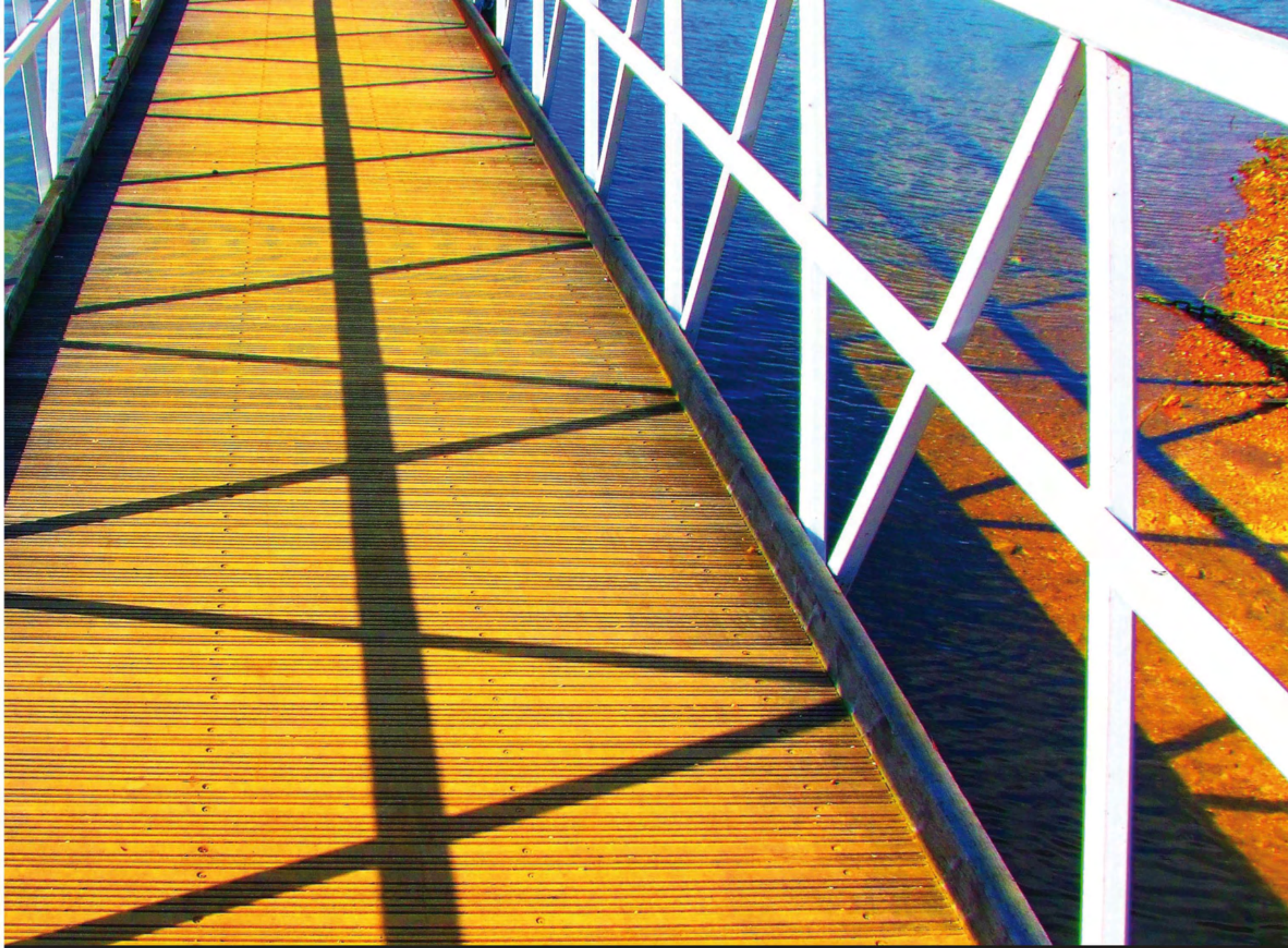


Professional Expertise Distilled

Microsoft Dynamics NAV 2009 Application Design

Design and extend complete applications using Microsoft Dynamics NAV 2009

Foreword by Michael Nielsen,
Director of Engineering, Dynamics NAV, Microsoft Development Center

Mark Brummel

[PACKT] enterprise
PUBLISHING
professional expertise distilled

Microsoft Dynamics NAV 2009 Application Design

Design and extend complete applications using Microsoft Dynamics NAV 2009

Mark Brummel

[PACKT] PUBLISHING enterprise

professional expertise distilled

BIRMINGHAM - MUMBAI

Microsoft Dynamics NAV 2009 Application Design

First published: June 2010

Production Reference: 1040610

Published by Packt Publishing Ltd.
32 Lincoln Road
Olton
Birmingham, B27 6PA, UK.

ISBN 978-1-84968-096-7

www.packtpub.com

Cover Image by Tina Negus (tina_manthorpe@sky.com)

Credits

Author
Mark Brummel

Reviewers
Daniel Rimmelzwaan
Max Traxinger
Jeremy Vyska
Eric Wauters

Acquisition Editor
Rashmi Phadnis

Development Editor
Mayuri Kokate

Technical Editors
Neha Damle
Bhavesh D. Bhasin

Editorial Team Leader
Gagandeep Singh

Project Team Leader
Lata Basantani

Project Coordinator
Poorvi Nair

Proofreader
Lesley Harrison

Indexer
Rekha Nair

Production Coordinator
Adline Swetha Jesuthas

Cover Work
Adline Swetha Jesuthas

Foreword

The history of the Dynamics NAV application started with a small accounting product called "PC-Plus" back in 1984. The idea was to create an application suitable for bookkeepers who had no knowledge of computers. It was done by copying real life artifacts like paper journals, orders, and invoices and it even had a copy of a Canon table calculator! "PC-Plus" was an overnight success – but soon a huge demand for customizations and verticals started developing. After the "PC-Plus" team developed a vertical for auto repair shops called "Auto-Plus", they realized that if they had to create all the functionality themselves, their business would never scale. Therefore, they got the idea to create a product including a base application and a set of tools suitable for others to build exactly the functionality they wanted. That was the start of Dynamics NAV as we know it today.

Over the years, the base application evolved. When "PC-Plus" was created, people loved it for its core G/L functionality—but soon the same people wanted the same simple solution to help run more of their business and eventually manage their entire business. This meant that the Dynamics NAV application grew from being all about finance management to cover manufacturing, distribution, service management, projects and CRM and is today a fully-fledged ERP solution. Even though the application has grown, it is still simple and consistent, which makes it possible for a single developer to overview it, understand it, and build exactly the necessary functionality.

Michael Nielsen

Director of Engineering, Dynamics NAV,
Microsoft Development Center Copenhagen — Denmark

About the Author

Mark Brummel is an all-round Microsoft Dynamics NAV specialist. He started in 1997 as an end user but quickly moved to the other side of the table. For ten years he has worked for resellers where designing and maintaining add-on systems was his specialization. Some of these add-on systems exceed the standard product when it comes to size and complexity. He has also been coaching colleagues and troubleshooting 'impossible' problems as a part of day-to-day work. Mark has trained most of the experienced NAV developers for the NAV 2009 product in The Netherlands and Belgium and he has been hired by almost every NAV reseller in the Benelux.

Mark works as a freelancer. His main focus area is to help out end users in implementations all over the world.

Mark was the first world wide to use the NAV 2009 (CTP3) product in a production system feeding back valuable information to Microsoft. Today, he is still involved in projects to provide input for future versions and test new CTP releases.

A special project has been performance tuning of the Dynamics NAV product on SQL Server. From 2007 to 2009, he was involved in the development of the 'SQL Perform Tools' as business partner of SQL Perform Benelux. As a unique specialist, he has done break through research in improving the performance of Dynamics NAV on SQL Server.

In his spare time, Mark maintains his blog on `www.brummelds.com`. This blog contains a wide range of articles about both the Microsoft Dynamics NAV and SQL Server product. He is also a frequent speaker at Microsoft events and a writer for independent Dynamics NAV websites and user groups. In 2006, Mark was rewarded by Microsoft with the Most Valuable Professional award for his contribution to the online and offline communities. In 2007 and 2009 he also reviewed "Programming Microsoft Dynamics NAV" and "Programming Microsoft Dynamics NAV 2009".

Acknowledgement

Just before I was asked to write this book, I sold my company and went back to freelancing in order to spend more time with my wife Dionel and kids Josephine (6), Wesley (4) and Saskia (1½). Therefore, I would like to thank them for allowing me this great opportunity and challenge. Writing this book has taken the better part of weekends, evenings, and vacations for a period of almost half a year. I will never forget this period where daddy is writing a book and Josephine asking, "What is your book about dad?" (in Dutch of course) and me finding it difficult to explain.

I would also like to thank Packt Publishing for giving me the opportunity to write a book about the application that I love so much. The people at Packt were very helpful and patient in helping me explore the life of an author and my continuous efforts in changing the outline and number of chapters and pages.

I have tried my best to write a book about the real intention of what Microsoft Dynamics NAV is. NAV is more than ERP and more than a Development Environment. It is about simplicity and being able to create applications using a unique structure and way of thinking.

Many people have helped to get this book at the great level it is. First of all, the technical reviewers, Daniel Rimmelzwaan, Matt Traxinger, Jeremy Vyska, and Eric Wauters who have done an excellent job in improving my efforts in writing. I would like to thank Andrew Good for his help with *Chapter 5*, *Production*.

The examples for implementing Microsoft Dynamics NAV in vertical industries in Chapters 5 and 6 are taken from real implementations and I would like to thank these companies for taking the time and effort to answer my questions:

- Arseus: Karel Jutte
- Bakery 't Stoepje: Sjacco Nel
- Bosman Medical Supplies: Renate Lukassen
- Rev'it Sport: Bas Stijntjes, Theo van Geel, and Peter Kuypers
- Wildkamp: Albert Smit and Gerard Nijlant

I would like to thank all the people who helped me during the first thirty three years of my life. Leaving school when I was 18, I studied at the university of life (sometimes the hard way) with the help of many who guided me in my personal and business life. Amongst all these people are of course my wife Dionel, my parents, great parents, brother Rene and sister Anna, and my parents-in-law.

Special thanks go to David and Karen Studebaker. I met them in 2006 at Tech-Ed in Boston, USA. They have been of invaluable help in bringing structure in my business and personal life.

About the Reviewers

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. When 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 10 years later, a long stint at a Microsoft partner, a few years as a freelancer, and now back in the partner channel, Daniel is still working with NAV. He currently works for Archerpoint, one of the largest and most experienced Microsoft Dynamics NAV partners in the USA, and he is enjoying his career more than ever.

Daniel has had the opportunity to work in a wide variety of roles such as Developer, Analyst, Designer, Team Lead, Project Manager, Consultant, and more. Although he has a very versatile experience with all things related to NAV, his main focus is custom development, with a bias toward helping his customers solve NAV performance issues on SQL Server.

Ever since he started working with NAV, Daniel 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, Daniel received his first of five consecutive Microsoft Most Valuable Professional Awards in July 2005, which was just the second year that the MVP Award was given out for NAV. The MVP Award is given out by Microsoft to independent members of technology communities around the world, and recognizes people that share their knowledge with other members of the community.

Daniel lives with his wife and two kids in Michigan in the USA.

Matt Traxinger graduated from the Georgia Institute of Technology in 2005 with a B.S. in Computer Science, specializing in Human Computer Interaction and Cognitive Science. After college, he took 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 following years, Matt learned all areas of the product and earned Microsoft Certified Business Solutions Professional certifications in both technical and functional areas of NAV. He continues to stay current with new releases of the product and is certified in multiple areas for versions 4.0, 5.0, and 2009.

Currently Matt works in Norcross, GA, for Canvas Systems, one of the largest resellers of new and refurbished computer equipment as an in-house NAV Developer and Business Analyst. He supports multiple offices in the United States as well as locations in the United Kingdom and the Netherlands.

Matt is also writing the NAV Development Cookbook for Packt Publishing which should be released later this year. In his spare time, you can find him on the online communities `Mibuso.com` and `DynamicsUser.net` under the name MattTrax, helping others learn more about the Dynamics NAV software.

Jeremy Vyska started his career in Microsoft Dynamics NAV in 2000. Since then, he has worked with, implemented, and customized almost every functional area of Dynamics NAV. He has fulfilled many business process roles relating to Dynamics NAV, including technical sales, requirements gathering, project planning, implementation, training, support, and of course, development. He has overseen teams of staff working on a project, as well as co-ordinating of independent resources.

Jeremy has also specialized in add-on and vertical development, which is a different skill-set than typical client development. Optimizations and maintainability are even more critical in those products, since partners may need to implement and customize the solution. He has been on the development and PM teams of at least half a dozen such solutions that are actively in the market space today, most notably Serenic Software on their premiere NAV-based solution for non-profits and NGO's.

Currently, Jeremy runs Small Square Services (founded in 2009), providing a wide variety of services to the worldwide Microsoft Dynamics NAV Partner Channel and supporting customers throughout the northeast US.

Eric Wauters is one of the founding partners of iFacto Business Solutions `www.ifacto.be`. With nine years of technical expertise, he is an everyday inspiration to its development team. As development manager, he continually acts upon iFacto's technical readiness and guarantees that he and iFacto are always on top of the latest Microsoft Dynamics NAV developments. Apart from that, Eric is also very active in Microsoft Dynamics NAV community-life where he tries to solve technical issues and thrives to share his knowledge with other Dynamics NAV enthusiasts. Surely, a lot amongst you will have read some of Eric's posts on `Mibuso.com`, `Dynamicsusers.net` or his own blog `www.waldo.be` which he invariably signs with "waldo". In 2008, he co-founded the Belgian Dynamics Community, a platform for all Belgian Dynamics NAV users, consultants and partners, enabling knowledge sharing and networking. His proven track record entitled him to be awarded in 2007, 2008, 2009, and 2010 as MVP (Microsoft Most Valuable Professional).

Table of Contents

Preface

In 1997, the company I worked for was looking for a replacement for their MS-DOS-based software package. We were very fortunate in finding Navision Financials 1.1 as a software package that supported the upcoming Windows platform, and was flexible enough to be implemented supporting our demands.

Even though the standard functionality was nowhere near what we have today, the structure of the application's design was simple and solid and has not changed since then.

In the years after that more companies embraced Navision as their answer to the changing demands in the market, and many vertical solutions that exist today started their life cycle. With the acquisition of Navision by Microsoft the interest of new partners grew in to the channel we know today.

The add-on solutions catalogue now has 620 pages with 548 solutions from 208 partners in 36 countries, making Microsoft Dynamics NAV a very popular development platform to create business software.

Microsoft Dynamics NAV offers a unique development experience that can only be fully used once you understand how the standard application parts are designed.

When properly licensed, everyone can change how the application works. With this great possibility comes great responsibility as this means that we can also easily break important business logic.

This results in a unique need for a designer of applications that run inside Microsoft Dynamics NAV to know more about the application without going into deep functional details.

The balance in this book will be between learning and understanding how the standard application features of Microsoft Dynamics NAV are designed, and learning how to use this knowledge when designing our own solutions. The area between understanding the application's functionality and its technical design is very thin.

In this book, we will make changes to the standard application and also create new solutions.

We will also discuss how Microsoft Dynamics NAV can interface with other applications.

What this book covers

Chapter 1, *Introduction to Microsoft Dynamics NAV*, will introduce you to Microsoft Dynamics NAV. We will talk briefly about the history of the application and talk about the concepts. We will cover some of the basics such as Number Series and Navigation. Then we will discuss the Data Model principles used by Microsoft Dynamics NAV using Master Data, Journals, and Ledger Entries covered by Documents.

Chapter 2, *A Sample Application*, will implement the theory we learned in the first chapter to make a sample application. The goal of this chapter is to better understand how Journals and Ledger entries work throughout the system and how to create your own Journal application. You will learn how to reverse engineer the standard application to learn from it and apply this to our own customizations. We will integrate the application with Relationship Management and Sales in Microsoft Dynamics NAV, and extend Navigation and Dimensions for our solution.

Chapter 3, *Financial Management*, will explore how the Financial Management part of the application can be used and how it is designed. You will learn important concepts such as VAT, Posting Groups, Closing Dates, Entry Application, and Financial Data Analysis. We will make some changes in the core application by adding new information to the General Ledger, and learn how to integrate Financial Management into our add-on solution.

Chapter 4, *Relationship Management*, will help us to analyze the sales data in our system and be more productive towards our customers. We will explore the unique design of this part of the application and integrate this with the sample application we created in *Chapter 2*.

Chapter 5, *Production*, will show us how to set up Microsoft Dynamics NAV for Production companies. We will discuss the BOM Journal, Manufacturing, and Kitting. Item Costing and Item Tracking are key elements when using this part of the application. We will look at the Planning Worksheet and how to create Production orders using Make-to-Order and Make-to-Stock policies. We will reverse engineer the Inventory Profile Offsetting codeunit and see how this leads to a planning and Purchase Orders. At the end of this chapter we will look at ten ways to customize Production for vertical industries.

Chapter 6, *Trade*, will discuss the relationship between Sales, Inventory Management, and Purchasing, and how Warehousing can be involved using different levels of complexity. We will learn how Reservation Entries are used in the system from a technical perspective.

Chapter 7, *Storage and Logistics*, will design and build a solution for planning routes for shipments, a feature that is not available in Microsoft Dynamics NAV. We will design a solution that can be used by trading companies not only for their own shipments but also for storage companies. The solution is seamlessly integrated with the Dynamics NAV product. We will extend the journal knowledge we learned in *Chapter 2* and *Chapter 3* with new document structures we learned in *Chapter 5* and *Chapter 6*.

Chapter 8, *Consulting*, will discuss how to implement the Job functionality using four example jobs, and extend jobs with an issue registration and timesheet application using resource groups and calculations.

Chapter 9, *Interfacing*, will discuss how to design a rock solid business to business interface. We will show what technologies are available to use for interfacing and how these technologies are implemented in the standard product. We will discuss all the built-in interfaces with other Microsoft applications such as Office, SharePoint, BizTalk, and Exchange.

Chapter 10, *Application Design*, will focus on the concepts of application design and how they apply to Microsoft Dynamics NAV. We will focus on Design to Use, Maintain, Support, Upgrade, Perform, and Analyze. This includes concepts for User Interface, Version Management, and the Development Methodology.

What you need for this book

To successfully follow the examples in this book you will need the following:

- The Microsoft Dynamics NAV 2009 product CD to install the application.
- Ideally you should have a full developers license. This can be obtained by being registering as a Microsoft Dynamics NAV partner. Alternatively, most of the example code can be explored using a demo license which can be downloaded from MSDN.
- Microsoft Office and SQL Server Management Studio for the interface examples in *Chapter 9*.

The appendix describes how to install these prerequisites.

Who this book is for

Basically, this book is for:

- NAV consultants and developers
- Designers of business applications
- Application Managers and End Users
- Business Owners and influencers

This book assumes that you have a basic understanding of business management systems and application development, with a working knowledge of Microsoft Dynamics NAV or another ERP system.

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 are shown as follows: "We need to reverse engineer this code in order to see what we need to create for our `CreateSquashPlayer` function."

A block of code is set as follows:

```
IF Type = Type::Company THEN
  ContComp := Rec
ELSE
  ContComp.GET("Company No.");
```

When we wish to draw your attention to a particular part of a code block, the relevant lines or items are set in bold:

```
No. - OnValidate()
IF "No." <> xRec."No." THEN BEGIN
  SquashSetup.GET;
  NoSeriesMgt.TestManual(SquashSetup."Squash Player Nos.");
  "No. Series" := '';
END;
```

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: "Now we are in the **Action Designer** and we can search for the **Create as** part."

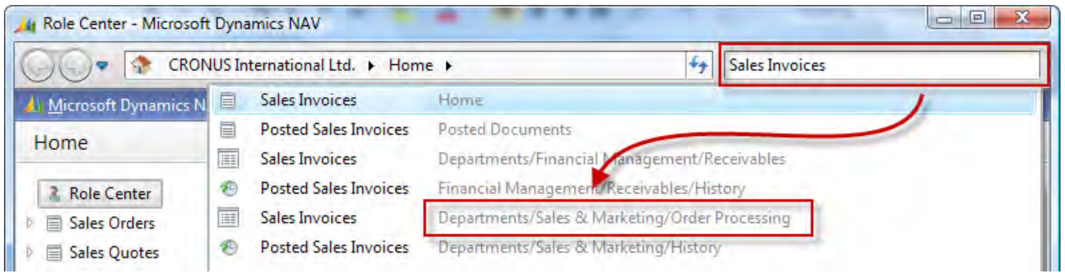
Warnings or important notes appear in a box like this.

Tips and tricks appear like this.

Where to find the Screens in this book

Most of the screens in the book were created using the Role Tailored Client introduced with Microsoft Dynamics NAV 2009. Wherever possible or necessary the Role Center that was used is mentioned. Some chapters had new or modified Role Centers.

To find a screen, type the name into the search window in the upper right corner of the Role Center (as shown in the following screenshot). This will tell you where in the application's menus the screen can be found:

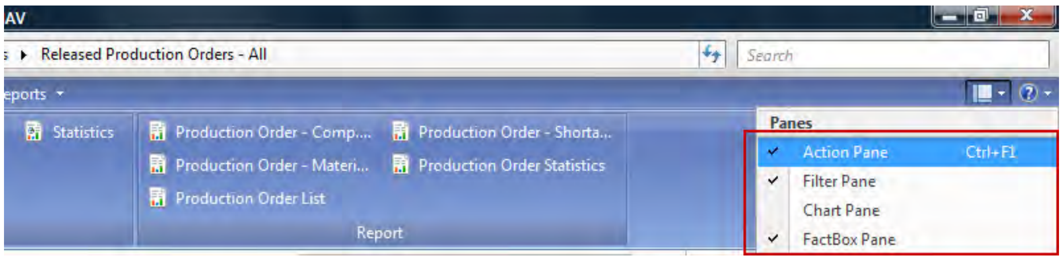


Sales Invoices	Home
Posted Sales Invoices	Posted Documents
Sales Invoices	Departments/Financial Management/Receivables
Posted Sales Invoices	Financial Management/Receivables/History
Sales Invoices	Departments/Sales & Marketing/Order Processing
Posted Sales Invoices	Departments/Sales & Marketing/History

Screenshots

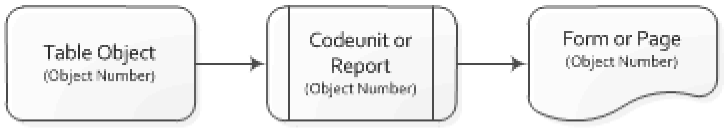
All the screenshots in this book were taken from the Role Tailored Client which was introduced with Microsoft Dynamics NAV 2009.

For most of the images the **Action Pane** and **FactBox Pane** were turned off to save space. This can be done using the **Customize** option on each page.


AV
Released Production Orders - All
Search
eports
Statistics
Production Order - Comp...
Production Order - Shorta...
Production Order - Materi...
Production Order Statistics
Production Order List
Report
Panes
Action Pane Ctrl+F1
Filter Pane
Chart Pane
FactBox Pane

How to read the application schemas

Most of the chapters in this book have application schemas to clarify the flow of data through the system. They are specially designed for this book.



To read the schemas follow the arrows. Wherever possible the functional areas are grouped using boxes. Some schemas might have more starting and ending points as this is how the application is designed. Multiple master data tables are processed using normalized business logic.

Number and date punctuation

This book was written by a Dutch author, which means that all the number and date formatting is done in Dutch formats, for example, 1.000,00 instead of 1,000.00 and 18-10-10 for October 18th 2010.

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 book that you need and would like to see us publish, please send us a note in the **SUGGEST A TITLE** form on `www.packtpub.com` or e-mail `suggest@packtpub.com`.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book on, 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 example code for the book

Visit `https://www.packtpub.com//sites/default/files/downloads/0967EN_Code.zip` to directly download the example code.

The downloadable files contain instructions on how to use them.

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/support`, selecting your book, clicking on the **let us know** 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 Introduction to Microsoft Dynamics NAV

Once Upon a Time; this is how fairytales often start and even though the story of Microsoft Dynamics NAV is anything but a fairytale, it sure has some magic.

With more than 1,350,000 seats and 75,000 installations it is one of the most popular ERP packages in the mid market. In this book, we will go through the magic of the Dynamics NAV application. We'll see how Dynamics NAV will give better information on how our business is doing and better insight into where the processes can be optimized or need to be changed.

In this chapter, we'll discuss the basic principles of the Microsoft Dynamics NAV application, how it's structured and why. After reading this chapter, you will have a better understanding of what to expect when implementing and designing Microsoft Dynamics NAV.

Versions and history

At the time of publishing this book, Microsoft Dynamics NAV 2009 (6.0) SP1 is the most recent version of the product. When the Windows version was first introduced in 1995, the product was called Navision Financials 1.0. The Danish software company that originally developed the product, Navision Software A/S, was not yet acquired by Microsoft and it was a revolution. It was a full Windows product and had all the basic functionality that small companies needed. It is important to understand that the original version was targeted at smaller companies.

Since then, we have had many (20+) versions. All new versions contained new functionality and with that, the product has gotten more mature and more suitable for bigger companies. This was especially empowered with the support of the Microsoft SQL Server platform allowing more concurrent users to work in the same application areas.

Until version 5.0, the technology of the product did not change. The original intention of Microsoft was to release a new technology platform together with the new functional changes. This turned out to be a very difficult task so they decided to split the improvements into two releases. Version 5.0 contains new functionality and improvements, whilst version 2009 or 6.0 which is the technical release number, is a technology release.

The technical challenge was to migrate from the old C++ platform to .NET and to move from a two tier to a three tier technology. This was also the first release with a drastic change in the user interface. Microsoft Dynamics NAV 2009 contains an entirely new user interface, the "Role Tailored Client", built new from the ground up—the existing ("Classic") user interface is the same with no changes. During this migration process, all application functionality was frozen although small improvements and bug fixes were made in 2009 SP1.

This book supports functionality from both the 5.0 and 2009 release even though we decided to use the new 2009 interface for all user interface screenshots and pages for the development examples. As the development environment is only available in the classic client, we have taken these screenshots from there.

What is this book about

The title of the book is "Microsoft Dynamics NAV 2009 Application Design". What does Application Design mean? And what does it mean in Microsoft Dynamics NAV 2009?

Microsoft Dynamics NAV 2009 is a very complete ERP package, but unlike other ERP packages it has a design capable of providing an open structure and a development platform. The idea is to provide 80% of the solution out of the box and allow the other 20% to be designed by qualified business application developers.

The partner channel is a unique part of Microsoft Dynamics NAV. From the first moment that Navision was introduced, company management decided that it would only make sense to have an indirect selling model and to let the resellers (called partners) have the availability to change the product and add new functionality.

This book is about both the 80% and the 20%. We'll see that the percentages differ as per the industry where it is applied. Some industries have close to a 100% fit while others have a need for 80% development.

So there is a thin line in this book between using the standard application and designing changes and expanding the product. Although this is not a development book, we'll dive into code and objects in almost every chapter.

To understand the code it should be enough to read this chapter but if you want to know more we highly recommend reading "Programming Microsoft Dynamics NAV 2009" written by David Studebaker and also published by Packt.

This book is not a manual for Microsoft Dynamics NAV 2009. It should give a clear idea of how the structure of the application is laid out and about its possibilities. We do not want to replace or rewrite the Microsoft Documentation but rather want to provide ideas you might not have thought about.

Setup versus customization

In Microsoft Dynamics NAV, the line between implementing and developing is very thin. Where you would do a lot of setup in other ERP packages, you'll see that it often makes more sense in Dynamics NAV to make a change with the Development Tools.

The standard package is very complete in its functionality but does not support all industries. It is more a framework for Partners to work with. In this book, we will explain this framework and what philosophy it is built on. Understanding this philosophy is critical to knowing how to expand the functionality.

But, expanding the functionality means customizing the application. Do end-users in 2009 still want customized applications? Mostly they will say they don't want their software customized, but in the next breath, they will say that the software should change to match their way of doing business, and that they should not have to change their business to fit the software.

This is why Microsoft pushes their Partners to create horizontal and vertical solutions on top of the standard product and release these solutions as products with their own versions like it was part of the standard applications. This way of using the partner channel is a unique concept that has proven to be very successful and has made Microsoft Dynamics NAV useable in almost any industry.

Most companies, however, have such a unique way of working that they will always require more or less customized solutions. The total cost of ownership depends on the level of customizations and how these customizations are designed.

The key is knowing when to do setup and when to do a customization. Only a solid understanding of the application will help you determine which is correct.

After reading this book, you will know how to design your application best to have a good balance between cost of ownership and functionality.

The beauty of simplicity

As discussed earlier, the application is designed to be expanded and changed by external partners. When this Partner program was created, a decision was made that partners could only do a good job if the application was completely open for them to add and change. This philosophy is very important to understand when you first start implementing or changing Microsoft Dynamics NAV.

Partners can change all business logic in the application. They can add new fields to tables and create their own tables. The only thing they cannot do is delete fields from the tables in the base application.

As you can see, Microsoft Dynamics NAV is an extremely flexible and open product with a lot of freedom. But with freedom comes responsibilities. In Dynamics NAV, you are responsible for the housekeeping in your system.

Horizontal versus vertical solutions

Because of this open system, partners have created thousands of smaller and larger changes to the system. Some of these changes were bundled into new functional pieces and called "add-ons". These add-ons are often solutions that change Dynamics NAV into a product for a specific industry rather than a generic ERP system. Other add-ons are specific features that can be used in all industries like EDI or Workflow. Microsoft calls the industry specific add-ons verticals and the generic add-ons horizontals.

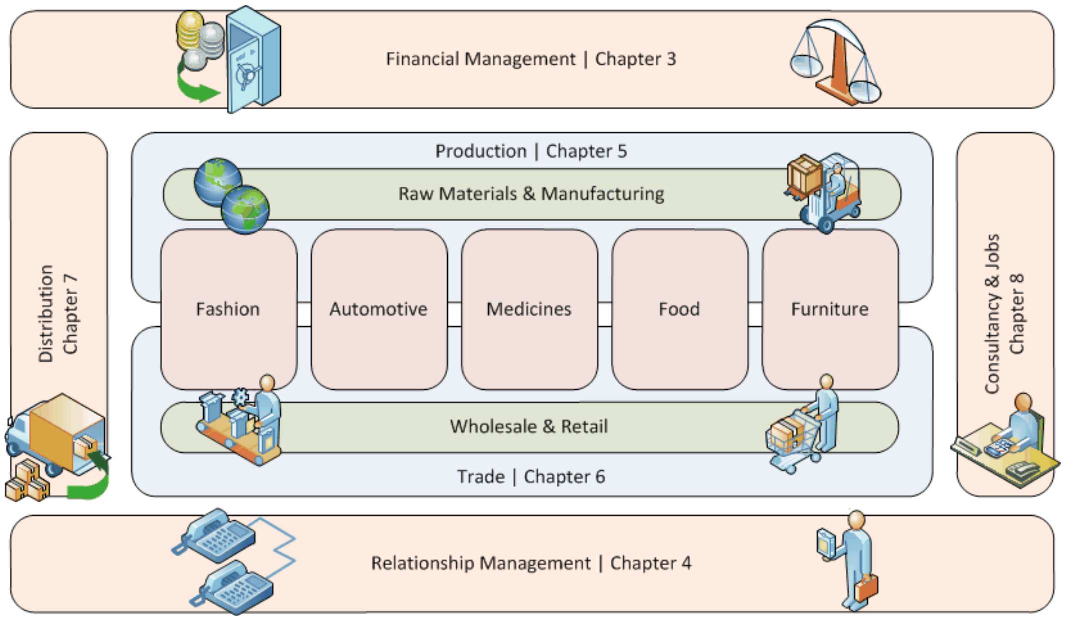
Open source

Even though Dynamics NAV has an open source for their partners, it does not come fully equipped with a development environment like most developers are used to. It has a customization tool that lets you customize the application like you would customize another ERP system with settings. This customization tool is a basic tool that is nice to work with but misses some development features such as version control and intellisense. This makes it more difficult to keep track of your changes.

Structure of this book

This book will cover most functional elements of Dynamics NAV in a number of vertical industries. We will do this in a supply chain matrix. The specific industries we will look at are fashion, automotive, medicines, food, and furniture. For production and trade we will look at the general process and we will see how consultancy and distribution companies help in this process.

The following image shows how this book is structured:


Financial Management | Chapter 3
Distribution Chapter 7
Production | Chapter 5
Raw Materials & Manufacturing
Fashion
Automotive
Medicines
Food
Furniture
Wholesale & Retail
Trade | Chapter 6
Consultancy & Jobs Chapter 8
Relationship Management | Chapter 4

For all these industries we will look at what parts of the standard product can be utilized and where we need vertical solutions. We'll discuss how these vertical solutions will interface with the standard package or maybe even change the behavior of the standard product.

Two parts of the product however are so general in their use and usability for all industries that we'll discuss them in their own chapter. These are **Financial Management** and **Relationship Management**.

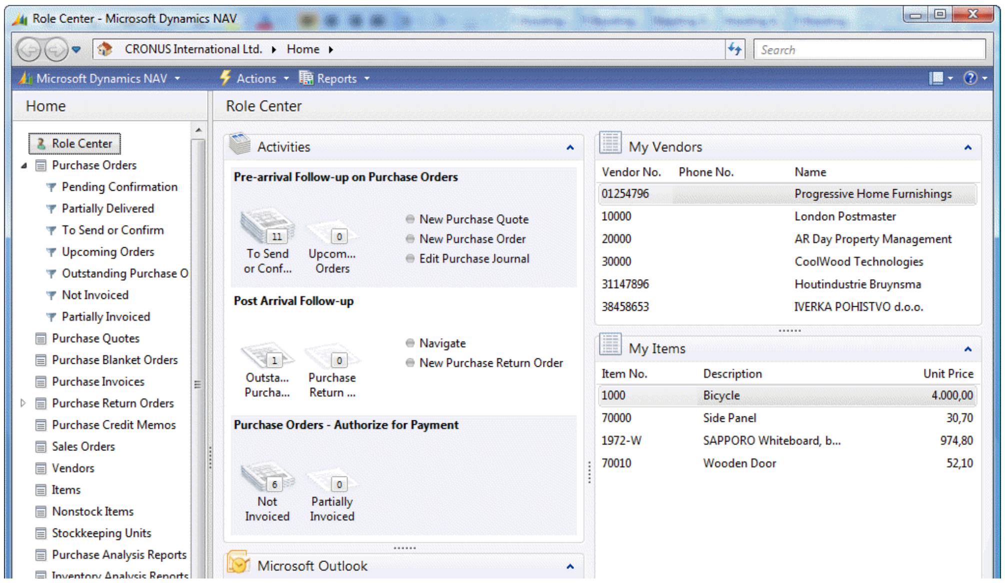
To emphasize the strength of the vertical concept, we'll design and create a vertical solution for a distribution company.

Now we will look at some of the basic concepts of the application.

The Role Tailored concept

With the NAV 2009 release, Microsoft marketing decided to introduce the concept of Role Tailored ERP. Until now, most ERP systems were module driven, which means that the application has an area for finance, CRM, sales, purchasing, and so on. The access to the individual modules was separated. A purchaser needs to switch to sales in order to see the sales orders.

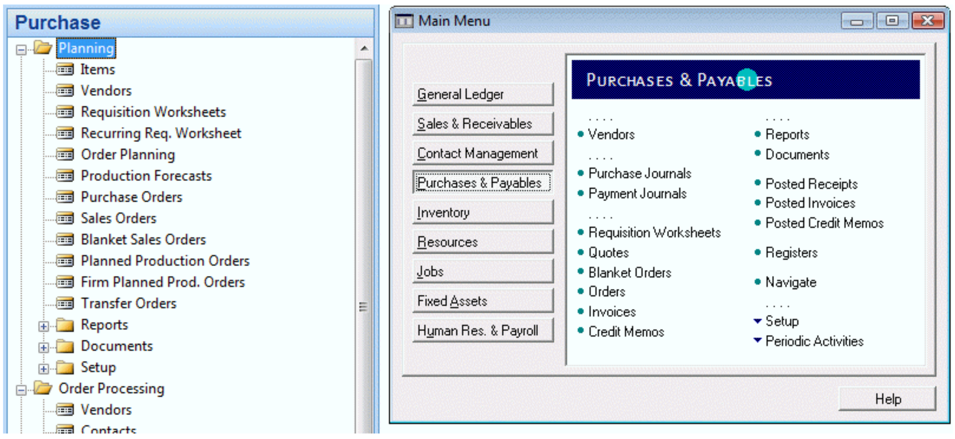
Most people in a company have specialized tasks that the ERP system should support. In a classic ERP interface, the users would have to decide themselves which parts they need. This has changed.



Vendor No.	Phone No.	Name
01254796		Progressive Home Furnishings
10000		London Postmaster
20000		AR Day Property Management
30000		CoolWood Technologies
31147896		Houtindustrie Bruynsma
38458653		IVERKA POHISTVO d.o.o.

Item No.	Description	Unit Price
1000	Bicycle	4.000,00
70000	Side Panel	30,70
1972-W	SAPPORO Whiteboard, b...	974,80
70010	Wooden Door	52,10

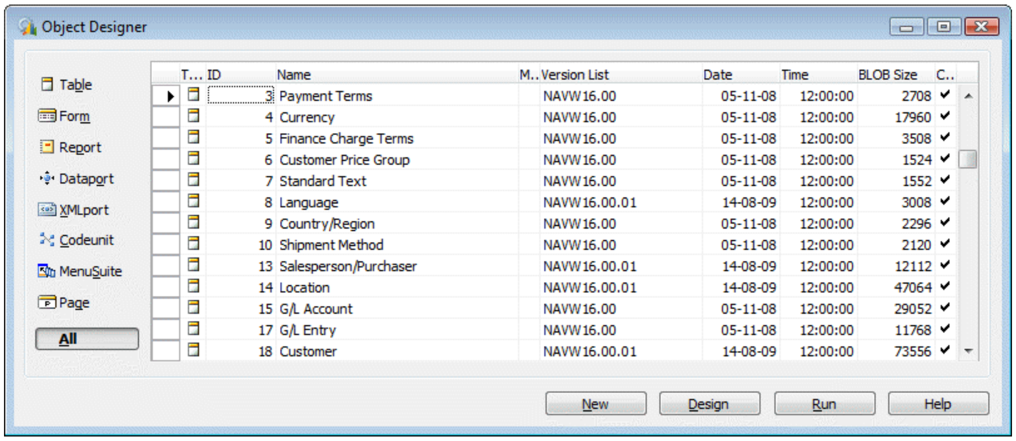
This is a "purchaser's" Role Center. As you can see, all the information needed by this person in the organization is in one place and usable in a workflow-like way. Also, the **Sales Orders** are accessible from the main menu. It is completely different to the menu found in version 5.0 or before



However, the Role Tailored concept is not new. Dynamics NAV partners have been implementing it for many years. In the classic menu, it was extremely easy to create new menus and most companies implemented their own menus per role. When the 'Microsoft Outlook' style Menu Suites where introduced in version 4.0, end users could create shortcut Menu Suites and these also quickly became role centers. You can clearly see that the role tailored concept is like coming home for Dynamics NAV.

The building blocks

To understand the development examples in this book, we will discuss some of the basic building blocks of Microsoft Dynamics NAV 2009.



T...	ID	Name	M..	Version List	Date	Time	BLOB Size	C..
	3	Payment Terms		NAVW16.00	05-11-08	12:00:00	2708	✓
	4	Currency		NAVW16.00	05-11-08	12:00:00	17960	✓
	5	Finance Charge Terms		NAVW16.00	05-11-08	12:00:00	3508	✓
	6	Customer Price Group		NAVW16.00	05-11-08	12:00:00	1524	✓
	7	Standard Text		NAVW16.00	05-11-08	12:00:00	1552	✓
	8	Language		NAVW16.00.01	14-08-09	12:00:00	3008	✓
	9	Country/Region		NAVW16.00	05-11-08	12:00:00	2296	✓
	10	Shipment Method		NAVW16.00	05-11-08	12:00:00	2120	✓
	13	Salesperson/Purchaser		NAVW16.00.01	14-08-09	12:00:00	12112	✓
	14	Location		NAVW16.00.01	14-08-09	12:00:00	47064	✓
	15	G/L Account		NAVW16.00	05-11-08	12:00:00	29052	✓
	17	G/L Entry		NAVW16.00	05-11-08	12:00:00	11768	✓
	18	Customer		NAVW16.00.01	14-08-09	12:00:00	73556	✓

Like all database applications, it starts with tables. They contain all the information displayed in a structured way. It is important to understand that the tables of Microsoft Dynamics NAV are not completely normalized. The tables are structured in the way the user interface works. This makes it easy for non- technical people to understand the data model. We'll discuss the unique structure of the application in the next chapter.

Tables, however, not only contain data, they contain business logic as well. As they are structured like the functionality in the database, tables contain simple functions like address validation, and more complex functions for VAT and discount calculation.

Whenever functionality gets more complex or can be shared across the application, it is better to move them to the Codeunit object. These are containers of business logic for a special purpose.

For the user interface there are three object types: Forms, reports, and pages. The first and latter are intended for user input. Reports are originally intended to be printed on paper but with the current status of technology, they are more and more used as information dashboards combining management information with drill-through possibilities.

Forms and pages are tightly linked to each other. Each form object has a page object with the same number and name. The form object is used in the 'Classic Client' only whilst the pages are used in the 'Role Tailored Client'.

The report object is used in both interfaces but has two layouts—a black and white layout for the Classic client and a RDLC layout for the Role Tailored client that supports colors and graphs.

As the tables are structured in the way the application works, the forms and pages are bound to one table. For people new to this concept, it sometimes takes a while to get used to this.

The Menu Suite defines the way the navigation is structured when people leave their Role Centers, or when using the Classic Client which does not support Role Centers.

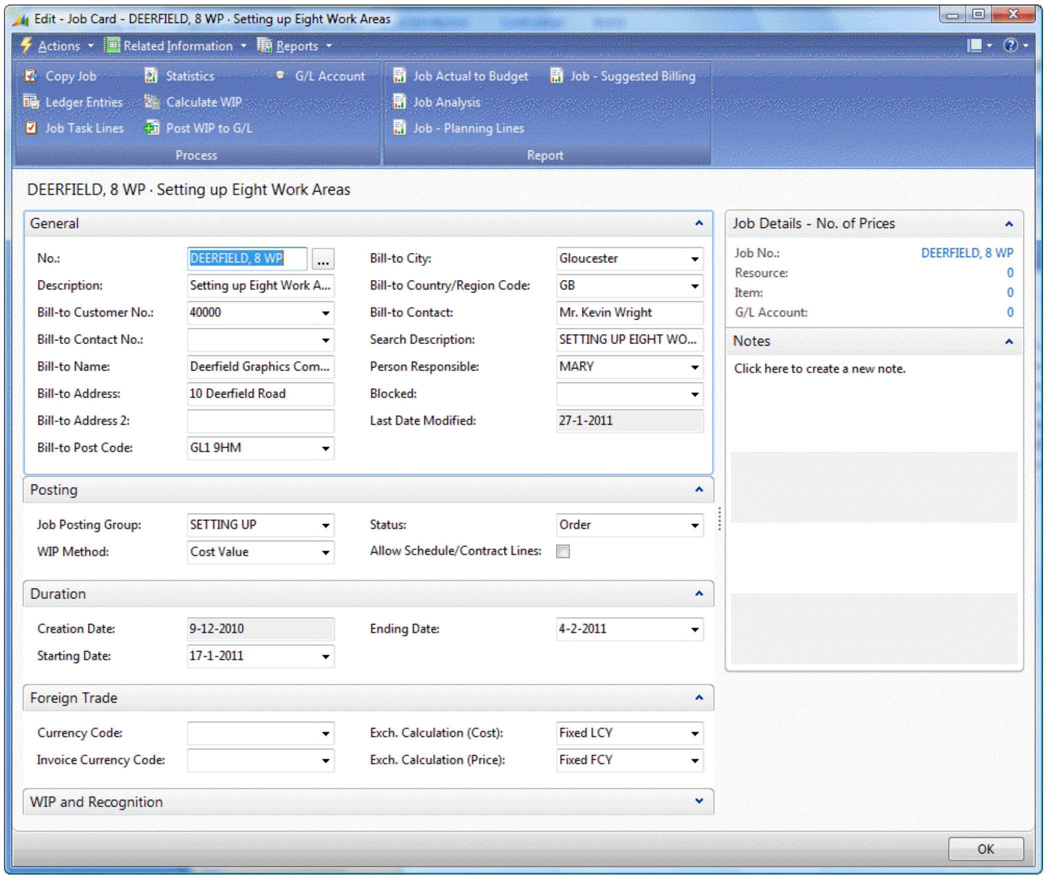
The last two object types are external interfacing objects. Data ports and XML ports make it possible to import and export data in and out of the system.

For this book, the table and page objects are the most important to understand. Most of this book, however, can also be applied to older versions but then forms should be applied wherever this book addresses pages.

Tables as user interface and business logic

The table object in Microsoft Dynamics NAV is very important. As it is not normalized, it contains a lot of information about how the database works.

For example the Job Card (88) is built on one table, the Job (167). This table contains all fields required for this screen.



DEERFIELD, 8 WP · Setting up Eight Work Areas

General

No.:	DEERFIELD, 8 WP	Bill-to City:	Gloucester
Description:	Setting up Eight Work A...	Bill-to Country/Region Code:	GB
Bill-to Customer No.:	40000	Bill-to Contact:	Mr. Kevin Wright
Bill-to Contact No.:		Search Description:	SETTING UP EIGHT WO...
Bill-to Name:	Deerfield Graphics Com...	Person Responsible:	MARY
Bill-to Address:	10 Deerfield Road	Blocked:	
Bill-to Address 2:		Last Date Modified:	27-1-2011
Bill-to Post Code:	GL1 9HM		

Posting

Job Posting Group:	SETTING UP	Status:	Order
WIP Method:	Cost Value	Allow Schedule/Contract Lines:	☐

Duration

Creation Date:	9-12-2010	Ending Date:	4-2-2011
Starting Date:	17-1-2011		

Foreign Trade

Currency Code:		Exch. Calculation (Cost):	Fixed LCY
Invoice Currency Code:		Exch. Calculation (Price):	Fixed FCY

WIP and Recognition

Job Details - No. of Prices

Job No.:	DEERFIELD, 8 WP
Resource:	0
Item:	0
G/L Account:	0

Notes

Click here to create a new note.

In a traditional development environment this screen would have a transaction `GetJobData` and `UpdateJobData`. These transactions would read the information from the database, map them to the screen, and save the information in the database if the user if finished. However, in Microsoft Dynamics NAV, all fields that are displayed in the interface are stored in one table. This makes it possible for the screen to have built-in triggers to get the data and update the database.

The table object then contains the business logic required for this document. Let's have a look at some of the fields in this table.

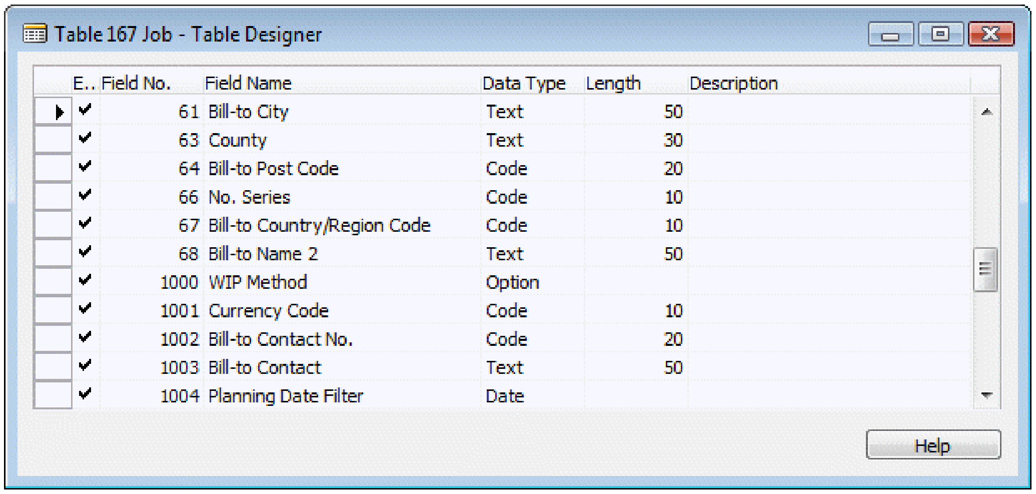
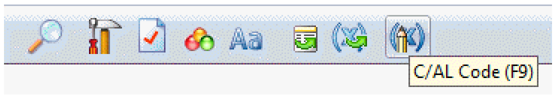


Table 167 Job - Table Designer

E..	Field No.	Field Name	Data Type	Length	Description
✓	61	Bill-to City	Text	50	
✓	63	County	Text	30	
✓	64	Bill-to Post Code	Code	20	
✓	66	No. Series	Code	10	
✓	67	Bill-to Country/Region Code	Code	10	
✓	68	Bill-to Name 2	Text	50	
✓	1000	WIP Method	Option		
✓	1001	Currency Code	Code	10	
✓	1002	Bill-to Contact No.	Code	20	
✓	1003	Bill-to Contact	Text	50	
✓	1004	Planning Date Filter	Date		

In this table you will see see a lot of fields that are required for a Job like **WIP Method**, **Currency Code**, and so on. But when we click on the **C/AL Code** icon and focus on Currency Code we get this.



```
Currency Code - OnValidate()
IF "Currency Code" <> xRec."Currency Code" THEN
  IF NOT JobLedgEntryExist THEN
    CurrencyUpdatePlanningLines
  ELSE
    ERROR(Text000,FIELDCAPTION("Currency Code"),TABLECAPTION);
```

It contains business logic that gets executed every time something happens with this field. In this case, the currency factor is recalculated and updated in the Sales Lines.

So, the tables in Microsoft Dynamics NAV are not just data containers, they are the foundation for both the business logic and the application workflow.

Dynamics NAV in throughout supply chain

The Dynamics NAV product is used almost everywhere in the business supply chain. This is mainly because it is a highly customizable ERP system. Dynamics NAV is used in the classical supply chain companies like manufacturing plants, wholesale companies, and in retail with or without many changes. But with an add-on, the product is also used in transportation companies or in the recycling industry.

In order to understand this better, it is important to know how companies work. A company is a person or a group of persons using materials and resources to deliver a product or a service to other companies or end consumers. A group of companies working together is called a supply chain. Dynamics NAV can be used in all these companies although it is traditionally used in companies with 5 to 250 concurrent users.

In order to serve this process, Dynamics NAV has a list of basic modules:

- **Financial management**: Traditionally, financial management was used in companies to comply with federal regulations of bookkeeping. For entrepreneurs starting their business, this is usually the part they least like. However, good bookkeeping can give a clear view on the company's well being and support strategic decisions with good financial information.
- **Inventory:** Every company that grows will reach a certain point where it is no longer possible to handle inventory without a system. Keeping too much inventory is expensive. A good inventory system can help you keep your stock management as efficient as possible.
- **Relationship management:** When it comes to people, a company is not only dealing with customers and vendors. RM will help you keep track of every company and person your company is dealing with.
- **Sales:** The sales process is usually the place where businesses make money. The system will help you keep track of orders that your customers place.
- **Purchasing:** The purchasing department is usually split in two pieces. One piece is the purchasing of goods the company needs for itself. This facility management can grow into a business of its own at large companies. The other purchasing part is buying the materials and resources you need for your sales process. For some trading companies, this can even be a drop shipment process where you never have the purchased goods in house.
- **Warehouse management:** Warehouses are getting bigger and bigger, making the need for a system that supports the picking and put-away process even greater. This is usually tightly connected to the sales and purchasing process.

- **Manufacturing:** When you make products yourself, you need a system that helps you create a new item from one or more purchased materials and resources.
- **Jobs:** In some companies the process of delivering a service is so complex that it requires its own administration process. Time and billing is usually a very important process for these companies.
- **Service management:** This supports the service process handling warranty and necessary periodical maintenance of your items.

Some basics

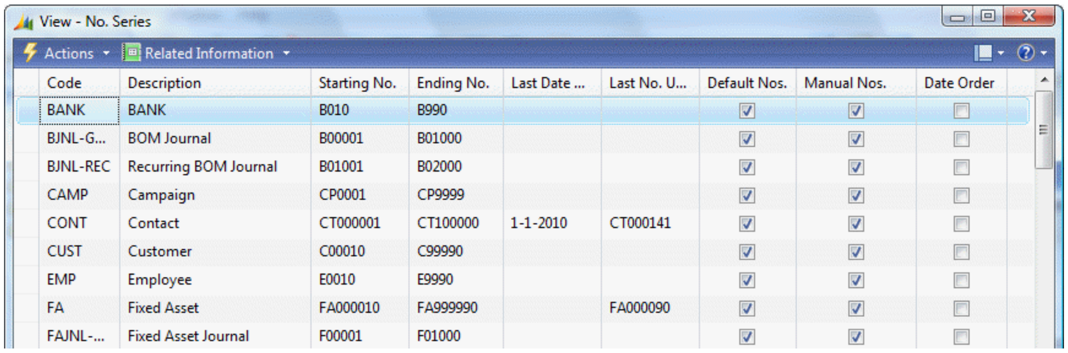
Microsoft Dynamics NAV has some basic structures that are reused throughout the application and are necessary to understand before you read the rest of this book.

Number series

Databases need unique records. The application has two ways of making this happen.

Some tables have automatic incremental numbering that cannot be influenced. These are often accounting tables that have auditable purposes. Examples of these tables are G/L entries, G/L registers, and VAT entries.

The other way is using a flexible alphanumeric code. In some setup tables, users are free to create their own numbers, like in the location table, but most of the time, number series functionality is used. These can be influenced by the end user depending on their access rights. Let's have a closer look at those:



Code	Description	Starting No.	Ending No.	Last Date ...	Last No. U...	Default Nos.	Manual Nos.	Date Order
BANK	BANK	B010	B990			✓	✓	☐
BJNL-G...	BOM Journal	B00001	B01000			✓	✓	☐
BJNL-REC	Recurring BOM Journal	B01001	B02000			✓	✓	☐
CAMP	Campaign	CP0001	CP9999			✓	✓	☐
CONT	Contact	CT000001	CT100000	1-1-2010	CT000141	✓	✓	☐
CUST	Customer	C00010	C99990			✓	✓	☐
EMP	Employee	E0010	E9990			✓	✓	☐
FA	Fixed Asset	FA000010	FA999990		FA000090	✓	✓	☐
FAJNL-...	Fixed Asset Journal	F00001	F01000			✓	✓	☐

Users can define their own numbering, usually starting with an alphanumeric character. Numbering can be done automatically, manually, or in a combination. Numbers can have a starting date and incremental number. This way you can number your Sales Invoices SI11-0001. **SI** means **Sales Invoice**. 11 means 2011 and 0001 is the incremental number.

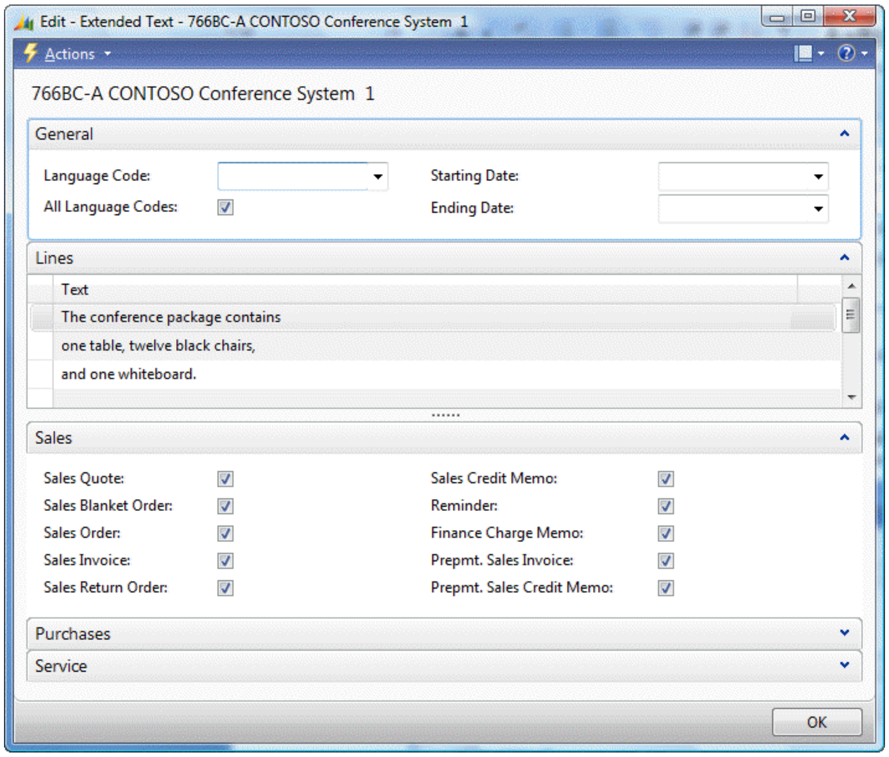
Number series can be linked to each other making it possible to have a different number series for national and international customers.

Extended text

Most master data tables in Microsoft Dynamics NAV have two description fields but it is possible to add extra text.

The text can be defined for all languages in the system and valid for a specific period.

We can enable or disable using the text for most documents available in the system, so we can have a long text for the **Sales Quote** and a shorter text for the **Sales Invoice**.



766BC-A CONTOSO Conference System 1

General

Language Code:		Starting Date:	
All Language Codes:	✓	Ending Date:	

Lines

Text
The conference package contains
one table, twelve black chairs,
and one whiteboard.

Sales

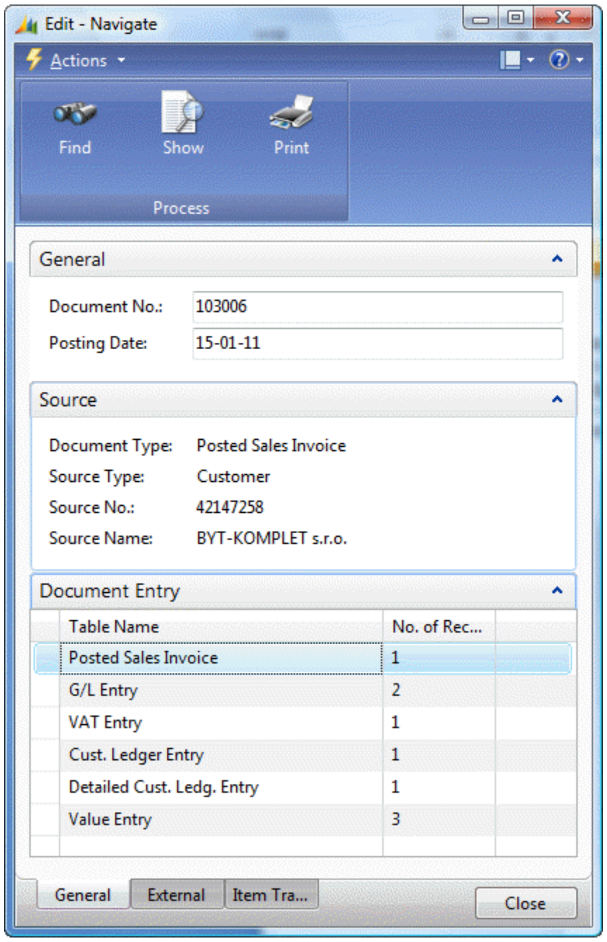
Sales Quote:	✓	Sales Credit Memo:	✓
Sales Blanket Order:	✓	Reminder:	✓
Sales Order:	✓	Finance Charge Memo:	✓
Sales Invoice:	✓	Prepmt. Sales Invoice:	✓
Sales Return Order:	✓	Prepmt. Sales Credit Memo:	✓

Purchases

Service

Navigate

The main reason Microsoft Dynamics NAV consultants like you to use numbers as SI11-0001 is the Navigate functionality. This functionality makes it possible to find all information in the database linked to this document. If you were to call your Sales Invoice 110001 and your Purchase Invoice the same, the system would not be able to find the correct information.



Document No.:	103006
Posting Date:	15-01-11

Document Type:	Posted Sales Invoice
Source Type:	Customer
Source No.:	42147258
Source Name:	BYT-KOMPLET s.r.o.

Table Name	No. of Rec...
Posted Sales Invoice	1
G/L Entry	2
VAT Entry	1
Cust. Ledger Entry	1
Detailed Cust. Ledg. Entry	1
Value Entry	3

When Navigating on **Posted Sales Invoice 103006** in the CRONUS Demo database, we get all the information that is linked to this number.

Navigation shows both documents and entries. Using the **Show** option, we can drill down into the records and go even deeper into the information.

Setup tables

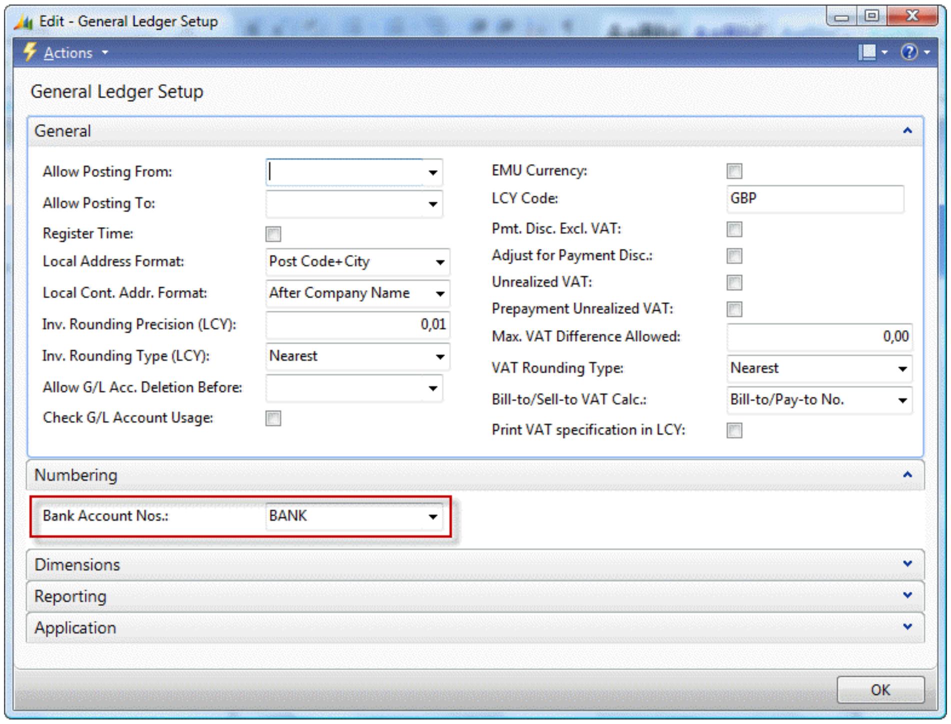
An ERP application can be used in many different ways and to make it work in the way we want, we need to set it up correctly. We already discussed that Dynamics NAV has far less setup work required than other ERP packages and is more likely to be changed but nonetheless, there is setup work to do.

Every part of the application has its own setup table. There are also some application-wide or cross application setup tables. During the implementation, we need to make sure to touch all of these tables. Changing these setups after the implementation should be done with great care.

This list shows all Microsoft Dynamics NAV setup tables grouped by type.

Specific setup tables	**Application wide setup tables**
General ledger setup	Source code setup
Sales & receivables setup	Change log setup
Purchases & payables setup	SMTP mail setup
Inventory setup	Approval setup
Resources setup	Job queue setup
Jobs setup	Online map setup
Marketing setup	Interaction template setup
Human resources setup	Employee portal setup
Production schedule setup	Notification setup
FA setup	Order promising setup
Nonstock item setup	BizTalk management setup
Warehouse setup	
Service mgt. setup	
Manufacturing setup	

When we open a setup from the application, we see several options, including the numbering we discussed earlier.



Allow Posting From:		EMU Currency:	☐
Allow Posting To:		LCY Code:	GBP
Register Time:	☐	Pmt. Disc. Excl. VAT:	☐
Local Address Format:	Post Code+City	Adjust for Payment Disc.:	☐
Local Cont. Addr. Format:	After Company Name	Unrealized VAT:	☐
Inv. Rounding Precision (LCY):	0,01	Prepayment Unrealized VAT:	☐
Inv. Rounding Type (LCY):	Nearest	Max. VAT Difference Allowed:	0,00
Allow G/L Acc. Deletion Before:		VAT Rounding Type:	Nearest
Check G/L Account Usage:	☐	Bill-to/Sell-to VAT Calc.:	Bill-to/Pay-to No.
		Print VAT specification in LCY:	☐

Bank Account Nos.: BANK

Posting groups

Microsoft Dynamics NAV is very flexible in its posting to the **General Ledger**. This is set up in posting groups. These form a matrix which is filtered out by the application.

Most application areas have one or more posting group tables:

- Customer posting group
- Vendor posting group
- Inventory posting group
- Job posting group
- Gen. Business posting group
- Gen. Product posting group
- Bank account posting group
- VAT business posting group

- VAT product posting group
- FA posting group

We'll discuss posting groups in more detail in *Chapter 3, Financial Management*.

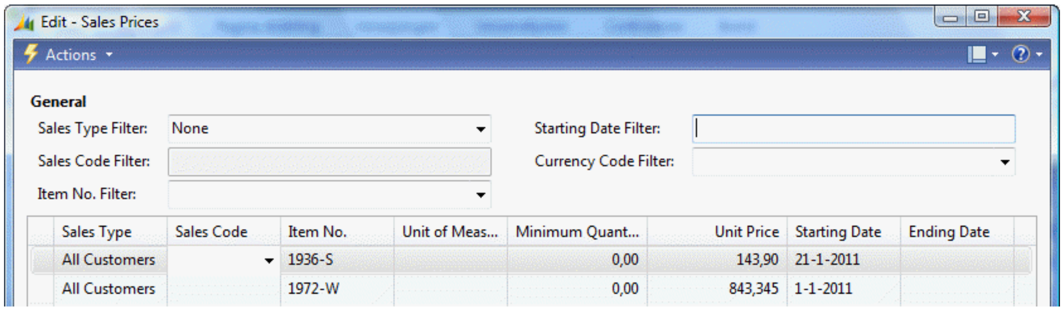
Pricing

When it comes to pricing and discounts, Microsoft Dynamics NAV has a very simple, yet effective way of calculating prices.

All sales and purchase prices are stored in four simple tables.

- 7002—Sales Price
- 7004—Sales Line Discount
- 7012—Purchase Price
- 7014—Purchase Line Discount

The system finds the appropriate price by filtering down in these tables. The narrower the filter, the more likely the price is applied.



General

Sales Type Filter:	None
Sales Code Filter:	
Item No. Filter:	
Starting Date Filter:	
Currency Code Filter:	

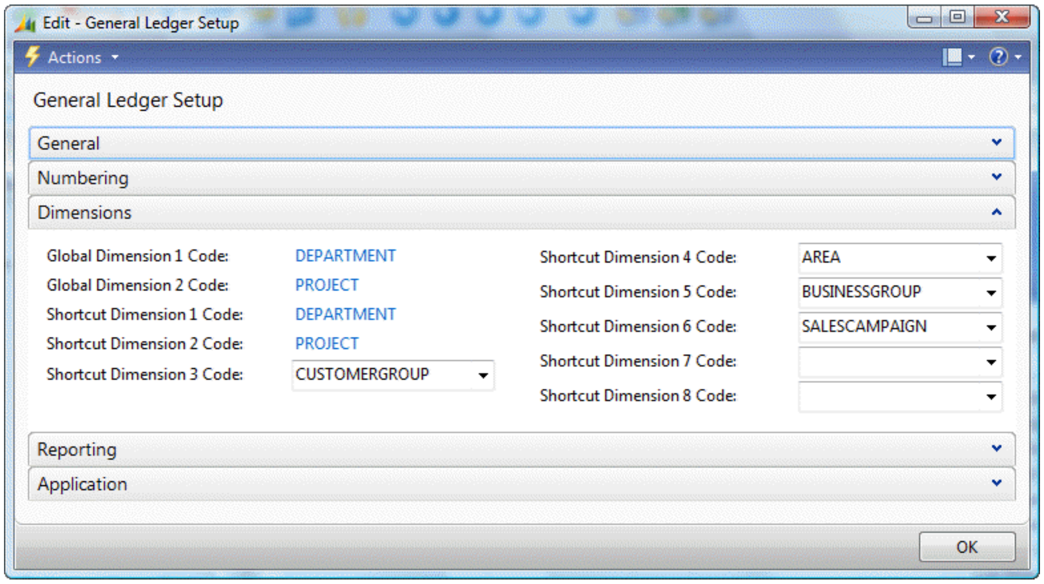
Sales Type	Sales Code	Item No.	Unit of Meas...	Minimum Quant...	Unit Price	Starting Date	Ending Date
All Customers		1936-S		0,00	143,90	21-1-2011	
All Customers		1972-W		0,00	843,345	1-1-2011	

For example: The normal price of item **1972-W** on the item card is 974,80 but from **1-1-2011** it is **843,345**.

The filtering is done in Codeunits Sales Price Calc. Mgt. (7000) and Purch. Price Calc. Mgt. (7010). We'll discuss this structure in *Chapter 2, An Example Application* where we will also create such a structure for our own application.

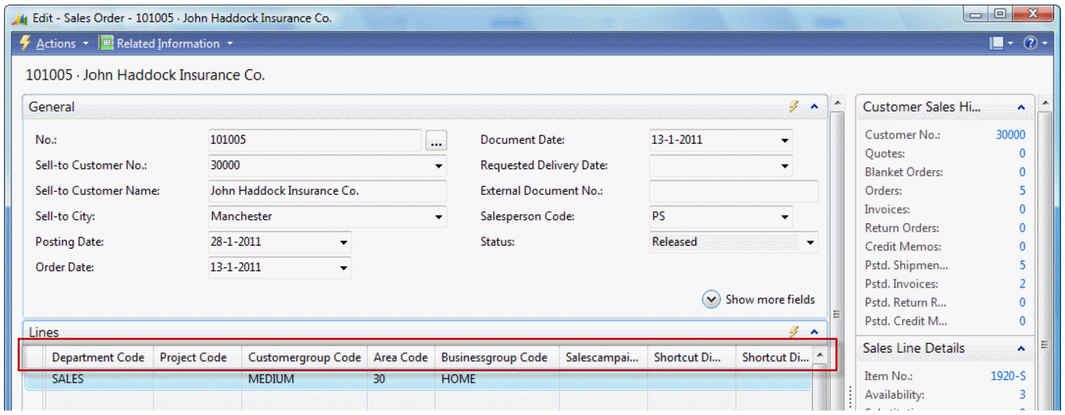
Dimensions

Throughout the application, an unlimited number of dimensions can be used to analyze the data. These dimensions are inherited from master data tables.



Global Dimension 1 Code:	DEPARTMENT	Shortcut Dimension 4 Code:	AREA
Global Dimension 2 Code:	PROJECT	Shortcut Dimension 5 Code:	BUSINESSGROUP
Shortcut Dimension 1 Code:	DEPARTMENT	Shortcut Dimension 6 Code:	SALESCAMPAIGN
Shortcut Dimension 2 Code:	PROJECT	Shortcut Dimension 7 Code:	
Shortcut Dimension 3 Code:	CUSTOMERGROUP	Shortcut Dimension 8 Code:	

The application has two global dimensions that are directly posted into each transaction. Six other dimensions can be defined as shortcut dimensions to be directly used in journals and documents. An unlimited number of additional dimensions can be added but need to be accessed with additional effort.



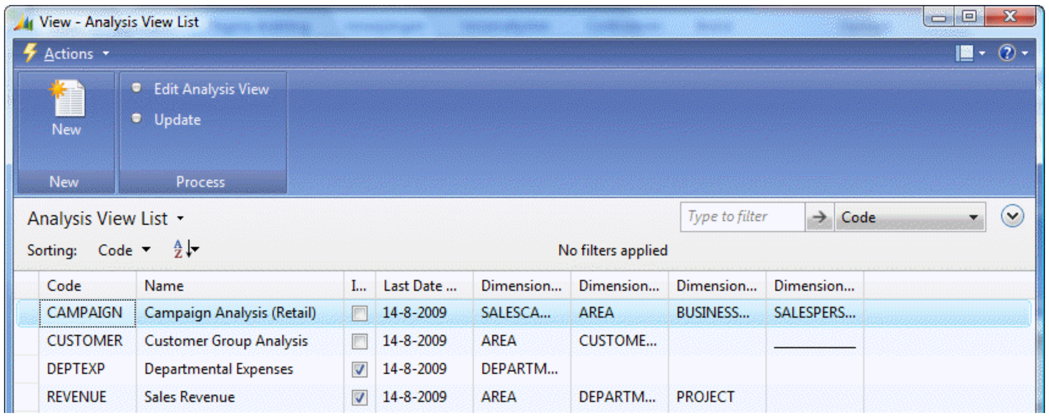
No.:	101005	Document Date:	13-1-2011
Sell-to Customer No.:	30000	Requested Delivery Date:	
Sell-to Customer Name:	John Haddock Insurance Co.	External Document No.:	
Sell-to City:	Manchester	Salesperson Code:	PS
Posting Date:	28-1-2011	Status:	Released
Order Date:	13-1-2011		

Department Code	Project Code	Customergroup Code	Area Code	Businessgroup Code	Salescampai...	Shortcut Di...	Shortcut Di...
SALES		MEDIUM	30	HOME			

Customer No.:	30000
Quotes:	0
Blanket Orders:	0
Orders:	5
Invoices:	0
Return Orders:	0
Credit Memos:	0
Pstd. Shipmen...	5
Pstd. Invoices:	2
Pstd. Return R...	0
Pstd. Credit M...	0
Item No.:	1920-S
Availability:	3

This screenshot shows how Global and Shortcut Dimensions can be used in a Sales Document.

As discussed earlier, Microsoft Dynamics NAV has built-in OLAP possibilities. It allows us to create cubes to be analyzed within the application or in SQL Server analysis services. .



Code	Name	I...	Last Date ...	Dimension...	Dimension...	Dimension...	Dimension...
CAMPAIGN	Campaign Analysis (Retail)	☐	14-8-2009	SALESCA...	AREA	BUSINESS...	SALESPERS...
CUSTOMER	Customer Group Analysis	☐	14-8-2009	AREA	CUSTOME...		
DEPTEXP	Departmental Expenses	✓	14-8-2009	DEPARTM...			
REVENUE	Sales Revenue	✓	14-8-2009	AREA	DEPARTM...	PROJECT	

Although the cubes can be updated real time during posting, it is highly recommended to update them periodically in a batch. Also, the number of dimensions has an impact on the performance of the system.

Data model principles

Microsoft Dynamics NAV has some specific data model principles that are very important to understand before you can create your own structure. The building blocks are layered and reused and rely on each other in order to secure data integrity.

Master data

The data model starts with master data. There are three types or levels of master data. They are all used in transactions. We differentiate helpers, core, and umbrella master data.

Examples of helper master data are Currencies, Locations, and Payment terms. They often do not use a number series but allow us to create our own unique codes.

Examples of core master data are G/L Accounts, Customers, Vendors, Items, Resources, and Fixed assets. They are numbered using number series and have their own journal structure.

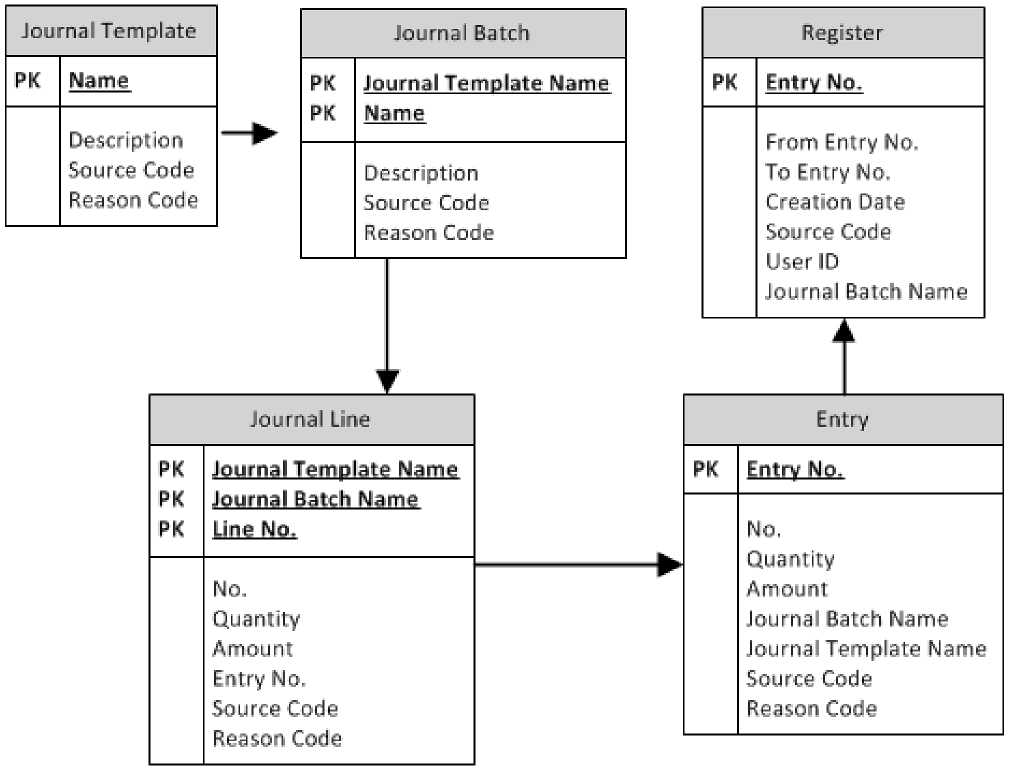
Umbrella master data consists of data tables such as Contacts, Jobs, and Production orders. They allow us to group other master data and documents.

The combination of all above combine the information allows us to quickly analyse the created data.

Journals

Every transaction starts with a journal. Each journal can contain a number of sub transactions that are treated by the system as one. This way the system is able to check, for example, if the integrity of the system is maintained after the transaction is completed.

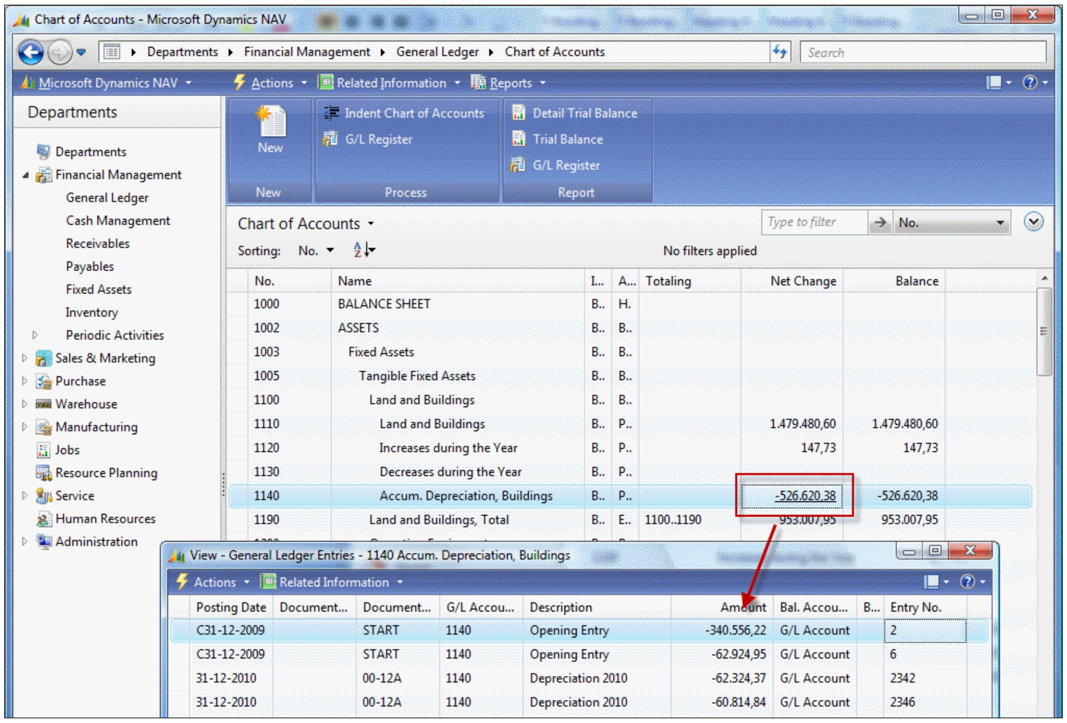
This diagram shows how a journal is structured. **PK** means **Primary Key** which is the unique identifier of the table.



Every journal can contain one or more templates with one or more batches, allowing multiple users to have multiple templates and batches. A journal line has a source number field that refers to, for example, the G/L Account number or the Item number we are changing. When we post the journal, the changes are stored in the entry table and a register is maintained for all the lines for the journal allowing auditors to check if the transactions are consistent.

The general ledger

To see how this works in the application we can best go to the **Chart of Accounts** and the **General Journals**.

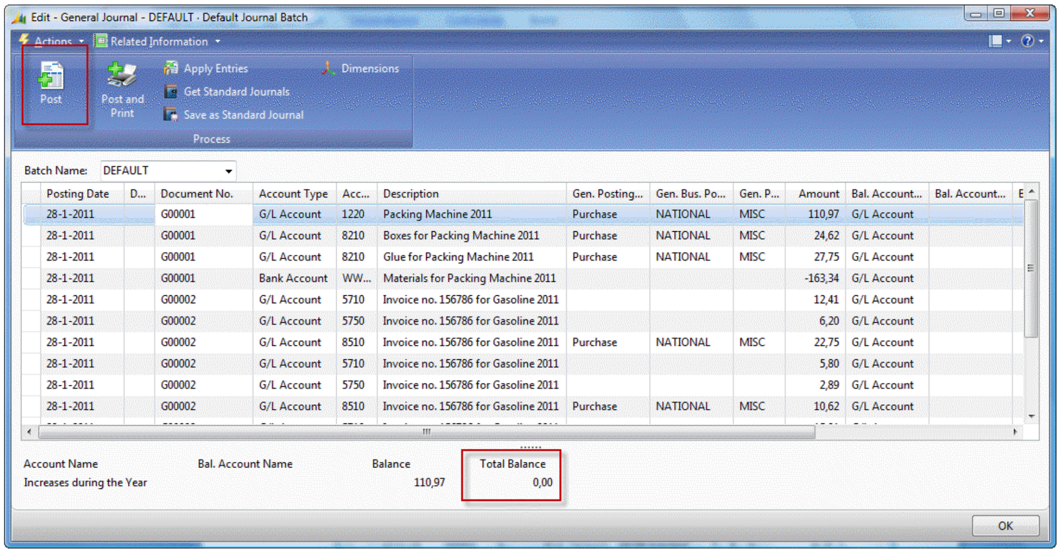


No.	Name	I...	A...	Totaling	Net Change	Balance
1000	BALANCE SHEET	B..	H.			
1002	ASSETS	B..	B..			
1003	Fixed Assets	B..	B..			
1005	Tangible Fixed Assets	B..	B..			
1100	Land and Buildings	B..	B..			
1110	Land and Buildings	B..	P..		1.479.480,60	1.479.480,60
1120	Increases during the Year	B..	P..		147,73	147,73
1130	Decreases during the Year	B..	P..			
1140	Accum. Depreciation, Buildings	B..	P..		-526.620,38	-526.620,38
1190	Land and Buildings, Total	B..	E..	1100..1190	953.007,95	953.007,95

Posting Date	Document...	Document...	G/L Accou...	Description	Amount	Bal. Accou...	B...	Entry No.
C31-12-2009		START	1140	Opening Entry	-340.556,22	G/L Account		2
C31-12-2009		START	1140	Opening Entry	-62.924,95	G/L Account		6
31-12-2010		00-12A	1140	Depreciation 2010	-62.324,37	G/L Account		2342
31-12-2010		00-12A	1140	Depreciation 2010	-60.814,84	G/L Account		2346

If we select **G/L Account 1140** and drill down, we see the details of this record.

These are created through journals, so let's open a journal.

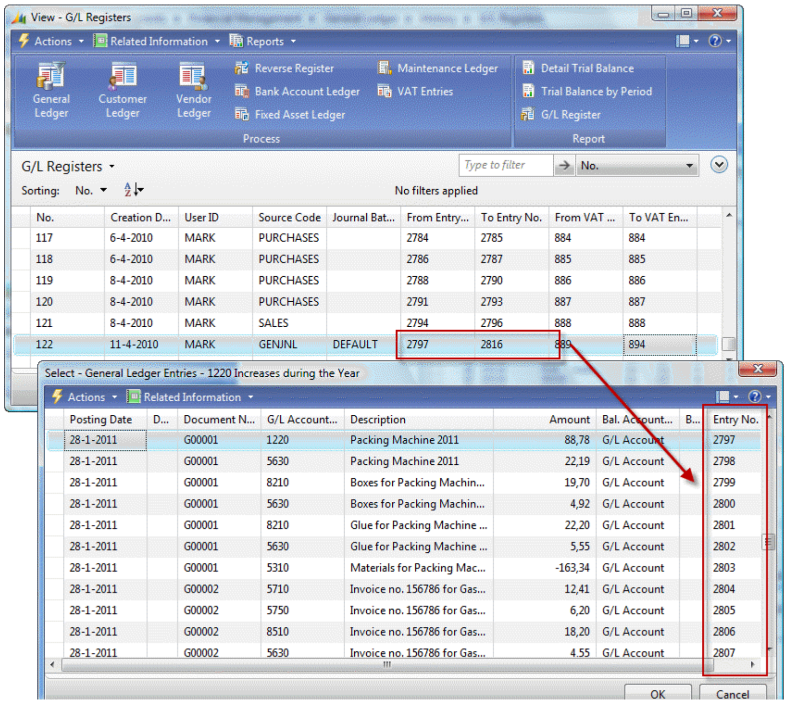


Batch Name: DEFAULT

Posting Date	D...	Document No.	Account Type	Acc...	Description	Gen. Posting...	Gen. Bus. Po...	Gen. P...	Amount	Bal. Account...	Bal. Account...
28-1-2011		G00001	G/L Account	1220	Packing Machine 2011	Purchase	NATIONAL	MISC	110,97	G/L Account	
28-1-2011		G00001	G/L Account	8210	Boxes for Packing Machine 2011	Purchase	NATIONAL	MISC	24,62	G/L Account	
28-1-2011		G00001	G/L Account	8210	Glue for Packing Machine 2011	Purchase	NATIONAL	MISC	27,75	G/L Account	
28-1-2011		G00001	Bank Account	WW...	Materials for Packing Machine 2011				-163,34	G/L Account	
28-1-2011		G00002	G/L Account	5710	Invoice no. 156786 for Gasoline 2011				12,41	G/L Account	
28-1-2011		G00002	G/L Account	5750	Invoice no. 156786 for Gasoline 2011				6,20	G/L Account	
28-1-2011		G00002	G/L Account	8510	Invoice no. 156786 for Gasoline 2011	Purchase	NATIONAL	MISC	22,75	G/L Account	
28-1-2011		G00002	G/L Account	5710	Invoice no. 156786 for Gasoline 2011				5,80	G/L Account	
28-1-2011		G00002	G/L Account	5750	Invoice no. 156786 for Gasoline 2011				2,89	G/L Account	
28-1-2011		G00002	G/L Account	8510	Invoice no. 156786 for Gasoline 2011	Purchase	NATIONAL	MISC	10,62	G/L Account	

Account Name	Bal. Account Name	Balance	Total Balance
Increases during the Year		110,97	0,00

This journal contains two documents on the same posting date and the balance is zero. When we post this journal, the system will create the ledger entries and a register.



G/L Registers

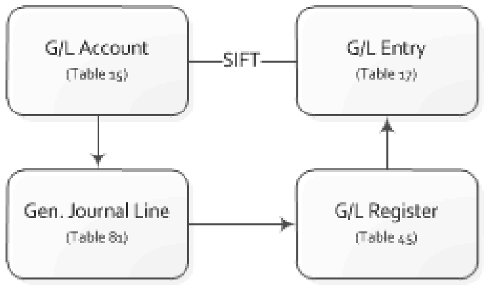
Sorting: No. No filters applied

No.	Creation D...	User ID	Source Code	Journal Bat...	From Entry...	To Entry No.	From VAT ...	To VAT En...
117	6-4-2010	MARK	PURCHASES		2784	2785	884	884
118	6-4-2010	MARK	PURCHASES		2786	2787	885	885
119	8-4-2010	MARK	PURCHASES		2788	2790	886	886
120	8-4-2010	MARK	PURCHASES		2791	2793	887	887
121	8-4-2010	MARK	SALES		2794	2796	888	888
122	11-4-2010	MARK	GENJNL	DEFAULT	2797	2816	889	894

Select - General Ledger Entries - 1220 Increases during the Year

Posting Date	D...	Document N...	G/L Account...	Description	Amount	Bal. Account...	B...	Entry No.
28-1-2011		G00001	1220	Packing Machine 2011	88,78	G/L Account		2797
28-1-2011		G00001	5630	Packing Machine 2011	22,19	G/L Account		2798
28-1-2011		G00001	8210	Boxes for Packing Machin...	19,70	G/L Account		2799
28-1-2011		G00001	5630	Boxes for Packing Machin...	4,92	G/L Account		2800
28-1-2011		G00001	8210	Glue for Packing Machine ...	22,20	G/L Account		2801
28-1-2011		G00001	5630	Glue for Packing Machine ...	5,55	G/L Account		2802
28-1-2011		G00001	5310	Materials for Packing Mac...	-163,34	G/L Account		2803
28-1-2011		G00002	5710	Invoice no. 156786 for Gas...	12,41	G/L Account		2804
28-1-2011		G00002	5750	Invoice no. 156786 for Gas...	6,20	G/L Account		2805
28-1-2011		G00002	8510	Invoice no. 156786 for Gas...	18,20	G/L Account		2806
28-1-2011		G00002	5630	Invoice no. 156786 for Gas...	4,55	G/L Account		2807

This is the basic building block for Dynamics NAV. Everything in Dynamics NAV is built on top of a journal, registers, and entries.



Balancing

In any ERP system, totaling and balancing is crucial, whether you are totaling the general ledger, customer payments or inventory, it is important to know the balance of each Account, Customer, or Item.

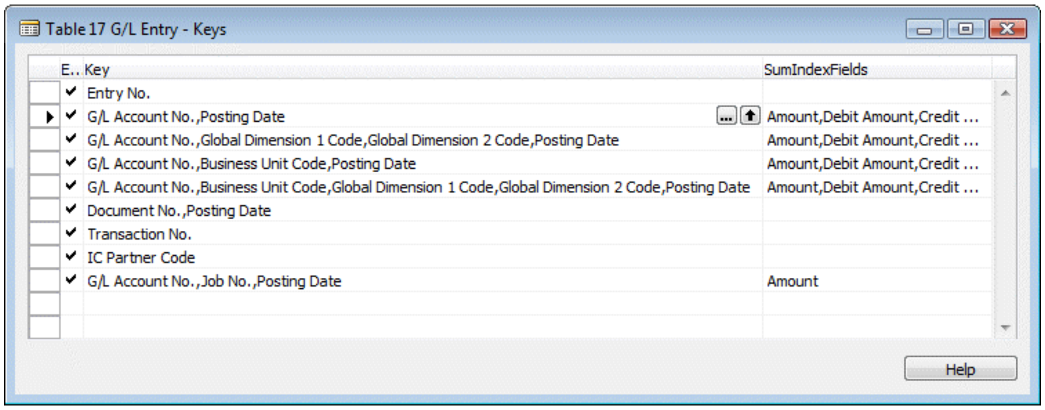
Traditionally, this requires calculating these balances and deciding a place to store the totals and subtotals. Not in Dynamics NAV. The system has built-in technology that will handle balancing and totaling for you, without effort and cost of performance.

This built-in technology is called **Sum Index Flow Technology**, **SIFT** in short. For Dynamics NAV it is the key feature to its success.

The way it works is that, as a developer, you define your totaling on an index level. By associating the totaling fields with a key, the system knows that it has to maintain the totals for you.

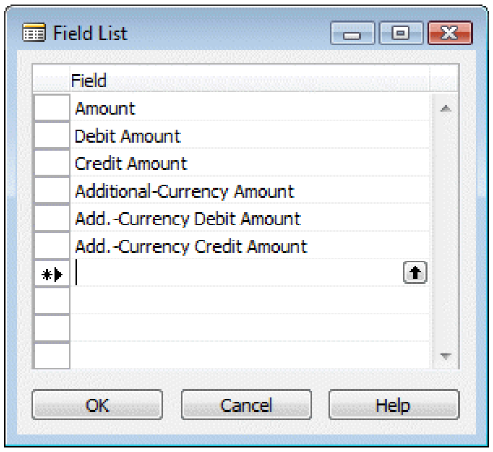
In the original proprietary database, this technique was built-in and invisible for the user but in the SQL Server database, we can see how it works.

If we go into the CRONUS database and open the G/L Entry table with its keys ,we see this information.



E..	Key	SumIndexFields
✓	Entry No.	
✓	G/L Account No.,Posting Date	Amount,Debit Amount,Credit ...
✓	G/L Account No.,Global Dimension 1 Code,Global Dimension 2 Code,Posting Date	Amount,Debit Amount,Credit ...
✓	G/L Account No.,Business Unit Code,Posting Date	Amount,Debit Amount,Credit ...
✓	G/L Account No.,Business Unit Code,Global Dimension 1 Code,Global Dimension 2 Code,Posting Date	Amount,Debit Amount,Credit ...
✓	Document No.,Posting Date	
✓	Transaction No.	
✓	IC Partner Code	
✓	G/L Account No.,Job No.,Posting Date	Amount

Let's take key number two as an example. The key contains the fields G/L Account number and Posting Date. If we take a closer look at the **SumIndexFields** column, we see the following fields listed.



Field
Amount
Debit Amount
Credit Amount
Additional-Currency Amount
Add.-Currency Debit Amount
Add.-Currency Credit Amount

Notice that these are all fields of type decimal. This is mandatory for `SumIndexfields`.

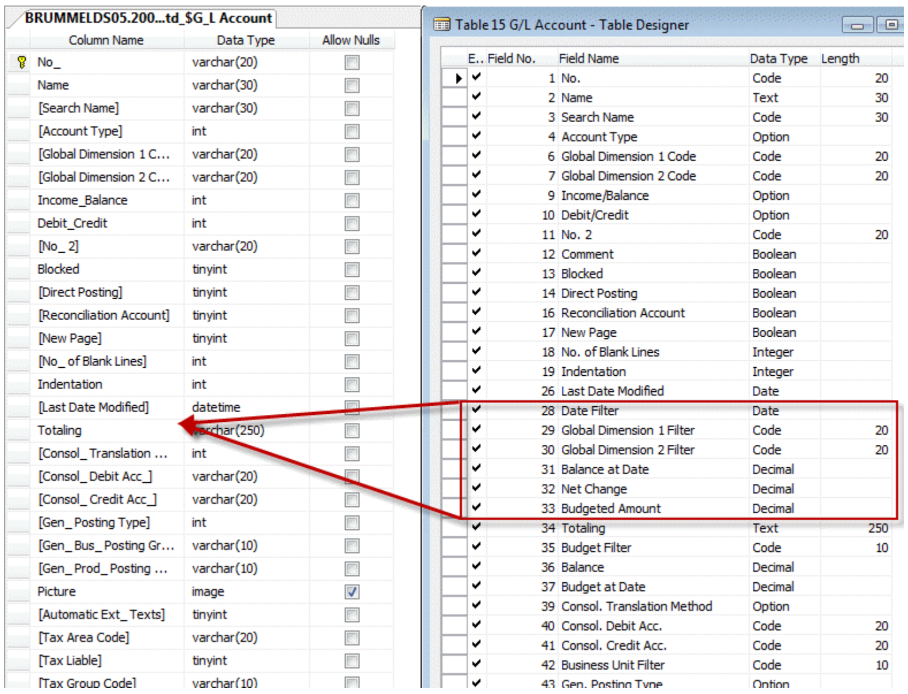
From the SQL Server Management studio you can see the generated data from the `SumIndexField` definition. Each key with a `SumIndexField` generates a view in the database. In older versions (prior to 5 SP 1) the `SumIndexFields` are saved in tables.

So now we know that we do not have to worry about maintaining the totals, we can spend our time on what's really important.

Flow fields and flow filters

As discussed earlier, screens in Microsoft Dynamics NAV are built directly on one table. These table definitions contain all fields including the totals. However, these totals are not real database fields.

This can be illustrated by comparing the table definition in Microsoft Dynamics NAV to the table definition in the SQL Server.



BRUMMELDS05.200...td_$G_L Account

Column Name	Data Type	Allow Nulls
No_	varchar(20)	☐
Name	varchar(30)	☐
[Search Name]	varchar(30)	☐
[Account Type]	int	☐
[Global Dimension 1 C...	varchar(20)	☐
[Global Dimension 2 C...	varchar(20)	☐
Income_Balance	int	☐
Debit_Credit	int	☐
[No_ 2]	varchar(20)	☐
Blocked	tinyint	☐
[Direct Posting]	tinyint	☐
[Reconciliation Account]	tinyint	☐
[New Page]	tinyint	☐
[No_ of Blank Lines]	int	☐
Indentation	int	☐
[Last Date Modified]	datetime	☐
Totaling	varchar(250)	☐
[Consol_ Translation ...	int	☐
[Consol_ Debit Acc_]	varchar(20)	☐
[Consol_ Credit Acc_]	varchar(20)	☐
[Gen_ Posting Type]	int	☐
[Gen_ Bus_ Posting Gr...	varchar(10)	☐
[Gen_ Prod_ Posting ...	varchar(10)	☐
Picture	image	✓
[Automatic Ext_ Texts]	tinyint	☐
[Tax Area Code]	varchar(20)	☐
[Tax Liable]	tinyint	☐
[Tax Group Code]	varchar(10)	☐

Table 15 G/L Account - Table Designer

E..	Field No.	Field Name	Data Type	Length
✓	1	No.	Code	20
✓	2	Name	Text	30
✓	3	Search Name	Code	30
✓	4	Account Type	Option	
✓	6	Global Dimension 1 Code	Code	20
✓	7	Global Dimension 2 Code	Code	20
✓	9	Income/Balance	Option	
✓	10	Debit/Credit	Option	
✓	11	No. 2	Code	20
✓	12	Comment	Boolean	
✓	13	Blocked	Boolean	
✓	14	Direct Posting	Boolean	
✓	16	Reconciliation Account	Boolean	
✓	17	New Page	Boolean	
✓	18	No. of Blank Lines	Integer	
✓	19	Indentation	Integer	
✓	26	Last Date Modified	Date	
✓	28	Date Filter	Date	
✓	29	Global Dimension 1 Filter	Code	20
✓	30	Global Dimension 2 Filter	Code	20
✓	31	Balance at Date	Decimal	
✓	32	Net Change	Decimal	
✓	33	Budgeted Amount	Decimal	
✓	34	Totaling	Text	250
✓	35	Budget Filter	Code	10
✓	36	Balance	Decimal	
✓	37	Budget at Date	Decimal	
✓	39	Consol. Translation Method	Option	
✓	40	Consol. Debit Acc.	Code	20
✓	41	Consol. Credit Acc.	Code	20
✓	42	Business Unit Filter	Code	10
✓	43	Gen. Posting Type	Option	

The fields **Date Filter** (28) to **Budgeted Amount** (33) are not actual fields in the database. They are helper fields to show data on screens.

Flow filters can have seven types; Sum, Average, Exist, Count, Min, Max, and Lookup and contain a Query to the database. For example, Balance at Date (31) shows:

```
Sum("G/L Entry".Amount
  WHERE (G/L Account No.=FIELD(No.),
         G/L Account No.=FIELD(FILTER(Totaling)),
         Business Unit Code=FIELD(Business Unit Filter),
         Global Dimension 1 Code=FIELD(Global Dimension 1 Filter),
         Global Dimension 2 Code=FIELD(Global Dimension 2 Filter),
         Posting Date=FIELD(UPPERLIMIT(Date Filter))))
```

This creates the Sum of the field Amount in the G/L Entry table (17) filtering on G/L Account, G/L Account No., Business Unit Code, Global Dimension 1 & 2 Code, and Posting Date.

Some of these filters are actual fields in the G/L Account table, but others are Flow filters. Non-existing fields that can be used as a runtime filter to limit the result of the Query.

We will use and discuss more of these Flow filters and Flow fields in this book.

More journals and entries

So now that we know how a journal works, it might be interesting to build a posting diagram of Dynamics NAV. Dynamics NAV has a number of journals, registers, and entries built on top of each other.

