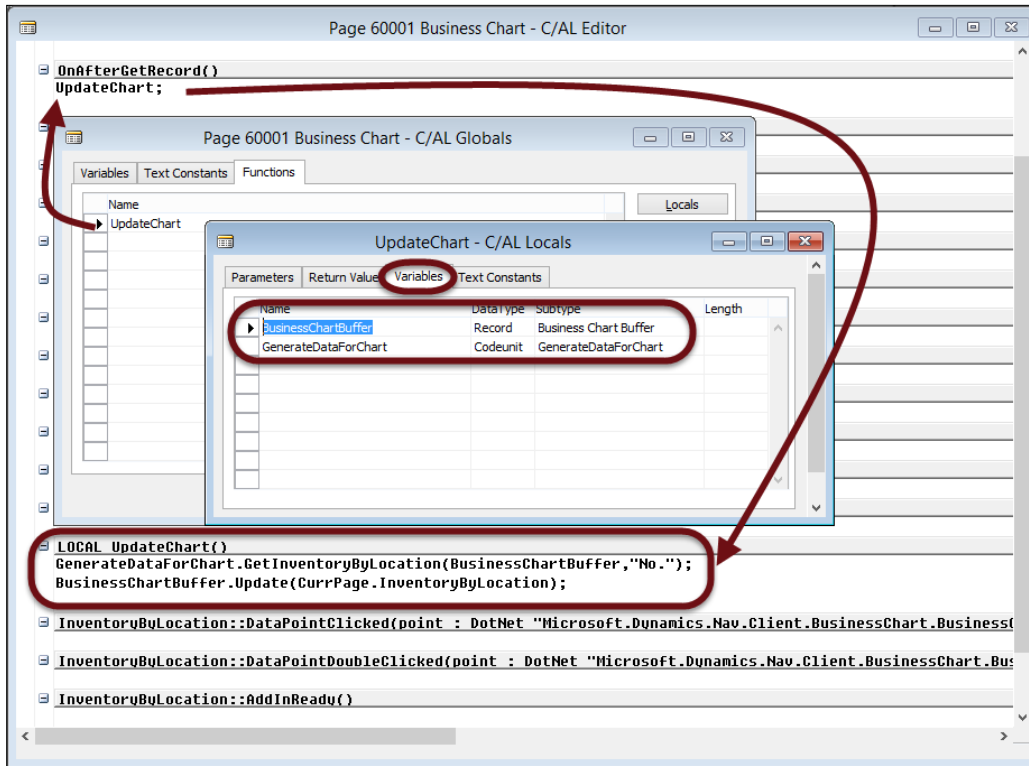
You can import the object: Page Business Chart, instead of creating the page.

Create the new page as a card part and then add a field to the content area container. Next, go to **ControlAddIn** in the properties of the field and select the business chart control add-on from the list, as follows:



Then, when you select *F9* to see the triggers, you will see that your field has three new triggers, inherited from the business chart .NET add-on.

Create a new function in the page, named `UpdateChart`, with the following variables:

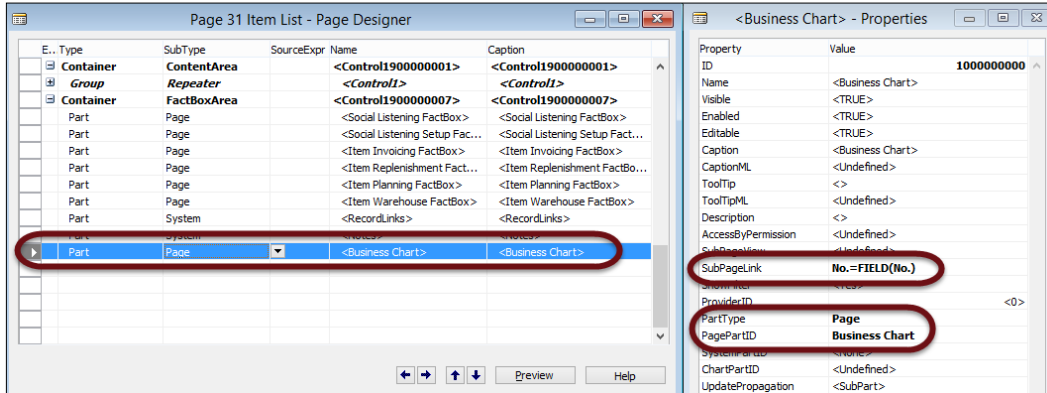


Then, call this function in the `OnAfterGetRecord()` trigger of the page. You need to pass the `Item Number` to the buffer table in the `UpdateChart` function. Then, the `Update` function from the buffer table will update your chart.

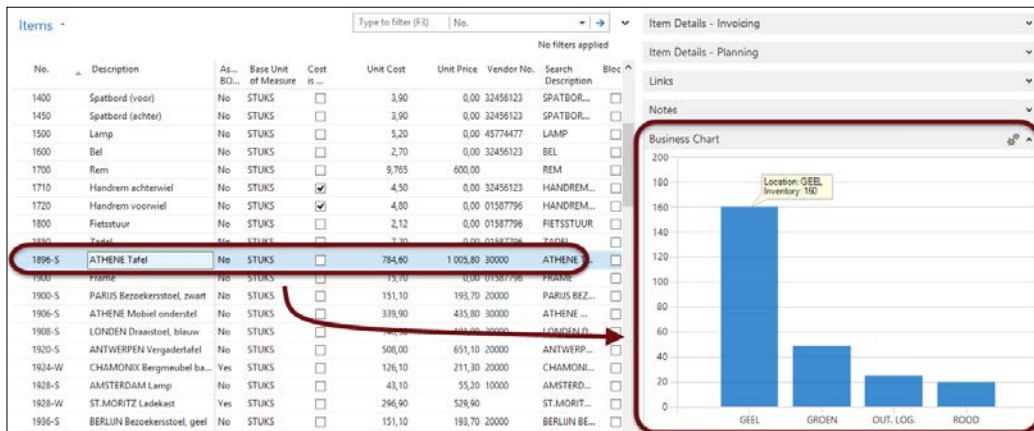


When you add the field in the page, you need to give the field a name so that you can use its name with the `CurrPage` variable to pass it to the `Update` function.