These are the most important journals, registers and entries:

Journals	Gen. Journal Line (81)
	Item Journal Line (83)
	Res. Journal Line (207)
	Job Journal Line (210)
Registers	G/L Register (45)
	Item Register (46)
	Resource Register (240)
	Job Register (241)

Entries	G/L Entry (17)
	Cust. Ledger Entry (21)
	Vendor Ledger Entry (25)
	Item Ledger Entry (32)
	Job Ledger Entry (169)
	Res. Ledger Entry (203)
	VAT Entry (254)
	Bank Account Ledger Entry (271)

Please notice that when you look in the database you'll find more of these tables, but these are the main building blocks.

Each journal is responsible for creating its own entries but may run another journal if that is required. For example, an Item Journal may generate G/L entries if required using a General Journal and a Job Journal may create Item Ledger Entries using the Item Journal.

We already discussed the G/L Entry table which is used to store the basic financial information. This is the basic administration table.

The other entry tables are sub ledger tables. They store redundant information but have extra information for their specific use. A total of a sub ledger should always balance with the G/L. We'll see how that works in *Chapter 3, Financial Management*.

- The Customer and Vendor ledger entry tables are used to store specific information about the accounts receivables. They are linked to Customer and Vendor master data tables.
- The VAT Entry table stores specific information to make registration easier. Most companies do monthly or quarterly VAT registrations with one or more governmental agencies.

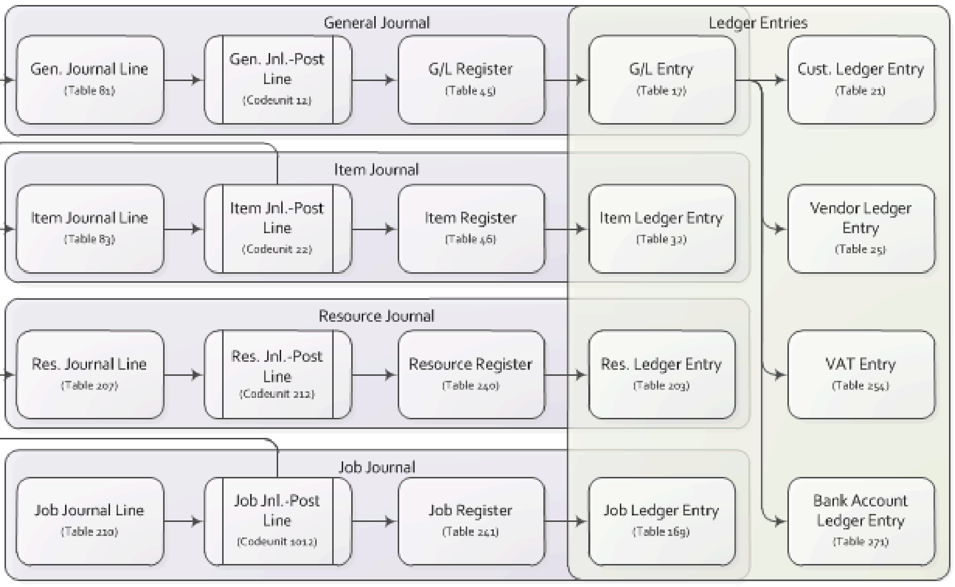
  VAT is different in many countries and could be different from what this book describes in localized country systems.
- The Bank Account entries should show exactly what transactions were carried out on our bank accounts.

The logistical part of the ERP package is handled by the Item Journal. Every item that is purchased, produced, or sold is handled though this journal. Services are handled through the Resource journal. A 'Resource' can either be a person or a piece of equipment, for example a lift.

The Job journal is an umbrella overlaying the entire application. It allows you to group transactions making it easier to analyze cost and profit for larger projects.

Posting Schema

When we combine all this information in a schema, we can create the following basic Microsoft Dynamics NAV posting schema.


General Journal
Ledger Entries
Gen. Journal Line (Table 81)
Gen. Jnl.-Post Line (Codeunit 12)
G/L Register (Table 45)
G/L Entry (Table 17)
Cust. Ledger Entry (Table 21)
Item Journal
Item Journal Line (Table 83)
Item Jnl.-Post Line (Codeunit 22)
Item Register (Table 46)
Item Ledger Entry (Table 32)
Vendor Ledger Entry (Table 25)
Resource Journal
Res. Journal Line (Table 207)
Res. Jnl.-Post Line (Codeunit 212)
Resource Register (Table 240)
Res. Ledger Entry (Table 203)
VAT Entry (Table 254)
Job Journal
Job Journal Line (Table 210)
Job Jnl.-Post Line (Codeunit 1012)
Job Register (Table 241)
Job Ledger Entry (Table 169)
Bank Account Ledger Entry (Table 271)

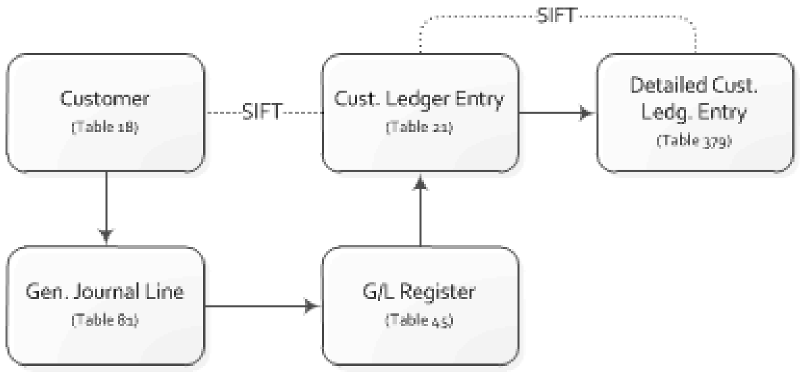
Here you can clearly see what journal is responsible for creating what entry. An entry table is always maintained by one process.

The **General Journal** is the heart of the application where the basic financial information is created in the ledger entries. All the basic information is in the G/L entry table which is grouped in the G/L Register which is always balanced. The **Customer**, **Vendor**, **VAT**, and **Bank Account Ledger** entries are sub tables that always refer to a G/L register. We can never create one of these entries without touching this part of the application.

Sub and detailed entries

When an entry is created, its basic structure should not be changed for audit ability. This is why most entries in Microsoft Dynamics NAV have sub- or detailed entries.

The Customer and Vendor Ledger Entry have details for application, unrealized loss and gain, various discounts, and corrections. This way we are able to keep track of what happens with an entry without changing the original information.



The Item Ledger Entries a have wide variety of sub entries depending on what you are doing with the items.

One of the most important tables in Microsoft Dynamics NAV is the Value Entry table. Each Item Ledger Entry has one or more of these. This table is the 'soft bridge' between the inventory and the financial part of the application.

Warehouse entries enable moving items within our organization without touching the basic inventory or financial application.

Documents—combining the journals into processes

The journal and entry tables make it possible for us to do the basic balancing in our company but people in companies are not used to working with journals.

Traditionally, companies work with documents. This was also the case before ERP applications were introduced. A sales representative would travel through the country with a paper order block and then come back to the back office. The back office then ships the orders with shipping documents and invoices.

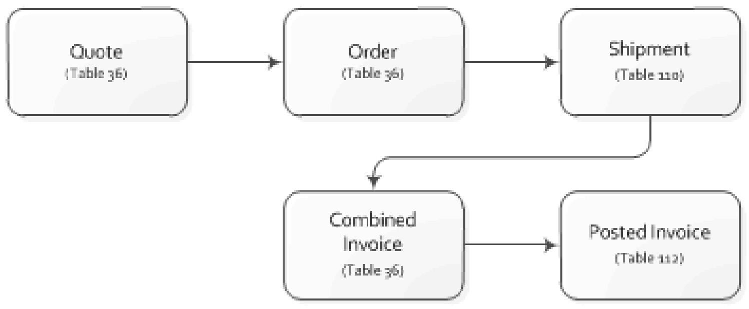
Microsoft Dynamics NAV supports working with documents. Traditionally, we divide the documents in sales and purchasing documents but the later versions of Microsoft Dynamics NAV also have warehouse documents. Other supported documents are reminders and service documents.

Document structure

A document in Microsoft Dynamics NAV always has a header and lines. The header contains the basic information about the transaction like shipment dates, addresses, and payment terms.

The lines contain information about what is sold or purchased. This can be a variety of G/L accounts, items, and resources.

A document can have different stages depending on the type of the transaction. A quote is a typical starting point in the sales or purchasing process. When a quote is approved it can be promoted to an order which is then shipped and invoiced. The process can be also reversed via a return order resulting in a credit memo.



Document transactions

Transactions in the database can be started via documents. When a document is processed the necessary journals are automatically populated. For example, when an order is shipped the goods leave the warehouse, thus an Item Journal is created and posted to handle this. When the invoice is posted, a General Journal is generated to create G/L Entries and a Customer or Vendor Ledger Entries.

Other structures

The previously discussed structure with journals and documents is by far the most important transaction structure. But Microsoft Dynamics NAV has other structures as well.

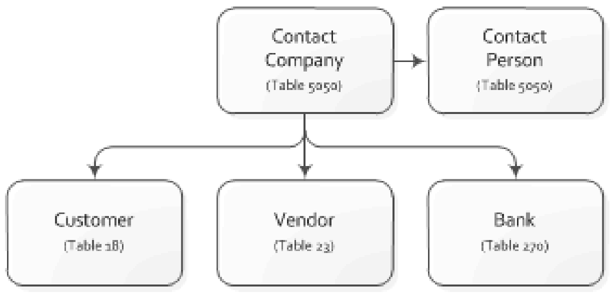
The three most important other structures are CRM, Jobs, and Manufacturing. These structures are all 'umbrella' structures for other processes.

Relationship management

Microsoft Dynamics NAV RM helps you to maintain master data and analyze transactional data. It is both at the very start of the data process and at the end.

We have already seen the Customer, Vendor, and Bank master data records. But what if a Vendor is also a customer or vice versa. We don't want to maintain the same data twice. We might also want to keep extra information of our customers and vendors like contact persons and their interests. We'll see more of that in the RM chapter later.

There is also a need to analyze the data we have created with the document and journal structure.

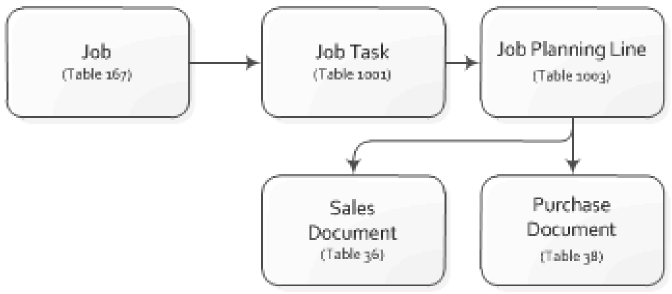


Jobs

Sometimes a project can be more comprehensive than just a purchase and/or a sales document. A project can take from several weeks to over a year and requires multiple documents.

The job structure in Dynamics NAV allows you to handle this. Every document and journal transaction can be attached to a job making it easy to analyze profit and loss, and even schedule your jobs.

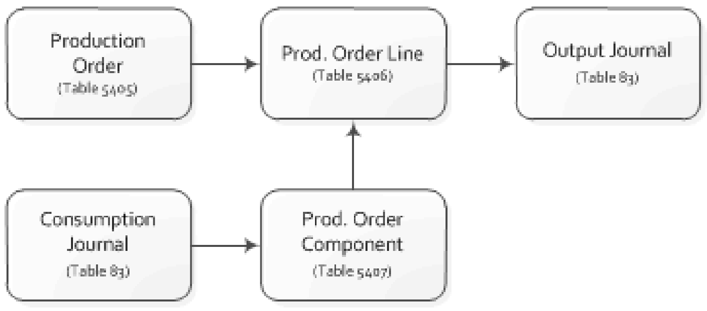
The jobs module also allows you to do a calculation before you start the project and balance this calculation throughout the process.


Job (Table 167)
Job Task (Table 1001)
Job Planning Line (Table 1003)
Sales Document (Table 36)
Purchase Document (Table 38)

Manufacturing

When you produce your own items, you have different needs in your ERP process than when you only purchase the items you sell.

The manufacturing module of Microsoft Dynamics NAV allows you to handle this process. Basically what it does is create an item out of one of the other items and resources.


Production Order (Table 5405)
Prod. Order Line (Table 5406)
Output Journal (Table 83)
Consumption Journal (Table 83)
Prod. Order Component (Table 5407)

Summary

In this chapter, we have covered the basic structure of Microsoft Dynamics NAV. We talked about the design philosophy, application objects, and the unique table structure. We discussed the role tailored concept and its reflection to older versions of the product. We talked about some basic functions of the product-like number series and application setup. Very important is the basic posting structure and the way SIFT works. We discussed how the document structure is overlaying the journal structure and how the umbrella structure is on top of that. In the next chapter, we will look at a sample industry application and its effect on the standard functionality.

2
A Sample Application

Let's create a structure of our own in Microsoft Dynamics NAV. To do this we must think of something that is not already available in the standard package but can be built on top of it.

For our example application we will run a squash court. Running a squash court is simple to understand but something we cannot do without changing and expanding the product. In order to define our changes we first need to make a fit-gap analysis.

After this chapter you will have a better understanding of how to reuse the framework of the Microsoft Dynamics NAV application. We will show how to reverse engineer the application and study its functionality by going into the application code.

For this example, some new and changed objects are required. The *Appendix* describes where to find the objects, and how to install and activate them.

In the first part we will look at how to reverse engineer the standard application to look and learn how it works, and how to reuse the structures in our own solutions.

In the second part of the chapter we will learn how to use the journals and entries in a custom application.

Lastly, we will look at how to integrate our solution with the standard application, in our case Sales Invoicing.

Fit-gap analysis

When we do a fit-gap analysis we look at the company's processes and define what we can and cannot do with the standard package. When a business process can be handled with the standard software we call this a **Fit**. When it cannot be done it's a **Gap**, we can fill a gap by developing a solution or purchasing an add-on.

But even when something could be done with standard software features, it does not necessarily mean that doing this is wise. The standard application should be used for what it was designed for. Using standard features for something else might work in the current version but if it changes in a new version it might no longer fit. For this reason it is better to design something new instead of wrongly using the standard features.

Designing a Squash Court application

The basic process of a squash court company is renting the courts to squash players; both members and non-members. There is a reservation and invoicing process handling different rates for members and non-members.

Although this could be implemented using items to represent squash courts and customers to represent players this would be a typical example of using standard features wrongly. Instead of doing this we will look at how items and customers are designed and use this to create a new Squash Court application.

Look, learn, and love

To determine the design for this application we will first look at the parts of the standard application (we could have used) to learn how they work. We will use this knowledge in our own design.

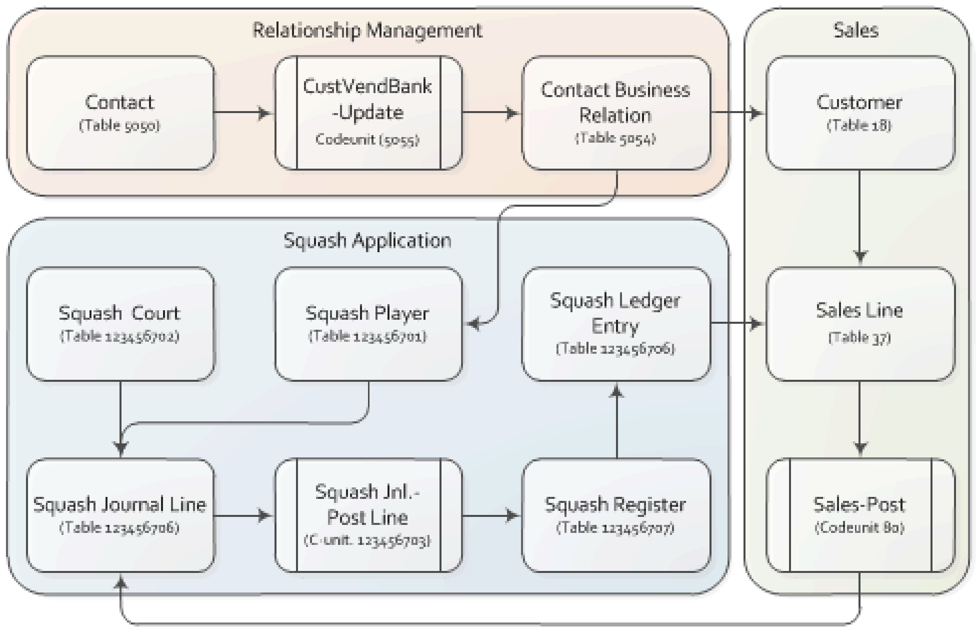
In Microsoft Dynamics NAV, customer and vendor master data are maintained using Relationship Management (RM). For our solution we will create a new master data for squash players being the business part of application. This will also be integrated with RM.

To design the Squash Court we will look at the design of items in the standard package. The Squash Court will be the product part of our application having a journal to create reservation entries, which we can invoice.

For this invoicing process we will use and integrate with the Sales part of Microsoft Dynamics NAV.

Drawing the table and posting schema

After we have decided what the design of our application will be, we can draw the tables and post the routines as we did in the previous chapter. This will clarify the design for others and guide us through the development process.


Relationship Management
Contact (Table 5050)
CustVendBank -Update Codeunit (5055)
Contact Business Relation (Table 5054)
Sales
Customer (Table 18)
Squash Application
Squash Court (Table 123456702)
Squash Player (Table 123456701)
Squash Ledger Entry (Table 123456706)
Sales Line (Table 37)
Squash Journal Line (Table 123456706)
Squash Jnl.-Post Line (C-unit. 123456703)
Squash Register (Table 123456707)
Sales-Post (Codeunit 80)

The objects in Relationship Management and Sales are standard objects that we will possibly need to modify. The objects for the Squash Application are new objects but based on similar objects in the standard application.

The Project approach

In order to keep track of our project we'll cut the changes into smaller tasks. The first task will be to do the changes in Relationship Management to be able to create a squash player from a contact. The second task is to create squash courts. The reservation and invoice processes are tasks three and four.

Interfacing with the standard application

In our schema we can see that we have two processes where we need to touch the standard Microsoft Dynamics NAV processes, which are Relationship Management and Sales.

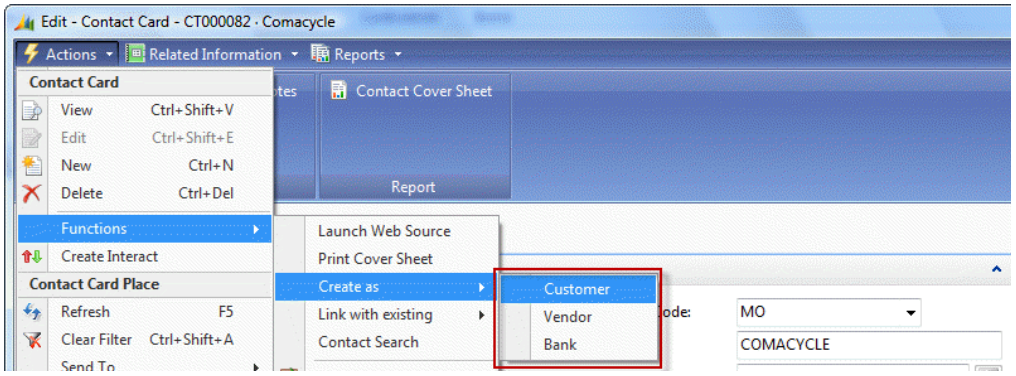
Getting started

In the first part of the design process we will look at how to reverse engineer the standard application in order to learn and reuse the knowledge in our own solution.

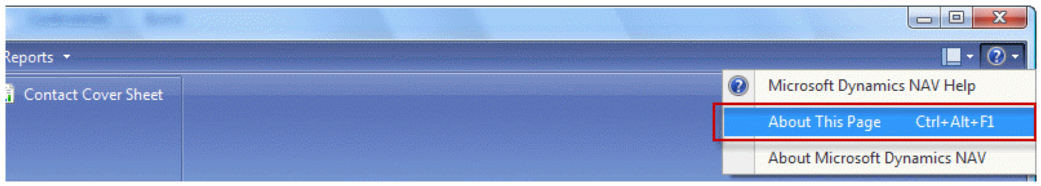
Creating squash players

For our squash players administration we will use the data from the Contact table. In the standard product it is possible to create a customer or vendor with the contact data. We require the same functionality to create squash players so let's have a look at how this is done by Microsoft.

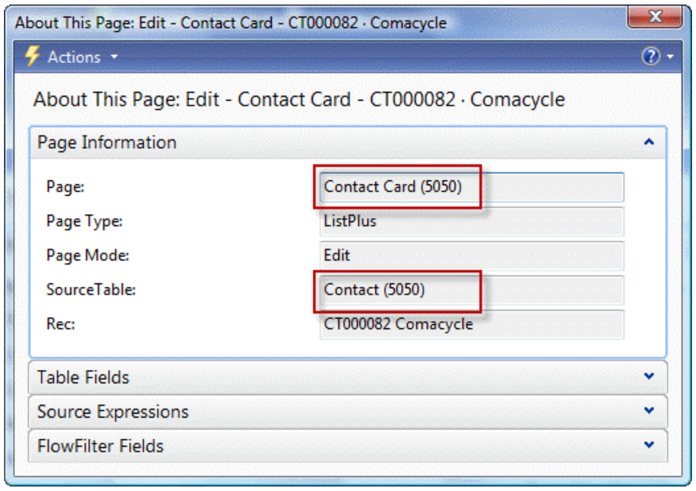
Open the contact card and try to find this function (as shown in the following screenshot):



We want a function like this for our squash players. So let's get in and see what it does. For this we need to design the page and look at the actions. The page number in this case is 5050 which we can find by clicking on **About This Page** in the upright corner of the page as shown in the following screenshot:


Reports
Contact Cover Sheet
Microsoft Dynamics NAV Help
About This Page Ctrl+Alt+F1
About Microsoft Dynamics NAV

This option can be useful for finding information about the fields that are not on the page, the filters or the source table:

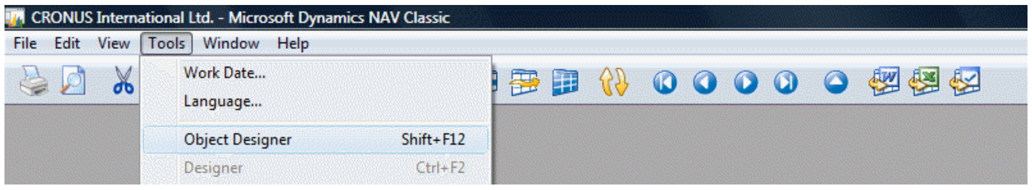


About This Page: Edit - Contact Card - CT000082 · Comacycle

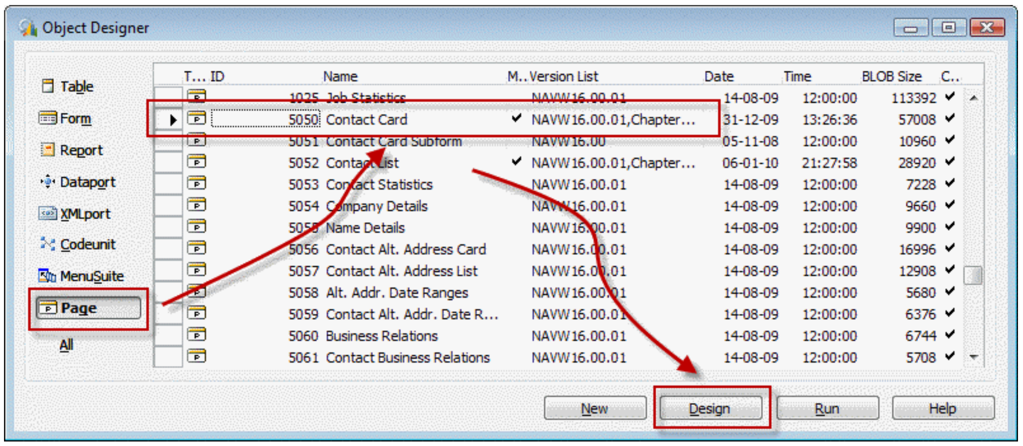
Page Information

Page:	Contact Card (5050)
Page Type:	ListPlus
Page Mode:	Edit
SourceTable:	Contact (5050)
Rec:	CT000082 Comacycle

To open the page we need to open the **Object Designer** (*Shift F12)* in the Classic client as shown in the following screenshot:

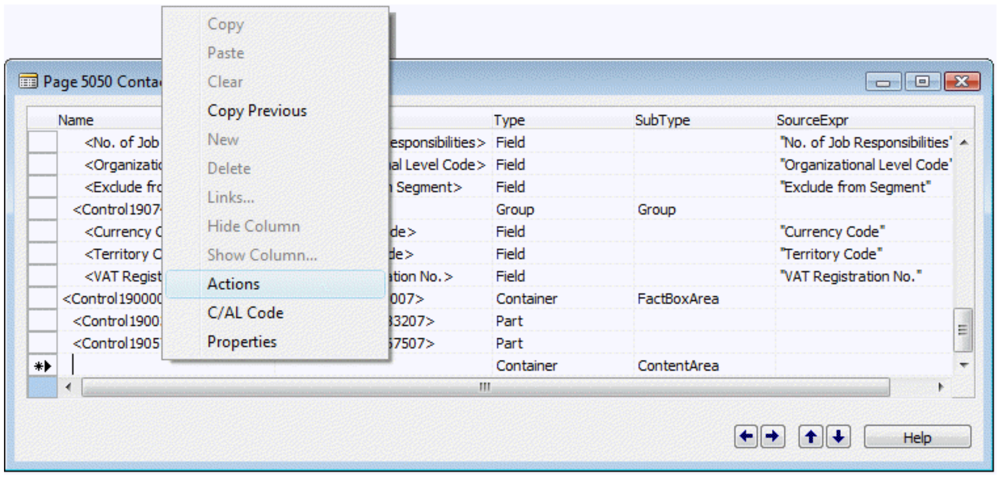

CRONUS International Ltd. - Microsoft Dynamics NAV Classic
File
Edit
View
Tools
Window
Help
Work Date...
Language...
Object Designer Shift+F12
Designer Ctrl+F2

Here in the **Object Designer** we can find page 5050, **Contact Card**, as shown:



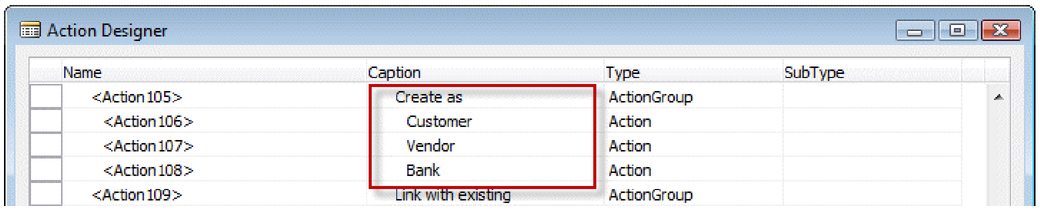
T...	ID	Name	M..	Version List	Date	Time	BLOB Size	C..
	1025	Job Statistics		NAVW16.00.01	14-08-09	12:00:00	113392	✓
	5050	Contact Card	✓	NAVW16.00.01,Chapter...	31-12-09	13:26:36	57008	✓
	5051	Contact Card Subform		NAVW16.00	05-11-08	12:00:00	10960	✓
	5052	Contact List	✓	NAVW16.00.01,Chapter...	06-01-10	21:27:58	28920	✓
	5053	Contact Statistics		NAVW16.00.01	14-08-09	12:00:00	7228	✓
	5054	Company Details		NAVW16.00.01	14-08-09	12:00:00	9660	✓
	5055	Name Details		NAVW16.00.01	14-08-09	12:00:00	9900	✓
	5056	Contact Alt. Address Card		NAVW16.00.01	14-08-09	12:00:00	16996	✓
	5057	Contact Alt. Address List		NAVW16.00.01	14-08-09	12:00:00	12908	✓
	5058	Alt. Addr. Date Ranges		NAVW16.00.01	14-08-09	12:00:00	5680	✓
	5059	Contact Alt. Addr. Date R...		NAVW16.00.01	14-08-09	12:00:00	6376	✓
	5060	Business Relations		NAVW16.00.01	14-08-09	12:00:00	6744	✓
	5061	Contact Business Relations		NAVW16.00.01	14-08-09	12:00:00	5708	✓

We are looking for the **Actions** on this page. They are kind of difficult to find if you are unfamiliar with the Page Designer. To open the **Actions**, the cursor should be on the blank line below the last populated line. Then press the *right mouse* button and **Actions** or select **Actions** from the **View** drop down menu:



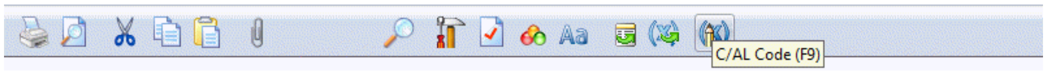
Name		Type	SubType	SourceExpr
<No. of Job	esponsibilities>	Field		"No. of Job Responsibilities"
<Organizatio	al Level Code>	Field		"Organizational Level Code"
<Exclude fr	Segment>	Field		"Exclude from Segment"
<Control1907		Group	Group	
<Currency C	de>	Field		"Currency Code"
<Territory C	de>	Field		"Territory Code"
<VAT Regist	tion No.>	Field		"VAT Registration No."
<Control190000	007>	Container	FactBoxArea	
<Control1900	3207>	Part		
<Control1905	7507>	Part		
		Container	ContentArea	

Now we are in the **Action Designer** and we can search for the **Create as** part.



Name	Caption	Type	SubType
<Action105>	Create as	ActionGroup	
<Action106>	Customer	Action	
<Action107>	Vendor	Action	
<Action108>	Bank	Action	
<Action109>	Link with existing	ActionGroup	

To see what it does we need to go into the C/AL code by pressing *F9* or by selecting **C/AL Code** from the **View** drop down menu as shown in the following screenshot:

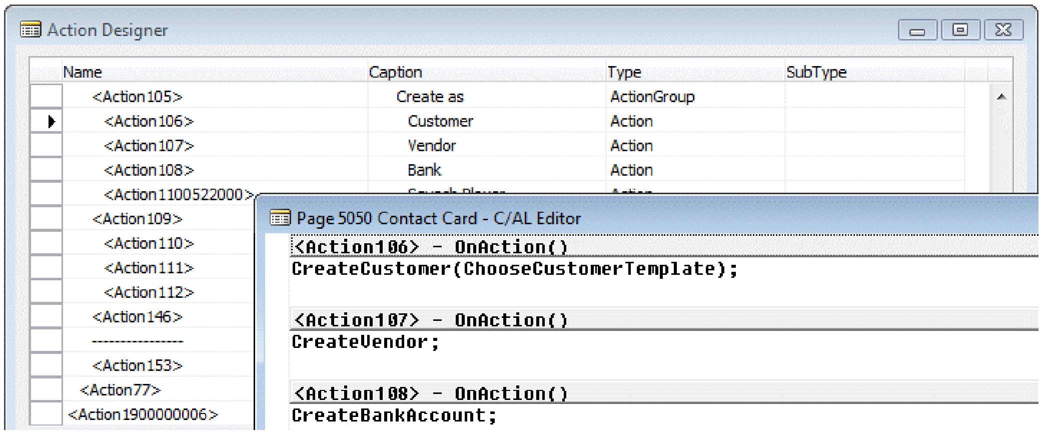


CreateVendor versus CreateCustomer

In Microsoft Dynamics NAV there is a small difference between creating a customer and a vendor from a contact. When creating a customer the system will ask us to select a customer template. The vendor option does not have this. Because of this simplicity we will look at and learn from the `Vendor` function in this chapter.

The customer and vendor table are almost identical in structure and fields are numbered similar in both tables. This is called transaction mirroring between sales and purchasing, which we will discuss further in *Chapter 6*, *Trade*. We will mirror our new table in a similar way to the other Microsoft Dynamics NAV tables.

The C/AL code in the **Action** tells us that while pushing the menu option, the function **CreateVendor** in the contact table is started. To copy this feature we need to create a new function `CreateSquashPlayer`. Let's keep this in mind while we dive further into this code.

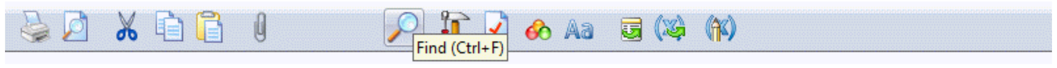


Name	Caption	Type	SubType
<Action105>	Create as	ActionGroup	
<Action106>	Customer	Action	
<Action107>	Vendor	Action	
<Action108>	Bank	Action	
<Action1100522000>			
<Action109>			
<Action110>			
<Action111>			
<Action112>			
<Action146>			

<Action153>			
<Action77>			
<Action1900000006>			

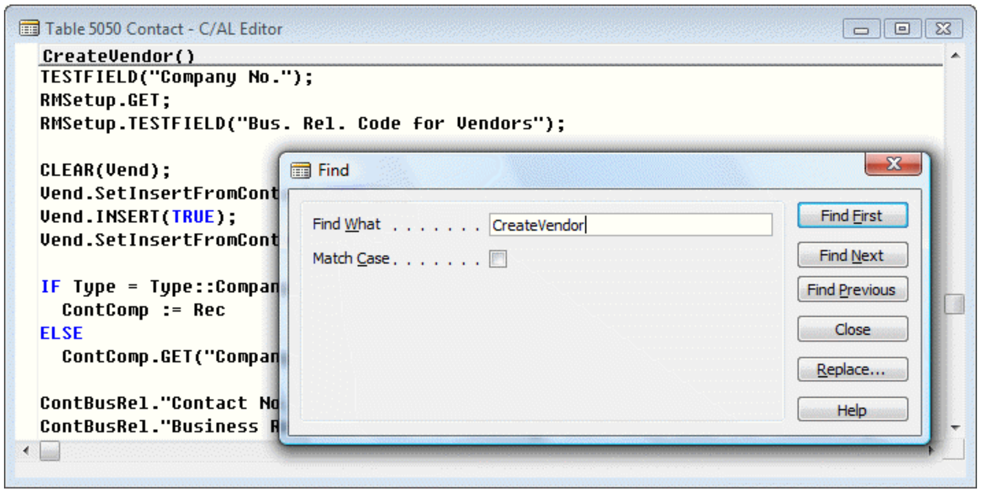
```
<Action106> - OnAction()
CreateCustomer(ChooseCustomerTemplate);

<Action107> - OnAction()
CreateVendor;

<Action108> - OnAction()
CreateBankAccount;
```

Open the Contact table (no. 5050), and search the function **CreateVendor**. You can find functions in a table by going into the C/AL code (*F9*) from anywhere in the table designer, and use the **Find (Ctrl+F)** function as shown in the following screenshot:



Reverse engineering

We need to reverse engineer this code in order to see what we need to create for our `CreateSquashPlayer` function. We will look at each part of the C/AL code in order to decide whether we need it or not.



```
CreateVendor()
TESTFIELD("Company No.");
RMSetup.GET;
RMSetup.TESTFIELD("Bus. Rel. Code for Vendors");

CLEAR(Vend);
Vend.SetInsertFromCont
Vend.INSERT(TRUE);
Vend.SetInsertFromCont

IF Type = Type::Compan
  ContComp := Rec
ELSE
  ContComp.GET("Compan

ContBusRel."Contact No
ContBusRel."Business R
```

Find What . . . . CreateVendor

What does this piece of code do?

```
TESTFIELD("Company No.");
```

This tests the current record for a valid `Company No`. If this fails we cannot continue and the end user get's a runtime error.

```
RMSetup.GET;
RMSetup.TESTFIELD("Bus. Rel. Code for Vendors");
```

This reads the marketing setup table from the system and tests whether the `Bus. Rel. Code for Vendors` is valid. We need a new code for squash players here, which will be added as a new field to the setup table.

```
CLEAR(Vend);
Vend.SetInsertFromContact(TRUE);
Vend.INSERT(TRUE);
Vend.SetInsertFromContact(FALSE);
```

Here, the vendor table is cleared and a function is called within that table, then a new record is inserted in the database while activating the necessary business logic. Then the same function is called again with another parameter. Since the vendor table is what we are copying we will write down that we might need a similar function as `SetInsertFromContact`.

```
IF Type = Type::Company THEN
  ContComp := Rec
ELSE
  ContComp.GET("Company No.");
```

This code checks whether the current contact is a company. If so it populates the variable `ContComp` with this record. If not it populates `ContComp` with the company our current contact is related to.

```
ContBusRel."Contact No." := ContComp."No.";
ContBusRel."Business Relation Code" := RMSetup."Bus. Rel. Code for
Vendors";
ContBusRel."Link to Table" := ContBusRel."Link to Table"::Vendor;
ContBusRel."No." := Vend."No.";
ContBusRel.INSERT(TRUE);
```

The `ContBusRel` refers to the table, Contact Business Relation (5054), and is a linking table in the Microsoft Dynamics NAV data model. Technically a contact can be connected to multiple customers and vendors although this does not make sense. This table is populated here. Let's write down that we need to look into this table and see if it needs changes.

```
UpdateCustVendBank.UpdateVendor(ContComp,ContBusRel);
```

`UpdateCustVendBank` is an external code unit that is used with the function, `UpdateVendor`, we might need a copy of this function for our Squash players.

```
MESSAGE(Text009,Vend.TABLECAPTION,Vend."No.");
```

This throws a message box for the end-user stating that the record was created with a new number.

Now we have a number of things on our to-do list:

1. Create a master data table which looks like the vendor table.
2. We need to copy the `CreateVendor` function.
3. Look at the contact business relation table and the `UpdateCustVendBank` (5055) code unit.

Let's look at the latter to learn something important before we start with the first.

```
UpdateVendor()
WITH Vend DO BEGIN
  GET(ContBusRel."No.");
  xRecRef.GETTABLE(Vend);
  NoSerie := "No. Series";
  PurchaserCode :=  Vend."Purchaser Code";
```

```
        TRANSFERFIELDS(Cont);
        "No." := ContBusRel."No.";
        "No. Series" := NoSerie;
        Vend."Purchaser Code" := PurchaserCode;
        MODIFY;
        RecRef.GETTABLE(Vend);
        ChangeLogMgt.LogModification(RecRef,xRecRef);
    END;
```

This code synchronizes the contact table with the vendor table. It does that by using the `TRANSFERFIELDS` function. This function transfers all fields with the same *number* from one table to another. This means that we cannot be creative with our field numbering. For example, in the contact table the **Name** field has number 2. If we were to use a different number for the **Name** field then `TRANSFERFIELDS` would not copy the information.

Using this information our table should look like this:

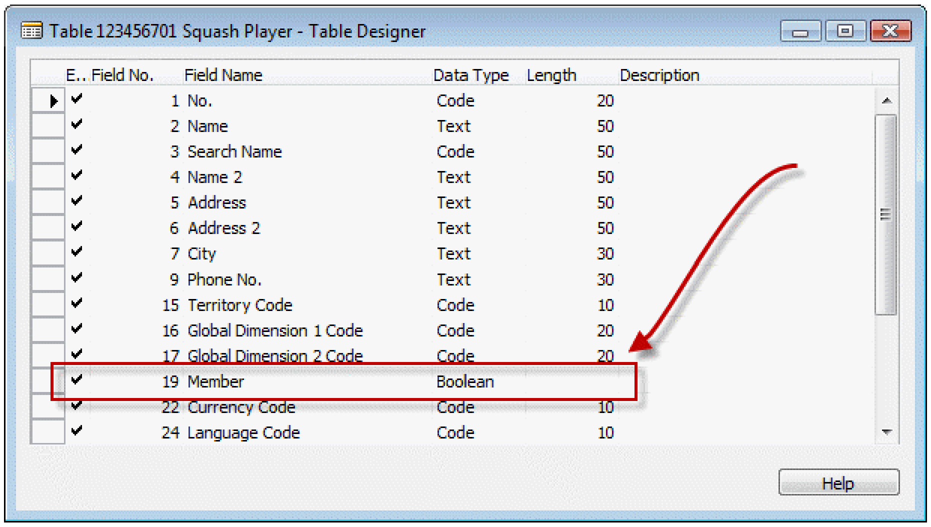


Table 123456701 Squash Player - Table Designer

E..	Field No.	Field Name	Data Type	Length	Description
✓	1	No.	Code	20	
✓	2	Name	Text	50	
✓	3	Search Name	Code	50	
✓	4	Name 2	Text	50	
✓	5	Address	Text	50	
✓	6	Address 2	Text	50	
✓	7	City	Text	30	
✓	9	Phone No.	Text	30	
✓	15	Territory Code	Code	10	
✓	16	Global Dimension 1 Code	Code	20	
✓	17	Global Dimension 2 Code	Code	20	
✓	19	Member	Boolean		
✓	22	Currency Code	Code	10	
✓	24	Language Code	Code	10	

Notice that we use field 19 for our Squash Player specific field. This is because field 19 was used for Budgeted Amount in the vendor table. We can therefore safely assume that Microsoft will not use field 19 in the contact table in the future.

An alternative approach for this, if we want to be even safer, is to add the fields that are specific to our solution as fields in our add-on number series. In our case that would be 123.456.700.

[You can copy and paste fields from one table to another table. Note that table relations and C/AL code in the `OnValidate` and `OnLookup` trigger is copied as well. If the table we want to create is similar to an existing table we could also use the **Save As** option from the **File** drop-down menu.]

The next step is to add some business logic to the table. We want this table to use number series functionality just like the vendor table. This requires some standard steps:

1. **Create the Setup table**: A number series is defined in a setup table. As the Squash Court module will be quite sophisticated, we'll create our own.

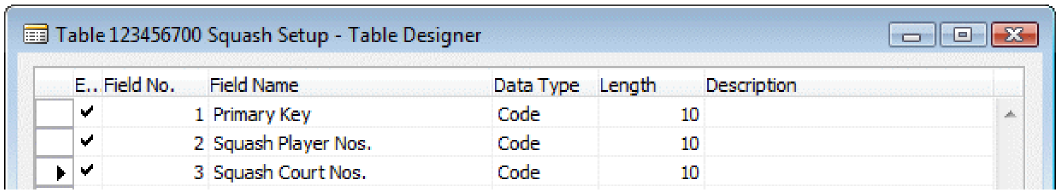


Table 123456700 Squash Setup - Table Designer

E..	Field No.	Field Name	Data Type	Length	Description
✓	1	Primary Key	Code	10	
✓	2	Squash Player Nos.	Code	10	
✓	3	Squash Court Nos.	Code	10	

A setup table always has a single **Primary Key** field (as shown in the previous screenshot) and the necessary setup fields. This table is designed to only have one single record.

2. **Create a link to the number series**: Our **Squash Player** table is now required to have a link to the number series. We can copy this field from the vendor table and can make a table relation to the **No. Series** table (as shown in the following screenshot):

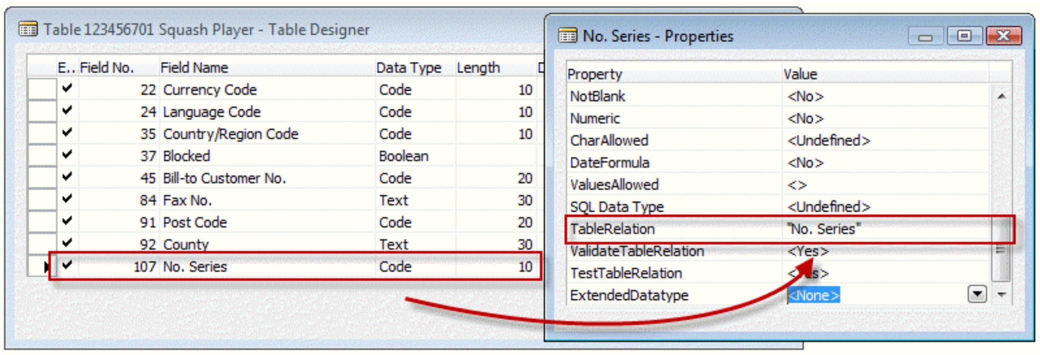


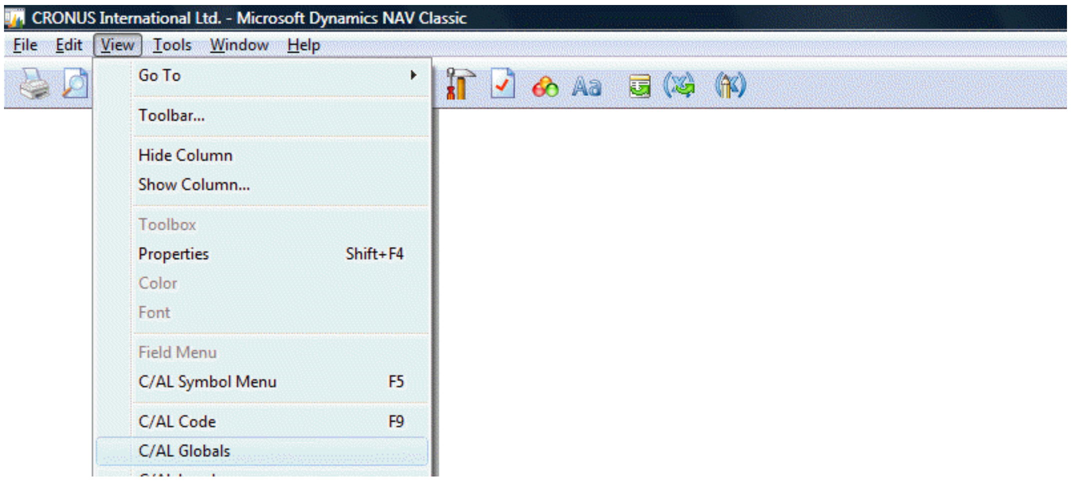
Table 123456701 Squash Player - Table Designer

E..	Field No.	Field Name	Data Type	Length
✓	22	Currency Code	Code	10
✓	24	Language Code	Code	10
✓	35	Country/Region Code	Code	10
✓	37	Blocked	Boolean	
✓	45	Bill-to Customer No.	Code	20
✓	84	Fax No.	Text	30
✓	91	Post Code	Code	20
✓	92	County	Text	30
✓	107	No. Series	Code	10

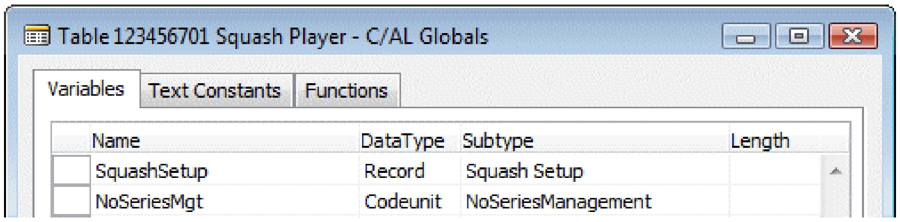
No. Series - Properties

Property	Value
NotBlank	<No>
Numeric	<No>
CharAllowed	<Undefined>
DateFormula	<No>
ValuesAllowed	<>
SQL Data Type	<Undefined>
TableRelation	"No. Series"
ValidateTableRelation	<Yes>
TestTableRelation	<Yes>
ExtendedDatatype	<None>

3 **Add the C/AL Business Logic**: Now we can add the business logic to our table, but first we need to define the variables that are required. These are our new **SquashSetup** table and the number series management codeunit.



We can define the variables in the specially created **C/AL Globals** menu (as shown in the screenshot):



Name	DataType	Subtype	Length
SquashSetup	Record	Squash Setup	
NoSeriesMgt	Codeunit	NoSeriesManagement	

[It is highly recommended that you use the Microsoft naming standard, which allows you to copy and paste a lot of code and makes it easier for others to read your code.]

Number Series require three pieces of code. This code makes sure that the business logic of the Number Series functionality is always followed:

1. The `OnInsert` Trigger: This code goes into the `OnInsert` trigger. It populates the No. field with the next value of the Number Series.

```
OnInsert()
IF "No." = '' THEN BEGIN
  SquashSetup.GET;
  SquashSetup.TESTFIELD("Squash Player Nos.");
  NoSeriesMgt.InitSeries(SquashSetup."Squash Player Nos.",
    xRec."No. Series",0D,"No.","No. Series");
END;
```

2. The `OnValidate` Trigger of the No. field: The `OnValidate` code of the No. field tests when a user manually enters a value if that is allowed.

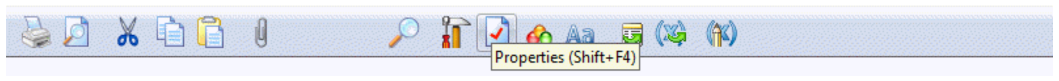
```
No. - OnValidate()
IF "No." <> xRec."No." THEN BEGIN
  SquashSetup.GET;
  NoSeriesMgt.TestManual(SquashSetup."Squash Player Nos.");
  "No. Series" := '';
END;
```

3. Create a new `AssistEdit` function: Lastly, we create a new function. This is always called `AssistEdit`. This is for readability and others reading your code afterwards. The code is used in the page or form and allows users to switch between linked number series.

```
AssistEdit() : Boolean
SquashSetup.GET;
SquashSetup.TESTFIELD("Squash Player Nos.");
IF NoSeriesMgt.SelectSeries(SquashSetup."Squash Player Nos.",
  xRec."No. Series","No. Series")
THEN BEGIN
  NoSeriesMgt.SetSeries("No.");
  EXIT(TRUE);
END;
```

When the number series are in place we can make the necessary changes in the Contact Business Relation table.

In this table we need to add the possibility to link squash players to contacts. This is done in the properties window of the **Table Designer** that can be accessed by pressing *Shift* +*F4* or by using the **Properties** option from the **View** drop-down menu (as shown in the following screenshot):



First we add the Squash player option to the **Link to Table** field (as shown in the following screenshot):

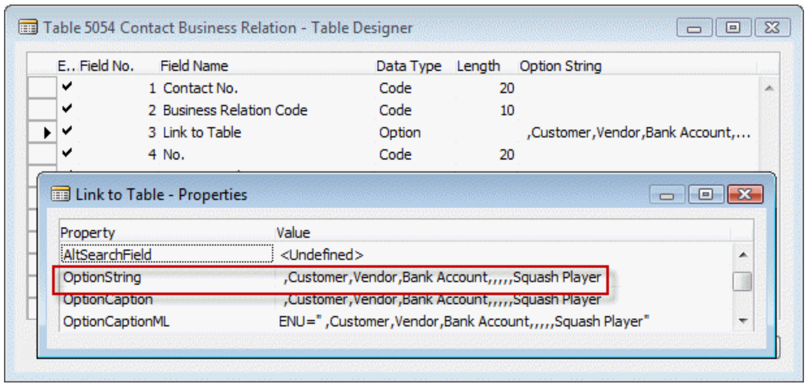


Table 5054 Contact Business Relation - Table Designer

E..	Field No.	Field Name	Data Type	Length	Option String
✓	1	Contact No.	Code	20	
✓	2	Business Relation Code	Code	10	
✓	3	Link to Table	Option		,Customer,Vendor,Bank Account,...
✓	4	No.	Code	20	

Link to Table - Properties

Property	Value
AltSearchField	<Undefined>
OptionString	,Customer,Vendor,Bank Account,,,,,Squash Player
OptionCaption	,Customer,Vendor,Bank Account,,,,,Squash Player
OptionCaptionML	ENU=",Customer,Vendor,Bank Account,,,,,Squash Player"

Options are converted to SQL Integer data types. Make sure to add some blank options so when Microsoft releases other functionality we are not impacted. Changing the integer value of an existing option field requires a lot of work.

Then we create a table relation with our new table (as shown in the following screenshot):

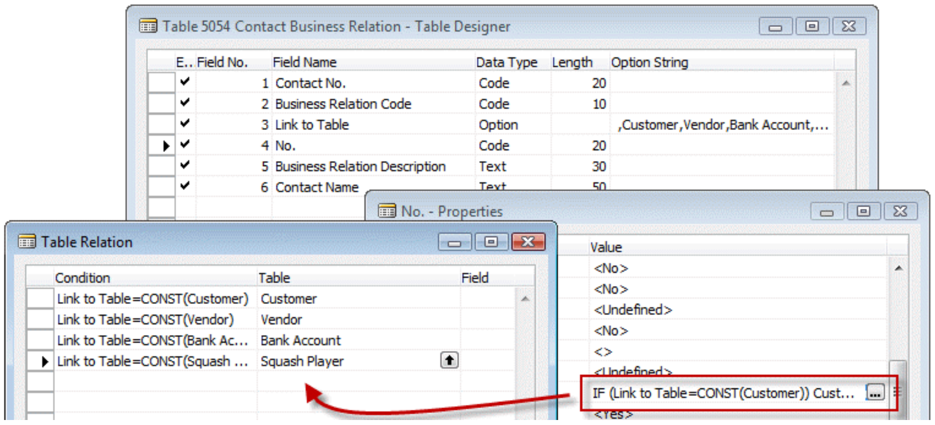


Table 5054 Contact Business Relation - Table Designer

E..	Field No.	Field Name	Data Type	Length	Option String
✓	1	Contact No.	Code	20	
✓	2	Business Relation Code	Code	10	
✓	3	Link to Table	Option		,Customer,Vendor,Bank Account,...
✓	4	No.	Code	20	
✓	5	Business Relation Description	Text	30	
✓	6	Contact Name	Text	50	

Table Relation

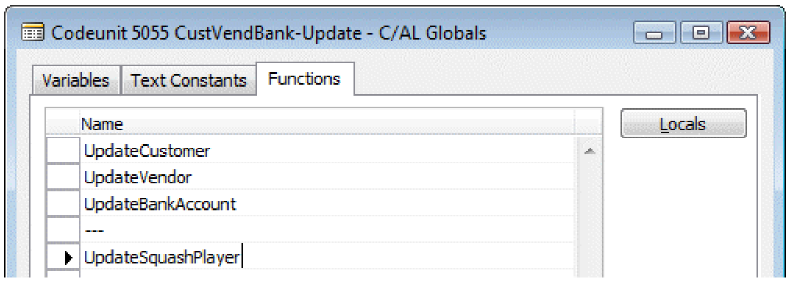
Condition	Table	Field
Link to Table=CONST(Customer)	Customer	
Link to Table=CONST(Vendor)	Vendor	
Link to Table=CONST(Bank Ac...	Bank Account	
Link to Table=CONST(Squash ...	Squash Player	

No. - Properties

Value
<No>
<No>
<Undefined>
<No>
<>
<Undefined>
IF (Link to Table=CONST(Customer)) Cust...
<Yes>

The next step is to expand the Codeunit **CustVendBank-Update** with a new function `UpdateSquashPlayer` (as shown in the following screenshot). This is a copy of the function `UpdateVendor` we discussed earlier. We can add functions in the **Globals** menu.

There are two ways to copy a function. We can create a new function manually and copy the C/AL code and variables, or we can select a function from the list and use copy and paste and then rename the function:


Codeunit 5055 CustVendBank-Update - C/AL Globals

Name
UpdateCustomer
UpdateVendor
UpdateBankAccount

UpdateSquashPlayer

When you add the — line to the function others can see that it is not a Microsoft function. You can also include the project name like — **Squash**.

This code also requires a new global variable, `SquashPlayer`:

```
UpdateSquashPlayer()
WITH SquashPlayer DO BEGIN
  GET(ContBusRel."No.");
  xRecRef.GETTABLE(SquashPlayer);
  NoSerie := "No. Series";
  TRANSFERFIELDS(Cont);
  "No." := ContBusRel."No.";
  "No. Series" := NoSerie;
  MODIFY;
  RecRef.GETTABLE(SquashPlayer);
  ChangeLogMgt.LogModification(RecRef,xRecRef);
END;
```

The final piece of preparation work is to add the **Bus. Rel. Code for Squash Players** field to the **Marketing Setup** table as shown in the following screenshot:

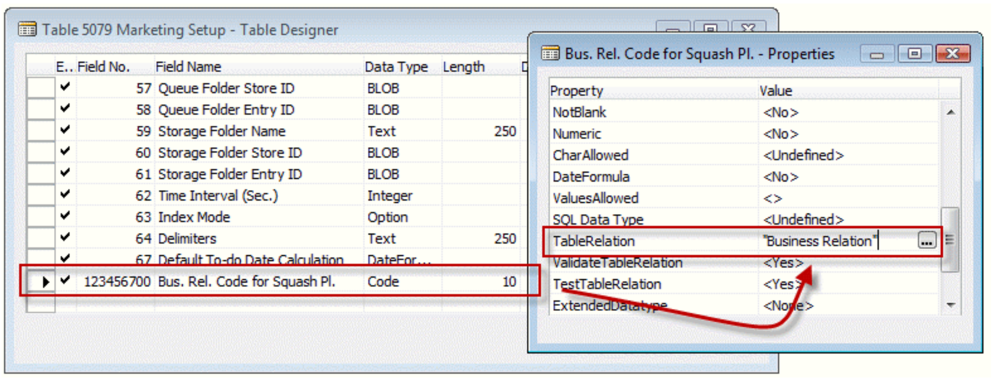


Table 5079 Marketing Setup - Table Designer

E..	Field No.	Field Name	Data Type	Length
✓	57	Queue Folder Store ID	BLOB	
✓	58	Queue Folder Entry ID	BLOB	
✓	59	Storage Folder Name	Text	250
✓	60	Storage Folder Store ID	BLOB	
✓	61	Storage Folder Entry ID	BLOB	
✓	62	Time Interval (Sec.)	Integer	
✓	63	Index Mode	Option	
✓	64	Delimiters	Text	250
✓	67	Default To-do Date Calculation	DateFor...	
✓	123456700	Bus. Rel. Code for Squash Pl.	Code	10

Bus. Rel. Code for Squash Pl. - Properties

Property	Value
NotBlank	<No>
Numeric	<No>
CharAllowed	<Undefined>
DateFormula	<No>
ValuesAllowed	<>
SQL Data Type	<Undefined>
TableRelation	"Business Relation"
ValidateTableRelation	<Yes>
TestTableRelation	<Yes>
ExtendedDatatype	<None>

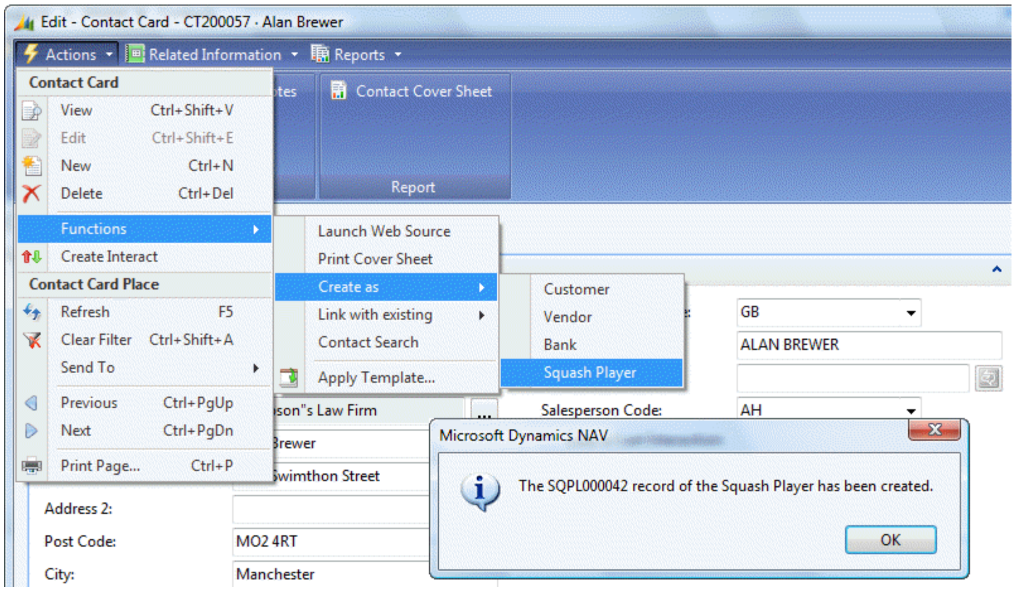
> We use the same numbering in our fields as in our objects. This makes it easier in the future to see what belongs to what if more functionality is added.

With all this preparation work we can now finally go ahead and make our function in the contact table (5050) that we can call from the user interface.

```
CreateSquashPlayer()
TESTFIELD(Type, Type::Person);

RMSetup.GET;
RMSetup.TESTFIELD("Bus. Rel. Code for Squash Pl.");

CLEAR(SquashPlayer);
SquashPlayer.INSERT(TRUE);

ContBusRel."Contact No." := Cont."No.";
ContBusRel."Business Relation Code" :=
  RMSetup."Bus. Rel. Code for Squash Pl.";
ContBusRel."Link to Table" :=
  ContBusRel."Link to Table"::"Squash Player";
ContBusRel."No." := SquashPlayer."No.";
ContBusRel.INSERT(TRUE);

UpdateCustVendBank.UpdateSquashPlayer(Cont,ContBusRel);

MESSAGE(Text009,SquashPlayer.TABLECAPTION,SquashPlayer."No.");
```

Please note that we do not need the `SetInsertFromContact` function. This function enables users to create a new Vendor first and Create a Contact using the Vendor information. We do not want to support this method in our application.

Now we can add the function to the page and test our functionality:



The SQPL000042 record of the Squash Player has been created.

Designing a journal

Now it is time to start on the product part of the Squash Application. In this part we will no longer reverse engineer in detail. We will learn how to search in the standard functionality and reuse parts in our own software.

For this part we will look at resources in Microsoft Dynamics NAV. Resources are similar to use products as Items but far less complex making it easier to look and learn.

Squash Court master data

Our company has 12 courts that we want to register in Microsoft Dynamics NAV. This master data is comparable to resources so we'll go ahead and copy this functionality. Resources are not attached to umbrella data like the vendor/squash player tables. We need the number series again so we'll add a new number series to our squash setup table.

The **Squash Court** table should look like this after creation:

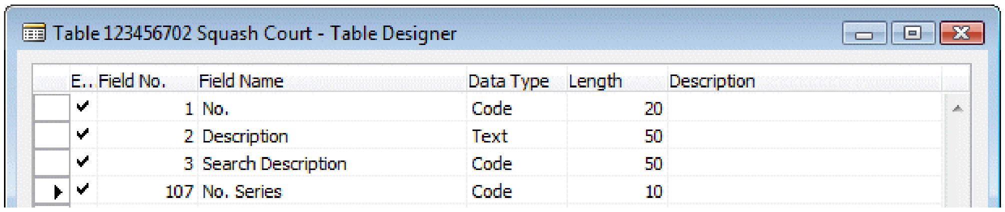
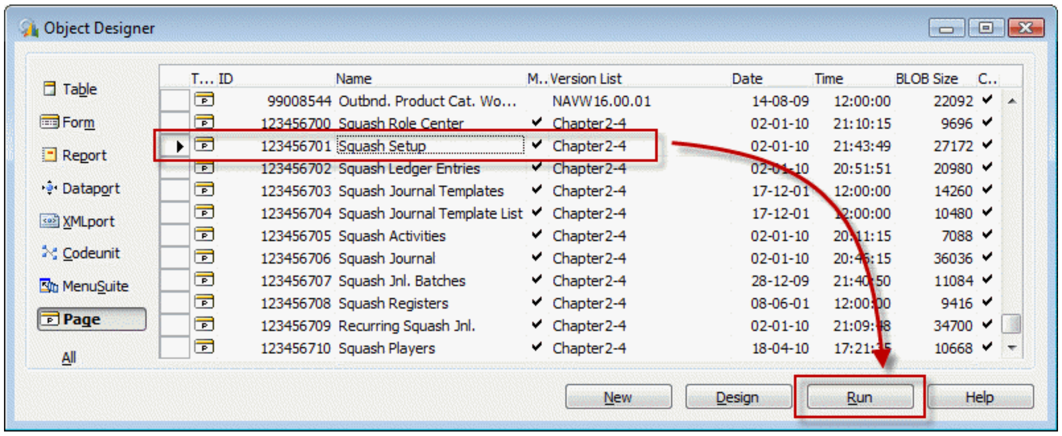


Table 123456702 Squash Court - Table Designer

E..	Field No.	Field Name	Data Type	Length	Description
✓	1	No.	Code	20	
✓	2	Description	Text	50	
✓	3	Search Description	Code	50	
✓	107	No. Series	Code	10	

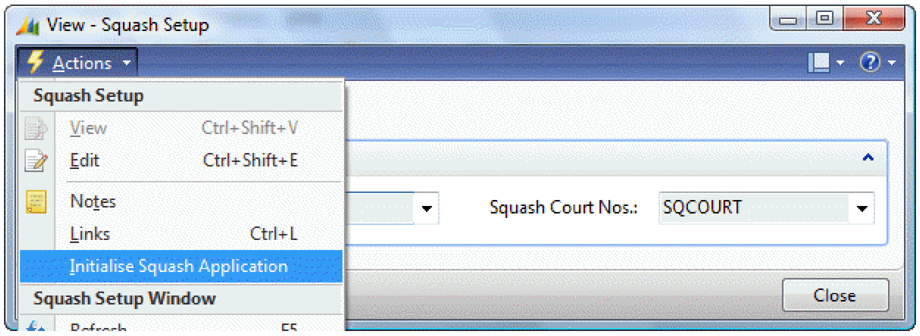
Chapter objects

With this chapter some objects are required. A description of how to import these objects can be found in the *Appendix*.


Object Designer

T...	ID	Name	M..	Version List	Date	Time	BLOB Size	C..
	99008544	Outbnd. Product Cat. Wo...		NAVW16.00.01	14-08-09	12:00:00	22092	✓
	123456700	Squash Role Center	✓	Chapter2-4	02-01-10	21:10:15	9696	✓
	123456701	Squash Setup	✓	Chapter2-4	02-01-10	21:43:49	27172	✓
	123456702	Squash Ledger Entries	✓	Chapter2-4	02-01-10	20:51:51	20980	✓
	123456703	Squash Journal Templates	✓	Chapter2-4	17-12-01	12:00:00	14260	✓
	123456704	Squash Journal Template List	✓	Chapter2-4	17-12-01	12:00:00	10480	✓
	123456705	Squash Activities	✓	Chapter2-4	02-01-10	20:11:15	7088	✓
	123456706	Squash Journal	✓	Chapter2-4	02-01-10	20:46:15	36036	✓
	123456707	Squash Jnl. Batches	✓	Chapter2-4	28-12-09	21:40:50	11084	✓
	123456708	Squash Registers	✓	Chapter2-4	08-06-01	12:00:00	9416	✓
	123456709	Recurring Squash Jnl.	✓	Chapter2-4	02-01-10	21:09:48	34700	✓
	123456710	Squash Players	✓	Chapter2-4	18-04-10	17:21:15	10668	✓

After the import process is completed make sure that your current database is the default database for the role tailored client and run Page 123456701, **Squash Setup**.


View - Squash Setup

Squash Court Nos.: SQCOURT

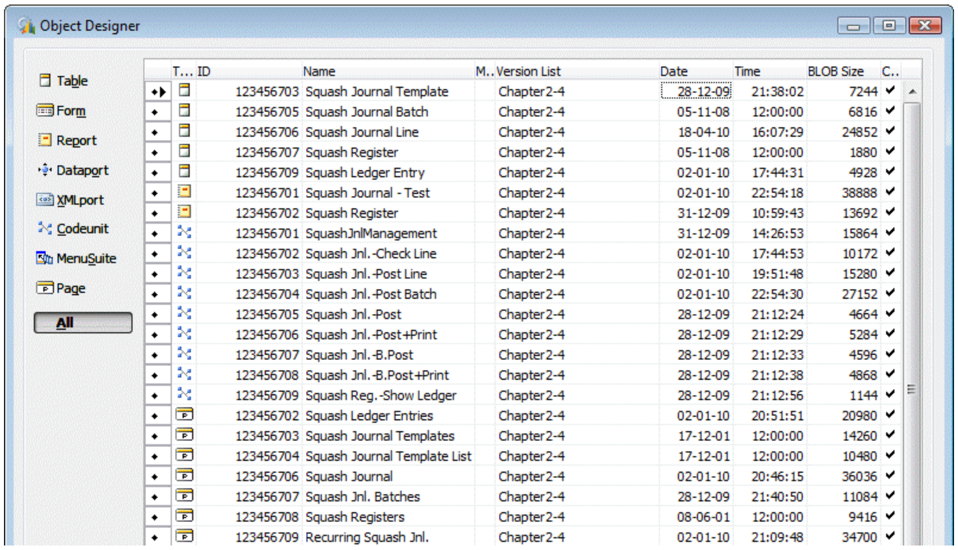
From this page select the action **Initialise Squash Application**. This will execute the C/AL code in the `InitSquashApp` function of this page, which will prepare demo data for us to play with. The objects are prepared and tested in a Microsoft Dynamics NAV 2009 SP1 W1 database.

Reservations

When running a squash court we want to be able to keep track of reservations. Looking at standard Dynamics NAV functionality it might be a good idea to create Squash player Journal. The Journal can create entries for reservations that can be invoiced.

A journal needs the object structure. The journal is prepared in the objects delivered with this chapter. Creating a new journal from scratch is a lot of work and can easily lead to making mistakes. It is easier and safer to copy an existing Journal structure from the standard application that is similar to the journal we need for our design.

In our example we have copied the Resource Journals.



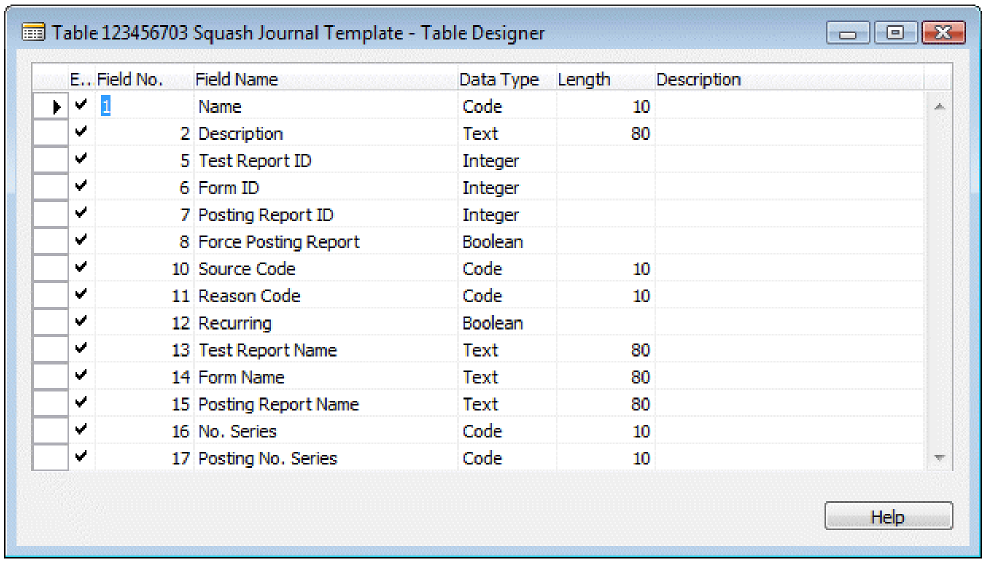
T...	ID	Name	M..	Version List	Date	Time	BLOB Size	C..
	123456703	Squash Journal Template		Chapter2-4	28-12-09	21:38:02	7244	✓
	123456705	Squash Journal Batch		Chapter2-4	05-11-08	12:00:00	6816	✓
	123456706	Squash Journal Line		Chapter2-4	18-04-10	16:07:29	24852	✓
	123456707	Squash Register		Chapter2-4	05-11-08	12:00:00	1880	✓
	123456709	Squash Ledger Entry		Chapter2-4	02-01-10	17:44:31	4928	✓
	123456701	Squash Journal - Test		Chapter2-4	02-01-10	22:54:18	38888	✓
	123456702	Squash Register		Chapter2-4	31-12-09	10:59:43	13692	✓
	123456701	SquashJnlManagement		Chapter2-4	31-12-09	14:26:53	15864	✓
	123456702	Squash Jnl.-Check Line		Chapter2-4	02-01-10	17:44:53	10172	✓
	123456703	Squash Jnl.-Post Line		Chapter2-4	02-01-10	19:51:48	15280	✓
	123456704	Squash Jnl.-Post Batch		Chapter2-4	02-01-10	22:54:30	27152	✓
	123456705	Squash Jnl.-Post		Chapter2-4	28-12-09	21:12:24	4664	✓
	123456706	Squash Jnl.-Post+Print		Chapter2-4	28-12-09	21:12:29	5284	✓
	123456707	Squash Jnl.-B.Post		Chapter2-4	28-12-09	21:12:33	4596	✓
	123456708	Squash Jnl.-B.Post+Print		Chapter2-4	28-12-09	21:12:38	4868	✓
	123456709	Squash Reg.-Show Ledger		Chapter2-4	28-12-09	21:12:56	1144	✓
	123456702	Squash Ledger Entries		Chapter2-4	02-01-10	20:51:51	20980	✓
	123456703	Squash Journal Templates		Chapter2-4	17-12-01	12:00:00	14260	✓
	123456704	Squash Journal Template List		Chapter2-4	17-12-01	12:00:00	10480	✓
	123456706	Squash Journal		Chapter2-4	02-01-10	20:46:15	36036	✓
	123456707	Squash Jnl. Batches		Chapter2-4	28-12-09	21:40:50	11084	✓
	123456708	Squash Registers		Chapter2-4	08-06-01	12:00:00	9416	✓
	123456709	Recurring Squash Jnl.		Chapter2-4	02-01-10	21:09:48	34700	✓

[You can export these objects to text format, and then rename and renumber the objects to be reused easily. The squash journal objects are renumbered and renamed from the resource journal.]

As explained in *Chapter 1*, *Introduction to Microsoft Dynamics NAV*, all journals have the same structure. The template, batch and register tables are almost always the same, whereas the journal line and ledger entry table contain function specific fields. Let's have a look at all of them one by one.

Journal Template

The Journal Template has several fields as shown in the following screenshot:



E..	Field No.	Field Name	Data Type	Length	Description
✓	1	Name	Code	10	
✓	2	Description	Text	80	
✓	5	Test Report ID	Integer		
✓	6	Form ID	Integer		
✓	7	Posting Report ID	Integer		
✓	8	Force Posting Report	Boolean		
✓	10	Source Code	Code	10	
✓	11	Reason Code	Code	10	
✓	12	Recurring	Boolean		
✓	13	Test Report Name	Text	80	
✓	14	Form Name	Text	80	
✓	15	Posting Report Name	Text	80	
✓	16	No. Series	Code	10	
✓	17	Posting No. Series	Code	10	

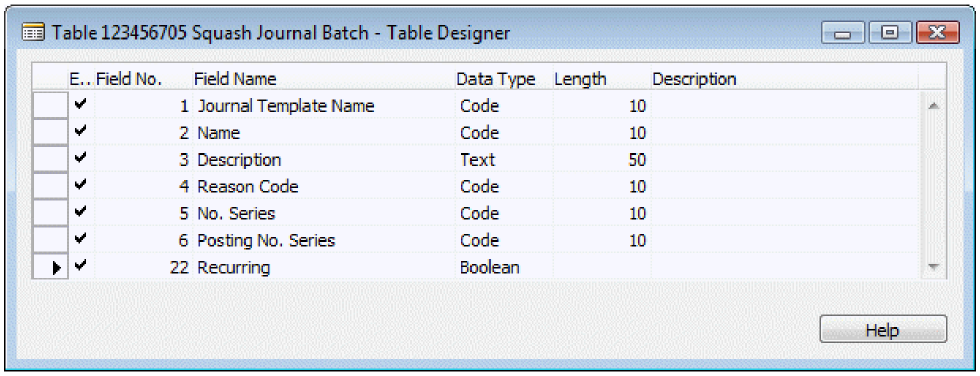
Lets discuss these fields in more detail:

- **Name**: This is the unique name. It is possible to define as many Templates as required but usually one Template per **Form ID** and one for **Recurring** will do. If you want journals with different source codes you need to have more templates.
- **Description**: A readable and understandable description of its purpose.
- **Test Report ID**: All Templates have a test report that allows the user to check for posting errors.
- **Form ID**: For some journals, more UI objects are required. For example, the General Journals have a special form for bank and cash.
- **Posting Report ID**: This report is printed when a user selects Post and Print.
- **Force Posting Report**: Use this option when a posting report is mandatory.
- **Source Code**: Here you can enter a Trail Code for all the postings done via this Journal.

- **Reason Code**: This functionality is similar to source sodes.
- **Recurring**: Whenever you post lines from a recurring journal, new lines are automatically created with a posting date defined in the recurring date formula.
- **No. Series**: When you use this feature the Document No. in the Journal Line is automatically populated with a new number from this Number Series.
- **Posting No. Series**: Use this feature for recurring journals.

Journal Batch

Journal Batch has various fields as shown in the following screenshot:



E..	Field No.	Field Name	Data Type	Length	Description
✓	1	Journal Template Name	Code	10	
✓	2	Name	Code	10	
✓	3	Description	Text	50	
✓	4	Reason Code	Code	10	
✓	5	No. Series	Code	10	
✓	6	Posting No. Series	Code	10	
✓	22	Recurring	Boolean		

Lets discuss these fields in more detail:

- **Journal Template Name**: The name of the Journal Template this batch refers to
- **Name** : Each batch should have a unique code
- **Description**: A readable and explaining description for this batch
- **Reason Code**: When populated, this **Reason Code** will overrule the **Reason Code** from the **Journal Template**
- **No. Series**: When populated this **No. Series** will overrule the **No. Series** from the **Journal Template**
- **Posting No. Series**: When populated this **Posting No. Series** will overrule the **Posting No. Series** from the **Journal Template**

Register

The Register table has various fields as shown in the following screenshot:

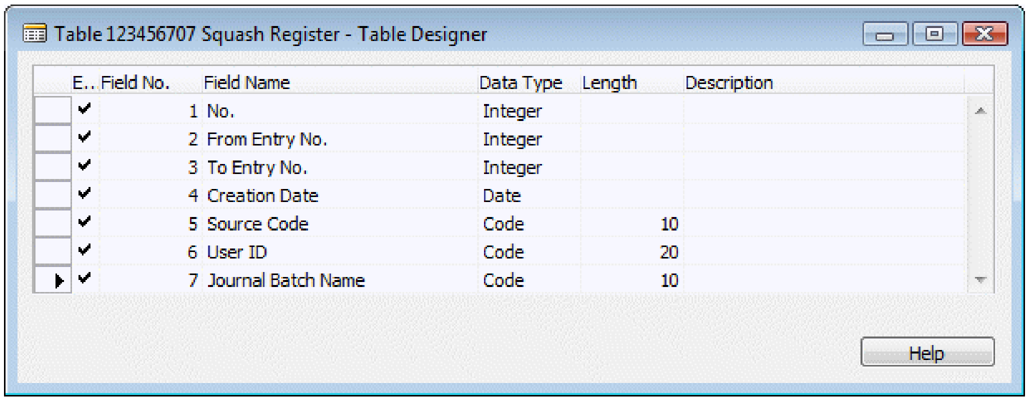


Table 123456707 Squash Register - Table Designer

E..	Field No.	Field Name	Data Type	Length	Description
✓	1	No.	Integer		
✓	2	From Entry No.	Integer		
✓	3	To Entry No.	Integer		
✓	4	Creation Date	Date		
✓	5	Source Code	Code	10	
✓	6	User ID	Code	20	
✓	7	Journal Batch Name	Code	10	

Lets discuss these fields in more detail:

- **No.**: This field is automatically and incrementally populated for each transaction with this journal. There are no gaps between the numbers.
- **From Entry No.**: A reference to the first Ledger Entry created is with this transaction.
- **To Entry No.**: A reference to the last Ledger Entry is created with this transaction.
- **Creation Date**: Always populated with the real date when the transaction was posted.
- **User ID**: The ID of the end user who has posted the transaction.

The Journal

The journal line has a number of mandatory fields that are required for all journals and some fields that are required for its designed functionality.

In our case the journal should create a reservation which then can be invoiced. This requires some information to be populated in the lines.

Reservation

The reservation process is a logistical process that requires us to know the number of the Squash Court, the date, and the time of the reservation. We also need to know how long the players want to play. To check the reservation it might also be useful to store the number of the Squash Player.

Invoicing

For the invoicing part we need to know the price we need to invoice. It might also be useful to store the cost to see our profit. For the system to figure out the proper G/L Account for the turnover we also need to define a General Product Posting Group. We will see more of how that works later in *Chapter 3, Financial Managemenet*.

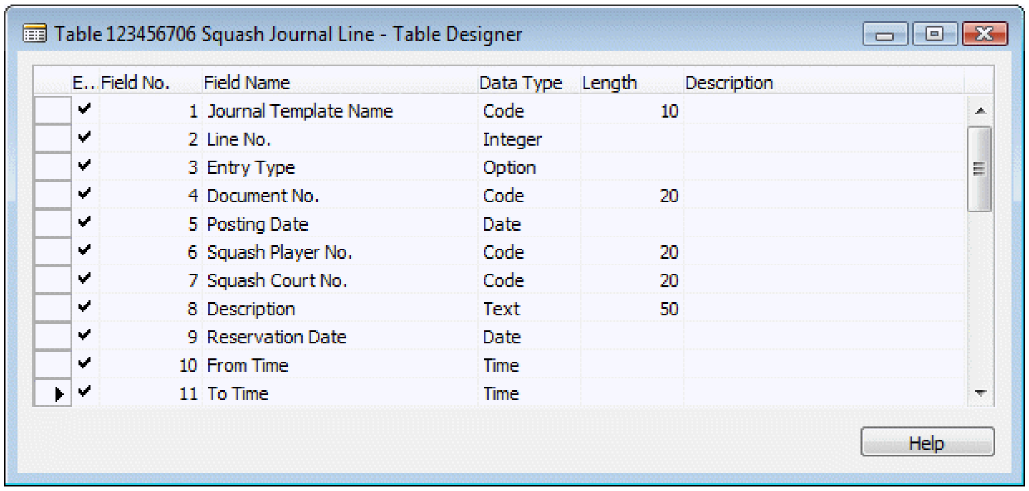


Table 123456706 Squash Journal Line - Table Designer

E..	Field No.	Field Name	Data Type	Length	Description
✓	1	Journal Template Name	Code	10	
✓	2	Line No.	Integer		
✓	3	Entry Type	Option		
✓	4	Document No.	Code	20	
✓	5	Posting Date	Date		
✓	6	Squash Player No.	Code	20	
✓	7	Squash Court No.	Code	20	
✓	8	Description	Text	50	
✓	9	Reservation Date	Date		
✓	10	From Time	Time		
✓	11	To Time	Time		

- **Journal Template Name**: This is a reference to the current journal template.
- **Line No.** : Each journal has virtually unlimited number of lines; this number is automatically incremented by 10000 allowing lines to be created in between.
- **Entry Type**: Reservation or invoice.
- **Document No.**: This number can be used to give to the squash player as a reservation number. When the entry type is invoice, it is the invoice number.
- **Posting Date**: Posting date is usually the reservation date but when the entry type is invoice it might be the date of the invoice which might differ from the posting date in the general ledger.
- **Squash Player No.**: A reference to the squash player who has made the reservation.

- **Squash Court No.**: A reference to the squash court.
- **Description**: This is automatically updated with the number of the squash court, reservation date and times, but can be changed by the user.
- **Reservation Date**: The actual date of the reservation.
- **From Time**: The starting time of the reservation. We allow only whole or half hours.
- **To Time**: The ending time of the reservation. We only allow whole and half hours. This is automatically populated when people enter a quantity.
- **Quantity**: The number of hours playing time. We only allow units of 0.5 to be entered here. This is automatically calculated when the times are populated.
- **Unit Cost**: The cost to run a Squash Court for one hour.
- **Total Cost**: The cost for this reservation.
- **Unit Price**: The invoice price for this reservation per hour. This depends on whether or not the squash player is a member or not.
- **Total Price**: The total invoice price for this reservation.
- **Shortcut Dimension Code 1 & 2**: A reference to the dimensions used for this transaction.
- **Applies-to Entry No.**: When a reservation is invoiced, this is the reference to the squash entry no. of the reservation.
- **Source Code**: Inherited from the journal batch or template and used when posting the transaction.
- **Chargeable**: When this option is used, there will not be an invoice for the reservation.
- **Journal Batch Name**: A reference to the journal batch that is used for this transaction.
- **Reason Code**: Inherited from the journal batch or template, and used when posting the transaction.
- **Recurring Method**: When the journal is a recurring journal you can use this field to determine whether the amount field is blanked after posting the lines.
- **Recurring Frequency**: This field determines the new posting date after the recurring lines are posted.
- **Gen. Bus. Posting Group**: The combination of general business and product posting group determines the G/L cccount for turnover when we invoice the reservation. The Gen. Bus. Posting Group is inherited from the bill-to customer.

- **Gen. Prod. Posting Group**: This will be inherited from the squash player.
- **External Document No.**: When a squash player wants us to note a reference number we can store it here.
- **Posting No. Series**: When the journal template has a posting no. series it is populated here to be used when posting.
- **Bill-to Customer No.**: This determines who is paying for the reservation. We will inherit this from the squash player.

So now we have a place to enter reservations but we have something to do before we can start doing this.

Some fields were determined to be inherited and calculated:

- The time field needs calculation to avoid people entering wrong values
- The **Unit Price** should be calculated
- The **Unit Cost**, **Posting groups,** and **Bill-to Customer No.** need to be inherited
- As final cherry on top, we will look at implementing dimensions

Time calculation

As it comes to the times we want only to allow specific start and end times. Our Squash Court can be used in blocks of half an hour. The quantity field should be calculated based on the entered times and vice versa.

To have the most flexible solution possible we will create a new table with allowed starting and ending times. This table will have two fields; **Reservation Time** and **Duration** as shown in the following screenshot:

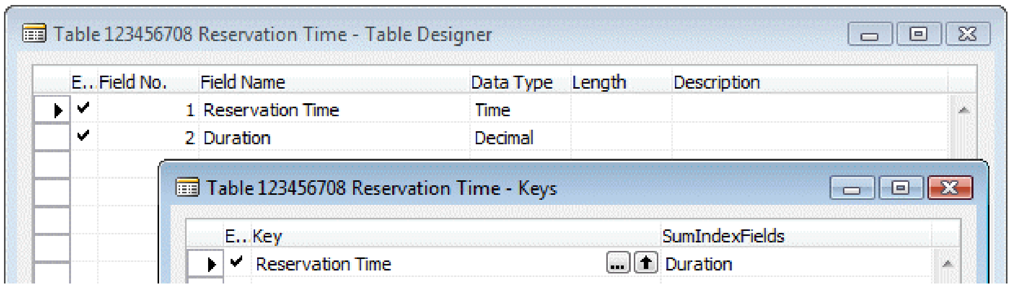


Table 123456708 Reservation Time - Table Designer

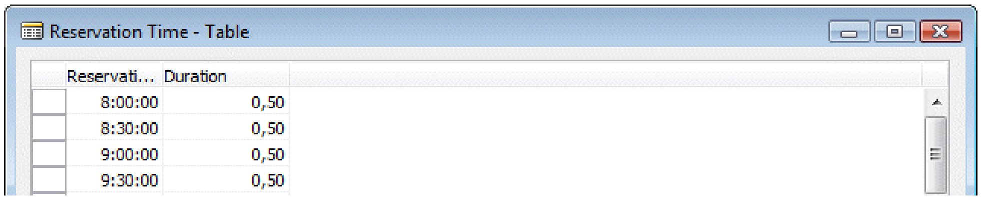
E..	Field No.	Field Name	Data Type	Length	Description
✓	1	Reservation Time	Time		
✓	2	Duration	Decimal		

Table 123456708 Reservation Time - Keys

E..	Key	SumIndexFields
✓	Reservation Time	Duration

Duration will be a decimal field that we will promote to a Sum Index Field. This will enable us to use SIFT to calculate the Quantity.

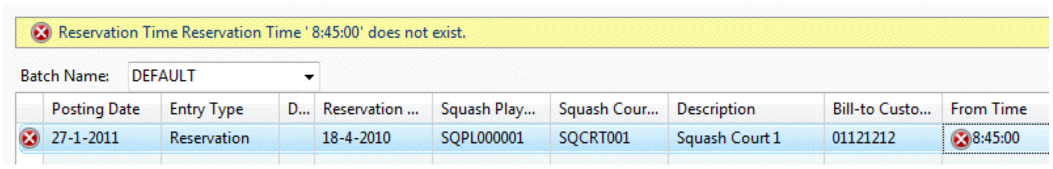
When populated the table will look like this:



Reservation Time - Table

Reservati...	Duration
8:00:00	0,50
8:30:00	0,50
9:00:00	0,50
9:30:00	0,50

The time fields in the squash journal table will now get a table relation with this table. This prevents a user from entering values that are not in the table thus allowing only valid starting and ending times. This is all done without any C/AL code and flexible when times change later.



Reservation Time Reservation Time ' 8:45:00' does not exist.

Batch Name: DEFAULT

Posting Date	Entry Type	D...	Reservation ...	Squash Play...	Squash Cour...	Description	Bill-to Custo...	From Time
27-1-2011	Reservation		18-4-2010	SQPL000001	SQCRT001	Squash Court 1	01121212	8:45:00

Now we need some code that calculates the quantity based on the user's input:

```
From Time - OnValidate()
CalcQty;

To Time - OnValidate()
CalcQty;

CalcQty()
IF ("From Time" <> 0T) AND ("To Time" <> 0T) THEN BEGIN
  IF "To Time" <= "From Time" THEN
    FIELDERROR("To Time");
  ResTime.SETRANGE("Reservation Time", "From Time",
    "To Time");
  ResTime.FIND('+');
  ResTime.NEXT(-1);
  ResTime.SETRANGE("Reservation Time", "From Time",
    ResTime."Reservation Time");
  ResTime.CALCSUMS(Duration);
  VALIDATE(Quantity, ResTime.Duration);
END;
```

When a user enters a value in the **From Time** or **To Time** the function `CalcQty` is executed. This checks if both fields have a value and then checks whether To Time is larger than From Time.

Then we place a filter on the Reservation Time table. When a user now makes a reservation from 8:00 to 9:00 there are three records in the filter making the result of the `CALCSUMS` (Total of all records) of duration 1,5. Therefore we find the previous reservation time and use that.

This example shows how easy it is to use built in Microsoft Dynamics NAV functionality like table relations and `CALCSUMS` instead of complex time calculations which we could have also used.

Price calculation

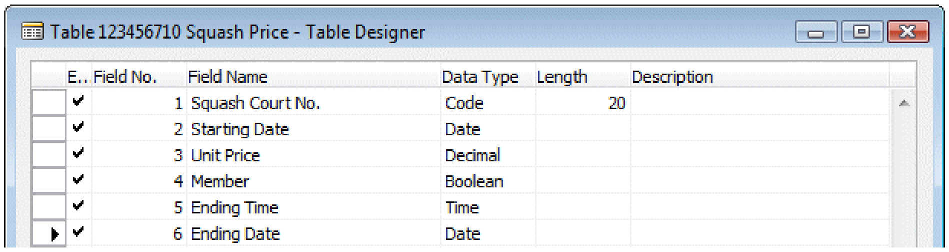
As discussed in *Chapter 1*, Microsoft Dynamics NAV has a special technique to determine prices. Prices are stored in a table with all possible parameters as fields and by filtering down on these fields the best price is determined. If required with extra logic to find the lowest (or highest) price if more prices are found.

To look, learn, and love this part of the standard application we have used table Sales Price (7002) and Sales Price Calc. Mgt. (7000) even though we only need a small part of this functionality. This mechanism of price calculation is used throughout the application and offers a normalized way of calculating sales prices. A similar construction is used for purchase prices with table purchase price (7012) and Purch. Price calc. Mgt. (7010).

Squash prices

In our case we have already determined that we have a special rate for members, but let's say we have also a special rate for daytime and evening in winter and summer.

This could make our table look like this:



E..	Field No.	Field Name	Data Type	Length	Description
✓	1	Squash Court No.	Code	20	
✓	2	Starting Date	Date		
✓	3	Unit Price	Decimal		
✓	4	Member	Boolean		
✓	5	Ending Time	Time		
✓	6	Ending Date	Date		

We can make special prices for members on dates for winter and summer and make a price only valid until a certain time. We can also make a special price for a court.

This table could be creatively expanded with all kinds of codes until we end up with table sales price (7002) in the standard product which was the template for our example.

Price calc mgt. codeunit

To calculate the price we need a codeunit similar to the standard product. This codeunit is called with a squash journal line record and stores all valid prices in a buffer table and then finds the lowest price if there is an overlap:

```
FindSquashPrice()
WITH FromSquashPrice DO BEGIN
  SETFILTER("Ending Date",'%1|>=%2',0D,StartingDate);
  SETRANGE("Starting Date",0D,StartingDate);

  ToSquashPrice.RESET;
  ToSquashPrice.DELETEALL;

  SETRANGE(Member, IsMember);

  SETRANGE("Ending Time", 0T);
  SETRANGE("Squash Court No.", '');
  CopySquashPriceToSquashPrice(FromSquashPrice,ToSquashPrice);

  SETRANGE("Ending Time", 0T);
  SETRANGE("Squash Court No.", CourtNo);
  CopySquashPriceToSquashPrice(FromSquashPrice,ToSquashPrice);

  SETRANGE("Squash Court No.", '');
  IF StartingTime <> 0T THEN BEGIN
    SETFILTER("Ending Time",'%1|>=%2',000001T,StartingTime);
    CopySquashPriceToSquashPrice(FromSquashPrice,
      ToSquashPrice);
  END;

  SETRANGE("Squash Court No.", CourtNo);
  IF StartingTime <> 0T THEN BEGIN
    SETFILTER("Ending Time",'%1|>=%2',000001T,StartingTime);
    CopySquashPriceToSquashPrice(FromSquashPrice,
      ToSquashPrice);
  END;
END;
```

If there is no price in the filter it uses the unit price from the Squash Court.

```
CalcBestUnitPrice()
WITH SquashPrice DO BEGIN
  FoundSquashPrice := FINDSET;
  IF FoundSquashPrice THEN BEGIN
```

```
      BestSquashPrice := SquashPrice;
      REPEAT
        IF SquashPrice."Unit Price" <
          BestSquashPrice."Unit Price"
        THEN
          BestSquashPrice := SquashPrice;
      UNTIL NEXT = 0;
    END;
  END;

  // No price found in agreement
  IF BestSquashPrice."Unit Price" = 0 THEN
    BestSquashPrice."Unit Price" := SquashCourt."Unit Price";

  SquashPrice := BestSquashPrice;
```

Inherited data

To use the journal for the product part of the application we want to inherit some of the fields from the master data tables. In order to make that possible we need to copy and paste these fields from other tables to our master data table and populate it.

In our example we can copy and paste the fields from the Resource table (156). We also need to add code to the `OnValidate` triggers in the Journal Line table.

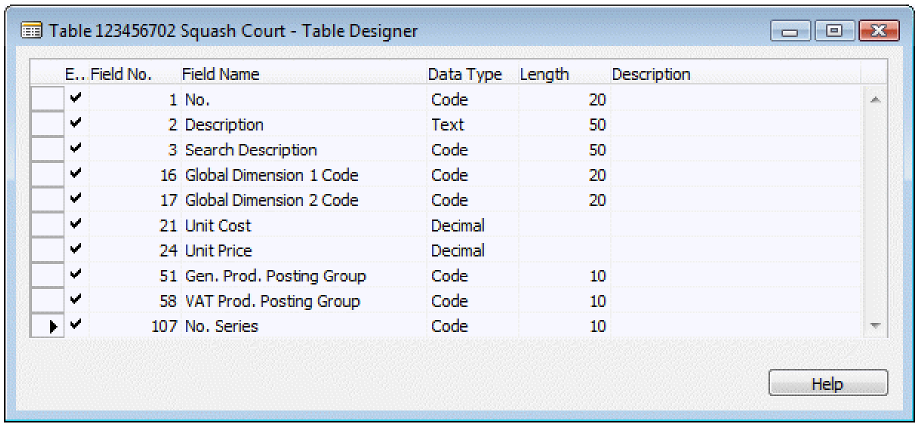


Table 123456702 Squash Court - Table Designer

E..	Field No.	Field Name	Data Type	Length	Description
✓	1	No.	Code	20	
✓	2	Description	Text	50	
✓	3	Search Description	Code	50	
✓	16	Global Dimension 1 Code	Code	20	
✓	17	Global Dimension 2 Code	Code	20	
✓	21	Unit Cost	Decimal		
✓	24	Unit Price	Decimal		
✓	51	Gen. Prod. Posting Group	Code	10	
✓	58	VAT Prod. Posting Group	Code	10	
✓	107	No. Series	Code	10	

The Squash Court table for example is expanded with the fields **Unit Code**, **Unit Price**, **Gen. Prod. Posting Group**, and **VAT Prod. Posting Group** (as shown in the previous screenshot)

We can now add code to the `OnValidate` of the `Squash Court No.` field in the journal line table:

```
Squash Court No. - OnValidate()
IF SquashCourt.GET("Squash Court No.") THEN BEGIN
  Description := SquashCourt.Description;
  "Unit Cost" := SquashCourt."Unit Cost";
  "Gen. Prod. Posting Group" := SquashCourt."Gen. Prod. Posting
Group";
  FindSquashPlayerPrice;
END;
```

Please note that unit price is used in the squash price calc. mgt. codeunit that is executed from the function `FindSquashPlayerPrice`.

Dimensions

In Microsoft Dynamics NAV, dimensions are defined in master data and posted to the ledger entries to be used in analysis view entries.

In *Chapter 3*,we will discuss how to analyse the data generated by dimensions.

In between that journey they move around a lot in different tables:

- Table 348 | Dimension: Here the main dimension codes are defined.
- Table 349 | Dimension Value: Here each dimension can have an unlimited number of values.
- Table 350 | Dimension Combination: In this table we can block certain combinations of dimension codes.
- Table 351 | Dimension Value Combination: In this table we can block certain combinations of dimension values. If this table is populated, the value **limited** is populated in the dimension combination table for these dimensions.
- Table 352 | Default Dimension: This table is populated for all master data that has dimensions defined.
- Table 354 | Default Dimension Priority: When more than one master-data record in one transaction have the same dimensions, it is possible here to set priorities.
- Table 355 | Ledger Entry Dimension: This is the history table for posted dimensions in ledger entries.
- Table 356 | Journal Line Dimension: This is a buffer table for dimensions in unposted journal lines.

- Table 357 | Document Dimension: This is a buffer table for dimensions in unposted documents.
- Table 358 | Production Document Dimension: This table saves dimensions especially for production orders.
- Table 359 | Posted Document Dimension: This is the history table for posted dimensions in posted documents.
- Codeunit 408 | DimensionManagement: This codeunit is the single point in the application where all dimension movement is done.

In our application dimensions are moved from the squash player, squash court, and customer table via the squash journal line to the squash ledger entries. When we create an invoice we move the dimensions from the ledger entries to the sales line table.

Master data

To connect dimensions to master data we first need to allow this changing codeunit 408 dimension management.

```
SetupObjectNoList()
TableIDArray[1] := DATABASE::"Salesperson/Purchaser";
TableIDArray[2] := DATABASE::"G/L Account";
TableIDArray[3] := DATABASE::Customer;
...
TableIDArray[22] := DATABASE::"Service Item Group";
TableIDArray[23] := DATABASE::"Service Item";

//* Squash Application
TableIDArray[49] := DATABASE::"Squash Player";
TableIDArray[50] := DATABASE::"Squash Court";
//* Squash Application

Object.SETRANGE(Type,Object.Type::Table);

FOR Index := 1 TO ARRAYLEN(TableIDArray) DO BEGIN
  ...
```

The variable `TableIDArray` has a default number of 23 dimensions. We have changed this to 50.

By leaving gaps we allow Microsoft to add master data tables in the future without us having to change our code

Without this change, the system would return an error message when we tried to use dimensions.

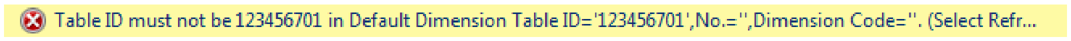
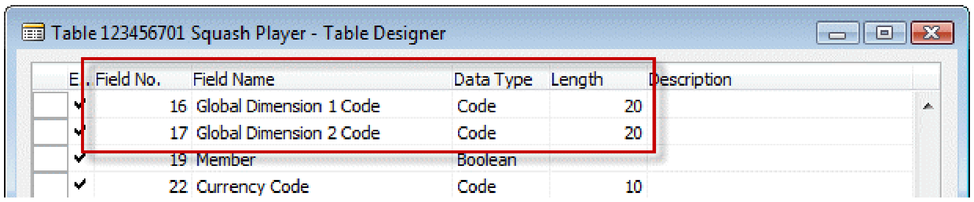


Table ID must not be 123456701 in Default Dimension Table ID='123456701',No.='',Dimension Code=''. (Select Refr...

Next change is to add the **Global Dimension** fields to the master data tables. They can be copied and pasted from other master data tables (as shown in the following screenshot):



E.	Field No.	Field Name	Data Type	Length	Description
✓	16	Global Dimension 1 Code	Code	20	
✓	17	Global Dimension 2 Code	Code	20	
✓	19	Member	Boolean		
✓	22	Currency Code	Code	10	

When these fields are validated the function `ValidateShortcutDimCode` is executed.

```
ValidateShortcutDimCode()
DimMgt.ValidateDimValueCode(FieldNumber,ShortcutDimCode);
DimMgt.SaveDefaultDim(DATABASE::"Squash Player","No.",
  FieldNumber,ShortcutDimCode);
MODIFY;
```

Journal

When we use the master data records in the Journal table, the dimensions are copied from the default dimension table to the journal line dimension table. This is done using this piece of code that is called from the `OnValidate` of each master data reference field:

```
CreateDim()
TableID[1] := Type1;
No[1] := No1;
TableID[2] := Type2;
No[2] := No2;
TableID[3] := Type3;
No[3] := No3;
"Shortcut Dimension 1 Code" := '';
"Shortcut Dimension 2 Code" := '';
DimMgt.GetDefaultDim(
  TableID,No,"Source Code",
  "Shortcut Dimension 1 Code","Shortcut Dimension 2 Code");
```

```
IF "Line No." <> 0 THEN
  DimMgt.UpdateJnlLineDefaultDim(
    DATABASE::"Squash Journal Line","Journal Template Name",
    "Journal Batch Name","Line No.",0,
    "Shortcut Dimension 1 Code","Shortcut Dimension 2 Code");
```

To decide which dimensions to inherit we should first analyze which master data is used in our journal that is using default dimensions.

```
Squash Court No. - OnValidate()
CreateDim(
  DATABASE::"Squash Court","Squash Court No.",
  DATABASE::"Squash Player","Squash Player No.",
  DATABASE::Customer,"Bill-to Customer No.");
```

In our case `Table[1]` is Squash Player, `Table[2]` is Squash Court, and `Table[3]` is Customer. The dimension management codeunit makes sure everything is copied. We can use standard Microsoft Dynamics NAV functions.

Posting

When we post a journal using codeunit squash jnl.-post line (123456703) the dimensions are first copied into a posting buffer table.

```
RunWithCheck()
SquashJnlLine.COPY(SquashJnlLine2);
TempJnlLineDim.RESET;
TempJnlLineDim.DELETEALL;
DimMgt.CopyJnlLineDimToJnlLineDim(TempJnlLineDim2,
  TempJnlLineDim);
Code;
SquashJnlLine2 := SquashJnlLine;
```

Then after we have created the ledger entry and know the number, the ledger entry dimensions are created. Again we use standard Microsoft Dynamics NAV functions without changing them.

```
Code()
...
SquashLedgEntry.INSERT;

DimMgt.MoveJnlLineDimToLedgEntryDim(
  TempJnlLineDim,DATABASE::"Squash Ledger Entry",
    SquashLedgEntry."Entry No.");

NextEntryNo := NextEntryNo + 1;
```

Document

For moving ledger entry dimensions to document dimensions, Microsoft Dynamics NAV does not have a standard function, so we have copied the function `CopyLedgEntryDimToJnlLnDim` to `CopyLedgEntryDimToDocDim`.

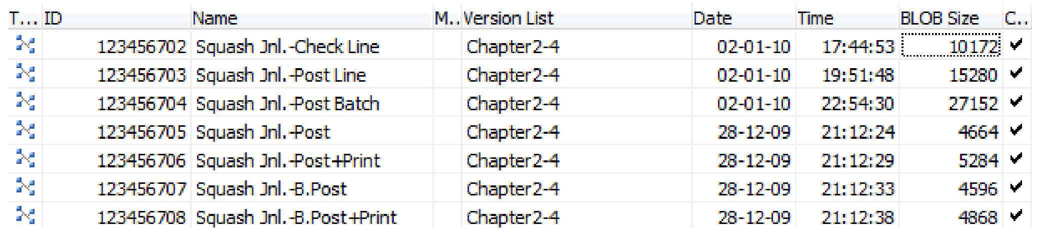
```
CopyLedgEntryDimToDocDim()
WITH FromLedgEntryDim DO
  IF FINDSET THEN
    REPEAT
      ToDocDim."Dimension Code" := "Dimension Code";
      ToDocDim."Dimension Value Code" :=
        "Dimension Value Code";
      ToDocDim.INSERT;
    UNTIL NEXT = 0;
```

This function we can then call from our combine invoicing report, which we will create later in this chapter in the invoicing section.

```
CreateLn()
...
DocDim."Table ID" := DATABASE::"Sales Line";
DocDim."Document Type" := SalesLn."Document Type";
DocDim."Document No." := SalesLn."Document No.";
DocDim."Line No." := SalesLn."Line No.";

LedgEntryDim.SETRANGE("Table ID",
  DATABASE::"Squash Ledger Entry");
LedgEntryDim.SETRANGE("Entry No.", "Entry No.");
DimMgt.CopyLedgEntryDimToDocDim(LedgEntryDim, DocDim);
```

The posting process

Our journal is now ready to be posted. We've implemented all business logic, except the posting code.



T...	ID	Name	M..	Version List	Date	Time	BLOB Size	C..
	123456702	Squash Jnl.-Check Line		Chapter2-4	02-01-10	17:44:53	10172	✓
	123456703	Squash Jnl.-Post Line		Chapter2-4	02-01-10	19:51:48	15280	✓
	123456704	Squash Jnl.-Post Batch		Chapter2-4	02-01-10	22:54:30	27152	✓
	123456705	Squash Jnl.-Post		Chapter2-4	28-12-09	21:12:24	4664	✓
	123456706	Squash Jnl.-Post+Print		Chapter2-4	28-12-09	21:12:29	5284	✓
	123456707	Squash Jnl.-B.Post		Chapter2-4	28-12-09	21:12:33	4596	✓
	123456708	Squash Jnl.-B.Post+Print		Chapter2-4	28-12-09	21:12:38	4868	✓

The posting process of a journal in Microsoft Dynamics NAV has several codeunits for the structure:

- **Jnl.-Check Line**: This codeunit checks if the journal line is valid for posting
- **Jnl.-Post Line**: This codeunit does the actual creation of the ledger entry and register tables, and calls other jnl.-post line codeunits if necessary to provide the transaction structure in *Chapter 1*.
- **Jnl.-Post Batch**: This codeunit loops through all journal lines in a journal batch and posts all the lines. There is a dialog.
- **Jnl.-Post**: This is the codeunit that is called from the page. It calls the codeunit jnl.-post batch and takes care of some user messaging.
- **Jnl.-Post+Print**: This is the codeunit that is called when you push post + print. It does the same as the jnl.-post codeunit but with the additional printing of a report defined in the journal template.
- **Jnl.-B.Post**: Posts all the journal lines that have no errors and marks the ones that have errors.
- **Jnl.-B.Post+Print**: Does the same as Jnl.-B.Post but with the additional printing of a report defined in the journal template.

Check line

Let's have a look at the check line codeunit. When it comes to testing, Microsoft Dynamics NAV has a simple rule:

Test near, Test far, Do-it, Clean up

First we need to test the field in the journal line table, then read external data tables to check if all is good, then post the lines and delete the data from the journal table.

It does not make sense to read the G/L setup table from the database if the document no. in our own table is blank, or to start the posting process and error out because the Posting Date is outside of a valid range. This would cause a lot of unnecessary I/O from the database to the client.

```
RunCheck()
WITH SquashJnlLine DO BEGIN
  IF EmptyLine THEN
    EXIT;

  TESTFIELD("Squash Player No.");
  TESTFIELD("Squash Court No.");
  TESTFIELD("Posting Date");
  TESTFIELD("Gen. Prod. Posting Group");
```

```
    TESTFIELD("From Time");
    TESTFIELD("To Time");
    TESTFIELD("Reservation Date");
    TESTFIELD("Bill-to Customer No.");
    IF "Entry Type" = "Entry Type"::Invoice THEN
      TESTFIELD("Applies-to Entry No.");
    IF "Applies-to Entry No." <> 0 THEN
      TESTFIELD("Entry Type", "Entry Type"::Invoice);
    IF "Posting Date" <> NORMALDATE("Posting Date") THEN
      FIELDERROR("Posting Date",Text000);
    IF (AllowPostingFrom = 0D) AND (AllowPostingTo = 0D) THEN
      ...
    END;
    ...
    IF NOT DimMgt.CheckJnlLineDimComb(JnlLineDim) THEN
      ...
    TableID[1] := DATABASE::"Squash Player";
    No[1] := "Squash Player No.";
    ...
    IF NOT DimMgt.CheckJnlLineDimValuePosting(JnlLineDim,
      TableID,No)
    THEN
      IF "Line No." <> 0 THEN
        ...
```

Here we can clearly see that fields in our table are checked first, then the date validation, and lastly the dimension checking.

Post line

The actual posting code turns out to be quite simple. The values are checked and then a Register is created or updated.

```
  Code()
  WITH SquashJnlLine DO BEGIN
    IF EmptyLine THEN
      EXIT;
    SquashJnlCheckLine.RunCheck(SquashJnlLine,TempJnlLineDim);
    IF NextEntryNo = 0 THEN BEGIN
      SquashLedgEntry.LOCKTABLE;
      IF SquashLedgEntry.FIND('+') THEN
        NextEntryNo := SquashLedgEntry."Entry No.";
      NextEntryNo := NextEntryNo + 1;
    END;
    IF SquashReg."No." = 0 THEN BEGIN
      SquashReg.LOCKTABLE;
      IF (NOT SquashReg.FIND('+')) OR ... THEN BEGIN
        SquashReg.INIT;
        SquashReg."No." := SquashReg."No." + 1;
        ...
        SquashReg.INSERT;
      END;
    END;
    SquashReg."To Entry No." := NextEntryNo;
    SquashReg.MODIFY;
    SquashPlayer.GET("Squash Player No.");
    SquashPlayer.TESTFIELD(Blocked,FALSE);
    IF (GenPostingSetup."Gen. Bus. Posting Group" <>
      "Gen. Bus. Posting Group") OR
      (GenPostingSetup."Gen. Prod. Posting Group" <>
      "Gen. Prod. Posting Group")
    THEN
      GenPostingSetup.GET("Gen. Bus. Posting Group",
        "Gen. Prod. Posting Group");
    SquashLedgEntry.INIT;
    SquashLedgEntry."Entry Type" := "Entry Type";
    SquashLedgEntry."Document No." := "Document No.";
    ...
    SquashLedgEntry."No. Series" := "Posting No. Series";
    SquashLedgEntry.INSERT;
```

All the fields are simply moved to the ledger entry table.

This is what makes Microsoft Dynamics NAV simple and powerful.

[Here we can clearly see how easy it is to add a field to a posting process. Just add the fields to the journal line, the ledger entry, and add one line of code to the posting process.]

Invoicing

The last issue on our to-do list is the invoicing process. For this we use a part of the standard application.

As explained in *Chapter 1*, invoicing is done using a document structure with a header and a line table. This has a posting routine that will start the journal transactions.

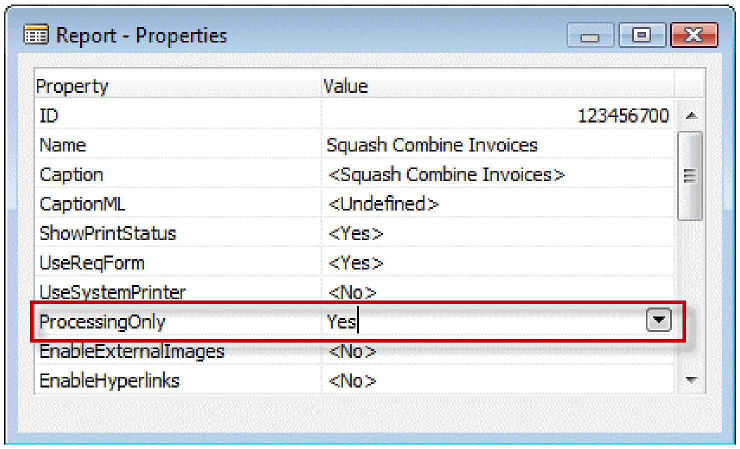
For our application we need to create the invoice document and make sure that when posted, it updates our sub administration.

Invoice document

The sales invoice documents in Microsoft Dynamics NAV are stored in the sales header (36) and sales line (37) tables. We will create a report that will combine the outstanding reservation entries into invoices allowing the user to filter on a specific entry or any other field value in the squash ledger entry table.

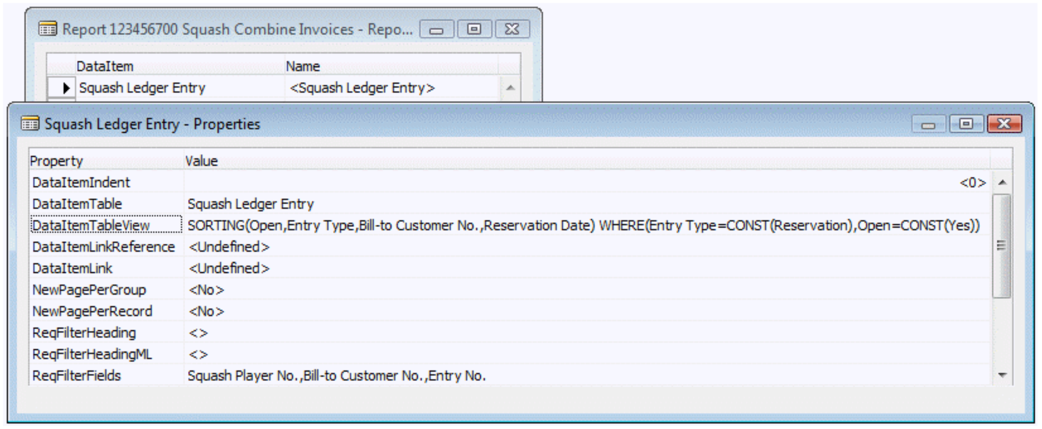
Reports in Microsoft Dynamics NAV are not just for printing documents, we can also use its dataset capabilities to start batch jobs.

To enable this, our batch job needs to have a special property, **ProcessingOnly** (as shown in the following screenshot), so let's start a blank report and do this.



Property	Value
ID	123456700
Name	Squash Combine Invoices
Caption	<Squash Combine Invoices>
CaptionML	<Undefined>
ShowPrintStatus	<Yes>
UseReqForm	<Yes>
UseSystemPrinter	<No>
ProcessingOnly	Yes
EnableExternalImages	<No>
EnableHyperlinks	<No>

The report will browse through the squash ledger entries filtered on entry type **reservation** and open (**yes)**. The sorting is **Open**, **Entry Type**, **Bill-to Customer No.**, and **Reservation Date** (as shown in the following screenshot):


Report 123456700 Squash Combine Invoices - Repo...
Squash Ledger Entry - Properties

DataItem	Name
Squash Ledger Entry	<Squash Ledger Entry>

Property	Value
DataItemIndent	<0>
DataItemTable	Squash Ledger Entry
DataItemTableView	SORTING(Open,Entry Type,Bill-to Customer No.,Reservation Date) WHERE(Entry Type=CONST(Reservation),Open=CONST(Yes))
DataItemLinkReference	<Undefined>
DataItemLink	<Undefined>
NewPagePerGroup	<No>
NewPagePerRecord	<No>
ReqFilterHeading	<>
ReqFilterHeadingML	<>
ReqFilterFields	Squash Player No.,Bill-to Customer No.,Entry No.

Because bill-to customer no. is the first non filtered value in the sorting we can assume that if this value changes we need a new sales header.

For every **Squash Ledger Entry** we will generate a sales line.

```
Squash Ledger Entry - OnAfterGetRecord()
IF "Bill-to Customer No." <> SalesHdr."Bill-to Customer No."
THEN
  CreateSalesHdr;

CreateLn;
```

Sales header

The code to create a sales header looks like this:

```
CreateSalesHdr()
CLEAR(SalesHdr);
SalesHdr.SetHideValidationDialog(TRUE);
SalesHdr."Document Type" := SalesHdr."Document Type"::Invoice;
SalesHdr."Document Date" := WORKDATE;
SalesHdr."Posting Date" := WORKDATE;
SalesHdr.VALIDATE("Sell-to Customer No.",
  "Squash Ledger Entry"."Bill-to Customer No.");
SalesHdr.INSERT(TRUE);
NextLineNo := 10000;
CounterOK := CounterOK + 1;
```

The function `SetHideValidationDialog` makes sure that we don't get pop-up messages while validating values. This is a standard function in Microsoft Dynamics NAV which is designed for this purpose.

Sales line

To create a sales line we need a minimum of this code. Please note that we added the field **Applies-to Squash Entry No.** to the sales line table.

```
CreateLn()
WITH "Squash Ledger Entry" DO BEGIN
  GenPstSetup.GET("Gen. Bus. Posting Group",
    "Gen. Prod. Posting Group");
  GenPstSetup.TESTFIELD("Sales Account");

  SalesLn.INIT;
  SalesLn."Document Type" := SalesHdr."Document Type";
  SalesLn."Document No." := SalesHdr."No.";
  SalesLn."Line No." := NextLineNo;
  SalesLn."System-Created Entry" := TRUE;

  SalesLn.Type := SalesLn.Type::"G/L Account";
  SalesLn.VALIDATE("No.", GenPstSetup."Sales Account");
  SalesLn.Description := Description;

  SalesLn.VALIDATE(Quantity, Quantity);
  SalesLn.VALIDATE("Unit Price", "Unit Price");
  SalesLn.VALIDATE("Unit Cost (LCY)", "Unit Cost");

  SalesLn."Applies-to Squash Entry No." := "Entry No.";
  SalesLn.INSERT(TRUE);

END;
NextLineNo := NextLineNo + 10000;
```

When you add fields to the sales and purchase document tables, make sure to also add these to the posted equivalents of these tables with the same number. This way you make sure that the information is copied to the historic data. This is done using the `TRANSFERFIELDS` command. We will discuss these tables in *Chapter 6*.

Dialog

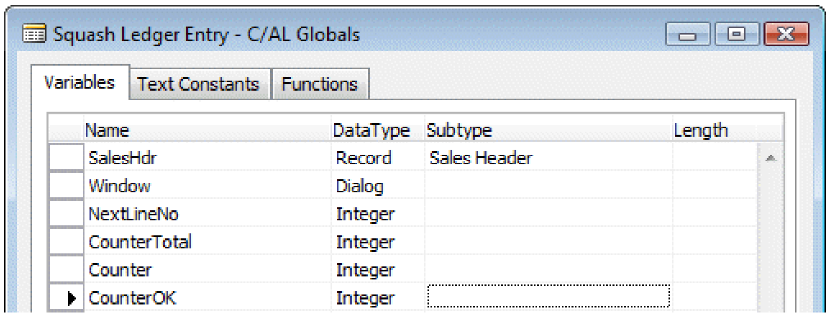
If the combined invoicing takes some time it might be good to show the user a process bar. For this Microsoft Dynamics NAV has a standard structure.

The window shows the bill-to customer no. it is currently processing and a bar going from 1 percent to 100 percent. This is calculated by keeping a counter.

At the end of the process we show a message telling the user how many invoices were created out of the number of squash ledger entries.

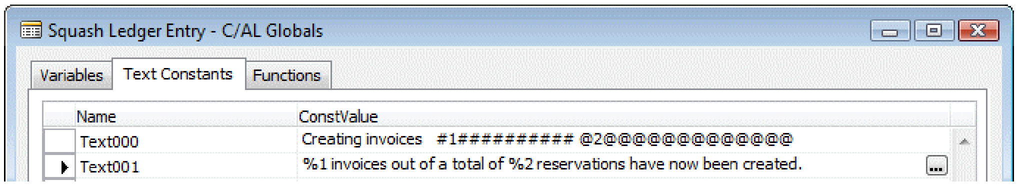
```
Squash Ledger Entry - OnPreDataItem()
CounterTotal := COUNT;
Window.OPEN(Text000);

Squash Ledger Entry - OnAfterGetRecord()
Counter := Counter + 1;
Window.UPDATE(1,"Bill-to Customer No.");
Window.UPDATE(2,ROUND(Counter / CounterTotal * 10000,1));

...

Squash Ledger Entry - OnPostDataItem()
Window.CLOSE;
MESSAGE(Text001,CounterOK,CounterTotal);
```

To do this we need some variables. The **Window** variable is of type **Dialog** whilst **Counter**, **CounterTotal**, and **CounterOK** are integers (as shown in the following screenshot):



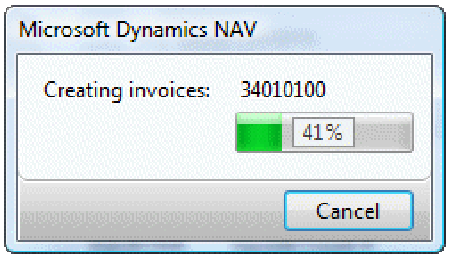
Name	DataType	Subtype	Length
SalesHdr	Record	Sales Header	
Window	Dialog		
NextLineNo	Integer		
CounterTotal	Integer		
Counter	Integer		
CounterOK	Integer		

The constant **Text000** has special values **#1##########** and **@2@@@@@@@@@@@@@** (as shown in the following screenshot). The first allows us to show and update the text; the latter is to create the process bar:

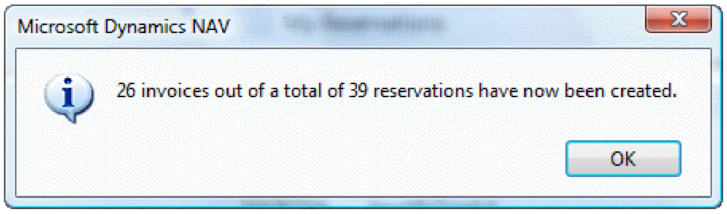


Name	ConstValue
Text000	Creating invoices #1########## @2@@@@@@@@@@@@@
Text001	%1 invoices out of a total of %2 reservations have now been created.

The result is this:



We see the following window when invoice creation process completes:



26 invoices out of a total of 39 reservations have now been created.

There is a best practice document about using progress bars in combination with the impact on performance at `http://www.mibuso.com/howtoinfo.asp?FileID=17`

Posting process

Now, our sales invoice is ready so we can start making the necessary changes to the posting process. Posting a sales document is done using a single posting codeunit and some helper objects:

- **Report 297**: This report can be used to post more than one document at the same time with a filter.

- **Codeunit 80**: This is the actual posting routine we are going to change.
- **Codeunit 81**: This codeunit is called from the user interface and has a dialog whether the user wants to ship, invoice or both if the document is an order, and a yes/no if the document is an invoice or credit memo.
- **Codeunit 82**: When the user chooses post and print this codeunit is executed which does the same as codeunit 81 plus printing a report.

So we will make a change to codeunit 80. This codeunit has a specific structure that we need to understand before we go in and make the change.

Analyse the object

The codeunit also has the *Test Near, Test Far, Do it and Cleanup* strategy so the first code is to make sure that everything is in place before the actual posting starts. Let's have a look at how this codeunit is structured.

[The Sales-Post codeunit is too long to discuss in detail. We will focus on the most important parts and learning how to read this type of code routines.]

This first part does the test near and a part of the test far. The `Ship`, `Invoice`, and `Receive` fields are set in codeunit 81 and 82 but checked and completed to make sure:

```
Code()
...
WITH SalesHeader DO BEGIN
  TESTFIELD("Document Type");
  TESTFIELD("Sell-to Customer No.");
  TESTFIELD("Bill-to Customer No.");
  TESTFIELD("Posting Date");
  TESTFIELD("Document Date");
  IF GenJnlCheckLine.DateNotAllowed("Posting Date") THEN
    FIELDERROR("Posting Date",Text045);

  CASE "Document Type" OF
    "Document Type"::Order:
      Receive := FALSE;
    "Document Type"::Invoice:
      BEGIN
        Ship := TRUE;
        Invoice := TRUE;
        Receive := FALSE;
      END;
    "Document Type"::"Return Order":
      Ship := FALSE;
    "Document Type"::"Credit Memo":
      BEGIN
        Ship := FALSE;
        Invoice := TRUE;
        Receive := TRUE;
      END;
  END;

  IF NOT (Ship OR Invoice OR Receive) THEN
    ERROR(...);

  WhseReference := "Posting from Whse. Ref.";
  "Posting from Whse. Ref." := 0;

  IF Invoice THEN
    CreatePrepaymentLines(...);
  CopyAndCheckDocDimToTempDocDim;
```

The next step is to move the sales header information to the history tables for shipment, invoice, credit memo, or return receipt header. These sections are commented like this:

```
// Insert invoice header or credit memo header
IF Invoice THEN
  IF "Document Type" IN ["Document Type"::Order,
    "Document Type"::Invoice]
  THEN BEGIN
    SalesInvHeader.INIT;
    SalesInvHeader.TRANSFERFIELDS(SalesHeader);
```

We will discuss the relation between a sales header and the sales shipment, sales invoice, sales credit memo, and return receipt in *Chapter 6*.

When this is done, the sales lines are processed. They are also moved to the various posted line tables. This is all part of the *Do-it* section of the posting routine.

```
// Lines
InvPostingBuffer[1].DELETEALL;
DropShipPostBuffer.DELETEALL;
EverythingInvoiced := TRUE;

SalesLine.RESET;
SalesLine.SETRANGE("Document Type","Document Type");
SalesLine.SETRANGE("Document No.","No.");
```

```
LineCount := 0;
RoundingLineInserted := FALSE;
MergeSaleslines(...);
```

If there is a drop shipment in a purchase order this is handled here. We will discuss Drop Shipments in *Chapter 6*.

```
// Post drop shipment of purchase order
PurchSetup.GET;
IF DropShipPostBuffer.FIND('-') THEN
  REPEAT
    PurchOrderHeader.GET(
      PurchOrderHeader."Document Type"::Order,
      DropShipPostBuffer."Order No.");
```

Then there is a section that creates the financial information in the general journal. We will go deeper into this section in *Chapter 3*:

```
IF Invoice THEN BEGIN
  // Post sales and VAT to G/L entries from posting buffer
  LineCount := 0;
  IF InvPostingBuffer[1].FIND('+') THEN
    REPEAT
      LineCount := LineCount + 1;
      Window.UPDATE(3,LineCount);

      GenJnlLine.INIT;
      GenJnlLine."Posting Date" := "Posting Date";
      GenJnlLine."Document Date" := "Document Date";
```

Then the *Clean up* section starts by calculating remaining quantities, VAT, and deleting the sales header and sales lines:

```
IF ("Document Type" IN ["Document Type"::Order,
  "Document Type"::"Return Order"]) AND
   (NOT EverythingInvoiced)
THEN BEGIN
  MODIFY;
  // Insert T336 records
  InsertTrackingSpecification;

  IF SalesLine.FINDSET THEN
    REPEAT
      IF SalesLine.Quantity <> 0 THEN BEGIN
        IF Ship THEN BEGIN
          SalesLine."Quantity Shipped" :=
            SalesLine."Quantity Shipped" +
            SalesLine."Qty. to Ship";
          SalesLine."Qty. Shipped (Base)" :=
            SalesLine."Qty. Shipped (Base)" +
            SalesLine."Qty. to Ship (Base)";
        END;
```

The *Clean up* ends by deleting the sales document and related information, and clearing the variables used:

```
IF HASLINKS THEN DELETELINKS;
DELETE;
...
SalesLine.DELETEALL;
DeleteItemChargeAssgnt;
...
CLEAR(WhsePostRcpt);
CLEAR(WhsePostShpt);
...
CLEAR(WhseJnlPostLine);
CLEAR(InvtAdjmt);
Window.CLOSE;
```

Making the change

The change we are going to make is in the section where the lines are handled:

```
// Squash Journal Line
IF SalesLine."Applies-to Squash Entry No." <> 0 THEN
  PostSquashJnlLn;
IF (SalesLine.Type >= SalesLine.Type::"G/L Account") AND
  (SalesLine."Qty. to Invoice" <> 0)
THEN BEGIN
  // Copy sales to buffer
```

We will create new function `PostSquashJnlLn`. This way we minimize the impact on standard code. And when upgrading to a newer version we can easily copy and paste our function, and only need to change the calling place if required.

Always try to design for easy upgrading whenever possible. Remember that Microsoft might change this code in newer versions so the more flexible we are and the more we minimize the impact on standard code the better.

```
PostSquashJnlLn()
WITH SalesHeader DO BEGIN
```

```
    OldSquashLedEnt.GET(
      SalesLine."Applies-to Squash Entry No.");
    OldSquashLedEnt.TESTFIELD(Open);
    OldSquashLedEnt.TESTFIELD("Bill-to Customer No.",
      "Bill-to Customer No.");

    SquashJnlLn.INIT;
    SquashJnlLn."Posting Date" := "Posting Date";
    SquashJnlLn."Reason Code" := "Reason Code";
    ...
    SquashJnlLn."Document No." := GenJnlLineDocNo;
    SquashJnlLn."External Document No." := GenJnlLineExtDocNo;
    SquashJnlLn.Quantity := -SalesLine."Qty. to Invoice";
    SquashJnlLn."Source Code" := SrcCode;
    SquashJnlLn.Chargeable := TRUE;
    SquashJnlLn."Posting No. Series" := "Posting No. Series";
    TempJnlLineDim.DELETEALL;
    TempDocDim.RESET;
    TempDocDim.SETRANGE("Table ID",DATABASE::"Sales Line");
    TempDocDim.SETRANGE("Line No.",SalesLine."Line No.");
    DimMgt.CopyDocDimToJnlLineDim(TempDocDim,TempJnlLineDim);
    SquashJnlPostLine.RunWithCheck(SquashJnlLn,TempJnlLineDim);
  END;
```

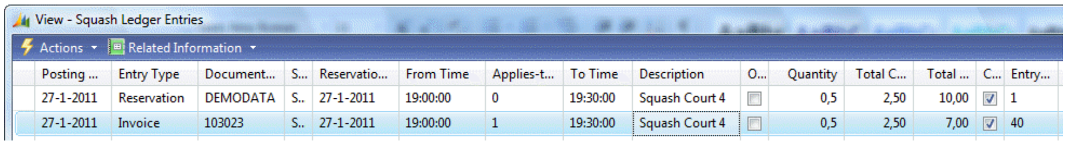
Our new function first gets the squash ledger entry it applies to, and tests if it's still open and the bill-to customer no. has not changed.

Then we populate squash journal line with the help of the sales line and the old squash ledger entry.

Then dimensions are handled and the squash journal line is posted.

Note that the journal lines are never actually inserted into the database. This is for performance and concurrency reasons. All journal transactions here are handled in the service tier cache. A journal is also never populated using `Validate`. This makes it very clear to see what happens.

Now when we post an invoice we can see that the invoice entries are created:



Posting ...	Entry Type	Document...	S...	Reservatio...	From Time	Applies-t...	To Time	Description	O...	Quantity	Total C...	Total ...	C...	Entry...
27-1-2011	Reservation	DEMODATA	S..	27-1-2011	19:00:00	0	19:30:00	Squash Court 4	☐	0,5	2,50	10,00	✓	1
27-1-2011	Invoice	103023	S..	27-1-2011	19:00:00	1	19:30:00	Squash Court 4	☐	0,5	2,50	7,00	✓	40

Navigate

We have now covered everything that is necessary for our Squash Court application to run but there is one special function of Microsoft Dynamics NAV that needs changing when we add new documents and ledger entries—the `Navigate` function.

The functionality was already discussed in *Chapter 1*. The object is a single page (344) in the application that requires two changes.

FindRecords

The first function we change is `FindRecords`. This browses though the database finding all possible combinations of document no. and posting date.

```
FindRecords()
...
// Squash Ledger Entries
IF SquashLedgEntry.READPERMISSION THEN BEGIN
  SquashLedgEntry.RESET;
  SquashLedgEntry.SETCURRENTKEY("Document No.",
    "Posting Date");
  SquashLedgEntry.SETFILTER("Document No.",DocNoFilter);
  SquashLedgEntry.SETFILTER("Posting Date",PostingDateFilter);
  InsertIntoDocEntry(
    DATABASE::"Squash Ledger Entry",0,
    SquashLedgEntry.TABLECAPTION,SquashLedgEntry.COUNT);
END;
// Squash Ledger Entries

DocExists := FINDFIRST;
```

The function first checks if we have permission to read the `Squash Ledger Entry` table. If our system administrator does not allow us to see this table it should not show up.

The filtering is done on the `Document No.` and `Posting Date`. When ready the system inserts the number of found records in the result table.

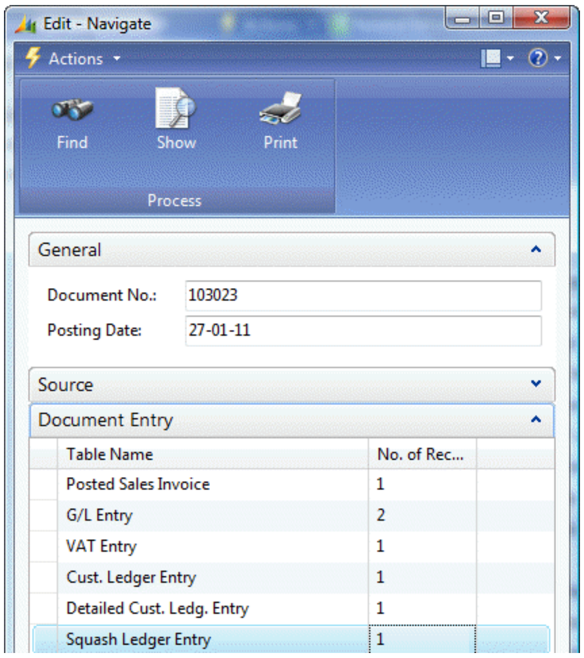
ShowRecords

Second function to change is `ShowRecords`. This makes sure we see the squash ledger entries when we push the show action.

```
ShowRecords()
...
    DATABASE::"Warranty Ledger Entry":
      FORM.RUN(0,WarrantyLedgerEntry);
//* Squash Ledger Entries
    DATABASE::"Squash Ledger Entry":
      FORM.RUN(0,SquashLedgEntry);
   END;
END;
```

Testing

Now when we navigate from the invoice we posted that was generated from our combine invoicing report we get this result:



Document No.:	103023
Posting Date:	27-01-11

Table Name	No. of Rec...
Posted Sales Invoice	1
G/L Entry	2
VAT Entry	1
Cust. Ledger Entry	1
Detailed Cust. Ledg. Entry	1
Squash Ledger Entry	1

Summary

In this chapter we created our own vertical add-on application for Microsoft Dynamics NAV. We have used similar data model and posting structures, and reused parts of the standard application where appropriate but never wrongly used standard features.

We saw how to reverse engineer Microsoft Dynamics NAV code in order to find out what similar standard functionality to copy, paste, and change for our application.

We also learned how a journal and document posting codeunit works and how to structure using *Test Near, Test Far, Do it and clean up*.

In the next chapter we will explore the financial functionality of Microsoft Dynamics NAV, and even make some changes to this part of the application.

3 Financial Management

Whether you run a company, a not-for-profit organization, or an educational institute, doing proper bookkeeping is mandatory and required by the government.

This makes financial management the most used part of Microsoft Dynamics NAV and the least obvious place to make changes, as federal regulations do not allow much creativity in this part of the application.

The first part of this chapter is all about the 'Look, Learn, and Love' principle we discussed in the previous chapter. We cannot integrate our application with Financial Management without knowing the basic functionality and structure of the tool.

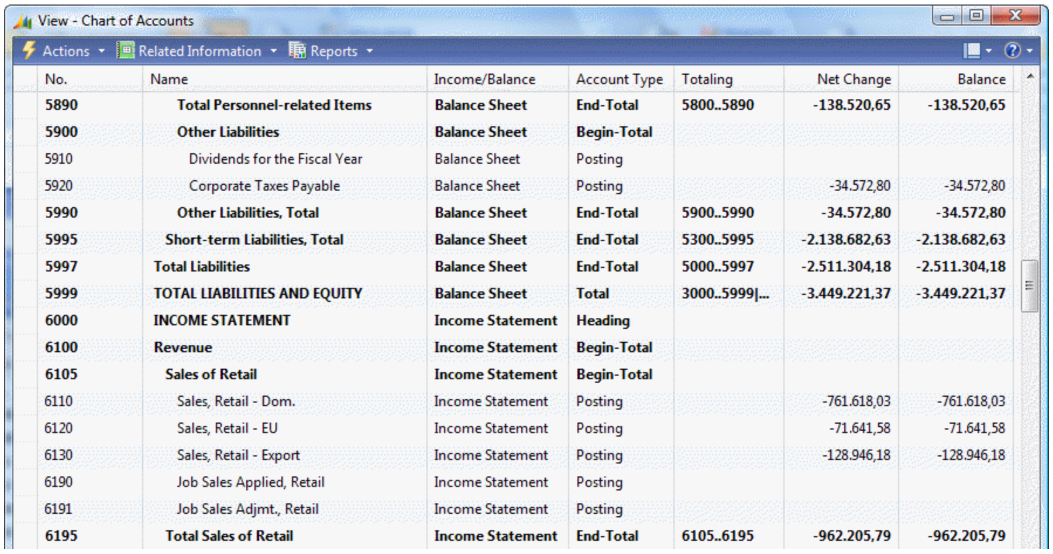
In the second part of the chapter, we will look at some examples of how to change or expand the way Financial Management works.

Lastly, we will look at how to create a posting in the General Ledger from a newly designed posting routine.

After studying this chapter, you should be able to set up Financial Management in a new database and create basic postings to the General Ledger, and you should know how to integrate Financial Management with your application.

Chart of accounts

Every financial system starts with a **Chart of Accounts** and although the numbering might differ from country to country, we all have income statements and balance accounts.



View - Chart of Accounts

Actions · Related Information · Reports

No.	Name	Income/Balance	Account Type	Totaling	Net Change	Balance
5890	Total Personnel-related Items	Balance Sheet	End-Total	5800..5890	-138.520,65	-138.520,65
5900	Other Liabilities	Balance Sheet	Begin-Total			
5910	Dividends for the Fiscal Year	Balance Sheet	Posting			
5920	Corporate Taxes Payable	Balance Sheet	Posting		-34.572,80	-34.572,80
5990	Other Liabilities, Total	Balance Sheet	End-Total	5900..5990	-34.572,80	-34.572,80
5995	Short-term Liabilities, Total	Balance Sheet	End-Total	5300..5995	-2.138.682,63	-2.138.682,63
5997	Total Liabilities	Balance Sheet	End-Total	5000..5997	-2.511.304,18	-2.511.304,18
5999	TOTAL LIABILITIES AND EQUITY	Balance Sheet	Total	3000..5999|...	-3.449.221,37	-3.449.221,37
6000	INCOME STATEMENT	Income Statement	Heading			
6100	Revenue	Income Statement	Begin-Total			
6105	Sales of Retail	Income Statement	Begin-Total			
6110	Sales, Retail - Dom.	Income Statement	Posting		-761.618,03	-761.618,03
6120	Sales, Retail - EU	Income Statement	Posting		-71.641,58	-71.641,58
6130	Sales, Retail - Export	Income Statement	Posting		-128.946,18	-128.946,18
6190	Job Sales Applied, Retail	Income Statement	Posting			
6191	Job Sales Adjmt., Retail	Income Statement	Posting			
6195	Total Sales of Retail	Income Statement	End-Total	6105..6195	-962.205,79	-962.205,79

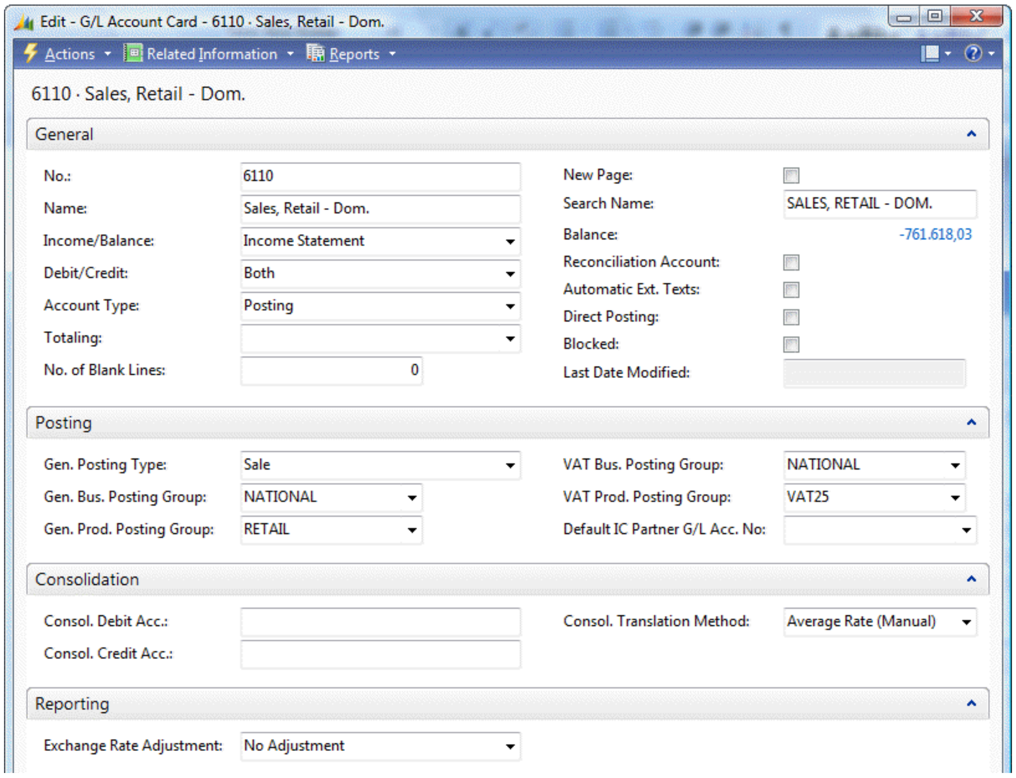
Microsoft Dynamics NAV also has some other special accounts; Heading, Begin-total, and End-total accounts.

With these accounts you can make the Chart of Accounts more readable. The accounts within the total accounts are automatically indented.

Posting accounts

When creating a new posting account, there are several options to choose from. Most of them are not mandatory but make it easier to push the end users to using the correct account while generating entries.

Let's have a look at the options by opening a **G/L Account Card**:



Edit - G/L Account Card - 6110 · Sales, Retail - Dom.

Actions · Related Information · Reports

6110 · Sales, Retail - Dom.

General

No.:	6110	New Page:	☐
Name:	Sales, Retail - Dom.	Search Name:	SALES, RETAIL - DOM.
Income/Balance:	Income Statement	Balance:	-761.618,03
Debit/Credit:	Both	Reconciliation Account:	☐
Account Type:	Posting	Automatic Ext. Texts:	☐
Totaling:		Direct Posting:	☐
No. of Blank Lines:	0	Blocked:	☐
		Last Date Modified:	

Posting

Gen. Posting Type:	Sale	VAT Bus. Posting Group:	NATIONAL
Gen. Bus. Posting Group:	NATIONAL	VAT Prod. Posting Group:	VAT25
Gen. Prod. Posting Group:	RETAIL	Default IC Partner G/L Acc. No:	

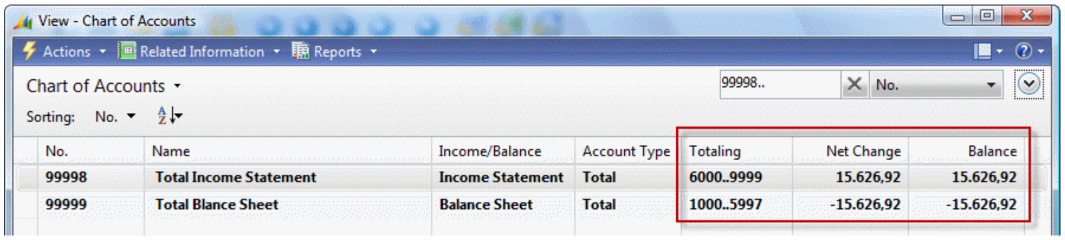
Consolidation

Consol. Debit Acc.:		Consol. Translation Method:	Average Rate (Manual)
Consol. Credit Acc.:			

Reporting

Exchange Rate Adjustment:	No Adjustment

The first and most important decision to make is the type of account. It can be an **Income Statement** or a **Balance Sheet**. Income statement accounts are reset to zero every new fiscal year while balance sheet accounts continue indefinitely. The total of the **Balance Sheet** accounts should always match the total of the **Income Statement** accounts.



View - Chart of Accounts

Actions · Related Information · Reports

Chart of Accounts · 99998.. · No.

Sorting: No.

No.	Name	Income/Balance	Account Type	Totaling	Net Change	Balance
99998	Total Income Statement	Income Statement	Total	6000..9999	15.626,92	15.626,92
99999	Total Blance Sheet	Balance Sheet	Total	1000..5997	-15.626,92	-15.626,92

You can create two total accounts to check if the balance in your G/L is accurate. Total all Income Statements and all Balance Sheets.

You can also force an account to only accept debit or credit postings. The **No. of Blank Lines** and **New Page** fields are for printing report purposes and have no effect on the system.

Reconciliation Account is hardly used anymore unless you do not use sub accounting.

Automatic Ext. Texts creates the extra texts discussed in Chapter 1 automatically when you use this account in a sales or purchasing document.

Direct Posting is a very important option. It is highly recommended to disable this option when an account is used in one of the posting setups. When Direct Posting is enabled, an end user can create entries on this account, disrupting the balance between the General Ledger and the sub administration. We'll discuss this in more detail later in this chapter.

When you do allow Direct Posting, the fields on the **Posting** tab are very important.

Gen. Posting Type determines if the account is used for purchase and/or sales for the VAT calculation and filtering in the VAT statements. More detailed VAT specification is determined by the VAT business and product posting group. The general business and product posting groups can be used to automatically populate these fields when this account is used.

In the **Consolidation** tab, you can populate the consolidation accounts used when consolidating two or more companies. We'll discuss consolidation later in this chapter.

Microsoft Dynamics NAV allows the use of an additional reporting currency. This is an inheritance from the days before the Euro in Europe and was very popular in the years of Euro introduction. Today, it is used by international companies, for example: A company based in the USA with a Dutch parent company. In the **Reporting** tab, you can determine how you want to handle exchange rate adjustments when using this functionality.

The entry tables

As discussed in Chapter 1, the entries for the General Ledger are also created in the General Journals when you post a sales or purchase document. So, let's have a closer look at this functionality and see what we can do with it.

Sub accounting

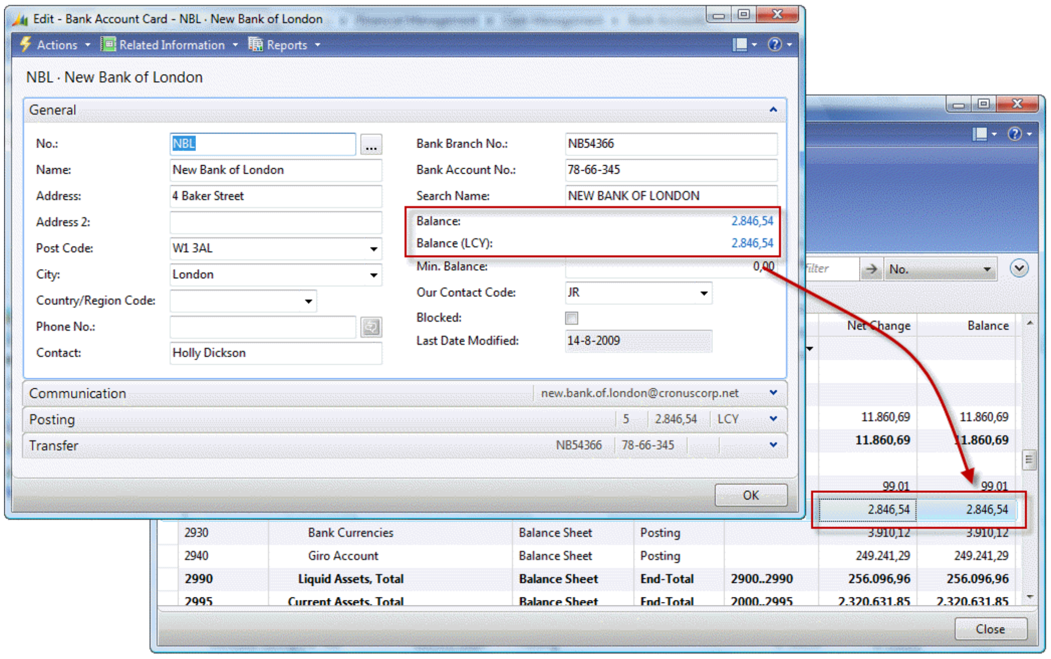
In theory, you could run Microsoft Dynamics NAV with just the G/L Entry table, but in accounting we have invented sub administrations.

Sub administrations are very old. Before computers were invented, people would have cards for all customers and vendors to keep track of their balance. Updating these cards was a manual and time consuming process with a high probability of making mistakes.

In Microsoft Dynamics NAV, this is taken care of automatically. In the General Ledger, we have the following four sub administrations:

- **Bank**: The total of all bank ledger entries should match your bank account's balance. It allows you to quickly find payments.
- **Customer**: Whenever you sell something to a customer or a customer pays, a customer ledger entry is created. It allows you to analyze payment history and send out reminders.
- **Vendor**: When we buy something from a vendor the system creates a vendor ledger entry. We can use the vendor ledger entries to determine which invoice needs to be paid. The vendor ledger entries are the opposite of the customer ledger entries.
- **VAT**: The VAT entries help us to easily create clear VAT statements.

As discussed earlier, it is very important that the total of the sub administration matches the General Ledger. For example, when your bank account balance is **2.846,54** the G/L account should also have that amount.



NBL · New Bank of London

General

No.:	NBL	Bank Branch No.:	NB54366
Name:	New Bank of London	Bank Account No.:	78-66-345
Address:	4 Baker Street	Search Name:	NEW BANK OF LONDON
Address 2:		Balance:	2.846,54
Post Code:	W1 3AL	Balance (LCY):	2.846,54
City:	London	Min. Balance:	0,00
Country/Region Code:		Our Contact Code:	JR
Phone No.:		Blocked:	☐
Contact:	Holly Dickson	Last Date Modified:	14-8-2009

Communication new.bank.of.london@cronuscorp.net

Posting 5 2.846,54 LCY

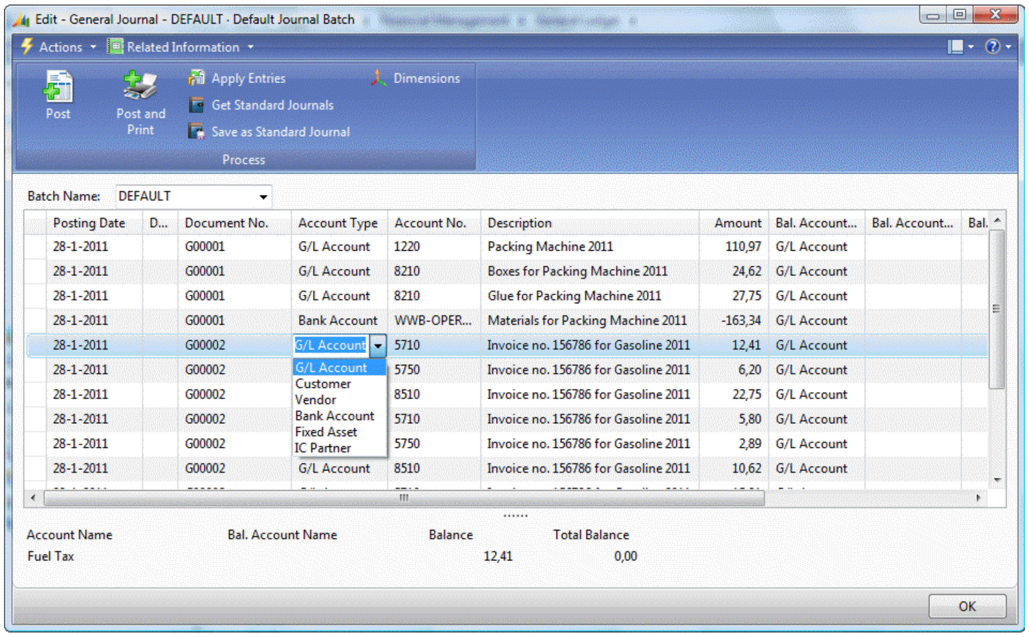
Transfer NB54366 78-66-345

					Net Change	Balance
					11.860,69	11.860,69
					11.860,69	**11.860,69**
					99.01	99.01
					2.846,54	2.846,54
2930	Bank Currencies	Balance Sheet	Posting		3.910,12	3.910,12
2940	Giro Account	Balance Sheet	Posting		249.241,29	249.241,29
2990	**Liquid Assets, Total**	**Balance Sheet**	**End-Total**	**2900..2990**	**256.096,96**	**256.096,96**
2995	**Current Assets, Total**	**Balance Sheet**	**End-Total**	**2000..2995**	**2.320.631,85**	**2.320.631,85**

For this, you can disable the **Direct Posting** option we discussed earlier.

General journals

When we open a general journal, we can start making transactions. Let's discuss the possibilities.



Batch Name: DEFAULT

Posting Date	D...	Document No.	Account Type	Account No.	Description	Amount	Bal. Account...	Bal. Account...	Bal.
28-1-2011		G00001	G/L Account	1220	Packing Machine 2011	110,97	G/L Account		
28-1-2011		G00001	G/L Account	8210	Boxes for Packing Machine 2011	24,62	G/L Account		
28-1-2011		G00001	G/L Account	8210	Glue for Packing Machine 2011	27,75	G/L Account		
28-1-2011		G00001	Bank Account	WWB-OPER...	Materials for Packing Machine 2011	-163,34	G/L Account		
28-1-2011		G00002	G/L Account	5710	Invoice no. 156786 for Gasoline 2011	12,41	G/L Account		
28-1-2011		G00002		5750	Invoice no. 156786 for Gasoline 2011	6,20	G/L Account		
28-1-2011		G00002		8510	Invoice no. 156786 for Gasoline 2011	22,75	G/L Account		
28-1-2011		G00002		5710	Invoice no. 156786 for Gasoline 2011	5,80	G/L Account		
28-1-2011		G00002		5750	Invoice no. 156786 for Gasoline 2011	2,89	G/L Account		
28-1-2011		G00002	G/L Account	8510	Invoice no. 156786 for Gasoline 2011	10,62	G/L Account		

Account Name	Bal. Account Name	Balance	Total Balance
Fuel Tax		12,41	0,00

The most important fields of a General Journal are the **Posting Date** and the **Document No**. The total of amounts for each combination of these fields should always be in balance. In other words, all journal lines for any particular combination of **Posting Date** and **Document No.** should always add up to zero.

There are different account types we can post to. When we post directly to a **G/L Account** it is clear what will happen. A new **G/L Entry** will be created for that amount. When we choose another **Account Type** the sub administrations will start to work. For example, when we choose **Customer**, a **Customer Ledger Entry** will be created as well as a **G/L Entry**. Which **G/L account** is used is determined by the posting group which we will discuss later in this chapter.

Here we also see the **Gen. Posting Type**, **General Business**, and **Product Posting** group and **VAT Business** and **Product Posting** group come back. These are inherited from the **G/L Account** we discussed earlier, but you can choose a different one if you want.

The VAT options determine the VAT calculation that is done automatically. A VAT entry is created with the VAT amount and additional G/L entries are created.

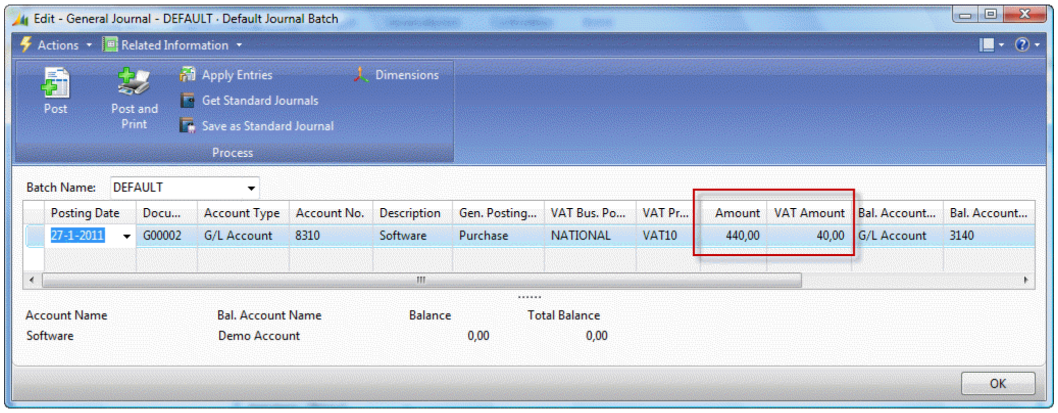
There are two ways of balancing a General Ledger. We can create two lines with the same debit and credit amount or we can use the balance fields.

Let's see some of this in an example.

We have made a purchase from an irregular vendor. All we have is a small cash receipt with the amount and the VAT which we want to bring into our company.

The amount is 440 including 10% VAT, so we want to create the following transaction

Cost	400,00
VAT	40,00
Current Account	440,00



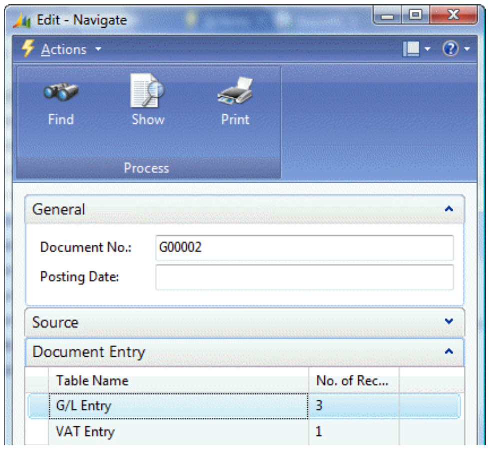
Batch Name: DEFAULT

Posting Date	Docu...	Account Type	Account No.	Description	Gen. Posting...	VAT Bus. Po...	VAT Pr...	Amount	VAT Amount	Bal. Account...	Bal. Account...
27-1-2011	G00002	G/L Account	8310	Software	Purchase	NATIONAL	VAT10	440,00	40,00	G/L Account	3140

Account Name	Bal. Account Name	Balance	Total Balance
Software	Demo Account	0,00	0,00

We can see that Microsoft Dynamics NAV calculates the **VAT Amount** and by populating the balance account we only need one line which is always in balance.

When we **Navigate** this transaction, we see that we have three **G/L Entries** and one **VAT Entry**.

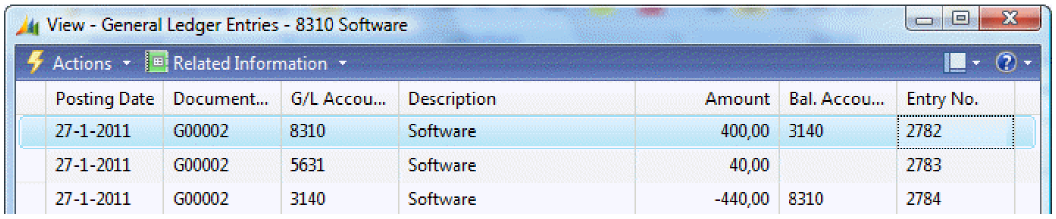


Document No.: G00002

Posting Date:

Table Name	No. of Rec...
G/L Entry	3
VAT Entry	1

And opening the **G/L Entries** shows the correct amounts.

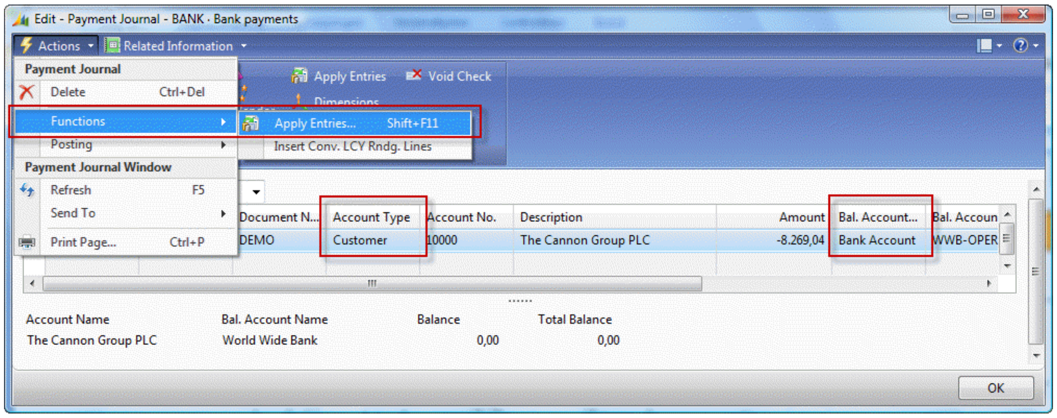


Posting Date	Document...	G/L Accou...	Description	Amount	Bal. Accou...	Entry No.
27-1-2011	G00002	8310	Software	400,00	3140	2782
27-1-2011	G00002	5631	Software	40,00		2783
27-1-2011	G00002	3140	Software	-440,00	8310	2784

In another example, we'll create a customer payment through the bank journal.

Entry application

A bank journal is a general journal with a specific Form ID. This allows the application to have a different user interface based on the same business logic. A specific feature of a bank journal is the possibility to easily apply payments to invoices.

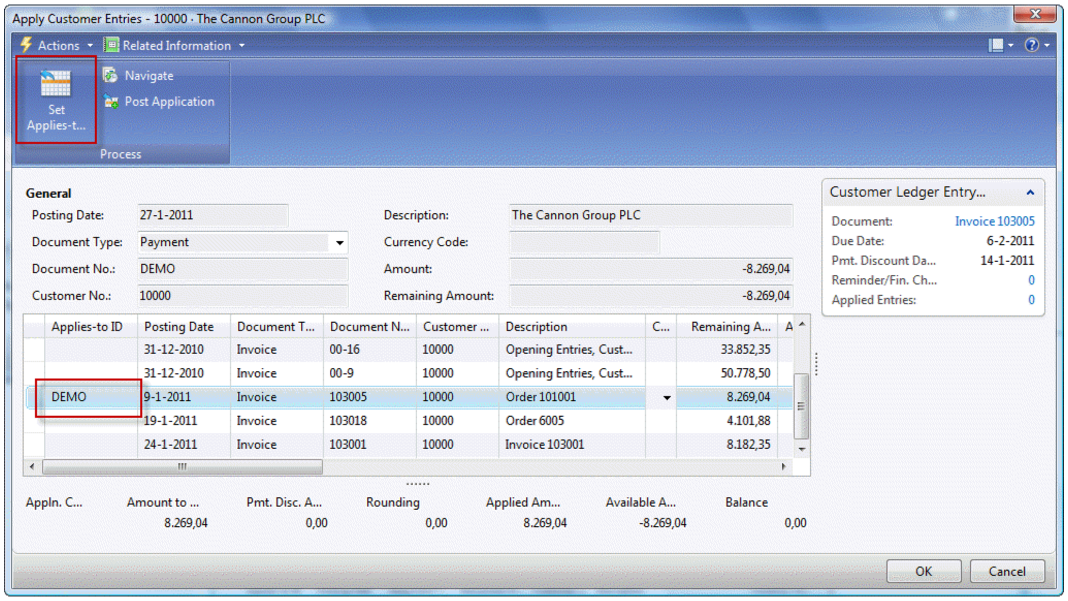


Document N...	Account Type	Account No.	Description	Amount	Bal. Account...	Bal. Accoun
DEMO	Customer	10000	The Cannon Group PLC	-8.269,04	Bank Account	WWB-OPER

Account Name	Bal. Account Name	Balance	Total Balance
The Cannon Group PLC	World Wide Bank	0,00	0,00

The bank journal does not directly post to a G/L account, but uses other account types. In this case, the **Account Type** is **Customer** and the **Balance Account Type** is a **Bank Account**. Instead of a list with G/L accounts, the **Account No.** field now refers to the **Customers** and the **Balance Account No.** fields refer to the **Bank Accounts**. The latter is automatically populated from the Journal Batch definition.

We'll use the Apply Entries feature to determine which invoice this payment applies to. If we did not do this, the system would not know which invoice is paid.



General

Posting Date:	27-1-2011	Description:	The Cannon Group PLC
Document Type:	Payment	Currency Code:	
Document No.:	DEMO	Amount:	-8.269,04
Customer No.:	10000	Remaining Amount:	-8.269,04

Customer Ledger Entry...

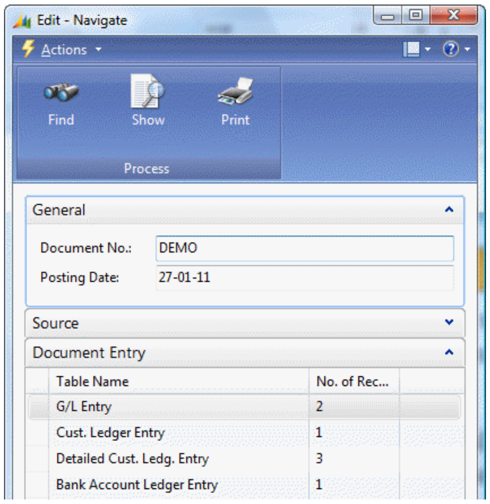
Document:	Invoice 103005
Due Date:	6-2-2011
Pmt. Discount Da...	14-1-2011
Reminder/Fin. Ch...	0
Applied Entries:	0

Applies-to ID	Posting Date	Document T...	Document N...	Customer ...	Description	C...	Remaining A...	A
	31-12-2010	Invoice	00-16	10000	Opening Entries, Cust...		33.852,35	
	31-12-2010	Invoice	00-9	10000	Opening Entries, Cust...		50.778,50	
DEMO	9-1-2011	Invoice	103005	10000	Order 101001		8.269,04	
	19-1-2011	Invoice	103018	10000	Order 6005		4.101,88	
	24-1-2011	Invoice	103001	10000	Invoice 103001		8.182,35	

Appln. C...	Amount to ...	Pmt. Disc. A...	Rounding	Applied Am...	Available A...	Balance
	8.269,04	0,00	0,00	8.269,04	-8.269,04	0,00

Another option would be to automatically apply entries, but when a customer decides to skip a payment the system might get confused, so it is highly recommended to apply entries manually.

When we post this journal and **Navigate** the entries, we see that all necessary sub administrations are updated.



General

Document No.:	DEMO
Posting Date:	27-01-11

Source

Document Entry

Table Name	No. of Rec...
G/L Entry	2
Cust. Ledger Entry	1
Detailed Cust. Ledg. Entry	3
Bank Account Ledger Entry	1

Posting groups

In the previous section, we talked about using customer numbers and bank account numbers as an account number in the General Journal. The system can then figure out what G/L Account numbers to use. But how does that work?

This is done using the various posting group matrices. Most application parts that post to the General Ledger have their own posting group table.

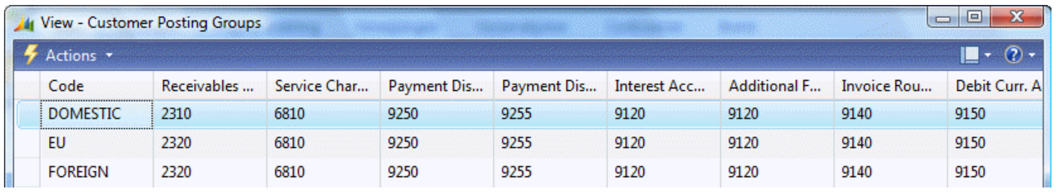
There are two types of posting groups—single layer and matrix layer.

The single layer has direct G/L Account columns, the matrix layer has an additional setup table.

Single layer	**Matrix layer**
Customer posting group	Gen. Business posting group
Vendor posting group	Gen. Product posting group
Inventory posting group	VAT business posting group
Job posting group	VAT product posting group
Bank account posting group	Inventory posting setup
FA posting group	

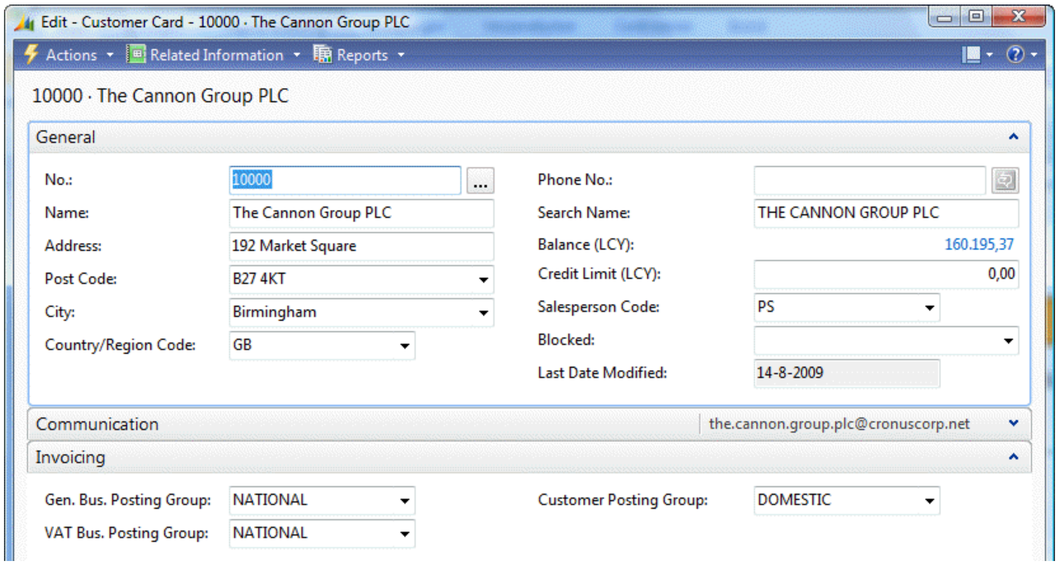
Each country uses different account schedules and regulations. The G/L accounts in this book are used for the CRONUS example database. These can be different in each country and implementation.

Let's have a look at the **Customer Posting Groups**:



Code	Receivables ...	Service Char...	Payment Dis...	Payment Dis...	Interest Acc...	Additional F...	Invoice Rou...	Debit Curr. A
DOMESTIC	2310	6810	9250	9255	9120	9120	9140	9150
EU	2320	6810	9250	9255	9120	9120	9140	9150
FOREIGN	2320	6810	9250	9255	9120	9120	9140	9150

We see three different codes having their own accounts. So, where is this code used? Let's open a customer card.



10000 · The Cannon Group PLC

General

No.:	10000	Phone No.:	
Name:	The Cannon Group PLC	Search Name:	THE CANNON GROUP PLC
Address:	192 Market Square	Balance (LCY):	160.195,37
Post Code:	B27 4KT	Credit Limit (LCY):	0,00
City:	Birmingham	Salesperson Code:	PS
Country/Region Code:	GB	Blocked:	
		Last Date Modified:	14-8-2009

Communication the.cannon.group.plc@cronuscorp.net

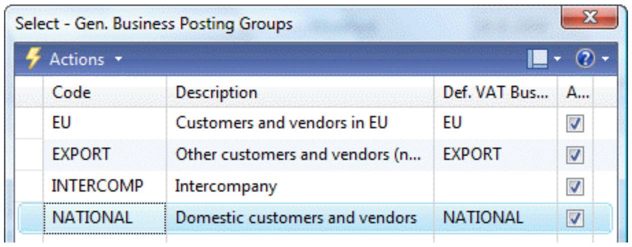
Invoicing

Gen. Bus. Posting Group:	NATIONAL	Customer Posting Group:	DOMESTIC
VAT Bus. Posting Group:	NATIONAL		

We see it on the invoicing tab. This is what determines the Customer G/L accounts.

We also see other posting groups on the Customer card. There is a **Gen. Bus. Posting Group** and a **VAT Bus. Posting Group**.

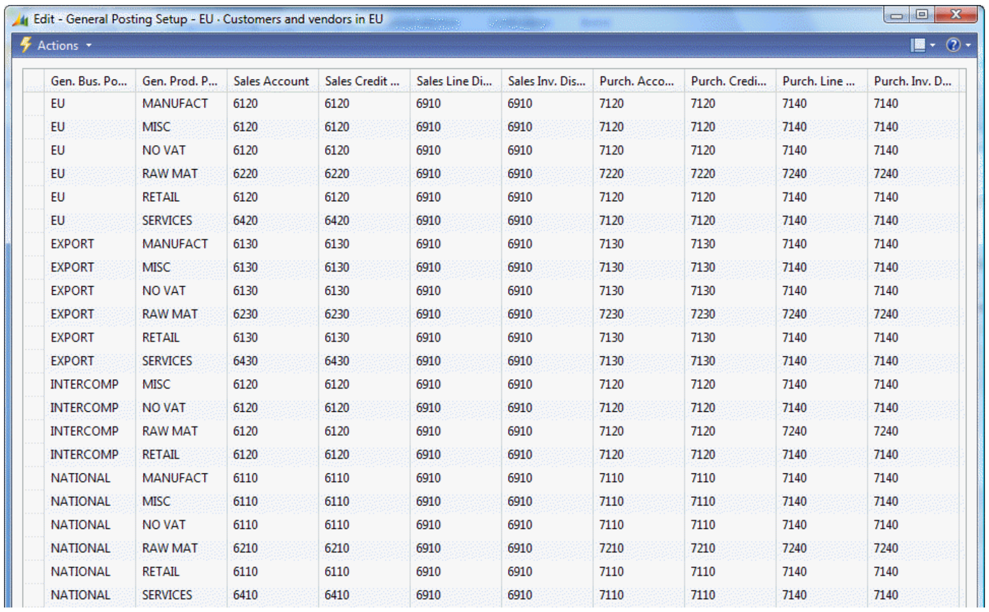
In our list, they are matrix layers. So they don't directly point to a G/L account. When we open the **Gen. Bus. Posting Group**, we see this.



Code	Description	Def. VAT Bus...	A...
EU	Customers and vendors in EU	EU	✓
EXPORT	Other customers and vendors (n...	EXPORT	✓
INTERCOMP	Intercompany		✓
NATIONAL	Domestic customers and vendors	NATIONAL	✓

Just a simple table connecting it to a **Default VAT Business Posting Group**.

To see where the G/L accounts are defined, we need to go to the **General Posting Setup**.



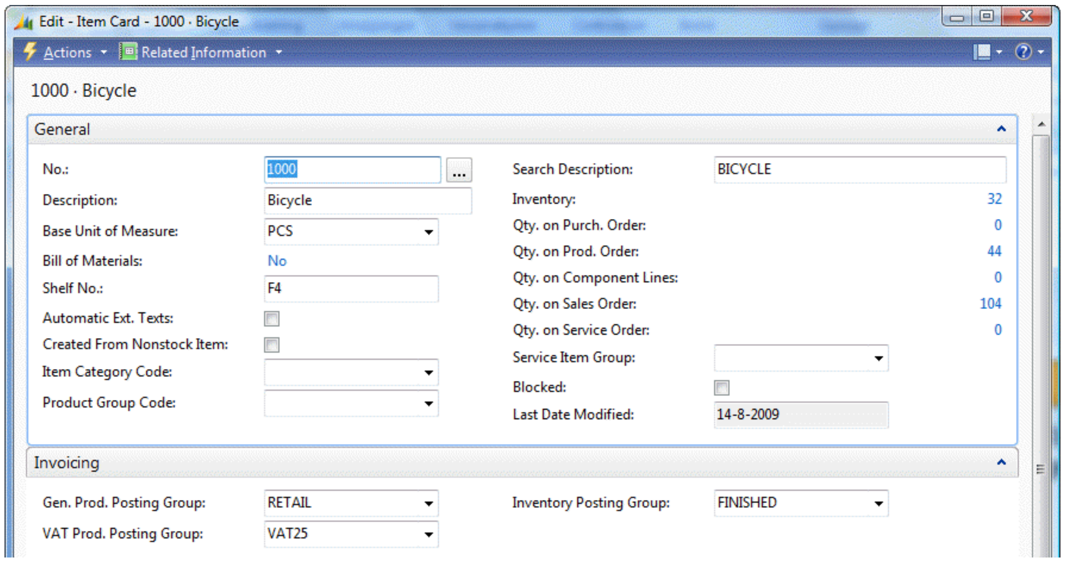
Edit - General Posting Setup - EU - Customers and vendors in EU

Actions

Gen. Bus. Po...	Gen. Prod. P...	Sales Account	Sales Credit ...	Sales Line Di...	Sales Inv. Dis...	Purch. Acco...	Purch. Credi...	Purch. Line ...	Purch. Inv. D...
EU	MANUFACT	6120	6120	6910	6910	7120	7120	7140	7140
EU	MISC	6120	6120	6910	6910	7120	7120	7140	7140
EU	NO VAT	6120	6120	6910	6910	7120	7120	7140	7140
EU	RAW MAT	6220	6220	6910	6910	7220	7220	7240	7240
EU	RETAIL	6120	6120	6910	6910	7120	7120	7140	7140
EU	SERVICES	6420	6420	6910	6910	7120	7120	7140	7140
EXPORT	MANUFACT	6130	6130	6910	6910	7130	7130	7140	7140
EXPORT	MISC	6130	6130	6910	6910	7130	7130	7140	7140
EXPORT	NO VAT	6130	6130	6910	6910	7130	7130	7140	7140
EXPORT	RAW MAT	6230	6230	6910	6910	7230	7230	7240	7240
EXPORT	RETAIL	6130	6130	6910	6910	7130	7130	7140	7140
EXPORT	SERVICES	6430	6430	6910	6910	7130	7130	7140	7140
INTERCOMP	MISC	6120	6120	6910	6910	7120	7120	7140	7140
INTERCOMP	NO VAT	6120	6120	6910	6910	7120	7120	7140	7140
INTERCOMP	RAW MAT	6120	6120	6910	6910	7120	7120	7240	7240
INTERCOMP	RETAIL	6120	6120	6910	6910	7120	7120	7140	7140
NATIONAL	MANUFACT	6110	6110	6910	6910	7110	7110	7140	7140
NATIONAL	MISC	6110	6110	6910	6910	7110	7110	7140	7140
NATIONAL	NO VAT	6110	6110	6910	6910	7110	7110	7140	7140
NATIONAL	RAW MAT	6210	6210	6910	6910	7210	7210	7240	7240
NATIONAL	RETAIL	6110	6110	6910	6910	7110	7110	7140	7140
NATIONAL	SERVICES	6410	6410	6910	6910	7110	7110	7140	7140

Here we can see that, when combined with a **Gen. Prod. Posting Group** the G/L accounts can be determined. So, where does the **Gen. Prod. Posting Group** come from?

For this, we need to go to the item card.



Edit - Item Card - 1000 · Bicycle

Actions · Related Information

1000 · Bicycle

General

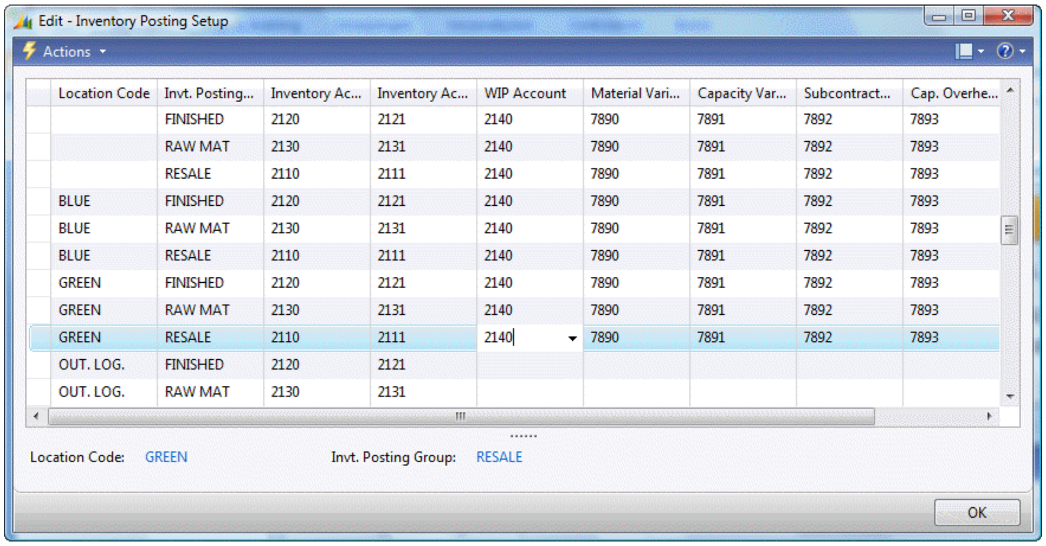
No.:	1000	Search Description:	BICYCLE
Description:	Bicycle	Inventory:	32
Base Unit of Measure:	PCS	Qty. on Purch. Order:	0
Bill of Materials:	No	Qty. on Prod. Order:	44
Shelf No.:	F4	Qty. on Component Lines:	0
Automatic Ext. Texts:	☐	Qty. on Sales Order:	104
Created From Nonstock Item:	☐	Qty. on Service Order:	0
Item Category Code:		Service Item Group:	
Product Group Code:		Blocked:	☐
		Last Date Modified:	14-8-2009

Invoicing

Gen. Prod. Posting Group:	RETAIL	Inventory Posting Group:	FINISHED
VAT Prod. Posting Group:	VAT25		

Here we can see the same tab, invoicing, with the product posting groups.

Our journey ends here as we can see the last matrix posting group, Inventory. When we open this setup we see that it is determined by the combination of **Inventory Posting Group** and **Location Code**.



Location Code	Invt. Posting...	Inventory Ac...	Inventory Ac...	WIP Account	Material Vari...	Capacity Var...	Subcontract...	Cap. Overhe...
	FINISHED	2120	2121	2140	7890	7891	7892	7893
	RAW MAT	2130	2131	2140	7890	7891	7892	7893
	RESALE	2110	2111	2140	7890	7891	7892	7893
BLUE	FINISHED	2120	2121	2140	7890	7891	7892	7893
BLUE	RAW MAT	2130	2131	2140	7890	7891	7892	7893
BLUE	RESALE	2110	2111	2140	7890	7891	7892	7893
GREEN	FINISHED	2120	2121	2140	7890	7891	7892	7893
GREEN	RAW MAT	2130	2131	2140	7890	7891	7892	7893
GREEN	RESALE	2110	2111	2140	7890	7891	7892	7893
OUT. LOG.	FINISHED	2120	2121					
OUT. LOG.	RAW MAT	2130	2131					

Location Code: GREEN Invt. Posting Group: RESALE

Dimensions

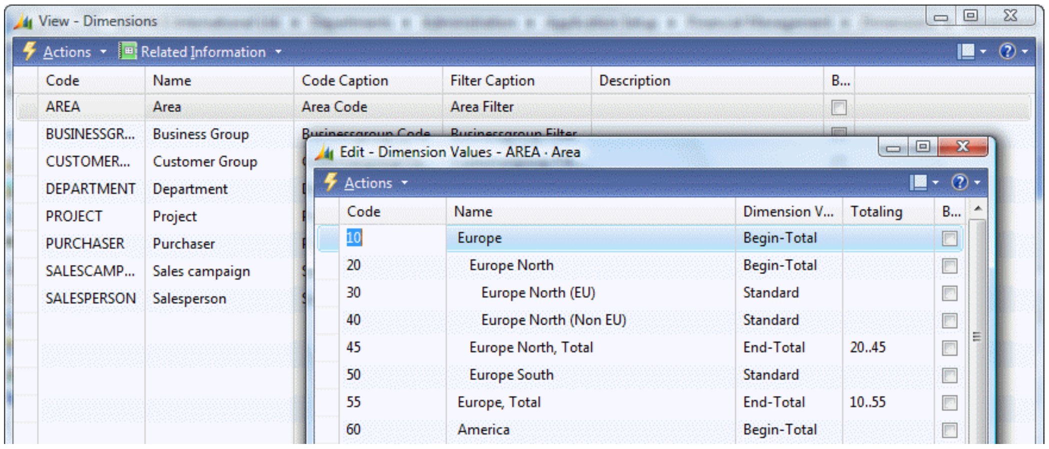
Apart from the General Ledger and sub administrations, Microsoft Dynamics NAV allows for a third level of posting. An unlimited number of dimensions can be attached to every posting and used to cross-analyze the system.

> Using more dimensions results in increased database activity during the processing of transactions and a more complex setup of the system. This should be carefully considered during the implementation.

Dimensions originated from the classic project code and department code functionality, allowing you to consolidate or differentiate costs and profits.

The dimensions are determined via a filtering mechanism. Every master data record can have dimension definitions.

Let's look at some sample dimension codes and values:



View - Dimensions

Actions · Related Information

Code	Name	Code Caption	Filter Caption	Description	B...
AREA	Area	Area Code	Area Filter		☐
BUSINESSGR...	Business Group	Businessgroup Code	Businessgroup Filter		
CUSTOMER...	Customer Group				
DEPARTMENT	Department				
PROJECT	Project				
PURCHASER	Purchaser				
SALESCAMP...	Sales campaign				
SALESPERSON	Salesperson				

Edit - Dimension Values - AREA · Area

Actions

Code	Name	Dimension V...	Totaling	B...
10	Europe	Begin-Total		☐
20	Europe North	Begin-Total		☐
30	Europe North (EU)	Standard		☐
40	Europe North (Non EU)	Standard		☐
45	Europe North, Total	End-Total	20..45	☐
50	Europe South	Standard		☐
55	Europe, Total	End-Total	10..55	☐
60	America	Begin-Total		☐

The **Dimension Code Area** has several **Dimension Values**. Here you can also have total records, just as in the General Ledger.

When more than one master data record has the same dimension code with different values, it is able to set priorities. It is also possible to block combination of dimensions to be posted.

Dimensions are a powerful tool for analyzing data and structuring the system to avoid incorrect entries. However, it requires a lot of time and special skills to determine these combinations and maintain the setup.

We'll see more of dimensions as we discuss the reporting possibilities.

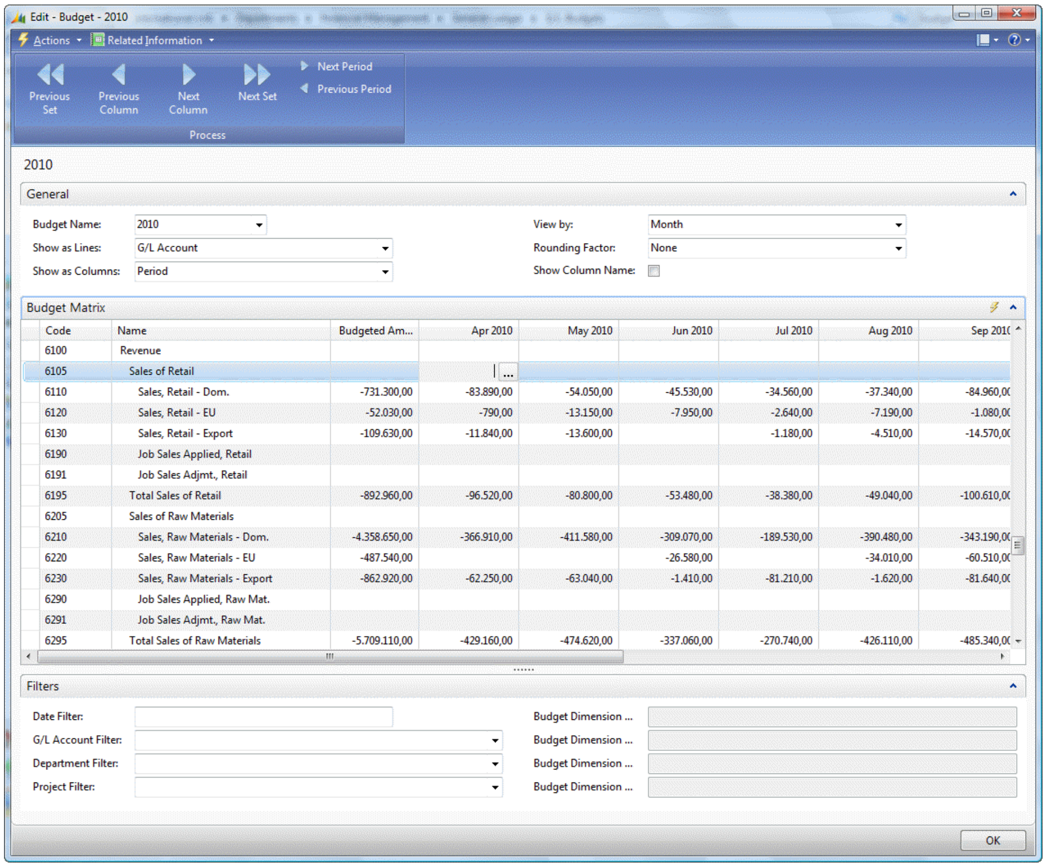
Budgeting

Microsoft Dynamics NAV allows for budgeting as well.

We can create our own budgeting codes. A budgeting code can be a year, a department, or just some budget we want to try and throw away later.

The budgeting can be done on G/L accounts but also on any dimension.

Another important thing to consider is budgeting periods. If you want to compare monthly budgets with real figures, it does not make sense to create a yearly budget. Most companies use monthly budgets. It is also most likely that we want to create budgets for Income Statement accounts, not for Balance Sheets.



2010

General

Budget Name:	2010	View by:	Month
Show as Lines:	G/L Account	Rounding Factor:	None
Show as Columns:	Period	Show Column Name:	☐

Budget Matrix

Code	Name	Budgeted Am...	Apr 2010	May 2010	Jun 2010	Jul 2010	Aug 2010	Sep 201
6100	Revenue							
6105	Sales of Retail							
6110	Sales, Retail - Dom.	-731.300,00	-83.890,00	-54.050,00	-45.530,00	-34.560,00	-37.340,00	-84.960,0
6120	Sales, Retail - EU	-52.030,00	-790,00	-13.150,00	-7.950,00	-2.640,00	-7.190,00	-1.080,0
6130	Sales, Retail - Export	-109.630,00	-11.840,00	-13.600,00		-1.180,00	-4.510,00	-14.570,0
6190	Job Sales Applied, Retail							
6191	Job Sales Adjmt., Retail							
6195	Total Sales of Retail	-892.960,00	-96.520,00	-80.800,00	-53.480,00	-38.380,00	-49.040,00	-100.610,0
6205	Sales of Raw Materials							
6210	Sales, Raw Materials - Dom.	-4.358.650,00	-366.910,00	-411.580,00	-309.070,00	-189.530,00	-390.480,00	-343.190,0
6220	Sales, Raw Materials - EU	-487.540,00			-26.580,00		-34.010,00	-60.510,0
6230	Sales, Raw Materials - Export	-862.920,00	-62.250,00	-63.040,00	-1.410,00	-81.210,00	-1.620,00	-81.640,0
6290	Job Sales Applied, Raw Mat.							
6291	Job Sales Adjmt., Raw Mat.							
6295	Total Sales of Raw Materials	-5.709.110,00	-429.160,00	-474.620,00	-337.060,00	-270.740,00	-426.110,00	-485.340,0

Filters

Date Filter:		Budget Dimension ...	
G/L Account Filter:		Budget Dimension ...	
Department Filter:		Budget Dimension ...	
Project Filter:		Budget Dimension ...	

Another powerful feature is the ability to import and export budgets to excel. Here we can easily copy and paste , for example, automatically have the same values each month.

Creating budget entries

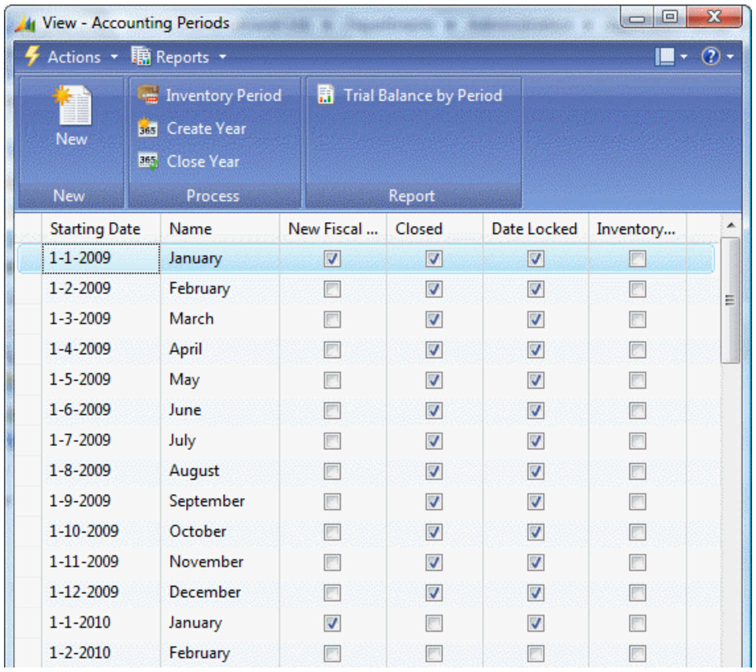
Budget entries are created by simply entering new amounts in the columns. In previous versions of Microsoft Dynamics NAV, a built-in mechanism would handle the creation of the entry based on deltas between the previous value and the newly entered value.

In Microsoft Dynamics NAV 2009, this has been changed for the Role Tailored Client to C/AL Code. The matrix page object that handles the amount is Budget Matrix (9203). This page uses the Matrix Management Codeunit (9200) to simulate the classic built-in algorithms.

Accounting periods

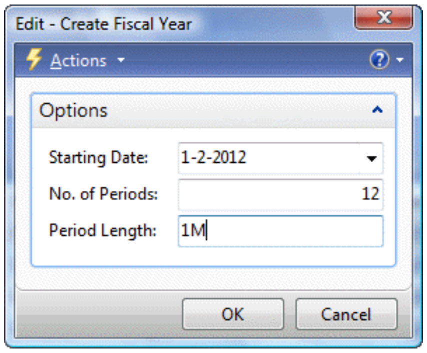
While most companies have accounting periods from January 1st to December 31st divided into months, there can be exceptions to this.

This is supported by Microsoft Dynamics NAV and set up in the **Accounting Periods**.



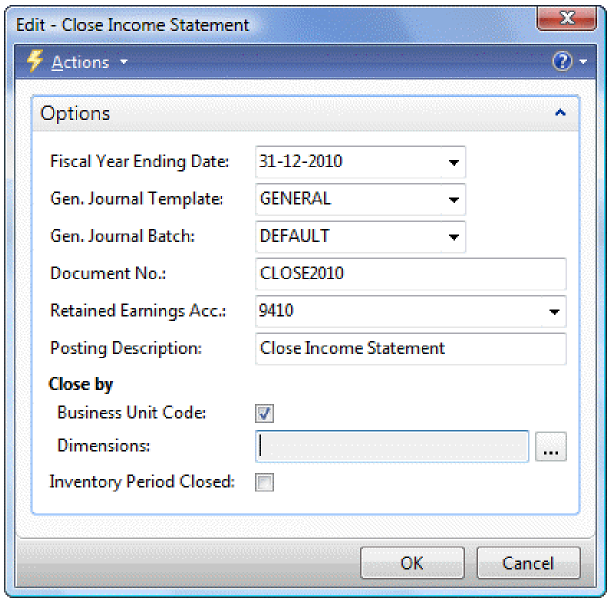
Starting Date	Name	New Fiscal ...	Closed	Date Locked	Inventory...
1-1-2009	January	✓	✓	✓	
1-2-2009	February		✓	✓	
1-3-2009	March		✓	✓	
1-4-2009	April		✓	✓	
1-5-2009	May		✓	✓	
1-6-2009	June		✓	✓	
1-7-2009	July		✓	✓	
1-8-2009	August		✓	✓	
1-9-2009	September		✓	✓	
1-10-2009	October		✓	✓	
1-11-2009	November		✓	✓	
1-12-2009	December		✓	✓	
1-1-2010	January	✓		✓	
1-2-2010	February				

We are completely free to set up our own desired posting periods as long as there is a date algorithm.



Starting Date: 1-2-2012
No. of Periods: 12
Period Length: 1M

A posting period should also be closed when appropriate. When closing a posting period, all Income Statement G/L accounts are set to zero and the profit/loss is posted to a Balance Account.

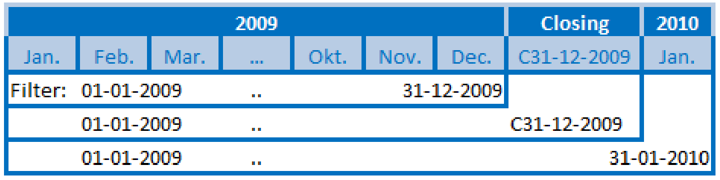


Fiscal Year Ending Date: 31-12-2010
Gen. Journal Template: GENERAL
Gen. Journal Batch: DEFAULT
Document No.: CLOSE2010
Retained Earnings Acc.: 9410
Posting Description: Close Income Statement
Close by
Business Unit Code: ✓
Dimensions:
Inventory Period Closed: ☐

When we run this batch, a General Journal is populated with the postings. It is not recommended to make changes here.

Closing dates

After closing, it is still possible to make transactions but with special posting dates called closing dates. If you put a 'C' character in front of the posting date, the system will accept this as a special transaction and allow you to post it.



When filtering on `01-01-2009..31-12-2009`, the system will not include the entries on the closing dates. Filtering on `01-01-2009..C31-12-2009` and `01-01-2009..31-01-2010` will include the entries on the closing dates.

Currencies

Besides having the possibility of the extra reporting currency, every transaction in Microsoft Dynamics NAV can have its own currency. The transaction is transformed into local currency (LCY) with the current currency exchange rates.

Handling currency is simple as long as the exchange rates do not change. After that, it can get complex.

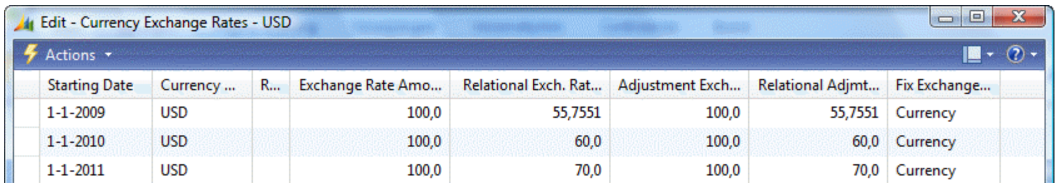
The exchange rate can change as often as you want but with a maximum of once per day.

Before you consider implementing daily changing of exchange rates, you should carefully look at the consequences.

When you change the currency exchange rate everything in the system gets adjusted, which can lead to a huge number of transactions in your system.

Changing the currency exchange rate takes two steps:

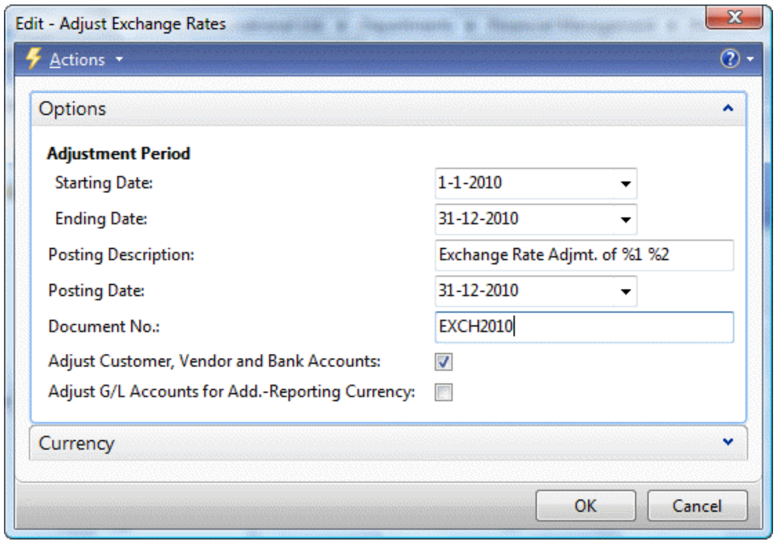
1. Enter the new values.



Starting Date	Currency ...	R...	Exchange Rate Amo...	Relational Exch. Rat...	Adjustment Exch...	Relational Adjmt...	Fix Exchange...
1-1-2009	USD		100,0	55,7551	100,0	55,7551	Currency
1-1-2010	USD		100,0	60,0	100,0	60,0	Currency
1-1-2011	USD		100,0	70,0	100,0	70,0	Currency

In our case, the new USD rate in 2010 is 60.

2. Implement the value and generate the entries.



Edit - Adjust Exchange Rates

Options

Adjustment Period

Starting Date:	1-1-2010
Ending Date:	31-12-2010
Posting Description:	Exchange Rate Adjmt. of %1 %2
Posting Date:	31-12-2010
Document No.:	EXCH2010
Adjust Customer, Vendor and Bank Accounts:	✓
Adjust G/L Accounts for Add.-Reporting Currency:	☐

Currency

Consolidation

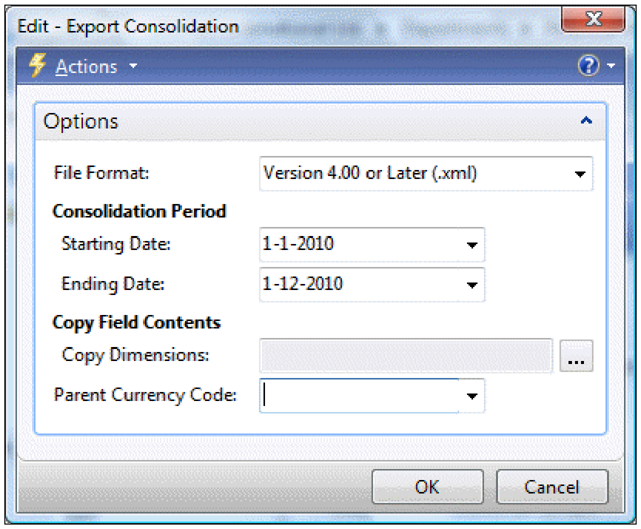
Consolidation means taking (part of) the General Ledger of two or more companies together in one consolidated company.