Our business chart page is now ready and we will now add it as a fact box in the item list page:

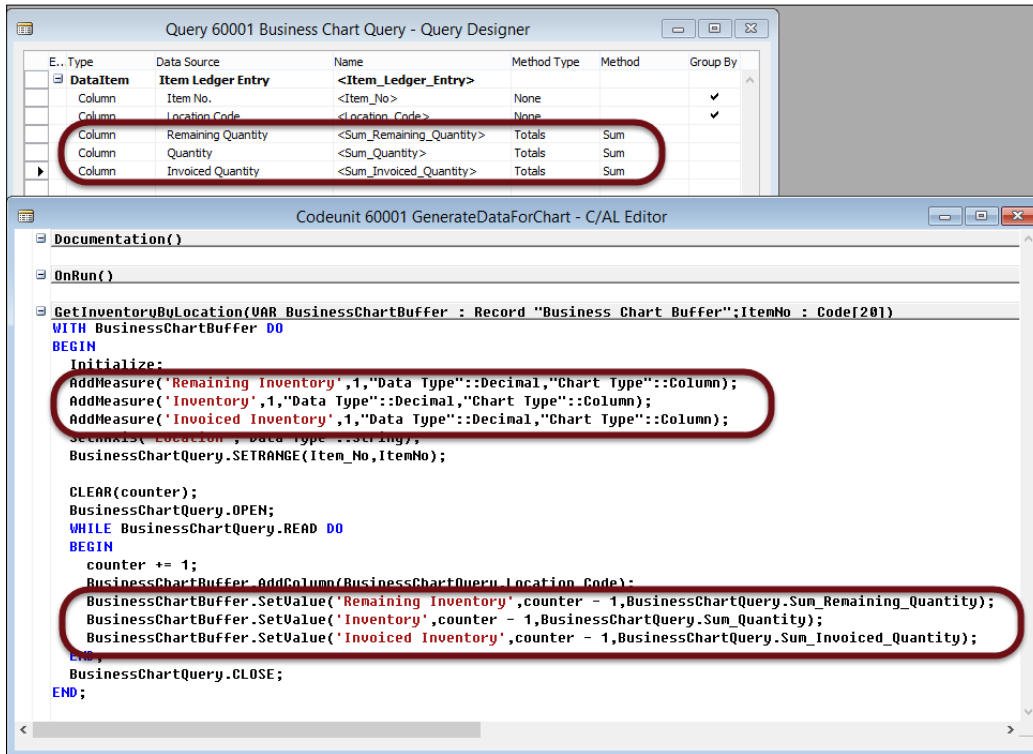


When you save and run the item list, the business chart you have just created is shown as a fact box and will be updated when you select an item in the list:



[ An example of the objects required for this business chart is available in the object: Business Chart]

All you need to do to extend this chart with extra columns is to add the columns in the query and code unit objects, as shown in the following example:



Then, when you update the chart (F5), it shows the following results:



Drill down your business chart

Information is displayed in the business chart that you created as inventory by location, for a specific item in the list. Now you will add drill-down functionality to the chart so that when you click on it, you will see the underlying ledger entries.

You need to add the following code in the codeunit that you created for the chart to do this. First, in the `GetInventoryByLocation` function, add the code that is marked:

```

GetInventoryByLocation(VAR BusinessChartBuffer : Record "Business Chart Buffer";ItemNo : Code[20])
WITH BusinessChartBuffer DO
BEGIN
    BusinessChartBuffer.Initialize;
    BusinessChartBuffer.AddMeasure('Remaining Inventory',1,"Data Type":=Decimal,"Chart Type":=Column);
    AddMeasure('Inventory',1,"Data Type":=Decimal,"Chart Type":=Column);
    AddMeasure('Invoiced Inventory',1,"Data Type":=Decimal,"Chart Type":=Column);
    SetXAxis('Location',"Data Type":=String);
    BusinessChartQuery.SETRANGE(Item_No,ItemNo);
    CLEAR(counter);
    /***
    ItemNumber := ItemNo;
    CLEAR(counter2);
    /***
    BusinessChartQuery.OPEN;
    WHILE BusinessChartQuery.READ DO
    BEGIN
        counter += 1;
        /***
        counter2 += 1;
        /***
        BusinessChartBuffer.AddColumn(BusinessChartQuery.Location_Code);
        BusinessChartBuffer.SetValue('Remaining Inventory',counter - 1,BusinessChartQuery.Sum_Remaining_Quantity);
        BusinessChartBuffer.SetValue('Inventory',counter - 1,BusinessChartQuery.Sum_Quantity);
        BusinessChartBuffer.SetValue('Invoiced Inventory',counter - 1,BusinessChartQuery.Sum_Invoiced_Quantity);
        /***
        LocationCode[counter2] := BusinessChartQuery.Location_Code;
        /***
    END;
    BusinessChartQuery.CLOSE;
END;

```

Then, in the corresponding triggers, add the following code:

```

OnDataPointClicked(VAR BusinessChartBuffer : Record "Business Chart Buffer")
    DrillDown(LocationCode[BusinessChartBuffer."Drill-Down X Index"+1]);

DrillDown(LocationCode : Code[20])
    ItemLedgerEntry.SETRANGE("Item No.",ItemNumber);
    ItemLedgerEntry.SETRANGE("Location Code",LocationCode);
    PAGE.RUN(PAGE:="Item Ledger Entries",ItemLedgerEntry);

```

Then, in the page, add the following code:

```

Page 60001 Business Chart - C/AL Editor
LOCAL UpdateChart()
IF NOT IsChartAddInReady THEN
    EXIT;

GenerateDataForChart.GetInventoryByLocation(BusinessChartBuffer,"No.");
BusinessChartBuffer.Update(CurrPage.InventoryByLocation);

InventoryByLocation::DataPointClicked(point : DotNet "Microsoft.Dynamics.Nav.CI
//***
BusinessChartBuffer.SetDrillDownIndexes(point);
GenerateDataForChart.OnDataPointClicked(BusinessChartBuffer);
//***

InventoryByLocation::DataPointDoubleClicked(point : DotNet "Microsoft.Dynamics.

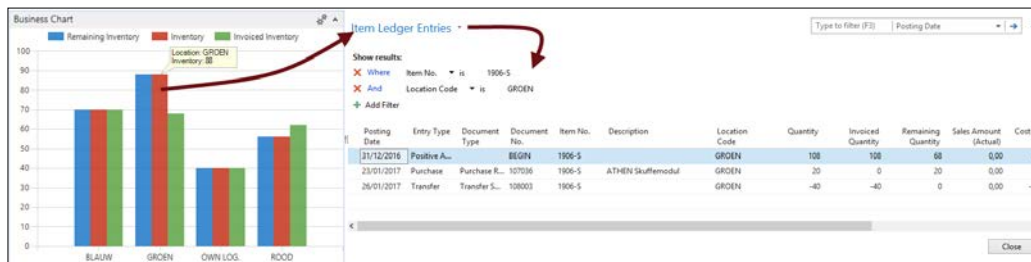
InventoryByLocation::AddInReady()
//***
IsChartAddInReady := TRUE;
UpdateChart;
//***


```



I have also built in a check to see if the chart is using the `IsChartAddInReady` variable. This makes sure that there are no errors when the .NET add-on is not yet initialized, depending on the version of Dynamics NAV and the client that you are using.