To handle consolidation in Microsoft Dynamics NAV, first the consolidation accounts have to be populated in the G/L Accounts.

These consolidation accounts have to be valid accounts in the consolidation company.

A consolidation company is a 'dummy' company in the database that just exists for consolidation purposes. The consolidation company has a business unit for each consolidated company.

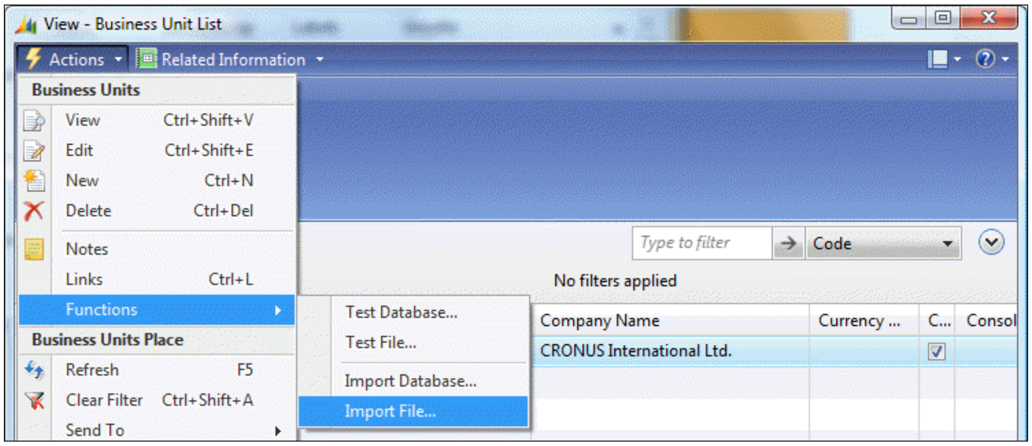
The data can be exported out of the database via XML or TXT format.


Edit - Export Consolidation

Options

File Format:	Version 4.00 or Later (.xml)
Consolidation Period	
Starting Date:	1-1-2010
Ending Date:	1-12-2010
Copy Field Contents	
Copy Dimensions:	
Parent Currency Code:	

It can be imported via the **Business Unit** list in the consolidation company.


View - Business Unit List

No filters applied

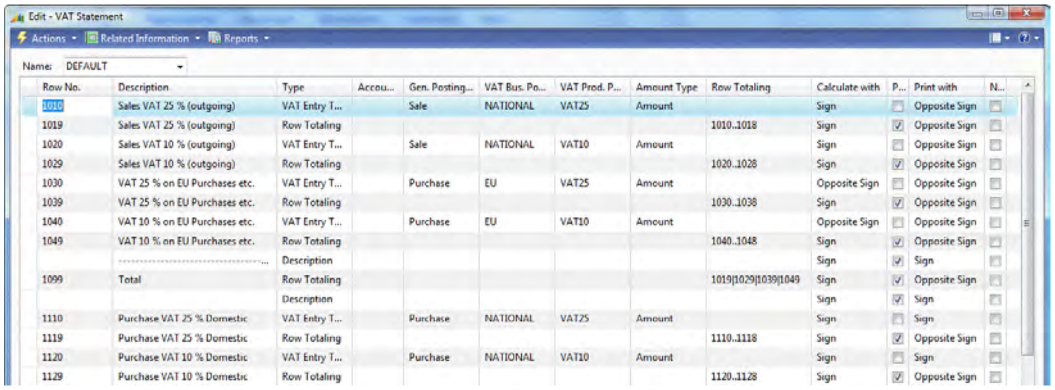
Company Name	Currency ...	C...	Consol
CRONUS International Ltd.		✓	

The other option is to import it from within the database with the **Import Database** function.

VAT statement

Most companies can issue VAT Statements to get back the VAT they paid to vendors and pay the VAT they've received from customers.

This is done in the VAT statement. This is a straightforward list where we can filter on the VAT entries.



Name: DEFAULT

Row No.	Description	Type	Accou...	Gen. Posting...	VAT Bus. Po...	VAT Prod. P...	Amount Type	Row Totaling	Calculate with	P...	Print with	N...
1010	Sales VAT 25 % (outgoing)	VAT Entry T...		Sale	NATIONAL	VAT25	Amount		Sign	☐	Opposite Sign	☐
1019	Sales VAT 25 % (outgoing)	Row Totaling						1010..1018	Sign	✓	Opposite Sign	☐
1020	Sales VAT 10 % (outgoing)	VAT Entry T...		Sale	NATIONAL	VAT10	Amount		Sign	☐	Opposite Sign	☐
1029	Sales VAT 10 % (outgoing)	Row Totaling						1020..1028	Sign	✓	Opposite Sign	☐
1030	VAT 25 % on EU Purchases etc.	VAT Entry T...		Purchase	EU	VAT25	Amount		Opposite Sign	☐	Opposite Sign	☐
1039	VAT 25 % on EU Purchases etc.	Row Totaling						1030..1038	Sign	✓	Opposite Sign	☐
1040	VAT 10 % on EU Purchases etc.	VAT Entry T...		Purchase	EU	VAT10	Amount		Opposite Sign	☐	Opposite Sign	☐
1049	VAT 10 % on EU Purchases etc.	Row Totaling						1040..1048	Sign	✓	Opposite Sign	☐
	------------------------------...	Description							Sign	✓	Sign	☐
1099	Total	Row Totaling						1019|1029|1039|1049	Sign	✓	Opposite Sign	☐
		Description							Sign	✓	Sign	☐
1110	Purchase VAT 25 % Domestic	VAT Entry T...		Purchase	NATIONAL	VAT25	Amount		Sign	☐	Sign	☐
1119	Purchase VAT 25 % Domestic	Row Totaling						1110..1118	Sign	✓	Opposite Sign	☐
1120	Purchase VAT 10 % Domestic	VAT Entry T...		Purchase	NATIONAL	VAT10	Amount		Sign	☐	Sign	☐
1129	Purchase VAT 10 % Domestic	Row Totaling						1120..1128	Sign	✓	Opposite Sign	☐

Every country has its own VAT statement and many countries have localizations in this application area.

Data analysis

Some companies do bookkeeping because it is mandatory and do very little with the generated information, but there is a lot you can do with the information the system creates.

In bigger companies, using analysis tools is often the only way to get a clear view on the company's assets.

General Ledger

The General Ledger is a reporting tool by itself. The total accounts give a lot of information and by applying limit totals (Flow filters) we can narrow this information.

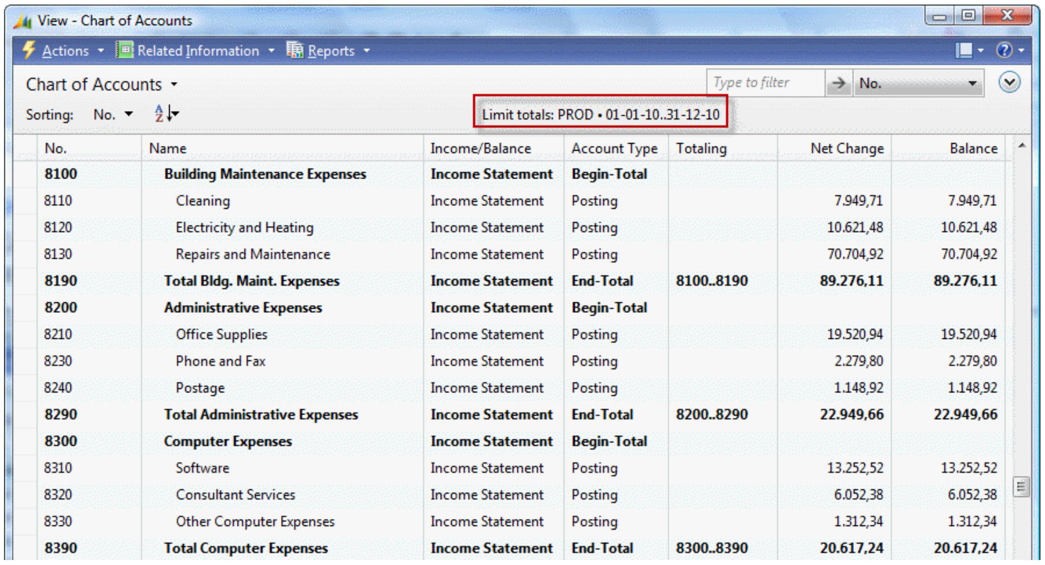


Chart of Accounts

Sorting: No.

Limit totals: PROD • 01-01-10..31-12-10

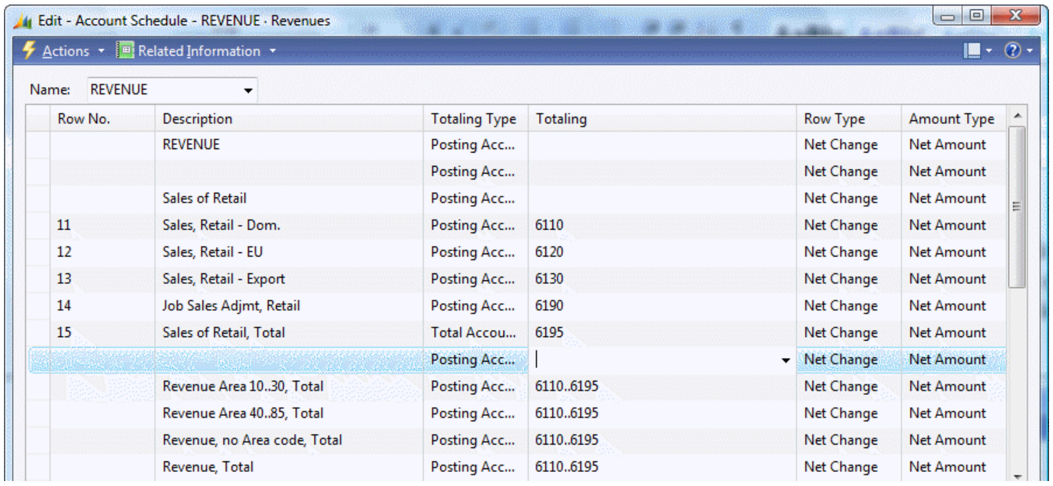
No.	Name	Income/Balance	Account Type	Totaling	Net Change	Balance
8100	**Building Maintenance Expenses**	**Income Statement**	**Begin-Total**			
8110	Cleaning	Income Statement	Posting		7.949,71	7.949,71
8120	Electricity and Heating	Income Statement	Posting		10.621,48	10.621,48
8130	Repairs and Maintenance	Income Statement	Posting		70.704,92	70.704,92
8190	**Total Bldg. Maint. Expenses**	**Income Statement**	**End-Total**	**8100..8190**	**89.276,11**	**89.276,11**
8200	**Administrative Expenses**	**Income Statement**	**Begin-Total**			
8210	Office Supplies	Income Statement	Posting		19.520,94	19.520,94
8230	Phone and Fax	Income Statement	Posting		2.279,80	2.279,80
8240	Postage	Income Statement	Posting		1.148,92	1.148,92
8290	**Total Administrative Expenses**	**Income Statement**	**End-Total**	**8200..8290**	**22.949,66**	**22.949,66**
8300	**Computer Expenses**	**Income Statement**	**Begin-Total**			
8310	Software	Income Statement	Posting		13.252,52	13.252,52
8320	Consultant Services	Income Statement	Posting		6.052,38	6.052,38
8330	Other Computer Expenses	Income Statement	Posting		1.312,34	1.312,34
8390	**Total Computer Expenses**	**Income Statement**	**End-Total**	**8300..8390**	**20.617,24**	**20.617,24**

This example filters on G/L Account No. larger than or equal to 6000 and limits total to 2010 and Department PROD.

You can save these views by clicking on the page name `<Chart of Accounts>` and then **Save View As...**. By choosing a name that makes sense, such as Income Statement 2010 Production, it is easy to find.

Account schedules

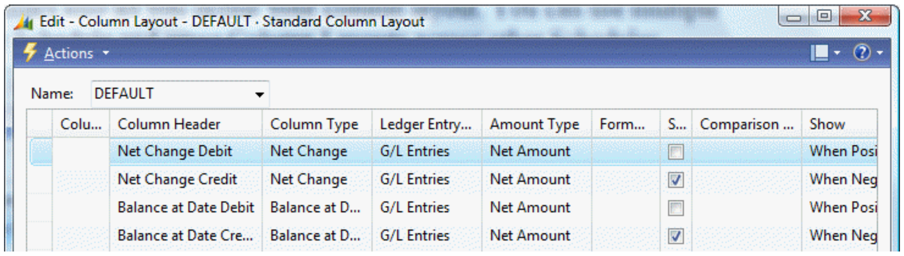
The account schedules are more advanced to use. Like the VAT statement, they allows us to filter on the G/L Entries. We can filter on individual G/L accounts or use the total filter. If the filter gets complex, we can sum individual rows and hide the source rows. We can also apply up to four dimensions to each account schedule.



Name: REVENUE

Row No.	Description	Totaling Type	Totaling	Row Type	Amount Type
	REVENUE	Posting Acc...		Net Change	Net Amount
		Posting Acc...		Net Change	Net Amount
	Sales of Retail	Posting Acc...		Net Change	Net Amount
11	Sales, Retail - Dom.	Posting Acc...	6110	Net Change	Net Amount
12	Sales, Retail - EU	Posting Acc...	6120	Net Change	Net Amount
13	Sales, Retail - Export	Posting Acc...	6130	Net Change	Net Amount
14	Job Sales Adjmt, Retail	Posting Acc...	6190	Net Change	Net Amount
15	Sales of Retail, Total	Total Accou...	6195	Net Change	Net Amount
		Posting Acc...		Net Change	Net Amount
	Revenue Area 10..30, Total	Posting Acc...	6110..6195	Net Change	Net Amount
	Revenue Area 40..85, Total	Posting Acc...	6110..6195	Net Change	Net Amount
	Revenue, no Area code, Total	Posting Acc...	6110..6195	Net Change	Net Amount
	Revenue, Total	Posting Acc...	6110..6195	Net Change	Net Amount

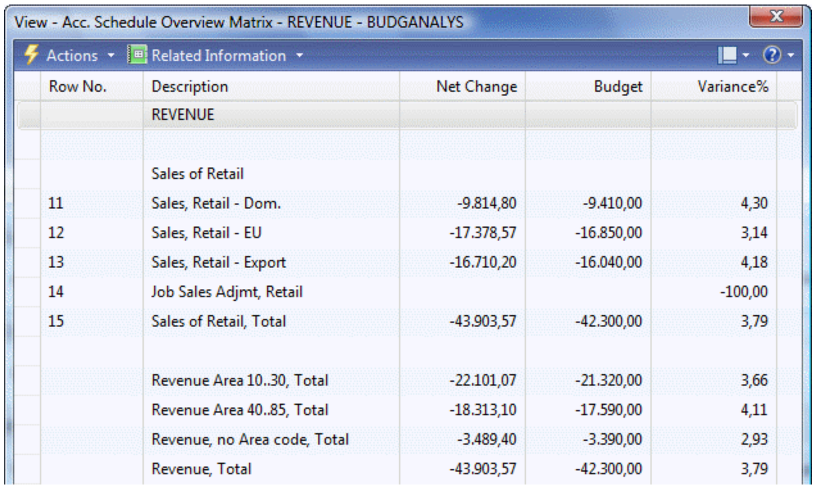
The account schedules also let you define your column layout. You can use multiple column layouts per schedule and reuse **Column Layouts** across other schedules.



Name: DEFAULT

Colu...	Column Header	Column Type	Ledger Entry...	Amount Type	Form...	S...	Comparison ...	Show
	Net Change Debit	Net Change	G/L Entries	Net Amount		☐		When Posi
	Net Change Credit	Net Change	G/L Entries	Net Amount		☑		When Neg
	Balance at Date Debit	Balance at D...	G/L Entries	Net Amount		☐		When Posi
	Balance at Date Cre...	Balance at D...	G/L Entries	Net Amount		☑		When Neg

The column layout can contain formulas and date filters. We can show either the budget or G/L entries per column.

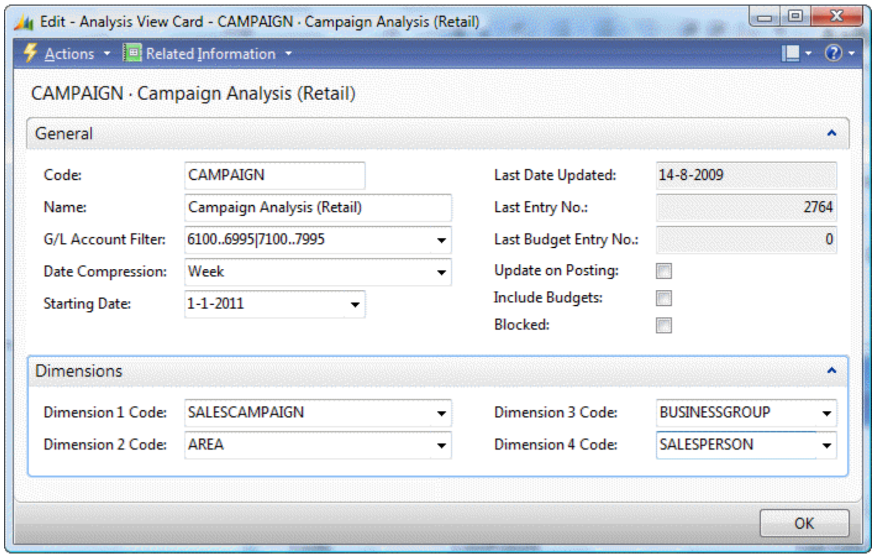

View - Acc. Schedule Overview Matrix - REVENUE - BUDGANALYS

Row No.	Description	Net Change	Budget	Variance%
	REVENUE			
	Sales of Retail			
11	Sales, Retail - Dom.	-9.814,80	-9.410,00	4,30
12	Sales, Retail - EU	-17.378,57	-16.850,00	3,14
13	Sales, Retail - Export	-16.710,20	-16.040,00	4,18
14	Job Sales Adjmt, Retail			-100,00
15	Sales of Retail, Total	-43.903,57	-42.300,00	3,79
	Revenue Area 10..30, Total	-22.101,07	-21.320,00	3,66
	Revenue Area 40..85, Total	-18.313,10	-17.590,00	4,11
	Revenue, no Area code, Total	-3.489,40	-3.390,00	2,93
	Revenue, Total	-43.903,57	-42.300,00	3,79

Analysis by dimensions

As discussed earlier in this chapter, Microsoft Dynamics NAV allows an unlimited number of dimensions to be posted in the General Ledger. To analyze this information, we need to tell the system what to compare. This is done in analysis by dimensions.

Each analysis view gets a unique code. An analysis view can be generated for an ad-hoc requirement and thrown away afterwards or be in the system permanently for periodical reporting. Analysis views generate redundant information that can always be discarded and regenerated.


Edit - Analysis View Card - CAMPAIGN · Campaign Analysis (Retail)

CAMPAIGN · Campaign Analysis (Retail)

General

Code:	CAMPAIGN	Last Date Updated:	14-8-2009
Name:	Campaign Analysis (Retail)	Last Entry No.:	2764
G/L Account Filter:	6100..6995|7100..7995	Last Budget Entry No.:	0
Date Compression:	Week	Update on Posting:	☐
Starting Date:	1-1-2011	Include Budgets:	☐
		Blocked:	☐

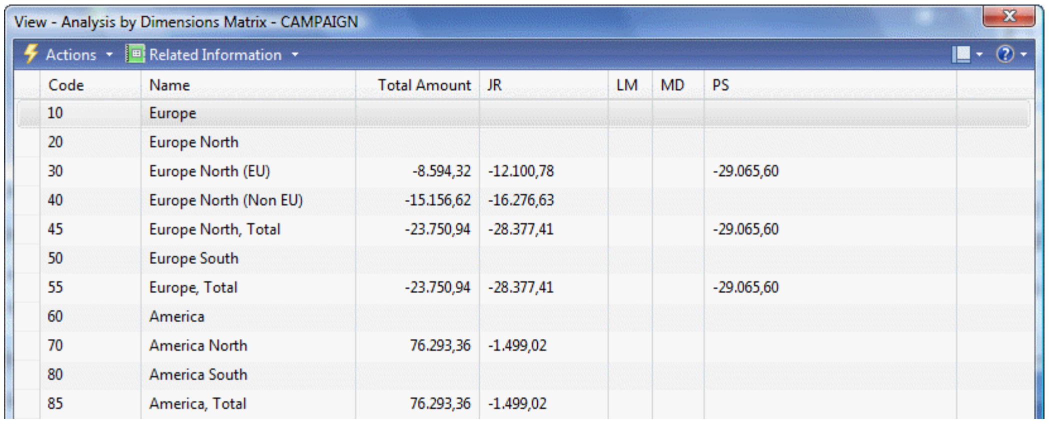
Dimensions

Dimension 1 Code:	SALESCAMPAIGN	Dimension 3 Code:	BUSINESSGROUP
Dimension 2 Code:	AREA	Dimension 4 Code:	SALESPERSON

It is recommended to use a copy of the database on a separate system to use with analysis views and to update them during the night.

When updated, the analysis view contains all data within the filters in the analysis view entries. When not properly maintained, this can become a massive table of data.

The result of an analysis view can be viewed in a matrix where all values can be used as rows, columns, and filters.



Code	Name	Total Amount	JR	LM	MD	PS
10	Europe					
20	Europe North					
30	Europe North (EU)	-8.594,32	-12.100,78			-29.065,60
40	Europe North (Non EU)	-15.156,62	-16.276,63			
45	Europe North, Total	-23.750,94	-28.377,41			-29.065,60
50	Europe South					
55	Europe, Total	-23.750,94	-28.377,41			-29.065,60
60	America					
70	America North	76.293,36	-1.499,02			
80	America South					
85	America, Total	76.293,36	-1.499,02			

In this example, we view the results of a sales campaign per area and sales person.

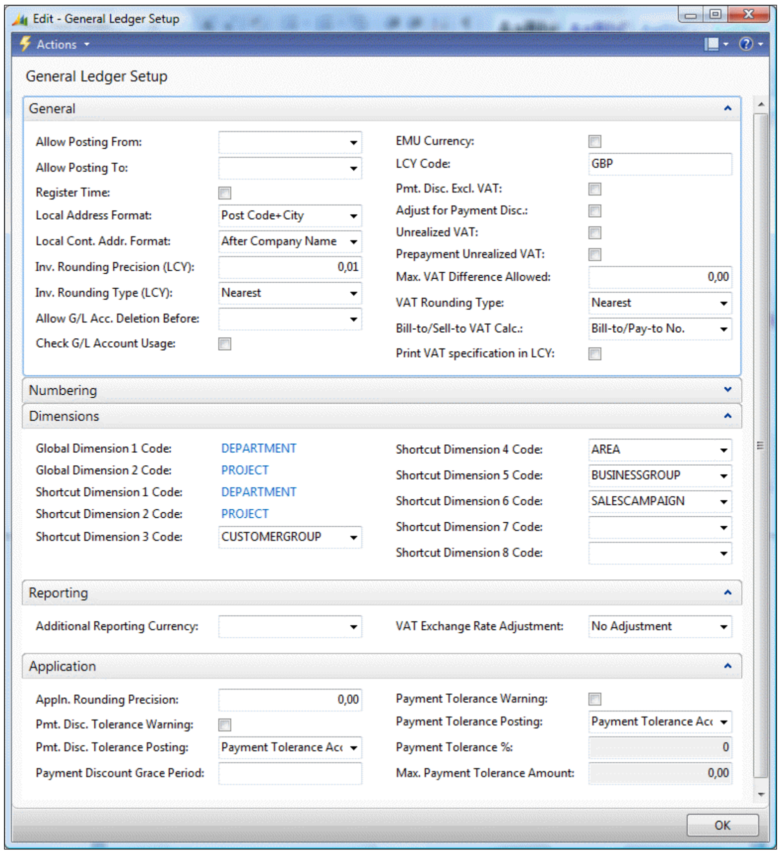
The setup

Financial Management has a single General Ledger Setup table which is important since many of these setup fields will determine how the core of Microsoft Dynamics NAV behaves.

We will discuss the setup options to find out what they do and to explore the possibilities of creating a flexible setup for an application:

- **Allow Posting From** and **Allow Posting To** limit the freedom of people to choose posting dates whilst posting to the General Ledger. It is highly recommended to enable this feature to avoid posting dates like 01012090 instead of 01012009.
- **Register Time** allows you to create an entry in the Time register each time a user logs in and out.

- **Local Address Format** and **Local Cont. Addr. Format** refer to how the address should be printed for the local country. In Microsoft Dynamics NAV, it is best practice to leave the **Country Code** and **Currency Code** blank for local values.
- **Inv. Rounding Precision (LCY)** and **Inv. Rounding Type (LCY)** define how the rounding on your invoices is calculated. **Nearest** is best practice and allows your customers to easily register your invoice in their system.
- **Allow G/L Acc Deletion Before** allows you to clean up closed fiscal years. It is hardly ever used and you should consult your partner before using this feature.
- **Check G/L Account Usage** checks if the G/L account is used in setup tables before it is deleted.
- If your company uses an **EMU Currency** and the system should know this when applying with other EMU currencies. EMU currencies are currencies that have a fixed conversion rate to the Euro in the European Union. The **LCY Code** field is used when printing reports to indicate the companies' local currency.



General Ledger Setup

General

Allow Posting From:		EMU Currency:	☐
Allow Posting To:		LCY Code:	GBP
Register Time:	☐	Pmt. Disc. Excl. VAT:	☐
Local Address Format:	Post Code+City	Adjust for Payment Disc.:	☐
Local Cont. Addr. Format:	After Company Name	Unrealized VAT:	☐
Inv. Rounding Precision (LCY):	0,01	Prepayment Unrealized VAT:	☐
Inv. Rounding Type (LCY):	Nearest	Max. VAT Difference Allowed:	0,00
Allow G/L Acc. Deletion Before:		VAT Rounding Type:	Nearest
Check G/L Account Usage:	☐	Bill-to/Sell-to VAT Calc.:	Bill-to/Pay-to No.
		Print VAT specification in LCY:	☐

Numbering

Dimensions

Global Dimension 1 Code:	DEPARTMENT	Shortcut Dimension 4 Code:	AREA
Global Dimension 2 Code:	PROJECT	Shortcut Dimension 5 Code:	BUSINESSGROUP
Shortcut Dimension 1 Code:	DEPARTMENT	Shortcut Dimension 6 Code:	SALESCAMPAIGN
Shortcut Dimension 2 Code:	PROJECT	Shortcut Dimension 7 Code:	
Shortcut Dimension 3 Code:	CUSTOMERGROUP	Shortcut Dimension 8 Code:	

Reporting

Additional Reporting Currency:		VAT Exchange Rate Adjustment:	No Adjustment

Application

Appln. Rounding Precision:	0,00	Payment Tolerance Warning:	☐
Pmt. Disc. Tolerance Warning:	☐	Payment Tolerance Posting:	Payment Tolerance Acc
Pmt. Disc. Tolerance Posting:	Payment Tolerance Acc	Payment Tolerance %:	0
Payment Discount Grace Period:		Max. Payment Tolerance Amount:	0,00

- **Pmt. Disc. Excl. VAT** indicates whether or not VAT is calculated when you apply payment discounts. When you check this field you need to think about the **Adjust for Payment Disc.** field as this will recalculate the VAT.
- **Unrealized VAT** should only be checked if your company has to deal with this issue. Otherwise, it will lead to unnecessary postings. **Unrealized VAT** is VAT that is only valid when the customer pays the invoice rather than when the invoice is issued.
- **Prepayment Unrealized VAT** should only be checked if your company handles **Unrealized VAT** and if you want to implement this for the prepayment features.
- **Max. VAT Difference Allowed** field determines the maximum amount of VAT differences. Most of the time the VAT difference will be not more than 0, 01.

You can post VAT differences by selecting **FULL VAT** in the **VAT Calculation Type** for the VAT business posting group.

- **VAT Rounding Type** determines how the VAT remainder is calculated. It is recommended using nearest.
- With the **Bill-to/Sell-to VAT Calc.** you can change what is the source for the VAT business posting group, whether it is the Bill-to Customer or
  Sell-to Customer and Pay-to Vendor or Buy-from Vendor.
- Check the **Print VAT specification in LCY** if you want the VAT on your invoices to always be in your local currency.
- **Bank Account Nos.** is almost always a number series that is manually determined. Most companies have up to 10 bank accounts.
- The **Global Dimensions** determine which dimensions are posted directly to the G/L entries and sub administrations. You can most often use these when limiting totals and should be considered carefully.
- The **Shortcut Dimensions** are easier to access when you enter journals and documents. They can easily be switched later.
- **Additional Reporting Currency** is a useful feature for international companies. Remember that it requires extra effort if the exchange rates change. You can change this later but if you do, a batch job will start, and the job might take a long time if you have a large database.

- **VAT Exchange Rate Adjustment** makes it possible to recalculate VAT if the reporting currency exchange rates change. Think about this thoroughly before you activate it. It is most likely to generate information that is difficult to analyze and use.
- **Appln Rounding Precision** can be used to allow rounding differences when applying different currencies.
- When **Pmt. Disc. Tolerance Warning** is checked, a warning will appear whenever a difference is posted.
- **Pmt. Disc. Tolerance Posting** determines if the payment tolerance amount is posted to a special account or to the normal discount account.
- **Payment Discount Grace Period** can be used if you want to be tolerant when people are one or two days late with their payment and still deduct the discount amount.
- The **Payment Tolerance Warning** option will show a warning whenever there is a tolerance amount posted to the General Ledger
- **Payment Tolerance Posting** determines if a special G/L account is used to post this amount.
- **Payment Tolerance %** determines the tolerance percentage. To change this, a batch function is used that updates open entries.
- **Max. Payment Tolerance Amount** sets a maximum to the amount so an invoice that is issued for 100.000 cannot have a tolerance amount of more than 5.000 if the percentage is set to 5%.

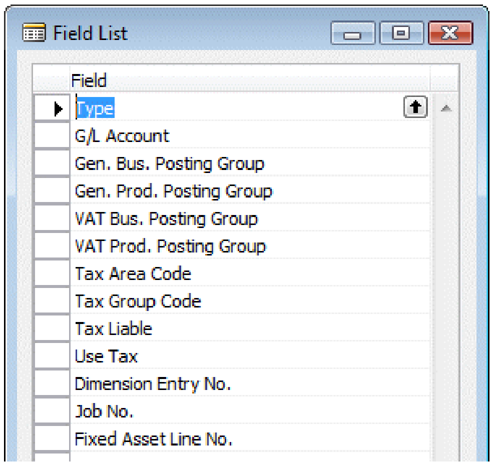
Customizing financial management

As financial management is regulated by the government and the standard functionality is already very complete, this application area is unlikely to have many changes, even though we have some examples of where the functionality is changed.

[The examples in this chapter are included in the objects we used in Chapter 2.]

Sales line description to G/L entries

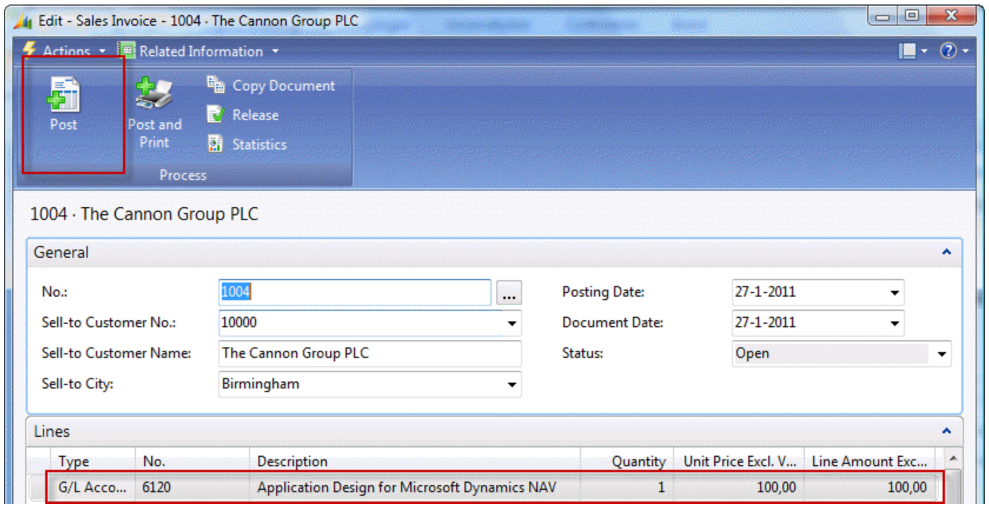
When we post a Sales Invoice, the system will generate the G/L entries based on the sales lines. To avoid creating too many entries they are compacted. This is done using a buffer table, the Invoice Post. Buffer.


Field List

Field
Type
G/L Account
Gen. Bus. Posting Group
Gen. Prod. Posting Group
VAT Bus. Posting Group
VAT Prod. Posting Group
Tax Area Code
Tax Group Code
Tax Liable
Use Tax
Dimension Entry No.
Job No.
Fixed Asset Line No.

Only for the combination of the above listed fields, a G/L entry record is created. As we can see, the description is not one of these. This results in G/L entries with the posting description of the Sales Header which is often confusing for accountants when looking at the G/L entries.

As an example, we will generate a **Sales Invoice** with one **G/L Account** line selling one of these books.



1004 · The Cannon Group PLC

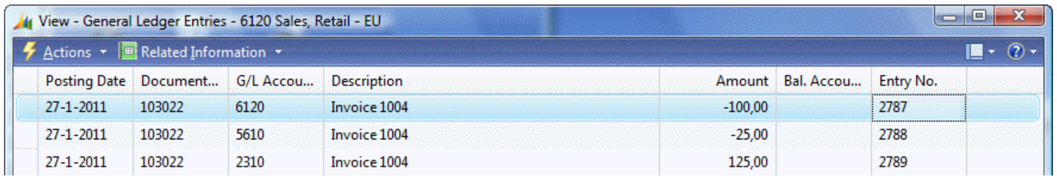
General

No.:	1004	Posting Date:	27-1-2011
Sell-to Customer No.:	10000	Document Date:	27-1-2011
Sell-to Customer Name:	The Cannon Group PLC	Status:	Open
Sell-to City:	Birmingham		

Lines

Type	No.	Description	Quantity	Unit Price Excl. V...	Line Amount Exc...
G/L Acco...	6120	Application Design for Microsoft Dynamics NAV	1	100,00	100,00

When we post this Invoice, we will get these G/L Entries. Note that the description is gone.



View - General Ledger Entries - 6120 Sales, Retail - EU

Posting Date	Document...	G/L Accou...	Description	Amount	Bal. Accou...	Entry No.
27-1-2011	103022	6120	Invoice 1004	-100,00		2787
27-1-2011	103022	5610	Invoice 1004	-25,00		2788
27-1-2011	103022	2310	Invoice 1004	125,00		2789

To change this behavior we have to change the **Invoice Post. Buffer** table. The description field needs to be part of the unique combination since the grouping is done using a `FIND` command in function `UpdInvPostingBuffer` in the `Sales-Post Codeunit (80)`.

```
UpdInvPostingBuffer()
...
InvPostingBuffer[2] := InvPostingBuffer[1];
IF InvPostingBuffer[2].FIND THEN BEGIN
  InvPostingBuffer[2].Amount :=
    InvPostingBuffer[2].Amount + InvPostingBuffer[1].Amount;
  ...
  InvPostingBuffer[2].MODIFY;
END ELSE
  InvPostingBuffer[1].INSERT;
```

This requires two steps:

1. We need to add the description field to the table.

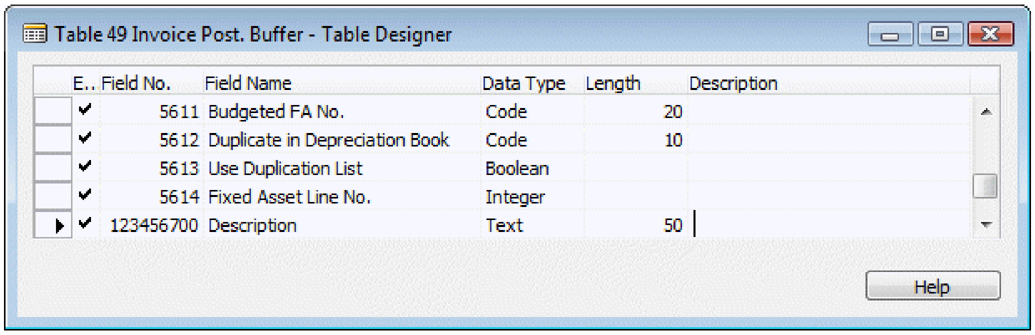
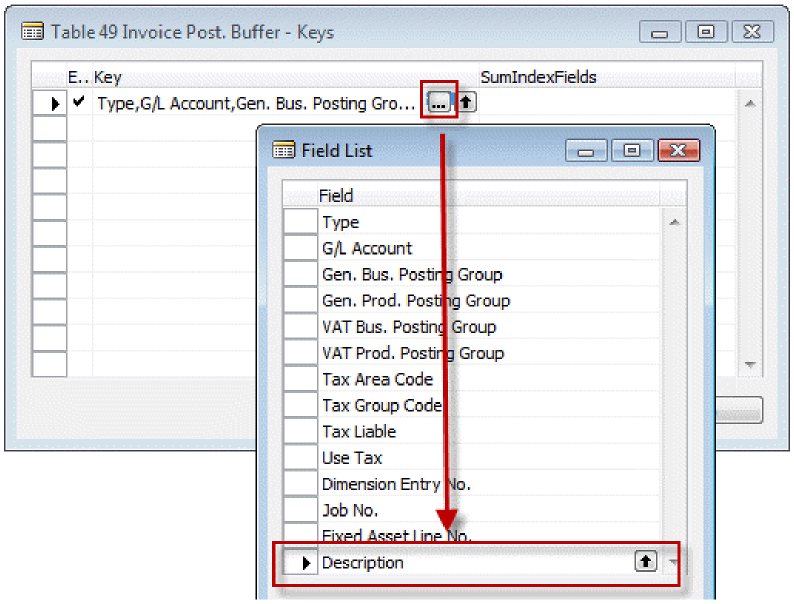


Table 49 Invoice Post. Buffer - Table Designer

E..	Field No.	Field Name	Data Type	Length	Description
✓	5611	Budgeted FA No.	Code	20	
✓	5612	Duplicate in Depreciation Book	Code	10	
✓	5613	Use Duplication List	Boolean		
✓	5614	Fixed Asset Line No.	Integer		
✓	123456700	Description	Text	50	

2. We need to add this new field to the key.



A key in Microsoft Dynamics NAV can only contain 252 bytes to be careful not to add too many fields to this table.

When this is done, a change is required in populating the buffer table. This is done in the `PrepareSales` function in the table `Invoice Post. Buffer (49)` itself.

The C/AL code has moved in Microsoft Dynamics NAV 2009. In previous versions, the code was located in the `Sales-Post Codeunit (80)` and `Purchase-Post Codeunit (90)`.

```
PrepareSales()
CLEAR(Rec);
Type := SalesLine.Type;
"System-Created Entry" := TRUE;
...
"Job No." := SalesLine."Job No.";
"VAT %" := SalesLine."VAT %";
"VAT Difference" := SalesLine."VAT Difference";
//* Description >>>
Description := SalesLine.Description;
//* Description <<<
```

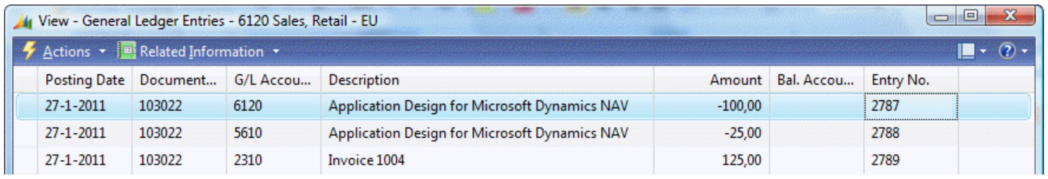
```
IF Type = Type::"Fixed Asset" THEN BEGIN
  ...
END;
```

The last change we are going to make is in the posting routine of the sales documents. This is the `Sales-Post Codeunit (80)` we discussed in *Chapter 2, An Example Application*.

```
IF Invoice THEN BEGIN
  // Post sales and VAT to G/L entries from posting buffer
  LineCount := 0;
  IF InvPostingBuffer[1].FIND('+') THEN
    REPEAT
      LineCount := LineCount + 1;
      Window.UPDATE(3,LineCount);

      GenJnlLine.INIT;
      GenJnlLine."Posting Date" := "Posting Date";
      GenJnlLine."Document Date" := "Document Date";
//* Posting Description now from buffer table >>>
//      GenJnlLine.Description := "Posting Description";
      GenJnlLine.Description :=
        InvPostingBuffer[1].Description;
//* Posting Description <<<
      GenJnlLine."Reason Code" := "Reason Code";
```

Instead of the **Posting Description** of the Sales Header, we will now use the new field in the buffer table.

When we post the same invoice again, this is the changed result:



Posting Date	Document...	G/L Accou...	Description	Amount	Bal. Accou...	Entry No.
27-1-2011	103022	6120	Application Design for Microsoft Dynamics NAV	-100,00		2787
27-1-2011	103022	5610	Application Design for Microsoft Dynamics NAV	-25,00		2788
27-1-2011	103022	2310	Invoice 1004	125,00		2789

This makes it a lot easier to read the General Ledger.

Making this change might cause our system to create significantly more G/L entries if we have large invoices with different descriptions. Creating extra G/L entries takes more time during a posting routine resulting in longer running posting transactions and a larger database.

Extra fields in the G/L entries

Although the G/L entry table has a lot of information, some companies want to add extra fields to it and populate these in the posting process.

For this example, we will use the database with the squash court application from *Chapter 2*. For this business, it might be useful to have the **Squash Court No.** as a field in the G/L entries to analyze.

The first step is to add the field to the G/L entry table and make sure we have a table relation with the source table.

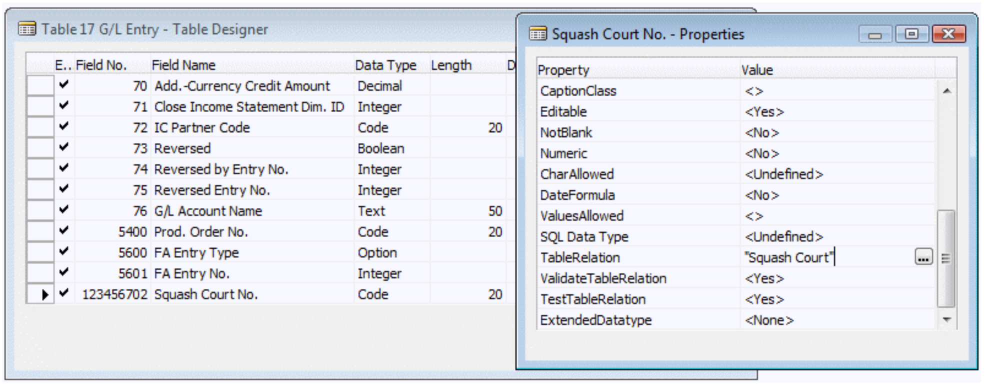


Table 17 G/L Entry - Table Designer

E..	Field No.	Field Name	Data Type	Length
✓	70	Add.-Currency Credit Amount	Decimal	
✓	71	Close Income Statement Dim. ID	Integer	
✓	72	IC Partner Code	Code	20
✓	73	Reversed	Boolean	
✓	74	Reversed by Entry No.	Integer	
✓	75	Reversed Entry No.	Integer	
✓	76	G/L Account Name	Text	50
✓	5400	Prod. Order No.	Code	20
✓	5600	FA Entry Type	Option	
✓	5601	FA Entry No.	Integer	
✓	123456702	Squash Court No.	Code	20

Squash Court No. - Properties

Property	Value
CaptionClass	<>
Editable	<Yes>
NotBlank	<No>
Numeric	<No>
CharAllowed	<Undefined>
DateFormula	<No>
ValuesAllowed	<>
SQL Data Type	<Undefined>
TableRelation	"Squash Court"
ValidateTableRelation	<Yes>
TestTableRelation	<Yes>
ExtendedDatatype	<None>

We have learned that the G/L entries are generated from the General Journal so we need to add this field there as well. This can be done with copy and paste.

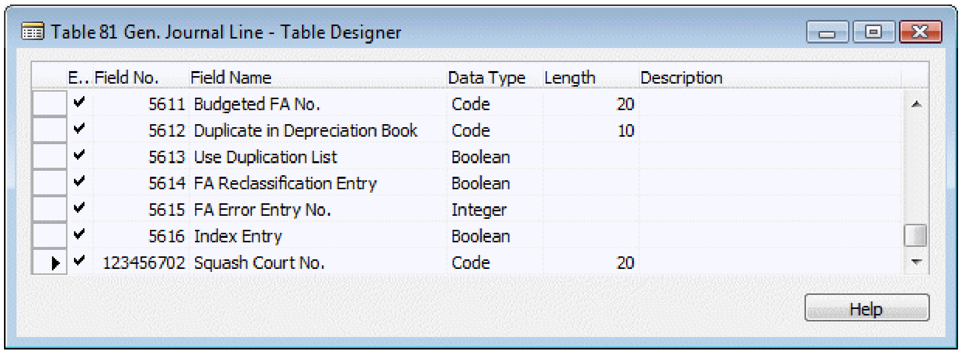


Table 81 Gen. Journal Line - Table Designer

E..	Field No.	Field Name	Data Type	Length	Description
✓	5611	Budgeted FA No.	Code	20	
✓	5612	Duplicate in Depreciation Book	Code	10	
✓	5613	Use Duplication List	Boolean		
✓	5614	FA Reclassification Entry	Boolean		
✓	5615	FA Error Entry No.	Integer		
✓	5616	Index Entry	Boolean		
✓	123456702	Squash Court No.	Code	20	

The last step is to make sure we move the information from the journal to the ledger entry table. Like in our sample squash application, this is done in the `Gen. Jnl.-Post Line Codeunit (12)` only this Codeunit has much more code.

We need to find the place where the G/L entries are created and add our field there. This is done in the function `InitGLEntry`.

```
InitGLEntry()
...
GLEntry.INIT;
GLEntry."Posting Date" := GenJnlLine."Posting Date";
GLEntry."Document Date" := GenJnlLine."Document Date";
GLEntry."Document Type" := GenJnlLine."Document Type";
GLEntry."Document No." := GenJnlLine."Document No.";
...
GLEntry."Source Code" := GenJnlLine."Source Code";
//* Squash App. >>>
GLEntry."Squash Court No." := GenJnlLine."Squash Court No.";
//* Squash App. <<<
IF GenJnlLine."Account Type" = ...
```

This is all that is required in Microsoft Dynamics NAV to add a field to the financial posting process. Of course, it does not make sense to do this unless we use it, so a logical next step could be to add this new field to the Invoice Post. Buffer table from our previous example.

This shows how easy it is to combine solutions in Microsoft Dynamics NAV.

Integrating with financial management

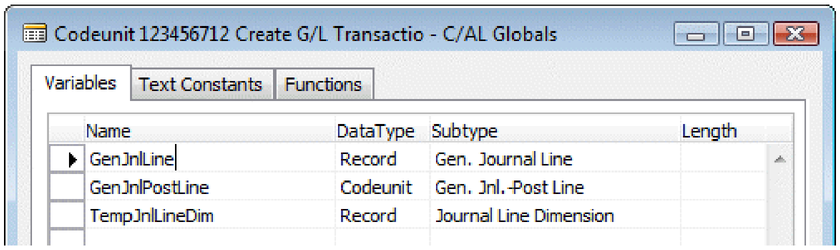
Although it is not likely to make big changes in financial management, it might be necessary to create G/L entries in a new posting routine.

In the previous chapter, we already pointed out briefly that during posting transactions in Microsoft Dynamics NAV, the actual records are never really inserted in the database. They are used as temporary containers to hold the data during posting. Doing an actual `INSERT` would require defining a journal template name, journal batch name, and line no. and could cause locking in the database.

Let's create a new Codeunit that would create a G/L transaction.

Creating a G/L transaction

After creating the Codeunit, we need to set up the three variables that are the minimum requirement to post something to the General Ledger.


Codeunit 123456712 Create G/L Transactio - C/AL Globals

Name	DataType	Subtype	Length
GenJnlLine	Record	Gen. Journal Line	
GenJnlPostLine	Codeunit	Gen. Jnl.-Post Line	
TempJnlLineDim	Record	Journal Line Dimension	

- `GenJnlLine`: This is a reference to the `General Journal Line table (81)`.
- `GenJnlPostLine`: The `Gen. Jnl.-Post Line Codeunit (12)` creates the G/L entries, the register, and the other financial entries.
- `TempJnlLineDim`: The `Journal Line Dimension table (356)` is used as a buffer table to transport dimensions to the `Gen. Jnl.-Post Line Codeunit`. This variable is always set up as a temporary table.

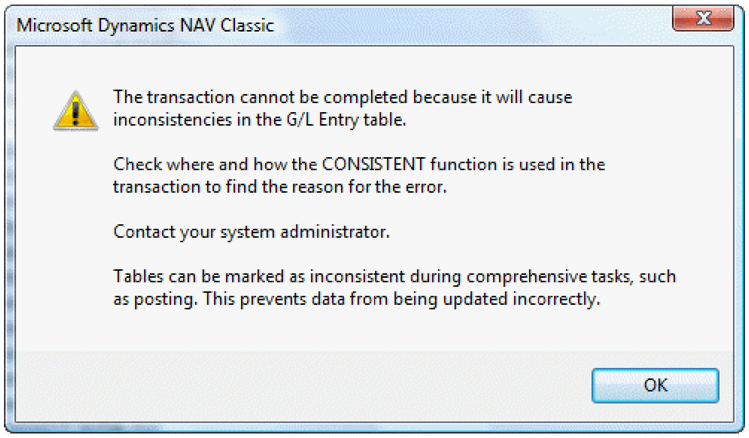
The C/AL code

Creating a new G/L entry requires a small amount of mandatory fields. All the other fields in the General Journal line are either optional for basic entries or mandatory in combination with more advanced postings as we will find out later.

We will start by writing this code to the `OnRun` trigger.

```
OnRun()
GenJnlLine.INIT;
GenJnlLine."Posting Date" := WORKDATE;
GenJnlLine.Description := 'Test Entry';
GenJnlLine."Document No." := 'PACKT';
GenJnlLine."Account No." := '6120';
GenJnlLine.Amount := 100;
GenJnlPostLine.RunWithCheck(GenJnlLine, TempJnlLineDim);
```

If we execute this C/AL code, we will receive this error message which indicates that our transaction will result in an unbalanced chart of accounts.



Microsoft Dynamics NAV Classic

The transaction cannot be completed because it will cause inconsistencies in the G/L Entry table.

Check where and how the CONSISTENT function is used in the transaction to find the reason for the error.

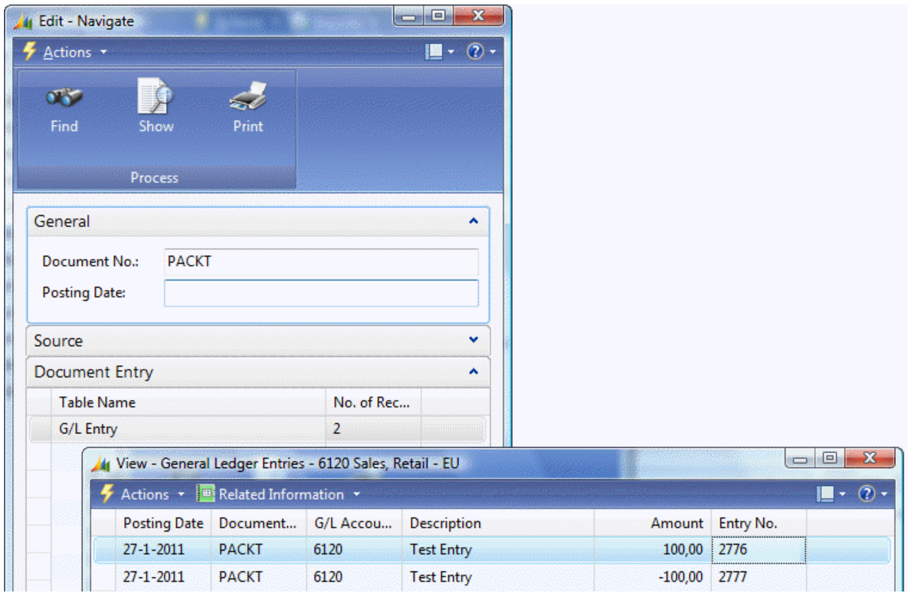
Contact your system administrator.

Tables can be marked as inconsistent during comprehensive tasks, such as posting. This prevents data from being updated incorrectly.

We can fix this by creating a balance transaction for `100` in the same `OnRun` trigger.

```
GenJnlLine.INIT;
GenJnlLine."Posting Date" := WORKDATE;
GenJnlLine.Description := 'Test Entry';
GenJnlLine."Document No." := 'PACKT';
GenJnlLine."Account No." := '6120';
GenJnlLine.Amount := -100;
GenJnlPostLine.RunWithCheck(GenJnlLine, TempJnlLineDim);
```

After executing the Codeunit we can **Navigate** on our **Document No.** to see the G/L entries we created.



Document No.:	PACKT
Posting Date:	

Table Name	No. of Rec...
G/L Entry	2

View - General Ledger Entries - 6120 Sales, Retail - EU

Posting Date	Document...	G/L Accou...	Description	Amount	Entry No.
27-1-2011	PACKT	6120	Test Entry	100,00	2776
27-1-2011	PACKT	6120	Test Entry	-100,00	2777

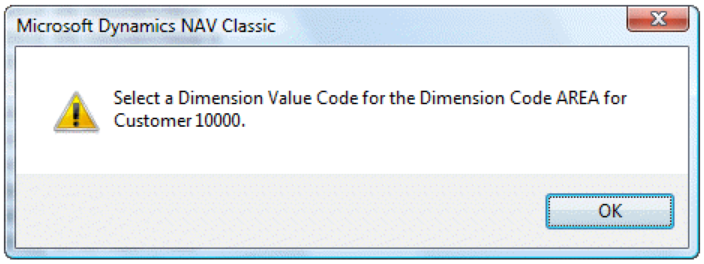
This was a very simple example of how to integrate with financial management. Let's create a more advanced example.

Advanced entries

We will create a new customer ledger entry with dimensions. To do this, we should change one of the C/AL parts we created to this code.

```
GenJnlLine.INIT;
GenJnlLine."Posting Date" := WORKDATE;
GenJnlLine.Description := 'Test Entry';
GenJnlLine."Document No." := 'PACKT2';
GenJnlLine."Account Type" := GenJnlLine."Account Type"::Customer;
GenJnlLine."Account No." := '10000';
GenJnlLine.Amount := 100;
GenJnlPostLine.RunWithCheck(GenJnlLine, TempJnlLineDim);
```

But, when we execute this C/AL code, we receive this error message:



Select a Dimension Value Code for the Dimension Code AREA for Customer 10000.

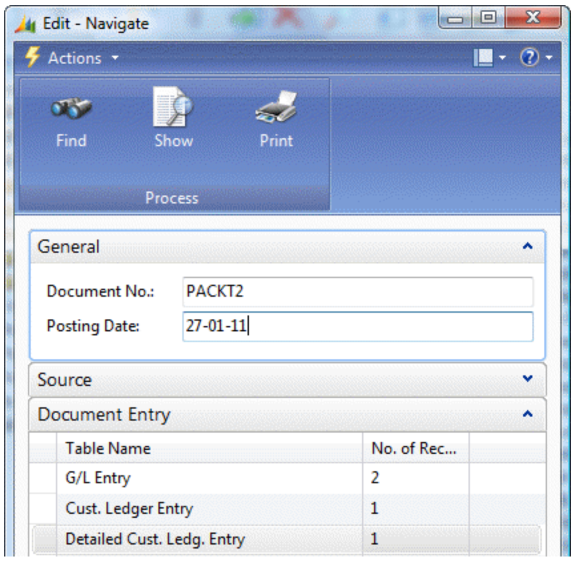
This means we need to implement dimensions.

Let's add the following C/AL code to the routine:

```
...
GenJnlLine.Amount := 100;
TempJnlLineDim."Dimension Code" := 'AREA';
TempJnlLineDim."Dimension Value Code" := '30';
TempJnlLineDim.INSERT;
TempJnlLineDim."Dimension Code" := 'CUSTOMERGROUP';
TempJnlLineDim."Dimension Value Code" := 'MEDIUM';
TempJnlLineDim.INSERT;
GenJnlPostLine.RunWithCheck(GenJnlLine, TempJnlLineDim);
```

This will insert the required dimensions to the buffer table.

When we now navigate to **PACKT2** we see that the system has created a **Customer Ledger Entry** and a **Detailed Cust. Ledg. Entry**.



Document No.: PACKT2
Posting Date: 27-01-11

Table Name	No. of Rec...
G/L Entry	2
Cust. Ledger Entry	1
Detailed Cust. Ledg. Entry	1

Look, learn, and love

In Microsoft Dynamics NAV, there are many examples of how to integrate with financial management. This is a list of interesting Codeunits that create General Journal lines:

- `Sales-Post (80)`
- `Purch.-Post (90)`
- `Job Calculate WIP (1000)`
- `CheckManagement (367)`
- `Sales-Post Prepayments (442)`
- `Inventory Posting To G/L (5802)`
- `Serv-Posting Journals Mgt. (5987)`

Go ahead and have a look inside these Codeunits to learn how Microsoft does the integration.

Summary

In this chapter, we have looked at the financial heart of Microsoft Dynamics NAV. Understanding the flow of the entries is as important as the way the posting groups are set up. It is important to regularly check the sub administrations balance with the General Ledger.

The reporting possibilities offer great insight if the system is set up correctly. Be careful with changing the setup options on a running system.

In the next chapter, we will look at the opposite of this module; Relationship management. Where the financial management system is strict, the relationship management system will be shown to be flexible and expandable.

4
Relationship Management

Relationship Management software is a typical result of what ERP applications have achieved.

In earlier days, everyone had a rolodex on their desk with phone numbers and addresses and salespeople would always know by heart who was a good customer and which customers were always late paying or had bad margins.

The introduction of RM software completely changed that allowing us to maintain all of our companies' contacts in a single place and analyze sales data easily.

Relationship management has been part of Microsoft Dynamics since version 2.0 and was dramatically changed and improved in version 3.0. The current Microsoft Dynamics NAV RM software is mostly the same as in that version except for the Microsoft Outlook integration that keeps changing in every version.

In this chapter, we will dive deep into this module. After reading this chapter, we will have a good understanding of the concepts and how to maintain master data and analyze transaction data.

We will also make some application changes in the relationship management part.

How companies work

In traditional accounting software, we differentiate customers and vendors as business relations for invoices, but many companies have many more relationships which we would like to register in our system.

Also, a company or person can have multiple relationships with our company. The best example is my relationship with Microsoft. Like everyone else, I use the software so I am a customer, both in my business and personal life. On the other hand, Microsoft hires me to teach workshops and do presentations, which makes me a vendor. As an MVP, I have a totally different relationship with them. They give me an award and invite me to special events and allow me to access the company store. They also ask for my advice on future versions, so to them I am their consultant.

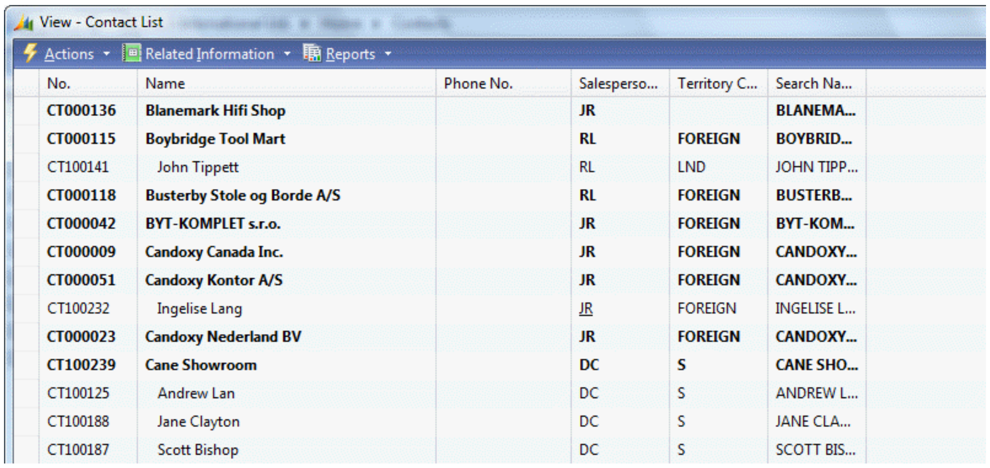
So one person or company can have different roles in RM. Microsoft Dynamics NAV is able to handle all that while maintaining a single point of data entry and maintenance.

Unlike financial applications, RM is much more flexible. The functionality and rules of financial applications are defined by government regulations and are mandatory for companies to comply with. Companies are not forced to use RM but once it is implemented, everyone understands the benefits and would never want to do without them.

Contacts

The starting point of the RM application is the **Contact** table. This is where we store the address, phone numbers, e-mail addresses, etc. of everyone we know.

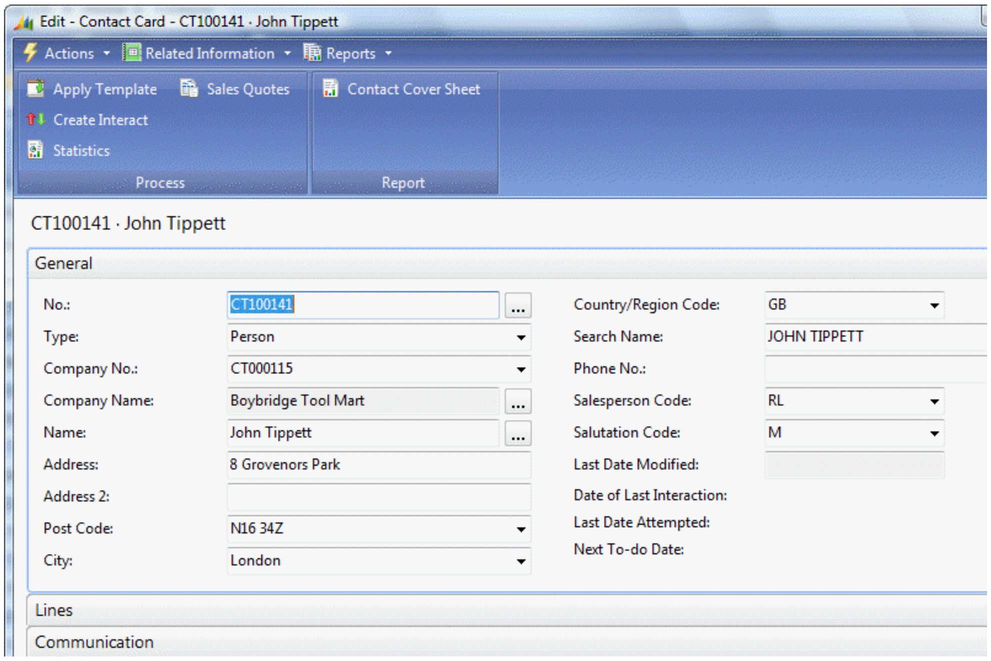
When we open the **Contact List**, we see that companies and persons are grouped for an easy overview.



No.	Name	Phone No.	Salesperso...	Territory C...	Search Na...
CT000136	**Blanemark Hifi Shop**		**JR**		**BLANEMA...**
CT000115	**Boybridge Tool Mart**		**RL**	**FOREIGN**	**BOYBRID...**
CT100141	John Tippett		RL	LND	JOHN TIPP...
CT000118	**Busterby Stole og Borde A/S**		**RL**	**FOREIGN**	**BUSTERB...**
CT000042	**BYT-KOMPLET s.r.o.**		**JR**	**FOREIGN**	**BYT-KOM...**
CT000009	**Candoxy Canada Inc.**		**JR**	**FOREIGN**	**CANDOXY...**
CT000051	**Candoxy Kontor A/S**		**JR**	**FOREIGN**	**CANDOXY...**
CT100232	Ingelise Lang		JR	FOREIGN	INGELISE L...
CT000023	**Candoxy Nederland BV**		**JR**	**FOREIGN**	**CANDOXY...**
CT100239	**Cane Showroom**		**DC**	**S**	**CANE SHO...**
CT100125	Andrew Lan		DC	S	ANDREW L...
CT100188	Jane Clayton		DC	S	JANE CLA...
CT100187	Scott Bishop		DC	S	SCOTT BIS...

As we learned in the previous chapters, a page in Microsoft Dynamics NAV is based on a single table, so that must mean that companies and persons are stored in the same table.

When we open the contact card we can clearly see that this is the case. The **Type** field indicates if the contact is a business or a person and if the person belongs to a company. The **Company No.** field refers to a contact with **Type Company**. This is a one-to-many relationship meaning that if a physical person has a relationship with more than one company, he or she needs to be maintained as many times as there are companies.

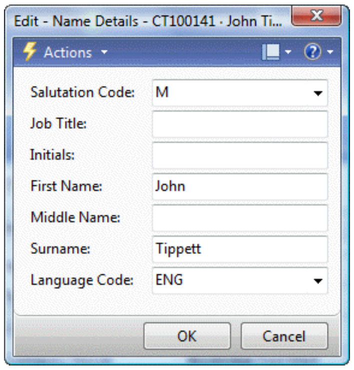


No.:	CT100141	Country/Region Code:	GB
Type:	Person	Search Name:	JOHN TIPPETT
Company No.:	CT000115	Phone No.:	
Company Name:	Boybridge Tool Mart	Salesperson Code:	RL
Name:	John Tippett	Salutation Code:	M
Address:	8 Grovenors Park	Last Date Modified:	
Address 2:		Date of Last Interaction:	
Post Code:	N16 34Z	Last Date Attempted:	
City:	London	Next To-do Date:	

Let's step through the tabs and look at some important fields.

- **No.**: This is the unique key value determined by a number series. Companies and persons have the same numbering.
- **Type**: This indicates if this contact is a person or a company.
- **Company Name**: When the contact is a person and connected to a company, it is automatically populated with that company's name.

- **Name**: This is the name of the contact. If the contact is a person, we can click the **AssistEdit** button to open the name details. The name is automatically broken down in first, middle, and last name depending on the number of words we enter. However, if our contact has a more complex name, like "Walter van den Broek" which is typical for Dutch people, the system is unable to break it down.



Salutation Code:	M
Job Title:	
Initials:	
First Name:	John
Middle Name:	
Surname:	Tippett
Language Code:	ENG

- **Address**: Enter the street where the contact lives or has office.

[It is always best practice to enter the postal address here as this will be used on all documents. For visiting address, use the **Alternative Address** feature]

- **Post Code** and **City**: These fields are connected via the **Post Code** table and one can populate the other if that table is maintained, which is an optional feature.

[Most companies maintain the post code table for their country and manually enter the post codes for foreign countries. Most countries offer a post code/city list for sale or as a web service which will speed up data entry and keep people from entering wrong master data]

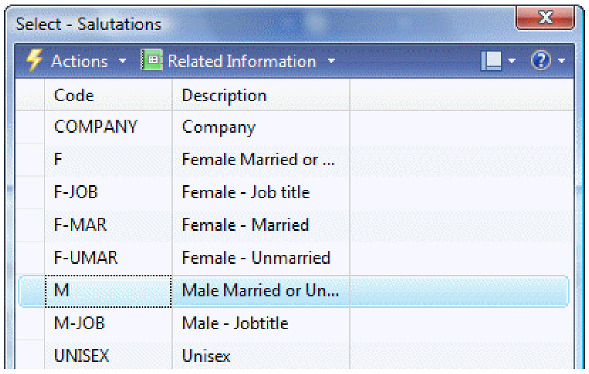
- **Search Name**: This is automatically populated with the **Name** field and lets you search for contacts faster as you can enter this field instead of the **No.** field when referencing to a contact.
- **(Mobile) Phone, Fax** and **Telex No.**: A reference to the phone and fax numbers of this company. The (mobile) phone field also allows you to start an interaction with this contact.

- **Sales Person**: This is the main salesperson for this contact. If this contact is promoted to a customer, the salespersons name will be printed on the order form and invoices.
- **Salutation Code**: This special field refers to how this contact should be addressed. The salutation code table allows you to build phrases like 'Dear Mrs. Brown'. We'll see more of salutation codes in Segments.
- **E-Mail**: This field contains the e-mail address of the contact. By pressing the **E-Mail** button, we can send an e-mail directly.
- **Homepage**: Here goes the URL of the contacts website. We can access the website by clicking on the URL button .
- **Correspondence Type**: This field is used when we create a Microsoft Word document in an interaction. It indicates if we send a hardcopy, email or fax.
- **Currency Code** and **VAT Registration No.**: When this contact is promoted to a customer or vendor, the currency code and VAT registration no. are inherited from here.
- **Territory Code**: This field can be used to in segments to filter on geographic regions.

Salutation codes

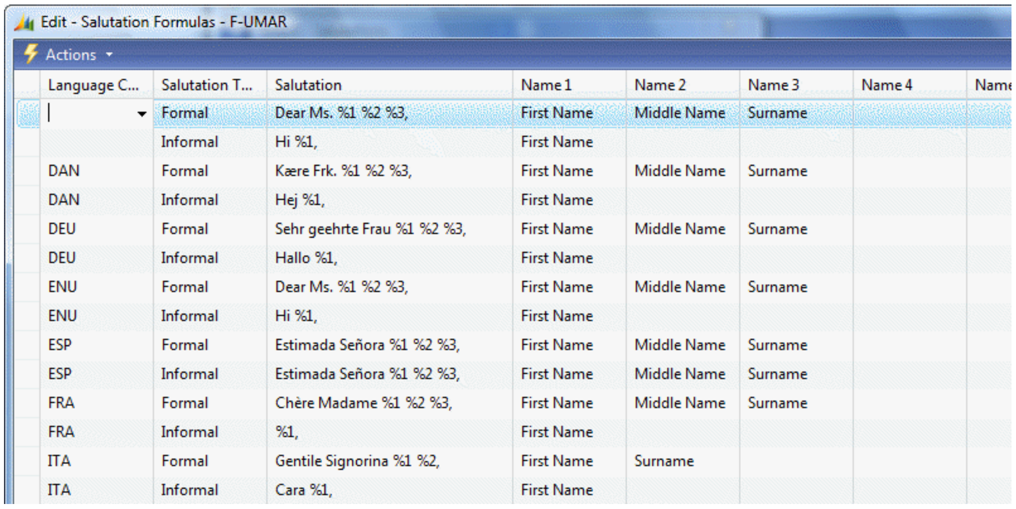
When we do mail merge, we want the letters to start nicely with "Dear Harry", or "Dear Mrs. Brown". This can be done using **Salutation Codes**.

We can create as many codes as we like but a contact can only use one. This is the list in the Cronus Demo Database.



Code	Description
COMPANY	Company
F	Female Married or ...
F-JOB	Female - Job title
F-MAR	Female - Married
F-UMAR	Female - Unmarried
M	Male Married or Un...
M-JOB	Male - Jobtitle
UNISEX	Unisex

There is one salutation code for companies, but most are for persons. When we look at the formulas for **Female Married** or **Unmarried** we see this screen.



Edit - Salutation Formulas - F-UMAR

Language C...	Salutation T...	Salutation	Name 1	Name 2	Name 3	Name 4	Name
	Formal	Dear Ms. %1 %2 %3,	First Name	Middle Name	Surname		
	Informal	Hi %1,	First Name				
DAN	Formal	Kære Frk. %1 %2 %3,	First Name	Middle Name	Surname		
DAN	Informal	Hej %1,	First Name				
DEU	Formal	Sehr geehrte Frau %1 %2 %3,	First Name	Middle Name	Surname		
DEU	Informal	Hallo %1,	First Name				
ENU	Formal	Dear Ms. %1 %2 %3,	First Name	Middle Name	Surname		
ENU	Informal	Hi %1,	First Name				
ESP	Formal	Estimada Señora %1 %2 %3,	First Name	Middle Name	Surname		
ESP	Informal	Estimada Señora %1 %2 %3,	First Name	Middle Name	Surname		
FRA	Formal	Chère Madame %1 %2 %3,	First Name	Middle Name	Surname		
FRA	Informal	%1,	First Name				
ITA	Formal	Gentile Signorina %1 %2,	First Name	Surname			
ITA	Informal	Cara %1,	First Name				

We can enter a formal and informal code. The salutation can have up to five variables pointing to **Job Title**, **First Name**, **Middle Name**, **Surname**, **Initials**, and **Company Name**.

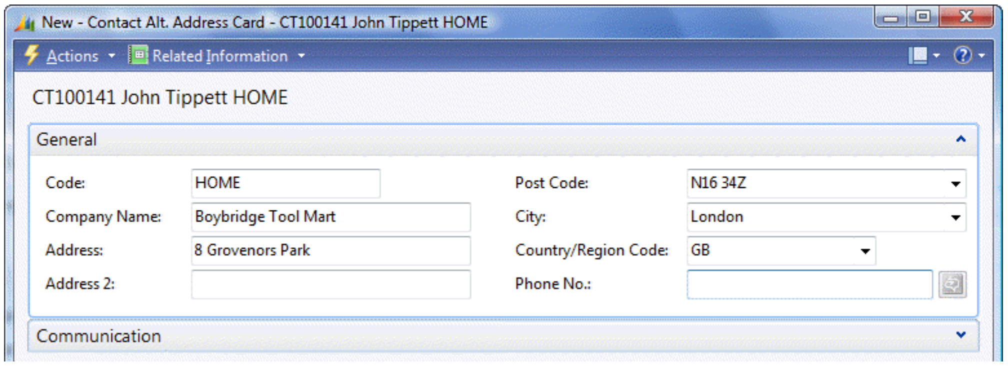
When we look at the result for Karen Friske it will be "Dear Ms. Karen Friske" or "Hi Karen".

At the end of this chapter, we will look at how to create extra salutation types.

Alternative addresses

Like we said earlier in this chapter, it is best practice to use the address fields in the contact table for the postal address as this will be printed on all documents.

In the **Alternative Address** table, we can add as many other addresses to a **Contact** as we want.


New - Contact Alt. Address Card - CT100141 John Tippett HOME
Actions
Related Information

CT100141 John Tippett HOME

General

Code:	HOME	Post Code:	N16 34Z
Company Name:	Boybridge Tool Mart	City:	London
Address:	8 Grovenors Park	Country/Region Code:	GB
Address 2:		Phone No.:	

Communication

Although the codes are not related to anything, it is best practice to have a rule here. Always use the same code for home or office addresses. We can later use this when printing labels or segments.

An alternative address can also have a valid to and valid end date to control which alternative address is currently active.

Create as

In *Chapter 2*, we saw that the contact table is the umbrella data of the customer, vendor, and bank account master data tables.

Each contact of the type company can be promoted as one of these tables. The benefit is that all address information fields have a single place of maintenance and are inherited. It also allows us to analyze sales data into relationship management as we will see later in this chapter, when discussing segments.

When we create master data, a different number series is used. At the end of this chapter, we will look at how to change that in the code.

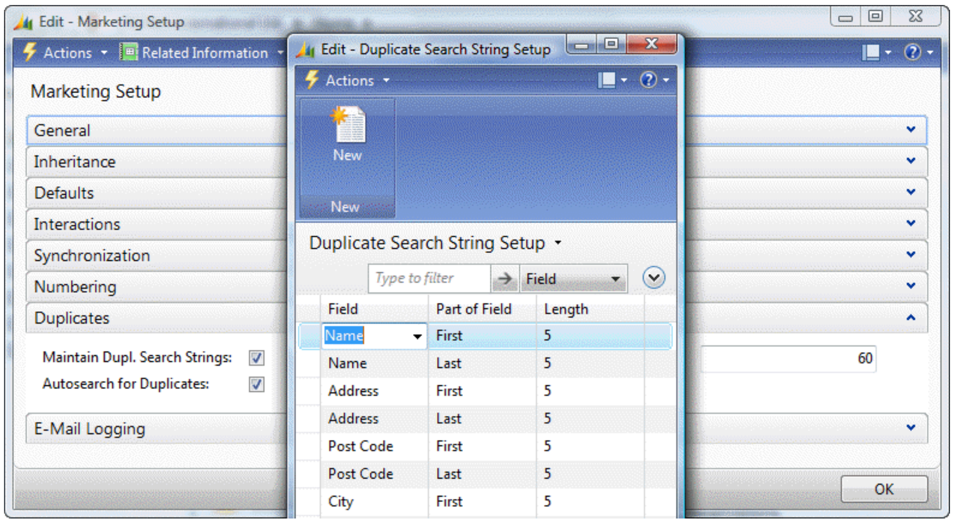
A contact of type person cannot be created as the **Customer**, **Vendor**, or **Bank Account**.

Duplicates

When entering new contacts the system can search for duplicate contacts. In the Duplicate Search String Setup table we can enable the filtering on eight fields: **Name**, **Name2**, **Address**, **Address** 2, **Post Code**, **City**, **Phone No.** and **VAT Registration No**.

For each field we can set up which part should be used when searching for a duplicate. We can use the option **First** and **Last** and a length which is useful for **Name**, **Address**, and **City** fields. Using **First** with the full length of the field will search for an exact match which is useful for **Post Code**, **Phone Nos**. and **VAT Registration No**.

In the **Marketing Setup** table, we can specify the percentage of matching criteria that should result in a warning.



Maintain Dupl. Search Strings: ✓
Autosearch for Duplicates: ✓
60

Field	Part of Field	Length
Name	First	5
Name	Last	5
Address	First	5
Address	Last	5
Post Code	First	5
Post Code	Last	5
City	First	5

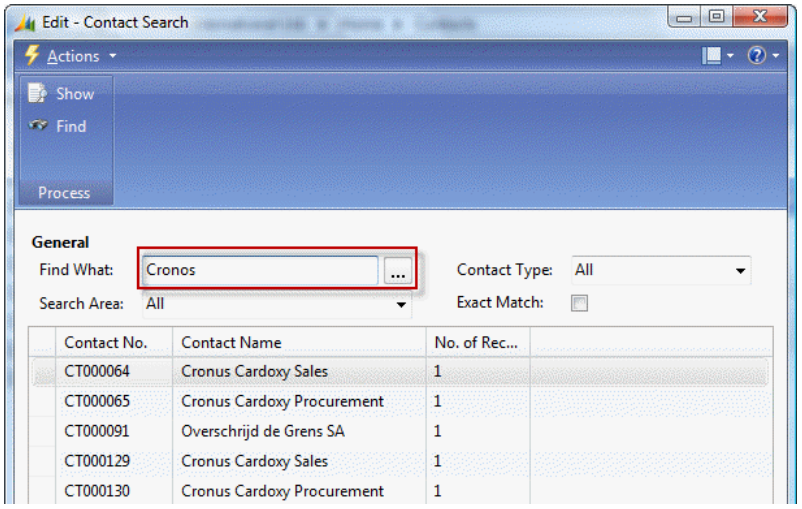
For each contact, the system will save these values in the `Cont. Duplicate Search String table (5086)`.

When we enter a new contact, the system will also generate the same strings and compare these to the ones in the database. When there is a match, the system will show a warning with the duplicate contacts.

Search

To search for a specific contact we can use the **Contact Search** functionality. This has a phonetic search feature allowing for small typos to be ignored. It searches most of the fields in the contact table for the value.

It is also possible to add `AND` and `OR` operators, and to exclude words.



Edit - Contact Search

Actions

Show

Find

Process

General

Find What: Cronos

Search Area: All

Contact Type: All

Exact Match:

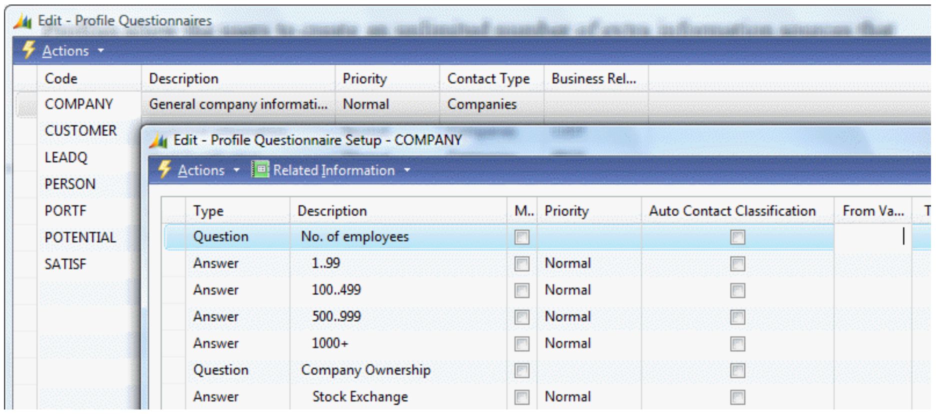
Contact No.	Contact Name	No. of Rec...
CT000064	Cronus Cardoxy Sales	1
CT000065	Cronus Cardoxy Procurement	1
CT000091	Overschrijd de Grens SA	1
CT000129	Cronus Cardoxy Sales	1
CT000130	Cronus Cardoxy Procurement	1

Profiles

The contact table has a very limited number of fields and does not allow for much creativity for us to add flexible information. This is where profiles are used.

Profiles allow the users to create an unlimited number of extra information sources that can be manually or automatically populated.

Let's have a look at an example profile:



Edit - Profile Questionnaires

Actions

Code	Description	Priority	Contact Type	Business Rel...
COMPANY	General company informati...	Normal	Companies	
CUSTOMER				
LEADQ				
PERSON				
PORTF				
POTENTIAL				
SATISF				

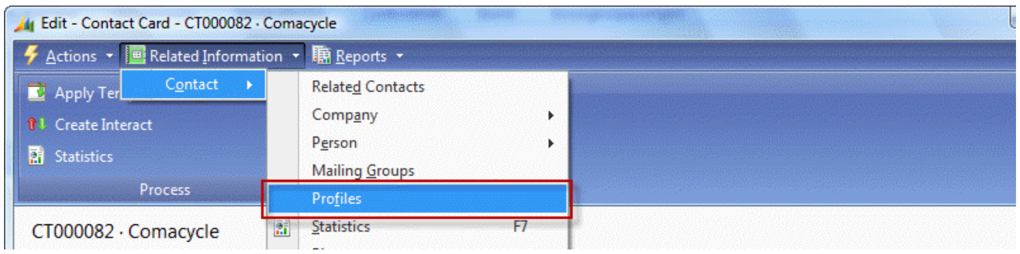
Edit - Profile Questionnaire Setup - COMPANY

Actions Related Information

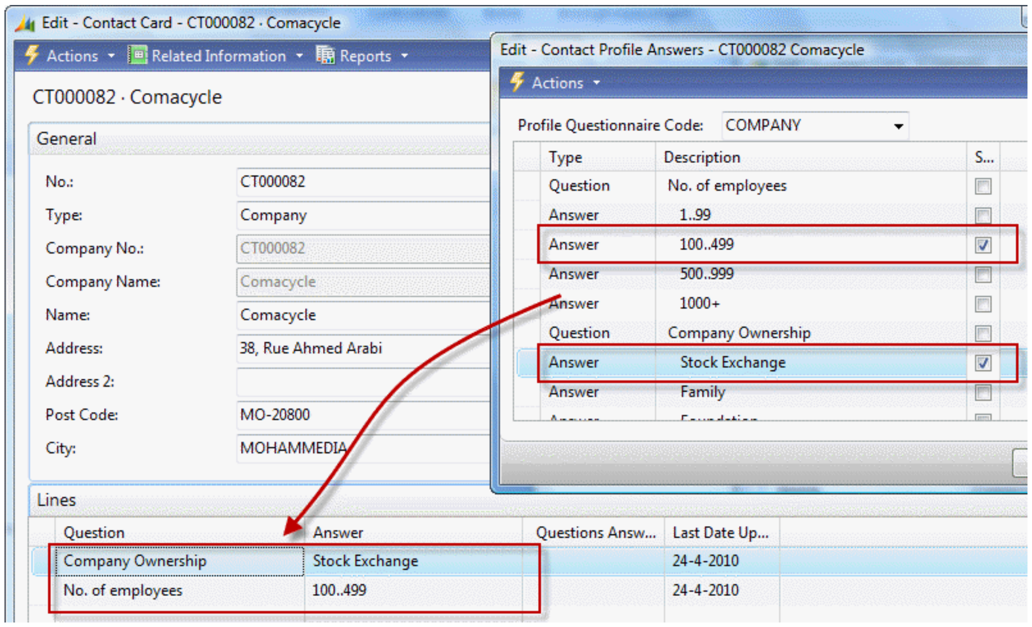
Type	Description	M..	Priority	Auto Contact Classification	From Va...	T
Question	No. of employees	☐		☐		
Answer	1..99	☐	Normal	☐		
Answer	100..499	☐	Normal	☐		
Answer	500..999	☐	Normal	☐		
Answer	1000+	☐	Normal	☐		
Question	Company Ownership	☐		☐		
Answer	Stock Exchange	☐	Normal	☐		

This profile is for contacts of type **Company**. It has **Question** and **Answer** lines. A question can have one or multiple answers and we can define as many questions and answers as we want. The last column shows how many contacts have this profile answer.

A profile is used from the **Contact** card.



When we click this, a new page opens where we can select the required profile and answer the questions.


Edit - Contact Card - CT000082 · Comacycle
Actions
Related Information
Reports
CT000082 · Comacycle
General
Edit - Contact Profile Answers - CT000082 Comacycle
Actions
Lines

No.:	CT000082
Type:	Company
Company No.:	CT000082
Company Name:	Comacycle
Name:	Comacycle
Address:	38, Rue Ahmed Arabi
Address 2:	
Post Code:	MO-20800
City:	MOHAMMEDIA

Profile Questionnaire Code: COMPANY

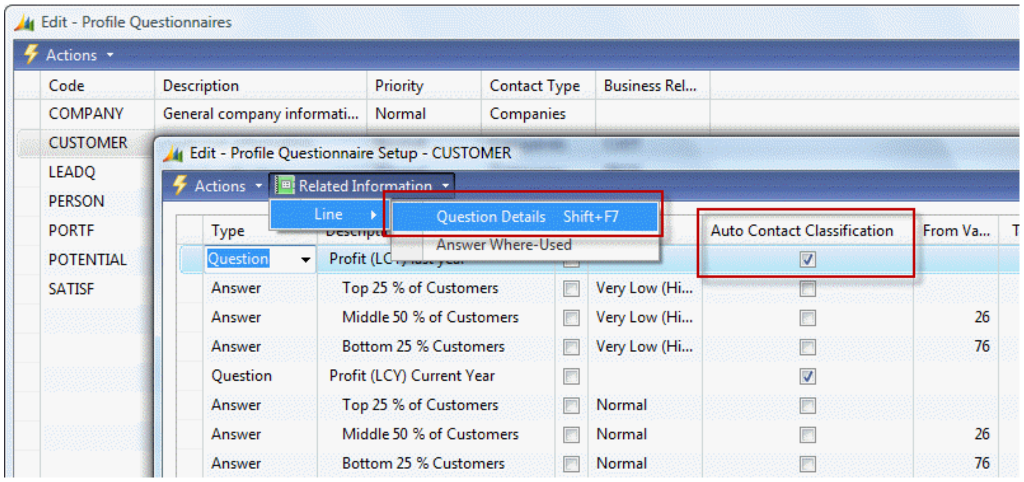
Type	Description	S...
Question	No. of employees	☐
Answer	1..99	☐
Answer	100..499	✓
Answer	500..999	☐
Answer	1000+	☐
Question	Company Ownership	☐
Answer	Stock Exchange	✓
Answer	Family	☐

Question	Answer	Questions Answ...	Last Date Up...
Company Ownership	Stock Exchange		24-4-2010
No. of employees	100..499		24-4-2010

The answers are displayed in the **Lines** subpage of the contact card.

Automatic profiles

Profiles can also be automatically answered based on formulas. This is done using the **Auto Contact Classification** option and setting up the **Question Details**.



Edit - Profile Questionnaires

Actions

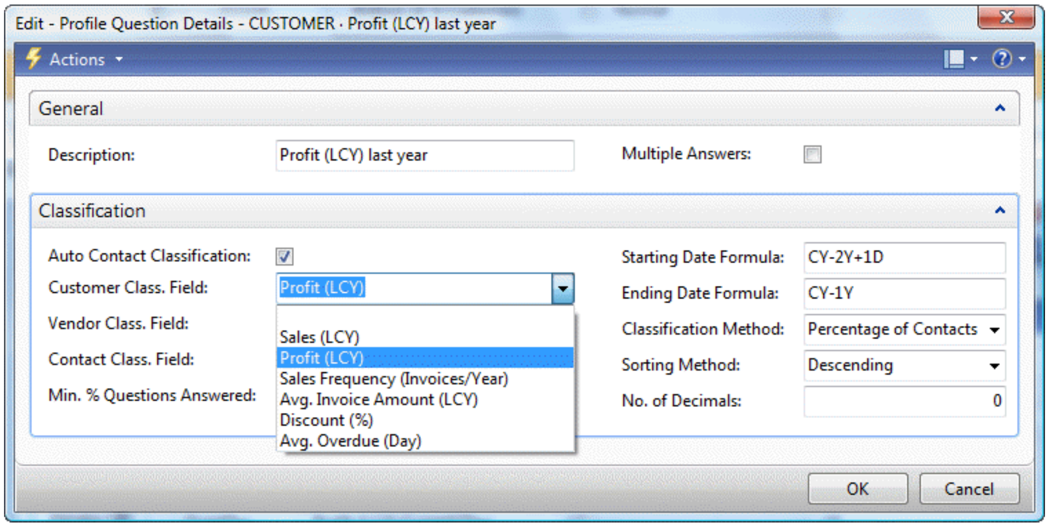
Code	Description	Priority	Contact Type	Business Rel...
COMPANY	General company informati...	Normal	Companies	
CUSTOMER				
LEADQ				
PERSON				
PORTF				
POTENTIAL				
SATISF				

Edit - Profile Questionnaire Setup - CUSTOMER

Actions · Related Information · Line · Question Details Shift+F7 · Answer Where-Used

Type	Descripti...			Auto Contact Classification	From Va...	T
Question	Profit (LC...			✓		
Answer	Top 25 % of Customers	☐	Very Low (Hi...	☐		
Answer	Middle 50 % of Customers	☐	Very Low (Hi...	☐	26	
Answer	Bottom 25 % Customers	☐	Very Low (Hi...	☐	76	
Question	Profit (LCY) Current Year	☐		✓		
Answer	Top 25 % of Customers	☐	Normal	☐		
Answer	Middle 50 % of Customers	☐	Normal	☐	26	
Answer	Bottom 25 % Customers	☐	Normal	☐	76	

The **Question Details** are fixed and hard coded. They depend on the relationship between a contact and a customer or vendor as discussed earlier in this chapter.



Edit - Profile Question Details - CUSTOMER · Profit (LCY) last year

Actions

General

Description:	Profit (LCY) last year	Multiple Answers:	☐

Classification

Auto Contact Classification:	✓	Starting Date Formula:	CY-2Y+1D
Customer Class. Field:	Profit (LCY)	Ending Date Formula:	CY-1Y
Vendor Class. Field:		Classification Method:	Percentage of Contacts
Contact Class. Field:		Sorting Method:	Descending
Min. % Questions Answered:		No. of Decimals:	0

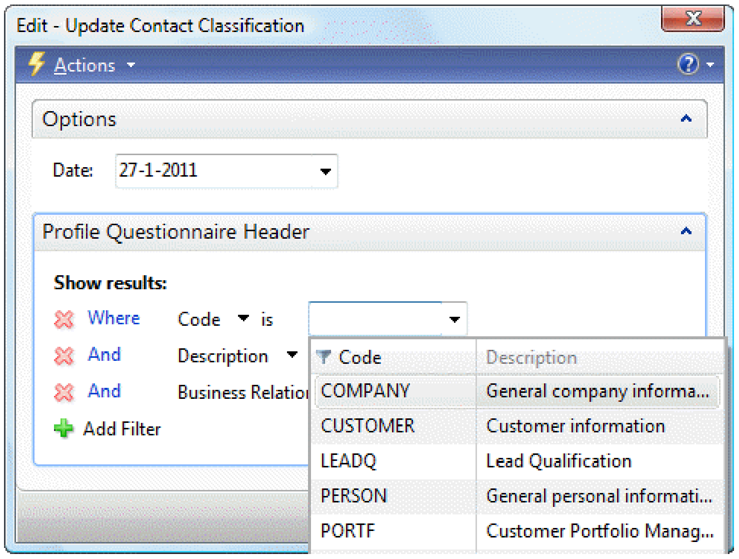
Sales (LCY)
Profit (LCY)
Sales Frequency (Invoices/Year)
Avg. Invoice Amount (LCY)
Discount (%)
Avg. Overdue (Day)

OK · Cancel

We will not describe all possibilities as this is very well covered in the online help.

When the questions are set up, the answers should have a **From Value** and **To Value** to allow the system to pick the right one.

To generate the answers, a batch job is used called **Update Contact Classification** where we can filter on a **Profile**.



Edit - Update Contact Classification

Options

Date: 27-1-2011

Profile Questionnaire Header

Show results:

Where Code is

And Description

And Business Relatio

Add Filter

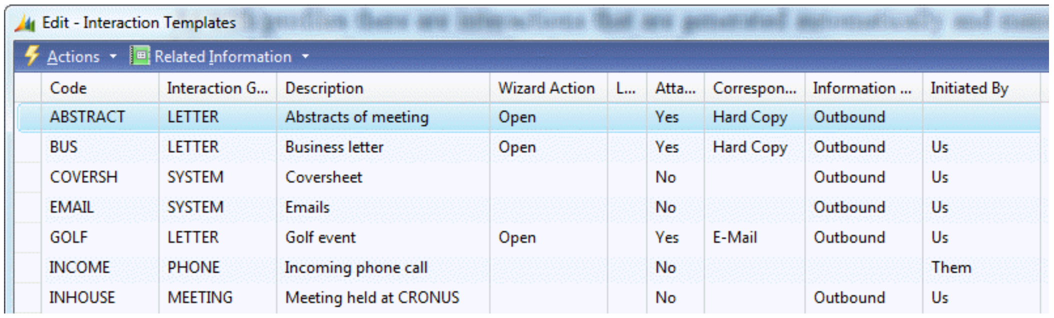
Code	Description
COMPANY	General company informa...
CUSTOMER	Customer information
LEADQ	Lead Qualification
PERSON	General personal informati...
PORTF	Customer Portfolio Manag...

Interactions

We have all kinds of interaction moments with our contacts. Whether they are phone calls, mailings, or sending an invoice, we can register them in Microsoft Dynamics NAV.

As with profiles, there are interactions that are generated automatically and manually. Manual interactions are created using a wizard.

All interactions relate to an **Interaction Template Code**. The system allows for an unlimited number of codes we can define ourselves. The interaction code will also determine how the rest of wizard will behave.



Edit - Interaction Templates

Code	Interaction G...	Description	Wizard Action	L...	Atta...	Correspon...	Information ...	Initiated By
ABSTRACT	LETTER	Abstracts of meeting	Open		Yes	Hard Copy	Outbound	
BUS	LETTER	Business letter	Open		Yes	Hard Copy	Outbound	Us
COVERSH	SYSTEM	Coversheet			No		Outbound	Us
EMAIL	SYSTEM	Emails			No		Outbound	Us
GOLF	LETTER	Golf event	Open		Yes	E-Mail	Outbound	Us
INCOME	PHONE	Incoming phone call			No			Them
INHOUSE	MEETING	Meeting held at CRONUS			No		Outbound	Us

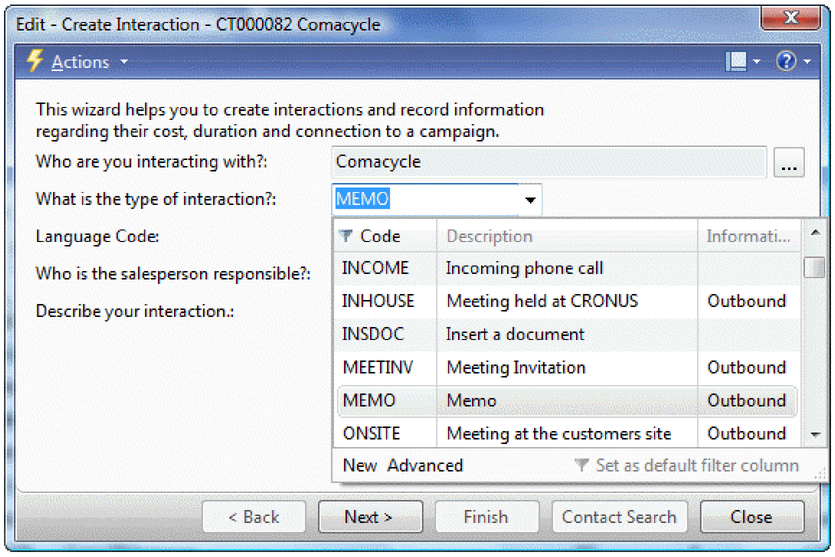
Interactions can be **Inbound** or **Outbound** and **Initiated by Us** or **Them**. These are informative fields.

The **Wizard Action** field determines if the wizard will generate a mail merge document, allow us to attach a previously created document, or do nothing.

Mail merge allows us to create a Word document with all fields from the contact table.

Let's create an interaction and look at how that is done.

To create an interaction we choose **Create Interaction** on the contact card or list and push the **Create Interaction** button from the **Process Actions**. This will open the wizard.


Edit - Create Interaction - CT000082 Comacycle
Actions
< Back
Next >
Finish
Contact Search
Close
New Advanced
Set as default filter column

This wizard helps you to create interactions and record information regarding their cost, duration and connection to a campaign.

Who are you interacting with?: Comacycle

What is the type of interaction?: MEMO

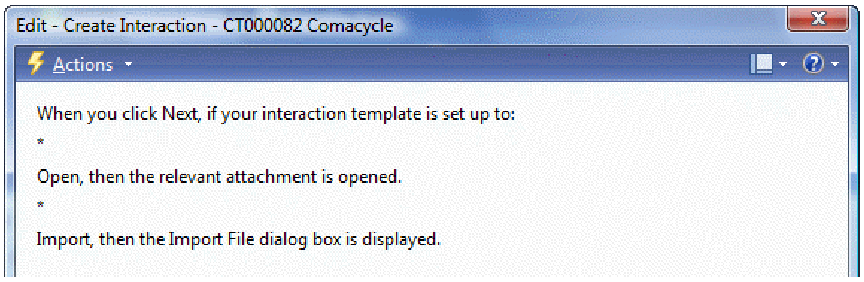
Language Code:

Who is the salesperson responsible?:

Describe your interaction.:

Code	Description	Informati...
INCOME	Incoming phone call	
INHOUSE	Meeting held at CRONUS	Outbound
INSDOC	Insert a document	
MEETINV	Meeting Invitation	Outbound
MEMO	Memo	Outbound
ONSITE	Meeting at the customers site	Outbound

The first page asks us what type of interaction we would like to start. Let's make a Memo.



When you click Next, if your interaction template is set up to:

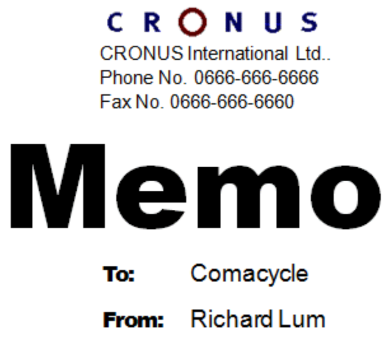
*

Open, then the relevant attachment is opened.

*

Import, then the Import File dialog box is displayed.

This is the next step as our interaction code defines that we will generate a mail merge.

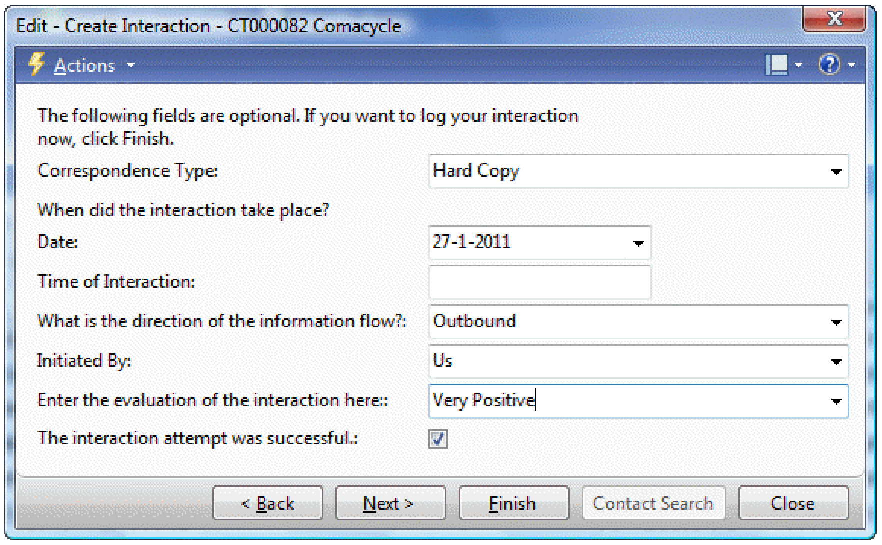


CRONUS International Ltd..
Phone No. 0666-666-6666
Fax No. 0666-666-6660

Memo

To:	Comacycle
From:	Richard Lum

We can now create the Memo in Microsoft Word with all necessary fields already filled.



Edit - Create Interaction - CT000082 Comacycle

The following fields are optional. If you want to log your interaction now, click Finish.

Correspondence Type:	Hard Copy
When did the interaction take place?	
Date:	27-1-2011
Time of Interaction:	
What is the direction of the information flow?:	Outbound
Initiated By:	Us
Enter the evaluation of the interaction here::	Very Positive
The interaction attempt was successful.:	✓

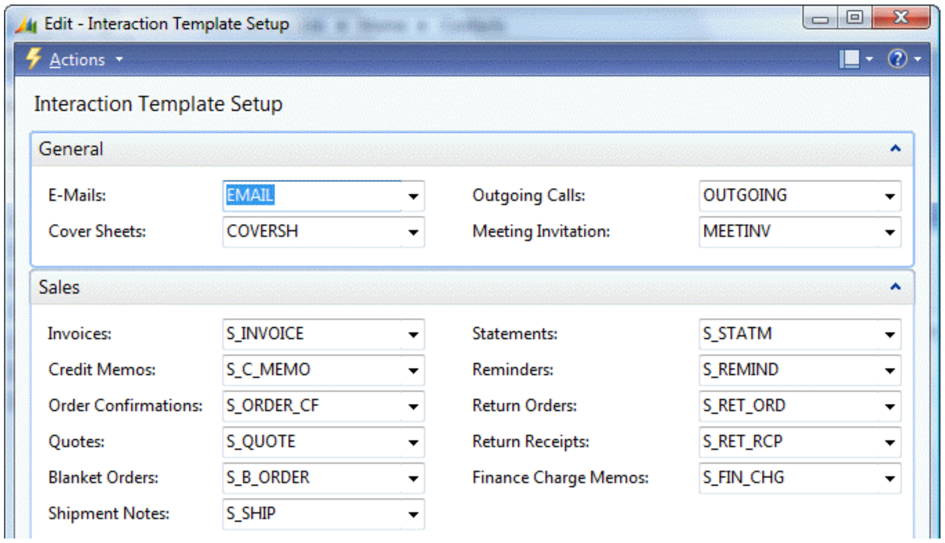
After closing Word, we move on to the next step and when we populate all fields we can finish the wizard. This will save the interaction in the database and print the memo as we choose **Hard Copy** as **Correspondence Type**.

It is also possible to postpone interactions and restart them later.

Automatic interactions

Some interactions can also be automatically generated. For example, every time we print an invoice or shipment.

Which interaction code is used is set up in the **Interaction Template Setup**. For every print, we want an interaction log entry to be generated and we need to set up a code.



E-Mails:	EMAIL	Outgoing Calls:	OUTGOING
Cover Sheets:	COVERSH	Meeting Invitation:	MEETINV

Invoices:	S_INVOICE	Statements:	S_STATM
Credit Memos:	S_C_MEMO	Reminders:	S_REMIND
Order Confirmations:	S_ORDER_CF	Return Orders:	S_RET_ORD
Quotes:	S_QUOTE	Return Receipts:	S_RET_RCP
Blanket Orders:	S_B_ORDER	Finance Charge Memos:	S_FIN_CHG
Shipment Notes:	S_SHIP		

Be careful that when printing a lot of documents, the interaction log entry table can be locked for a longer period forcing other users in the database to wait until the process is completed. To avoid this, enable auto increment on this table like described in this blog. `http://dynamicsuser.net/blogs/mark_brummel/archive/2009/06/28/tip-14-autoincrement-interaction-log-entries.aspx`

Other automatically created interaction log entries are created by segments which we will look at later in the chapter.

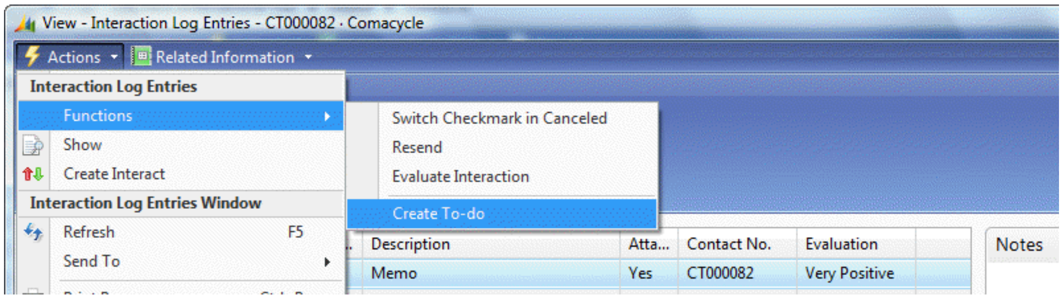
Finished interactions

When completed, the interactions are connected to a contact and can be used for analysis purposes. It is also possible to start a To-Do from an interaction. We'll look at that in the next paragraph.

To-do's

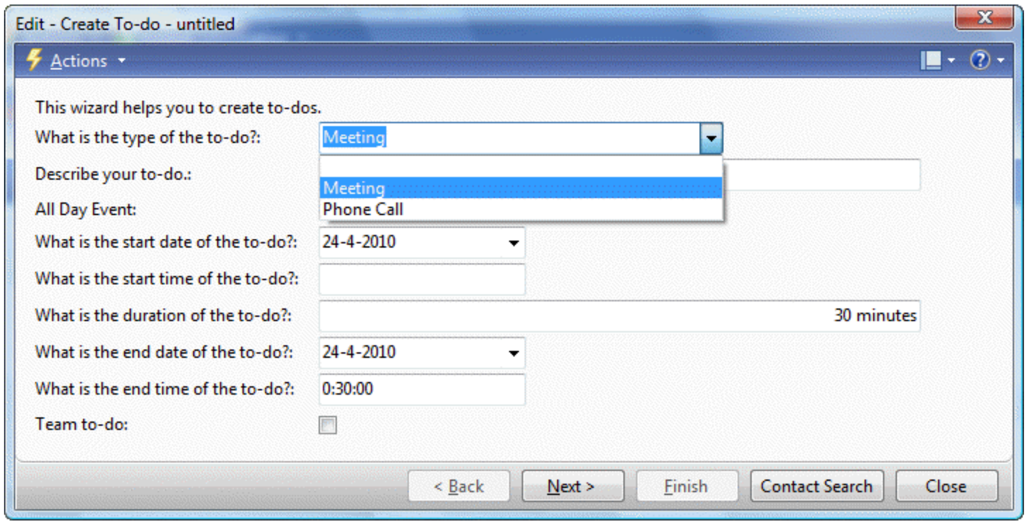
The To-do's are the lowest level of activities in the relationship management model. They are best compared to Masks or Meetings in Microsoft Outlook.

To-do's can be created directly in the system or from another event. We can create a To-do from the interaction we just created. Let's do this.



Description	Atta...	Contact No.	Evaluation
Memo	Yes	CT000082	Very Positive

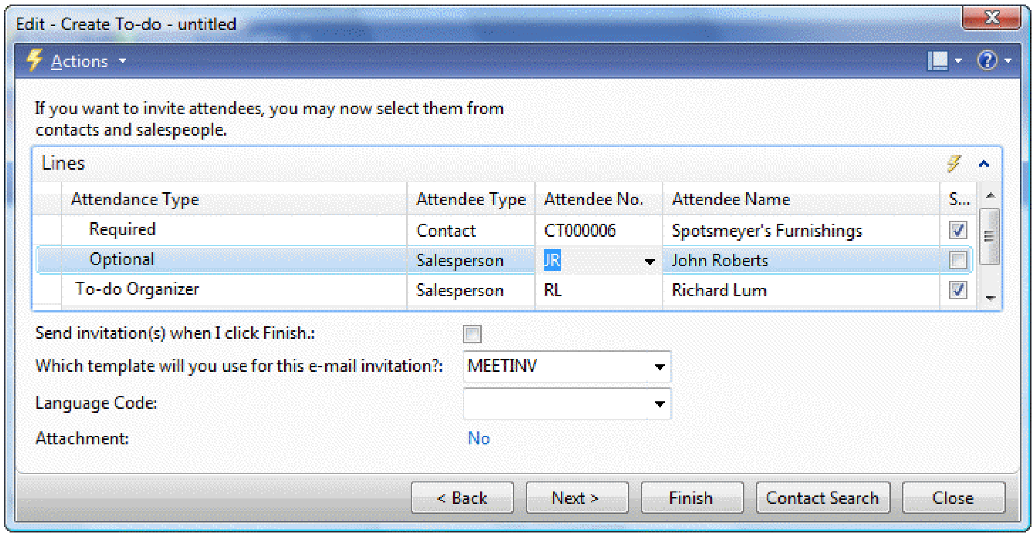
When we click **Create To-do** from the **Interaction Log Entries**, the system shows us a wizard that will guide us through the process, just like the Interaction wizard.



This wizard helps you to create to-dos.

What is the type of the to-do?:	Meeting
Describe your to-do.:	
All Day Event:	
What is the start date of the to-do?:	24-4-2010
What is the start time of the to-do?:	
What is the duration of the to-do?:	30 minutes
What is the end date of the to-do?:	24-4-2010
What is the end time of the to-do?:	0:30:00
Team to-do:	☐

There are three types of To-do's: Standard (Blank), Meeting, and Phone Call. The steps in the wizard depend on the type we select. Let's select a meeting.


Edit - Create To-do - untitled

Actions

If you want to invite attendees, you may now select them from contacts and salespeople.

Lines

Attendance Type	Attendee Type	Attendee No.	Attendee Name	S...
Required	Contact	CT000006	Spotsmeyer's Furnishings	✓
Optional	Salesperson	JR	John Roberts	☐
To-do Organizer	Salesperson	RL	Richard Lum	✓

Send invitation(s) when I click Finish.: ☐

Which template will you use for this e-mail invitation?: MEETINV

Language Code:

Attachment: No

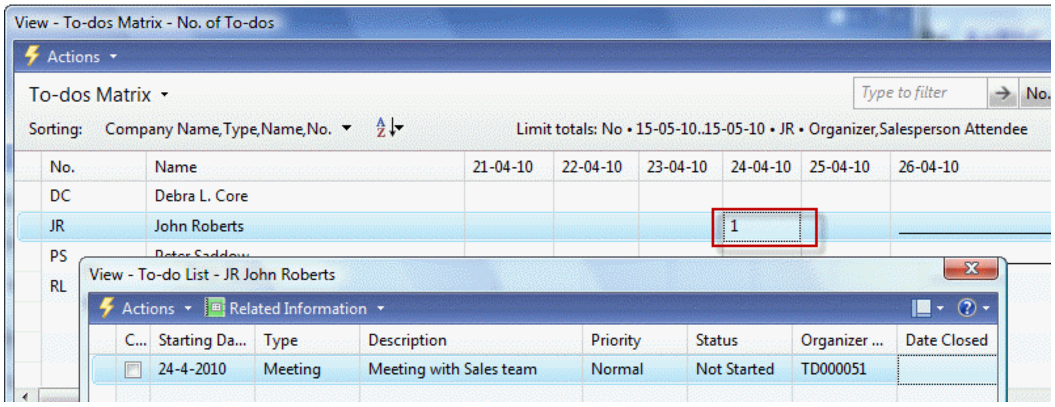
< Back | Next > | Finish | Contact Search | Close

The next step asks the attendees for the meeting and allows a template for the invitation which then again will create an interaction log entry.

To perform this step, the To-do organizer should have a valid e-mail address. This can be set up in the Sales Persons.

The next step only asks for a location, so we will click **Finish**.

When we now open the To-do's from the Sales & Marketing department, we can open the **Sales Person** per day matrix which shows us the meeting we just created.


View - To-dos Matrix - No. of To-dos

Actions

To-dos Matrix

Sorting: Company Name,Type,Name,No.

Limit totals: No • 15-05-10..15-05-10 • JR • Organizer,Salesperson Attendee

No.	Name	21-04-10	22-04-10	23-04-10	24-04-10	25-04-10	26-04-10
DC	Debra L. Core						
JR	John Roberts				1		
PS	Peter Saddow						
RL							

View - To-do List - JR John Roberts

Actions | Related Information

C...	Starting Da...	Type	Description	Priority	Status	Organizer ...	Date Closed
☐	24-4-2010	Meeting	Meeting with Sales team	Normal	Not Started	TD000051	

We'll see more of To-do's when we discuss **Opportunities** and **Outlook Integration**.

Opportunities

When we discussed profiles we could already see that Relationship Management is tightly integrated with the ERP part of the application. This is also the case for opportunities.

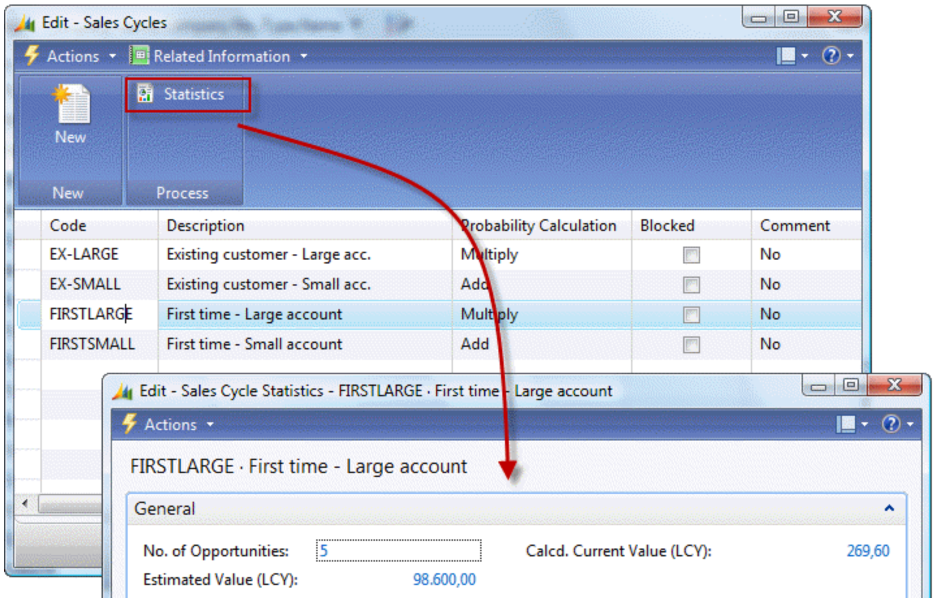
Opportunities allow us to manage all the quote requests we get from our prospects creating a workflow that will guide us to a deal that is won or lost. This then allows us to analyze the won and lost deals and change our business based on this information.

We can analyze the sales pipeline and make a proper judgment of our future order position allowing us to schedule capacity in time.

Workflow

Each opportunity we create will follow a **Sales Cycle** in the system. This will guide us step by step through the process.

Let's have a look at the sales cycles in the Cronus Database.



Code	Description	Probability Calculation	Blocked	Comment
EX-LARGE	Existing customer - Large acc.	Multiply	☐	No
EX-SMALL	Existing customer - Small acc.	Add	☐	No
FIRSTLARGE	First time - Large account	Multiply	☐	No
FIRSTSMALL	First time - Small account	Add	☐	No

FIRSTLARGE · First time - Large account

No. of Opportunities:	5	Calcd. Current Value (LCY):	269,60
Estimated Value (LCY):	98.600,00		

There are four sales **cycles** defined. The most important field is the Probability **Calculation** formula. This will determine how the system calculates the **Calculated Current Value** of all opportunities with this code. We can see the **Calculated Current Value** by opening the **Statistics** window of a **Sales Cycle**.

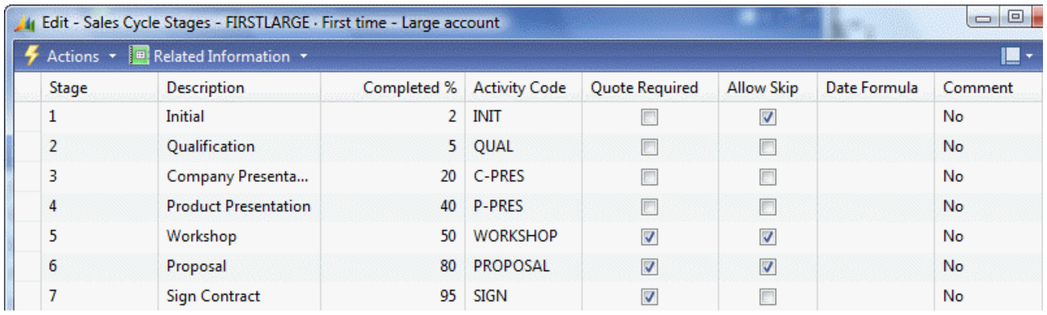
There are four options to choose from: `Multiply`, `Add`, `Chances of Success %` and `Completed %`. The function `UpdateEstimates` in the `Opportunity Entry (5093)` table calculates this.

```
UpdateEstimates()
IF SalesCycleStage.GET("Sales Cycle Code","Sales Cycle Stage")
THEN BEGIN
  SalesCycle.GET("Sales Cycle Code");
  CASE SalesCycle."Probability Calculation" OF
    SalesCycle."Probability Calculation"::Multiply:
      BEGIN
        "Probability %" := "Chances of Success %" *
                      SalesCycleStage."Completed %" / 100;
      END;
    SalesCycle."Probability Calculation"::Add:
      BEGIN
        "Probability %" := ("Chances of Success %" +
                      SalesCycleStage."Completed %") / 2;
      END;
    SalesCycle."Probability Calculation"::"Chances of Success %":
      BEGIN
        "Probability %" := "Chances of Success %";
      END;
    SalesCycle."Probability Calculation"::"Completed %":
      BEGIN
        "Probability %" := SalesCycleStage."Completed %";
      END;
  END;
  "Completed %" := SalesCycleStage."Completed %";
  "Calcd. Current Value (LCY)" := "Estimated Value (LCY)" *
                           "Probability %" / 100;
END;
```

The `Probability Calculation` first calculated a field `Probability %` which then leads to the required `Calculated Current Value`.

Sales stages

Each sales cycle has different stages that will guide us through the sales process.



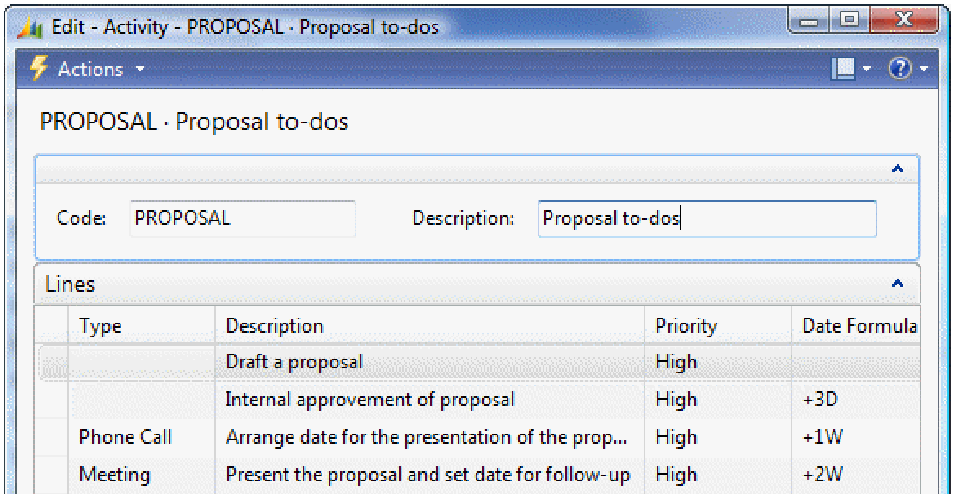
Edit - Sales Cycle Stages - FIRSTLARGE · First time - Large account

Stage	Description	Completed %	Activity Code	Quote Required	Allow Skip	Date Formula	Comment
1	Initial	2	INIT	☐	✓		No
2	Qualification	5	QUAL	☐	☐		No
3	Company Presenta...	20	C-PRES	☐	☐		No
4	Product Presentation	40	P-PRES	☐	☐		No
5	Workshop	50	WORKSHOP	✓	✓		No
6	Proposal	80	PROPOSAL	✓	✓		No
7	Sign Contract	95	SIGN	✓	☐		No

The current **Sales Stage** of an **Opportunity** defines the **Completed** %. We can decide with **Allow Skip** if a sales stage is mandatory. The **Quote Required** will force us to assign a **Sales Quote** to this opportunity as we will see later when we create an opportunity.

Activity codes

Each sales stage has an activity code. This will define which To-do's are created to support us in the sales process.



Edit - Activity - PROPOSAL · Proposal to-dos

PROPOSAL · Proposal to-dos

Code: PROPOSAL   Description: Proposal to-dos

Lines

Type	Description	Priority	Date Formula
	Draft a proposal	High	
	Internal approvement of proposal	High	+3D
Phone Call	Arrange date for the presentation of the prop...	High	+1W
Meeting	Present the proposal and set date for follow-up	High	+2W

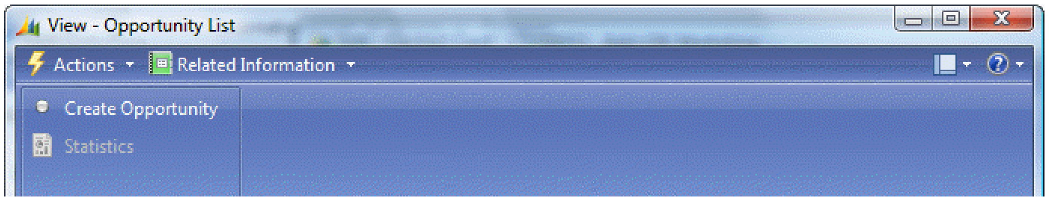
This is a very powerful tool enabling sales people to create a workflow for each sales process.

Let's create an opportunity and see what happens in the system.

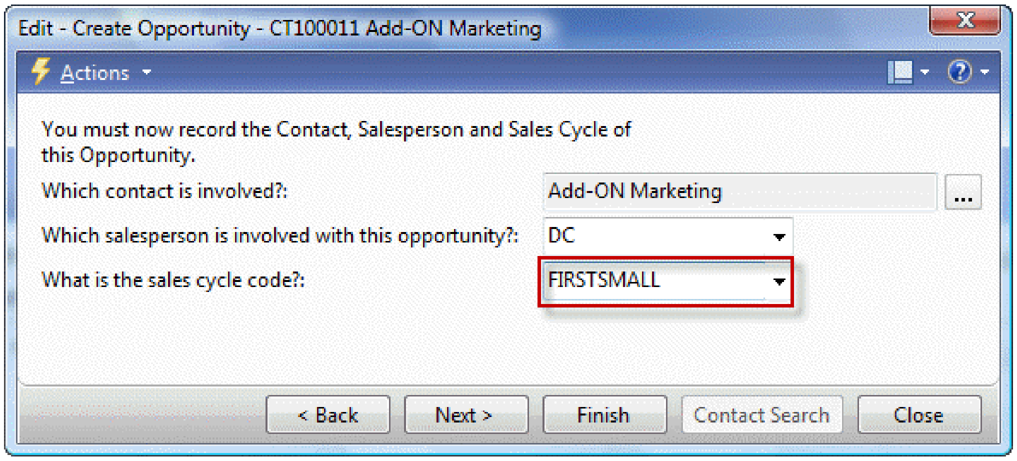
Creating an opportunity

An opportunity starts by selecting an existing contact or creating a new one. From the contact card we can select **Related Information | Contact | Opportunities | List**.

This leads us to a filtered list of opportunities linked to this contact. Here we can select **Create Opportunity**.



This opens the wizard that will guide us though the process. In the first window, we enter the **Description** "Sell Chairs" and click **Next** to take us to the second step.


Edit - Create Opportunity - CT100011 Add-ON Marketing

You must now record the Contact, Salesperson and Sales Cycle of this Opportunity.

Which contact is involved?: Add-ON Marketing

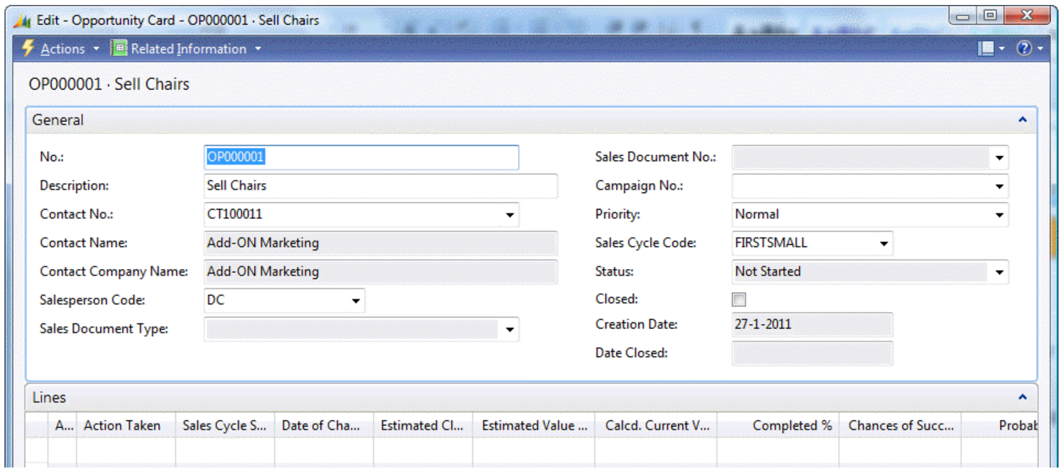
Which salesperson is involved with this opportunity?: DC

What is the sales cycle code?: FIRSTSMALL

In this step, we set the sales cycle code to **FIRSTSMALL** and select **Finish**.

Selecting **Next** will allow us to enter additional information like assigning a sales campaign and it will activate the first stage. We will skip that now and discuss campaigns later in this chapter.

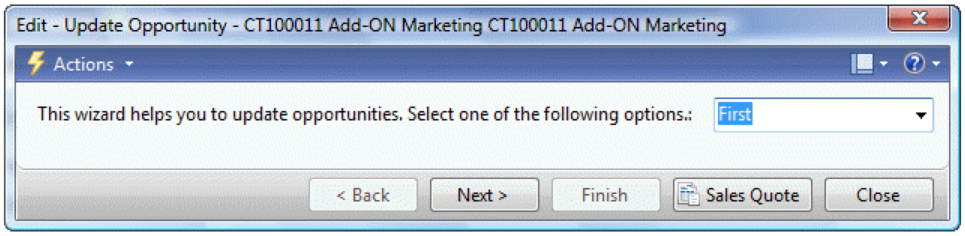
When we now open the created Opportunity, the information should look like the following screenshot. There are no activity lines as we have not yet activated the first stage.



No.:	OP000001	Sales Document No.:	
Description:	Sell Chairs	Campaign No.:	
Contact No.:	CT100011	Priority:	Normal
Contact Name:	Add-ON Marketing	Sales Cycle Code:	FIRSTSMALL
Contact Company Name:	Add-ON Marketing	Status:	Not Started
Salesperson Code:	DC	Closed:	☐
Sales Document Type:		Creation Date:	27-1-2011
		Date Closed:	

A...	Action Taken	Sales Cycle S...	Date of Cha...	Estimated Cl...	Estimated Value ...	Calcd. Current V...	Completed %	Chances of Succ...	Probat

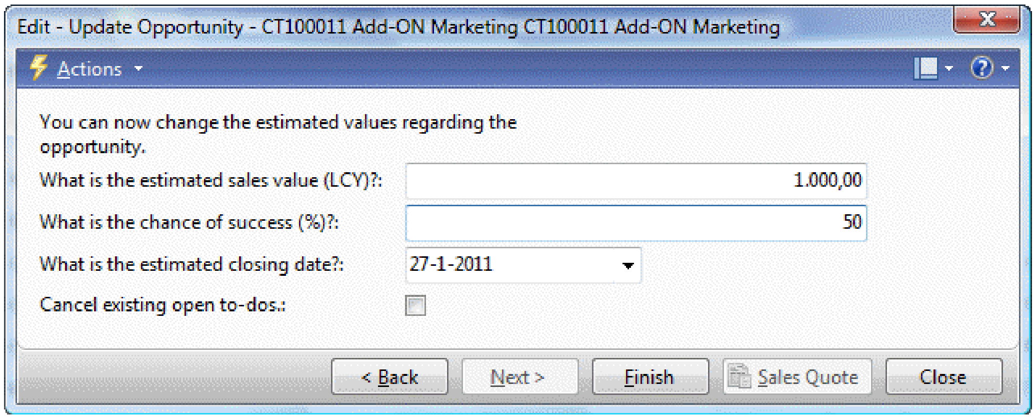
Let's activate the first stage and see what happens. We do that by clicking **Actions** | **Functions** | **Update**. We enter a wizard where we select **First**.



Edit - Update Opportunity - CT100011 Add-ON Marketing CT100011 Add-ON Marketing
Actions
< Back
Next >
Finish
Sales Quote
Close

This wizard helps you to update opportunities. Select one of the following options.: First

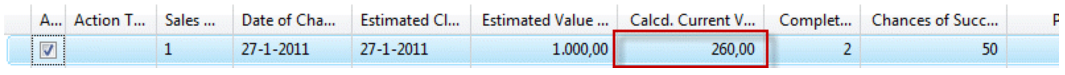
We'll click **Next** twice and enter step three of the wizard. In this step, we should enter the estimated sales value and chance of success (%) of getting this deal. This is important to calculate the **Calculated Estimated Value** that we discussed earlier.


Edit - Update Opportunity - CT100011 Add-ON Marketing CT100011 Add-ON Marketing
Actions
< Back
Next >
Finish
Sales Quote
Close

You can now change the estimated values regarding the opportunity.

What is the estimated sales value (LCY)?:	1.000,00
What is the chance of success (%)?:	50
What is the estimated closing date?:	27-1-2011
Cancel existing open to-dos.:	☐

When we click **Finish**, we come back in the **Opportunity** and see that the current value is 260.



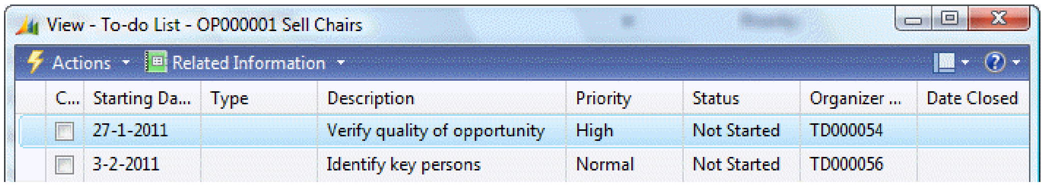
A...	Action T...	Sales ...	Date of Cha...	Estimated Cl...	Estimated Value ...	Calcd. Current V...	Complet...	Chances of Succ...	P
✓		1	27-1-2011	27-1-2011	1.000,00	260,00	2	50	

Since the probability calculation of this sales cycle is **Add**, the formula is:

```
"Probability %" := ("Chances of Success %" +
  SalesCycleStage."Completed %") / 2;

"Calcd. Current Value (LCY)" := "Estimated Value (LCY)" *
  "Probability %" / 100;
```

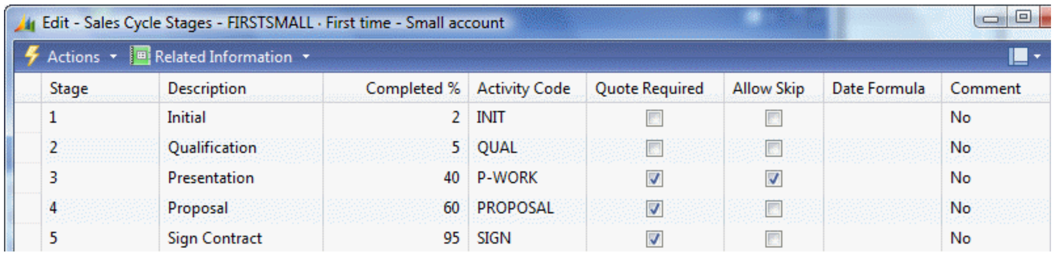
This leads to `(50 + 2) / 2 = 26` and `1000 * 26 / 100 = 260`.

Now we click **Related Information** | **Opportunity** | **To-dos** and see that the system has created two To-do's for us that we have to complete.



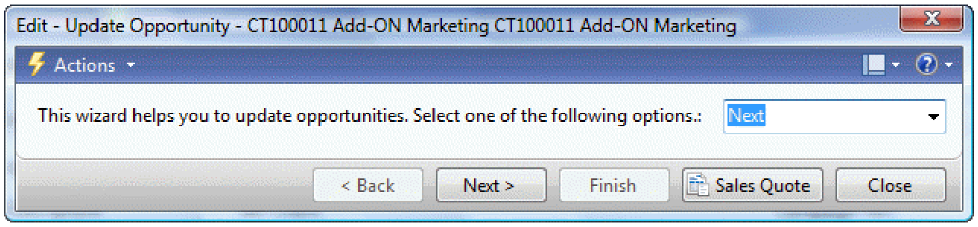
C...	Starting Da...	Type	Description	Priority	Status	Organizer ...	Date Closed
☐	27-1-2011		Verify quality of opportunity	High	Not Started	TD000054	
☐	3-2-2011		Identify key persons	Normal	Not Started	TD000056	

This will help us remember our daily tasks and allow management to see that nothing has been forgotten.


Edit - Sales Cycle Stages - FIRSTSMALL · First time - Small account
Actions
Related Information

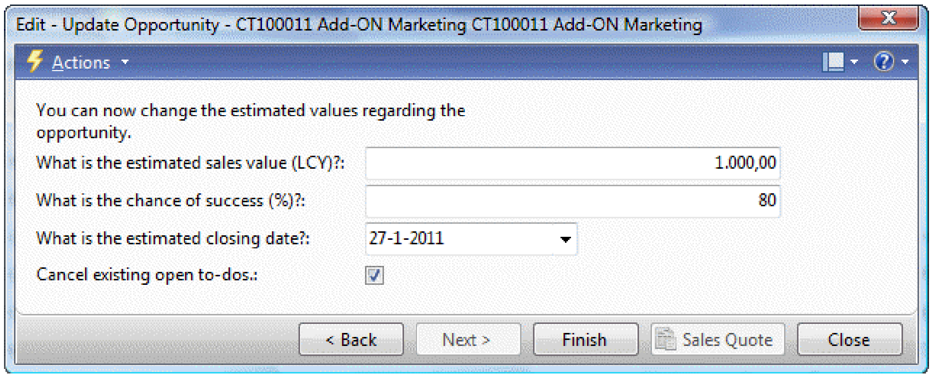
Stage	Description	Completed %	Activity Code	Quote Required	Allow Skip	Date Formula	Comment
1	Initial	2	INIT	☐	☐		No
2	Qualification	5	QUAL	☐	☐		No
3	Presentation	40	P-WORK	✓	✓		No
4	Proposal	60	PROPOSAL	✓	☐		No
5	Sign Contract	95	SIGN	✓	☐		No

The next **Stages** in this sales cycle are **Qualification** and **Presentation**. We can enter these stages by entering the wizard again and selecting **Next**.


Edit - Update Opportunity - CT100011 Add-ON Marketing CT100011 Add-ON Marketing
Actions
< Back
Next >
Finish
Sales Quote
Close

This wizard helps you to update opportunities. Select one of the following options.: Next

After selecting the Next button twice we hit step three. Since one of our To-Do's was verifying the quality of the opportunity, we can now say, for example, that the chance of success is 80%.

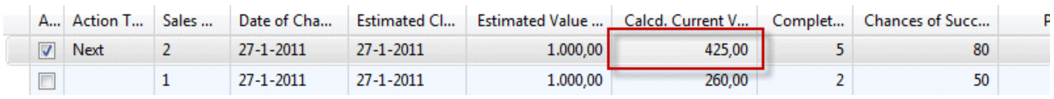

Edit - Update Opportunity - CT100011 Add-ON Marketing CT100011 Add-ON Marketing
Actions
< Back
Next >
Finish
Sales Quote
Close

You can now change the estimated values regarding the opportunity.

What is the estimated sales value (LCY)?:	1.000,00
What is the chance of success (%)?:	80
What is the estimated closing date?:	27-1-2011
Cancel existing open to-dos.:	✓

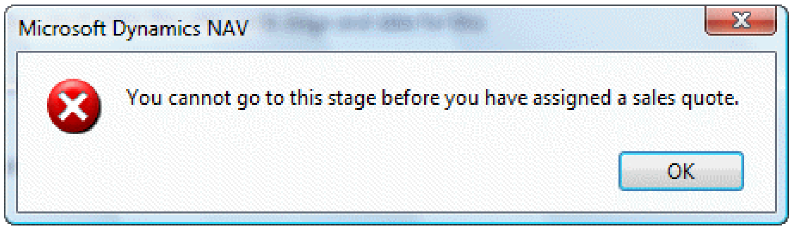
We'll select the **Cancel existing open to-dos** to make sure our workflow is updated.

The **Calculated Current Value** has increased to **425**.



A...	Action T...	Sales ...	Date of Cha...	Estimated Cl...	Estimated Value ...	Calcd. Current V...	Complet...	Chances of Succ...	F
✓	Next	2	27-1-2011	27-1-2011	1.000,00	425,00	5	80	
☐		1	27-1-2011	27-1-2011	1.000,00	260,00	2	50	

When we enter the next stage, we will see this error message telling us that assigning a quote is mandatory to enter the next step.

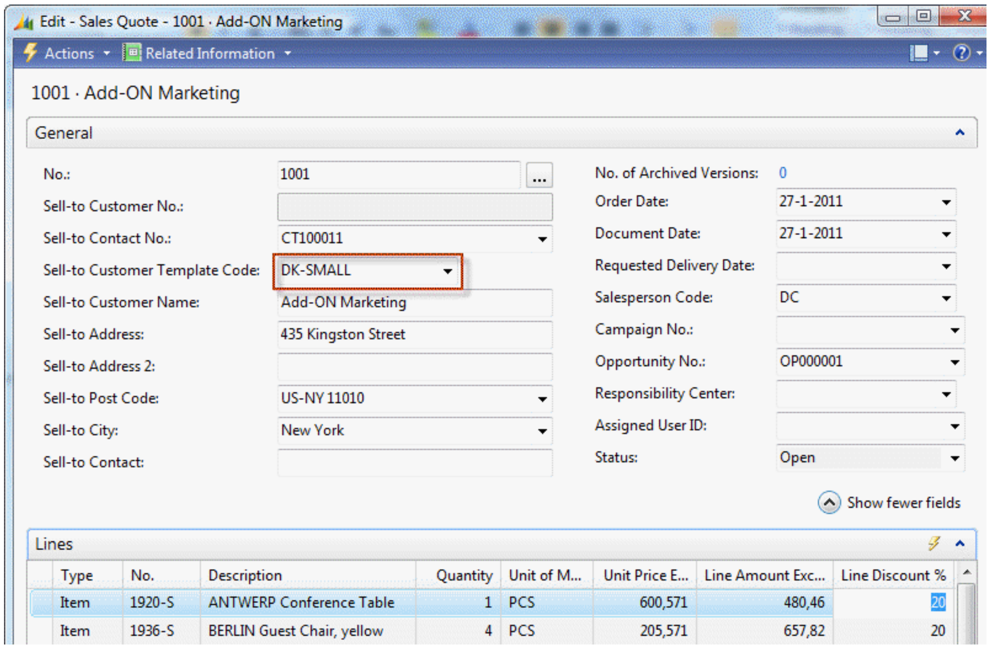


Microsoft Dynamics NAV

You cannot go to this stage before you have assigned a sales quote.

Sales quote

To assign a **Sales Quote** to an opportunity, we select **Actions**, **Functions**, and **Assign Sales Quote** from the opportunity card. This will open a new sales quote with all fields populated from the opportunity.



1001 · Add-ON Marketing

General

No.:	1001	No. of Archived Versions:	0
Sell-to Customer No.:		Order Date:	27-1-2011
Sell-to Contact No.:	CT100011	Document Date:	27-1-2011
Sell-to Customer Template Code:	DK-SMALL	Requested Delivery Date:	
Sell-to Customer Name:	Add-ON Marketing	Salesperson Code:	DC
Sell-to Address:	435 Kingston Street	Campaign No.:	
Sell-to Address 2:		Opportunity No.:	OP000001
Sell-to Post Code:	US-NY 11010	Responsibility Center:	
Sell-to City:	New York	Assigned User ID:	
Sell-to Contact:		Status:	Open

Lines

Type	No.	Description	Quantity	Unit of M...	Unit Price E...	Line Amount Exc...	Line Discount %
Item	1920-S	ANTWERP Conference Table	1	PCS	600,571	480,46	20
Item	1936-S	BERLIN Guest Chair, yellow	4	PCS	205,571	657,82	20

To assign a quote to a contact without a **Sell-to Customer No.** we need to use the **Sell-to Customer Template Code**. This can be used when the **Show more fields** option is activated on the **General** fast tab.

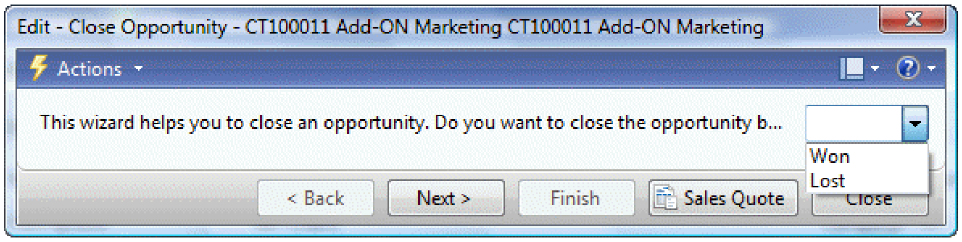
We will select two furniture items and populate the **Quantity** and **Line Discount %** field.

When we now update the opportunity, we can use the quote amount of 1.138,28 which will lead to a Calculated Current Value of 682,97 in step three and 796,80 in step four.

To update the opportunity to step 4, the sales person should have a valid e-mail address which can be set up in the Sales Persons.

Close the deal

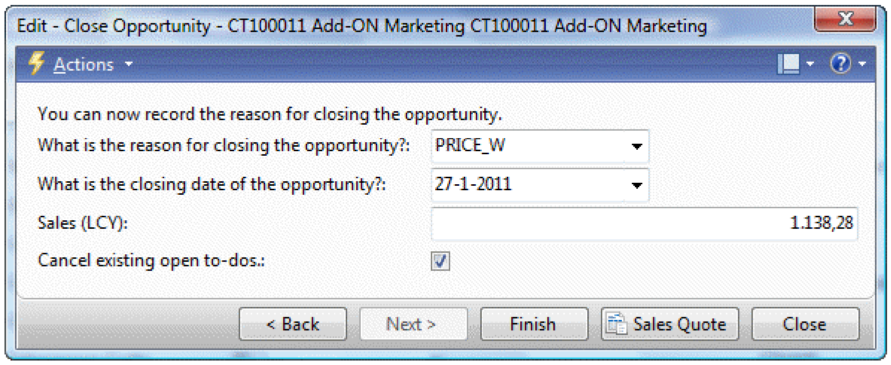
Step five is the final step in the sales cycle stages we used in our example. Now we need to tell the system if the deal is won or lost. To do this, we select **Actions** | **Functions** | **Close** from the opportunity card. We will select **Won** and click **Next**.


Edit - Close Opportunity - CT100011 Add-ON Marketing CT100011 Add-ON Marketing

This wizard helps you to close an opportunity. Do you want to close the opportunity b...

Won
Lost

After selecting a valid reason and the sales amount, we can close the deal.


Edit - Close Opportunity - CT100011 Add-ON Marketing CT100011 Add-ON Marketing

You can now record the reason for closing the opportunity.

What is the reason for closing the opportunity?:	PRICE_W
What is the closing date of the opportunity?:	27-1-2011
Sales (LCY):	1.138,28
Cancel existing open to-dos.:	✓

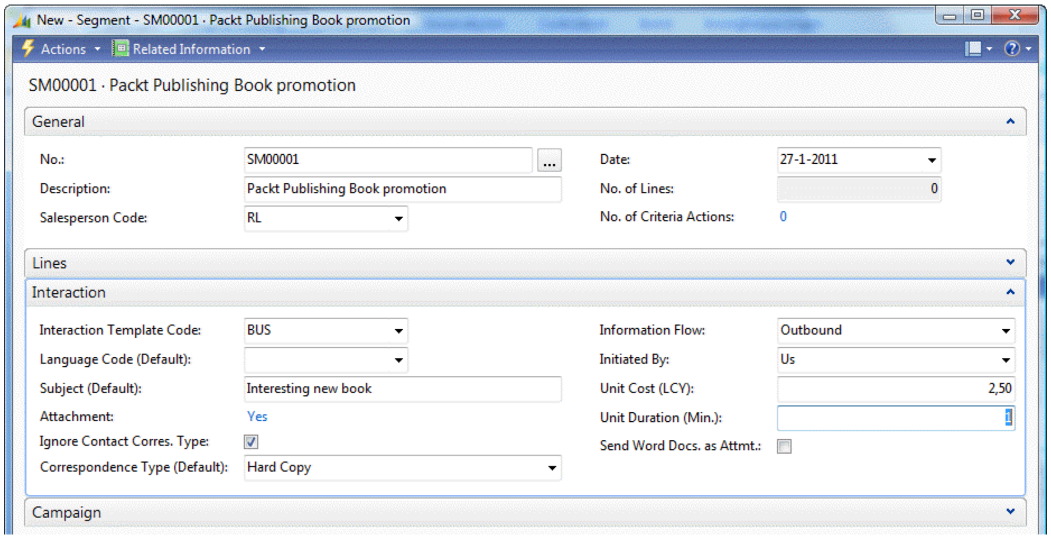
The system now creates a customer for this contact and updates the quote with this number. We need to promote the quote to an order manually.

Segments

Segments allow us to slice and dice the data in our system to create a filtered list of contacts. This information can then be used to create an interaction such as a mailing or start a Sales Campaign.

Since Microsoft Dynamics NAV Relationship Management is integrated with the ERP system, we can filter on both RM and ERP data.

Let's create a new segment and look at the possibilities:



SM00001 · Packt Publishing Book promotion

No.:	SM00001	Date:	27-1-2011
Description:	Packt Publishing Book promotion	No. of Lines:	0
Salesperson Code:	RL	No. of Criteria Actions:	0

Interaction Template Code:	BUS	Information Flow:	Outbound
Language Code (Default):		Initiated By:	Us
Subject (Default):	Interesting new book	Unit Cost (LCY):	2,50
Attachment:	Yes	Unit Duration (Min.):	1
Ignore Contact Corres. Type:	✓	Send Word Docs. as Attmt.:	☐
Correspondence Type (Default):	Hard Copy		

The segment has a **No.** and a **Description**. The No. can be defined using number series.

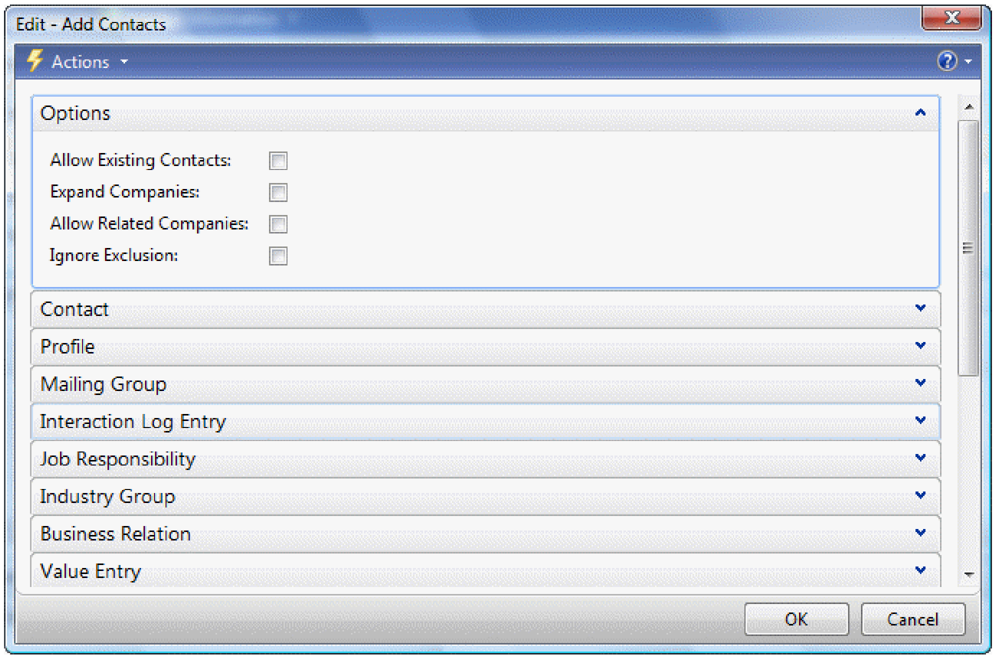
On the **Interaction** tab, we select the **Interaction Template Code**. We will select an interaction that generates a word document, so we can use the mail merge capabilities of segments.

The **Unit Cost** is important for determining the total cost of this segment, especially when we use it with **Campaigns**, as we will see later in the chapter.

Add contacts

With our segment defined, we can now start filtering the system for contact information by clicking **Actions** | **Functions** | **Contacts** | **Add Contacts**.

This opens a selection window allowing us to filter in different parts of the application.



Allow Existing Contacts: ☐
Expand Companies: ☐
Allow Related Companies: ☐
Ignore Exclusion: ☐

- **Contact**: Here we can filter directly on all fields in the contact table. For example: all Contacts in Country 'NL'.
- **Profile**: This allows us to filter on any profile answer. When we use automatic profile answers we can, for example, filter on customers with a specific turnover or profit value.
- **Mailing Group**: We can save any segment to a mailing group allowing easy reuse of previously generated filters.
- **Interaction Log Entry**: We can filter on contacts that have had specific interaction codes. For example, everyone who had a sales invoice in the last year.
- **Job Responsibility**: If we want to send out a mailing to all managers, we select the matching job responsibility code.
- **Industry Group**: This allows us to filter out companies in specific industries.

- **Business Relation**: Business relations are by default used to integrate with customers, vendors, and bank accounts, but can also be expanded with extra information.
- **Value Entry**: This is probably the most powerful filter where we can filter on specific Item Numbers and posting dates from the related contact.
- **Options**:
  - **Allow Existing Contacts**: If you run multiple selections and check this option, the system will create new segment line each time a contact is within the selection.
  - **Expand Companies**: When you select this option, the system will add the persons related to the companies in the selection.
  - **Allow Related Companies**: When **Expand Companies** is selected, this option will delete the company record if a company has one or more persons in the filter.
  - **Ignore Exclusion**: A contact can be ignored on segments. Checking this flag will ignore this field.

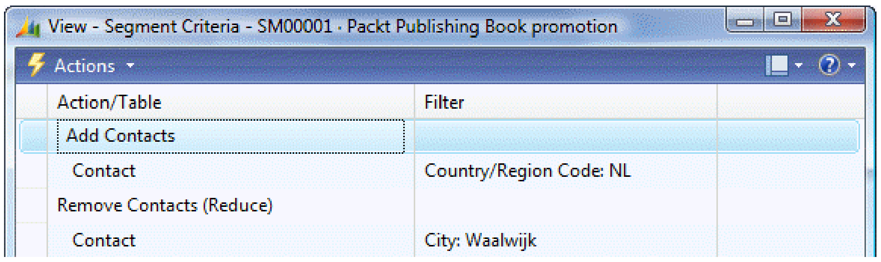
Refine/Reduce contacts

After adding all contacts from The Netherlands, we might want to refine or reduce this. That can be done with the same filtering as Add Contacts. Refine will check if the contacts in the segments match the specific filter criteria, and will remove all contacts in the segment that match the criteria.

We will reduce the segment with the criteria City Waalwijk.

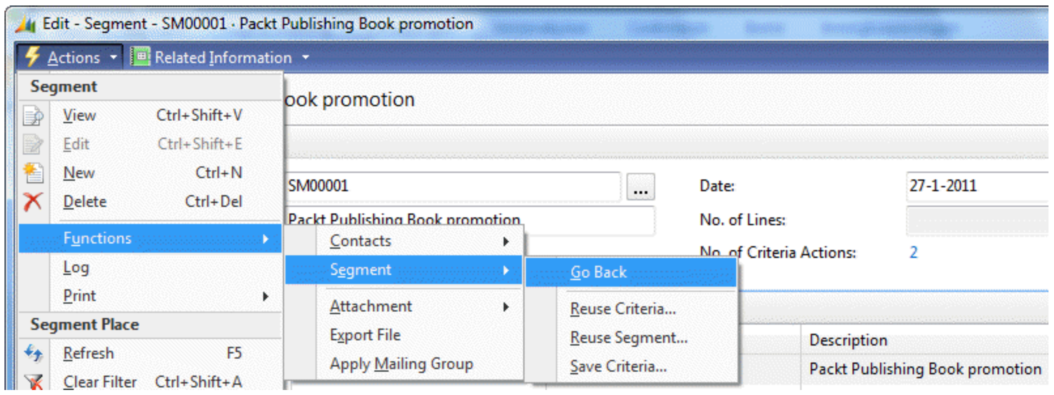
Segment criteria

We can now ask the system what criteria we used by clicking **Related Information | Segments | Criteria**.


View - Segment Criteria - SM00001 · Packt Publishing Book promotion
Actions

Action/Table	Filter
Add Contacts	
Contact	Country/Region Code: NL
Remove Contacts (Reduce)	
Contact	City: Waalwijk

This allows us to see what we did, but also to undo the last actions or save the criteria.



ook promotion

SM00001

Packt Publishing Book promotion

Date: 27-1-2011

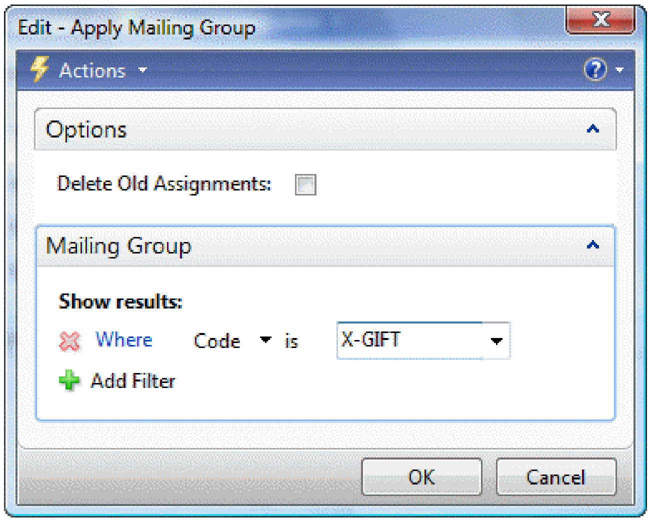
No. of Lines:

No. of Criteria Actions: 2

Description
Packt Publishing Book promotion

Mailing groups

Another option to reuse a segment is to apply a **Mailing Group** to all contacts in a segment. To start this we click **Actions** | **Functions** | **Apply Mailing Group** from the segment card.



Delete Old Assignments: ☐

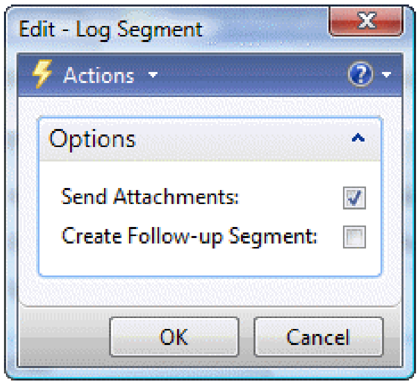
Show results:

Where Code is X-GIFT

This will create a record in the **Contact Mailing Group** table for each contact in the segment.

Log segment

When the segment is finished, it should be logged. Logging the segment will start the mail merge process in our segment and create the **Interaction Log Entries**.



Send Attachments: ✓
Create Follow-up Segment: ☐

Using this option will also print the letters in this example. For an exercise, it might be useful to enable a PDF printer or turn off your printer and remove the print job.

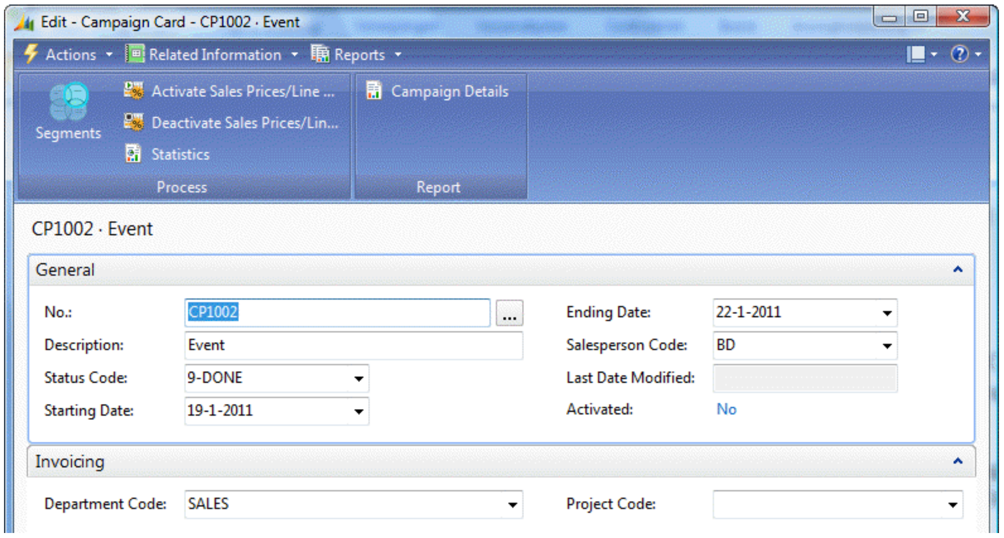
If required the system can directly generate a follow up segment if we would use this segment with a campaign.

Campaigns

Most larger companies with marketing departments have sales campaigns to improve their sales. These are typically periods where some items are more interesting for customers to buy than others.

With the campaigns in Microsoft Dynamics NAV, we can manage the sales prices and see the results of a specific campaign both from a cost and profit viewpoint.

Let's open a campaign and see what information it contains.


Edit - Campaign Card - CP1002 · Event
Actions
Related Information
Reports
Segments
Activate Sales Prices/Line ...
Deactivate Sales Prices/Lin...
Statistics
Process
Campaign Details
Report

CP1002 · Event

General

No.:	CP1002	Ending Date:	22-1-2011
Description:	Event	Salesperson Code:	BD
Status Code:	9-DONE	Last Date Modified:	
Starting Date:	19-1-2011	Activated:	No

Invoicing

Department Code:	SALES	Project Code:	

Each campaign has a unique **No.** field that can be created using number series and a **Description**. The **No.** field should be carefully chosen since that will be used throughout the application where this campaign is used.

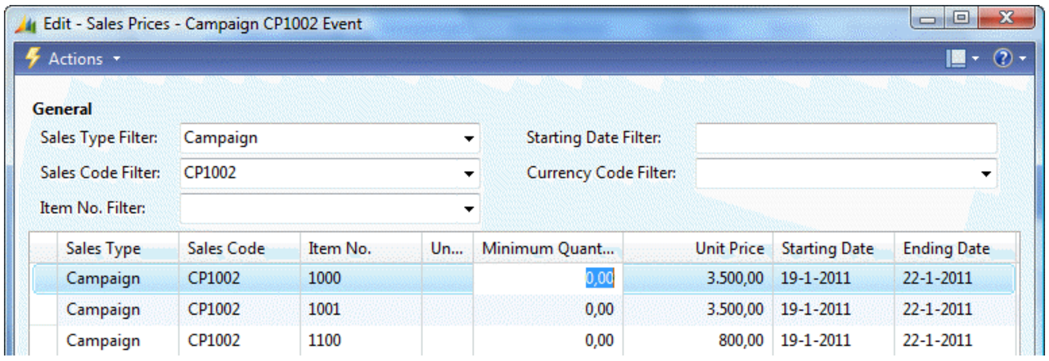
The **Status Code** options can be custom defined, but do not impact business logic.

The **Starting Date** and **Ending Date** are important for the pricing information. The special price and discounts will only be valid within these periods.

Via the **Invoicing** tab, we can see that the campaigns are integrated with dimensions. This gives us the powerful option to define a dimension code for each campaign and create an analysis view to analyze the results in the financial part of Microsoft Dynamics NAV, like we discussed in the previous chapter.

Pricing

By clicking on **Related Information** | **Campaign Sales Prices**, we can enter the pricing information for this campaign.



Edit - Sales Prices - Campaign CP1002 Event

Actions

General

Sales Type Filter:	Campaign	Starting Date Filter:	
Sales Code Filter:	CP1002	Currency Code Filter:	
Item No. Filter:			

Sales Type	Sales Code	Item No.	Un...	Minimum Quant...	Unit Price	Starting Date	Ending Date
Campaign	CP1002	1000		0,00	3.500,00	19-1-2011	22-1-2011
Campaign	CP1002	1001		0,00	3.500,00	19-1-2011	22-1-2011
Campaign	CP1002	1100		0,00	800,00	19-1-2011	22-1-2011

This price table is filtered exactly the same way as discussed in our example application in *Chapter 2*.

Segments

To select customers or prospects for a campaign, we need to create one or more segments. These segments should be connected to the campaign using the **Campaign No.** field. Everyone related to these segments will get the specific prices and discounts.

The segments are also used to create the interaction log entries and To-do's for this campaign. We need to make our target group aware that this campaign exists by sending them a letter, fax, e-mail, or even a phone call.

Activate

By activating the campaign, the system will add all contacts to the campaign group and create interaction log entries.

The interaction log entries will be used to calculate the cost of a campaign. Each interaction has a specific cost and all costs add up to the total amount on the campaign.

When an opportunity comes in, we can point this to a specific campaign. The value of this opportunity is also used in estimating the success of the campaign.

The campaign is also copied into the sales documents using the dimensions attached to the campaign. This allows us to further analyze the results.

Outlook integration

Salespeople are often on the road without online access to the ERP system, and Microsoft Dynamics NAV does not have an offline mode. To solve this problem, Microsoft Dynamics NAV is integrated with Microsoft Outlook.

This allows salespeople to view contacts and tasks offline and replicate with the back office system when possible.

If salespeople use a Windows Mobile phone with Microsoft Outlook they can even have all their Microsoft Dynamics NAV information on their device.

Using User Defined Views will also enable us to synchronize other Microsoft Dynamics NAV data to Microsoft Outlook, for example, the customer table with the current value of the balance field or the item table with the current inventory.

We will discuss the possibilities of interfacing with Microsoft Outlook in *Chapter 9, Interfacing*.

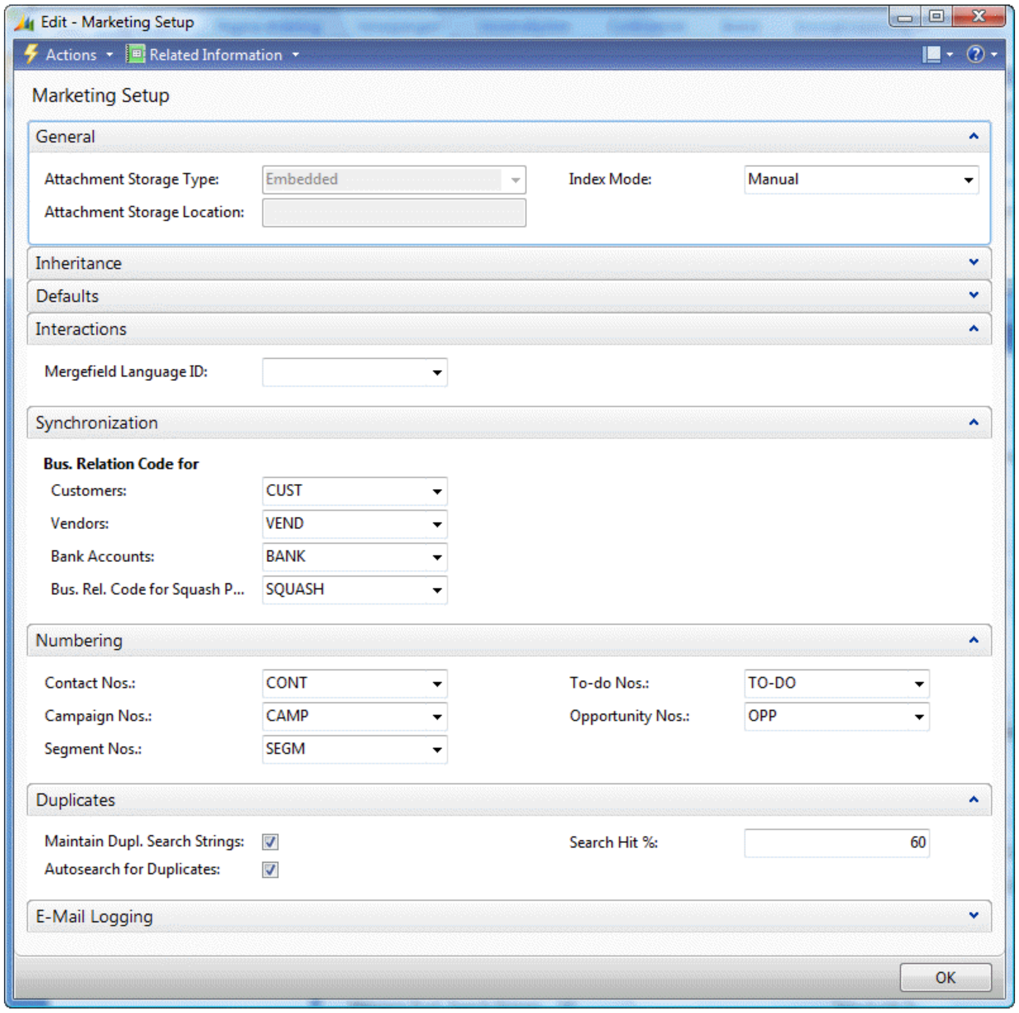
E-mail logging

Microsoft Dynamics NAV also has a capability to read exchange shared folders such as *info@* mailboxes.

For each e-mail the system can generate Interaction Log Entries and To-Do's.

The setup

Before implementing relationship management, we should properly set up the options. This can be done in the **Marketing Setup**.



Marketing Setup

General

Attachment Storage Type:	Embedded	Index Mode:	Manual
Attachment Storage Location:			

Inheritance

Defaults

Interactions

Mergefield Language ID:	

Synchronization

Bus. Relation Code for

Customers:	CUST
Vendors:	VEND
Bank Accounts:	BANK
Bus. Rel. Code for Squash P...	SQUASH

Numbering

Contact Nos.:	CONT	To-do Nos.:	TO-DO
Campaign Nos.:	CAMP	Opportunity Nos.:	OPP
Segment Nos.:	SEGM		

Duplicates

Maintain Dupl. Search Strings:	✓	Search Hit %:	60
Autosearch for Duplicates:	✓		

E-Mail Logging

- **Attachment Storage Type:** The attachments in the interaction log entries can either be stored in the database (embedded) or on the file system (Disk File). It is highly recommended to store them on the file system.
- **Attachment Storage Location:** If we choose to store the attachments on file system, this is where we specify the path.
- **Index Mode:** When using Contact Search, this should be set to **Auto**. It might have a small drawback on performance and can cause the database to be bigger.

- **Inheritance:** When entering a person, it can inherit the salesperson code, territory code, country/region code, language code, address details, and communication details from the company it belongs to.
- **Defaults:** A new contact can get a default salesperson code, territory code, country/region code, language code, or correspondence type. There is a different default salutation code for companies and persons.
- **Default Sales Cycle Code:** Every new opportunity will automatically get this code.
- **Mergefield Language ID:** This defines if the Word merge fields are in local language or in English.
- **Synchronization:** Here we enter the default **Business Relation Code** for **Customers**, **Vendors**, and **Bank Accounts**.
- **Maintain Dupl. Search strings:** Check this field if the duplicate contact functions are used.
- **Autosearch for Duplicates:** Use this option if the system should automatically search when entering new contacts.
- **Search Hit %:** This determines the percentage of match lines that the duplicate search string setup should have to qualify as a duplicate contact.

Customizing relationship management

RM is a pretty complete module that is not often highly customized or verticalised. However, we will describe some possible changes and how to integrate an add-on, in our case the Squash application, with Relationship Management.

All examples in this chapter are part of the objects downloaded for *Chapter 2*, *A Sample Application*.

Salutation formula types

By default, the system has two salutation formula types— formal and informal, allowing us to print Dear Mrs. Brown, or Dear Angela. But, what if we want to print Attn. Mrs. Brown?

For this, we need to first add an option to the **Salutation Type** field in the **Salutation Formula** table.

Add the option

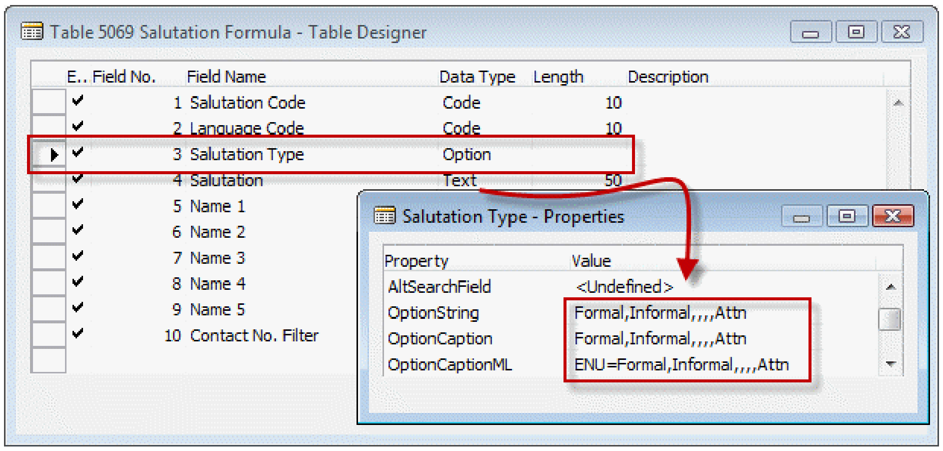


Table 5069 Salutation Formula - Table Designer

E..	Field No.	Field Name	Data Type	Length	Description
✓	1	Salutation Code	Code	10	
✓	2	Language Code	Code	10	
✓	3	Salutation Type	Option		
✓	4	Salutation	Text	50	
✓	5	Name 1			
✓	6	Name 2			
✓	7	Name 3			
✓	8	Name 4			
✓	9	Name 5			
✓	10	Contact No. Filter			

Salutation Type - Properties

Property	Value
AltSearchField	<Undefined>
OptionString	Formal,Informal,,,,Attn
OptionCaption	Formal,Informal,,,,Attn
OptionCaptionML	ENU=Formal,Informal,,,,Attn

Support the formula

Next, we want to use the formula when printing a Contact Cover Sheet. This uses the **Format Address** functionality from Codeunit 365.

This Codeunit is the single point in Dynamics NAV where all the address formatting is done.

The formatting of contact persons is done in the function `ContactAddrAlt`. We should make the following change.

```
ContactAddrAlt()
...
  ContIdenticalAddress:
    WITH ContAltAddr DO BEGIN
      GET(Cont."Company No.",CompanyAltAddressCode);
      FormatAddr(
        AddrArray,"Company Name","Company Name 2",
        Cont.Name,Address,"Address 2",
        City,"Post Code",County,"Country/Region Code");
    END;
  (Cont.Type=Cont.Type::Person) AND
  (Cont."Company No." <> ''):
    WITH Cont DO
      FormatAddr(
//        AddrArray,ContCompany.Name,ContCompany."Name 2",
//        Name,Address,"Address 2",
        AddrArray,ContCompany.Name,ContCompany."Name 2",
        GetSalutation(5, Cont."Language Code"),Address,
        "Address 2",City,"Post Code",County,
        "Country/Region Code")
```

Always comment out the original line of code before you make a change. This will enable you to always go back to standard code and help when upgrading this solution to a newer version.

Most NAV partners and developers have their own way of documenting and commenting. The example in here is the *Minimum* comment requirement. We will discuss versioning of objects in *Chapter 10, Application Design*.

The GetSalutation function

In our modification, we use the `GetSalutation` function in the `Contact table (5050)` instead of the **Name** field. Let's have a look at that function and analyze what it does.

```
GetSalutation()
IF NOT SalutationFormula.GET("Salutation Code",LanguageCode,
  SalutationType)
THEN
  ERROR(Text021,LanguageCode,"No.");

SalutationFormula.TESTFIELD(Salutation);

CASE SalutationFormula."Name 1" OF
  SalutationFormula."Name 1"::"Job Title":
    NamePart[1] := "Job Title";
  SalutationFormula."Name 1"::"First Name":
    NamePart[1] := "First Name";
  SalutationFormula."Name 1"::"Middle Name":
    NamePart[1] := "Middle Name";
  SalutationFormula."Name 1"::Surname:
    NamePart[1] := Surname;
  SalutationFormula."Name 1"::Initials:
    NamePart[1] := Initials;
  SalutationFormula."Name 1"::"Company Name":
    NamePart[1] := "Company Name";
END;

CASE SalutationFormula."Name 2" OF
  ...
END;
```

```
...
FOR i := 1 TO 5 DO
  IF NamePart[i] = '' THEN BEGIN
    SubStr := '%' + FORMAT(i) + ' ';
    IF STRPOS(SalutationFormula.Salutation,SubStr) > 0 THEN
      SalutationFormula.Salutation :=
        DELSTR(SalutationFormula.Salutation,STRPOS(SalutationFormula.
Salutation,SubStr),3);
  END;

EXIT(STRSUBSTNO(SalutationFormula.Salutation,NamePart[1],
  NamePart[2],NamePart[3],NamePart[4],NamePart[5]))
```

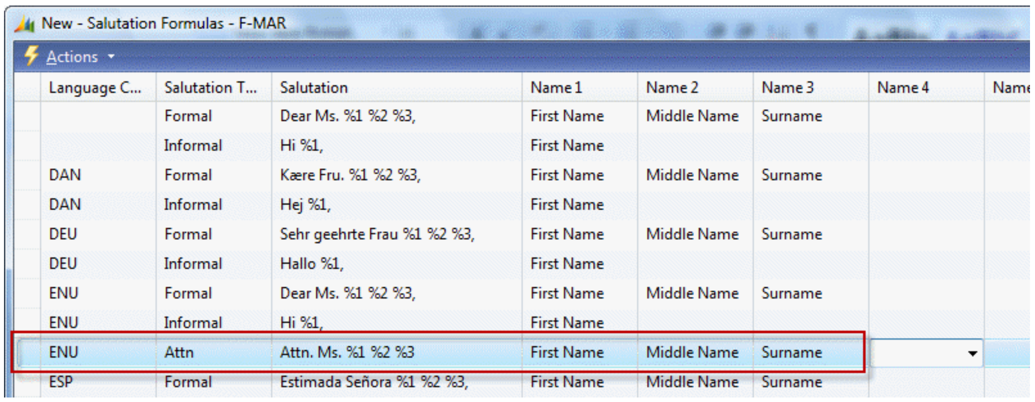
The function uses two parameters: `SalutationType` and `LanguageCode`. With these values and the salutation code of the contact, it checks if there is a valid formula. Since we only added a new option, the code still works because at database level, the **Option** field is translated to an **Integer**.

For documentation purposes, we could also implement the new option value in this function. The downside of that would be that we do a modification that is not technically necessary, but needs to be maintained and upgraded.

Depending on the order of the formula, the necessary name fields are combined and used as the return value of the function.

Set up the salutation formula

If we want to use our new Salutation formula, we need to set it up first. We will do this for F-MAR to test it with CT100191 Megan Sherman from American Wood Exports.



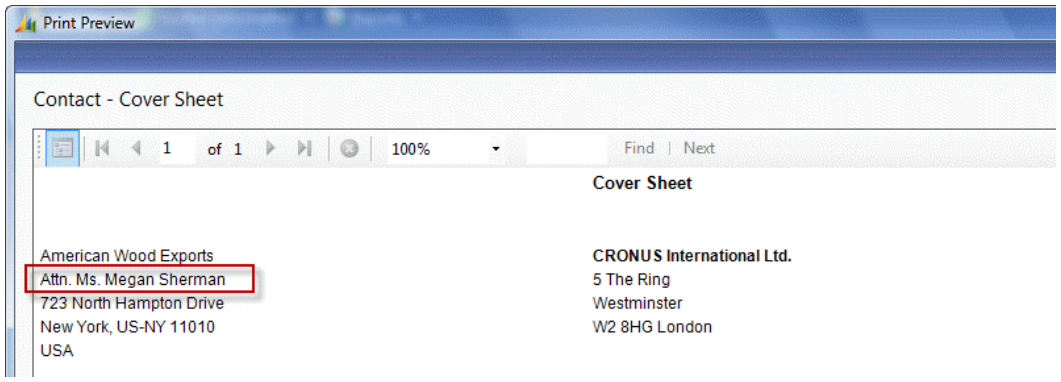
New - Salutation Formulas - F-MAR

Actions

Language C...	Salutation T...	Salutation	Name 1	Name 2	Name 3	Name 4	Name
	Formal	Dear Ms. %1 %2 %3,	First Name	Middle Name	Surname		
	Informal	Hi %1,	First Name				
DAN	Formal	Kære Fru. %1 %2 %3,	First Name	Middle Name	Surname		
DAN	Informal	Hej %1,	First Name				
DEU	Formal	Sehr geehrte Frau %1 %2 %3,	First Name	Middle Name	Surname		
DEU	Informal	Hallo %1,	First Name				
ENU	Formal	Dear Ms. %1 %2 %3,	First Name	Middle Name	Surname		
ENU	Informal	Hi %1,	First Name				
ENU	Attn	Attn. Ms. %1 %2 %3	First Name	Middle Name	Surname		
ESP	Formal	Estimada Señora %1 %2 %3,	First Name	Middle Name	Surname		

Test the solution

After adding the new formula, we print a cover sheet from the **Contact Card** using the **Contact Cover Sheet** option from the **Report** actions. The result will look like this:



Cover Sheet

American Wood Exports
Attn. Ms. Megan Sherman
723 North Hampton Drive
New York, US-NY 11010
USA

CRONUS International Ltd.
5 The Ring
Westminster
W2 8HG London

Customer and vendor numbering

Another common requirement from end users is to maintain the same number when creating a customer or vendor from a contact.

This can be done by adding one line of code to the `CreateCustomer` function in the **Contact** table.

```
CreateCustomer()
...
CLEAR(Cust);
Cust.SetInsertFromContact(TRUE);
//* Maintain Contact No. >>>
Cust."No." := "No.";
//* Maintain Contact No. <<<
Cust.INSERT(TRUE);
Cust.SetInsertFromContact(FALSE);
```

This works, because by populating the **No.** field the number series functionality in the `OnInsert` trigger does not start.

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

Disabling direct creation of customers and vendors

When using this option, it should be disabled to directly create a customer or vendor. This can be done easily by removing the No. series from the Sales & Receivables setup and Purchases & Payables setup. This results in a runtime error message when creating the customer or vendor.

Sharing contact information across companies

When more companies have their administration in Microsoft Dynamics NAV, they most often have the same owner or group of owners that want their Contact data to span across their companies.

This can be achieved by sharing some tables over all the companies and changing some business logic.

Share tables

By default, Microsoft Dynamics NAV will create a separate instance of each table for each company. This can be changed with the `DataPerCompany` property in the **Table Designer**.

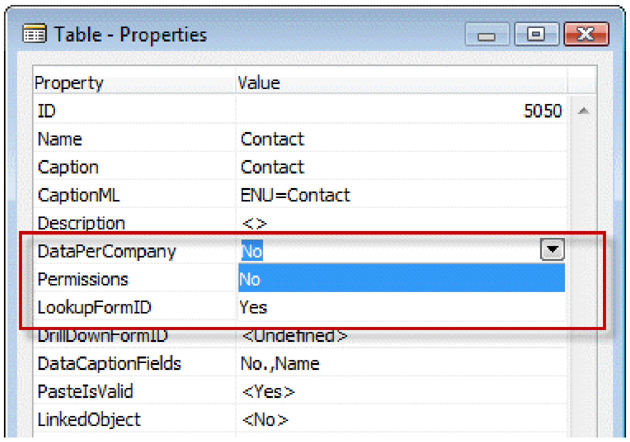


Table - Properties

Property	Value
ID	5050
Name	Contact
Caption	Contact
CaptionML	ENU=Contact
Description	<>
DataPerCompany	No
Permissions	No
LookupFormID	Yes
DrillDownFormID	<Undefined>
DataCaptionFields	No.,Name
PasteIsValid	<Yes>
LinkedObject	<No>

The following tables should be shared across the database since they contain the main contact information and the link to the customer and vendor data.

- 5050 - Contact
- 5051 - Contact Alt. Address

- 5052 - Contact Alt. Addr. Date Range
- 5053 - Business Relation
- 5054 - Contact Business Relation

This will allow us to reuse contact data in all companies. Sharing other tables is optional, but sharing them might be useful in some cases.

By sharing the **Contact Profile Answer** table other companies can see how a customer is doing within the group.

The **Segment** tables could be shared in order to cross company slice and dice information. This also requires the **Criteria** tables to be shared.

[When you share the profile or segment tables, the reports that calculate them should be started for each company individually in the database.]

Campaigns and **Opportunities** should not be shared since that interfaces with the ERP system. Never share financial tables like the value entry or document tables.

Interaction log entries could be shared but we should realize that most table relations to sales and purchase documents will not work when we are in the wrong company.

Business relations

When sharing contacts across companies, we are interested to see in which company the contacts are customers and vendors. We also want to maintain those tables when the contact information changes.

This means that besides sharing the Contact Business Relation table, we should also add a field indicating the company and add this field to the primary key.

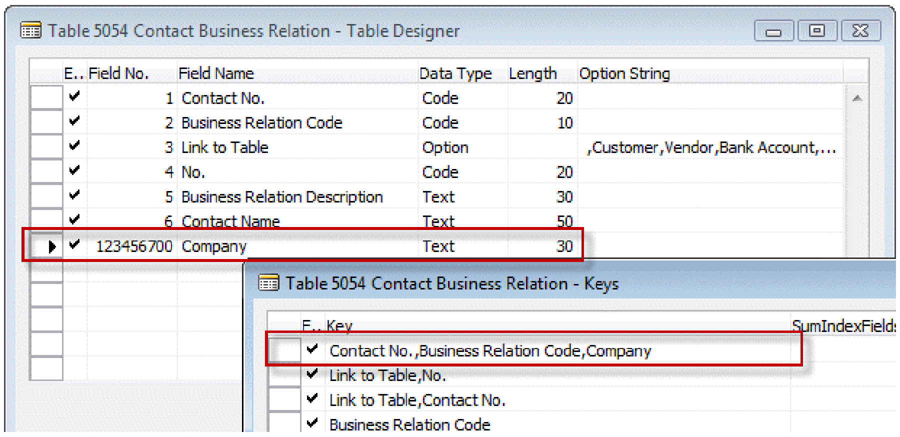


Table 5054 Contact Business Relation - Table Designer

E..	Field No.	Field Name	Data Type	Length	Option String
✓	1	Contact No.	Code	20	
✓	2	Business Relation Code	Code	10	
✓	3	Link to Table	Option		,Customer,Vendor,Bank Account,...
✓	4	No.	Code	20	
✓	5	Business Relation Description	Text	30	
✓	6	Contact Name	Text	50	
✓	123456700	Company	Text	30	

Table 5054 Contact Business Relation - Keys

E..	Key	SumIndexFields
✓	Contact No.,Business Relation Code,Company	
✓	Link to Table,No.	
✓	Link to Table,Contact No.	
✓	Business Relation Code	

C/AL code modifications

The functions that created the customer and vendor records that we saw in chapter 2, should also be checked.

For example, the function `CreateCustomer` in the **Contact** table.

```
CreateCustomer()
...

ContBusRel.RESET;
ContBusRel.SETRANGE("Contact No.","No.");
ContBusRel.SETRANGE("Link to Table",ContBusRel."Link to
Table"::Customer);
//* Company Sharing >>>
ContBusRel.SETRANGE(Company, COMPANYNAME);
//* Company Sharing <<<
IF ContBusRel.FIND('-') THEN
  ERROR(
    Text019,

...
```

And a little bit further up in the C/AL code:

```
ContBusRel."Contact No." := ContComp."No.";
ContBusRel."Business Relation Code" := RMSetup."Bus. Rel. Code for
Customers";
ContBusRel."Link to Table" := ContBusRel."Link to Table"::Customer;
//* Company Sharing >>>
ContBusRel.Company := COMPANYNAME;
```

```
//* Company Sharing <<<
ContBusRel."No." := Cust."No.";
ContBusRel.INSERT(TRUE);
```

We should also check the code that maintains data integrity which is the `CustVendBank-Update Codeunit (5055)` that we discussed in chapter 2.

```
UpdateCustomer()
WITH Cust DO BEGIN
//* Company Sharing >>>
  CHANGECOMPANY(COMPANYNAME);
//* Company Sharing <<<
  GET(ContBusRel."No.");
  ...
END;
```

Here we use the `CHANGECOMPANY` C/AL command to change the company for a specific instance of a variable.

There are more functions impacted such as the `UpdateQuotes` function in the Contact table. Analyze your database before implementing this feature.

Number series

The last change we should do for a properly working system is create a new instance of the number series functionality.

This can be achieved relatively easily since the number series are an isolated set of objects.

In the Object Designer, we should filter on this set of objects and export them to a `.txt` file.

- Table (308) : No. Series
- Table (309) : No. Series Line
- Table (310) : No. Series Relationship
- Report (21) : No. Series
- Report (22) : No. Series Check
- Codeunit (396) : NoSeriesManagement
- Page (456) : No. Series
- Page (457) : No. Series Lines
- Page (458) : No. Series Relationships

In this file, we can renumber them and rename them so we get something like this:

- Table (123456721) : No. Series (Shared)
- Table (123456722) : No. Series Line (Shared)
- Table (123456723) : No. Series Rel. (Shared)
- Report (123456721) : No. Series (Shared)
- Report (123456722) : No. Series Check (Shared)
- Codeunit (123456721) : NoSeriesManagement (Shared)
- Page (123456721) : No. Series (Shared)
- Page (123456722) : No. Series Lines (Shared)
- Page (123456723) : No. Series Rel. (Shared)

Where the tables should be `DataPerCompany` No.

Final steps

When we have shared number series functionality, we can implement this in the existing objects.

1. The field Contact Nos. in the marketing setup table should change the table relation to the shared No. series table as well as the No. series field in the contact table.
2. The variable `NoSeriesMgt` in the contact table should move from `NoSeriesManagement` to `SharedNoSeriesMgt`.

Alternative approaches

Sharing contact information across companies is a change that has been implemented by many companies and can be considered safe. Other tables in Microsoft Dynamics NAV are more difficult to share because of financial or operational information.

A typical example in the standard application is the `Item table (27)`. This contains a field Cost is Adjusted (29) that is used when running Cost Adjustment. If this table would be shared across all companies we would create a major issue with running this function.

[We will discuss Cost Adjustment in *Chapter 5, Production*.]

For this issue there are two commonly implemented solutions:

- **Shared Master Items**: We can create a new table called Master Item. This table is shared across all companies and contains the information we share like descriptions and pricing. When the data in this table is changed, it should enable a mechanism comparable to the `CustVendBank-Update Codeunit (5055)` that updates the Items in the other companies using the `CHANGECOMPANY` C/AL function.
- External Synchronization: We could implement something that will export the changes done in company A to an XML file. An application server can run in the background and read this XML file and implement these changes to other companies in the database or even other databases.

The first solution with Master Items looks a lot like the way Contacts work in the standard application and is a perfect example of Look, Learn, and Love using proven data structures in customized solutions.

Add contacts to segments

The last change we are implementing in Relationship Management is adding a table to the Add Contacts functionality in Segments.

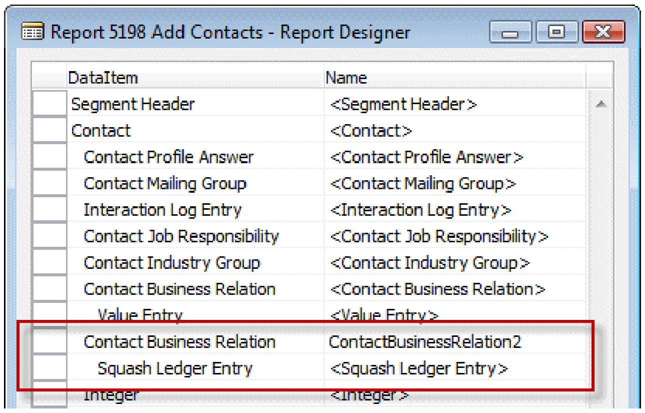
We have seen that it is already complete but a vertical solution might want to integrate its ledger entry tables here.

For this example, we will make it possible to filter in the Squash Ledger Entries from the example application in Chapter 2.

Expand report

The first step is to add the Squash Ledger Entries as a `DataItem` to the Add Contacts report (5198). We will copy the functionality from the Value Entries as this is comparable functionality.

[Always find comparable standard application functionality to learn from. Never just copy and paste this but learn how it's done and apply your own knowledge.]



Report 5198 Add Contacts - Report Designer

DataItem	Name
Segment Header	<Segment Header>
Contact	<Contact>
Contact Profile Answer	<Contact Profile Answer>
Contact Mailing Group	<Contact Mailing Group>
Interaction Log Entry	<Interaction Log Entry>
Contact Job Responsibility	<Contact Job Responsibility>
Contact Industry Group	<Contact Industry Group>
Contact Business Relation	<Contact Business Relation>
Value Entry	<Value Entry>
Contact Business Relation	ContactBusinessRelation2
Squash Ledger Entry	<Squash Ledger Entry>
Integer	<Integer>

We cannot copy and paste the table relation from the other Contact Business Relation DataItem since squash players are contact persons, not companies.

Our table relation should be `Contact No.=FIELD(No.)`.

The code in our Contact Business Relation table tells us we need two new variables of type Boolean— `SquashFilters` and `SkipSquashLedgerEntry`.

```
ContactBusinessRelation2 - OnPreDataItem()
IF ContactOK AND ((GETFILTERS<>'') OR SquashFilters) THEN
  ContactOK := FALSE
ELSE
  CurrReport.BREAK;

ContactBusinessRelation2 - OnAfterGetRecord()
SkipSquashLedgerEntry := FALSE;
IF NOT SquashFilters THEN BEGIN
  ContactOK := TRUE;
  SkipSquashLedgerEntry := TRUE;
  CurrReport.BREAK;
END;
```

The `SquashFilters` is determined in the `OnPreReport` trigger.

```
Report - OnPreReport()
ItemFilters := "Value Entry".HASFILTER;

//* Squash >>>
SquashFilters := "Squash Ledger Entry".HASFILTER;
//* Squash <<<
...
```

The code in the Squash Ledger Entry `DataItem` should look like this:

```
Squash Ledger Entry - OnPreDataItem()
IF SkipSquashLedgerEntry THEN
  CurrReport.BREAK;

CASE ContactBusinessRelation2."Link to Table" OF
  ContactBusinessRelation2."Link to Table"::"Squash Player":
  BEGIN
    SETRANGE("Squash Player No.",
      ContactBusinessRelation2."No.");
  END;
  ELSE
    CurrReport.BREAK;
END;

Squash Ledger Entry - OnAfterGetRecord()
ContactOK := TRUE;

IF ContactOK THEN
  CurrReport.BREAK;
```

Make sure we filter on our instance of Contact Business Relation and that we filter on Link to Table Squash player.

The `ContactOK` indicates that all contact persons connected to this Squash Ledger Entry will be inserted.

Implement criteria filters

To support the criteria filter functionality we need to make two changes. One to the **Add Contacts** report and one to Codeunit `SegCriteriaManagement`.

In the add contacts report, we add this C/AL code to the `OnPreReport` trigger. This will make a call to the `SegCriteriaManagement Codeunit (5062)`.

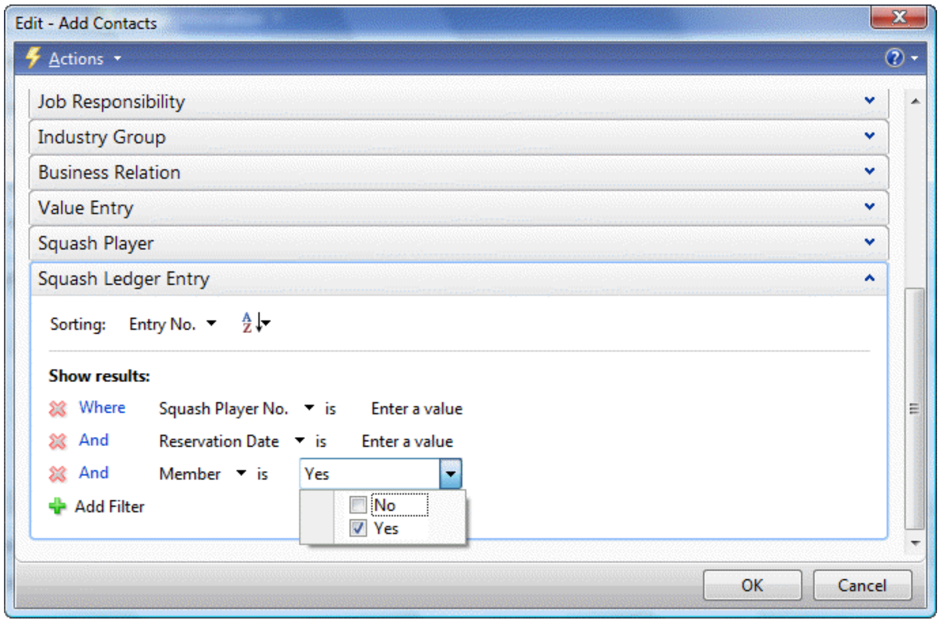
```
OnPreReport()
...
SegCriteriaManagement.InsertCriteriaFilter(
  "Segment Header".GETFILTER("No."),DATABASE::"Value Entry",
  "Value Entry".GETFILTERS,"Value Entry".GETVIEW(FALSE));
//* Squash >>>
SegCriteriaManagement.InsertCriteriaFilter(
  "Segment Header".GETFILTER("No."),
    DATABASE::"Squash Ledger Entry",
    "Squash Ledger Entry".GETFILTERS,
    "Squash Ledger Entry".GETVIEW(FALSE));
//* Squash <<<
```

In the `SegCriteriaManagement` Codeunit we add this code to the `SegCriteriaFilter` function which will require a new local variable for `Squash Ledger Entry`.

```
SegCriteriaFilter()
...

CASE TableNo OF
   ...
//* Squash Ledger Entry >>>
  DATABASE::"Squash Ledger Entry":
    BEGIN
      SquashLedgEntry.SETVIEW(View);
      EXIT(SquashLedgEntry.GETFILTERS);
    END;
//* Squash Ledger Entry <<<
END;
```

Test solution

Now we can test the solution by trying to add all **Squash Players** of type **Member** to a **Segment**.



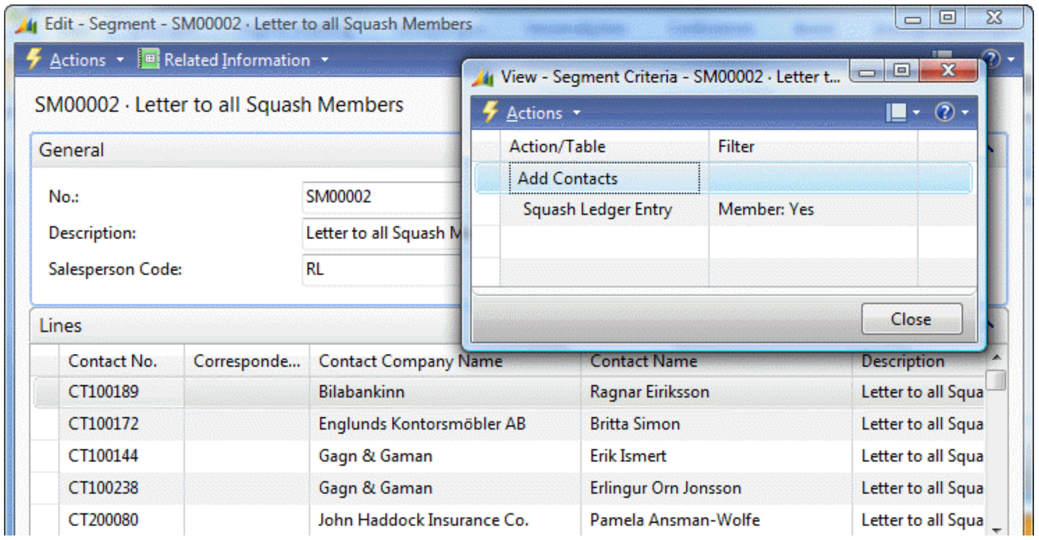
Sorting: Entry No.

Show results:

Where Squash Player No. is Enter a value
And Reservation Date is Enter a value
And Member is Yes
Add Filter

No
✓ Yes

The result is a segment with the required squash players:


Edit - Segment - SM00002 · Letter to all Squash Members
Actions
Related Information
View - Segment Criteria - SM00002 · Letter t...
Close

SM00002 · Letter to all Squash Members

General

No.:	SM00002
Description:	Letter to all Squash M
Salesperson Code:	RL

Action/Table	Filter
Add Contacts	
Squash Ledger Entry	Member: Yes

Lines

Contact No.	Corresponde...	Contact Company Name	Contact Name	Description
CT100189		Bilabankinn	Ragnar Eiriksson	Letter to all Squa
CT100172		Englunds Kontorsmöbler AB	Britta Simon	Letter to all Squa
CT100144		Gagn & Gaman	Erik Ismert	Letter to all Squa
CT100238		Gagn & Gaman	Erlingur Orn Jonsson	Letter to all Squa
CT200080		John Haddock Insurance Co.	Pamela Ansman-Wolfe	Letter to all Squa

This change also needs to be implemented to the reduce/refine functionality which works similar to the add contacts report.

Summary

In this chapter, we took a deep dive into the Microsoft Dynamics NAV Relationship Management functionality. We learned how it is integrated with the ERP part of the system.

Relationship management can be very useful to analyze sales data. With profiles, we can filter on turnover and profit figures and use them in segments.

Interaction Log Entries allow us to keep track of all the contact moments with the people we do business with.

Outlook integration can be used for sales people to work remotely and synchronize with the system.

Campaigns and opportunities help us to keep track of the quote process and improve our sales working and make it more efficient.

Lastly, we looked at some common requirements for changing the relationship management system to our company's specific requirements.

In the next chapters, we will look at the ERP part of Microsoft Dynamics NAV starting with the Production process in Chapter 5 and Trade in Chapter 6.

5 Production

The previous chapters introduced the key concepts of Microsoft Dynamics NAV as well as the details of the financial application and CRM. These horizontal modules can be implemented in most industries without big structural changes.

In this chapter, we will discuss three ways of implementing production in Microsoft Dynamics NAV using standard functionality and customized features.

We'll discuss Item Tracking and Item Costing and what procedures and objects are used to get this working correctly in the application.

For Manufacturing, we will discuss the general concepts and data model rather than go into details of each and every functional possibility.

We will also discuss Kitting which is only available in a limited number of countries such as North America, France, and Australia but will most likely be moved to the worldwide version in future versions.

At the end of this chapter, we will look at the five different vertical industries and highlight two specific features of these industries that are not implemented in the standard product and discuss how the problems could be solved.

After reading this chapter, you will have a better understanding about the concepts of production in Microsoft Dynamics NAV, how this fits together with the rest of the application and when to think out of the box if it does not immediately fit your process.