Now, when you click on a bar in the chart, a popup window opens to display the details, or drilldown data:

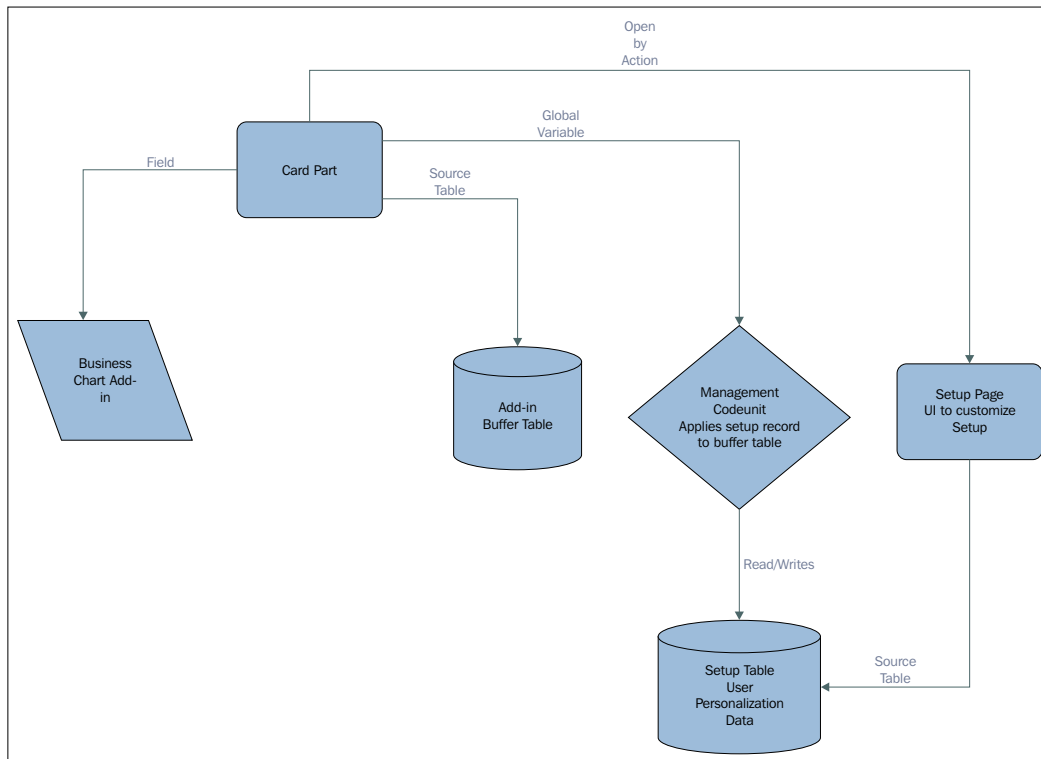


 An example of the objects for the business chart including drilldown functionality is available in the object: Business Chart with DrillDown

Preserving the user personalization

Some charts can be personalized by the user and you can also implement features in your own charts. You should do this in a generic way so that you can apply the personalization every time the chart is loaded. You need a table in which to store these values to accomplish this. You can use the `Business Chart User Setup (487)` table for this purpose or you can create a new table with a similar setup.

The following diagram shows the relationship between these objects:



The following Dynamics NAV charts use this setup:

- Charts that use a setup record:
 - **Page 772, Inventory Performance**
 - **Page 771, Purchase Performance**
 - **Page 770, Sales Performance**
 - **Page 762, Finance Performance**
- Chart that uses the `Business Chart User Setup` table:
- **Page 768, Aged Acc. Receivable Chart**

There are other charts in the application, such as the following:

- **Page 972, Time Sheet Chart**
- **Page 869, Cash Flow Chart**
- **Page 760, Trailing Sales Orders Chart**



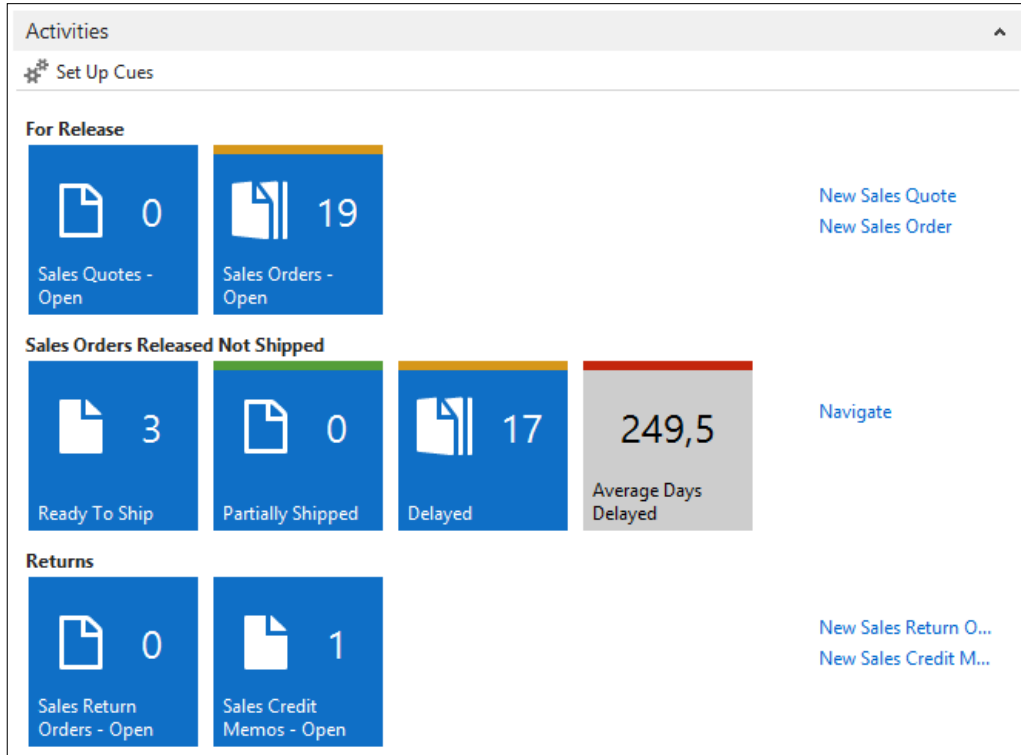
You should have a look at the code of these charts to find some more inspiration for creating and reusing business logic when you create your own charts.

There are also ways to use one chart object to display different charts at once. An example of this is the `Mini Generic Chart` (page 1390), which uses the `MiniChartManagement CodeUnit` to manage separate management codeunits for charts and their setup records, and also the `Mini Chart Definition (1310)` table which stores the last used visualization.

Implementing cues and colored indicators

I have explained in this chapter how to create charts to visualize trends and key performance indicators so that you can see how your business is doing, and one thing I have to mention is that the page where all of this usually comes together is the role center page. This is the first page you see when you launch the application, and is an ideal place to host your charts. But a role center page also contains an activity pane.

Let's look at a typical example:



The preceding example comes from the role center page of the order processor. When a user with this profile launches Dynamics NAV, they get an instant view of their work and activities. You can see the amount of quotes, orders, returns and credit notes and how many orders still need to be shipped or completed. There's even an indication of the average number of days a typical sales order is delayed. The pane automatically updates when transactions are completed in the application, so it always displays an up-to-date view. When you click on one of the cue icons, you are redirected to the corresponding list page and, on the right, there are shortcuts that navigate to the pages to create new quotes, orders, returns and credit notes, including a link to the navigation page, where you can search for posted records, depending on a posting date, document number or reference.