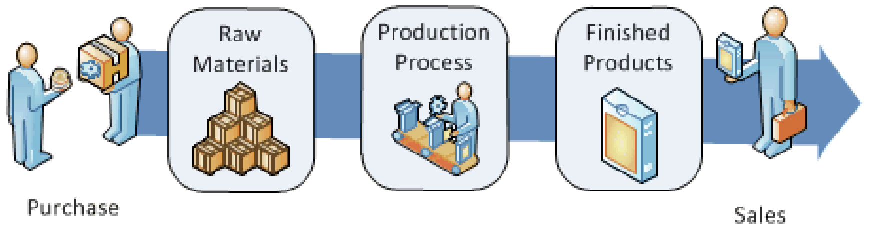
What is production?

Production is creating a new product using raw materials or prefabricated items and resources.

History of production

Production as we know it today started centuries before the industrial revolution with craftsmen and assistants creating products using raw materials produced by nature and farmers. Today this method of production still exists for many luxury items such as custom made furniture or clothes.

The industrial revolution changed production into manufacturing with the introduction of machines and mass production. This allowed production to grow because it was less dependent on craftsmen and manual labor.



The introduction of computers in manufacturing companies allows the production of more sophisticated items and has made manufacturing more flexible.

Before implementing Microsoft Dynamics NAV for production companies, it is crucial to understand which level of production is being used in your company.

Production methodologies

In this chapter, we will differentiate three production methodologies:

- Assembling production: When production is combining items into one new item without changing the items or scrap, we will refer to it as assembling.
- Manufactured production: This is the most complex production method to be implemented in Microsoft Dynamics NAV. Raw materials are combined into one or more products leaving scrap.
- Specialized production: These are often one off items or items produced in small numbers. The system should support the basics of the production process but should still be flexible enough to fit the company.

Raw materials

Each product we consume starts out as raw materials such as cotton, iron ore, logs and oil that is then processed to be used in a production process. Other raw materials include water and air or fruits and vegetables. All raw materials are produced by Mother Nature. Some production of raw materials can be influenced by humans such as logs, cotton, fruits, and vegetables. Other raw materials are more limited, such as iron ore, oil, and water.

Basic production principles

Before going into Microsoft Dynamics NAV, we will discuss some terminology that is important to understand as a basis for the concepts of production in ERP.

Bill of materials

The bill of materials defines what components are used to assemble or manufacture one item. The components in the Bill of Materials are also items, so before creating a new Bill of Materials all component items must be created in the system.

In Microsoft Dynamics NAV there are two separate Bill of Material definitions. One is for assembling and the other is for manufacturing.

MRP

Material requirements planning was introduced in the 1960's as a calculation method for production scheduling and was quickly replaced by Manufacturing resource planning or MRP II.

While ERP replaces MRP, MRP is still a crucial part of ERP Applications.

Microsoft Dynamics NAV has a built-in MRP algorithm but also allows developers to create their own algorithms using the built in data model.

MRP analyzes dependant demand, that is, demand that comes from production orders for components.

GIGO

The biggest risk in running MRP algorithms is the Garbage In Garbage Out principle. To create a good plan, the data in the system must be absolutely correct or the planning will contain errors.

If, for example, the shipment dates in the Sales Orders are not entered correctly, the planning algorithm has no chance of giving correct results. The garbage in (wrong dates) will result in garbage out (wrong planning).

MPS

Master Production Schedule (**MPS**) is the term used for the production planning and scheduling. A MPS is used for decision making linking supply and demand. It analyzes independent demand, that is, demand that comes from Sales Orders, Service Orders, and the Production Forecast.

Item costing

For manufacturing companies, it is crucial to be able to calculate real item costing and profitability. The cost of an item consists of all the components it was created from as well as the production time and the cost of any machinery used.

In production companies, high costs are incurred before an item can even be manufactured and sold. Machines need to be purchased and installed and new manufacturing plants may need to be built.

Item tracking

Item tracking is a relatively new concept that was introduced by the need to be able to trace back an item to its original production batch in the supply chain. Whenever something is wrong with a specific item it can be interesting to see if other items that were produced in the same batch have the same issue. A recall of all items in that batch may be required.

Quality control

During the production process, and especially at the end, quality control is a crucial stage. Items can be rejected completely or may require extra handling.

In quality control, items are checked for mistakes. The way this is done depends on the production process. In the automobile industry, all cars are checked individually while in the chemical industry parts of batches are taken out and checked assuming that the rest of the batch has the same quality.

Quality control is always at the end of the production process but can also be in between each of the main production processes.

Sometimes, the item that is manufactured depends on the result of quality control. In this case, each level of quality is represented by a special item number.

Energy and waste

When manufacturing an item, the obvious components are the items in the bill of materials. Using less energy and producing less waste material is becoming more crucial in this process as our environment is becoming more and more vulnerable and recycling is becoming increasingly important, these components have a bigger pressure on production cost and planning.

APICS

To learn more about Production it is interesting to study the materials provided by APICS, The Association for Operations Management. APICS is the organization that is recognized worldwide as the leading authority on manufacturing standards, similar to how the W3C is considered the authority on XML standards.

> More information about APICS can be found on `http://www.apics.org/`.

Getting started

Let's walk through two scripts to generate a new item with a Bill of Materials. One for an assembling process, the other for Manufacturing.

We will set up both Item Costing and Item Tracking for these products.

> The examples are created using a CRONUS W1 Microsoft Dynamics NAV 2009 Service Pack 1 database without changes.

Assembling

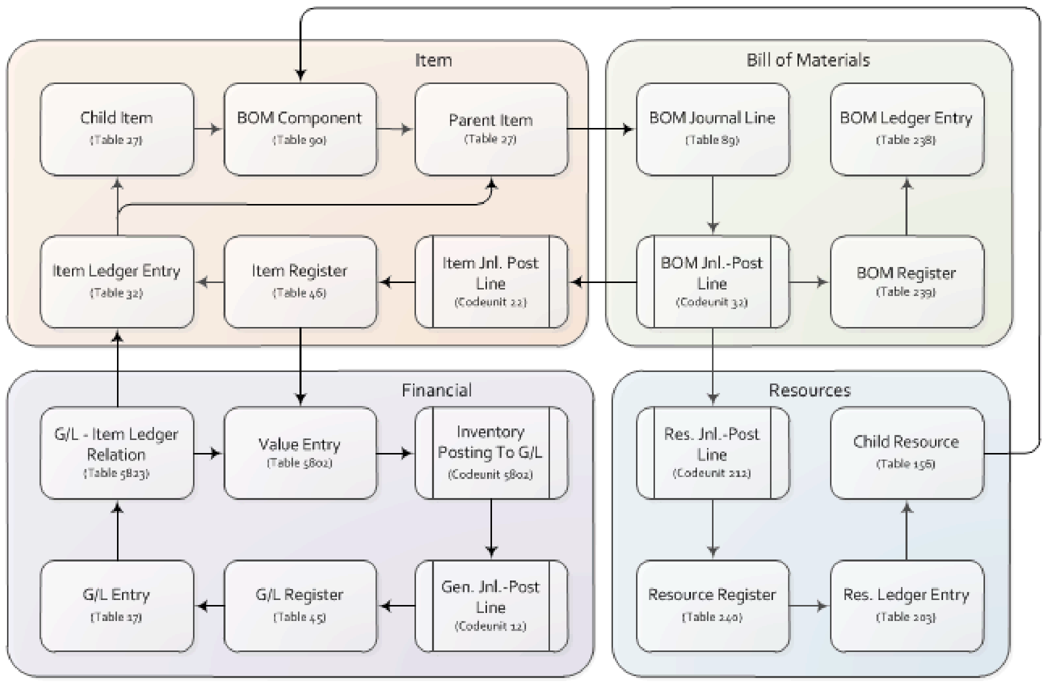
In our company, we want to start producing office chairs. These chairs consist of five wheels, a pedestal, a seat, and two arm rests. We will create these four components as a new item and one new item for the end product.

All the items will have a different costing method to demonstrate the effect of cost changes. The end product will support serial number item tracking with a one year warranty period.

When using the BOM Journal it is not possible to use Item Tracking for the components, only for the end product.

The table and posting schema

Before going into the application, we will have a look at how this process is solved in Microsoft Dynamics NAV.



Item
Child Item (Table 27)
BOM Component (Table 90)
Parent Item (Table 27)
Item Ledger Entry (Table 32)
Item Register (Table 46)
Item Jnl. Post Line (Codeunit 22)
Bill of Materials
BOM Journal Line (Table 89)
BOM Ledger Entry (Table 238)
BOM Jnl.-Post Line (Codeunit 32)
BOM Register (Table 239)
Financial
G/L - Item Ledger Relation (Table 5823)
Value Entry (Table 5802)
Inventory Posting To G/L (Codeunit 5802)
G/L Entry (Table 17)
G/L Register (Table 45)
Gen. Jnl.-Post Line (Codeunit 12)
Resources
Res. Jnl.-Post Line (Codeunit 212)
Child Resource (Table 156)
Resource Register (Table 240)
Res. Ledger Entry (Table 203)

It starts with creating the components and end products as items in the database and connecting them using the BOM Component table. A BOM Component can also contain **Resources**.

If the components are in stock we can use the BOM Journal to create the products. When posting the BOM Journal the components are consumed and the product is created.

During this process, the system will create BOM Ledger Entries, Resource Ledger Entries, Item Ledger Entries, and Value Entries.

The Value Entries can be posted on the General Ledger using the Inventory Posting to G/L Routine either manual or real time. This completes the process.

The items

For this example, we will need to create five items, four components, and one end product.

We will assign an estimated Unit Cost to the components and a Unit Price to the end product.

No.	Description	Base UOM	Unit Cost	Unit Price	Costing
CHAPTER5-C1	Chapter 5 | Wheel	PCS	5		FIFO
CHAPTER5-C2	Chapter 5 | Pedestal	PCS	60		LIFO
CHAPTER5-C3	Chapter 5 | Seat	PCS	120		Average
CHAPTER5-C4	Chapter 5 | Arm Rest	PCS	35		Standard
CHAPTER5-P1	Chapter 5 | Office Chair	PCS		500	Specific

In a real implementation, we would never set up a bill of materials with so many different costing methods for each item. This is strictly for the purpose of explaining what each costing method does and that Microsoft Dynamics NAV is technically capable of dealing with this.

Item costing

We will assign a different costing method to each Item. Let's briefly discuss the available costing methods in Microsoft Dynamics NAV.

- **FIFO**: First in First out. The cost of the oldest Item Ledger Entry is used.
- **LIFO**: Last in First out. The cost of the newest Item Ledger Entry is used.

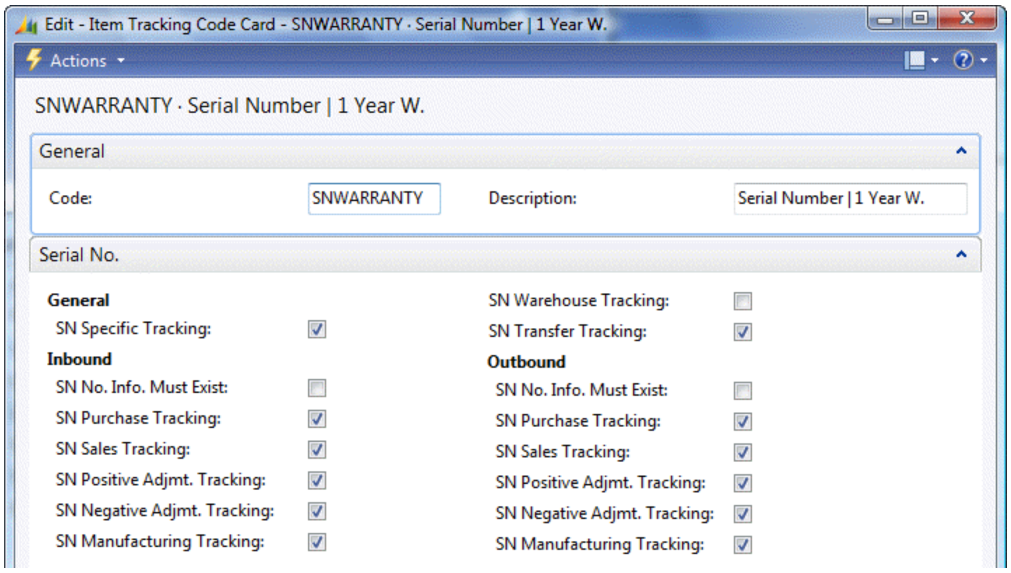
When using FIFO or LIFO, the cost is applied within the **Lot No.** if Item Tracking is used with Lot Numbering. That is, the cost associated with the specific Lot No. is used.

- **Average**: Each time we purchase items the total costs are divided by the total quantity. The result is used as Unit Cost.
- **Standard**: The user will define the Unit Cost manually. All deviations in purchase pricing are posted as profit or loss when invoicing.
- **Specific**: This is always combined with Item Tracking and Serial Numbers. Each Serial Number uses its own unique Unit Cost.

The Costing Methods are **NOT** related to the warehouse picking method but only apply to financial costing calculations.

Item tracking

All our chairs will get a serial number with a one year warranty period. This enables us to track all individual chairs when they come back to the factory with issues.


Edit - Item Tracking Code Card - SNWARRANTY · Serial Number | 1 Year W.
Actions

SNWARRANTY · Serial Number | 1 Year W.

General

Code:	SNWARRANTY	Description:	Serial Number | 1 Year W.

Serial No.

General		SN Warehouse Tracking:	☐
SN Specific Tracking:	✓	SN Transfer Tracking:	✓
Inbound		**Outbound**	
SN No. Info. Must Exist:	☐	SN No. Info. Must Exist:	☐
SN Purchase Tracking:	✓	SN Purchase Tracking:	✓
SN Sales Tracking:	✓	SN Sales Tracking:	✓
SN Positive Adjmt. Tracking:	✓	SN Positive Adjmt. Tracking:	✓
SN Negative Adjmt. Tracking:	✓	SN Negative Adjmt. Tracking:	✓
SN Manufacturing Tracking:	✓	SN Manufacturing Tracking:	✓

Item tracking in Microsoft Dynamics NAV can be done both on individual Serial Numbers and on Lot numbers for a group of items.

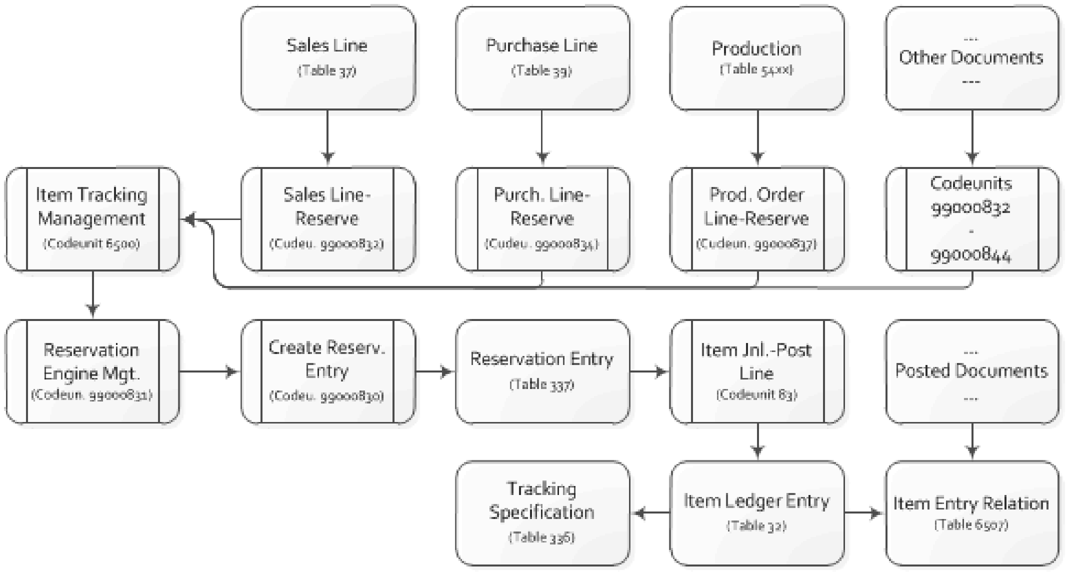
The Serial Numbers and Lot Numbers are fields in the Item Ledger Entry table (32). The consequence of this will be that for each serial number or lot number an individual record will be created in the table. When using serial numbers, this can lead to a massive increase in table size.

The lot numbers and item numbers are saved in the Reservation Entry table (337) during the document entry process. A reservation entry can be assigned to any table in Microsoft Dynamics NAV, for example, Sales Lines, Item Journal Lines, or Production Order.

Then a document is posted and the Item Ledger Entry is created. The Reservation Entry is removed and replaced by a Tracking Specification record that has the same value in the Entry No. field as the corresponding Item Ledger Entry.

A Reservation Entry used for Item Tracking should not be confused with normal Reservation Entries in the Sales and Purchase process.

The process of Item tracking in Microsoft Dynamics NAV works as shown in the following schema:


Sales Line (Table 37)
Purchase Line (Table 39)
Production (Table 54xx)
--- Other Documents ---
Item Tracking Management (Codeunit 6500)
Sales Line-Reserve (Cudeu. 99000832)
Purch. Line-Reserve (Cudeu. 99000834)
Prod. Order Line-Reserve (Cudeun. 99000837)
Codeunits 99000832 - 99000844
Reservation Engine Mgt. (Codeun. 99000831)
Create Reserv. Entry (Codeu. 99000830)
Reservation Entry (Table 337)
Item Jnl.-Post Line (Codeunit 83)
--- Posted Documents ---
Tracking Specification (Table 336)
Item Ledger Entry (Table 32)
Item Entry Relation (Table 6507)

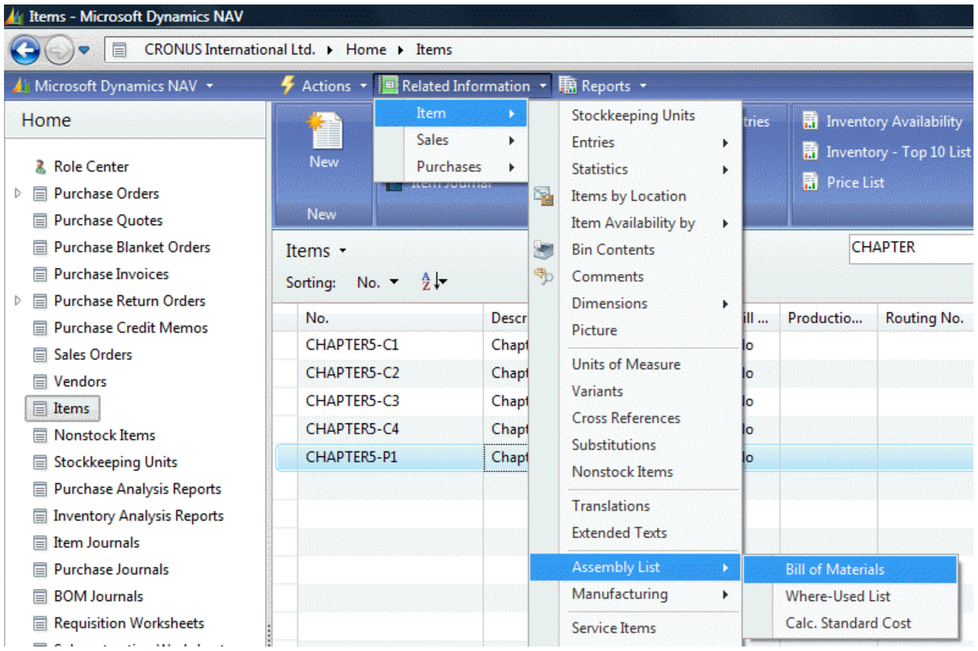
Technically, Item Tracking in Microsoft Dynamics NAV is very complex and should only be changed by experienced developers after careful analysis.

We will discuss the Reservation process in more detail in *Chapter 6, Trade*.

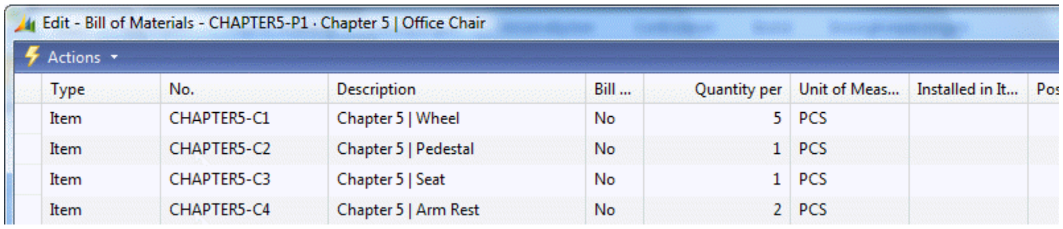
The bill of materials

When the items are created, the Costing Method is defined and Item Tracking is set up, the next step is to create the bill of materials for the Office Chair. This can be done using the **Bill of Materials** option in the **Assembly List** in the Item List or Card page.

The **Bill of Materials** defines the component items and resources that will be used to create one new end product.



We set up the components as discussed earlier using five wheels, one pedestal and seat, and two arm rests.


Edit - Bill of Materials - CHAPTER5-P1 · Chapter 5 | Office Chair

Actions

Type	No.	Description	Bill ...	Quantity per	Unit of Meas...	Installed in It...	Pos
Item	CHAPTER5-C1	Chapter 5 | Wheel	No	5	PCS		
Item	CHAPTER5-C2	Chapter 5 | Pedestal	No	1	PCS		
Item	CHAPTER5-C3	Chapter 5 | Seat	No	1	PCS		
Item	CHAPTER5-C4	Chapter 5 | Arm Rest	No	2	PCS		

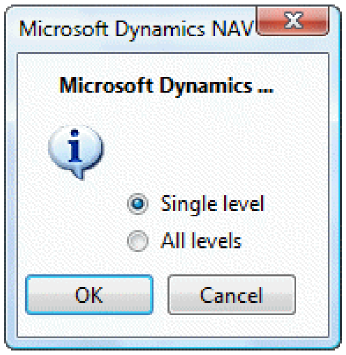
Calculate standard cost

The components we selected for the Bill of Materials have a Unit Cost. Together these items can determine the Unit Cost of our end product.

We can calculate the Standard Cost of the Office Chair by selecting **Calc. Standard Cost** in the same **Assembly List** we used for the Bill of Materials.

> The name Calc. Standard Cost in the calculation option is not to be confused with the costing method. This function should be executed for all costing methods.

The system asks if we want to calculate **Single level** or **All levels**. Since our example BOM has only one level, we select **Single level**.


Microsoft Dynamics NAV

Microsoft Dynamics ...

[✓ Single level] All levels

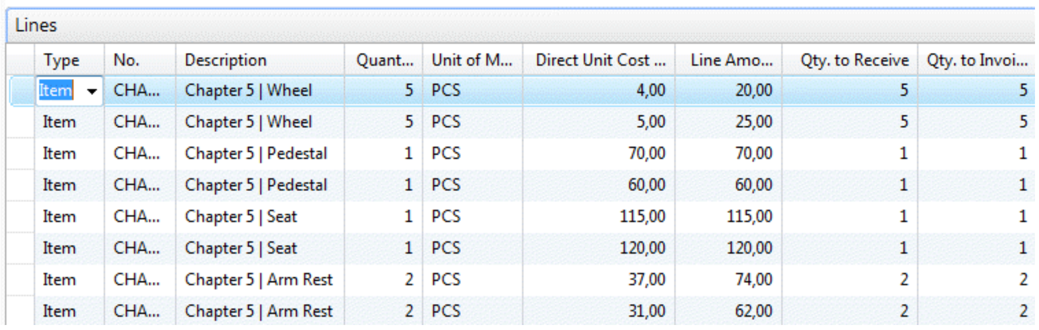
Single level means that only the first level of the BOM is checked. If an item in the BOM has a BOM of itself, the value of that item is not recalculated.

The Standard Cost is now calculated using the Unit Cost of the components and the Overhead rate and Indirect Cost %. We will not use the latter in our example.

```
((5*5) + (1*60) + (1*120) + (2*35)) = 275
```

Creating the inventory

Before we can assemble the chairs we need to purchase the components. For this, we will create a purchase order with 8 purchase lines. We will purchase the components for other prices than the unit cost in the system allowing us to show what the impacts of the costing methods are.



Lines

Type	No.	Description	Quant...	Unit of M...	Direct Unit Cost ...	Line Amo...	Qty. to Receive	Qty. to Invoi...
Item	CHA...	Chapter 5 | Wheel	5	PCS	4,00	20,00	5	5
Item	CHA...	Chapter 5 | Wheel	5	PCS	5,00	25,00	5	5
Item	CHA...	Chapter 5 | Pedestal	1	PCS	70,00	70,00	1	1
Item	CHA...	Chapter 5 | Pedestal	1	PCS	60,00	60,00	1	1
Item	CHA...	Chapter 5 | Seat	1	PCS	115,00	115,00	1	1
Item	CHA...	Chapter 5 | Seat	1	PCS	120,00	120,00	1	1
Item	CHA...	Chapter 5 | Arm Rest	2	PCS	37,00	74,00	2	2
Item	CHA...	Chapter 5 | Arm Rest	2	PCS	31,00	62,00	2	2

The purchase order will be received and invoiced.

If the Purchase Order is only received and not invoiced, the example might not work because **Expected Cost posting to G/L** is not activated in the CRONUS database.

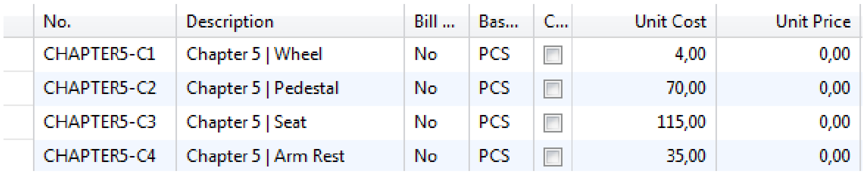
Adjusting cost item entries

The Purchase Order we just created, received, and invoiced has a different Unit Cost compared to the Unit Cost we initially set up in our Items.

Depending on the Costing Method of the Items this will have an impact on the Unit Cost.

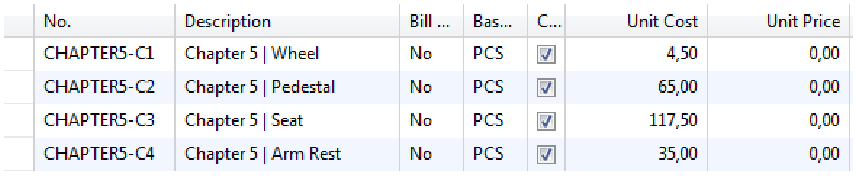
When we take a new look at the Unit Cost of the Items we created, we can see that this was impacted by the purchase order.

However, the values are not correct. The system only adopts the first change of **Unit Cost**.



No.	Description	Bill ...	Bas...	C...	Unit Cost	Unit Price
CHAPTER5-C1	Chapter 5 | Wheel	No	PCS	☐	4,00	0,00
CHAPTER5-C2	Chapter 5 | Pedestal	No	PCS	☐	70,00	0,00
CHAPTER5-C3	Chapter 5 | Seat	No	PCS	☐	115,00	0,00
CHAPTER5-C4	Chapter 5 | Arm Rest	No	PCS	☐	35,00	0,00

To correct this we need to run the Adjust Cost Item Entries (Report 795) batch. This will determine the new Unit Cost based on the Costing Method.



No.	Description	Bill ...	Bas...	C...	Unit Cost	Unit Price
CHAPTER5-C1	Chapter 5 | Wheel	No	PCS	☑	4,50	0,00
CHAPTER5-C2	Chapter 5 | Pedestal	No	PCS	☑	65,00	0,00
CHAPTER5-C3	Chapter 5 | Seat	No	PCS	☑	117,50	0,00
CHAPTER5-C4	Chapter 5 | Arm Rest	No	PCS	☑	35,00	0,00

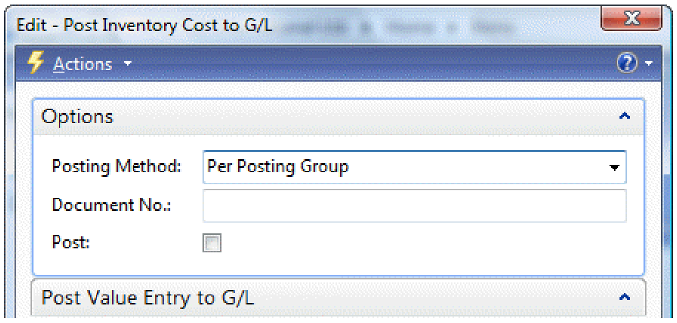
The Unit Cost for FIFO, LIFO, and Average has been recalculated whist the Standard Cost has not been impacted by the transactions.

The Adjust Cost Item Entries report should be scheduled to run periodically in your database. Even if the database is set to use Expected Cost Posting and Auto Cost Posting.

Posting inventory cost to G/L

Microsoft Dynamics NAV supports posting the Inventory Cost to the General Ledger. This enables accountants to have a single point for data analysis rather than printing an inventory report and using the figures manually for reporting to the management.

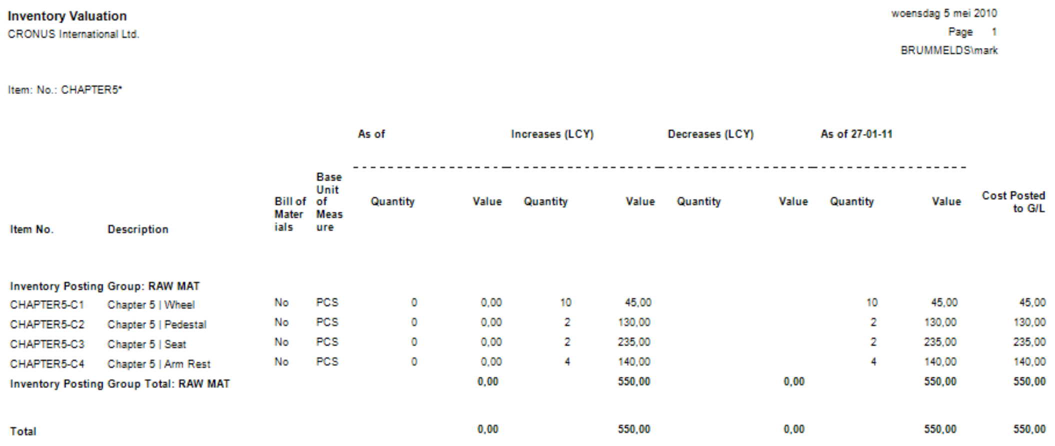
This can be done using the **Post Inventory Cost to G/L** function (report 1002).



Posting Method: Per Posting Group
Document No.:
Post: ☐

Check, check, and double check

To check if the Value Entries and the General Ledger and synchronized, we can run **Inventory Valuation** (Report 1001). This will show us the Inventory Value versus the Amount posted to the General Ledger.



Inventory Valuation
CRONUS International Ltd.

woensdag 5 mei 2010
Page 1
BRUMMELDS\mark

Item: No.: CHAPTER5*

				As of		Increases (LCY)		Decreases (LCY)		As of 27-01-11		
Item No.	Description	Bill of Materials	Base Unit of Measure	Quantity	Value	Quantity	Value	Quantity	Value	Quantity	Value	Cost Posted to G/L
Inventory Posting Group: RAW MAT												
CHAPTER5-C1	Chapter 5 | Wheel	No	PCS	0	0,00	10	45,00			10	45,00	45,00
CHAPTER5-C2	Chapter 5 | Pedestal	No	PCS	0	0,00	2	130,00			2	130,00	130,00
CHAPTER5-C3	Chapter 5 | Seat	No	PCS	0	0,00	2	235,00			2	235,00	235,00
CHAPTER5-C4	Chapter 5 | Arm Rest	No	PCS	0	0,00	4	140,00			4	140,00	140,00
Inventory Posting Group Total: RAW MAT					**0,00**		**550,00**		**0,00**		**550,00**	**550,00**
Total					**0,00**		**550,00**		**0,00**		**550,00**	**550,00**

Recalculating standard unit cost

The Standard Unit Cost we calculated for our Office chair was 275. This was based on our assumption of purchase prices. Now that we have purchased and received the components we can calculate a new Unit Cost based on the real prices.

In this example, the price will still be 275 since the total price of all purchased items is 550. The inventory allows us to make two chairs with these materials.

```
(550 / 2) = 275
```

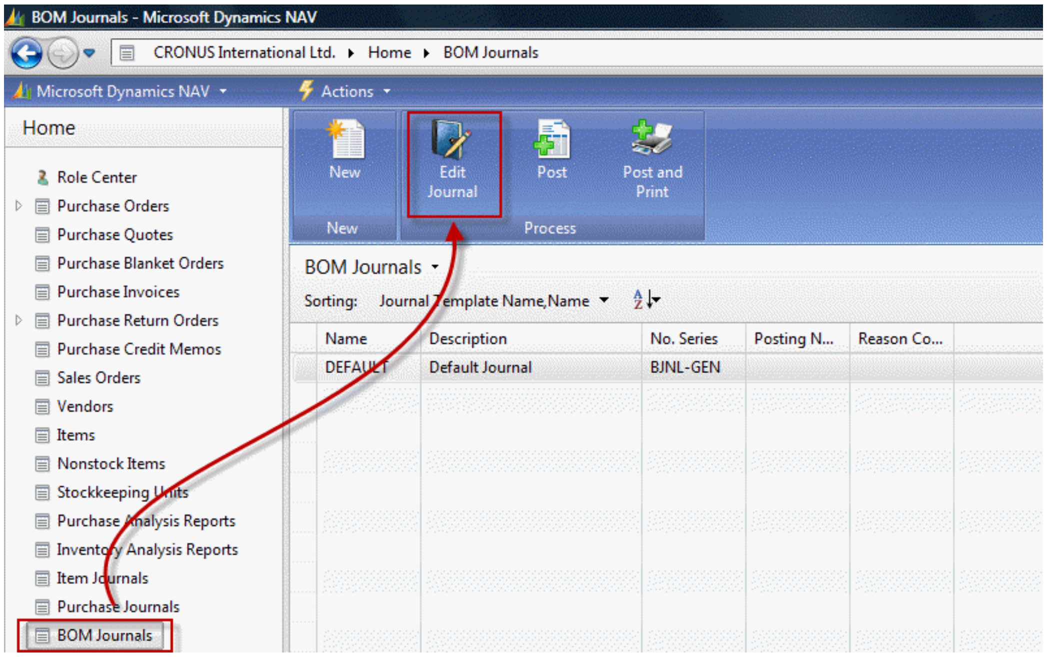
[With this calculation method it is possible to check the results of the Calculate Unit Cost algorithm.]

BOM journal

Now that we have the components on stock and the Unit Cost correct we can create a chair. We will do this using the BOM Journal.

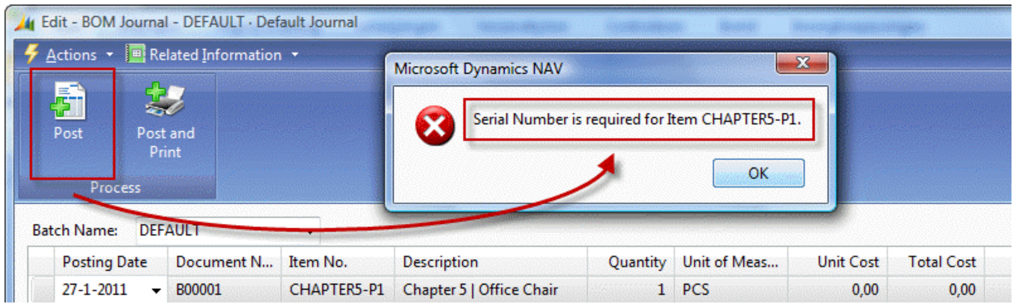
In the BOM Journal, we need to create one line for each Item we want to assemble. The components are automatically used when posting the Journal Lines.

We will use the Purchasing Agent Role Center (9007) for this.



Name	Description	No. Series	Posting N...	Reason Co...
DEFAULT	Default Journal	BJNL-GEN		

After creating a Journal Line for the Office Chair and trying to Post the Journal we will receive an error message because we first need to specify the **Serial Numbers**.

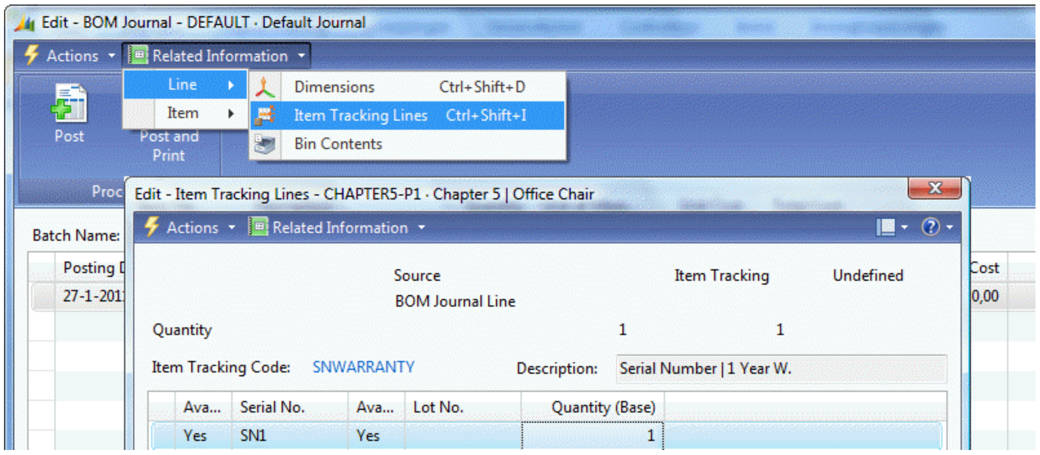


Serial Number is required for Item CHAPTER5-P1.

Posting Date	Document N...	Item No.	Description	Quantity	Unit of Meas...	Unit Cost	Total Cost
27-1-2011	B00001	CHAPTER5-P1	Chapter 5 | Office Chair	1	PCS	0,00	0,00

Serial Numbers and Lot Numbers can be set up using the Item Tracking Lines option. This opens the Item Tracking Line page (6510). This page is able to show both the Reservation Entries during the registration process as well as the Tracking Specification if the Item Ledger entry is already created. The latter is not possible in a Journal, only when using Documents.

We will manually create a new **Serial Number**. Microsoft Dynamics NAV also supports system generated Serial Numbers.


Edit - BOM Journal - DEFAULT · Default Journal
Actions
Related Information
Line
Item
Dimensions Ctrl+Shift+D
Item Tracking Lines Ctrl+Shift+I
Bin Contents
Post
Post and Print
Batch Name:
Posting D
27-1-201
Cost
0,00
Edit - Item Tracking Lines - CHAPTER5-P1 · Chapter 5 | Office Chair

	Source	Item Tracking	Undefined
	BOM Journal Line		
Quantity	1	1	

Item Tracking Code: SNWARRANTY Description: Serial Number | 1 Year W.

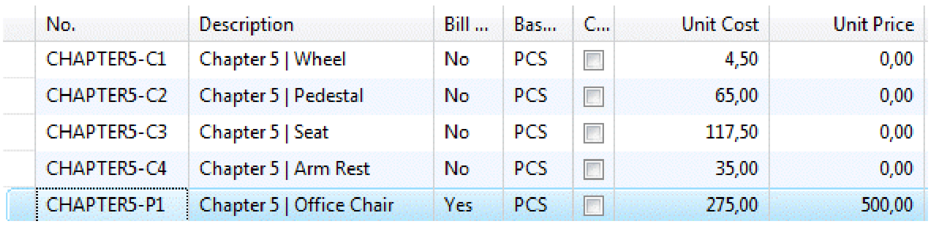
Ava...	Serial No.	Ava...	Lot No.	Quantity (Base)
Yes	SN1	Yes		1

We can now post this BOM Journal and will have one office chair on stock with a Serial Number.

Check costing (again)

Creating the office chair changed the inventory of the component items and therefore might have affected the Costing of our Items.

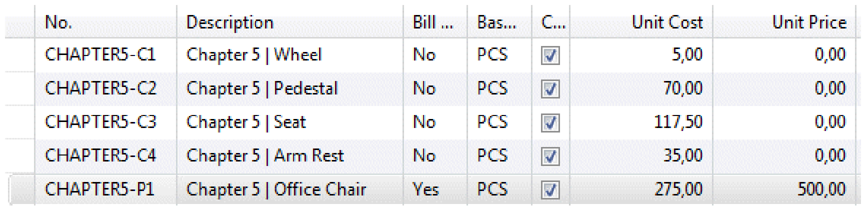
However, when we now check our Items, the Unit Cost has not changed.



No.	Description	Bill ...	Bas...	C...	Unit Cost	Unit Price
CHAPTER5-C1	Chapter 5 | Wheel	No	PCS	☐	4,50	0,00
CHAPTER5-C2	Chapter 5 | Pedestal	No	PCS	☐	65,00	0,00
CHAPTER5-C3	Chapter 5 | Seat	No	PCS	☐	117,50	0,00
CHAPTER5-C4	Chapter 5 | Arm Rest	No	PCS	☐	35,00	0,00
CHAPTER5-P1	Chapter 5 | Office Chair	Yes	PCS	☐	275,00	500,00

Even so, with current inventory the Unit Cost might be different. Remember we used 5 wheels using FIFO costing and one seat using LIFO costing 70.

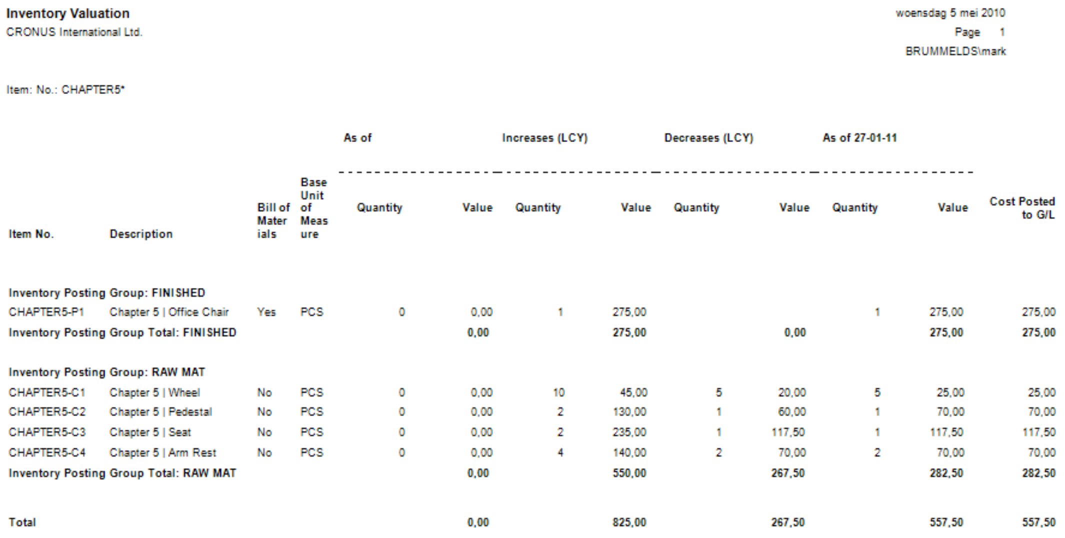
Let's run Adjust Cost Entries using the Posting to G/L option.



No.	Description	Bill ...	Bas...	C...	Unit Cost	Unit Price
CHAPTER5-C1	Chapter 5 | Wheel	No	PCS	✓	5,00	0,00
CHAPTER5-C2	Chapter 5 | Pedestal	No	PCS	✓	70,00	0,00
CHAPTER5-C3	Chapter 5 | Seat	No	PCS	✓	117,50	0,00
CHAPTER5-C4	Chapter 5 | Arm Rest	No	PCS	✓	35,00	0,00
CHAPTER5-P1	Chapter 5 | Office Chair	Yes	PCS	✓	275,00	500,00

The Unit Cost has changed and now shows that have used the first 5 wheels using FIFO leaving the other 5 wheels in the inventory for 5. We used the last seat using LIFO leaving the first seat in inventory for 70.

When we run the **Inventory Valuation** we can see that producing the first chair actually cost 267,50 but we posted 275.



Inventory Valuation
CRONUS International Ltd.

woensdag 5 mei 2010
Page 1
BRUMMELDS\mark

Item: No.: CHAPTER5*

Item No.	Description	Bill of Materials	Base Unit of Measure	As of Quantity	As of Value	Increases (LCY) Quantity	Increases (LCY) Value	Decreases (LCY) Quantity	Decreases (LCY) Value	As of 27-01-11 Quantity	As of 27-01-11 Value	Cost Posted to G/L
Inventory Posting Group: FINISHED												
CHAPTER5-P1	Chapter 5 | Office Chair	Yes	PCS	0	0,00	1	275,00			1	275,00	275,00
Inventory Posting Group Total: FINISHED					**0,00**		**275,00**		**0,00**		**275,00**	**275,00**
Inventory Posting Group: RAW MAT												
CHAPTER5-C1	Chapter 5 | Wheel	No	PCS	0	0,00	10	45,00	5	20,00	5	25,00	25,00
CHAPTER5-C2	Chapter 5 | Pedestal	No	PCS	0	0,00	2	130,00	1	60,00	1	70,00	70,00
CHAPTER5-C3	Chapter 5 | Seat	No	PCS	0	0,00	2	235,00	1	117,50	1	117,50	117,50
CHAPTER5-C4	Chapter 5 | Arm Rest	No	PCS	0	0,00	4	140,00	2	70,00	2	70,00	70,00
Inventory Posting Group Total: RAW MAT					**0,00**		**550,00**		**267,50**		**282,50**	**282,50**
Total					**0,00**		**825,00**		**267,50**		**557,50**	**557,50**

Recalculating unit cost (again)

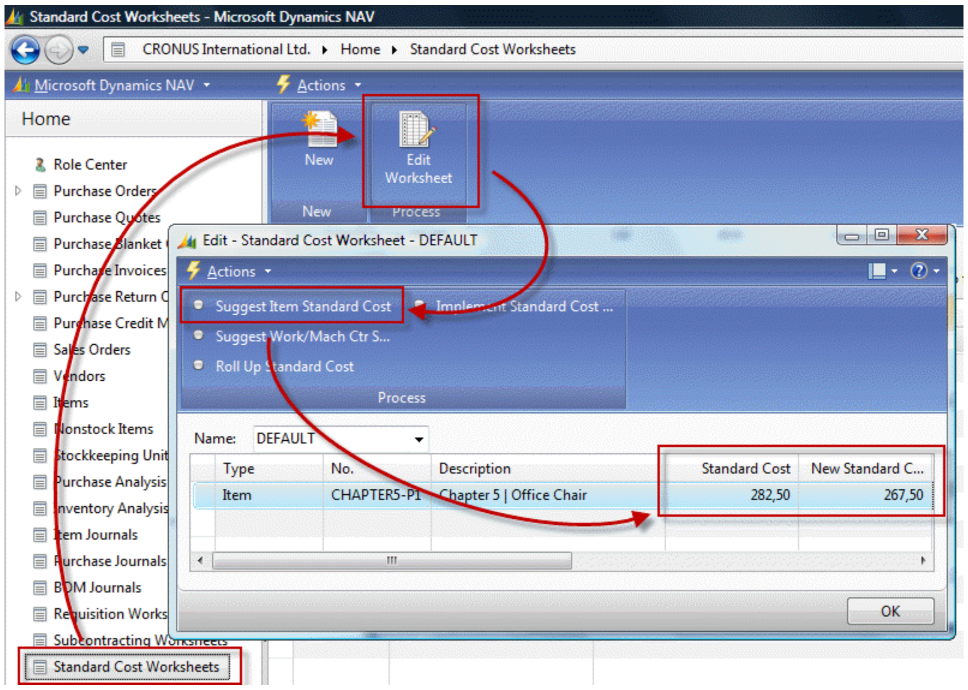
When we run **Calculate Unit Cost** for our office Chair, we can see that the new cost will be 282,50.

```
(5*5) + (1*70) + (1* 117,50) + (2*35) = 282,50
```

Together with the first chair worth 267,50 we match our purchase invoice worth 550.

Standard cost worksheet

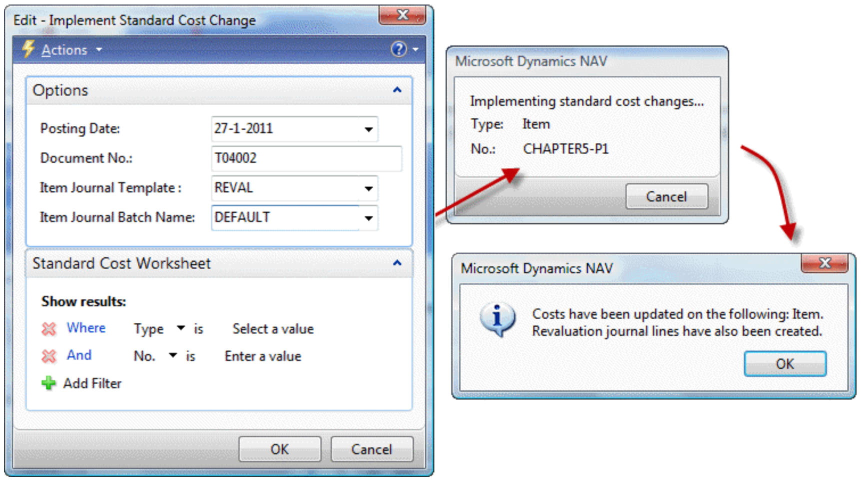
We need to correct the cost of the first chair which is currently on inventory to have a correct inventory value. We can do this using the **Standard Cost Worksheet**.



Name: DEFAULT

Type	No.	Description	Standard Cost	New Standard C...
Item	CHAPTER5-P1	Chapter 5 | Office Chair	282,50	267,50

This worksheet allows us to correct old Value Entries by creating an entry in the Item Revaluation Journal when we select the option **Implement Standard Cost Changes**.



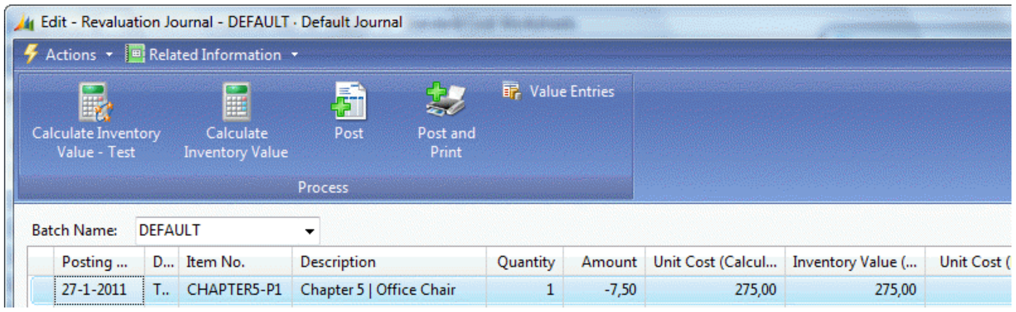
Posting Date: 27-1-2011
Document No.: T04002
Item Journal Template : REVAL
Item Journal Batch Name: DEFAULT

Implementing standard cost changes...
Type: Item
No.: CHAPTER5-P1

Costs have been updated on the following: Item.
Revaluation journal lines have also been created.

Item revaluation journal

The last step is to post the Item Revaluation Journal and run the Post Inventory Cost to G/L routine.


Edit - Revaluation Journal - DEFAULT · Default Journal
Actions
Related Information
Calculate Inventory Value - Test
Calculate Inventory Value
Post
Post and Print
Value Entries
Process

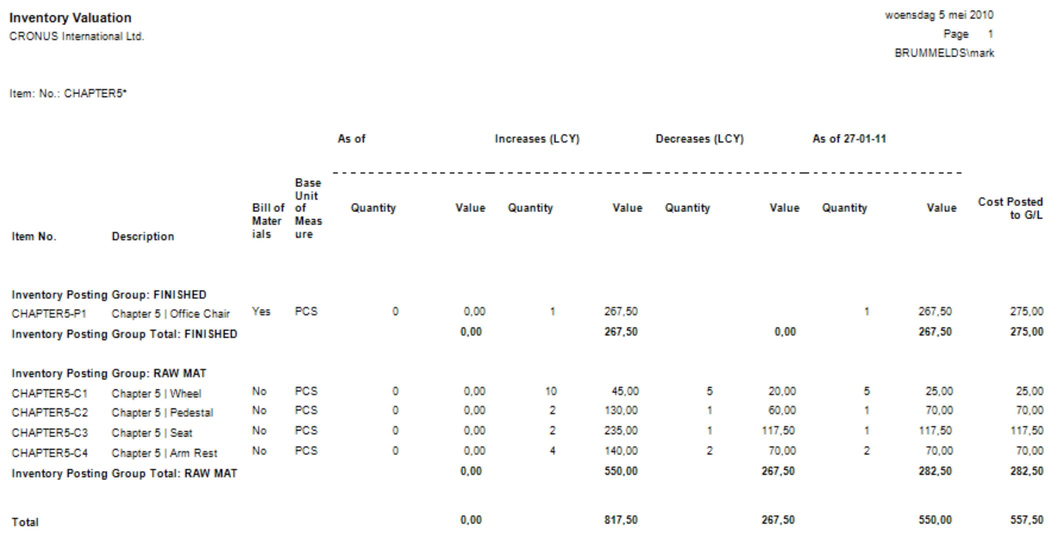
Batch Name: DEFAULT

Posting ...	D...	Item No.	Description	Quantity	Amount	Unit Cost (Calcul...	Inventory Value (...	Unit Cost (
27-1-2011	T..	CHAPTER5-P1	Chapter 5 | Office Chair	1	-7,50	275,00	275,00	

We also need to run the Calculate Standard Cost for the Office Chair since this batch will reset the Unit Cost to 267,50.

The result

When we now run the Inventory Valuation Report we can see that the Office Chair on stock is worth 267,50 and the remaining inventory is 282,50



Inventory Valuation
CRONUS International Ltd.

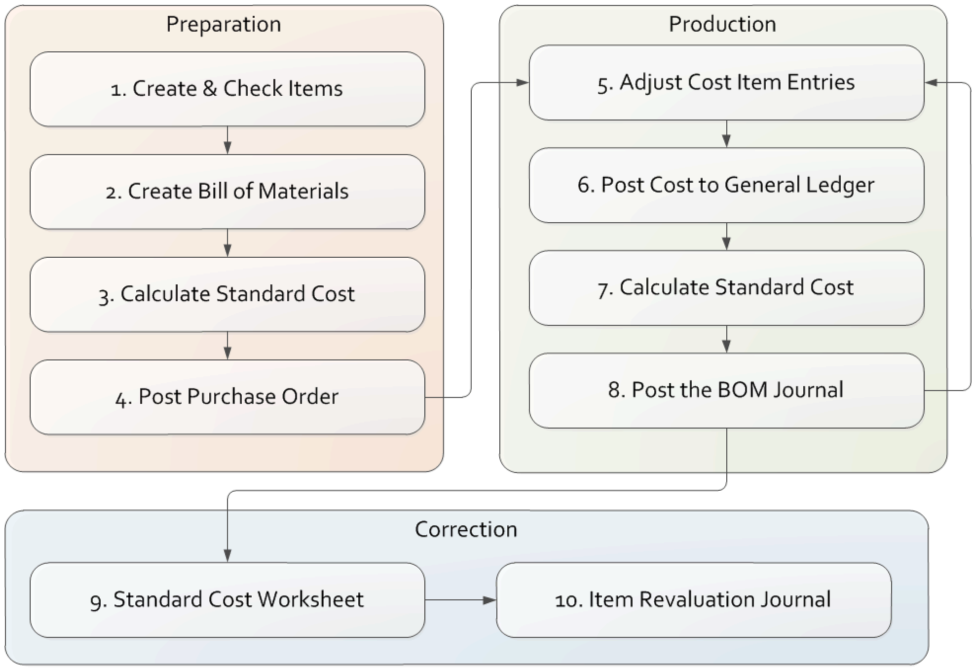
woensdag 5 mei 2010
Page 1
BRUMMELDS\mark

Item: No.: CHAPTER5*

				As of		Increases (LCY)		Decreases (LCY)		As of 27-01-11		
Item No.	Description	Bill of Materials	Base Unit of Measure	Quantity	Value	Quantity	Value	Quantity	Value	Quantity	Value	Cost Posted to G/L
Inventory Posting Group: FINISHED												
CHAPTER5-P1	Chapter 5 | Office Chair	Yes	PCS	0	0,00	1	267,50			1	267,50	275,00
Inventory Posting Group Total: FINISHED					**0,00**		**267,50**		**0,00**		**267,50**	**275,00**
Inventory Posting Group: RAW MAT												
CHAPTER5-C1	Chapter 5 | Wheel	No	PCS	0	0,00	10	45,00	5	20,00	5	25,00	25,00
CHAPTER5-C2	Chapter 5 | Pedestal	No	PCS	0	0,00	2	130,00	1	60,00	1	70,00	70,00
CHAPTER5-C3	Chapter 5 | Seat	No	PCS	0	0,00	2	235,00	1	117,50	1	117,50	117,50
CHAPTER5-C4	Chapter 5 | Arm Rest	No	PCS	0	0,00	4	140,00	2	70,00	2	70,00	70,00
Inventory Posting Group Total: RAW MAT					**0,00**		**550,00**		**267,50**		**282,50**	**282,50**
Total					**0,00**		**817,50**		**267,50**		**550,00**	**557,50**

Item costing in ten steps

All the steps we performed in the example can be summarized in this ten step process diagram.


Preparation
1. Create & Check Items
2. Create Bill of Materials
3. Calculate Standard Cost
4. Post Purchase Order
Production
5. Adjust Cost Item Entries
6. Post Cost to General Ledger
7. Calculate Standard Cost
8. Post the BOM Journal
Correction
9. Standard Cost Worksheet
10. Item Revaluation Journal

1. We start by creating new Items and setting up the Costing Method.
2. Then, we create the Bill of Materials for the production item.
3. Run the Calculate Standard Cost routine to have a Unit Cost for the production Item.
4. Purchase the necessary items for production.
5. Run the Adjust Cost Item Entries routine.
6. Synchronize the Value entries with the G/L Entries using the Post Inventory Cost to G/L routine.
7. Recalculate the Standard Cost if desired.
8. Create a BOM Journal Line and post the Journal.
9. Run the Standard Cost Worksheet to change the Standard Cost.
10. Run the Item Revaluation Journal to implement the Standard Cost for old Value Entries.

Manufacturing

The Bill of Materials Journal has been part of the Microsoft Dynamics NAV product since the introduction of the Windows version in 1995 and enables us to create a new item out of other items.

In version 2.01, Navision introduced the first version of Navision Manufacturing. This was a separate product from Navision Financials and only available for certified partners because of its complexity.

With version 3.00, Manufacturing became part of the standard Navision Attain package and was available for all partners.

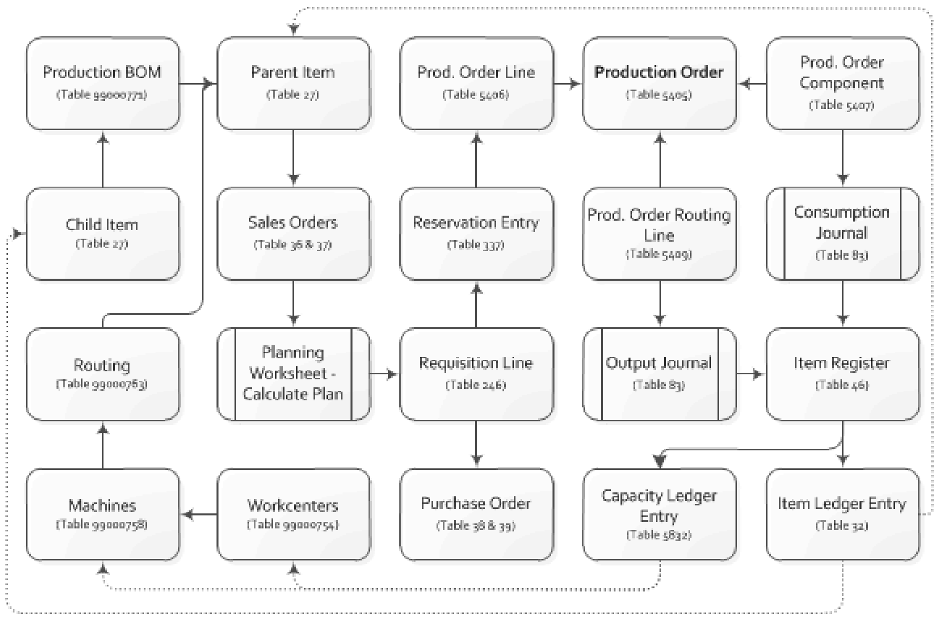
Manufacturing offers Microsoft Dynamics NAV users much more functionality than just creating an item from a Bill of Materials. Production orders can be scheduled using work centers, machine centers, tools, and a capacity calendar.

The items can be scheduled for production using either a Make-To-Stock or a Make-To-Order policy in a Planning run.

The system calculates the required BOM components and can create purchase orders if the inventory is insufficient using a complex demand and supply process.

The table and posting schema

If we put this into the table and posting schema it looks like this:


Production BOM (Table 99000771)
Parent Item (Table 27)
Prod. Order Line (Table 5406)
Production Order (Table 5405)
Prod. Order Component (Table 5407)
Child Item (Table 27)
Sales Orders (Table 36 & 37)
Reservation Entry (Table 337)
Prod. Order Routing Line (Table 5409)
Consumption Journal (Table 83)
Routing (Table 99000763)
Planning Worksheet - Calculate Plan
Requisition Line (Table 246)
Output Journal (Table 83)
Item Register (Table 46)
Machines (Table 99000758)
Workcenters (Table 99000754)
Purchase Order (Table 38 & 39)
Capacity Ledger Entry (Table 5832)
Item Ledger Entry (Table 32)

The Production Order is the center of the process which is created by items having a Production BOM that are either on Sales Order or low on inventory.

The Planning Run populates the Planning Worksheet which is based on the Requisition Line table (246). The Planning worksheet can be used to create the production orders and purchase orders.

During the production process, the consumption journal is used to record the use of the child items from Production BOM and the Output journal creates the new item once it is finished. Alternatively, these steps can be combined in the Production Journal.

Let's demonstrate this with the next example using the Production Planner role center (9010).

We will create mahogany English desks using raw materials, machines, and resources.

The items, machines, and work centers

For the desks we need mahogany wood, green leather, glue, lacquer, and handles. To create one desk, the carpenter needs four days and one carpentry unit with tools. The painter needs one day to varnish the desk in the paint booth and the assembly department needs four hours to wrap the components and put them into boxes.

Items

No.	Description	Base UOM	Replenishment System	Unit Cost	Unit Price	Manufacturing Policy
CHAPTER5-P1	Mahogany English Desk	PCS	Prod. Order	286,25	999	Make-to-Order
CHAPTER5-C1	Mahogany Log	PCS	Purchase	100		-
CHAPTER5-C2	Green Leather	PCS	Purchase	60		-
CHAPTER5-C3	Glue	CAN	Purchase	15		-
CHAPTER5-C4	Mahogany Lacquer	CAN	Purchase	25		-
CHAPTER5-C5	English Desk Handles	PCS	Purchase	10		-

Machine centers

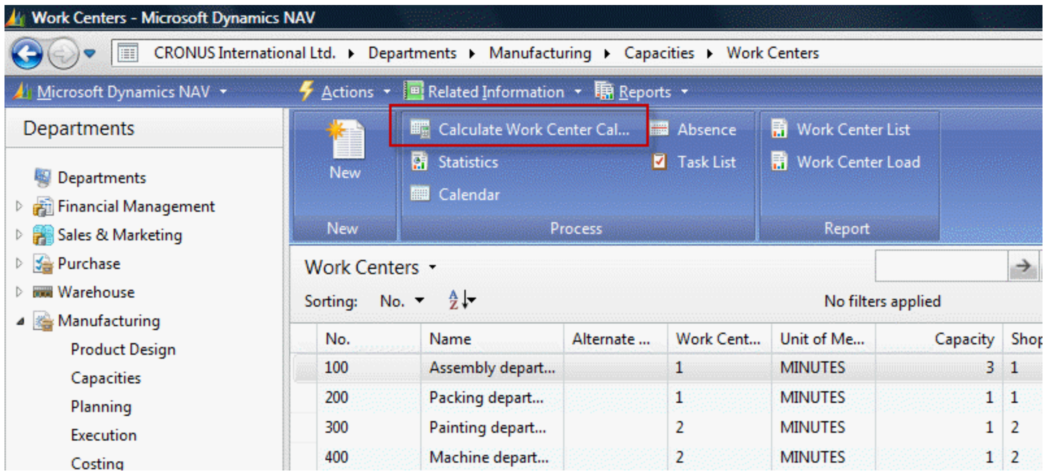
No.	Name	Work Center No.	Capacity	Efficiency
01-CARP	Std. Carpentry Unit w. Tools	400	1	100
02-PAINT	Paint Booth	300	1	100
03-PACK	Packaging Department	200	1	100

Capacity

The Planning run and the Requisition Worksheets will use capacity if it is defined. The capacity is defined for each Work Center and Machine Center.

The capacity is stored in the Calendar Entries which are created using Codeunit CalendarManagement (99000755) and reports Calculate Work Center Calendar (99001046) and Calc. Machine Center Calendar (99001045). Capacity is based on the Concurrent Capacity, from either the Machine or Work Center, and the assigned Shop calendar.

Just like the Interaction Log Entries, the Calendar Entries are directly inserted instead of going through a journal first.



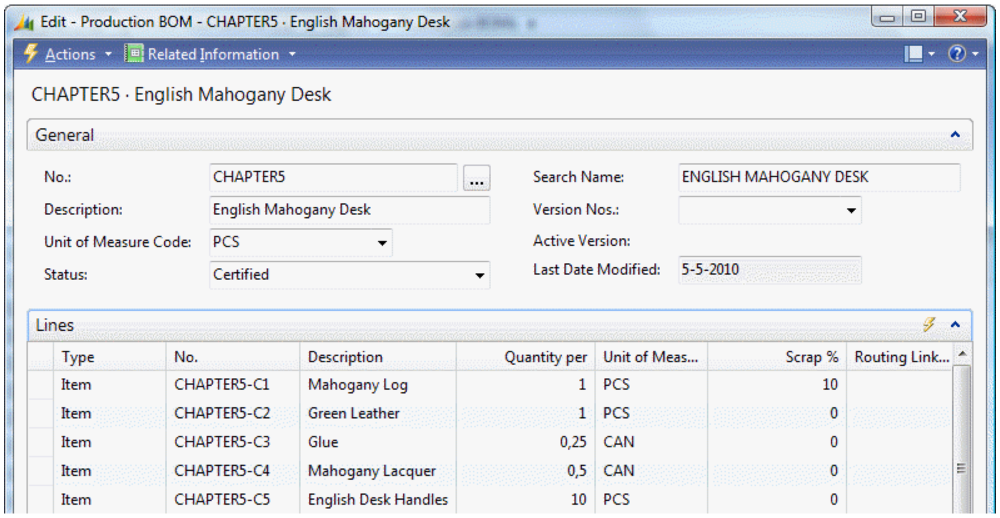
No.	Name	Alternate ...	Work Cent...	Unit of Me...	Capacity	Shop
100	Assembly depart...		1	MINUTES	3	1
200	Packing depart...		1	MINUTES	1	1
300	Painting depart...		2	MINUTES	1	2
400	Machine depart...		2	MINUTES	1	2

When properly configured, the Planning Worksheet will calculate the starting and ending dates for the production order and each operation in order to meet the Shipment Date on the Sales Order Line.

Production bill of materials

Setting up the Production Bill of Materials for manufacturing is not much different from the BOM Journal functionality but it contains extra functionality.

The Production BOM uses its own header record with a number series, description, and search description. The **Status** field is used to determine if the product is new, certified, under development, or closed. Together with the versioning it enables us to maintain multiple BOMs during the product's lifecycle.



CHAPTER5 · English Mahogany Desk

General

No.:	CHAPTER5	Search Name:	ENGLISH MAHOGANY DESK
Description:	English Mahogany Desk	Version Nos.:	
Unit of Measure Code:	PCS	Active Version:	
Status:	Certified	Last Date Modified:	5-5-2010

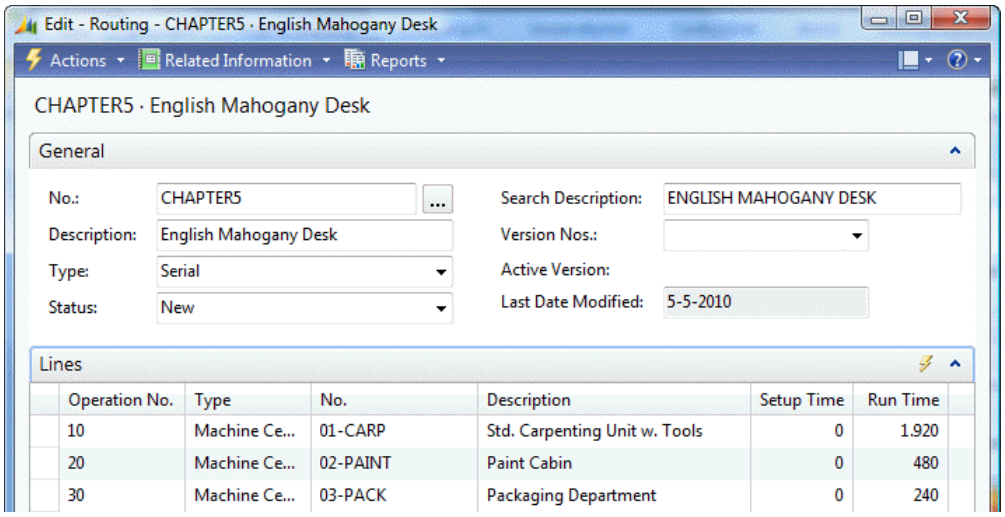
Lines

Type	No.	Description	Quantity per	Unit of Meas...	Scrap %	Routing Link...
Item	CHAPTER5-C1	Mahogany Log	1	PCS	10	
Item	CHAPTER5-C2	Green Leather	1	PCS	0	
Item	CHAPTER5-C3	Glue	0,25	CAN	0	
Item	CHAPTER5-C4	Mahogany Lacquer	0,5	CAN	0	
Item	CHAPTER5-C5	English Desk Handles	10	PCS	0	

The components of the Bill of Materials are saved as lines and support using Scrap. The scrap % is calculated when running the MRP and calculating the Unit Cost.

Routing

The Routing Setup determines how long it will take to produce one item and which **Work Centers** and **Machine Centers** are used in the process.



CHAPTER5 · English Mahogany Desk

General

No.:	CHAPTER5	Search Description:	ENGLISH MAHOGANY DESK
Description:	English Mahogany Desk	Version Nos.:	
Type:	Serial	Active Version:	
Status:	New	Last Date Modified:	5-5-2010

Lines

Operation No.	Type	No.	Description	Setup Time	Run Time
10	Machine Ce...	01-CARP	Std. Carpenting Unit w. Tools	0	1.920
20	Machine Ce...	02-PAINT	Paint Cabin	0	480
30	Machine Ce...	03-PACK	Packaging Department	0	240

The Routing Setup allows for advanced features such as parallel and serial planning, setup time, and so on. For our example, we will keep things simple and only use the Run Time.

Testing and low level code

We are now almost ready to start testing our Manufacturing Item. We have set up the items and machine centers, calculated the calendar entries, and set up a routing.

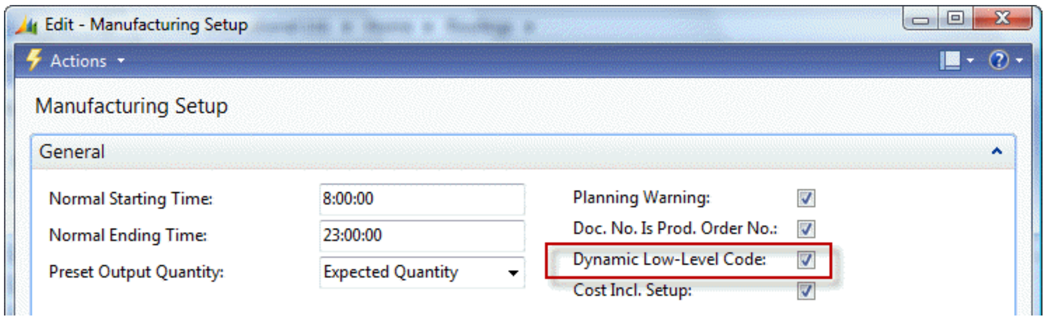
The final step in the process is to calculate the Low Level Code. This field, which is stored both in the Item and Production BOM table, determines how low the Item is in the BOM ranking. Low Level Code zero means this is a Parent Item and one or higher is a child item or a child of a child item.

[The maximum value of the Low Level Code can be 50 but in reality this will be very difficult to work with and bad for system performance.]

[If you received an error stating that you have exceeded 50 levels check the Production BOMs to ensure that there are no circular references. It is possible to have a parent item consume a child item that consumes the parent.]

The Low Level Codes can be calculated automatically or manually. For automatic calculation, the Dynamic Low Level Code should be activated.

Due to NAV's ability to create a Production BOM before it is attached to an Item, the **Dynamic Low Level Code** is not always accurate. Prior to a planning run, it is good practice to run the Calculate Low Level Code (Codeunit 99000793).



Normal Starting Time:	8:00:00	Planning Warning:	✓
Normal Ending Time:	23:00:00	Doc. No. Is Prod. Order No.:	✓
Preset Output Quantity:	Expected Quantity	Dynamic Low-Level Code:	✓
		Cost Incl. Setup:	✓

Activating Dynamic Low Level Code can however impact the performance of your system, so for most installations, it is preferable to periodically calculate this using Codeunit Calc. Low-level code (99000853).

Simulation, sales orders, or inventory

There are three ways to create a production order in Microsoft Dynamics NAV. The easiest way is to manually enter them one-by-one. This can even be a simulation production order to test if everything is set up as required.

Manual order entry is very time consuming and not often used by manufacturing companies. Most of them use MRP programs to plan the orders. When this is done using an external application, the interface will then create the production orders.

The MRP algorithm in Microsoft Dynamics NAV supports two policies: Make-to-stock and make-to-order.

Make-to-stock

Also called "Build to Stock", this is often used for high volume items which are sold to trading companies.

When this Manufacturing Policy is used, the Reordering Policy should be used.

Reordering Policies will be discussed in *Chapter 6, Trade*.

Make-to-order

This is often used in demand driven items such as automobiles. Keeping these items in inventory is very expensive. The manufacturing process is started after the item is sold.

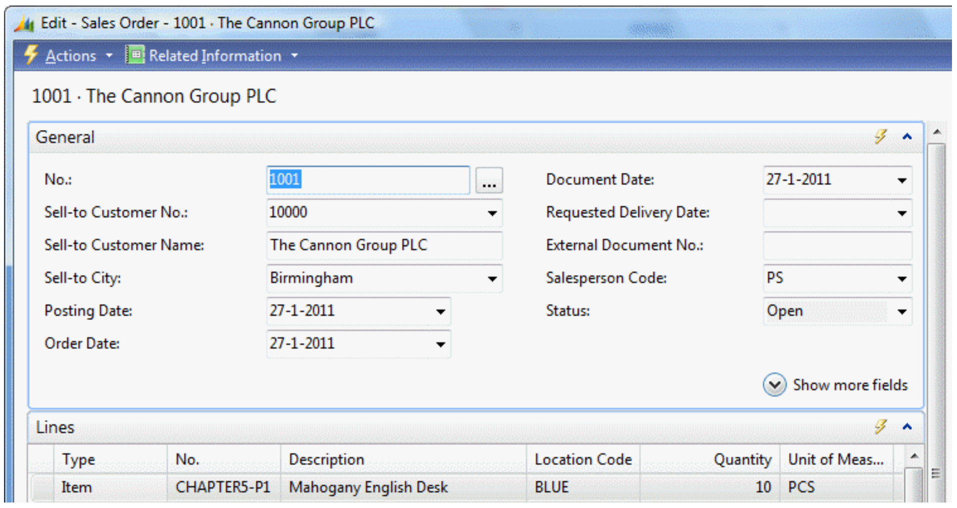
However, most companies that use Make-to-order have reserved time slots where these items can be scheduled so the production capacity is already reserved but the item is not yet determined.

When using Make-to-order, the MRP run will create production orders for all sales orders. We will use this Manufacturing policy in our example.

The sales order

For our example, we need a sales order for 1 or more English Desks.

> Be careful when picking a location since this will be the location where the desk will be manufactured.



1001 · The Cannon Group PLC

General

No.:	1001	Document Date:	27-1-2011
Sell-to Customer No.:	10000	Requested Delivery Date:	
Sell-to Customer Name:	The Cannon Group PLC	External Document No.:	
Sell-to City:	Birmingham	Salesperson Code:	PS
Posting Date:	27-1-2011	Status:	Open
Order Date:	27-1-2011		

Lines

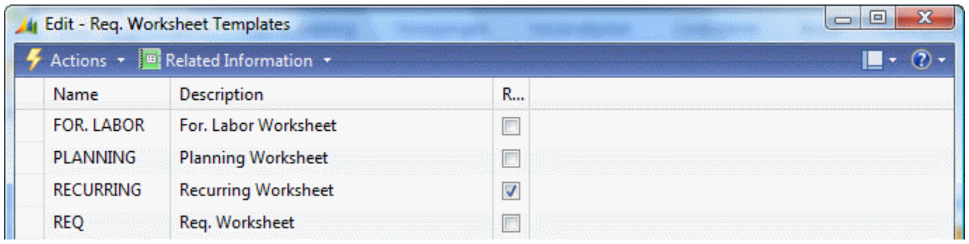
Type	No.	Description	Location Code	Quantity	Unit of Meas...
Item	CHAPTER5-P1	Mahogany English Desk	BLUE	10	PCS

Calculating MPS and MRP

The Planning run in Microsoft Dynamics NAV creates lines in the Requisition or Planning Worksheets. This Worksheet Structure is important in the Sales/Purchase/Production process. This worksheet can create purchase orders and production orders for sales orders.

Requisition versus planning versus subcontracting worksheets

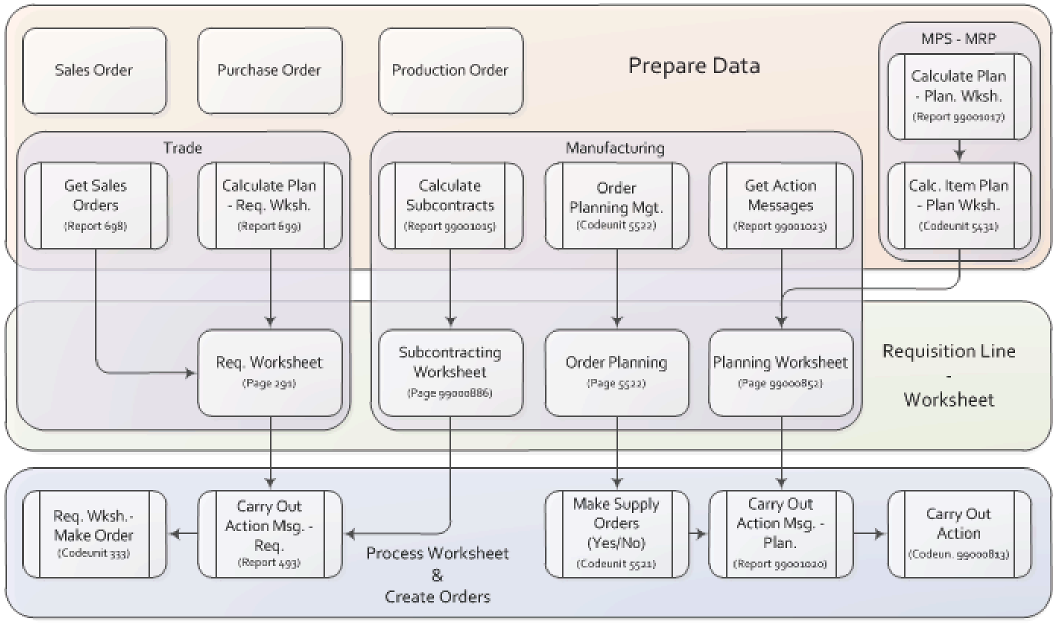
The Requisition Worksheet can show different user interfaces (Forms or Pages) allowing users to do different tasks.



Name	Description	R...
FOR. LABOR	For. Labor Worksheet	☐
PLANNING	Planning Worksheet	☐
RECURRING	Recurring Worksheet	✓
REQ	Req. Worksheet	☐

The Requisition Worksheet does not have a general Post Line routine like the other journals. Each worksheet type uses another process.

This schema shows how the Requisition process ties together:


Sales Order
Purchase Order
Production Order
Prepare Data
MPS - MRP
Calculate Plan - Plan. Wksh. (Report 99001017)
Calc. Item Plan - Plan Wksh. (Codeunit 5431)
Trade
Get Sales Orders (Report 698)
Calculate Plan - Req. Wksh. (Report 699)
Manufacturing
Calculate Subcontracts (Report 99001015)
Order Planning Mgt. (Codeunit 5522)
Get Action Messages (Report 99001023)
Req. Worksheet (Page 291)
Subcontracting Worksheet (Page 99000886)
Order Planning (Page 5522)
Planning Worksheet (Page 99000852)
Requisition Line - Worksheet
Req. Wksh.- Make Order (Codeunit 333)
Carry Out Action Msg. - Req. (Report 493)
Process Worksheet & Create Orders
Make Supply Orders (Yes/No) (Codeunit 5521)
Carry Out Action Msg. - Plan. (Report 99001020)
Carry Out Action (Codeun. 99000813)

The **Trade** section will be discussed in Chapter 6. In this chapter, we will focus on the Planning (MPS & MRP) process and the Manufacturing part.

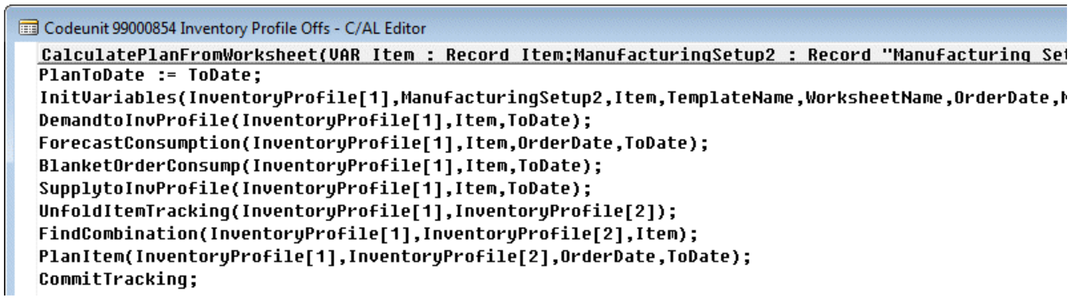
The Requisition Worksheet process allows us to create our own process to prepare data using custom settings to generate the worksheet lines and even to build our own planning algorithm in a new C/AL object that will create Requisition lines.

Inventory profile offsetting

The actual heart of the MRP calculation in Microsoft Dynamics NAV is Codeunit Inventory Profile Offsetting (99000854) which is called from Codeunit Calc. Item Plan - Plan Wksh. (5431) in our schema.

This Codeunit is not easy to understand and should only be changed by specialized developers after careful analysis.

The process uses the Inventory Profile buffer table during the calculation to build up information and starts with the function `CalculatePlanFromWorksheet`.



Codeunit 99000854 Inventory Profile Offs - C/AL Editor

```
CalculatePlanFromWorksheet(VAR Item : Record Item;ManufacturingSetup2 : Record "Manufacturing Se
PlanToDate := ToDate;
InitVariables(InventoryProfile[1],ManufacturingSetup2,Item,TemplateName,WorksheetName,OrderDate,M
DemandtoInvProfile(InventoryProfile[1],Item,ToDate);
ForecastConsumption(InventoryProfile[1],Item,OrderDate,ToDate);
BlanketOrderConsump(InventoryProfile[1],Item,ToDate);
SupplytoInvProfile(InventoryProfile[1],Item,ToDate);
UnfoldItemTracking(InventoryProfile[1],InventoryProfile[2]);
FindCombination(InventoryProfile[1],InventoryProfile[2],Item);
PlanItem(InventoryProfile[1],InventoryProfile[2],OrderDate,ToDate);
CommitTracking;
```

- **InitVariables**: This function is used to clear and initialize the variables used in this Codeunit.
- **DemandtoInvProfile**: Here the system creates records in the Inventory Profile table for Sales Orders, Service Orders, Production Orders that may require items.
- **ForecastConsumption**: If Use Forecast on Locations is used in the Manufacturing Setup, additional demand lines are created in the Inventory Profile Buffer based on the Production Forecast.
- **BlanketOrderConsump**: Additional demand is inserted for all Blanket Sales Orders with a Shipment Date and Outstanding Qty. within the calculation period.
- **SupplytoInvProfile**: The current inventory, purchase orders, and production orders are added to the Inventory Profile as possible supply.

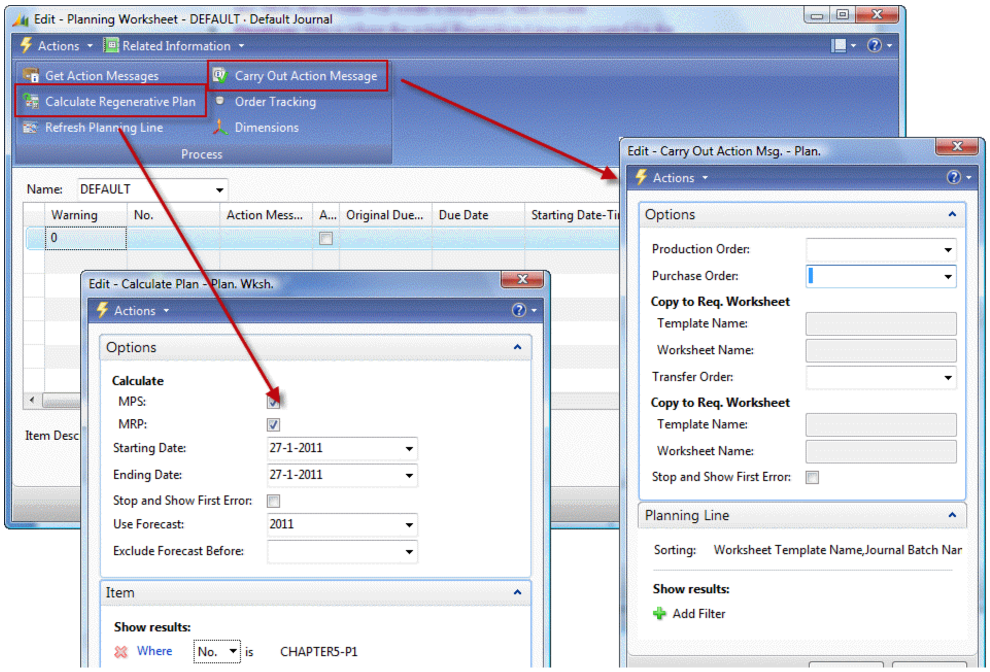
- **UnfoldItemTracking**: If the Item uses Item Tracking, this function makes sure that lot numbers and serial numbers will match.

> In this function, Microsoft developers use a trick that when using a temporary table with more dimensions the values in both tables are identical. This blog entry explains how this works. `http://dynamicsuser.net/blogs/mark_brummel/archive/2010/05/05/tip-27-using-temp-tables-in-arrays.aspx`

- **FindCombination**: This function creates temporary Stock Keeping Unit records for each SKU that requires replenishment. If the Item does not have any SKU, the system will create a temporary SKU record.
- **PlanItem**: This is where the actual Requisition Lines are created for the Item, based on the information in the Inventory Profile table and the setup.
- **CommitTracking**: This function saves information stored in temporary record variables to actual data in the database for Reservation Entries and Action Messages.

Calculating a plan

Let's run the Planning Worksheet for our English Desk and see what Planning Lines we get.



Name: DEFAULT

Warning	No.	Action Mess...	A...	Original Due...	Due Date	Starting Date-Ti
0			☐			

Edit - Calculate Plan - Plan. Wksh.

Calculate
MPS: ✓
MRP: ✓
Starting Date: 27-1-2011
Ending Date: 27-1-2011
Stop and Show First Error: ☐
Use Forecast: 2011
Exclude Forecast Before:

Item
Show results:
Where No. is CHAPTER5-P1

Edit - Carry Out Action Msg. - Plan.

Production Order:
Purchase Order:
Copy to Req. Worksheet
Template Name:
Worksheet Name:
Transfer Order:
Copy to Req. Worksheet
Template Name:
Worksheet Name:
Stop and Show First Error: ☐

Planning Line
Sorting: Worksheet Template Name,Journal Batch Nar

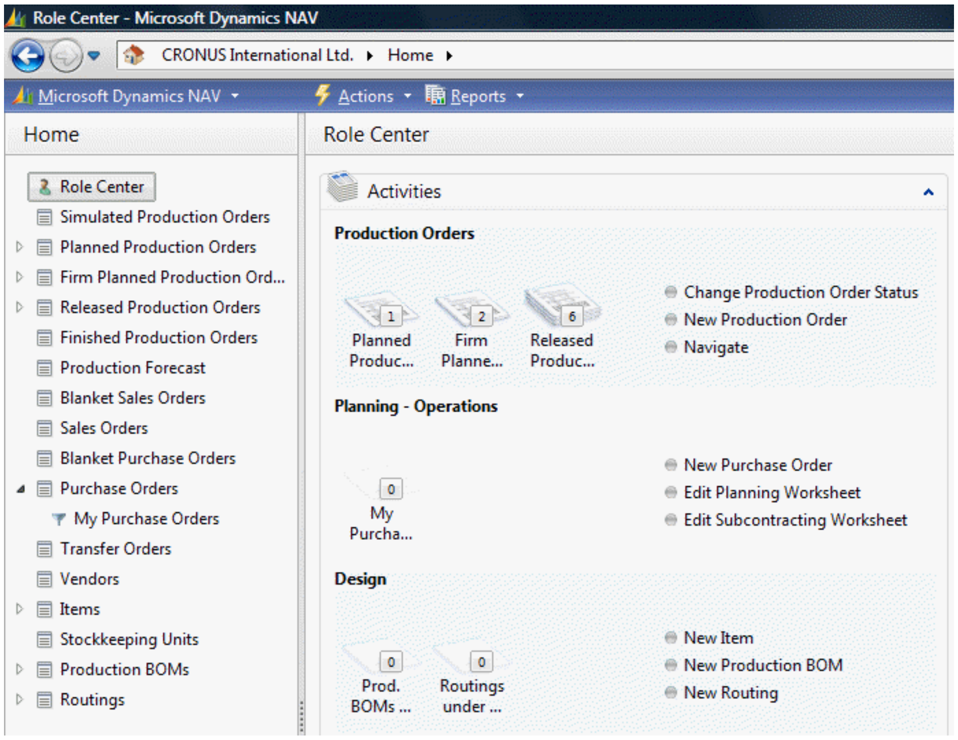
The MPS & MRP run is started from the Planning Worksheet. We need to enter a starting date and an ending date. In the CRONUS database, using our example, we can use the current Workdate.

When the MPS & MRP run is finished we can start the process to Carry Out Action Message to create the Production order.

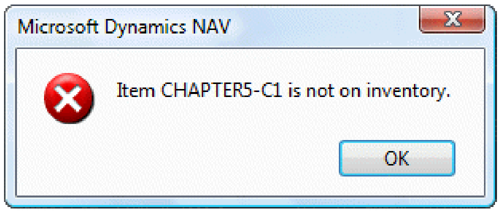
Production order workflow

The Production order is now created and ready to be started. The first status is **Planned** or **Firm Planned**. During the planned status Microsoft Dynamics NAV can automatically change the production order.

Once the production order is released, it can no longer be automatically changed.



To release a production order, the components need to be available. In our test scenario this is not yet true since we created new items which are not purchased.



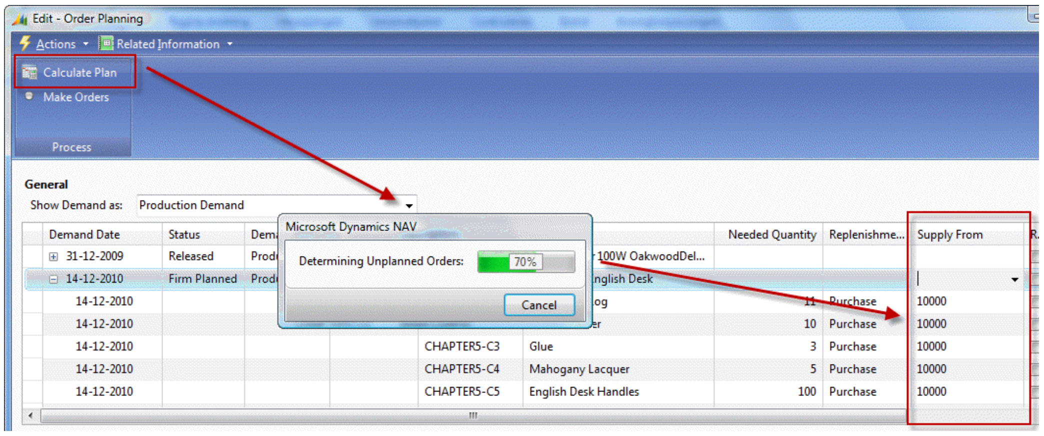
Microsoft Dynamics NAV

Item CHAPTER5-C1 is not on inventory.

Let's see how we can do this.

Purchase orders

To create the purchase orders, we'll use the Order Planning worksheet, to illustrate another method of planning. This will create Requisition Lines for the Production order we just released.



General

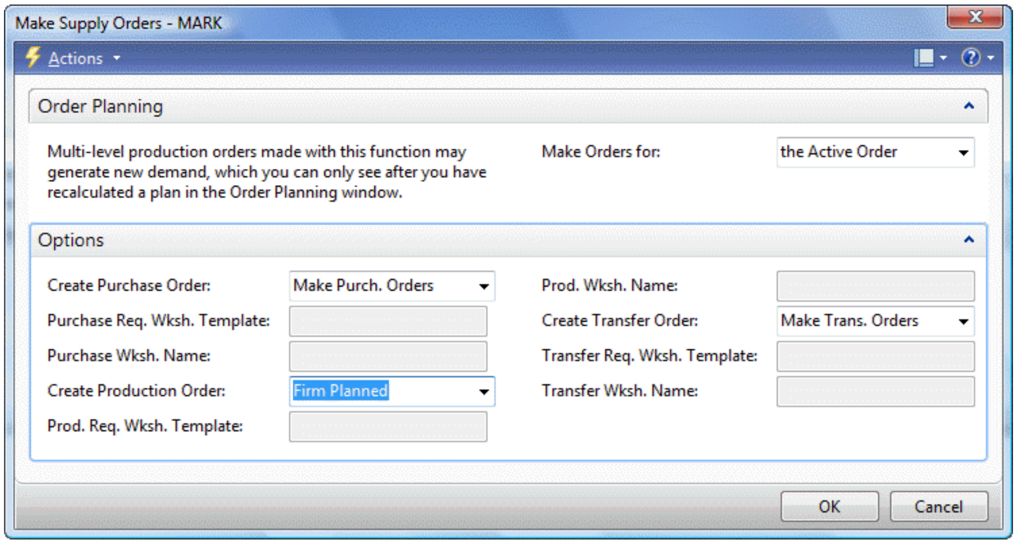
Show Demand as: Production Demand

Microsoft Dynamics NAV

Determining Unplanned Orders: 70%

Demand Date	Status	Dem				Needed Quantity	Replenishme...	Supply From
31-12-2009	Released	Prod			100W OakwoodDel...			
14-12-2010	Firm Planned	Prod			nglish Desk			
14-12-2010					og	11	Purchase	10000
14-12-2010					er	10	Purchase	10000
14-12-2010				CHAPTER5-C3	Glue	3	Purchase	10000
14-12-2010				CHAPTER5-C4	Mahogany Lacquer	5	Purchase	10000
14-12-2010				CHAPTER5-C5	English Desk Handles	100	Purchase	10000

Once the Requisition Lines are created, we need to specify a vendor number in the **Supply From** field and then start the **Make Orders** process to generate the **Purchase Orders**.


Make Supply Orders - MARK

Actions

Order Planning

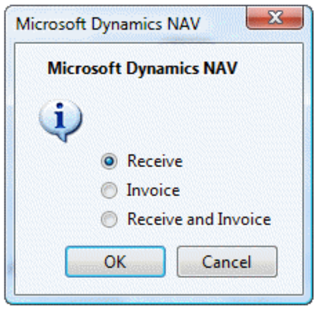
Multi-level production orders made with this function may generate new demand, which you can only see after you have recalculated a plan in the Order Planning window.

Make Orders for:	the Active Order		

Options

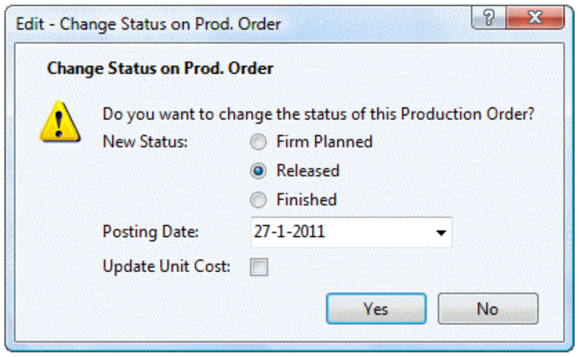
Create Purchase Order:	Make Purch. Orders	Prod. Wksh. Name:	
Purchase Req. Wksh. Template:		Create Transfer Order:	Make Trans. Orders
Purchase Wksh. Name:		Transfer Req. Wksh. Template:	
Create Production Order:	Firm Planned	Transfer Wksh. Name:	
Prod. Req. Wksh. Template:			

The Purchase Order can be received. This allows us to release the **Production Order**.


Microsoft Dynamics NAV

Microsoft Dynamics NAV

[✓ Receive] Invoice Receive and Invoice


Edit - Change Status on Prod. Order

Change Status on Prod. Order

Do you want to change the status of this Production Order?

New Status: Firm Planned [✓ Released] Finished

Posting Date: 27-1-2011

Update Unit Cost: ☐

Finishing production

When the Production Order is finished, the end product should be in inventory and the components should be consumed. This consumption process is called flushing.

This is done using the Consumption journal and Output Journal and can be done automatically and manually.

An alternative to the Consumption and Output Journal is the Production Journal that combines the functionality into one journal.

The Consumption Journal is automatically posted when the Flushing Method in the Item Card is set to `Forward`, `Backward` or `Pick + Forward`, `Pick + Backward`.

When using `Forward`, the Consumption Journal is posted when the Production order is released, `Backward` will post when the Production order is set to finished. `Pick + Forward` and `Pick + Backward` can be used in combination with Warehouse Management Locations.

Specialized production

The last production methodology we discussed in the beginning of the chapter is Specialized Production.

These are items produced in small numbers or items which have very different specifications each time they go into production.

For these companies, creating a Bill of Material each time an item changes is just too much work compared to the extra information they would get.

Jobs

Still these companies want to register their production orders and get a clear view of their inventory. Most companies doing this kind of production are using the Jobs functionality of Microsoft Dynamics NAV.

We will discuss this in *Chapter 8*, *Consultancy* where we will assemble custom made computer systems with different components as an example.

Kitting

In Microsoft Dynamics NAV version 5, the North American localized region introduced a granule called Kitting. Kitting is a term used to describe the sale of a combination of items (a Kit) as one Item.

The solution can be functionally positioned between the BOM Journal and full Manufacturing and uses technical components of both. One of the functional differences between the BOM Journal and Kitting is the support of Setup Resources.

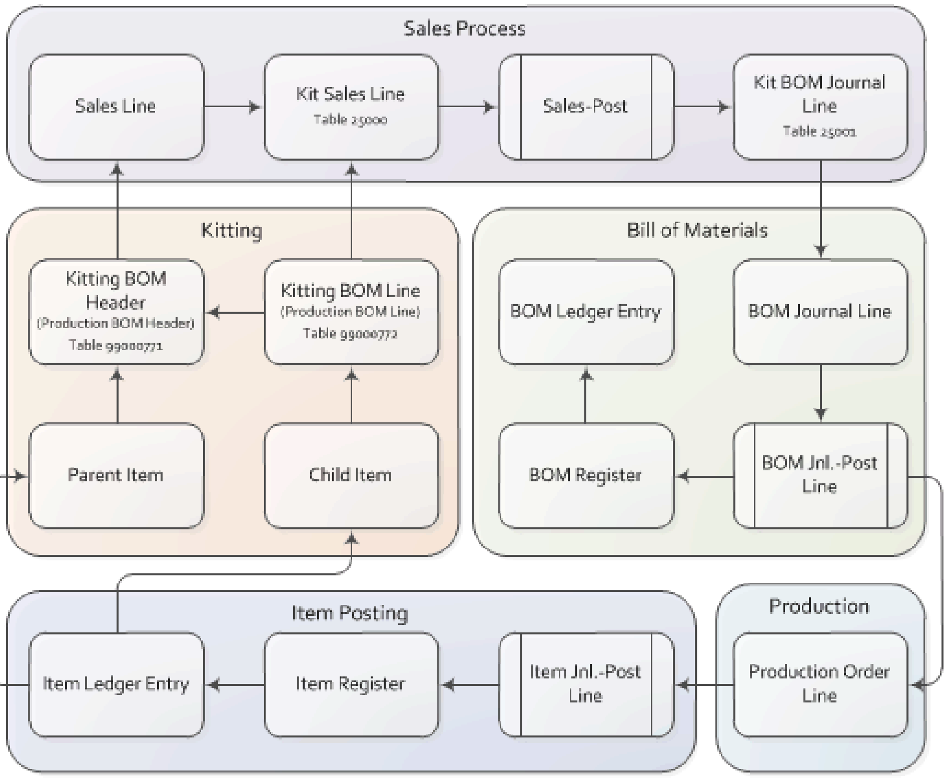
The Kitting BOM is defined in the same tables as the Production BOM. This table has been expanded with a field **Type** with options **Manufacturing** and **Kitting**. It is also possible in Kitting BOMs to use Resources and Setup Resources.

The posting is done using the BOM Journal table and Posting routine, however the posting routine has been expanded with the use of a dummy production order which exists for Kitting.

The Kitting objects can be identified within the numbers 25000 range.

[With the introduction of Kitting it is no longer possible to explode the classic Item BOM Components on a Sales or Purchase document.]

Let's look at the table relation and posting model of Kitting.



For reasons of simplicity the Resources and Value Entries have been removed from this schema. This works exactly the same as shown in the schema for the original BOM Journal functionality we drafted earlier in this chapter.

Sales process

Kitting takes place in the Sales process of Microsoft Dynamics NAV. Each Sales Line table has a corresponding table for kitting.

The Sales Post Codeunit (80) has been changed in order to create the Kit using the BOM Journal just before the original Item Journal Posting process.

```
TempJnlLineDim.DELETEALL;
TempDocDim.RESET;
TempDocDim.SETRANGE("Table ID",DATABASE::"Sales Line");
TempDocDim.SETRANGE("Line No.",SalesLine."Line No.");
DimMgt.CopyDocDimToJnlLineDim(TempDocDim,TempJnlLineDim);

IF SalesLine."Build Kit" AND (QtyToBeShipped <> 0) THEN
  PostBOMJnlLine(ItemJnlLine,SalesLine);

OriginalItemJnlLine := ItemJnlLine;
ItemJnlPostLine.RunWithCheck(ItemJnlLine,TempJnlLineDim);
```

Functionality has also been implemented to move the information stored in table Kit `Sales Line` (25001) to the corresponding posted information.

```
SalesShptLine.INSERT;
IF SalesShptLine."Build Kit" THEN BEGIN
  KitSalesLine.RESET;
  KitSalesLine.SETRANGE("Document Type",SalesLine."Document Type");
  KitSalesLine.SETRANGE("Document No.",SalesLine."Document No.");
  KitSalesLine.SETRANGE("Document Line No.",SalesLine."Line No.");
  IF KitSalesLine.FINDSET THEN
    REPEAT
      KitSalesShptLine.INIT;
      ...
      KitSalesShptLine.INSERT;
    UNTIL KitSalesLine.NEXT = 0;
END;
```

[More information about Kitting can be found in the functional and technical whitepapers by Microsoft.]

Kitting in Microsoft Dynamics NAV "7"

In the last statement of directions published by Microsoft it was announced that kitting will be introduced in the Worldwide version (W1).

This does not necessarily mean that the functionality and technical solution will be identical.

Vertical industry implementation

Microsoft Dynamics NAV is used in many different vertical industries. Vertical Industries often require specific features. Rather than trying to implement all these features in the standard product, Microsoft Dynamics NAV supports the framework and allows developers to design and create their own vertical features.

For these features the 80/20 rule applies, Microsoft delivers 80% of the framework which costs 20% of our time to implement. The missing 20% of the functionality is developed costing 80% of the budgeted time.

In this chapter, we will discuss how Microsoft Dynamics NAV is used for production in five different vertical industries. For each industry, we will discuss two specific vertical features and how they could be solved.

> Most industries have solid add-on solutions available which are designed by certified Microsoft Dynamics NAV partners that have been implemented at multiple sites. It is highly recommended to look at those add-on solutions instead of reinventing the wheel and rewrite an add-on that already exists.

Fashion

The general challenge in the Fashion industry is sizes and colors. Each Item can be produced and sold in sizes ranging from XXS to XXXL and in colors ranging from pink via orange to green while remaining the same item.

This calls for the creative use of variants which is heavily used by the available vertical solutions on the market.

Bill of materials

To use Manufacturing with variants, the Bill of Materials structure should be changed since this exists by default on item level. However each size uses different quantities of fabric and the different colors of fabric are often represented by other item number in the raw materials.

A solution for this might be to move the Bill of Materials from Item level to Stock Keeping Unit level. An SKU supports Variants for costing and inventory.

Shipping worksheet

Fashion companies produce items for a collection. Customers have the option to reserve on a collection in order for the production manager to determine how many to produce. Based on these numbers and an extra safety inventory, the production orders are created. Once the production orders are finished the company needs to decide who gets the first items. This can be best described as a reversed make-to-order mechanism.

To enable this in Microsoft Dynamics NAV, we could create a worksheet that will create lines for each combination of Production orders and Sales Orders. For each Sales Order that will be shipped, we could create a Warehouse Pick and Shipment from the Shipping Worksheet.

We will discuss Stock Keeping Units ,Warehouse Picks, and Shipments in Chapter 6.

Automotive

In the Automotive industry, Microsoft Dynamics NAV is mostly used by car manufacturing suppliers. These companies make prefabricated parts out of raw materials.

Tooling and amortization

In these companies, the **Production Part Approval Process** (**PPAP**) process as well as the tooling amortization is very important as the initial investments in tooling before the production process starts is high.

To support this, extra functionality needs to be developed for the tooling and BOM process. For example, the table Routing Tool (table 99000802) can be connected to a Fixed Asset (table 5600).

Item tracking

When something is wrong with a component of a car it is important to be able to see which other cars have the same components built by the same factory and tools using the same base materials.

In Microsoft Dynamics NAV it is possible to use a single Lot No. for a component or an end product and to trace this back. It is not possible to simply move the Lot No. of the component to the end product or copy information from the component's Lot No. such as a container no. or a quality code to the end product.

To support this we need to change the Item Tracking process. A good place to start would be the Item Journal where the Reservation Entry is moved to the Item Ledger Entry.

Medicines

When used by companies that manufacture medicines, using the Expiration Date for Lot Numbers correctly is highly important.

Lot numbers and expiration dates

In Microsoft Dynamics NAV expiration dates are defined in the Item Ledger Entries and the Warehouse Entries.

It is not possible to define a single Expiration date for a Lot. This can be changed by adding this field to the Lot No. Information table (6505). This table allows companies to predefine Lot numbers to be used in the production process.

By default, the Expiration Date is calculated based on the Document Date multiplied with the Expiration Calculation field in the Item table (27).

The Lot No. Information table can be used to save additional information about the specific production batch.

Quality control

Quality control is important in most production processes but maybe extra important when dealing with medicines. Usually a small part of a Lot is taken for quality control.

In Microsoft Dynamics NAV we can define Quality Measures in the Prod. Order Rtng Qlty Meas. Table (5413). However, these values are only saved as information for the Production Order.

To enhance quality control, we could add a document structure where a Quality Check document is created from a production order. The information should be saved in the Lot No. Information table.

When a Lot is not having the required quality a workflow should be started. This workflow will lead the user though a process where decisions can be made. Sometimes the quality can be improved and the items can still be used. Sometimes the Item number even depends on the quality of the product.

Quality control is in between the consumption journal and the output journal. During the final quality check the BOM items are used, but the final item is not yet available.

Food

In the food industry, everything is about expiration dates and fresh products. Inventory is never high and the rate of circulation is very high.

Zero inventory

For this reason it should be possible for fresh food companies to Zero the inventory of certain lot numbers once the expiration date is closing or expired.

This could normally be done using the Physical Inventory Journal. Doing this manually with Lot Numbers can be quite a job for someone to do every day so for this vertical solution we could create a function to do this. This function would create a Item Journal Line (83) with the field Phys. Inventory (56) activated and also create the Reservation Entry for Tracking and post the line automatically.

Ordering schedules

Fresh food companies use daily production processes that start on scheduled times. Each day, the factory starts the production process but the production numbers can be different based on the orders.

This can be done using the make-to-order policy but we need to make sure that there will be no new sales orders when the calculation process starts.

This can be achieved by creating an order schedule policy. New sales lines can be created for each item until a certain time, and when the time has elapsed the sales people will get an error message. This allows the production planner to start the calculation process at a fixed time each day, knowing the sales orders quantities can be trusted.

Furniture

The furniture industry is a large and very old industry that existed long before the industrial revolution and introduction of computers.

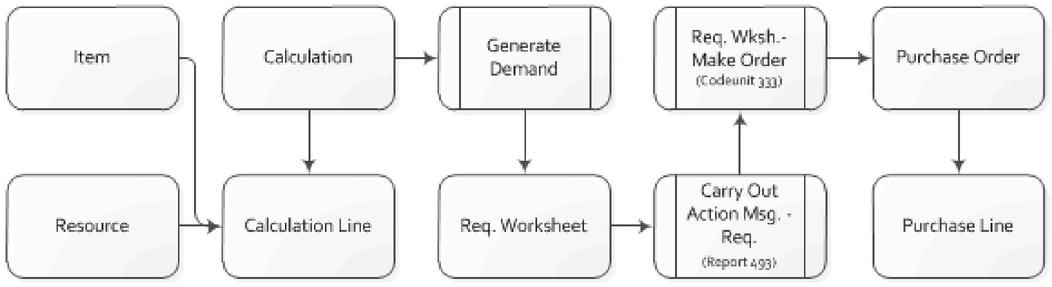
We can roughly split the furniture industry into two parts. The first part has moved production to be standardized using size and color matrixes which we can compare to the Fashion industry. Buying a table or kitchen, the customer can choose from different sizes and colors. Depending on the number of choices, the products are either made-to-stock (IKEA) or made-to-order.

The second part consists of furniture manufacturers who still produce custom made items. A desk or kitchen at these manufacturers can have any size or color. For these companies, it is next to impossible to create a Bill of Material for each custom item so they use pre-defined calculations with item categories.

For the examples in this book we will discuss the second category.

Calculations

Companies building custom made furniture need a possibility to calculate the use of materials and resources, both at item category and real item level. For this, we can create a calculation module with this data and posting model.



The basic structure of this calculation module is explained in Chapter 8 where we have combined this into the Jobs functionality of Microsoft Dynamics NAV.

Inventory

Our furniture company uses a combination of product specific items that are one-of-a kind and inventory items that are used in most of the products.

These items are combined into the end product and should be consumed when the product is finished. When the calculation module is integrated with Jobs for example, it would be possible to flush the components when the Job is completed. This functionality can be compared to the posting of the Consumption Journal when a production order is finished.

The inventory items can be updated weekly using the Physical Inventory Journal and inventory counting. This enables us to use the Requisition Worksheet and Reordering Policy we will discuss in the next chapter.

Summary

In this chapter, we discussed how three different production methodologies can be implemented in the Microsoft Dynamics NAV product.

We introduced the concepts of Item Tracking and Item Costing.

We have taken a tour through the Manufacturing process using different Requisition Worksheets and talked about the solution for MRP.

Finally, we looked at how production can be implemented for different vertical industries.

In the next chapter, we will take a closer look at the trade process in Microsoft Dynamics NAV.

6
Trade

In the previous chapter we discussed how Microsoft Dynamics NAV can help us to streamline our production process using both the standard application as well as customized solutions. We talked about five vertical industries and how to fit the application for them.

In this chapter we will discuss how to use Microsoft Dynamics NAV for these companies using sales and purchase documents, and how to integrate this with the built-in Warehouse Management and Reservation processes.

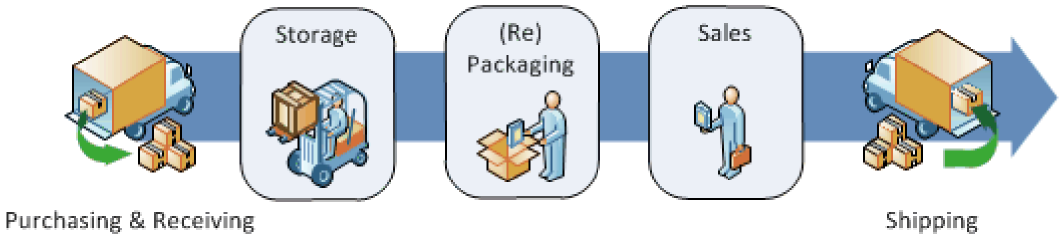
The primary focus of this chapter is how the application is designed, and where to go to change or enhance the design. Basic knowledge of how to create and process sales and purchase documents in Microsoft Dynamics NAV is a prerequisite.

We will use examples from the same vertical industries we discussed in the previous chapters; Automotive, Fashion, Medicines, Food, and Furniture.

After reading this chapter you will have a good understanding of how to implement Microsoft Dynamics NAV in trading companies.

The process

A trading company purchases and sells items without changing them. The main activities are purchase, storage, packaging, sales, and shipping (as shown in the following screenshot):



Managing inventory is very important in these companies. Having inventory is crucial for delivering on time and not having to say "no" to customers.

Wholesale versus retail

Traditionally, trading companies are divided into wholesale and retail companies. Wholesale companies sell to business and retail companies sell to consumers. Microsoft Dynamics NAV supports both and from the perspective of design (table and posting structure) there is not much difference.

The biggest difference between wholesale and retail for the application is the transaction volume. Where the total turnover of a wholesale company can be much higher compared to a retailer; the retailer often has more, smaller transactions. It can be a challenge from an application design perspective to retain a solution that performs well.

Another issue with high volume transaction systems is traceability of the data. Whenever something goes wrong it is very important to see where the error started and how much data was impacted by the mistake. In low transaction systems it is easier to find this manually.

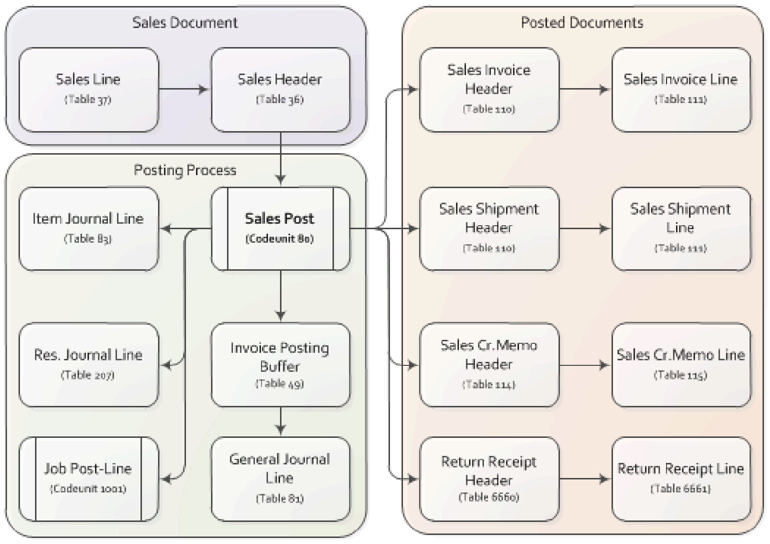
Sales and purchasing

Traditionally, salespersons are used to working with paper order forms. They would write down the customer name and address, and the items or services required.

In Microsoft Dynamics NAV the paper document is replaced by a sales and purchase document using a header for the general order information and lines to register the items and services.

The posting process breaks down the information in the document into the journals, and posts them so that the end user does not have to worry about this. The application reuses the same posting routines as we discussed in earlier chapters.

Let's look at how the documents and journals tie together by drawing the table and transaction scheme for this:


Sales Document
Sales Line (Table 37)
Sales Header (Table 36)
Posting Process
Item Journal Line (Table 83)
Sales Post (Codeunit 80)
Res. Journal Line (Table 207)
Invoice Posting Buffer (Table 49)
Job Post-Line (Codeunit 1001)
General Journal Line (Table 81)
Posted Documents
Sales Invoice Header (Table 110)
Sales Invoice Line (Table 111)
Sales Shipment Header (Table 110)
Sales Shipment Line (Table 111)
Sales Cr.Memo Header (Table 114)
Sales Cr.Memo Line (Table 115)
Return Receipt Header (Table 6660)
Return Receipt Line (Table 6661)

The first step is creating the document. When we create this **Sales Document** (**Sales Header** and **Sales Line**) nothing is posted. We are only entering information into the system that can be changed at any time.

When we start the **Sales Post (Codeunit 80)** the system will create all the journals for us and post them. When we sell an item, the system will create an **Item Journal Line**, when we sell a resource the system creates a **Resource Journal Line** and so on.

The **Invoice Posting Buffer** is used to create the entries in the **General Journal Line**. We already discussed this feature in *Chapter 3*, *Financial Management*.

Microsoft Dynamics NAV allows us to create four different kinds of posted sales documents: Invoices, Shipments, Credit Memos and Return Receipts. We will discuss all these types later in this chapter.

Transaction mirroring

The unique concept of sales and purchase in Microsoft Dynamics NAV is the mirroring of the transaction structure. Once we understand how the sales transactions fit together it is easy to also understand the structure of purchase.

Let's demonstrate this by comparing the first fields in the **Sales Line** (**37**) and **Purchase Line** (**39**) table (as shown in the following screenshot):

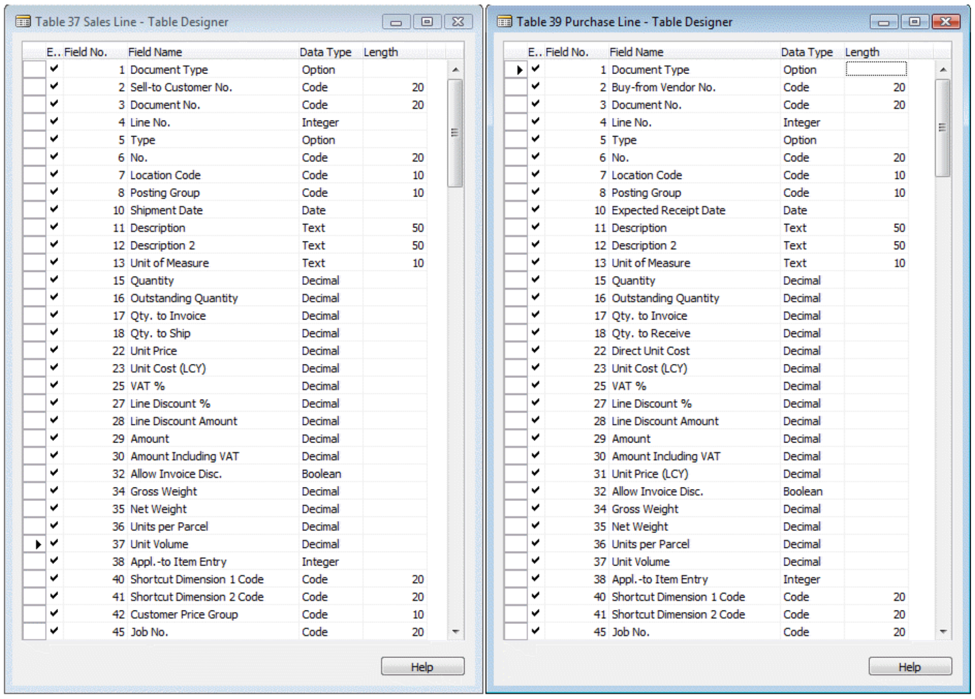


Table 37 Sales Line - Table Designer

E..	Field No.	Field Name	Data Type	Length
✓	1	Document Type	Option	
✓	2	Sell-to Customer No.	Code	20
✓	3	Document No.	Code	20
✓	4	Line No.	Integer	
✓	5	Type	Option	
✓	6	No.	Code	20
✓	7	Location Code	Code	10
✓	8	Posting Group	Code	10
✓	10	Shipment Date	Date	
✓	11	Description	Text	50
✓	12	Description 2	Text	50
✓	13	Unit of Measure	Text	10
✓	15	Quantity	Decimal	
✓	16	Outstanding Quantity	Decimal	
✓	17	Qty. to Invoice	Decimal	
✓	18	Qty. to Ship	Decimal	
✓	22	Unit Price	Decimal	
✓	23	Unit Cost (LCY)	Decimal	
✓	25	VAT %	Decimal	
✓	27	Line Discount %	Decimal	
✓	28	Line Discount Amount	Decimal	
✓	29	Amount	Decimal	
✓	30	Amount Including VAT	Decimal	
✓	32	Allow Invoice Disc.	Boolean	
✓	34	Gross Weight	Decimal	
✓	35	Net Weight	Decimal	
✓	36	Units per Parcel	Decimal	
✓	37	Unit Volume	Decimal	
✓	38	Appl.-to Item Entry	Integer	
✓	40	Shortcut Dimension 1 Code	Code	20
✓	41	Shortcut Dimension 2 Code	Code	20
✓	42	Customer Price Group	Code	10
✓	45	Job No.	Code	20

Table 39 Purchase Line - Table Designer

E..	Field No.	Field Name	Data Type	Length
✓	1	Document Type	Option	
✓	2	Buy-from Vendor No.	Code	20
✓	3	Document No.	Code	20
✓	4	Line No.	Integer	
✓	5	Type	Option	
✓	6	No.	Code	20
✓	7	Location Code	Code	10
✓	8	Posting Group	Code	10
✓	10	Expected Receipt Date	Date	
✓	11	Description	Text	50
✓	12	Description 2	Text	50
✓	13	Unit of Measure	Text	10
✓	15	Quantity	Decimal	
✓	16	Outstanding Quantity	Decimal	
✓	17	Qty. to Invoice	Decimal	
✓	18	Qty. to Receive	Decimal	
✓	22	Direct Unit Cost	Decimal	
✓	23	Unit Cost (LCY)	Decimal	
✓	25	VAT %	Decimal	
✓	27	Line Discount %	Decimal	
✓	28	Line Discount Amount	Decimal	
✓	29	Amount	Decimal	
✓	30	Amount Including VAT	Decimal	
✓	31	Unit Price (LCY)	Decimal	
✓	32	Allow Invoice Disc.	Boolean	
✓	34	Gross Weight	Decimal	
✓	35	Net Weight	Decimal	
✓	36	Units per Parcel	Decimal	
✓	37	Unit Volume	Decimal	
✓	38	Appl.-to Item Entry	Integer	
✓	40	Shortcut Dimension 1 Code	Code	20
✓	41	Shortcut Dimension 2 Code	Code	20
✓	45	Job No.	Code	20

The fields in both tables are equally numbered and serve the same process even though they use different terminology. For example, field 18, **Qty. to Ship** (sale) and **Qty. to Receive** (purchase).

Some fields are different because they don't make sense to be in both processes, for example, **Unit Price (LCY)** (field 31) in purchase and **Customer Price Group** (field 42) in sales.

The purchase process also uses the same posting methodology. The purchase header (38) and purchase line (39) tables are posted using codeunit purch.-post (90) into the purchase receipt, invoice, credit memo, and return shipment documents.

Let's have a closer look at the sales process.

Sales

The sales process supports six document types which are normalized into two tables, sales header (36) and sales line (37).

Each process can have their own number series, and has a special card and list page but they all share the same business logic.

Let's discuss the document types:

- **Quote**: When a customer would like to know the terms and conditions of making a purchase we can make a quote. This will show all the calculations such as pricing and VAT.
- **Blanket order**: This is a pre-order status. When used we have an agreement with the customer without knowing the exact shipment date.
- **Order**: This is used for the actual order document.
- **Invoice**: An invoice can be used in two ways. Directly, without a sales order if the company only invoices directly on G/L accounts, or we can use the invoices to invoice one or more shipments.
- **Credit memo**: We can use a credit memo when we credit on a G/L account.
- **Return order**: If a customer returns an item we can use a return order to reverse the inventory process.

Let's discuss these document types in more detail.

Orders

The main process is the order. The other document types are designed to support this.

Sales orders can be created directly or via a quote or blanket order. There are two differences between a quote and a blanket order.

1. Quotes can only be fully transferred into a sales order, not in parts. For example, a blanket order of 100 items can be split into 10 deliveries of 10 items with different shipping dates.
2. A customer with a quote has the option to say "yes" or "no". When they say "no", there will not be a transaction. Therefore, quotes are not used in the supply and demand calculation we discussed in the previous chapter. A blanket order is a real order. The customer should eventually purchase the complete quantity that was agreed. Therefore the blanket orders are used in the supply and demand calculation.

Quote and blanket order to order

Although the quotes, blanket orders, and orders are stored in the same table, the records are physically deleted from the table and inserted using another document type. This is done in codeunits sales-quote to order (86) and blanket sales order to order (87).

When comparing these codeunits in a compare tool such as Beyond Compare or Araxis, we can see that there is a lot of similarity. They both create a new sales order.

Quote to order

When moving a quote to an order the complete quote is copied and then deleted. A quote can be created from an opportunity in CRM as we discussed in *Chapter 4, Relationship Management*. Therefore the opportunity is updated when this happens.

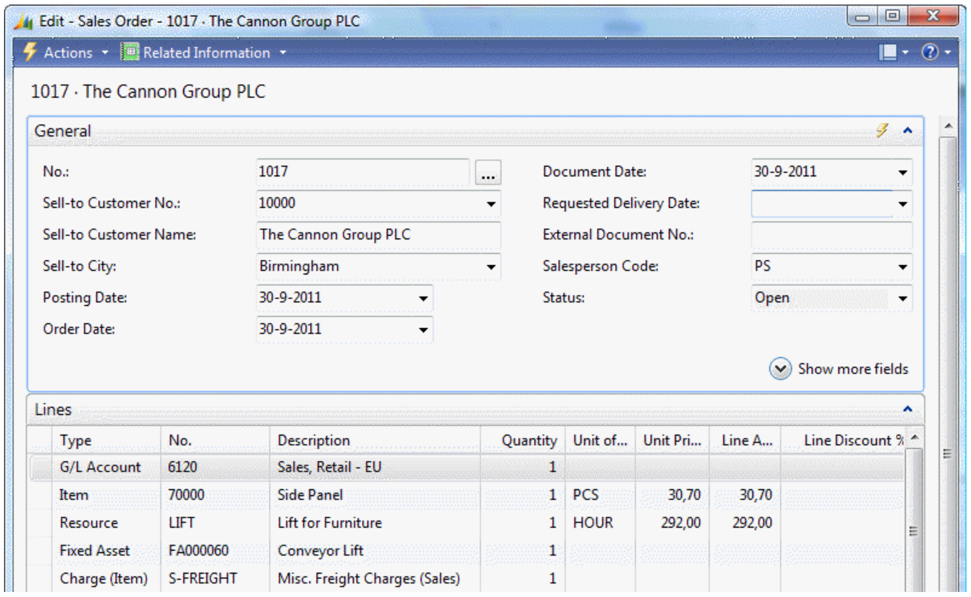
Blanket order to order

A blanket order can be moved in parts. Therefore business logic is implemented to calculate the remaining quantity. There is no link between blanket orders and CRM, and it is also not possible to create a blanket order from a quote.

Creating a new sales order

In order to understand the examples in the chapter we will discuss the most important fields of the sales order. A sales document contains one header and multiple lines.

While the sales header table contains more static registration of information, the sales line has more real business logic such as price calculation, inventory availability, and VAT. We will discuss how this business logic is normalized.


Edit - Sales Order - 1017 · The Cannon Group PLC
Actions
Related Information
1017 · The Cannon Group PLC
General
Show more fields
Lines

No.:	1017	Document Date:	30-9-2011
Sell-to Customer No.:	10000	Requested Delivery Date:	
Sell-to Customer Name:	The Cannon Group PLC	External Document No.:	
Sell-to City:	Birmingham	Salesperson Code:	PS
Posting Date:	30-9-2011	Status:	Open
Order Date:	30-9-2011		

Type	No.	Description	Quantity	Unit of...	Unit Pri...	Line A...	Line Discount %
G/L Account	6120	Sales, Retail - EU	1				
Item	70000	Side Panel	1	PCS	30,70	30,70	
Resource	LIFT	Lift for Furniture	1	HOUR	292,00	292,00	
Fixed Asset	FA000060	Conveyor Lift	1				
Charge (Item)	S-FREIGHT	Misc. Freight Charges (Sales)	1				

Sales header

All document types are uniquely numbered. The primary key fields of the sales header table are **Document Type** and **No**.

It is useful to use a number series code that makes sense to your end users. For example, SO09-0012 for sales order 12 in the year 2009 and SQ10-0312 for sales quote 312 for 2010.

The sales document contains two different customer no. Fields:

- **Sell-to Customer No.**: This is the primary customer no. field which defines the customer who requested the order to be created. This customer number is used to calculate the discounts.

- **Bill-to Customer No.**: By default the sell-to customer no. will also receive the invoice. By changing this field to another customer this will make the invoice to be printed with other customer details.

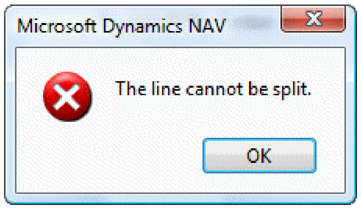
The sales document contains some dates that are used for different purposes:

- **Posting Date**: This date is used for posting to the various ledgers
- **Document Date**: This date is used for the accounts receivable
- **Shipment Date**: This date is for the calculation or the inventory availability
- **Due Date**: This date is the last date at which the bill-to customer is expected to pay the invoice

Sales lines

Each sales document can contain an almost unlimited number of sales lines. By default the sales lines are numbered 10000, 20000, 30000, and so on.

The numbering is done using the AutoSplitKey property on the sales line page and the increment cannot be changed. When a user inserts new records between two existing lines the program will calculate the new number to be exactly between the old values, for example 10000, 15000, 17500, 18750, 19375, 19687, 19843, 19921, 19960, 19980, 19990, 19995, 19997, 19998, 19999 ,20000. If there is no more room the system will generate a run-time error message (as shown in the following screenshot):



The line cannot be split.

Master data options

A sales line can contain a reference to six types of master data defined by the type field. These types are Text (blank option), G/L Account, Item, Resource, Fixed Asset, and Charge (Item).

The type that we specify here determines which journal will be used later when we post this sales document. However, each line can contain financial information which will be processed to the general ledger via the posting buffer table.

In the next chapter we will discuss how to add a new type to this process.

Sales line fields

To create a new sales line and start the important business logic in Microsoft Dynamics NAV we need to know about these fields:

- **Type**: This defines the master data type this sales line uses and eventually the journal that will be used during posting

> When the type field changes after the sales line was created the record is cleared and the fields get their default values.

- **No.**: This is the actual reference to the unique number of the master data type that is used

> When the No. field is changed, the previous Quantity is used to recreate the Sales Line with the new master data.

- **Quantity**: This is used to calculate the sales amounts for the invoicing and in case of an item also the physical quantity of the changes in inventory
- **Outstanding Quantity, Qty. to Invoice and Qty. to Ship**: These fields are designed to be used for partial shipping and invoicing of an order
- **Unit Price and Unit Cost (LCY)**: These fields are used to calculate the sales amount and profit
- **Line Discount % and Line Discount Amount**: These fields are used to determine the discounts

Validation flow

The sales line table has a specific validation flow of functions that is it important to understand before making changes to the table. This flow is based on the normal way an end user creates a sales line.

To create a sales line, only four fields are populated and the line is ready to use. After setting the type and choosing a no. the end user types in the quantity field and if necessary the unit price.

Let's analyze the C/AL code in the `OnValidate` trigger of the three fields that can calculate the sales line.

When changing these C/AL routines make sure to use the Test Near, Test Far, Do It, Clean up method we discussed in *Chapter 1, Introduction to Microsoft Dynamics NAV*.

No. | field 6

The C/AL code in the `OnValidate` trigger starts by doing the initial testing if the change is allowed.

After this the record is cleared and the old values for the no. field and quantity fields are applied.

```
TempSalesLine := Rec;
INIT;
Type := TempSalesLine.Type;
"No." := TempSalesLine."No.";
IF "No." = '' THEN
  EXIT;
IF Type <> Type::" " THEN
  Quantity := TempSalesLine.Quantity;
```

Then the Sales Line inherits the values from the Sales Header if required and the date fields are calculated.

```
"Sell-to Customer No." := SalesHeader."Sell-to Customer No.";
"Currency Code" := SalesHeader."Currency Code";
...
"Promised Delivery Date" := SalesHeader."Promised Delivery Date";
...
UpdateDates;
```

The sales header information is NOT present in the sales line when an end user picks a value for the no. field. We cannot use the customer information for table relations.

When this is done we see a `CASE` statement where the master data is acquired. This would be the place where we would move newly added fields from master data to the sales line table.

```
CASE Type OF
  Type::" ":
      ...
  Type::"G/L Account":
      ...
  Type::Item:
      ...
 Type::Resource:
      ...
  Type::"Fixed Asset":
      ...
  Type::"Charge (Item)":
      ...
END;
```

When this is done the quantities are calculated and the unit price is calculated.

```
IF Type <> Type::" " THEN BEGIN
  IF Type <> Type::"Fixed Asset" THEN
    VALIDATE("VAT Prod. Posting Group");
  VALIDATE("Unit of Measure Code");
  IF Quantity <> 0 THEN BEGIN
    InitOutstanding;
    IF "Document Type" IN ["Document Type"::"Return Order","Document 
Type"::"Credit Memo"] THEN
      InitQtyToReceive
    ELSE
      InitQtyToShip;
    UpdateWithWarehouseShip;
  END;
  UpdateUnitPrice(FIELDNO("No."));
END;
```

The latter is important for our analysis. After this function, other code is executed but this is not important for this example.

Quantity | field 15

Just like the no. field, the quantity field also first checks whether the change is allowed. When this is done this section of C/AL code is important:

```
IF Type = Type::Item THEN BEGIN
  UpdateUnitPrice(FIELDNO(Quantity));
  ...
  CheckApplFromItemLedgEntry(ItemLedgEntry);
END ELSE
  VALIDATE("Line Discount %");
```

In this C/AL code we should notice again the `UpdateUnitPrice` function and also the validation of the `Line Discount %` field.

Unit price | field 22

This field has little C/AL code. When changing the unit price manually the C/AL code will trigger the `Line Discount %` field.

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

Before going to this field, let's first have a look at the `UpdateUnitPrice` function we noticed earlier in the *Quantity* and *No.* field.

UpdateUnitPrice

The `UpdateUnitPrice` function executes this C/AL code:

```
IF (CalledByFieldNo <> CurrFieldNo) AND (CurrFieldNo <> 0) THEN
  EXIT;

GetSalesHeader;
TESTFIELD("Qty. per Unit of Measure");

CASE Type OF
  Type::Item,Type::Resource:
    BEGIN
      PriceCalcMgt.FindSalesLineLineDisc(SalesHeader,Rec);
      PriceCalcMgt.FindSalesLinePrice(SalesHeader,Rec,
        CalledByFieldNo);
    END;
END;
VALIDATE("Unit Price");
```

After doing the checks the sales price calculation routines we discussed in *Chapter 2, A Sample Application,* are executed. This is codeunit sales price calc. mgt. 7000.

When this is done it validates the field unit price which we already analyzed.

This leads us to one single point; the OnValidate trigger of `Line Discount %`.

Line discount % | field 27

The C/AL code in this OnValidate trigger first calculates the line discount amount based on the unit price, and then starts the function `UpdateAmounts`.

```
TestStatusOpen;
"Line Discount Amount" :=
  ROUND(
    ROUND(Quantity * "Unit Price",Currency."Amount Rounding
Precision") *
    "Line Discount %" / 100,Currency."Amount Rounding Precision");
"Inv. Discount Amount" := 0;
```

```
"Inv. Disc. Amount to Invoice" := 0;
UpdateAmounts;
```

UpdateAmounts

The `UpdateAmounts` function completes the creation of the sales line and this is where our quest ends.

The two most important other functions that are executed in this function are the `UpdateVATAmounts` for VAT calculation, and the credit limit check for the customer in `CustCheckCreditLimit.SalesLineCheck(Rec)`.

VAT calculation

The VAT calculation in Microsoft Dynamics NAV is not normalized in one application area but redeveloped everywhere. This makes VAT calculation one of the most complex application areas to make changes to.

VAT calculation is not only done in the sales line, purchase line, and general journal line table but also in more specific function tables such as service line.

It is therefore highly recommended not to change VAT calculation in Microsoft Dynamics NAV.

If VAT calculation is required in a customized solution it can be done using the general journal line as a temporary table. By populating the necessary fields and starting the calculation we can use the results without copying the VAT calculation to our own solution.

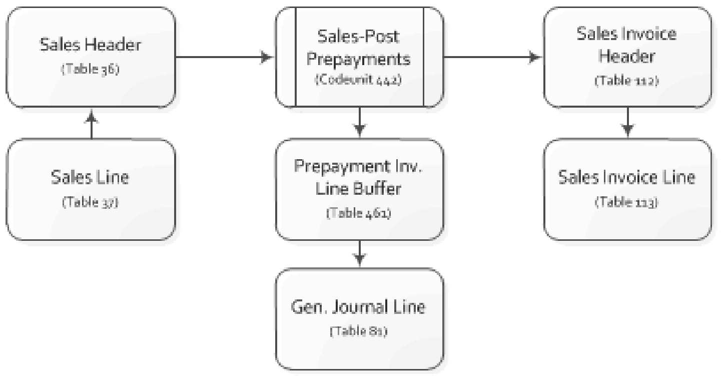
Invoicing

In Microsoft Dynamics NAV, a sales order can be shipped and invoiced directly from the document.

However, not all companies have a combined shipping and invoicing process. Some companies ship the goods first and send the invoice later; most of the time using combined invoicing.

Prepayments

Besides separating the invoice moment from the shipping moment Microsoft Dynamics NAV also allows for a prepayment process. This prepayment process is designed to work on top of the normal invoicing process. This means it does not replace the invoice but instead creates an extra invoice.


Sales Header (Table 36)
Sales Line (Table 37)
Sales-Post Prepayments (Codeunit 442)
Prepayment Inv. Line Buffer (Table 461)
Gen. Journal Line (Table 81)
Sales Invoice Header (Table 112)
Sales Invoice Line (Table 113)

This invoice is not created in codeunit sales-post (80) but in codeunit sales-post prepayments (461).

Using prepayments in Microsoft Dynamics NAV will always generate a minimum of two invoices per sales order

When the order is eventually invoiced the prepayment invoice is deducted from the invoice amount.

The design of this solution by Microsoft teaches us and demonstrates that to generate a posted sales invoice it is not specifically necessary to start codeunit sales-post 80.

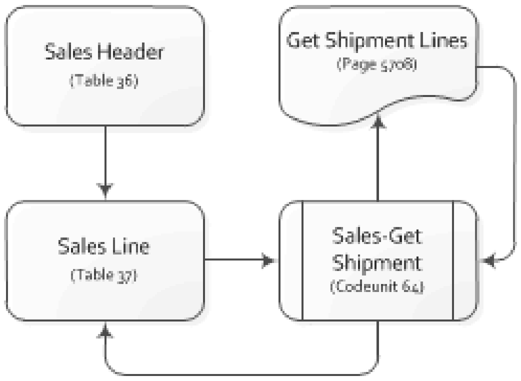
Combined invoicing

Combined invoicing of shipments can be done manually or using a batch report.

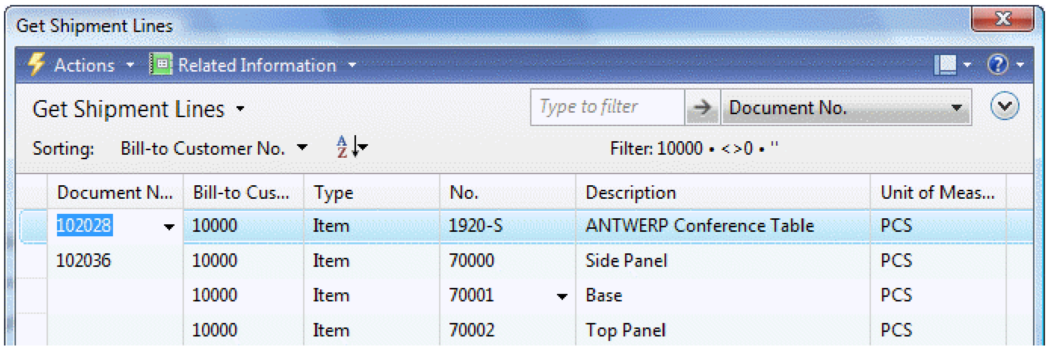
Manual

To manually combine shipments on a sales invoice we can use the **Sales-Get Shipment (Codeunit 64)**.

This codeunit can be started from the actions on a sales invoice subpage (47) and displays the sales shipment lines that are not yet completely invoiced.



The C/AL code however is not completely within the codeunit; the process starts in the codeunit and runs the page. The page then again starts a function in the codeunit.



Sorting: Bill-to Customer No.

Filter: 10000 • <>0 • ''

Document N...	Bill-to Cus...	Type	No.	Description	Unit of Meas...
102028	10000	Item	1920-S	ANTWERP Conference Table	PCS
102036	10000	Item	70000	Side Panel	PCS
	10000	Item	70001	Base	PCS
	10000	Item	70002	Top Panel	PCS

Batch

The Combine Shipments report (295) can be used to create one invoice for multiple shipments in batch.

It works in a similar way to the combine invoice report we created in *Chapter 2, An Example Application*.

The C/AL code that creates the Sales Line for the Invoice is normalized and used in both the Codeunit Sales-Get Shipment (64) and Combine Shipments report (295). The function is located in the Sales Shipment Line table (111) and is called `InsertInvLineFromShptLine`.

To enable Combined Shipments, the Boolean field Combine Shipments (87) should be set to **Yes** in the Customer table. This value is inherited into the Sales Header for the Sales Order document.

Credit Memo and Return Orders

The Credit Memo and Return Order Document Types are used for reversing the order process.

Purchasing

Before we can ship the items we sold we first need to purchase or produce them. We discussed the production process in the previous chapter so let's focus on the Purchasing process.

Technically, the Sales and Purchase process are mirrored transactions and the application design is similar. The Purchase Header table has the same Document Types Quote, Order, Invoice, Credit Memo, Blanket Order, and Return Order, and the same posting process.

So instead of going into the similarities we will discuss the differences.

Resources

In Microsoft Dynamics NAV it is not possible to purchase resources. When we take a closer look at the Type field (5) we can see that the option is left blank.

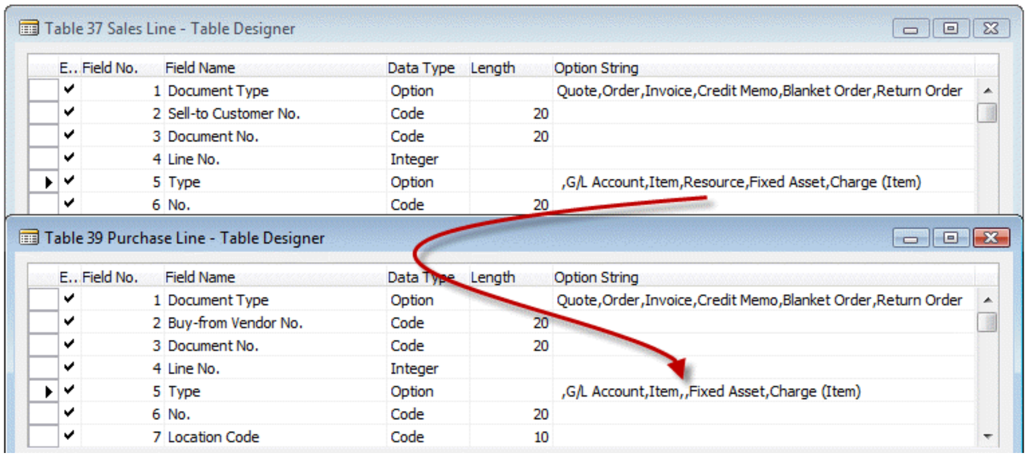


Table 37 Sales Line - Table Designer

E..	Field No.	Field Name	Data Type	Length	Option String
✓	1	Document Type	Option		Quote,Order,Invoice,Credit Memo,Blanket Order,Return Order
✓	2	Sell-to Customer No.	Code	20	
✓	3	Document No.	Code	20	
✓	4	Line No.	Integer		
✓	5	Type	Option		,G/L Account,Item,Resource,Fixed Asset,Charge (Item)
✓	6	No.	Code	20	

Table 39 Purchase Line - Table Designer

E..	Field No.	Field Name	Data Type	Length	Option String
✓	1	Document Type	Option		Quote,Order,Invoice,Credit Memo,Blanket Order,Return Order
✓	2	Buy-from Vendor No.	Code	20	
✓	3	Document No.	Code	20	
✓	4	Line No.	Integer		
✓	5	Type	Option		,G/L Account,Item,,Fixed Asset,Charge (Item)
✓	6	No.	Code	20	
✓	7	Location Code	Code	10	

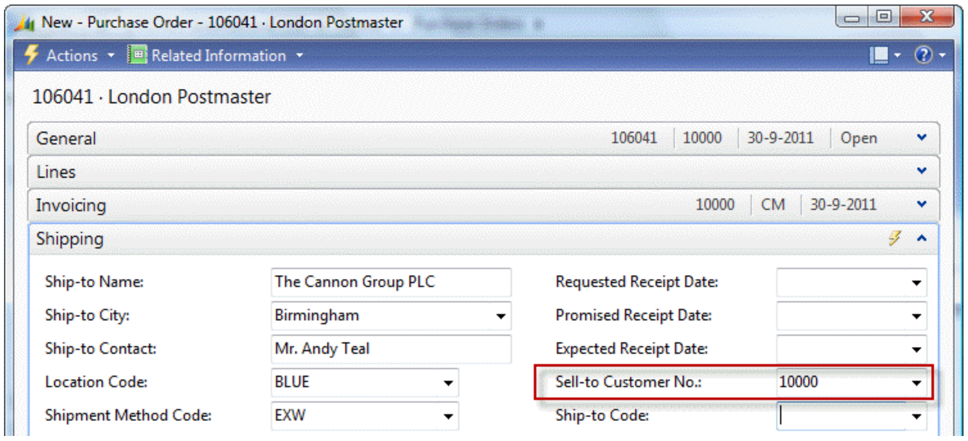
Drop shipments

When selling items which are not on inventory it is possible to purchase the items from a vendor and have them directly shipped to the customer. This process is called **Drop shipments**.

This process can be handled manually, or by using the requisition worksheet.

Manual

To create a drop shipment manually the purchase order should first be created using the **Sell-to Customer No.** from the corresponding Sales Order as the Shipping Address.



106041 · London Postmaster

General: 106041 | 10000 | 30-9-2011 | Open

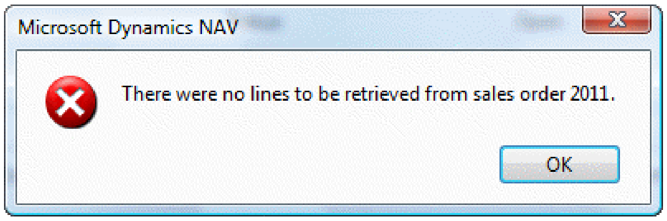
Invoicing: 10000 | CM | 30-9-2011

Ship-to Name:	The Cannon Group PLC	Requested Receipt Date:	
Ship-to City:	Birmingham	Promised Receipt Date:	
Ship-to Contact:	Mr. Andy Teal	Expected Receipt Date:	
Location Code:	BLUE	Sell-to Customer No.:	10000
Shipment Method Code:	EXW	Ship-to Code:	

When this is done we can start the Codeunit Purch.-Get Drop Shpt. (76) from the Actions on the Purchase Order.

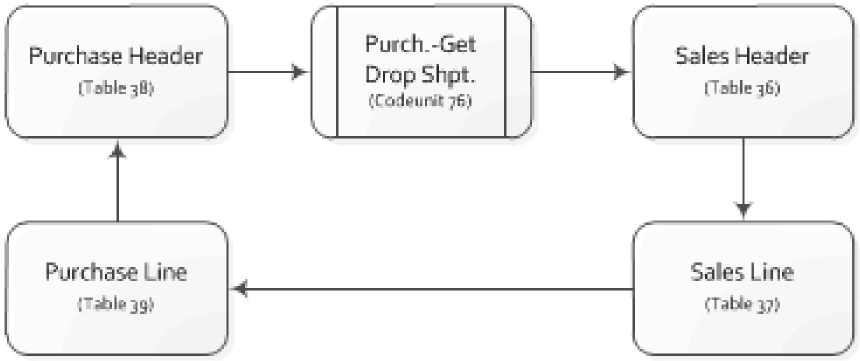
This function will show a list of all Sales Orders for this **Sell-to Customer No.** regardless if drop shipment is possible.

If we select a Sales Order without Sales Lines that are marked for Drop Shipment we get this error message:



There were no lines to be retrieved from sales order 2011.

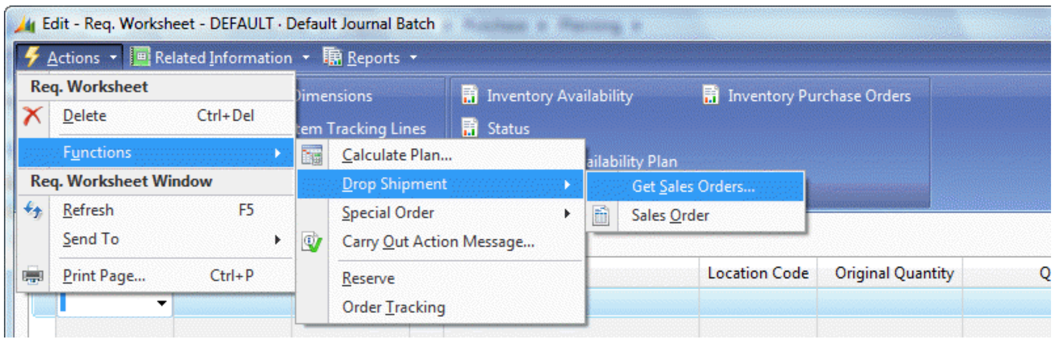
After retrieving the sales information the Sales Line and Purchase Line table are connected to each other by populating the Purchase Order No., Purch. Order Line No., Sales Order No., and Sales Order Line No. fields.



These fields are numbered 71 & 72 in the Sales Line and Purchase Line table.

Requisition worksheet

We introduced the Requisition Worksheet in the previous chapter when we discussed the planning process. The Requisition Worksheet can also be used for the Drop Shipment functionality.



This will start report Get Sales Orders (698) which will filter on all Sales Lines marked for Drop Shipment and creates a line in the Requisition Worksheet table.

This line can be processed by carrying out the action messages. This function will also connect the Sales Order to the Purchase Order using fields 71 & 72.

The C/AL code for manual Drop Shipment and using the Requisition Worksheet is not normalized. This means that changes made to one method should also be made in the other method, and maintained twice.

Document releasing and approval process

Within the Sales and Purchase document process there is a workflow available for releasing and approving a document.

This is taken care of by a single status field and two processes.

Status

The Status field (120) in the Sales header and Purchase header table indicates this. There are four options: Open, Released, Pending Approval, and Pending Prepayment.

Two of these status fields are mandatory to use, Open and Released. Pending Approval and Pending Prepayment are optional.

We have already discussed prepayments earlier in this chapter.

Releasing a document

Before a document can be posted it is mandatory to release it. This is done by codeunits Release Sales Document (414) and Release Purchase Document (415). These codeunits are, as you may have guessed, almost identical.

The codeunit performs a number of tests before setting the status to Released.

Let's discuss some of these checks.

```
TESTFIELD("Sell-to Customer No.");
```

A typical example of Test Near, the Customer number should not be blank.

```
SalesLine.SETRANGE("Document Type","Document Type");
SalesLine.SETRANGE("Document No.","No.");
SalesLine.SETFILTER(Type,'>0');
SalesLine.SETFILTER(Quantity,'<>0');
IF NOT SalesLine.FIND('-') THEN
  ERROR(Text001,"Document Type","No.");
```

There should be at least one Sales Line with a Quantity.

When the testing is done, some final calculations are implemented. These calculations are document calculations that span over the individual Sales Lines.

```
SalesSetup.GET;
IF SalesSetup."Calc. Inv. Discount" THEN BEGIN
  CODEUNIT.RUN(CODEUNIT::"Sales-Calc. Discount",SalesLine);
  GET("Document Type","No.");
END;
```

This codeunit calculates the Invoice Discount.

```
SalesLine.SetSalesHeader(Rec);
SalesLine.CalcVATAmountLines(0,Rec,SalesLine,TempVATAmountLine0);
SalesLine.CalcVATAmountLines(1,Rec,SalesLine,TempVATAmountLine1);
SalesLine.UpdateVATOnLines(0,Rec,SalesLine,TempVATAmountLine0);
SalesLine.UpdateVATOnLines(1,Rec,SalesLine,TempVATAmountLine1);
```

At the end of the releasing process the VAT calculation is completed.

Releasing a document also calculates the Amount and Amount Including VAT fields on the Sales Line.

Manual versus automatic releasing

By default, Microsoft Dynamics NAV releases the document automatically. The posting codeunits Sales-Post (80) and Purchase-Post(90) contain this C/AL code.

```
IF (Status = Status::Open) OR (Status = Status::"Pending Prepayment")
THEN BEGIN
  TempInvoice := Invoice;
  TempShpt := Ship;
  TempReturn := Receive;
  CODEUNIT.RUN(CODEUNIT::"Release Sales Document",SalesHeader);
  Status := Status::Open;
  Invoice := TempInvoice;
  Ship := TempShpt;
  Receive := TempReturn;
  MODIFY;
  COMMIT;
  Status := Status::Released;
END;
```

This code temporarily releases the document by starting the Release codeunit and then sets the status back to Open, modifies the records and commits the transaction. Then the status is set to Released.

Whenever there is an error afterwards the status will still be Open since that was the status before the `COMMIT`.

Document approval

On top of the release process is a Document Approval workflow. This feature is designed to work on top of the functionality we already discussed and is optional.

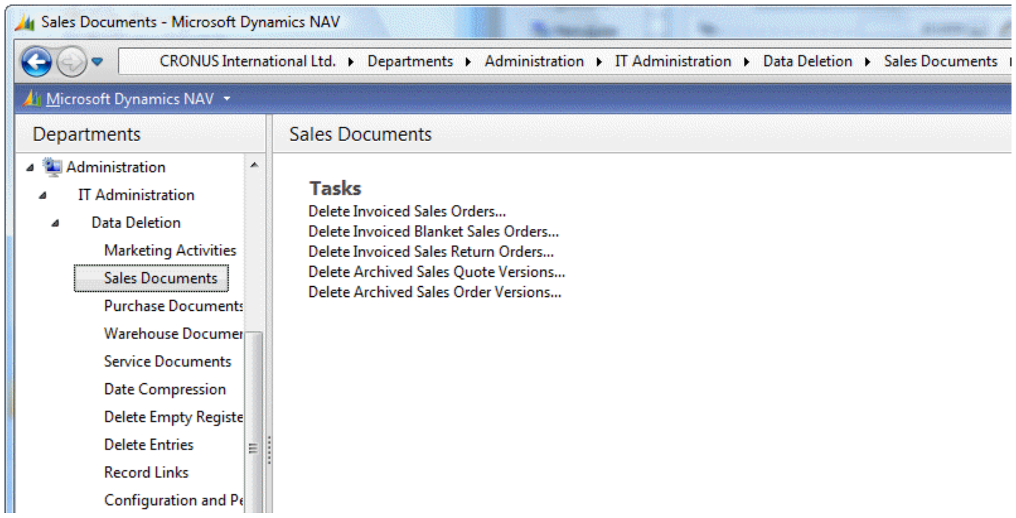
> Microsoft provides a good whitepaper on Document Approval on `https://mbs.microsoft.com/downloads/customer/nav/general/NAV5_0_SalesDocFactSheet_A4_July07.pdf`.

Deleting sales and purchase documents

During the life cycle of our application many documents will be created. There might come a day when this exceeds the point where doing some maintenance is required.

Data deletion

In the **IT Administration** part of the Departments Role Center we can find a **Data Deletion** section which is designed for IT administrators to use to clean up data.

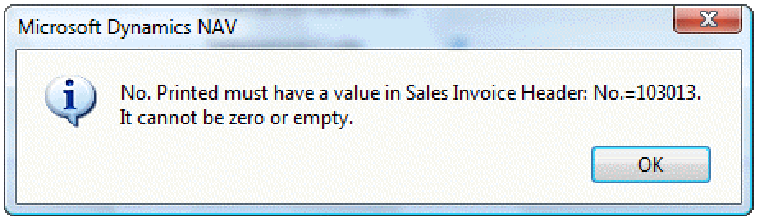

Sales Documents - Microsoft Dynamics NAV
CRONUS International Ltd. ▸ Departments ▸ Administration ▸ IT Administration ▸ Data Deletion ▸ Sales Documents
Microsoft Dynamics NAV
Departments
Administration
IT Administration
Data Deletion
Marketing Activities
Sales Documents
Purchase Document
Warehouse Documer
Service Documents
Date Compression
Delete Empty Registe
Delete Entries
Record Links
Configuration and P
Sales Documents
Tasks
Delete Invoiced Sales Orders...
Delete Invoiced Blanket Sales Orders...
Delete Invoiced Sales Return Orders...
Delete Archived Sales Quote Versions...
Delete Archived Sales Order Versions...

When a sales order is invoiced using Get Shipment Lines or Combined Invoicing, the sales order is NOT automatically deleted, nor are completely handled blanket orders.

Leaving old orders in the database may lead to large tables. Since these Document tables are heavily inserted and modified throughout the working day by many people, this may lead to unnecessary overhead in the database.

Deletion of shipments and invoices

Microsoft Dynamics NAV allows users to delete posted shipments and invoices when they are printed.



No. Printed must have a value in Sales Invoice Header: No.=103013. It cannot be zero or empty.

Although it should be considered carefully, it might be necessary for some companies to periodically clean up this data. Most companies never look at the shipments once the items are delivered to their customers.

Cleaning up these tables will have a positive impact on the performance and maintainability of your system if it reaches the size of roughly 50-100GB.

While designing business analysis reports, NEVER use data from the Sales Shipment Header or Line table as they might get deleted. Always use the Ledger Entry tables instead.

Document tables and row level locking

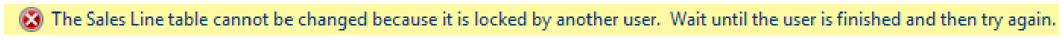
With the introduction of the SQL Server option for Microsoft Dynamics NAV Row-Level Locking was introduced. The classic database supports Table Level Locking only.

Record Level Locking does not benefit concurrency in transactions as Microsoft Dynamics NAV isolates them as we will demonstrate in *Chapter 10, Application Design*.

However, it will have an impact for creation of documents as there is no transaction involved. Or so it seems.

Range locks in documents

Even if end-users work on different documents they still might experience this error message every now and then:



The Sales Line table cannot be changed because it is locked by another user. Wait until the user is finished and then try again.

This is caused by Range locking in the SQL Server database.

Range locking is a special SQL Server locking mechanism which is always used by SQL Server if a query does not exactly specify the clustered index.

Let's illustrate this with an example from the standard application.

UpdateVATOnLines

The Sales Line table contains a VAT function that is used whenever a sales document is released.

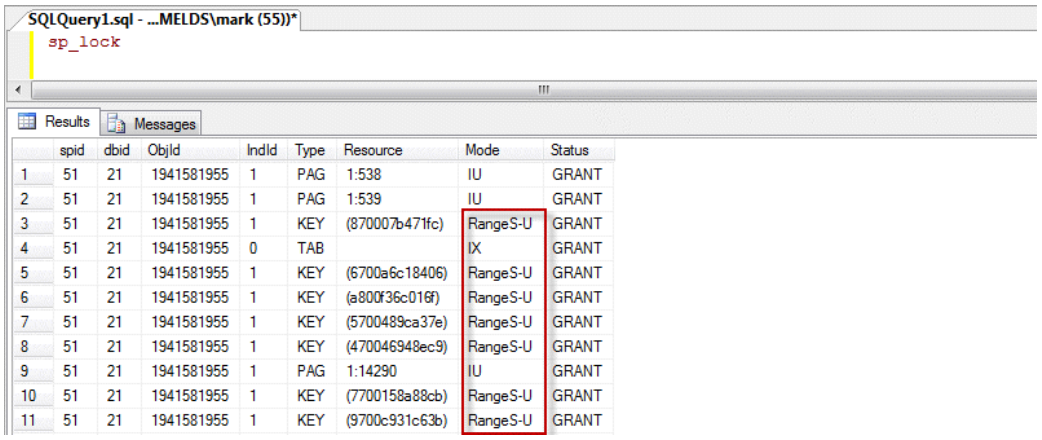
```
IF QtyType = QtyType::Shipping THEN
  EXIT;
IF SalesHeader."Currency Code" = '' THEN
  Currency.InitRoundingPrecision
ELSE
  Currency.GET(SalesHeader."Currency Code");

TempVATAmountLineRemainder.DELETEALL;

WITH SalesLine DO BEGIN
  SETRANGE("Document Type",SalesHeader."Document Type");
  SETRANGE("Document No.",SalesHeader."No.");
  LOCKTABLE;
  IF FINDSET THEN
    REPEAT
```

In this example the sales line table is filtered on the `Document Type` and `Document No.` field which is not the complete primary key.

This will result in an SQL statement looking like this:

```
SELECT TOP 51 * FROM "CRONUS International Ltd_$Sales Line" WITH
(UPDLOCK)
  WHERE (("Document Type"=1)) AND (("Document No_"='6004'))
ORDER BY "Document Type","Document No_","Line No_"
```

This will issue a range lock that we can check with the SQL `sp_lock` command.



SQLQuery1.sql - ...MELDS\mark (55))*

```
sp_lock
```

	spid	dbid	ObjId	IndId	Type	Resource	Mode	Status
1	51	21	1941581955	1	PAG	1:538	IU	GRANT
2	51	21	1941581955	1	PAG	1:539	IU	GRANT
3	51	21	1941581955	1	KEY	(870007b471fc)	RangeS-U	GRANT
4	51	21	1941581955	0	TAB		IX	GRANT
5	51	21	1941581955	1	KEY	(6700a6c18406)	RangeS-U	GRANT
6	51	21	1941581955	1	KEY	(a800f36c016f)	RangeS-U	GRANT
7	51	21	1941581955	1	KEY	(5700489ca37e)	RangeS-U	GRANT
8	51	21	1941581955	1	KEY	(470046948ec9)	RangeS-U	GRANT
9	51	21	1941581955	1	PAG	1:14290	IU	GRANT
10	51	21	1941581955	1	KEY	(7700158a88cb)	RangeS-U	GRANT
11	51	21	1941581955	1	KEY	(9700c931c63b)	RangeS-U	GRANT

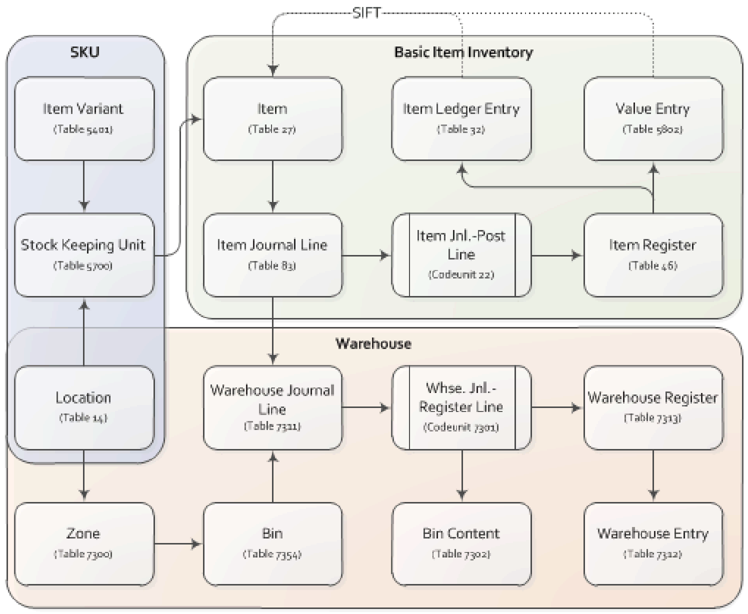
More information on range locks and workarounds can be found on this blog: `http://dynamicsuser.net/blogs/mark_brummel/archive/2009/06/18/tip-7-sql-range-locks.aspx`.

Inventory management

In Microsoft Dynamics NAV inventory is kept for items on locations using item ledger entries and value entries.

On top of this we can use Stock Keeping Units (SKU) to have different inventory settings per item, location, and variant.

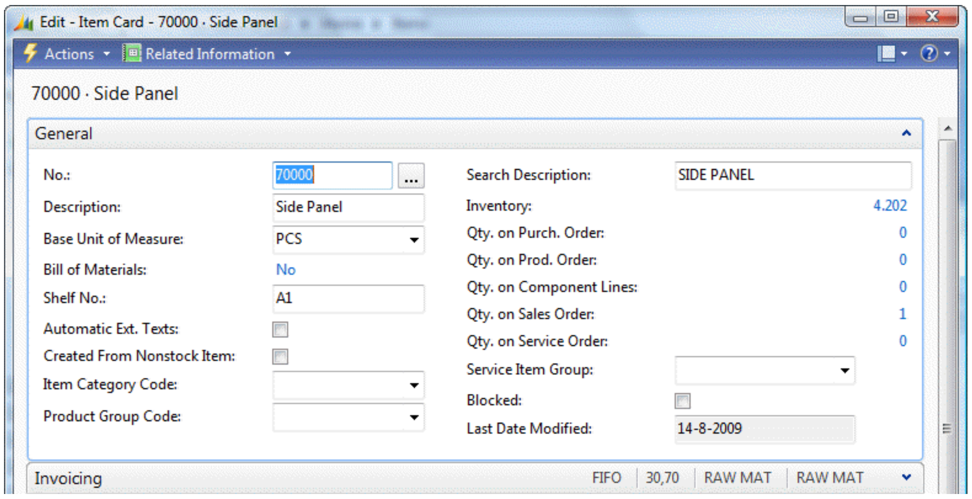
Let's start by looking at the table and posting model of inventory in Microsoft Dynamics NAV:


SIFT
SKU
Item Variant (Table 5401)
Stock Keeping Unit (Table 5700)
Location (Table 14)
Basic Item Inventory
Item (Table 27)
Item Ledger Entry (Table 32)
Value Entry (Table 5802)
Item Journal Line (Table 83)
Item Jnl.-Post Line (Codeunit 22)
Item Register (Table 46)
Warehouse
Warehouse Journal Line (Table 7311)
Whse. Jnl.-Register Line (Codeunit 7301)
Warehouse Register (Table 7313)
Zone (Table 7300)
Bin (Table 7354)
Bin Content (Table 7302)
Warehouse Entry (Table 7312)

Keeping inventory can be extended with the use of warehouse management. This is designed to run on top of the **Basic Item Inventory** functionality.

Items

The Item table hosts the Master Data for Inventory Management like a G/L Account does for Financial Management.



70000 · Side Panel

General

No.:	70000	Search Description:	SIDE PANEL
Description:	Side Panel	Inventory:	4.202
Base Unit of Measure:	PCS	Qty. on Purch. Order:	0
Bill of Materials:	No	Qty. on Prod. Order:	0
Shelf No.:	A1	Qty. on Component Lines:	0
Automatic Ext. Texts:	☐	Qty. on Sales Order:	1
Created From Nonstock Item:	☐	Qty. on Service Order:	0
Item Category Code:		Service Item Group:	
Product Group Code:		Blocked:	☐
		Last Date Modified:	14-8-2009

Invoicing FIFO 30,70 RAW MAT RAW MAT

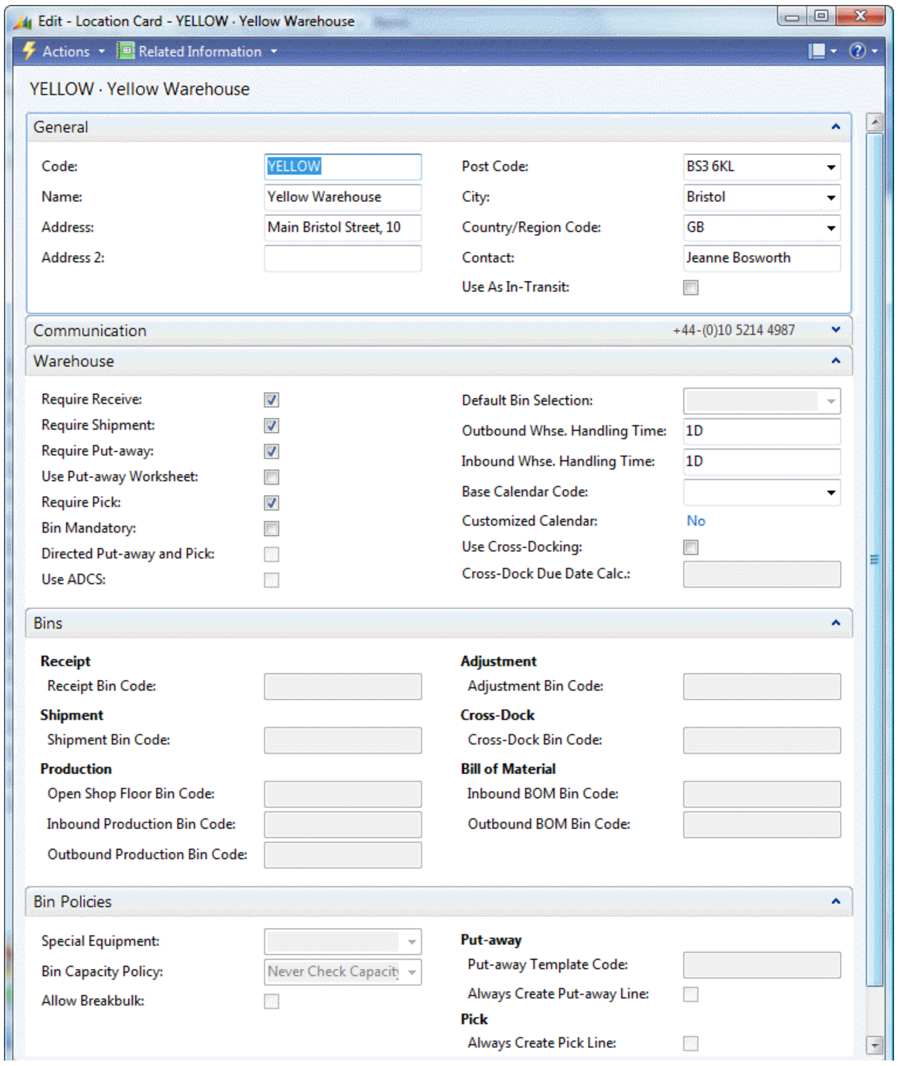
In this table we can do the setup for each individual item, including pricing, inventory and production strategies, and tracking options.

Locations

The Location table defines which level of inventory management is done.

The location can either be a physical warehouse somewhere or a part of a warehouse if one warehouse uses different warehouse strategies.

If we look at the **Location Card** we see what we can set up:



YELLOW · Yellow Warehouse

General

Code:	YELLOW	Post Code:	BS3 6KL
Name:	Yellow Warehouse	City:	Bristol
Address:	Main Bristol Street, 10	Country/Region Code:	GB
Address 2:		Contact:	Jeanne Bosworth
		Use As In-Transit:	☐

Communication +44-(0)10 5214 4987

Warehouse

Require Receive:	✓	Default Bin Selection:	
Require Shipment:	✓	Outbound Whse. Handling Time:	1D
Require Put-away:	✓	Inbound Whse. Handling Time:	1D
Use Put-away Worksheet:	☐	Base Calendar Code:	
Require Pick:	✓	Customized Calendar:	No
Bin Mandatory:	☐	Use Cross-Docking:	☐
Directed Put-away and Pick:	☐	Cross-Dock Due Date Calc.:	
Use ADCS:	☐		

Bins

Receipt		**Adjustment**	
Receipt Bin Code:		Adjustment Bin Code:	
Shipment		**Cross-Dock**	
Shipment Bin Code:		Cross-Dock Bin Code:	
Production		**Bill of Material**	
Open Shop Floor Bin Code:		Inbound BOM Bin Code:	
Inbound Production Bin Code:		Outbound BOM Bin Code:	
Outbound Production Bin Code:			

Bin Policies

Special Equipment:		**Put-away**	
Bin Capacity Policy:	Never Check Capacity	Put-away Template Code:	
Allow Breakbulk:	☐	Always Create Put-away Line:	☐
		Pick	
		Always Create Pick Line:	☐

Let's see these settings in detail:

- **General**: Here we can specify the physical location of the warehouse. We can also specify **Use As In-Transit**. When this is specified we can only use transfer orders to move inventory to this location.
- **Warehouse**: On this tab we specify which level of warehouse management functionality we want to use. If everything is left blank, no warehouse entries are created when this location is used.
- **Bins**: This tab contains the default bins for most inventory activities such as **Receipt**, **Shipment**, and so on. These values can be changed when creating the warehouse documents.
- **Bin Policies**: This tab contains some more advanced options for warehouse management.

Variants

Item variants are a powerful feature of Microsoft Dynamics NAV. It enables us to differentiate an item into different categories without having to create a new item.

The variant code is maintained in the item ledger entries and used when applying them.

Example

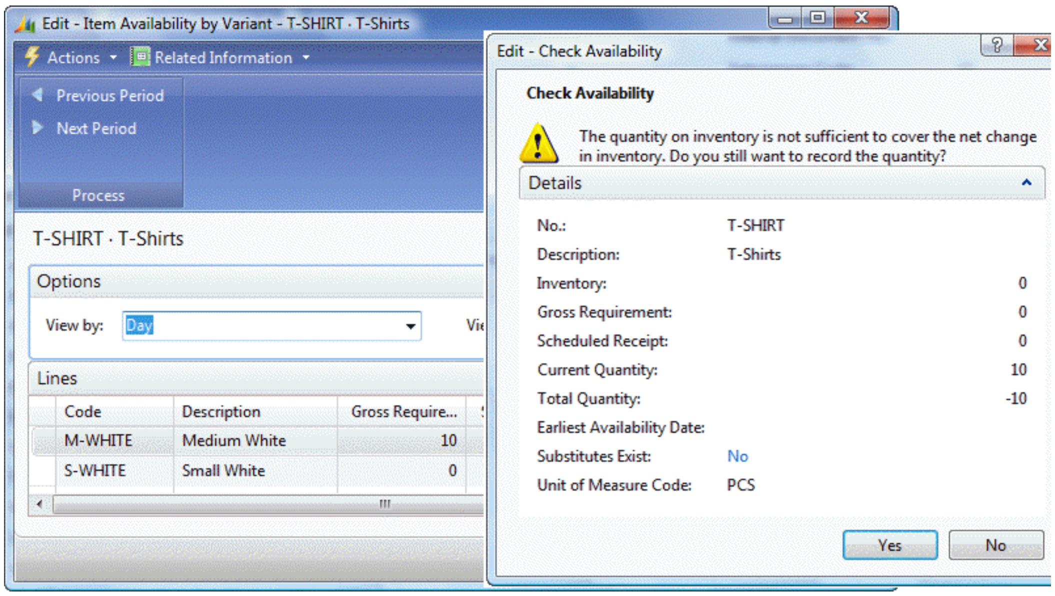
Our company sells t-shirts. We have three sizes: Small, Medium, and Large, and four colours: White, Black, Red, and Blue.

This enables us to create twelve unique variant codes.

S-WHITE	S-BLACK	S-RED	S-BLUE
M-WHITE	M-BLACK	M-RED	M-BLUE
L-WHITE	L-BLACK	L-RED	L-BLUE

When we purchase or produce these t-shirts we need to specify the variant code which is inherited into the item ledger entry.

If we sell or transfer one of these items we can specify the same variant code. Microsoft Dynamics NAV will then use this variant code when searching for inventory.


Edit - Item Availability by Variant - T-SHIRT · T-Shirts
Actions
Related Information
Previous Period
Next Period
Process
Edit - Check Availability
Details
Yes
No

T-SHIRT · T-Shirts

Options

View by: Day

Lines

Code	Description	Gross Require...
M-WHITE	Medium White	10
S-WHITE	Small White	0

Check Availability

The quantity on inventory is not sufficient to cover the net change in inventory. Do you still want to record the quantity?

No.:	T-SHIRT	
Description:	T-Shirts	
Inventory:		0
Gross Requirement:		0
Scheduled Receipt:		0
Current Quantity:		10
Total Quantity:		-10
Earliest Availability Date:		
Substitutes Exist:	No	
Unit of Measure Code:	PCS	

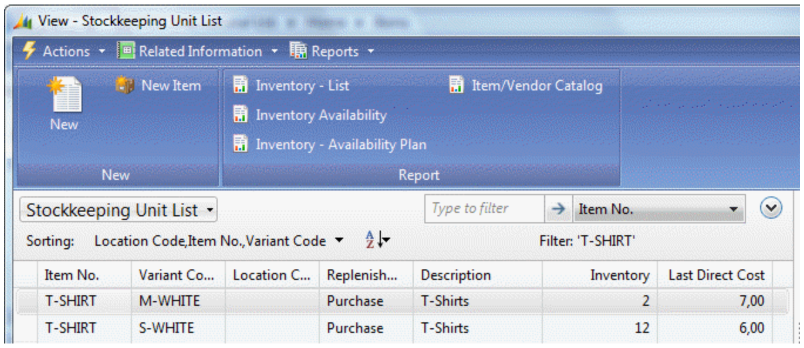
Stock keeping units

Sometimes the same item can have more than one unit cost, Replenishment, system or production method. To support this we can use stock keeping units.

Stock keeping units refer to an existing item, location, and variant. These three fields are the unique primary key.

Example

Our t-shirts need to have different unit costs. In order to do this we need to create an SKU for each variant.



Stockkeeping Unit List

Sorting: Location Code,Item No.,Variant Code

Filter: 'T-SHIRT'

Item No.	Variant Co...	Location C...	Replenish...	Description	Inventory	Last Direct Cost
T-SHIRT	M-WHITE		Purchase	T-Shirts	2	7,00
T-SHIRT	S-WHITE		Purchase	T-Shirts	12	6,00

Now, when we create two purchase order lines for the same item with a different **Variant Code** we can see that the **Last Direct Cost** is different for each variant (as shown in the previous screenshot).

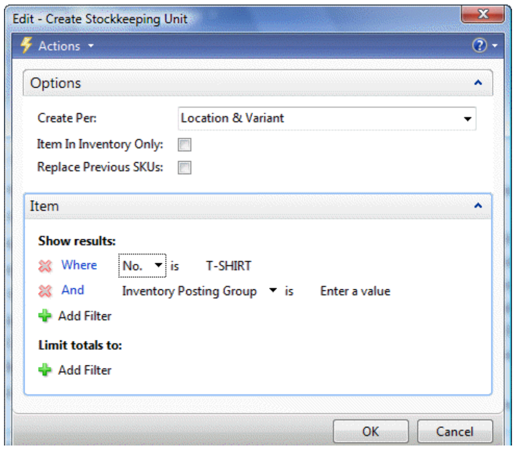
A stock keeping unit is a powerful feature of Microsoft Dynamics that enables you to change settings for an item after it is created using variant codes for each setting. Make sure the code of the variant is self explanatory.

Creating SKU function

When an item has many variants and locations, creating the SKU for each combination can be quite a challenge.

To help in this process we can use the **Create Stockkeeping Unit** report (5706).

The newly created SKU will inherit all the necessary fields from the item. After this we can go in and make necessary changes to the individual SKU records.



Create Per: Location & Variant

Item In Inventory Only: ☐

Replace Previous SKUs: ☐

Show results:

Where No. is T-SHIRT

And Inventory Posting Group is Enter a value

Add Filter

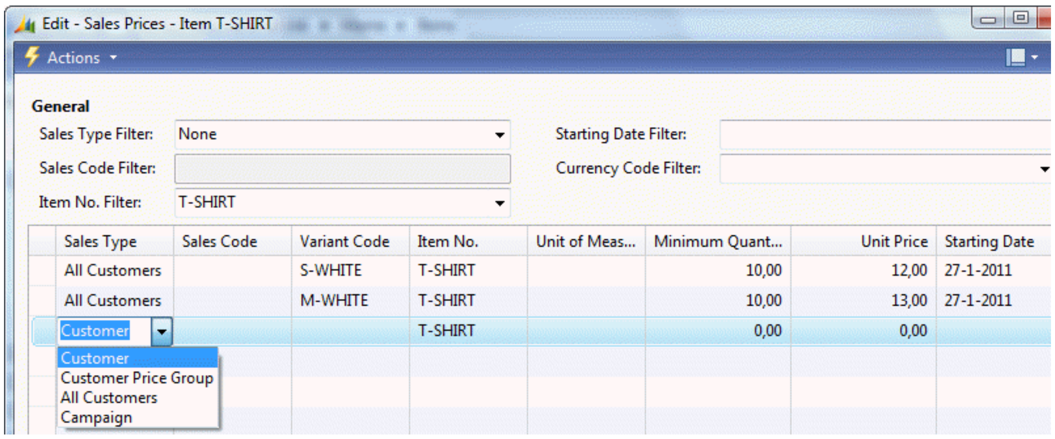
Limit totals to:

Add Filter

Sales pricing

The basic unit price of an item can be set in the Item table. This is a static field which is used when a new sales document is created.

To use more flexible unit prices we can use the sales prices and sales discounts functionality.



Sales Type Filter: None
Sales Code Filter:
Item No. Filter: T-SHIRT
Starting Date Filter:
Currency Code Filter:

Sales Type	Sales Code	Variant Code	Item No.	Unit of Meas...	Minimum Quant...	Unit Price	Starting Date
All Customers		S-WHITE	T-SHIRT		10,00	12,00	27-1-2011
All Customers		M-WHITE	T-SHIRT		10,00	13,00	27-1-2011
Customer			T-SHIRT		0,00	0,00	

More information about pricing can be found in *Chapter 1* and *Chapter 2*.

Item ledger entry application

When inventory is created and used, the system will apply and close positive and negative Item Ledger entries with each other. This enables us to trace inventory.

The Application is saved in the Item Application Entry (339). Let's have a look at the C/AL code that handles the item application.

Item application C/AL routine

This is done in codeunit Item Jnl.-Post Line (22) in the function `ApplyItemLedgEntry`.

The function starts with checking whether reservations are used. Using reservations changes the way the inventory application is used. We'll discuss reservations later in this chapter in the section *Reservations*.

```
ApplyItemLedgEntry
...
CLEAR(OldItemLedgEntry);
```

```
...
REPEAT
  ItemJnlLine.CALCFIELDS("Reserved Qty. (Base)");
  IF ItemJnlLine."Reserved Qty. (Base)" <> 0 THEN
    IF ItemLedgEntry."Applies-to Entry" <> 0 THEN
      ItemLedgEntry.FIELDERROR(
        "Applies-to Entry",Text99000000);
    ...
  END ELSE
    StartApplication := TRUE;
```

If there are no reservations made the system will start the application code. This allows two possibilities: manual application and automatic application.

Manual application is done when the user populates the **Applies-to Entry** field in the Item Journal Line. This is also used when users change the application.

```
IF StartApplication THEN BEGIN
  ItemLedgEntry.CALCFIELDS("Reserved Quantity");
  IF ItemLedgEntry."Applies-to Entry" <> 0 THEN BEGIN
    IF FirstApplication THEN BEGIN
      FirstApplication := FALSE;
      OldItemLedgEntry.GET(ItemLedgEntry."Applies-to Entry");
      OldItemLedgEntry.TESTFIELD("Item No.",ItemLedgEntry."Item No.");
      OldItemLedgEntry.TESTFIELD("Variant Code",ItemLedgEntry."Variant
Code");

      OldItemLedgEntry.TESTFIELD(Positive,NOT ItemLedgEntry.Positive);
      OldItemLedgEntry.TESTFIELD("Location
Code",ItemLedgEntry."Location Code");
```

In this case the system checks if the item ledger entry we have specified matches the requirements.

When the application is done the system will automatically search for the best item ledger entry based on the same requirements.

```
END ELSE BEGIN
  IF FirstApplication THEN BEGIN
    FirstApplication := FALSE;
    ItemLedgEntry2.SETCURRENTKEY("Item No.",Open,"Variant Code",
      Positive,"Location Code","Posting Date");
    ItemLedgEntry2.SETRANGE("Item No.",ItemLedgEntry."Item No.");
    ItemLedgEntry2.SETRANGE(Open,TRUE);
    ItemLedgEntry2.SETRANGE("Variant Code",ItemLedgEntry.
      "Variant Code");
    ItemLedgEntry2.SETRANGE(Positive,NOT ItemLedgEntry.Positive);
    ItemLedgEntry2.SETRANGE("Location Code",
      ItemLedgEntry."Location Code");
    IF ItemLedgEntry."Job Purchase" = TRUE THEN BEGIN
      ItemLedgEntry2.SETRANGE("Job No.",ItemLedgEntry."Job No.");
      ItemLedgEntry2.SETRANGE("Job Task No.",
        ItemLedgEntry."Job Task No.");
    END;
    IF ItemTrackingCode."SN Specific Tracking" THEN
      ItemLedgEntry2.SETRANGE("Serial No.",
        ItemLedgEntry."Serial No.");
    IF ItemTrackingCode."Lot Specific Tracking" THEN
      ItemLedgEntry2.SETRANGE("Lot No.",ItemLedgEntry."Lot No.");
    IF Location.GET(ItemLedgEntry."Location Code") THEN
      IF Location."Use As In-Transit" THEN
        ItemLedgEntry2.SETRANGE("Transfer Order No.",
          ItemLedgEntry."Transfer Order No.");
    IF Item."Costing Method" = Item."Costing Method"::LIFO THEN
      EntryFindMethod := '+'
    ELSE
      EntryFindMethod := '-';
    IF NOT ItemLedgEntry2.FIND(EntryFindMethod) THEN
      EXIT;
```

The actual application entry is created in the function `InsertApplEntry`.

Requirements

In order to apply an item ledger entry to another item ledger entry, certain requirements should be taken into account. We can read these requirements from the C/AL code:

1. The Item No. should be the same.
2. The old item ledger entry should be 'Open'. When an item ledger entry is fully applied the boolean field 'Open' is set to 'FALSE'.
3. The variant code and location code should be the same.
4. The boolean field 'Positive' should have a reverse sign. This results in the limitation of not being able to apply one negative entry to another negative entry.

Other requirements are conditional based on system setup. For example, if the item uses a lot no. or a serial no. this should also match.

When the system has defined the filter it tries to find the first record. The search method depends on the costing method. If the cost method is LIFO the system will try to find the last record in the filter. For all other costing methods it will find the first.

We can also see that when using lot numbers, the application and the costing is done within the lot number.

Value entries

In Microsoft Dynamics NAV the physical information for inventory is stored separately from the financial information. This information is stored in a one-to-many relation, meaning one item ledger entry can have multiple value entries.

This enables us to specify the value information in detail in a time dimension and cost type dimension.

Direct cost

Each item ledger entry starts with at least one value entry of the type Direct Cost. This defines the initial value of the inventory.

During the inventory lifetime the item ledger entry can get four other types of value entries:

- **Revaluation**: This entry type is used when the item revaluation batch is started and the value of the item is different compared to the direct cost.
- **Rounding**: Sometimes the inventory adjustment leads into rounding issues. The rounding is stored as a special entry type for traceability.
- **Indirect cost**: When the indirect cost % is used on the item card the system will create additional value entries for the indirect cost amount.
- **Variance**: When the item uses standard cost, the difference between the invoiced amount and the standard cost is saved as an entry type variance.

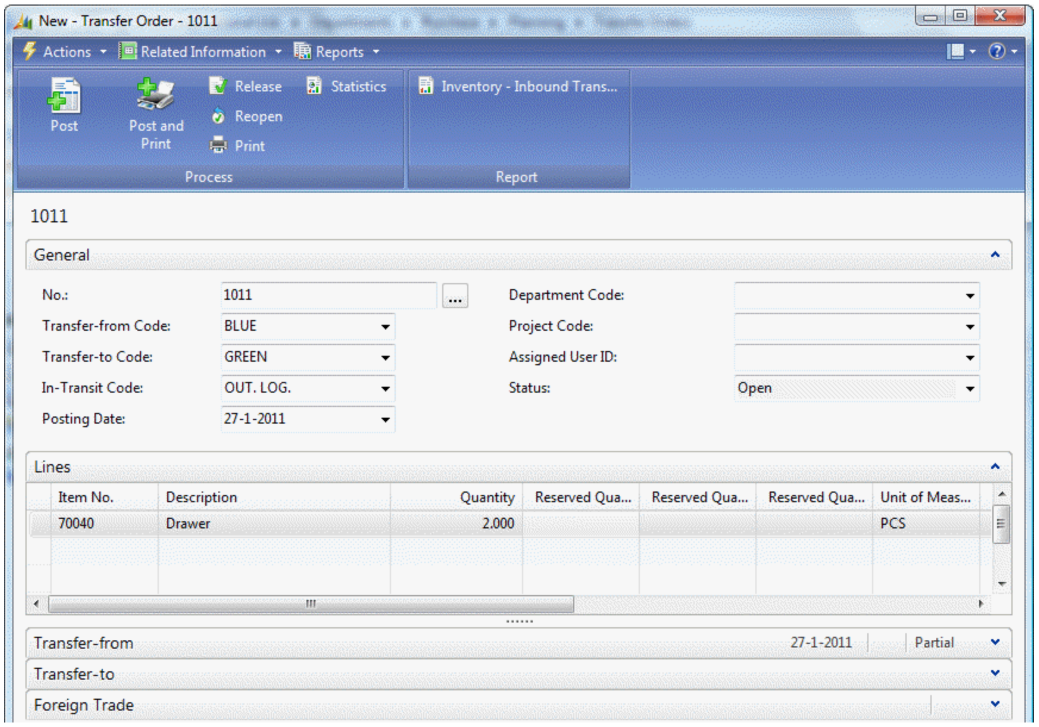
Value entries and general ledger entries

The Value Entries and General Ledger Entries are linked through the G/L - Item Ledger Relation table (5823).

Each general ledger entry is linked to one or more value entries. This enables traceability and helps auditors to analyze the system.

Transfer orders

In order to move inventory from one location to another, it is possible to do a negative and a positive adjustment in the Item Journal Line, but we can also use a **Transfer Order**.



1011

General

No.:	1011	Department Code:	
Transfer-from Code:	BLUE	Project Code:	
Transfer-to Code:	GREEN	Assigned User ID:	
In-Transit Code:	OUT. LOG.	Status:	Open
Posting Date:	27-1-2011		

Lines

Item No.	Description	Quantity	Reserved Qua...	Reserved Qua...	Reserved Qua...	Unit of Meas...
70040	Drawer	2.000				PCS

Transfer-from 27-1-2011 Partial

Transfer-to

Foreign Trade

The transfer order creates the item ledger entries for each location and maintains the link for the value entries.

This means that if we move 100 items from the location blue to green without having received the purchase invoice, yet the system will create value entries for the moved inventory when the invoice is posted.

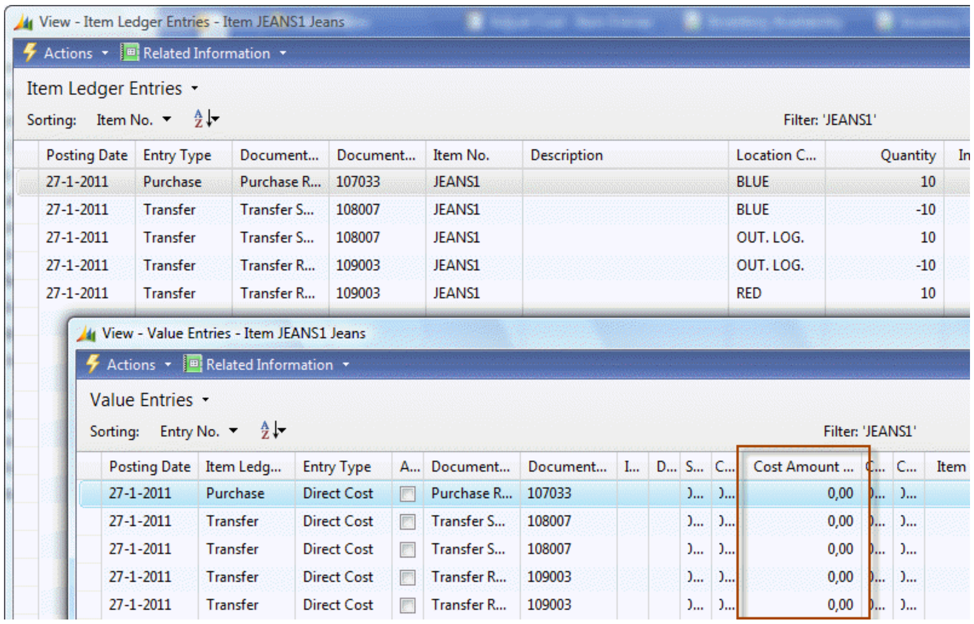
Let's try this for a new item.

Example

The item we will use is Jeans. The first step is to create the item.

1. We define the No., Description, Base UOM, and the Posting Groups only.
2. Now we create a new purchase order with quantity 10 in location **BLUE** and unit cost, **10**.
3. We receive the purchase order.
4. Using a new transfer order we move the inventory from BLUE to RED.

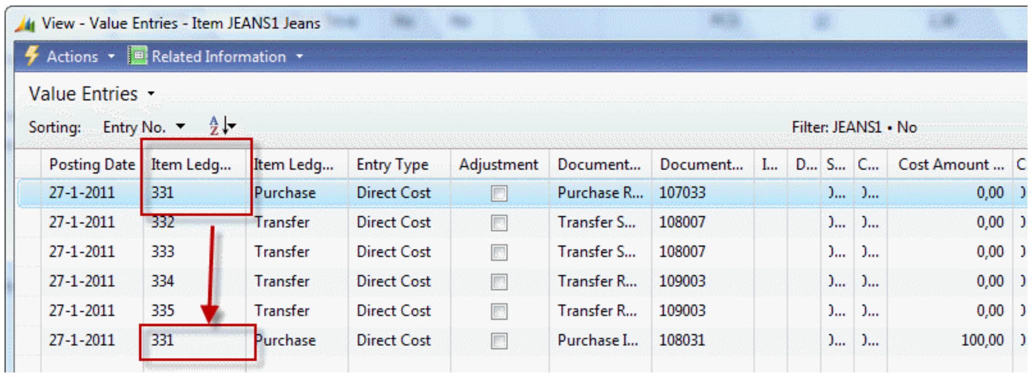
   This will result in five **Item Ledger Entries** with five **Value Entries** but the total cost is zero as we have not yet received the purchase invoice:



Posting Date	Entry Type	Document...	Document...	Item No.	Description	Location C...	Quantity	In
27-1-2011	Purchase	Purchase R...	107033	JEANS1		BLUE	10	
27-1-2011	Transfer	Transfer S...	108007	JEANS1		BLUE	-10	
27-1-2011	Transfer	Transfer S...	108007	JEANS1		OUT. LOG.	10	
27-1-2011	Transfer	Transfer R...	109003	JEANS1		OUT. LOG.	-10	
27-1-2011	Transfer	Transfer R...	109003	JEANS1		RED	10	

Posting Date	Item Ledg...	Entry Type	A...	Document...	Document...	I...	D...	S...	C...	Cost Amount ...	C...	C...	Item
27-1-2011	Purchase	Direct Cost	☐	Purchase R...	107033			)...	)...	0,00	)...	)...	
27-1-2011	Transfer	Direct Cost	☐	Transfer S...	108007			)...	)...	0,00	)...	)...	
27-1-2011	Transfer	Direct Cost	☐	Transfer S...	108007			)...	)...	0,00	)...	)...	
27-1-2011	Transfer	Direct Cost	☐	Transfer R...	109003			)...	)...	0,00	)...	)...	
27-1-2011	Transfer	Direct Cost	☐	Transfer R...	109003			)...	)...	0,00	)...	)...	

5. Now we create a new purchase invoice and get the receipt lines. We use an unit cost of 10.


View - Value Entries - Item JEANS1 Jeans

Actions · Related Information

Value Entries

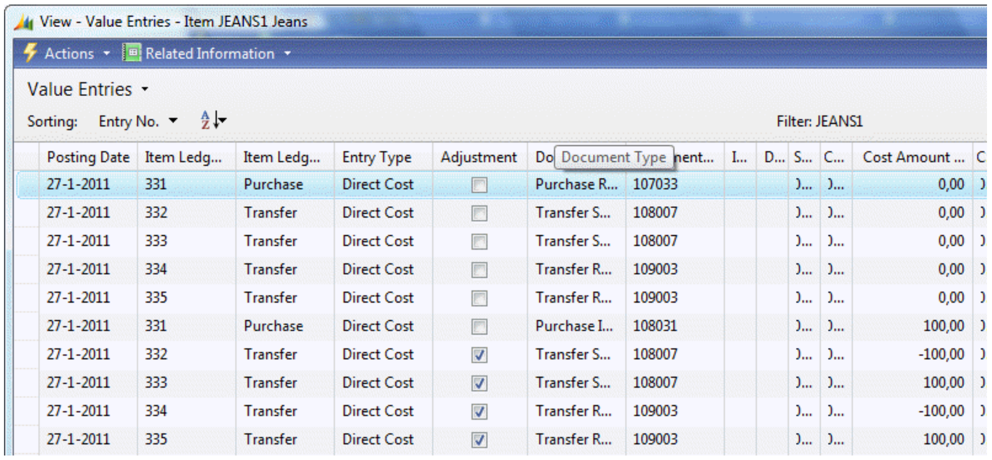
Sorting: Entry No. Filter: JEANS1 • No

Posting Date	Item Ledg...	Item Ledg...	Entry Type	Adjustment	Document...	Document...	I...	D...	S...	C...	Cost Amount ...	C
27-1-2011	331	Purchase	Direct Cost	☐	Purchase R...	107033			)...	)...	0,00	)
27-1-2011	332	Transfer	Direct Cost	☐	Transfer S...	108007			)...	)...	0,00	)
27-1-2011	333	Transfer	Direct Cost	☐	Transfer S...	108007			)...	)...	0,00	)
27-1-2011	334	Transfer	Direct Cost	☐	Transfer R...	109003			)...	)...	0,00	)
27-1-2011	335	Transfer	Direct Cost	☐	Transfer R...	109003			)...	)...	0,00	)
27-1-2011	331	Purchase	Direct Cost	☐	Purchase I...	108031			)...	)...	100,00	)

This results in a value entry for the original item ledger entry.

6. To create the value entries for the transfers we need to run the Adjust Cost - Item Entries report (795).

   This results in all item ledger entries having the same value entries.


View - Value Entries - Item JEANS1 Jeans

Actions · Related Information

Value Entries

Sorting: Entry No. Filter: JEANS1

Posting Date	Item Ledg...	Item Ledg...	Entry Type	Adjustment	Do Document Type	ment...	I...	D...	S...	C...	Cost Amount ...	C
27-1-2011	331	Purchase	Direct Cost	☐	Purchase R...	107033			)...	)...	0,00	)
27-1-2011	332	Transfer	Direct Cost	☐	Transfer S...	108007			)...	)...	0,00	)
27-1-2011	333	Transfer	Direct Cost	☐	Transfer S...	108007			)...	)...	0,00	)
27-1-2011	334	Transfer	Direct Cost	☐	Transfer R...	109003			)...	)...	0,00	)
27-1-2011	335	Transfer	Direct Cost	☐	Transfer R...	109003			)...	)...	0,00	)
27-1-2011	331	Purchase	Direct Cost	☐	Purchase I...	108031			)...	)...	100,00	)
27-1-2011	332	Transfer	Direct Cost	☑	Transfer S...	108007			)...	)...	-100,00	)
27-1-2011	333	Transfer	Direct Cost	☑	Transfer S...	108007			)...	)...	100,00	)
27-1-2011	334	Transfer	Direct Cost	☑	Transfer R...	109003			)...	)...	-100,00	)
27-1-2011	335	Transfer	Direct Cost	☑	Transfer R...	109003			)...	)...	100,00	)

Requisition journals

For trading companies it is important to have enough inventories, not too much, not too few.

In order to do this we can use the Requisition Journals together with the Reordering Policy on the item.

Reordering policy

Microsoft Dynamics NAV uses four different reordering policies:

- **Fixed Reorder Qty.**: Each time we run the requisition journal the system will purchase the same, fixed quantity of items. This quantity is specified in the Reorder Quantity field.
- **Maximum Qty**.: The system will purchase as many items as to meet the value of the field, Maximum Inventory.
- **Order**: For each sales order a purchase order will be created. This automatically enables the reservation process for this item.
- **Lot-for-Lot**: This option will calculate the required inventory necessary to deliver the outstanding sales orders.