Let's have a look at how such a page is constructed, so you understand the underlying building blocks, because you can then customize or create new activity panes based on your own business.

The activity pane is in fact a subpage that is placed in the role center. You can always find out the source information from a page or a part of a page, when you select it and then use the **About This Page** feature or shortcut *Ctrl + Alt + F1*. In this example, it's the **SO Processor Activities (9060)** page:

About This Page: Activities

Page Information ^

Page:	SO Processor Activities (9060)
Page Type:	CardPart
Page Mode:	View
SourceTable:	Sales Cue (9053)
Rec:	

Table Fields v

Source Expressions v

FlowFilter Fields v

Filters v

URLs v



About This Page

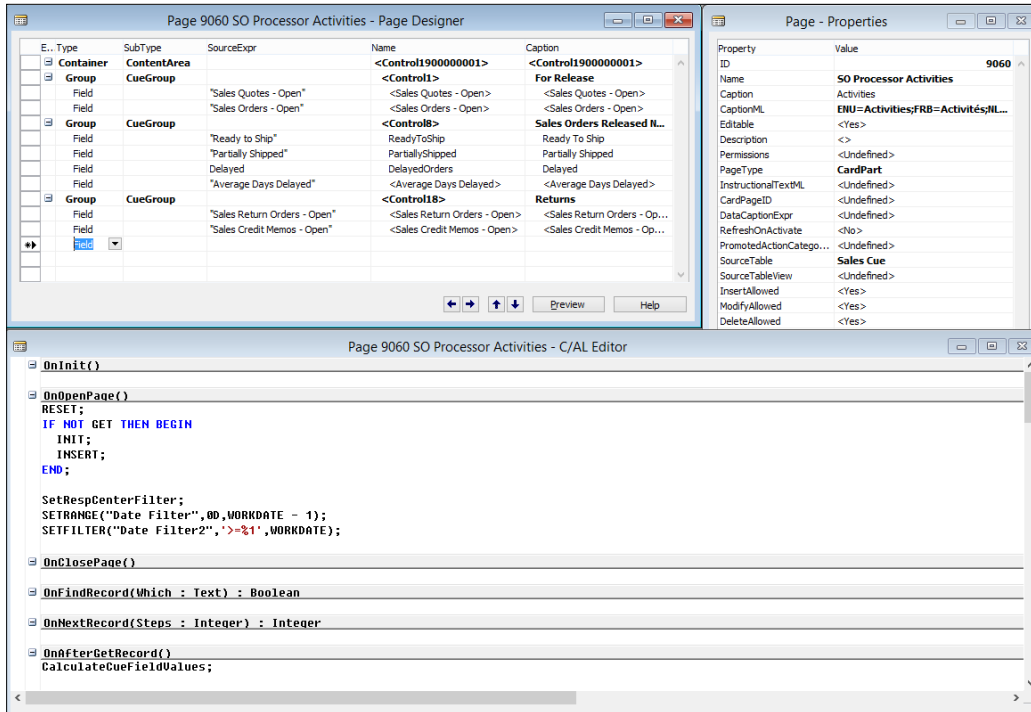
The **About This Page** feature is very useful when you are reverse engineering in Dynamics NAV. It gives you information about the source table and page (or report) you are looking at, and you can also see the table fields, source expressions, filters, URLs, and so on. The information can also be exported to XML and Excel, which is good when you are supporting customers (helpdesk). By asking a user to send you this information, you know exactly which page they are talking about, and you can provide accurate support.

If you activate the **About This Page** feature in the role center, it will show **Order Processor Role Center (9006)** as the source. It will only show the source of the activity in this role center page when you select the activities on the page and then activate the **About This Page** feature. The activities page is a subpage in the role center page.

Let's have a look at the **SO Processor Activities (9060)** page.

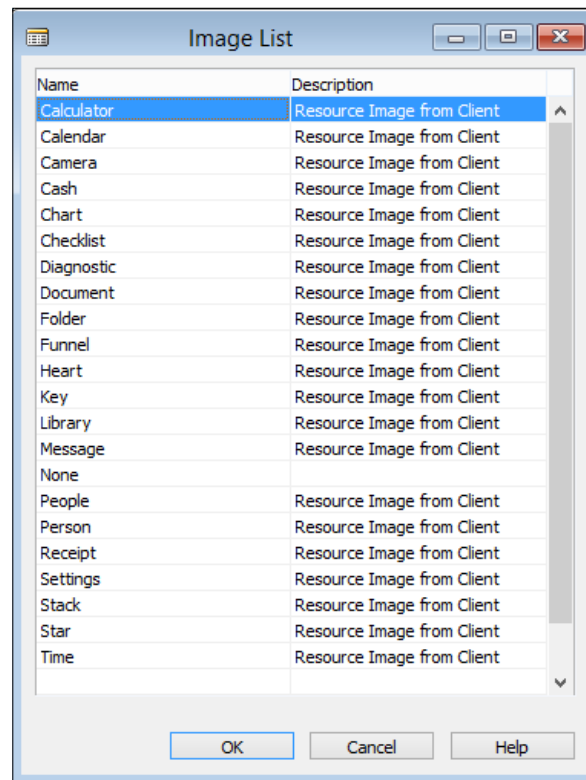
A typical activities page

The design of the SO Processor Activities (9060) page is as follows:



The page contains several fields that are placed in special groups named `CueGroup`. The fields display numerical data at runtime and they are visualized as cue icons because they are in these special `CueGroup` containers.

The exact image that is used for the cue is determined by the image property, which can be one of the following:



You can see some code in the `OnOpenPage` trigger. This code initializes the source table, which, in this case, is the `Sales Cue` table. It ensures there's a record to display and flow filters are set. A function is called in the `OnAfterGetRecord` trigger: `CalculateCueFieldValues`, which contains the following code:

```
IF FIELDACTIVE("Average Days Delayed") THEN
    "Average Days Delayed" := CalculateAverageDaysDelayed;

IF FIELDACTIVE("Ready to Ship") THEN
    "Ready to Ship" := CountOrders(FIELDNO("Ready to Ship"));

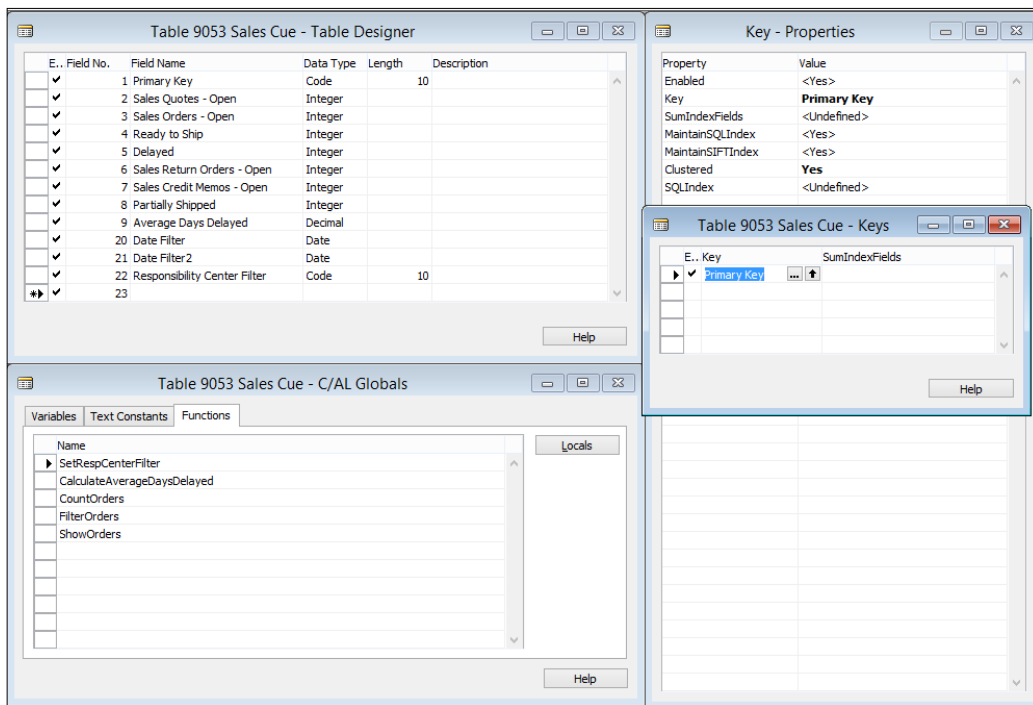
IF FIELDACTIVE("Partially Shipped") THEN
    "Partially Shipped" := CountOrders(FIELDNO("Partially Shipped"));

IF FIELDACTIVE(Delayed) THEN
    Delayed := CountOrders(FIELDNO(Delayed));
```

This code ensures that fields displayed as cue icons are calculated using functions from the underlying source table. Not all of the fields are calculated in this way, some of them come directly from the table, and they are simply FlowFields. When you create an activity page, the fields you use as cue icons need to be numerical (decimal or integer) and they can be normal fields or FlowFields. You use functions to calculate the value of normal fields, while you use FlowFilters to filter and calculate the FlowFields at runtime. Let's have a look at the underlying **Table 9053 Sales Cue**.

A typical cue table

The following screenshot shows the design of a cue table, in this example, it's **Table 9053 Sales Cue**:

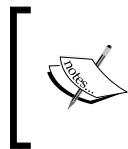


The table has a primary key, with the name `Primary Key`, and contains several `FlowFields` and `FlowFilters`. You can see in the **Globals** that the table also contains several functions, used to calculate some of the fields at runtime or to set the `FlowFilters` for the `FlowFields`. Most `FlowFields` use a count method, as in the `Sales Orders - Open` field, which has the following `CalcFormula`:

```
Count ("Sales Header" WHERE (Document
    Type=FILTER (Order) ,Status=FILTER (Open) ,Responsibility
    Center=FIELD (Responsibility Center Filter)))
```

This field simply counts the number of records in the `Sales Header` table, where the type is `Order` and `Status` is `Open`. The `Responsibility Center` field is a `FlowFilter`.

The `Average Days Delayed` field is a normal decimal field. Its value is calculated at runtime, in the page, via the function `CountAverageDaysDelayed`.

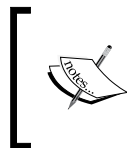


This type of table, a `Cue` table, is an example of the implementation of the singleton pattern. More information about this and other patterns is available here: <https://community.dynamics.com/nav/w/designpatterns/151.singleton-table>

Colored indicators

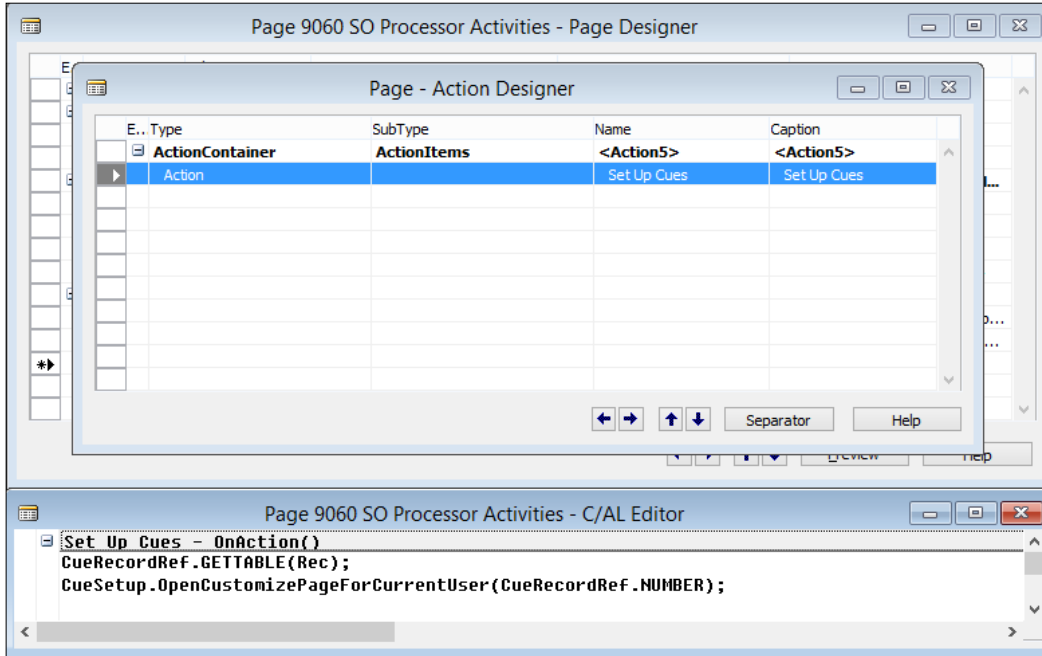
Now that you know how the page and underlying source table are designed, the next question is, where do the colors for the different cue icons come from?

The answer is twofold. Every field in the cue groups, or cue field, has a `Style` property and a `StyleExpr` property. The styles determine the different colors. You can set these properties in the page and give them a static value, in the properties window. Or you can set the values of these properties via the `C/AL` code. Clearly, this method is not used in this example, but if you would like to know how to implement it, have a look at [https://msdn.microsoft.com/en-us/library/dn789598\(v=nav.80\).aspx](https://msdn.microsoft.com/en-us/library/dn789598(v=nav.80).aspx).

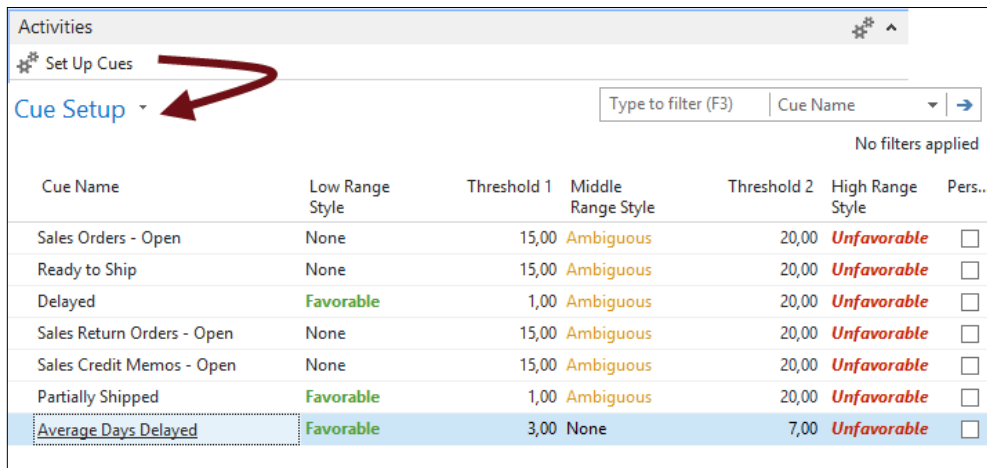


Dynamics NAV 2013 and 2009 used the `Style` and `StyleExpr` properties, and a variable, to dynamically implement colors or styles on fields. This is implemented in an even more flexible way in Dynamics NAV 2015.

If you go back to the page **SO Processor Activities (9060)** and open the page actions, you will see the following:



This shows that you can use this action to open the **Cue Setup** page and, at runtime, determine, as a user, which colors should be used for the cue icons:



This page uses the table Cue Setup (9701) and the codeunit Cue Setup (9701) to store and manage the cue icons styles at runtime. At runtime, when you run an activity page, the application uses **Codeunit 1 Application Management** in which there's a `GetCueStyle` function. This function determines which style should be used for every cue icon at runtime, as follows:

```

Codeunit 1 ApplicationManagement - C/AL Editor
GetCueStyle(TableId : Integer;FieldNo : Integer;CueValue : Decimal) : Text
EXIT(CueSetup.GetCustomizedCueStyle(TableId,FieldNo,CueValue));

Codeunit 9701 Cue Setup - C/AL Editor
GetCustomizedCueStyle(TableId : Integer;FieldId : Integer;CueValue : Decimal) : Text
Style := GetCustomizedCueStyleOption(TableId,FieldId,CueValue);
EXIT(CueSetup.ConvertStyleToStyleText(Style));

OpenCustomizePageForCurrentUser(TableId : Integer)
PopulateTempCueSetupRecords(VAR TempCueSetupPageSourceRec : TEMPORARY Record "Cue Setup")
CopyTempCueSetupRecordsToTable(VAR TempCueSetupPageSourceRec : TEMPORARY Record "Cue Setup")
ValidatePersonalizedField(VAR TempCueSetupPageSourceRec : TEMPORARY Record "Cue Setup")
LOCAL GetCustomizedCueStyleOption(TableId : Integer;FieldNo : Integer;CueValue : Decimal)
IF FindCueSetup(CueSetup,TableId,FieldNo) THEN
EXIT(CueSetup.GetStyleForValue(CueValue));

EXIT(CueSetup."Low Range Style"::None);

LOCAL FindCueSetup(VAR CueSetup : Record "Cue Setup";TableId : Integer;FieldNo : Integer)
CueSetup.SETRANGE("Table ID",TableId);
CueSetup.SETRANGE("Field No.",FieldNo);

// First see if we have a record for the current user
CueSetup.SETRANGE("User Name",USERID);
IF NOT CueSetup.FINDFIRST THEN BEGIN
// We didn't, so see if we have a record for all users
CueSetup.SETRANGE("User Name",'');
IF NOT CueSetup.FINDFIRST THEN
EXIT(FALSE)
END;

EXIT(TRUE);

```

A user can determine, via the cue setup table, when which style (or color) should be used. So, when you create your own activity page with cue icons and follow the same pattern, the same is achieved.

Cue style objects in Dynamics NAV

The following table summarizes the objects involved in determining the style, color, and indicator, of a cue icon at runtime:

Object	Description
Codeunit 9701, Cue Setup	This codeunit is called from the <code>GetCueStyle</code> trigger in codeunit 1 Application Management to determine and set the color of the cue icon.
Page 9701, Cue Setup Administration	You can use this page to set up indicators on the cues that are available in the application. It can be per user or per company in the database.
Page 9702, Cue Setup End User	This page is used by the end user to personalize the indicators that appear on a role center page.
Table 9701, Cue Setup	This table stores the customization settings for the individual cues.

Summary

In this chapter, I have explained and demonstrated the different types of charts that you can create in Dynamics NAV. The charts can be created by a user using the generic chart designer, or by a developer using the chart .NET add-on. Furthermore, I introduced how a typical activity pane is constructed, which you usually find on a role center page, and how you can set it up dynamically to determine the colors and indicators to be used at runtime. If you put all of this knowledge together, you will have all the building blocks required to create a dashboard role center, where you can visualize key performance indicators and different types of charts, giving a user an ideal starting page on which they will have a clear overview of the state of their business.

We have reached the end of this final chapter and I hope it was interesting and that you learned a lot. If you are interested, you can go to the following websites where you will find more information and examples so that you can apply the knowledge gained from this book to any project.

- My personal blog: <http://thinkaboutit.be/>
- The Dynamics NAV MSDN page: [https://msdn.microsoft.com/en-us/library/dd338686\(v=nav.80\).aspx](https://msdn.microsoft.com/en-us/library/dd338686(v=nav.80).aspx)
- The Dynamics NAV team blog: <http://blogs.msdn.com/b/nav/>
- The Dynamics NAV channel on YouTube: <https://www.youtube.com/playlist?list=PL5B63EF419A3B59C8>
- Plataan: <http://www.Plataan.tv>

Index

A

About This Page feature 465
array 240

B

barcodes
printing 253-256
Binary Large Object (BLOB) 274
BLOB field usage
optimizing 274-276
bookmark
used, for creating links 248-250
buffer table
used, for creating dataset 301
using 91
built-in layout 14
business chart
about 449, 450
advantages 449
creating 450-459
defining 451, 452
information, displaying 459-462
types 454
user personalization, preserving 462, 463

C

caching 435
captions and labels
dataset, flattening 22-27
IncludeCaption, versus
FIELDCAPTION 21
including 20, 21

charts

optimizing 169

code unit

using 91

collections

about 106, 107
Datasets 107
Fields 107
Globals 106
Parameters 107
ReportItems 107
User 106
Variables 107

colored indicators

about 469-471
setting up, URL 469

Common Language Specification (CLS)

about 18
URL 18

complex expression

 104, 105

conditional formatting

used, in report 145-149
using 125

cues and colored indicators

activities page 466-468
cue table 468, 469
implementing 463-465

cue style objects, in Dynamics NAV

 472

custom functions

creating 112-116

custom layout

 14

Custom Numeric Format Strings

URL 84

D

dashboard

references 375, 383

data analysis

with data bars 149-156
with indicators 149-156

data model

address formatting 184-187
building 14-16
captions and labels, including 20, 21
dataset, building 16
defining 178-181
InitializeRequest 203
logging 201, 202
logos, including 188-191
multilanguage, implementing 181-184
No. of Copies option 191-196
references 375
report dataset designer, defining 16
report triggers 28
totaling and VAT 197-201

data regions 50

dataset

building 16
building, alternatives 294
captions and labels 270-272
columns 17-19
correct filters, applying 284-286
data items 17-19
defining 269, 270
number formatting 281-284
query object, using for 301-307
recommendations, of Dynamics NAV 287
report totals 279, 280
techniques 270
temporary table, using 294-301
unnecessary rows, avoiding 278
unused columns, removing 272-274

data visualization 139

date format codes

URL 121

design guidelines

reporting 43

document

defining 177, 178

document reports 2

Dynamics NAV charts

setup, using 463

Dynamics NAV OData web service

calling 431-434

Dynamics NAV Team blog

URL 286

Dynamics NAV versions

URL 287

Dynamics NAV web services

about 351-353
URL 353

E

entity relationship (ER) model 11

ESRI shape file

selecting 174

Excel

Power Pivot, activating 357-359
using 354-356

expression examples

about 117
column header, repeating on
 every page 131-134
decision functions 122-126
page breaks, generating in code 126-130
working, with dates 117-120
working, with strings 120-122

expression language

defining 104
rules 104

expression placeholders

symbols, using 106

expressions

expression language 104
using, for properties 99-103

F

FIELDCAPTION function 184

filtering 56

filters

implementing 56-62
properties 56
reference link 247
used, for creating links 244-247

fixed header problem 265-268
FLOOR function 238

G

generic chart designer
about 439-445
list, displaying as chart 447-449
text management 445-447

Get function

implementing 212-217

GETURL() function

used, for creating links 250-252

global variable and functions

declaring 211, 212

green-bar effect 124

green-bar-matrix example

defining 135-137

grouping

about 66
adjacent group, adding to Tablix 74-82
expand/collapse, implementing 72-74
implementing 66
parent-child group, adding to Tablix 66-71

H

headers and footers

GetData, defining 210, 211
SetData, defining 210, 211
working with 205-210

hotfixes

implementing 293

HTML tags 89

I

information, visualizing

charts, using 164-169
defining 159
gauges, using 160-163
maps, using 170-173

internal bookmarks

used, for creating links 252, 253

item dashboard report

creating 93-96

J

Jet Reports website

URL 336

L

layout

about 203
aggregate functions 292
building 42
complex grouping 292
conditional visibility, avoiding on
 big dataset 288
creating 31
creating, in Report Builder 31
creating, in Visual Studio 37-40
dataset, filtering 204, 205
defining 287
expressions, in page header or
 footer 291, 292
headers and footers, working
 with 205-210
managing, in code 344-348
optimization, for rendering format 292
page x of y, implementing 219-221
print layout, versus print preview 287
report design guidelines 293
testing 42
visualizing information, best
 practices 289, 290
Visual Studio, versus Report Builder 31

license file 295

links

creating 242, 243
creating, bookmark used 248-250
creating, filter used 244-247
creating, GETURL() function used 250-252
creating, internal bookmarks used 252, 253

List

versus Matrix 53-55
versus Table 53-55

M

Matrix

- versus List 53-56
- versus Table 53-56

Microsoft report and user interface design guidelines

URL 293

mini-document 217-219

N

Notepad

URL 274

Notepad++

URL 274

O

objects

- defining 472
- URL 455

OData web services

references 387

P

pagination and layout

testing, in different rendering extensions 43

performance recommendations

- about 269
- dataset, creating 269, 270
- layout 287

Power BI and Q&A

references 397

PowerBI.com

about 391-397
URL 392

Power BI Designer

about 383-391
URL 383

Power Map 376-382

Power Pivot

- activating, in Excel 357-359
- data, importing into 360-365
- data model, building 359
- defining 357
- relations, creating 366-368

Power Pivot Excel data model

URL 368

Power Query 383

Power View 369-375

print support

URL 9

ProcessingOnly report

defining 30

Q

Query Execution Plan (QEP) 428

Query object

URL 361

R

recipes

- used, for implementing top x filtering 139-145

report

- about 2-5
- creating 434
- creating, in SSRS 404-415
- report viewer 7-9
- request page 6, 7
- scheduling 348-350
- testing, in different clients 43

Report Builder

- features 32, 33
- layout, creating 31
- wizards, for prototyping 33-37

report creation workflow 45, 46

report description 45

report design guidelines

URL 44

report development phases

- about 9
- data model phase 9-11
- layout phase 11
- testing phase 12

report development tools

- about 12
- data model, developing 13
- report layout, creating 13

report execution flow

- about 343
- Word report execution flow 344

Reporting Services

about 399, 400

URL 401

reporting user

using 406

report items

CanGrow property 92, 93

CanShrink property 92, 93

defining 49, 50, 423

formatting 82-84

placeholders, using 85-91

report layouts

Custom Layouts button, selecting 339, 340

Custom RDLC layout, editing 341, 342

managing 338

report logging 263-265

Report Manager

about 400

URL 403, 413

report pagination 223-225

report server

configuring 400-403

installing 400-403

Report Server

URL 413

Report Server Project Template

URL 415

report setup table

using 262, 263

report templates 256-261

Report Totals Buffer 217

report triggers

about 28

report, running 28

sequence 29, 30

request page 44

reusability

datasets 419-423

functions, creating 426-428

implementing 419

shared data sources 419-423

shared report parts 423-426

stored procedures, using 428-430

reusable custom functions 117

Roles and Profiles

URL 450

rollup updates

implementing 293

RowNumber function 238

run and run modal 350

S

scheduling 436, 437

scope

defining 108-112

Section Designer 239

Set function

implementing 212-217

simple expression 104, 105

snapshots 436

sorting

about 56

implementing 62, 63

interactive sorting 63-66

Sparklines

used, for visualizing trends 156-159

SQL Server Evaluation version

URL 401

SQL statement

FROM 302

GROUP BY 302

HAVING 302

JOIN Type 302

ON 302

ORDER BY 302

SELECT 302

TOP 302

WHERE 302

SSDT-BI

URL 404

SSRS

report, creating 404-415

report project, publishing 417-419

SQL Server Data Tools, using 415, 416

standard reports

URL 2

static and dynamic rows

URL 66

static report items

defining 50

subscribing 436, 437

T

Table

- versus List 53-56
- versus Matrix 53-56

table header

- repeating 326, 327

Tablix

- about 52
- defining 50
- Document Outline window, opening 51, 52
- name, changing 52, 53

TED

- URL 369

temporary table

- about 295
- disadvantage 295

textbox format property

- URL 85

tips and tricks, report layouts

- barcodes, printing 253-256
- fixed header problem 265-268
- fixed number of rows 235-239
- footer or header, displaying on last page 226-229
- information, placing in bottom of page 229-235
- links, creating 242, 243
- report logging 263-265
- report pagination 223-225
- report setup table, using 262, 263
- report templates 256-261
- transfooter 239
- transheader 239-241

top x filtering

- implementing, recipes used 139-145

transfooter 239-241

transheader 239-241

U

unused columns

- variables and setup information 277, 278

User Access Control (UAC) 415

V

version, of Visual Studio

- URL 31

Visual Studio

- document outline 41, 42
- features 40
- layout, creating 37-40
- report formatting 41, 42
- toolbars 41, 42

W

Waldo

- URL 294

web service

- about 351
- types 351
- URL 351

Windows Azure Marketplace

- URL 362

Word report

- dataset, optimizing for 335-337

Word report execution flow

- about 344
- at design time 344
- at runtime 344

Word report layout

- advantages 311
- creating 311-321
- defining 309-311
- formatting 322-326
- limitations 311
- Word templates, using 327-335

X

XML data sources, querying

- URL 434

Y

Year To Date (YTD) 118



**Thank you for buying
Microsoft Dynamics NAV 2015
Professional Reporting**

About Packt Publishing

Packt, pronounced 'packed', published its first book, *Mastering phpMyAdmin for Effective MySQL Management*, in April 2004, and subsequently continued to specialize in publishing highly focused books on specific technologies and solutions.

Our books and publications share the experiences of your fellow IT professionals in adapting and customizing today's systems, applications, and frameworks. Our solution-based books give you the knowledge and power to customize the software and technologies you're using to get the job done. Packt books are more specific and less general than the IT books you have seen in the past. Our unique business model allows us to bring you more focused information, giving you more of what you need to know, and less of what you don't.

Packt is a modern yet unique publishing company that focuses on producing quality, cutting-edge books for communities of developers, administrators, and newbies alike. For more information, please visit our website at www.packtpub.com.

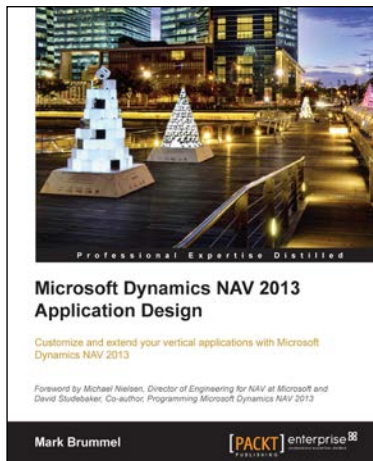
About Packt Enterprise

In 2010, Packt launched two new brands, Packt Enterprise and Packt Open Source, in order to continue its focus on specialization. This book is part of the Packt Enterprise brand, home to books published on enterprise software – software created by major vendors, including (but not limited to) IBM, Microsoft, and Oracle, often for use in other corporations. Its titles will offer information relevant to a range of users of this software, including administrators, developers, architects, and end users.

Writing for Packt

We welcome all inquiries from people who are interested in authoring. Book proposals should be sent to author@packtpub.com. If your book idea is still at an early stage and you would like to discuss it first before writing a formal book proposal, then please contact us; one of our commissioning editors will get in touch with you.

We're not just looking for published authors; if you have strong technical skills but no writing experience, our experienced editors can help you develop a writing career, or simply get some additional reward for your expertise.

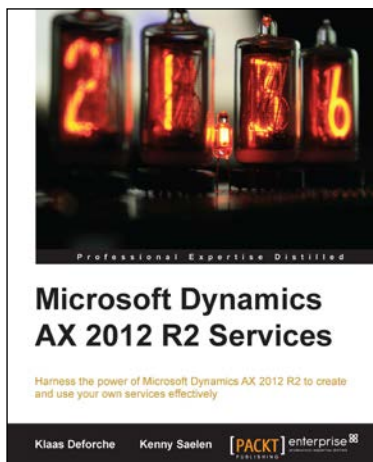


Microsoft Dynamics NAV 2013 Application Design

ISBN: 978-1-78217-036-5 Paperback: 504 pages

Customize and extend your vertical applications with Microsoft Dynamics NAV 2013

1. Set up your application for a number of vertical industries and scenarios.
2. Get acquainted with Dynamics NAV's data model and transaction schema with the help of highly efficient design patterns.
3. Consists of two completely designed and explained vertical solutions, including application objects.



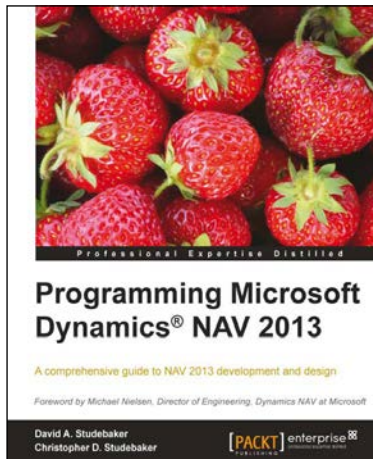
Microsoft Dynamics AX 2012 R2 Services

ISBN: 978-1-78217-672-5 Paperback: 264 pages

Harness the power of Microsoft Dynamics AX 2012 R2 to create and use your own services effectively

1. Develop your Java applications using JDBC and Oracle JDeveloper.
2. Explore the new features of JDBC 4.0.
3. Use JDBC and the data tools in Oracle JDeveloper.

Please check www.PacktPub.com for information on our titles

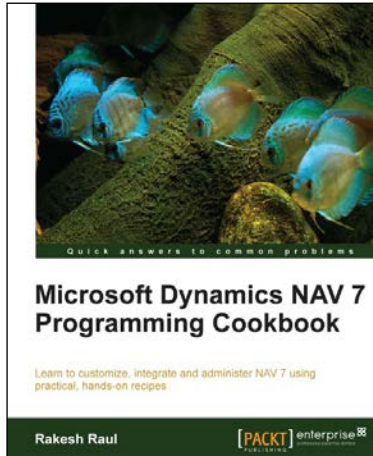


Programming Microsoft Dynamics® NAV 2013

ISBN: 978-1-84968-648-8 Paperback: 360 pages

A comprehensive guide to NAV 2013 development and design

1. A comprehensive reference for development in Microsoft Dynamics NAV 2013, with C/SIDE and C/AL.
2. Brimming with detailed documentation that is additionally supplemented by fantastic examples.
3. The perfect companion for experienced programmers, managers and consultants.



Microsoft Dynamics NAV 7 Programming Cookbook

ISBN: 978-1-84968-910-6 Paperback: 312 pages

Learn to customize, integrate and administer NAV 7 using practical, hands-on recipes

1. Integrate NAV with external applications, using the C/AL or SQL server.
2. Develop .NET code to extend NAV programming possibilities.
3. Administer the Microsoft NAV 7 server and database.

Please check www.PacktPub.com for information on our titles