The quantity is calculated in codeunit Inventory Profile Offsetting (99000854) in the function `CalcReorderQty`.

Extending reordering policy

The ordering policy algorithms in Microsoft Dynamics NAV are very static and some trading companies need more flexibility.

One example is seasonal and weather influences. Toy stores need extra inventory during Christmas and garden tool stores have their peak in spring.

During these peaks the delivery times and availability are also different compared to the other times of the year.

Virtual inventory

An upcoming trend in trading companies is virtual inventory. This is inventory that we do not control but is available to sell to our customers.

The computer industry uses this frequently. Everyone can start a website for computer equipment and use the inventory of the large wholesale companies.

[In order for this to work the information should always be real time and reliable. In Microsoft Dynamics NAV we could solve this using web services.]

Warehouse management

With Inventory Management we can use the locations to see where the inventory is. For some trading companies this is good enough but some would like to be more specific in where the items are in the warehouse.

For this we can use the Warehouse Management functionality in Microsoft Dynamics NAV. WMS enables us to specify Zones and Bins within each location.

Another feature in Warehouse Management is the possibility of combining sales shipments and purchase receipts in warehouse documents. Using these documents warehouse employees can pick or put away more for than one order at the same time resulting in a more efficient way of handling logistics.

Warehouse strategy levels

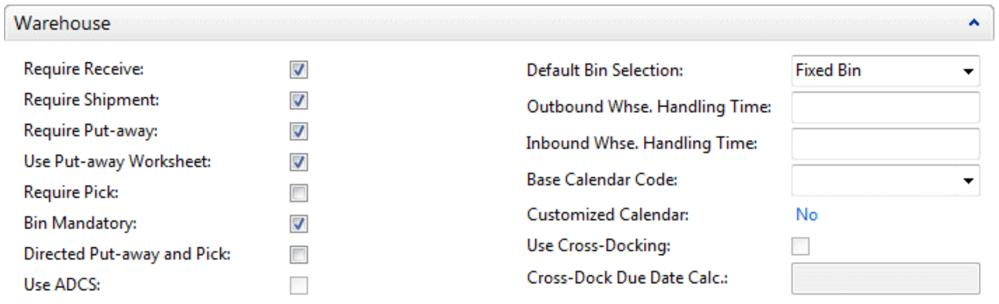
Warehouse Management can be used and implemented in various fashions, from very simple to highly advanced ones. To demonstrate the application design of WMS in Microsoft Dynamics NAV we will discuss five possible levels of implementation. For each level we will show the table and posting models.

1. Bin Code: Using the Bin Code field in the sales and purchase document enables the system to start creating warehouse entries.
2. Warehouse Receipt and Shipment: This allows us to combine sales shipments and purchase receipts into one warehouse document. We cannot use the Pick & Put-Away activities.
3. Warehouse Put-Away and Pick: For each purchase receipt or sales shipment we can create a Put-Away or Pick journal.
4. Warehouse Receipt and Shipment + Use Put-Away Worksheet: This allows us to implement a real two step Warehouse process. Receiving the items on a staging location and create Put-Away documents to move the items to their storage location in the warehouse.
5. Directed Put-Away and Pick: This is the full option WMS functionality in Microsoft Dynamics NAV. We use Receipts, Shipments, Put-Aways, and Picks. Microsoft Dynamics NAV will suggest the Bin Codes. We can use Zones, Cross Docking and so on.

Location setup

The setup options in the Location table (14) enable or disable the WMS options in Microsoft Dynamics NAV.

This is done on the Warehouse tab (as shown in the following screenshot). Each level requires a special combination of settings:

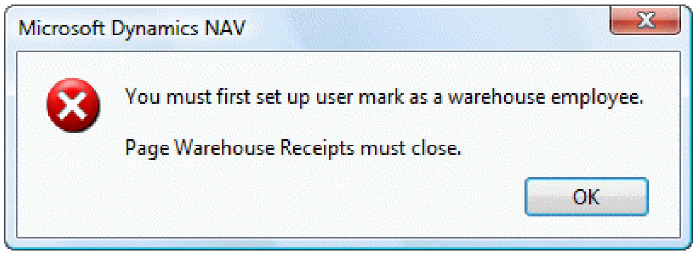


Require Receive:	✓	Default Bin Selection:	Fixed Bin
Require Shipment:	✓	Outbound Whse. Handling Time:	
Require Put-away:	✓	Inbound Whse. Handling Time:	
Use Put-away Worksheet:	✓	Base Calendar Code:	
Require Pick:	☐	Customized Calendar:	No
Bin Mandatory:	✓	Use Cross-Docking:	☐
Directed Put-away and Pick:	☐	Cross-Dock Due Date Calc.:	
Use ADCS:	☐		

- **Level 1**: Enable Bin Mandatory
- **Level 2**: Enable Require Receive, Require Shipment, and Bin Mandatory
- **Level 3**: Enable Require Put-away, Require Pick, and Bin Mandatory
- **Level 4**: Enable Require Receive, Shipment, Require Put-away, Require Pick, Bin Mandatory , and Use Put-Away Worksheet
- **Level 5**: Enable Require Receive, Require Shipment, Require Put-away, Require Pick, Bin Mandatory, and Directed Put-away and Pick

Warehouse employees

Before we can start the current user should be set up as a warehouse employee.



You must first set up user mark as a warehouse employee.

Page Warehouse Receipts must close.

This can be done by creating a new record in the Warehouse Employee table (7301).

Each user can be a warehouse employee in each location and can only do warehouse actions in the locations that they are assigned to.

Bin code | level 1

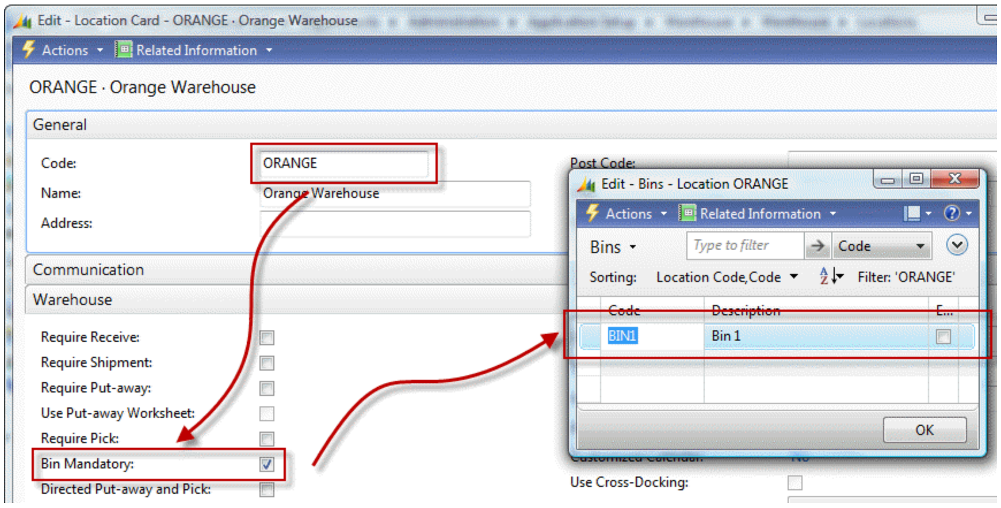
The starting level of implementing WMS is using the Bin table. This is done by enabling the Bin Mandatory field on the location.

The Bin Code field is available in all the necessary tables such as Purchase Line, Sales Line, and Item Journal Line.

When the Bin Code is used, codeunit Item Jnl.-Post Line (22) will create a Warehouse Journal Line and start the Whse. Jnl.-Register Line Codeunit (7301). This will result in the creation of Warehouse Entries (7312) and a Bin Content (7302).

Example

We will create a new location **ORANGE** with a bin **Bin1**. The Location uses the **Bin Mandatory** option, as shown in the following screenshot:


Edit - Location Card - ORANGE · Orange Warehouse
Actions
Related Information
ORANGE · Orange Warehouse
General
Communication
Warehouse
Post Code:
Edit - Bins - Location ORANGE
Bins
Type to filter
Code
OK

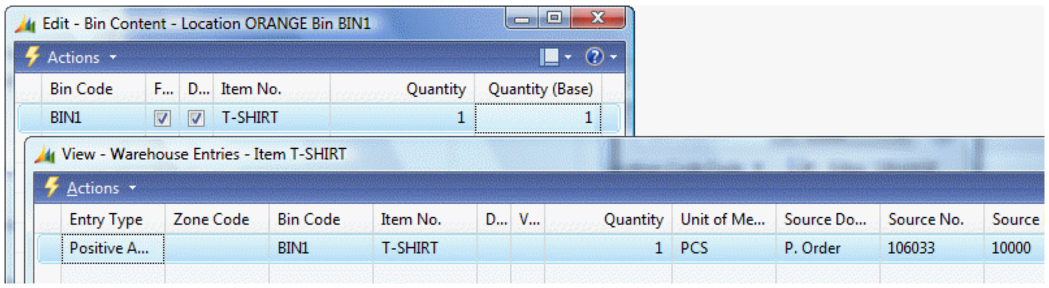
Code:	ORANGE
Name:	Orange Warehouse
Address:	
Require Receive:	☐
Require Shipment:	☐
Require Put-away:	☐
Use Put-away Worksheet:	☐
Require Pick:	☐
Bin Mandatory:	✓
Directed Put-away and Pick:	☐
Use Cross-Docking:	☐

Sorting: Location Code,Code Filter: 'ORANGE'

Code	Description	E...
BIN1	Bin 1	☐

In a new purchase order we can now select this Bin Code and post a receipt.

The system now creates a new record in the Bin Content table which enables us to see the created Warehouse Entries as shown in the following screenshot:



Edit - Bin Content - Location ORANGE Bin BIN1

Bin Code	F...	D...	Item No.	Quantity	Quantity (Base)
BIN1	☑	☑	T-SHIRT	1	1

View - Warehouse Entries - Item T-SHIRT

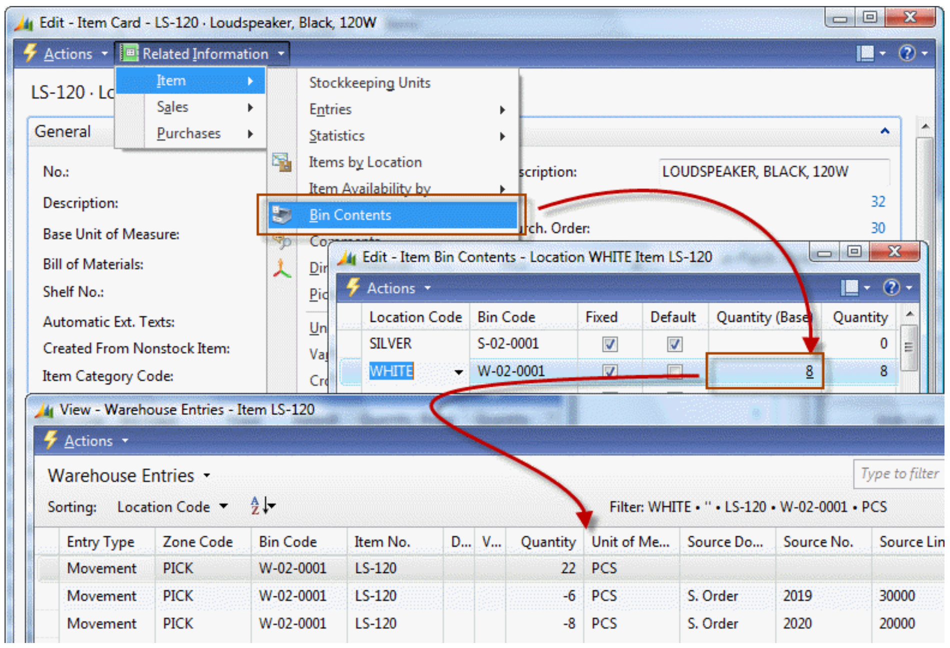
Entry Type	Zone Code	Bin Code	Item No.	D...	V...	Quantity	Unit of Me...	Source Do...	Source No.	Source
Positive A...		BIN1	T-SHIRT			1	PCS	P. Order	106033	10000

Bin content

Whenever a Bin is used for the first time, Microsoft Dynamics NAV will create a Bin Content record. A bin content record is neither master data nor a ledger entry or document. It is a special kind of table in the philosophy of Microsoft Dynamics NAV.

The C/AL code for the bin content handling can be found in codeunit Whse. Jnl.-Register Line (7301).

To see which bins are used for an item at any moment we can open the **Bin Contents** from the **Item Card**, as shown in the following screenshot:



No.:	
Description:	LOUDSPEAKER, BLACK, 120W
Base Unit of Measure:	
Bill of Materials:	
Shelf No.:	
Automatic Ext. Texts:	
Created From Nonstock Item:	
Item Category Code:	

Edit - Item Bin Contents - Location WHITE Item LS-120

Location Code	Bin Code	Fixed	Default	Quantity (Base)	Quantity
SILVER	S-02-0001	☑	☑		0
WHITE	W-02-0001	☑	☐	8	8

View - Warehouse Entries - Item LS-120

Warehouse Entries

Sorting: Location Code

Filter: WHITE • '' • LS-120 • W-02-0001 • PCS

Entry Type	Zone Code	Bin Code	Item No.	D...	V...	Quantity	Unit of Me...	Source Do...	Source No.	Source Lin
Movement	PICK	W-02-0001	LS-120			22	PCS			
Movement	PICK	W-02-0001	LS-120			-6	PCS	S. Order	2019	30000
Movement	PICK	W-02-0001	LS-120			-8	PCS	S. Order	2020	20000

The warehouse entries can be displayed by clicking on the **Quantity (Base)** field.

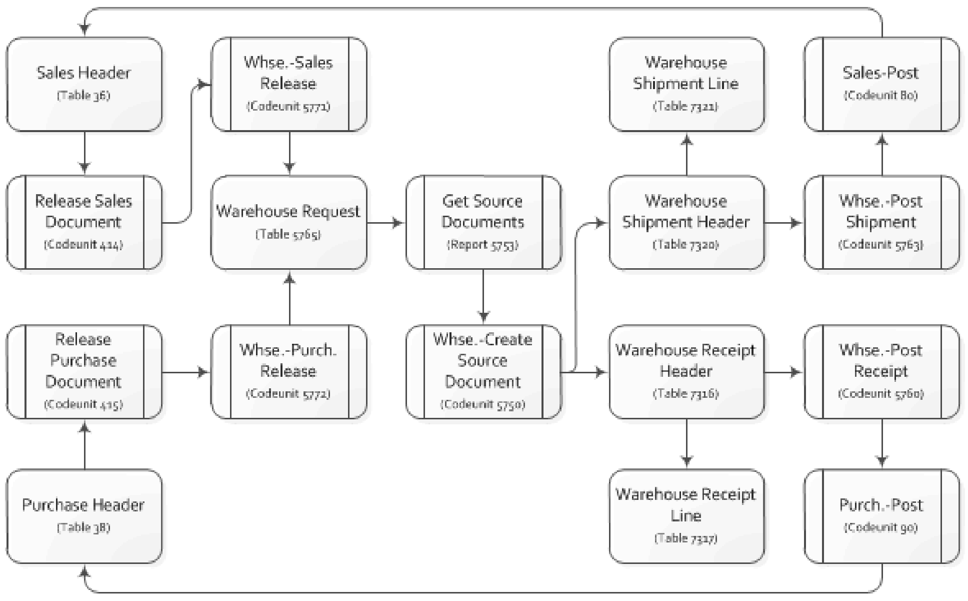
The Warehouse Entries are only used internally by the program and are difficult to find in the end user interface.

Receipt and shipment | level 2

When we enable Require Receive and Require Shipment in the Location we can start using the Warehouse Receipt and Shipment Documents.

These documents allow us to receive or ship multiple purchase or sales orders in one document.

Let's have a look at how this is done in the application:


Sales Header (Table 36)
Whse.-Sales Release (Codeunit 5771)
Warehouse Shipment Line (Table 7321)
Sales-Post (Codeunit 80)
Release Sales Document (Codeunit 414)
Warehouse Request (Table 5765)
Get Source Documents (Report 5753)
Warehouse Shipment Header (Table 7320)
Whse.-Post Shipment (Codeunit 5763)
Release Purchase Document (Codeunit 415)
Whse.-Purch. Release (Codeunit 5772)
Whse.-Create Source Document (Codeunit 5750)
Warehouse Receipt Header (Table 7316)
Whse.-Post Receipt (Codeunit 5760)
Purchase Header (Table 38)
Warehouse Receipt Line (Table 7317)
Purch.-Post (Codeunit 90)

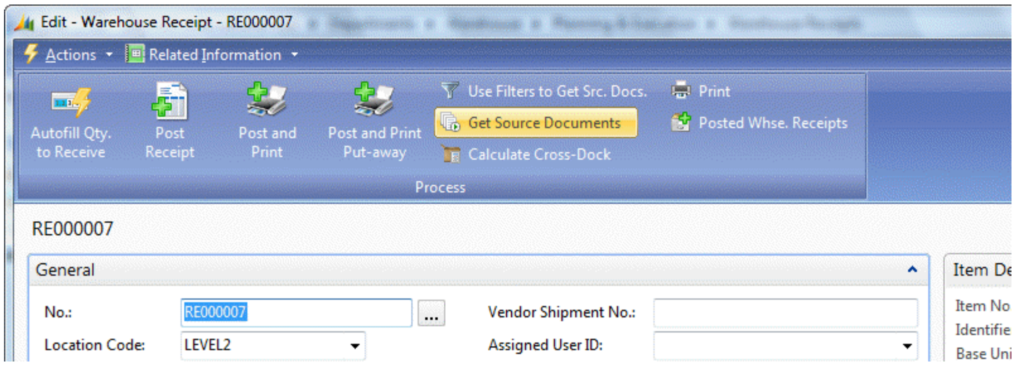
Warehouse request

All warehouse documents start with a record Warehouse Request table (5765). These records are created when a sales or purchase document is released.

The warehouse receipt or shipment can be created in three ways.

1. Using the option **Create Whse. Receipt** or **Create Whse. Shipment** from the Purchase and Sales Order.
2. Using report **Get Source Documents** (5723).
3. Using the option **Get Source Documents** on the **Warehouse Receipt** or **Shipment Card**.

The first two options will create a new warehouse document for each sales or purchase document. The latter allows us to combine orders in one warehouse document.



RE000007

No.: RE000007
Location Code: LEVEL2
Vendor Shipment No.:
Assigned User ID:

Limitations

Using only the warehouse receipt and shipment document is basically just adding one layer on top of the sales and purchase document. The posting routines, Whse. Post Shipment (5763) and Whse. Post Receipt (5760), does not actually post something to the warehouse, they just write back the bin code to the Sales Line and Purchase Line table. Technically this uses the same C/AL code as level 1.

We can see how this is done by looking at the function `InitSourceDocumentLines` of, for example, codeunit Whse. Post Receipt (5760).

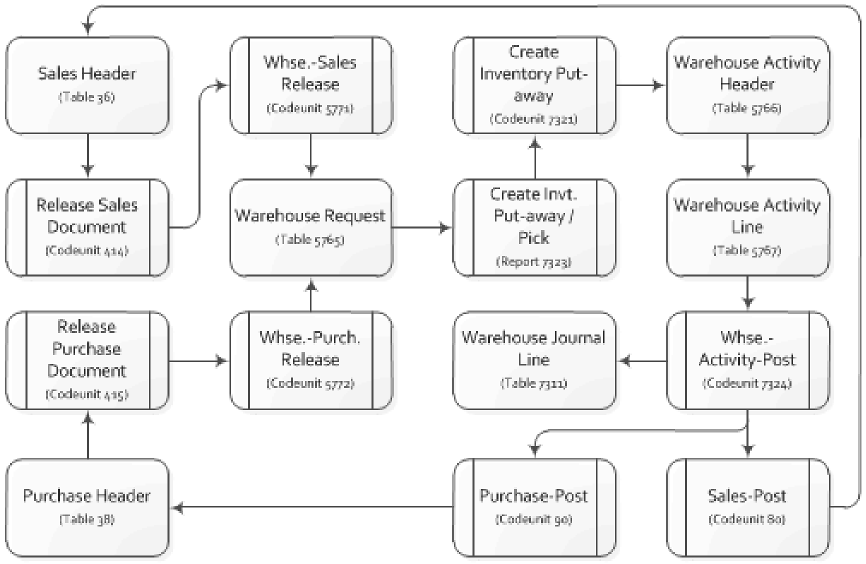
```
InitSourceDocumentLines
WhseRcptLine2.COPY(WhseRcptLine);
WITH WhseRcptLine2 DO BEGIN
  CASE "Source Type" OF
    DATABASE::"Purchase Line":
      BEGIN
        PurchLine.SETRANGE("Document Type","Source Subtype");
        PurchLine.SETRANGE("Document No.","Source No.");
        IF PurchLine.FIND('-') THEN
          REPEAT
        ...
        IF PurchLine."Bin Code" <> "Bin Code" THEN BEGIN
          PurchLine."Bin Code" := "Bin Code";
          ModifyLine := TRUE;
        END;
        ...
      IF ModifyLine THEN
        PurchLine.MODIFY;
```

When the source tables are updated the system creates a normal purchase receipt or sales shipment using codeunits Sales-Post (80) and Purch. Post (90).

Put-Away and Pick | level 3

Instead of creating a warehouse receipt or shipment we can also directly create a Put-Away or Pick from the sales or purchase order.

To enable this we need to activate the options Require Put-away and Require Pick on the Location Card.


Sales Header (Table 36)
Whse.-Sales Release (Codeunit 5771)
Create Inventory Put-away (Codeunit 7321)
Warehouse Activity Header (Table 5766)
Release Sales Document (Codeunit 414)
Warehouse Request (Table 5765)
Create Invt. Put-away / Pick (Report 7323)
Warehouse Activity Line (Table 5767)
Release Purchase Document (Codeunit 415)
Whse.-Purch. Release (Codeunit 5772)
Warehouse Journal Line (Table 7311)
Whse.-Activity-Post (Codeunit 7324)
Purchase Header (Table 38)
Purchase-Post (Codeunit 90)
Sales-Post (Codeunit 80)

Warehouse request

The Warehouse Request record is exactly the same as in level 2, but instead of creating a warehouse receipt or shipment the system directly creates a warehouse activity header and line.

Warehouse activities

The Warehouse Activity Header and Line table are the internal Microsoft Dynamics NAV Warehouse documents. There are five types of warehouse activity documents:

- **Put-away**: This document is used to move items from the receipt bin to a put away bin. The document is generated from a warehouse receipt.
- **Pick**: This document used to move items from a storage bin to a shipment bin. The document is generated from a warehouse shipment.
- **Movement**: This is an internal document that is used to move items internally in the warehouse.
- **Invt. Put-away**: This document is used to receive items and put them directly into the warehouse on their permanent bin. The document is created from a warehouse request.
- **Invt. Pick**: This document is used to ship items directly from the warehouse in one step. The document is created from a warehouse request.

When only using the Require Put-away and Require Pick option on a location, document types Invt. Put-Away and Invt. Pick are used. This will make sure that the purchase order and sales order will also be processed by starting codeunit Sales-Post (80) and Purch. Post (90).

Level 2 and level 3 comparison

Both level 2 and level 3 set up options are one-step warehouse implementations. When receiving an item we must provide the storage bin where the item will be stored until it is sold. There is no additional step.

Using the warehouse receipt and shipment documents allows us to combine sales and purchase documents on one warehouse document. This cannot be done using direct Put-Away and Pick.

Using Direct Put-Away and Pick we can split one sales line or purchase line into multiple Bins. This cannot be done using warehouse receipt and shipment documents.

The reason for this is the way the warehouse entries are created. Level 2 uses the bin field in the item journal line to create the warehouse entries.

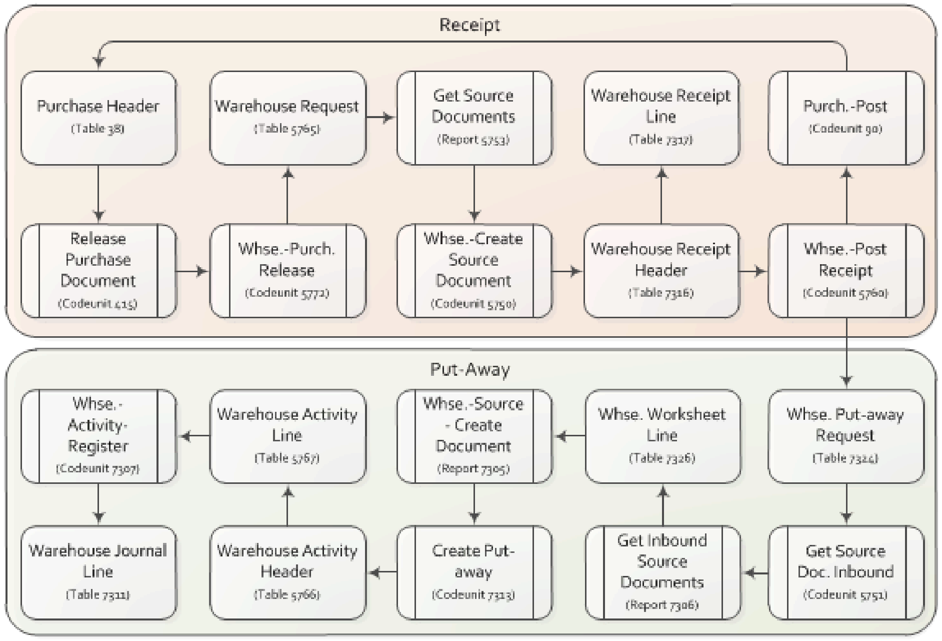
Using level 3, the warehouse entries are created using codeunit Whse.-Activity-Post (7324). The bin code is not written back into the sales line or purchase line. This means we also cannot use the bin code field in the purchase receipt and sales shipment documents.

Receipt + use put-away worksheet | level 4

Most warehouses use a two step receipt and shipment process. The first step is receiving the items on a receipt location which is often close to the unloading docks. In the next step the items are stored in their warehouse location until they are required for the production or sales process.

To enable this two step process we can combine level 2 and 3 by using the options Require Receive, Shipment, Put-Away and Pick + Bin Mandatory + Use Put-Away Worksheet in the location card.

This allows us to first perform the warehouse receipt and shipment as discussed in level 2. When we process this document it will not only post the sales order and purchase order but it will also generate a record in the Whse. Put-Away Request table (7324).


Receipt
Purchase Header (Table 38)
Warehouse Request (Table 5765)
Get Source Documents (Report 5753)
Warehouse Receipt Line (Table 7317)
Purch.-Post (Codeunit 90)
Release Purchase Document (Codeunit 415)
Whse.-Purch. Release (Codeunit 5772)
Whse.-Create Source Document (Codeunit 5750)
Warehouse Receipt Header (Table 7316)
Whse.-Post Receipt (Codeunit 5760)
Put-Away
Whse.-Activity-Register (Codeunit 7307)
Warehouse Activity Line (Table 5767)
Whse.-Source - Create Document (Report 7305)
Whse. Worksheet Line (Table 7326)
Whse. Put-away Request (Table 7324)
Warehouse Journal Line (Table 7311)
Warehouse Activity Header (Table 5766)
Create Put-away (Codeunit 7313)
Get Inbound Source Documents (Report 7306)
Get Source Doc. Inbound (Codeunit 5751)

Whse.- activity register versus whse.-activity-post

When the whse.-put-away request is processed using the warehouse worksheet it will result in a warehouse activity header and line. In this context the system will use the Put-Away and Pick document type we discussed in the previous section about level 3.

Technically the documents for level 3 and level 4 are equal, but there are two differences.

1 In a two step warehouse set up the items are already in a warehouse entry. This means we have to move them. This results in two new warehouse entries but also two lines in the warehouse document.
2 The two step warehouse documents are not posted but registered. This means that the system will only create warehouse entries and no longer update the sales and purchase documents.

Directed put-away and pick | level 5

Combining warehouse receipts and shipments with Put-Aways and Picks completes the table and posting diagram of WMS in Microsoft Dynamics NAV. But there are additional options to enrich the functionality.

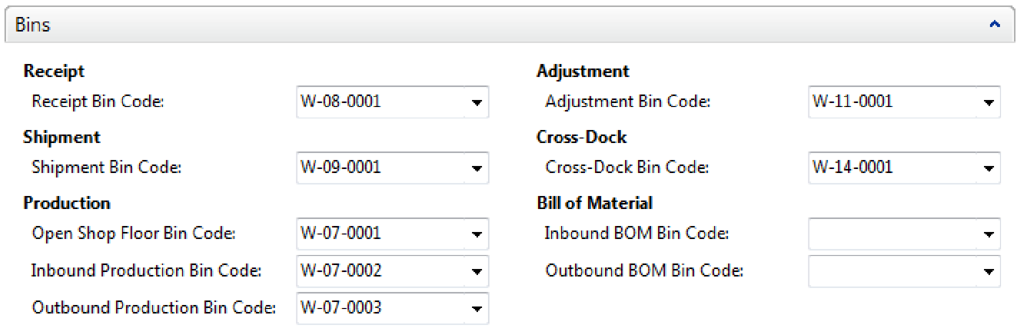
One of these options is directed Put-Away and Pick. When this option is activated the system can and will help us in finding the correct bins for each warehouse activity.

Zones and default bins

This starts with defining Zones and default Bins. A Zone is a group of Bins. Usually they are located near each other physically but more important, they share some properties.

For each zone we need to specify if it is allowed to receive, ship, Put-Away, and Pick. This is done in the Bin Type list.

While defining the bins, it is recommended to use a logical name such as R-01-001 for receipt row one, shelf one.



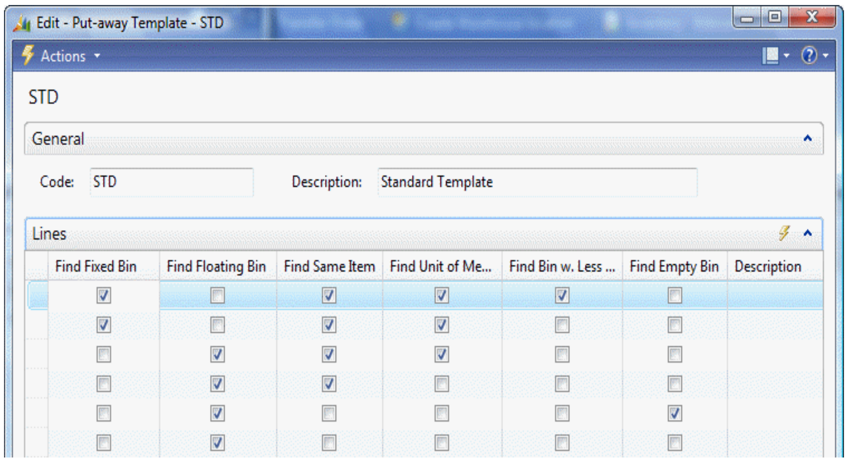
Bins

Receipt		**Adjustment**	
Receipt Bin Code:	W-08-0001	Adjustment Bin Code:	W-11-0001
Shipment		**Cross-Dock**	
Shipment Bin Code:	W-09-0001	Cross-Dock Bin Code:	W-14-0001
Production		**Bill of Material**	
Open Shop Floor Bin Code:	W-07-0001	Inbound BOM Bin Code:	
Inbound Production Bin Code:	W-07-0002	Outbound BOM Bin Code:	
Outbound Production Bin Code:	W-07-0003		

The default bins are set up in the Location Card on the Bin Tab. These bins can always be changed on each document.

Bin calculation

The bin calculation is done for the put-away documents using templates. This template defines the rules for finding the correct bin to store the items.



STD

General

Code: STD Description: Standard Template

Lines

Find Fixed Bin	Find Floating Bin	Find Same Item	Find Unit of Me...	Find Bin w. Less ...	Find Empty Bin	Description
✓	☐	✓	✓	✓	☐	
✓	☐	✓	✓	☐	☐	
☐	✓	✓	✓	☐	☐	
☐	✓	✓	☐	☐	☐	
☐	✓	☐	☐	☐	✓	
☐	✓	☐	☐	☐	☐	

The find options are stored in the Put-away Template Line table (7308).

- **Find Fixed Bin**: The system will try to find a bin which is fixed. A fixed bin is usually reserved for a specific Item.
- **Find Floating Bin**: This will try to find the first available bin.

- **Find Same Item**: This will filter on an available bin that already contains this item.
- **Find Unit of Measure Match**: This option can be used if parts of the warehouse are designed to handle a specific kind of carrier such as euro or us pallet.
- **Find Bin w. Less than Min. Qty**: Use this option to find bins what are not fully used. If this option is not used with the Find Same Item it might result in two items in the same bin.
- **Find Empty Bin**: This option will make sure we find an empty bin.

The C/AL code that handles the bin calculation is located in the create Put-Away codeunit (7313). Let's have a look:

```
Code()
IF Location."Directed Put-away and Pick" THEN BEGIN
  BinType.CreateBinTypeFilter(BinTypeFilter,2);
  REPEAT
    QtyToPutAwayBase := RemQtyToPutAwayBase;
    IF NOT (PutAwayTemplLine."Find Empty Bin" OR
      PutAwayTemplLine."Find Floating Bin") OR
      PutAwayTemplLine."Find Fixed Bin" OR
      PutAwayTemplLine."Find Same Item" OR
      PutAwayTemplLine."Find Unit of Measure Match" OR
      PutAwayTemplLine."Find Bin w. Less than Min. Qty"
    THEN BEGIN
        //Calc Availability per Bin Content
        IF FindBinContent("Location Code","Item No.",
          "Variant Code",WarehouseClassCode)
        THEN
          REPEAT
            ...
          UNTIL (BinContent.NEXT(-1) = 0) OR EverythingHandled
      END ELSE BEGIN
        //Calc Availability per Bin
        IF FindBin("Location Code",WarehouseClassCode) THEN
          REPEAT
            IF Bin.Code <> "Bin Code" THEN BEGIN
              ...
            END;
          UNTIL (Bin.NEXT(-1) = 0) OR EverythingHandled
      END
    UNTIL (PutAwayTemplLine.NEXT = 0) OR EverythingHandled;
```

For each record in the Put-away Template Line table the system will try to find a bin. This means that if the rules of the first template line fail, it will use the second template line and so forth.

The two options Find Empty Bin and Find Floating Bin eliminate using the others. If these are true the system will call the `FindBin` function. For the other options it will use the `FindBinContent` function.

Implementing and customizing warehouse management

As there are many ways to setup WMS in Microsoft Dynamics NAV it is important to make the correct decisions in the start of the implementation. Moving the system from one strategy to another is quite a challenge.

It is therefore important to discuss all possibilities and compare them to the way your company works.

A common mistake when implementing WMS software is trying to solve procedural issues with a computer system. The simple rule is: "If it does not work without a computer system, it will most certainly not work with a computer system".

Customizing and changing WMS should be done very carefully since the data flow is very complex, especially for Microsoft Dynamics NAV standards.

Reservations

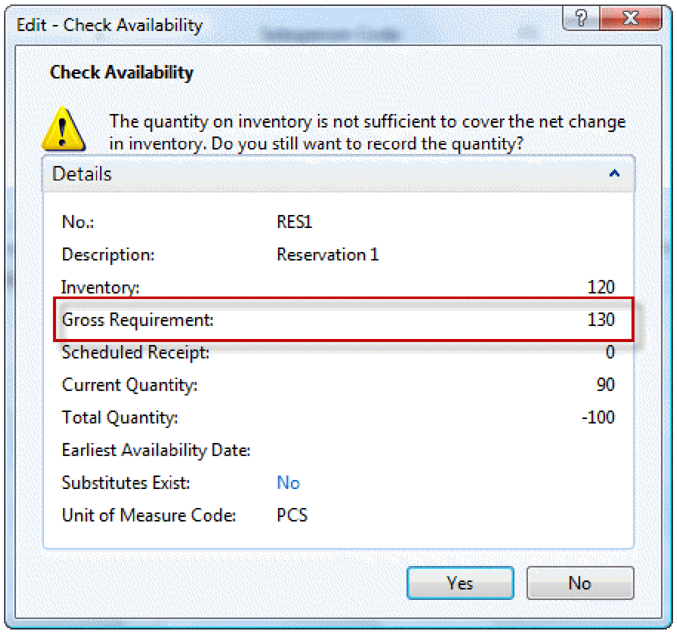
In Microsoft Dynamics NAV it is possible to do reservations on inventory. This can help us managing our inventory more effectively. Let's discuss the reservation process with a customer scenario.

Scenario

One of our customers orders 100 black t-shirts of size M on January 27th 2011. Currently we have 120 on inventory so we can ship them without problems. The customer wants to have them delivered in October on the 18th. We enter a sales order with the shipping date and release the order.

The next day another customer calls for 40 black t-shirts of size M. Our inventory is still 120. This customer wants to have them delivered on the 31st of May. We enter the sales order without a warning.

Lastly, we will create a new sales order for 90 of the same t-shirts with a delivery date of July 25th. Now we get this error message:



Edit - Check Availability

Check Availability

The quantity on inventory is not sufficient to cover the net change in inventory. Do you still want to record the quantity?

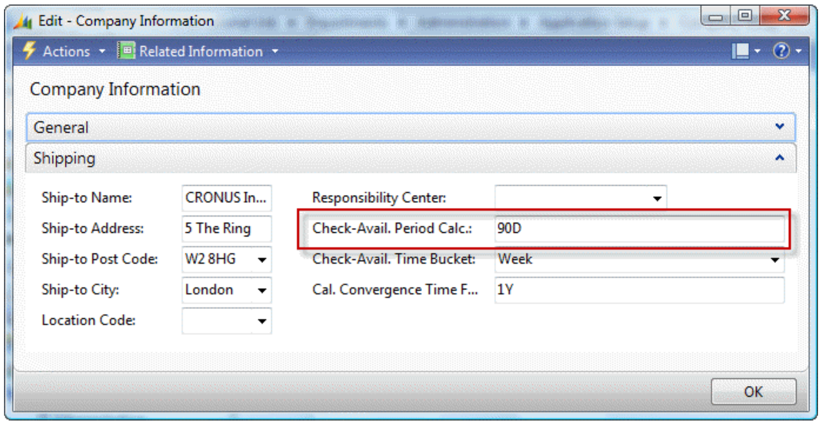
Details

No.:	RES1	
Description:	Reservation 1	
Inventory:		120
Gross Requirement:		130
Scheduled Receipt:		0
Current Quantity:		90
Total Quantity:		-100
Earliest Availability Date:		
Substitutes Exist:	No	
Unit of Measure Code:	PCS	

And if we now go back to the second sales order and re-enter the quantity we will see a similar message.

Check-avail. period calc.

The reason this happens lies in the way Microsoft Dynamics NAV calculates the gross requirement.



Edit - Company Information

Company Information

General

Shipping

Ship-to Name:	CRONUS In...	Responsibility Center:	
Ship-to Address:	5 The Ring	Check-Avail. Period Calc.:	90D
Ship-to Post Code:	W2 8HG	Check-Avail. Time Bucket:	Week
Ship-to City:	London	Cal. Convergence Time F...	1Y
Location Code:			

This is a two step method where first the requirement is calculated until the shipment date of the sales line and then a look-ahead function is called using a date formula which is defined in the Company Information table.

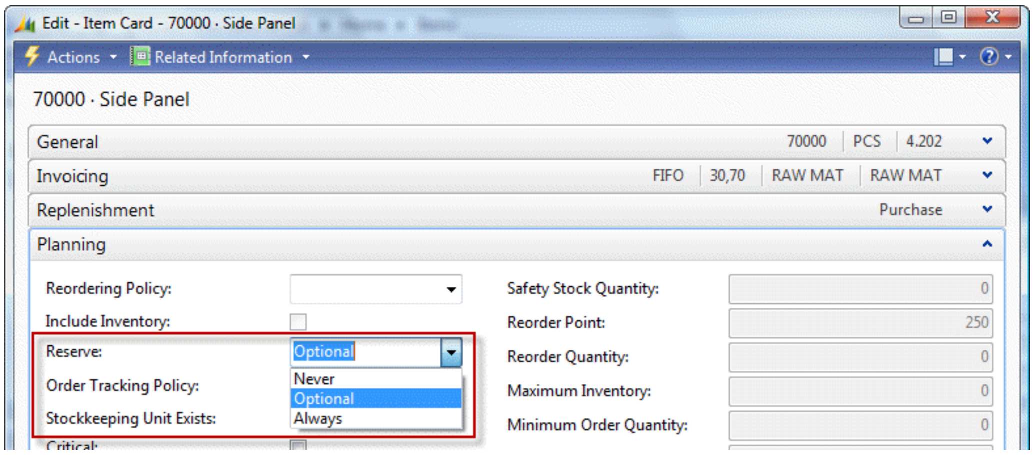
The C/AL code that is used to calculate the look ahead can be found in the function `QtyAvailabletoPromise` in codeunit Available to Promise (5790).

```
QtyAvailabletoPromise
Item.CALCFIELDS(Inventory,"Reserved Qty. on Inventory");
ScheduledReceipt := CalcScheduledReceipt(Item);
GrossRequirement := CalcGrossRequirement(Item);

IF FORMAT(LookaheadDateFormula) <> '' THEN BEGIN
  GrossRequirement :=
    GrossRequirement +
    CalculateLookahead(
      Item,PeriodType,
      AvailabilityDate + 1,
```

If this look-ahead functionality is not detailed enough we can start using the reservation process.

Always versus optional reservation

The reservation option can be activated on the item level and the customer level, and can be set to **Never**, **Optional**, and **Always,** as shown in the following screenshot:



General	70000	PCS	4.202
Invoicing	FIFO	30,70	RAW MAT / RAW MAT
Replenishment	Purchase		

Reordering Policy:		Safety Stock Quantity:	0
Include Inventory:	☐	Reorder Point:	250
Reserve:	Optional	Reorder Quantity:	0
Order Tracking Policy:		Maximum Inventory:	0
Stockkeeping Unit Exists:		Minimum Order Quantity:	0
Critical:	☐		

Let's see what these options signify:

- **Never**: Reservations on this Item or Customer are impossible. If the Item is Reserve Always and the Customer Never, the Item wins.
- **Optional**: It is possible to reserve items for this customer; however, salespersons and warehouse employees can decide to overrule the reservation.
- **Always**: Shipping is not possible without a proper reservation. If the demand is larger than the supply the salespersons and warehouse employees must make manual decisions of who gets what.

Reservation entries

Microsoft Dynamics NAV uses the Reservation Entry (337) table to store the reservation entries.

Reservation entries can be connected to all outstanding documents and journals, and posted entries. This is done using the source fields:

- **Source Type**: An integer field representing the table the record is linked to, for example 37 means sales line and 5406 prod. order line.
- **Source Subtype**: An option field which is linked to the document type field when the record is linked to a sales line or purchase line record or the status of a production order.
- **Source ID**: The link to the document no. of the record this line is linked to.
- **Source Batch Name**: If the record is linked to a journal this field represents the journal batch name. If this field is used the source ID is empty and vice versa.
- **Source Prod. Order Line**: When the record is used for a production order line or component this field represents the production order line number.
- **Source Ref. No.**: An integer field which is used to link the record to a line no. in a document or journal or the production component. If the line is linked to a ledger entry this field represents the entry no. field.

There are four types of Reservation Entries in Microsoft Dynamics NAV represented by the Reservation Status field:

- **Reservation**: These are 'real' reservation entries, which means that a part of the current or future inventory is reserved for a production order or sales order. If the item uses the Always reservation option it is not possible to work around this. If the reservation is optional it is possible that someone else might still use these items in another process.

- **Tracking**: This option is used by the Order Tracking Policy option in Microsoft Dynamics NAV. This is an 'under water' process that can link supply and demand automatically. The status Tracking means that there is a supply as well as a demand.
- **Surplus**: This option is used for both item tracking as discussed in *Chapter 5, Production*, and the Order Tracking Policy. The records can be identified by using the value of the item tracking field. This is set to None for Order Tracking Policy records and Serial No., Lot No. and Lot and Serial No. for item tracking.
- **Prospect**: When item tracking is used, a prospect reservation records indicates an internal journal action, for example, assigning a serial number to an item journal line.

Creating a reservation

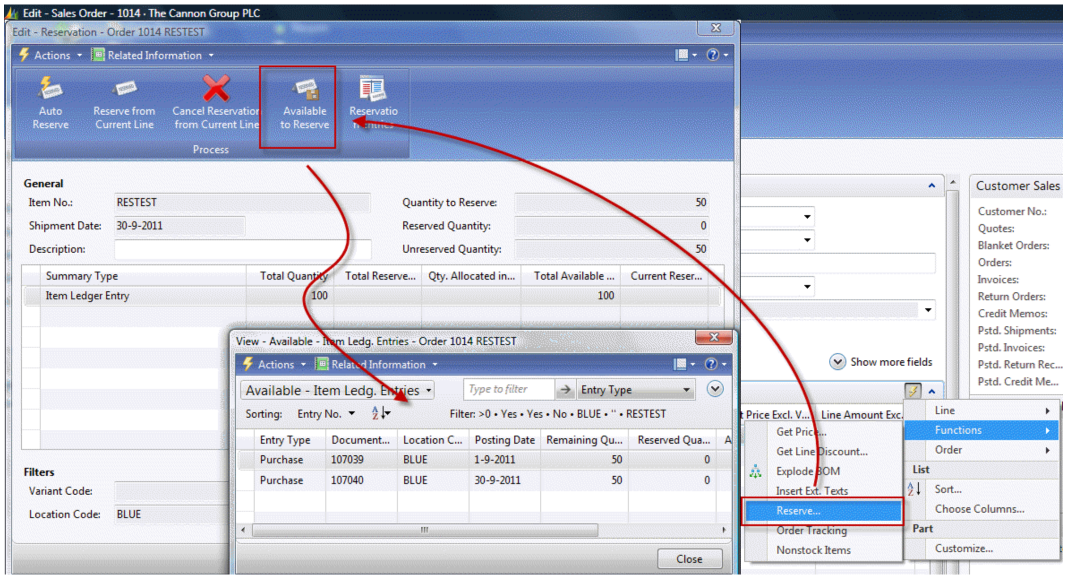
Let's go into the application and create a reservation to see what entries we get in the database.

We will do this using a new item. The item should have a Description, Base Unit of Measure and a Gen. Prod., VAT Prod., and Inventory Posting Group. The default value for Reserve is Optional which we will use for this example. The default costing method is FIFO which we will also use.

To demonstrate the real value of reservations we should create two purchase orders with different dates and unit costs. With FIFO the system would normally apply the sales order to the first item ledger entry. We will reserve on the second item ledger entry to demonstrate the impact on item costing and application.

When this is done we can create a new sales order with one sales line containing the item and half the inventory and select **Functions | Reserve...**

In this screen we can take a look at the available inventory by clicking on **Available to Reserve.**



General

Item No.: RESTEST
Shipment Date: 30-9-2011
Description:
Quantity to Reserve: 50
Reserved Quantity: 0
Unreserved Quantity: 50

Summary Type	Total Quantity	Total Reserve...	Qty. Allocated in...	Total Available ...	Current Reser...
Item Ledger Entry	100			100	

Filters

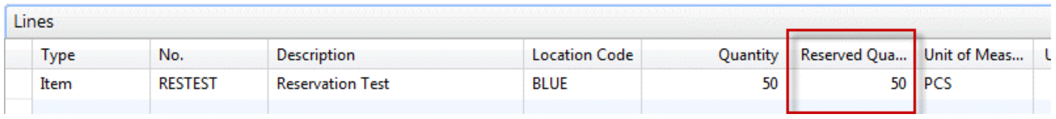
Variant Code:
Location Code: BLUE

Sorting: Entry No.
Filter: >0 • Yes • Yes • No • BLUE • '' • RESTEST

Entry Type	Document...	Location C...	Posting Date	Remaining Qu...	Reserved Qua...
Purchase	107039	BLUE	1-9-2011	50	0
Purchase	107040	BLUE	30-9-2011	50	0

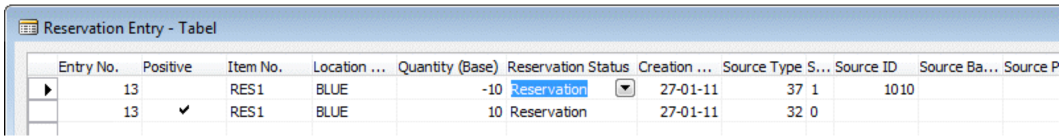
Here we select the second receipt and select **Actions** | **Functions** | **Reserve**. Then we close the screen.

In the sales order lines we can now see the **Reserved Quantity** being **50** as shown in the following screenshot:


Lines

Type	No.	Description	Location Code	Quantity	Reserved Qua...	Unit of Meas...
Item	RESTEST	Reservation Test	BLUE	50	50	PCS

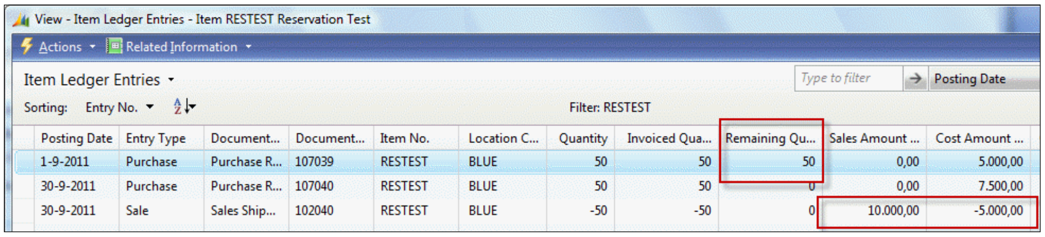
Let's have a look at the reservation entries created in the database by running the table from the Object Designer.


Reservation Entry - Tabel

Entry No.	Positive	Item No.	Location ...	Quantity (Base)	Reservation Status	Creation ...	Source Type	S...	Source ID	Source Ba...	Source P
13		RES1	BLUE	-10	Reservation	27-01-11	37	1	1010		
13	✓	RES1	BLUE	10	Reservation	27-01-11	32	0			

A reservation entry of the type Reservation always uses two lines with the same Entry No. The Source Type for the first entry links to the Sales Line table (37) and the second uses the Item Ledger Entry Table (32) as shown in the previous screenshot.

We ship and invoice the sales order and look at the Item Ledger Entries for out item:



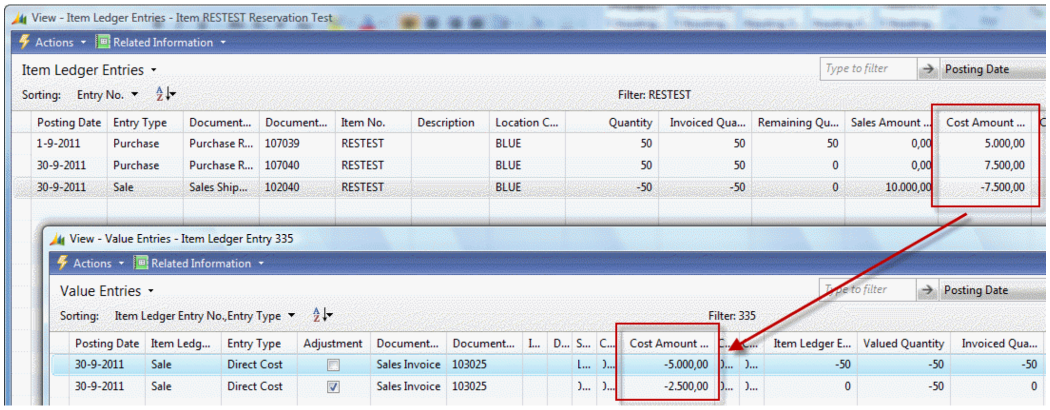
View - Item Ledger Entries - Item RESTEST Reservation Test

Item Ledger Entries

Sorting: Entry No. Filter: RESTEST

Posting Date	Entry Type	Document...	Document...	Item No.	Location C...	Quantity	Invoiced Qua...	Remaining Qu...	Sales Amount ...	Cost Amount ...
1-9-2011	Purchase	Purchase R...	107039	RESTEST	BLUE	50	50	50	0,00	5.000,00
30-9-2011	Purchase	Purchase R...	107040	RESTEST	BLUE	50	50	0	0,00	7.500,00
30-9-2011	Sale	Sales Ship...	102040	RESTEST	BLUE	-50	-50	0	10.000,00	-5.000,00

We see that Microsoft Dynamics NAV has used the second item ledger entry, but the cost is **5000**, not the **7500** from the second entry.



View - Item Ledger Entries - Item RESTEST Reservation Test

Item Ledger Entries

Sorting: Entry No. Filter: RESTEST

Posting Date	Entry Type	Document...	Document...	Item No.	Description	Location C...	Quantity	Invoiced Qua...	Remaining Qu...	Sales Amount ...	Cost Amount ...
1-9-2011	Purchase	Purchase R...	107039	RESTEST		BLUE	50	50	50	0,00	5.000,00
30-9-2011	Purchase	Purchase R...	107040	RESTEST		BLUE	50	50	0	0,00	7.500,00
30-9-2011	Sale	Sales Ship...	102040	RESTEST		BLUE	-50	-50	0	10.000,00	-7.500,00

View - Value Entries - Item Ledger Entry 335

Value Entries

Sorting: Item Ledger Entry No.,Entry Type Filter: 335

Posting Date	Item Ledg...	Entry Type	Adjustment	Document...	Document...	I...	D...	S...	C...	Cost Amount ...	C...	C...	Item Ledger E...	Valued Quantity	Invoiced Qua...
30-9-2011	Sale	Direct Cost	☐	Sales Invoice	103025			I...	)...	-5.000,00	)...	)...	-50	-50	-50
30-9-2011	Sale	Direct Cost	☑	Sales Invoice	103025			)...	)...	-2.500,00	)...	)...	0	-50	0

To correct this we run report Adjust Cost - Item Entries (795) and have another look at the **Item Ledger Entries** and the **Value Entries** to see that it is corrected as shown in the previous screenshot.

Order tracking policy

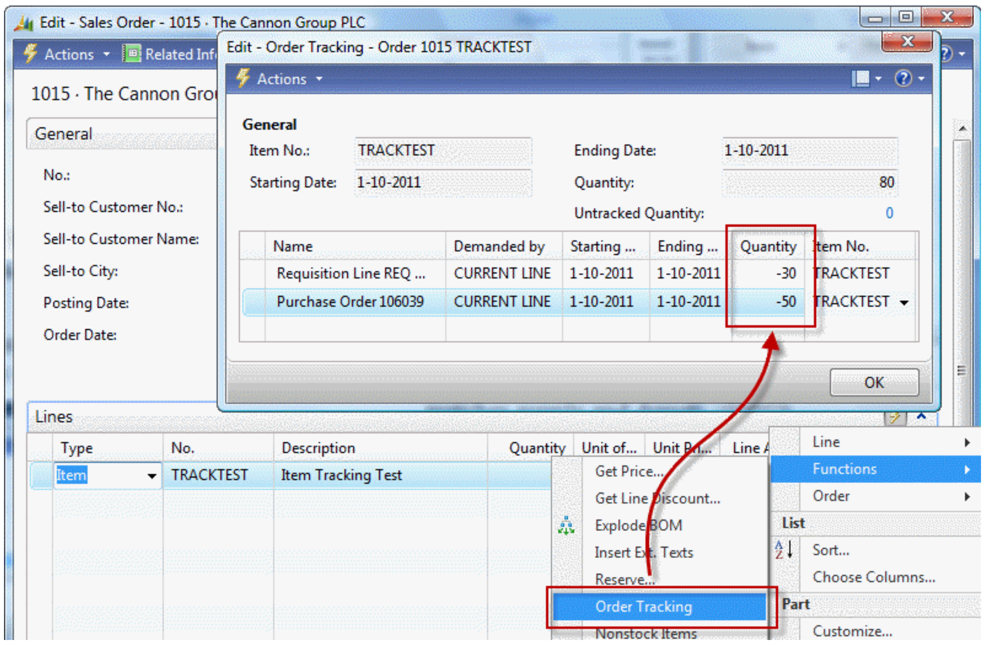
We have seen that reservation entries are not only used for the reservation process of inventory and item tracking but also for balancing supply and demand. This is an internal option within Microsoft Dynamics NAV that allows us to link inventory internally.

The entries are used in the supply and demand calculation to create the Requisition Worksheets.

Example

Let's create a copy of our reservation test item to see the differences between reservations and item tracking. This new item should have the Order Tracking, Policy Tracking and Action Msg.

We will create two purchase orders with both a quantity of 50 without receiving them and create a sales order for the same item with quantity 80.



Edit - Order Tracking - Order 1015 TRACKTEST

General

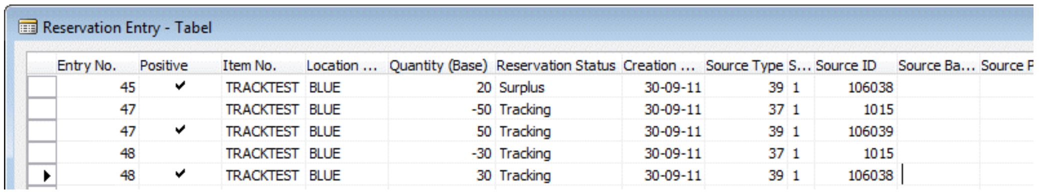
Item No.:	TRACKTEST	Ending Date:	1-10-2011
Starting Date:	1-10-2011	Quantity:	80
		Untracked Quantity:	0

Name	Demanded by	Starting ...	Ending ...	Quantity	Item No.
Requisition Line REQ ...	CURRENT LINE	1-10-2011	1-10-2011	-30	TRACKTEST
Purchase Order 106039	CURRENT LINE	1-10-2011	1-10-2011	-50	TRACKTEST

Type	No.	Description	Quantity	Unit of...	Unit Pri...	Line A
Item	TRACKTEST	Item Tracking Test				

If we now select **Order Tracking** from the sales line Functions we will see that the system matches supply and demand.

Let's have a look at the reservation entries



Reservation Entry - Tabel

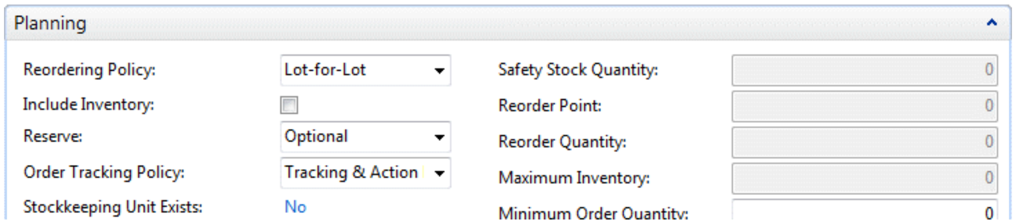
Entry No.	Positive	Item No.	Location ...	Quantity (Base)	Reservation Status	Creation ...	Source Type	S...	Source ID	Source Ba...	Source F
45	✓	TRACKTEST	BLUE	20	Surplus	30-09-11	39	1	106038		
47		TRACKTEST	BLUE	-50	Tracking	30-09-11	37	1	1015		
47	✓	TRACKTEST	BLUE	50	Tracking	30-09-11	39	1	106039		
48		TRACKTEST	BLUE	-30	Tracking	30-09-11	37	1	1015		
48	✓	TRACKTEST	BLUE	30	Tracking	30-09-11	39	1	106038		

We can see that Microsoft Dynamics NAV is now using the **Surplus** and **Tracking** types. The 20 items we have left are not linked to a demand.

Let's start the requisition worksheet for this item and see what Microsoft Dynamics NAV can do with this information.

Replenishment

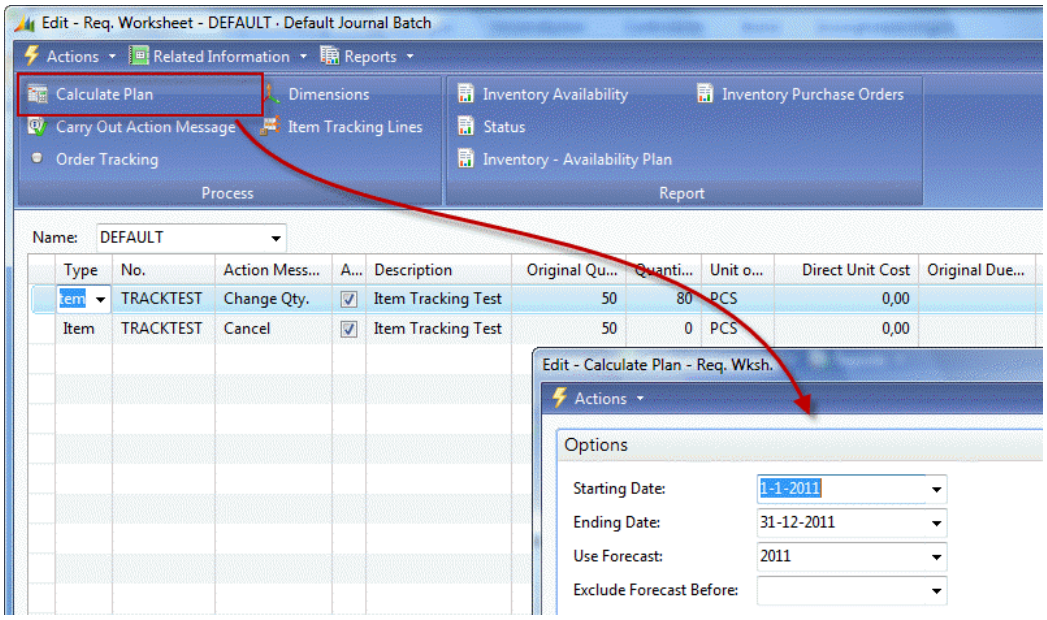
Let's change the reordering policy of the item to **Lot-for-Lot** and run the **Requisition Worksheet** for this item.



Planning

Reordering Policy:	Lot-for-Lot	Safety Stock Quantity:	0
Include Inventory:	☐	Reorder Point:	0
Reserve:	Optional	Reorder Quantity:	0
Order Tracking Policy:	Tracking & Action	Maximum Inventory:	0
Stockkeeping Unit Exists:	No	Minimum Order Quantity:	0

This will result in the suggestion to combine both purchase orders into one document with a different quantity.



Name: DEFAULT

Type	No.	Action Mess...	A...	Description	Original Qu...	Quanti...	Unit o...	Direct Unit Cost	Original Due...
Item	TRACKTEST	Change Qty.	✓	Item Tracking Test	50	80	PCS	0,00	
Item	TRACKTEST	Cancel	✓	Item Tracking Test	50	0	PCS	0,00	

Starting Date:	1-1-2011
Ending Date:	31-12-2011
Use Forecast:	2011
Exclude Forecast Before:	

Trade in vertical industries

Microsoft Dynamics NAV is used in many different vertical industries that often require specific features. Rather than trying to implement all these features in the standard product, Microsoft Dynamics NAV supports the framework, and allows developers to design and create vertical features.

For these features the 80/20 rule applies. Microsoft delivers 80% of the framework which costs 20% of our time to implement. The missing 20% of the functionality is developed at a cost of 80% of the budgeted time.

In this chapter we will discuss how Microsoft Dynamics NAV is used for trade in five different vertical industries. For each industry we will discuss two specific vertical features and how they could be solved.

> Most industries have solid add-on solutions available designed by certified Microsoft Dynamics NAV partners that have been implemented at multiple sites. It is highly recommended to look at those add-on solutions instead of reinventing the wheel and rewrite an add-on that already exists.

Fashion

The Fashion industry has trade periods within the seasons. During the spring, shops need to order the collection for the next winter and in autumn they buy summer clothes.

Sales orders

The sales orders for each collection are created as normal sales orders but with a shipment date in the future, sometimes six months or more ahead. When using variants there should be a separate sales line for each variant, meaning size and color.

This can be quite a hassle to enter for sales people so we could speed this up using a template sales line for the main item and hide the individual sizes.

Using a matrix where the x-axis represents the size and y-axis the color, sales people can quickly enter the quantities. When the matrix is closed we can update the hidden sales lines.

These hidden sales lines are used to calculate the production orders as discussed in *Chapter 5, Production*.

Reservations

When the production orders return from the factory, the warehouse and sales people need to decide which customer gets the items first. This can be done using the shipment date but that might not be completely fair if one customer orders in time, meaning six months ahead, and another customer orders too late with an earlier or the same shipment date.

This is where we can start using reservations. The reservations already support variants but the auto reserve functionality of Microsoft Dynamics NAV might not just do what we like.

Changing this functionality is a complex task. The C/AL code for AutoReserve can be found in codeunit Reservation Management (99000845) but should only be changed by experienced developers.

Fortunately Reservations are layered on top of the normal Inventory, Production, Purchase and Sales process. If we change the algorithm we can remove the current reservations and retest the code to see if the newly created reservation entries are good. This testing process should be done very carefully on a dataset that is small enough to analyze using Microsoft Excel.

Automotive

Microsoft Dynamics NAV is used by many car dealer companies and garages because there are some strong add-on products available for this vertical industry.

On top of the normal trade process supported by Microsoft Dynamics NAV these companies have additional business requirements. Let's discuss two of them.

Vehicle Information

Each vehicle that is sold needs to be configured and ordered. The configuration should be stored in the database for future maintenance and warranty.

We can compare this to Serial Numbers or the Lot No. Information table in the standard product. We could create a new master data table called Vehicle and create a record in this table for each car we configure or sell. The number we create for the Vehicle can be used as a Serial Number in the Item Ledger Entry.

For maintenance we could have a vehicle journal that creates vehicle ledger entries each time the car comes back for servicing. This helps us keep track of the history and should include information such as mileage. The technical design of this solution can be compared to the Squash application we created in *Chapter 2*.

Parts management

In the automotive industry, using the right part is crucial. Different parts can be used on different types of cars and parts often differ between brands.

Many vendors offer their catalogues in digital formats allowing us to create interfaces with them. Parts should be defined as items using standard features such as substitutions.

As many parts can be expensive and have low turnover rate, keeping them on inventory can be expensive thus a minimum inventory should be maintained.

Parts can be connected to vehicle types. For example, a car interior mirror could be used for five types of cars. When a service engineer wants to replace such a mirror he can use a filtered item list of all available parts.

Pharmaceuticals/medicines

In a pharmacy or at another medicine supplier it is normal that not just every customer is allowed to purchase any item. They cannot sell medicines against cardiac arrhythmia to a healthy person.

Even when someone is allowed to use a certain medication it is often limited to a certain number of doses.

People are often insured for the cost of these medicines but most insurance companies require a contribution.

Medication card

Microsoft Dynamics NAV does not support item regulation. To support this we should create new functionality that links items to customers but also allows us to enter the doses and frequency.

From this template we could periodically create sales orders and shipments. Whenever we ship the medicines we need to update the template.

Contribution invoicing

When customers need to do pay a part of the cost of the medication as their own contribution we require the system to create two sales invoices for one sales order.

This is possible using the standard pre-payment functionality in Microsoft Dynamics NAV. We could send a pre-payment invoice to the customer and handle the other invoices to the insurance companies using combined invoicing. The pre-payments will be automatically deducted from the invoice amount but the value entries on the items will remain intact.

However the standard system does not allow us to create a pre-payment invoice to another Bill-to Customer No. This would have to be designed and developed.

Food

Where fashion companies have two or three large ordering moments per year where customers carefully consider what to order, most food companies have a daily ordering process of high volume items.

This ordering process is often done by phone or fax where the retailer calls and tells the call center employees what to ship the next day.

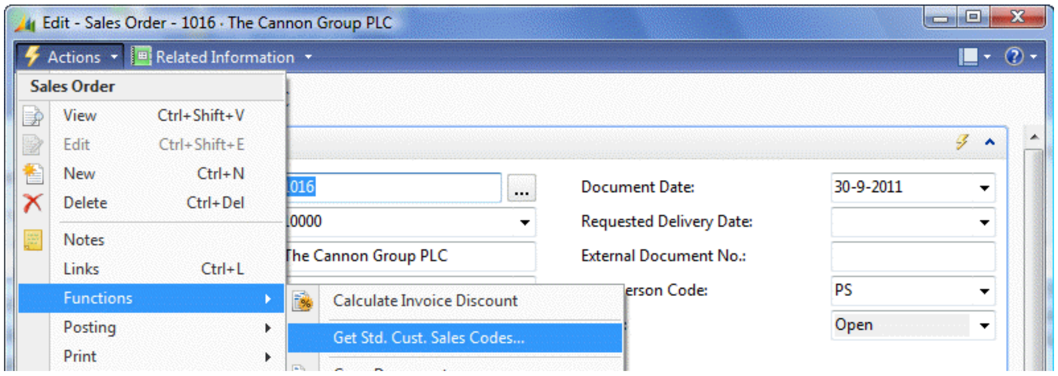
Assortment

Most food companies use an assortment of products. This assortment can change from season to season or contain special action items but is usually stable since that is what most consumers want. Meatballs on Monday and Pork Chops on Tuesday and so on.

To save valuable time creating a new sales order with the same items each day we could have the system do this at night.

This could be done using Standard Customer Sales Codes. This standard function in Microsoft Dynamics NAV allows us to create template sales orders with multiple items or other master data supported by the sales process. It also supports fixed quantities that can be adjusted when the sales orders are created.

The sales order can be created from the customer sales codes using the function **Get. Std. Cust Sales Codes...**.



1016, 10000, The Cannon Group PLC

Document Date: 30-9-2011
Requested Delivery Date:
External Document No.:
erson Code: PS
Open

This function could be scheduled in the Job Scheduler to create new sales order each night for the next day. We will discuss the Job Scheduler in *Chapter 9, Interfacing*.

Fast order entry

When the retailer contacts the call centre to complete the sales order the order entry person should be able to quickly find the correct sales line. If the assortment contains 150 items this can be quite challenging.

This can be solved by implementing a fast order entry functionality that enables users to key in an item number and quantity on the sales header. The values will be updated in the correct sales line and blanked for the next entry.

With this functionality the end user can always work from the same place without searching for the correct sales line.

Furniture

Furniture trading companies have similar issues with variants to the fashion companies with some key differences.

Items such as office chairs and desks have far more options compared to clothes, and when sold most consumers by a few with the same specifications rather than a collection of different sizes.

Variant configuring

The price of furniture is dependant of the configuration, which fabric we want for the seat, the type of armrest, or even the type of wheel. This configuration also determines the item number.

An office chair or desk can have as much as 1200 possible combinations. We do not want to register all these combinations as items or even as variant.

Most furniture suppliers offer online systems or small external software packages to determine the combination. Once the combination is identified we can create a new variant code or see whether the variant already exists and create the sales order.

One-off items

Furniture retailers often have many collections they can sell with thousands of items. Most of the items in these collections will never be sold or as one-off items that are only sold once to one customer.

In this case it does not make sense to create an item with an item cost and inventory value but we want to have some traceability of the item.

This allows for two solutions. We could create a collection item which we can reuse each time we sell an item that is similar to another item but not exactly the same, for example, a lamp with a different foot color. On the sales line we create the possibility for sales people to enter the vendor/collection and an item category. The system should then search for the template item.

Another solution is to create a new item runtime from the sales order. The sales person will also select the vendor/collection and item category and the system should show a list with items already in the database. If the item is not created the system should create the item using a template for the posting groups and so on. The sales person can immediately use it and we have traceability of the items we sold.

Summary

With this chapter we end our quest to explore the production and trade functionality around the items in Microsoft Dynamics NAV.

We discussed the application design of sales and purchase documents in Microsoft Dynamics NAV and how they are mirrored. We've talked about the different Document Types and how they work together from Quote or Blanket Order to Order and Invoice or Return Order, and Credit Memo.

The sales and purchase line validation methodology helps us to calculate the pricing, inventory, and VAT, using a special structure of functions that is linked to the way end users create these lines.

Sales and Purchase orders have a mandatory release process that can be extended with Document Approval and Prepayments.

The items have a two layer inventory process using Item Ledger Entries on Locations and Warehouse Entries on Bins and Zones. We can use Transfer Orders to move items from one location to another and Warehouse Documents to move items from bin to bin and zone to zone. A Warehouse is set up done in the location and can have different levels. The set up level should match the physical process in the warehouse.

Item Application and Costing is tied together with the Reservation process in Microsoft Dynamics NAV. The Reservation Entry table adds a new level to the inventory process linking documents, journals and entries together to level supply and demand. When used, the reservation process can overrule the Costing Method.

At the end of the chapter we talked about different ways to implement this in vertical solutions and what gaps would have to be solved and how. This demonstrates the flexibility and the power of the standard transaction structure in Microsoft Dynamics NAV.

In the next chapter we will take this to a new level where we will design and build a real world vertical solution for Microsoft Dynamics NAV that will enable us to create combine sales shipment in routes for trucks. We will also create a new solution on top of Microsoft Dynamics NAV using the application as a development environment to build something new with respect to the methodology of the application.

7
Storage and Logistics

In the previous chapters, we took a look at how companies work with ERP in the Production and Trade businesses. All these companies work together to bring finished products to the stores where end consumers can buy them.

During this process, the products move around between the companies. This is done using different kinds of transportation, for example trucks, ships, trains, and airplanes. It may also be necessary to store the products in a warehouse until they are sold or moved to the shops.

More and more companies make a decision to outsource logistics rather than having their own transportation. When this is the case, logistics can be a separate part of the supply chain. This chapter discusses this process and the effects on the ERP system.

One of the specific aspects of logistics companies is that the products they handle are not their property. Although they are a part of the total cost of the consumer product, they don't care about the detailed value of their inventory. Logistics companies sell warehouse handling, storage, and transportation as services.

Microsoft Dynamics NAV does not have built-in functionality to handle this so, in this chapter, we will discuss how to design an application to do this.

There are several add-on solutions for this business and in a real world situation, those add-ons should be evaluated as potential solutions. In this chapter, we will discuss how to design and create a basic framework for such an add-on application that can be easily extended without adding too much complexity.

The objects provided with this chapter should never be implemented at a real customer scenario. They are for the purpose of this chapter's examples only.

After reading this chapter, you will have a better understanding of how to design a solid add-on solution and how to integrate this into the standard Microsoft Dynamics NAV product.

How to read this chapter

In this chapter, we will demonstrate how an add-on for Microsoft Dynamics NAV should be designed. In this example, we create a solution for a Storage & Logistics company. This is chosen because the functionality is similar to existing functionality in Microsoft Dynamics NAV (Warehousing) and is a good example of building on top of standard application features.

We will start by analyzing the business process and discuss reasons why we won't use standard application features and explain the modules our new application will have.

The next step is to go deeper into these modules and define the Table and Posting model for each of them. We will then walk through the application like we did in the previous chapters and reverse engineer it to explain how all the pieces where designed.

To do this, we need to download and install the application. As we progress through the chapter, we will discuss most of the objects which can be opened and analyzed in the Microsoft Dynamics NAV classic client.

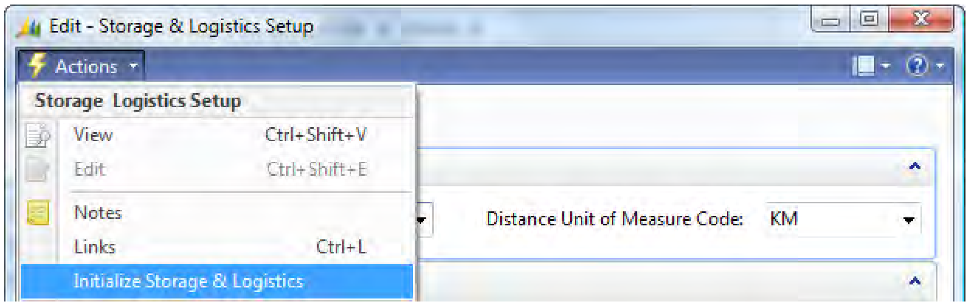
Open the objects as we move along in the chapter to learn more. The objects are rich in functionality which cannot all be discussed in detail in this book.

Chapter objects

With this chapter, some objects and Dynamic-Link library (dll) files are required. The appendix describes how to import and activate them.

After the import process is completed make sure that your current database is the default database for the Role Tailored Client and run Page 123456701, **Storage & Logistics Setup** from the Object Designer in the Classic Client.

From this page, select the option **Initialize Storage & Logistics.**



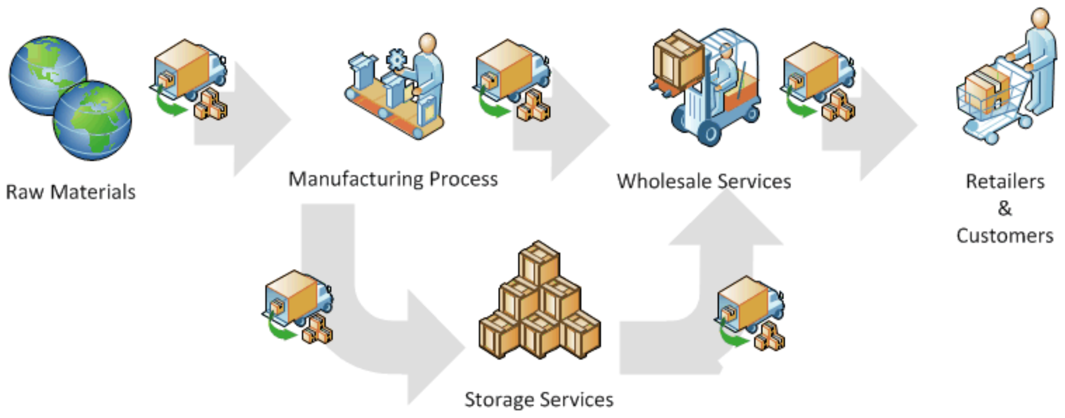
Distance Unit of Measure Code: KM

The process

To design a solid solution for a specific market we first need to analyze the business processes and see where the fits and gaps are with the standard product.

The companies that will be using this solution are logistics providers. These companies do not buy and sell products but sell logistics services like transportation and storage.

There can be various moments in the supply chain where these companies are required. Products are often manufactured in companies all over the world and shipped to consumers elsewhere. Products can cover great distances.



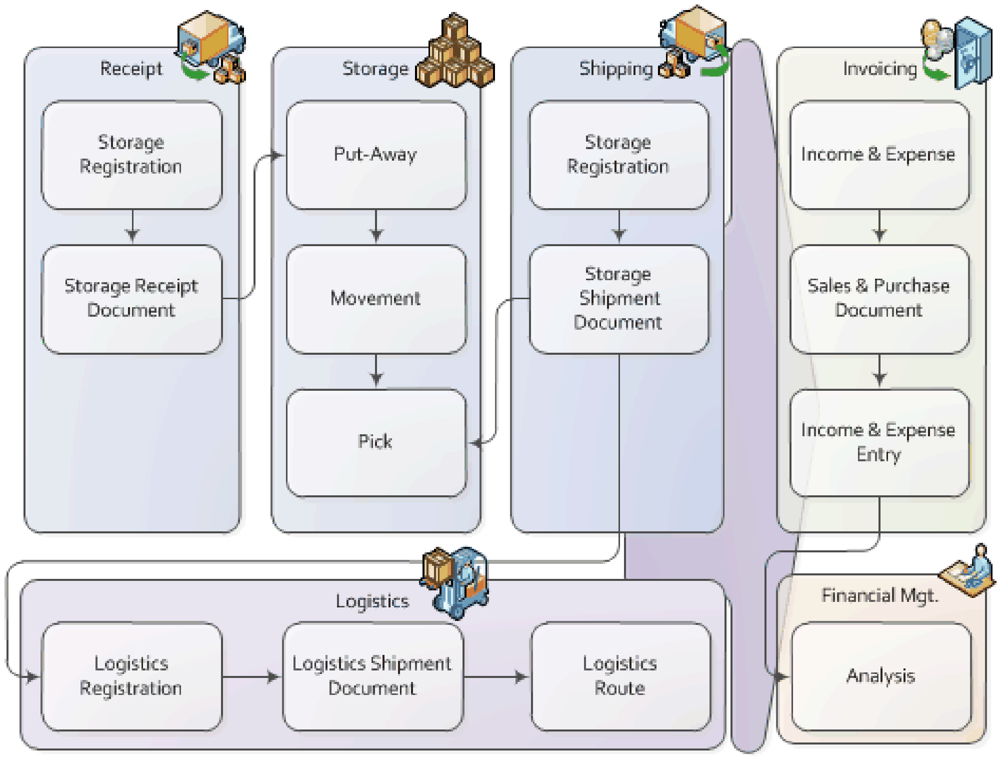
Using standard features

Microsoft Dynamics NAV, like many ERP systems, is designed for people to handle their own products and supports the process of costing like we have seen in the previous chapters. For logistics service providers, this inventory control and valuation functionality is not necessary as the products are not their property. This means that they would want to use the warehouse functionality without the Item Ledger Entries which is very difficult in Microsoft Dynamics NAV.

Logistics service providers also offer transportation solutions. They will pick up the products and deliver them to the customer. The process includes combining different stops in routes resulting in a more cost efficient way of transportation. This functionality is not available in Microsoft Dynamics NAV.

Defining the modules

In this chapter, we will design three new modules on top of Microsoft Dynamics NAV that integrate with each other and could still be used separately. These modules also integrate with the standard application though **Sales and Purchase documents**.



Storage

The first part of the application is the storage module. This allows us to receive and ship products and move them internally in the warehouse.

The design of this module is very similar to the warehouse documents in the standard application that we discussed previously in *Chapter 6*, *Trade*.

Logistics

The logistics module supports the planning of routes delivering the products to the consumers. This is integrated to the storage module, but can also be used from sales shipment documents in the standard application.

For the design of this module, we have looked at the production orders in Microsoft Dynamics NAV in *Chapter 5, Production*. The routes and shipments have a status field that indicate the progress similar to a production order.

Invoicing

The storage and transportation services are then invoiced to the customer periodically or when the products leave the warehouse.

For this, we will use the standard Microsoft Dynamics NAV invoicing solutions but we will add a new **Income & Expenses** module in between the logistical solution and the invoicing functionality.

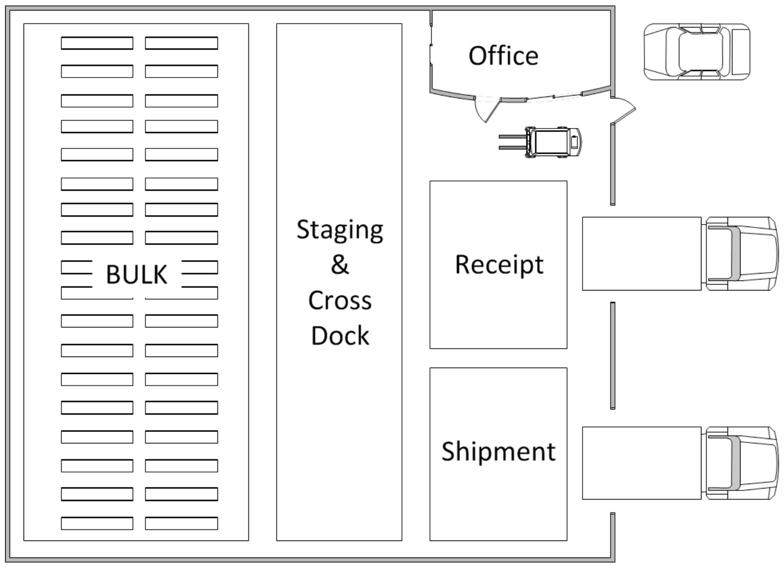
We have looked at the design of Job Ledger Entries and how they are invoiced. This will be discussed in the next chapter.

The storage application

In a storage warehouse, products come and go all the time. A big difference between a storage company and a production plant is that the storage company does not care about the exact products they have but the amount of space they require for storage. The business is selling storage handling, storage space, and transportation.

For our application, we'll assume that our warehouse has a receipt and a shipping region, an 'in between' staging region and a bulk storage region.

If we simplify the warehouse it might look like this floor plan:



- Receipt: When the products come in, they are first unloaded from the truck into a receipt region. This is often located close to the unloading dock so the truck can quickly move on to its next stop after the products are unloaded and the loading documents are checked.
- From the receiving location, the products should be stored away as quickly as possible since another truck might come and the space may be needed again. The products can now go to either the staging region or the bulk region.
- Staging: The staging region is an in-between region where products can be stored that will leave the warehouse quickly or when it is too busy to properly store in the bulk area and we need the space in the receipt region.
- Shipment: When products leave the warehouse they will first be moved to the shipment region. This allows us to quickly load the trucks when they arrive and easily compare the loading documents with the real products.
- Bulk: When we expect products to be in the warehouse for a longer period, they will be stored in the bulk area where we can define shelves. A shelf can have a capacity for one or more products depending on the setup in the system.

Documents

The first step is to set up a registration of what will be coming to our warehouse by creating the 'Receipt documents'. In the old days, we would often receive this information by phone or fax, but today most companies use interfaces such as EDI and web portals for this. This keeps us from making mistakes when typing in the information in the system and allows us to automatically populate the receipt document.

The receipt documents will be combined into 'Put-away documents' that register the transfer from one region to the other. The software will also suggest a shelf to store the products.

When the products leave the warehouse, our customers will also register a 'Shipment document'. On their call, we will start the order picking process and combine the shipments. The 'Pick documents' will tell us on which shelf the products are stored.

Incidentally, it may also be necessary to move products in the warehouse. This will be registered in internal 'Movement documents'.

The storage documents are connected to the logistics document structure, which we will see later in this chapter, while discussing logistics.

Look, learn, and love

In *Chapter 2, A Sample Application* we learned how to use a journal and entry structure to register usage. In this chapter, we will continue with this and add some document structures.

To design our application, we will look at how existing pieces of Microsoft Dynamics NAV are designed and reuse that.

Journal

The core of our application is the **Storage Journal** which is created from the same template as the **Squash Journal** earlier. The difference is that people in a warehouse use documents rather than journals.

Documents

We will support the five types of documents we discussed earlier; Receipt, Shipment, Put-Away, Pick, and Movement. The documents can be created manually by end users or they can be created automatically. We will also provide an interface structure to allow customers to register receipts and shipments.

As all the documents have the same structure and mostly the same fields, they are in the same table to share business logic.

Sharing the same table for multiple document types allows easier sharing of business logic across the application.

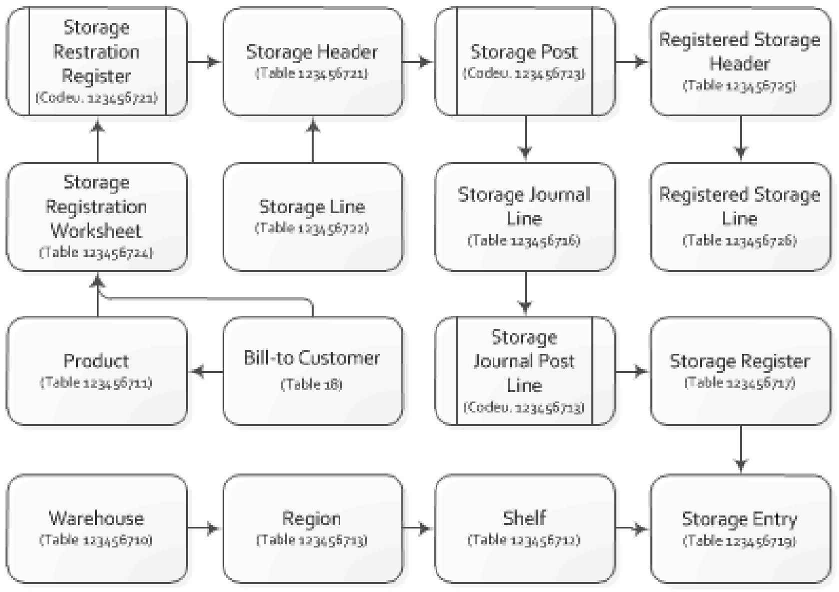
This is also done in the standard Microsoft Dynamics NAV application for sales and purchase documents as we discussed in *Chapter 6*, *Trade*.

Master data

To define what we are storing in the warehouse, we will use a new table called **Product** which is similar to the Item table in the standard system. By creating a new table, we will improve upgradability of our solution and we will be more in control of our own application or in other words, less likely to be impacted by changes that Microsoft may implement in the standard product.

Designing the table and posting schema

If we combined this information into a table and transaction structure, it would look like this diagram:


Storage Restration Register (Codeu. 123456721)
Storage Header (Table 123456721)
Storage Post (Codeu. 123456723)
Registered Storage Header (Table 123456725)
Storage Registration Worksheet (Table 123456724)
Storage Line (Table 123456722)
Storage Journal Line (Table 123456716)
Registered Storage Line (Table 123456726)
Product (Table 123456711)
Bill-to Customer (Table 18)
Storage Journal Post Line (Codeu. 123456713)
Storage Register (Table 123456717)
Warehouse (Table 123456710)
Region (Table 123456713)
Shelf (Table 123456712)
Storage Entry (Table 123456719)

The actual inventory is kept in the **Storage Entries**. By filtering on a warehouse code, region code, or shelf number, the inventory can be calculated.

Sharing tables

The storage and logistics add-on application also has some shared tables. It does not make sense to have a product or warehouse table for each part of the add-on. We also choose to share the setup and the cue tables for the role center definition. The storage and logistics application has four role centers.

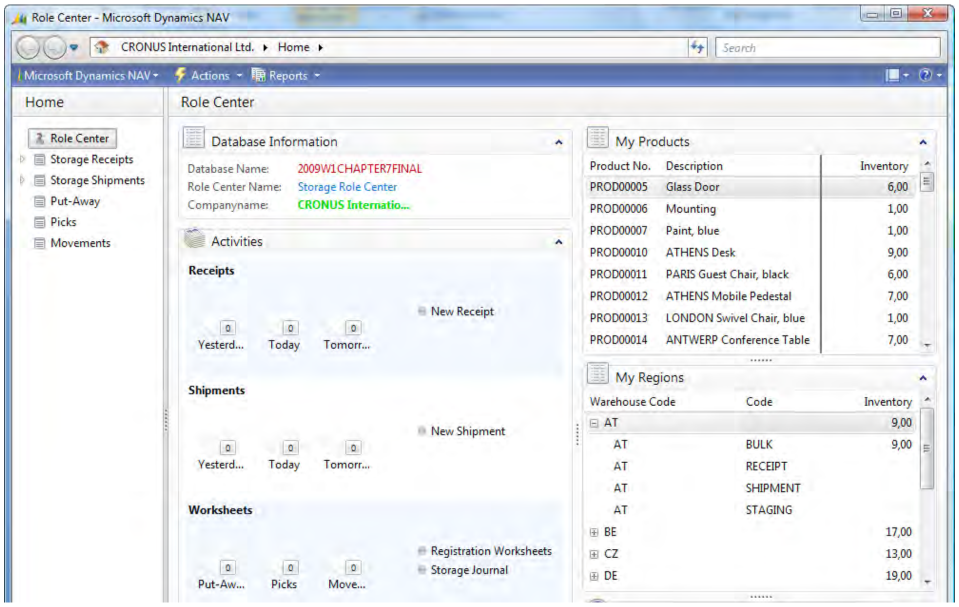
> By sharing the cue table, it is much easier to place the same cues on different role centers. If we were to create one table for each role center, we would need to copy and paste the cue definition to the table for each change request.

Getting started

In our scenario, we'll ship and receive products for a company called CRONUS International Ltd. We have warehouses in Austria, Belgium, Czech Republic, Germany, Great Britain, Iceland, Netherlands, Norway, Sweden, Slovenia, Slovakia, and the USA.

Each warehouse has the same basic layout as explained earlier in this chapter. From the warehouse, we plan routes to transport the products to the consumer.

After initializing the application and restarting the application, the **Role Center** should look like this:



Role Center

Database Information

Database Name:	2009W1CHAPTER7FINAL
Role Center Name:	Storage Role Center
Companyname:	CRONUS Internatio...

My Products

Product No.	Description	Inventory
PROD00005	Glass Door	6,00
PROD00006	Mounting	1,00
PROD00007	Paint, blue	1,00
PROD00010	ATHENS Desk	9,00
PROD00011	PARIS Guest Chair, black	6,00
PROD00012	ATHENS Mobile Pedestal	7,00
PROD00013	LONDON Swivel Chair, blue	1,00
PROD00014	ANTWERP Conference Table	7,00

My Regions

Warehouse Code	Code	Inventory
⊟ AT		9,00
AT	BULK	9,00
AT	RECEIPT	
AT	SHIPMENT	
AT	STAGING	
⊞ BE		17,00
⊞ CZ		13,00
⊞ DE		19,00

The **Activities** window shows the workflow for the warehouse floor. **My Products** contains all customer products we have on inventory and **My Regions** allows us to see what inventory is where in our warehouses.

Opening balance

The opening balance was created using the Storage Journal. By using the journal to create opening entries, we are sure that business rules are followed.

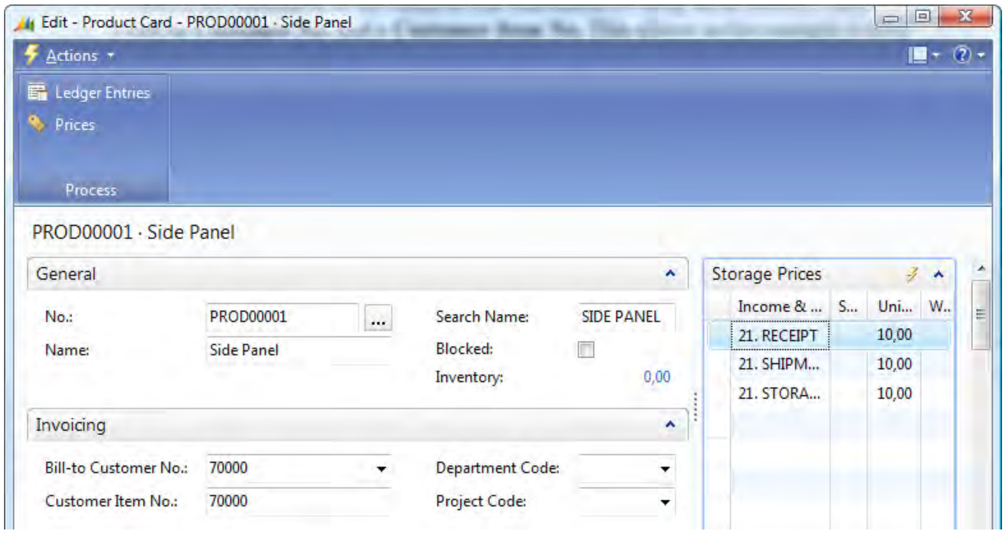
In our design, we decided that end users are not allowed to directly register inventory on the bulk location. We start by receiving it, then we create a put-away document to move it to the bulk location. We'll see how that is done later in this chapter when we discuss the storage documents.

> Have a look in the `Page Storage & Logistics Setup (123456701)` to see how this was done in the function `CreateOpeningBalance()`.

Products

Products are references to the items of our customers that we keep on the inventory. They contain a **Bill-to Customer No.** and a **Customer Item No.** This allows us, for example, to keep item with number 10000 for two different customers.

We can also see and set up the **Storage Prices** for this product, which we will later use for the invoicing.



PROD00001 · Side Panel

General

No.:	PROD00001	Search Name:	SIDE PANEL
Name:	Side Panel	Blocked:	☐
		Inventory:	0,00

Invoicing

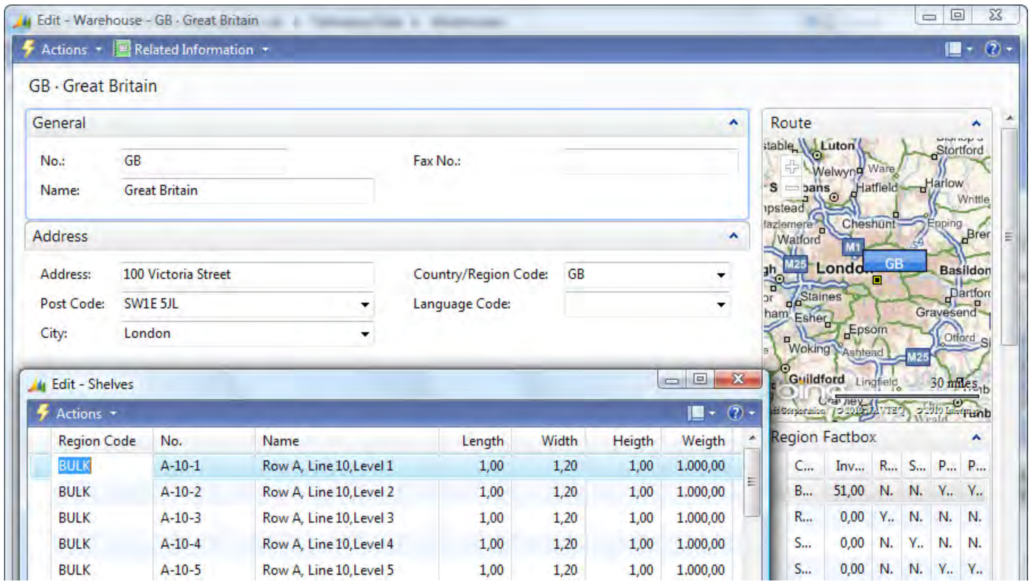
Bill-to Customer No.:	70000	Department Code:	
Customer Item No.:	70000	Project Code:	

Storage Prices

Income & ...	S...	Uni...	W...
21. RECEIPT		10,00	
21. SHIPM...		10,00	
21. STORA...		10,00	

Warehouse

A warehouse is a physical building with an address. To move products from one warehouse to another, we would need to ship them, create a route, and then physically receive them in the other building.



GB · Great Britain

General

No.:	GB	Fax No.:	
Name:	Great Britain		

Address

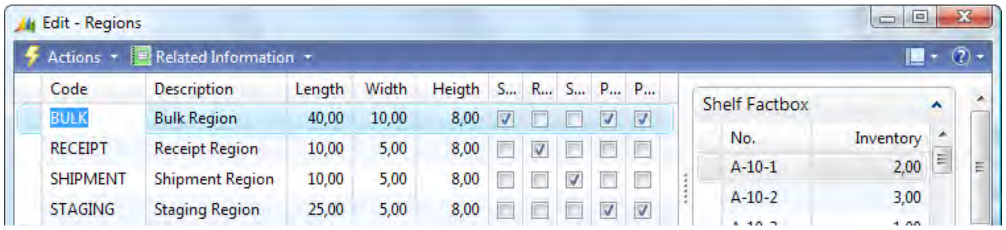
Address:	100 Victoria Street	Country/Region Code:	GB
Post Code:	SW1E 5JL	Language Code:	
City:	London		

Region Code	No.	Name	Length	Width	Heigth	Weigth
BULK	A-10-1	Row A, Line 10, Level 1	1,00	1,20	1,00	1.000,00
BULK	A-10-2	Row A, Line 10, Level 2	1,00	1,20	1,00	1.000,00
BULK	A-10-3	Row A, Line 10, Level 3	1,00	1,20	1,00	1.000,00
BULK	A-10-4	Row A, Line 10, Level 4	1,00	1,20	1,00	1.000,00
BULK	A-10-5	Row A, Line 10, Level 5	1,00	1,20	1,00	1.000,00

C...	Inv...	R...	S...	P...	P...
B...	51,00	N.	N.	Y..	Y..
R...	0,00	Y..	N.	N.	N.
S...	0,00	N.	Y..	N.	N.
S...	0,00	N.	N.	Y..	Y..

Regions

A region is a part of the warehouse that is used for a specific storage activity. In our example, we have a receipt, staging, bulk, and shipment region. To move products from one region to another, we should create a put-away, movement, or pick document.



Code	Description	Length	Width	Heigth	S...	R...	S...	P...	P...
BULK	Bulk Region	40,00	10,00	8,00	✓			✓	✓
RECEIPT	Receipt Region	10,00	5,00	8,00		✓			
SHIPMENT	Shipment Region	10,00	5,00	8,00			✓		
STAGING	Staging Region	25,00	5,00	8,00				✓	✓

No.	Inventory
A-10-1	2,00
A-10-2	3,00

Shelves

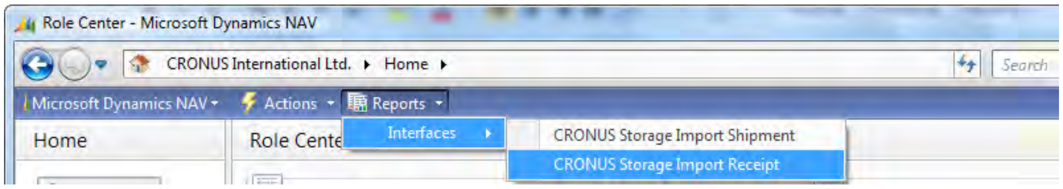
A shelf is a specific part of a region. The specific code of a shelf often indicates its position in the warehouse. For example: Our warehouses have two rows, A and B with 18 lines and 8 levels where each shelf can contain one pallet.

Registration worksheet

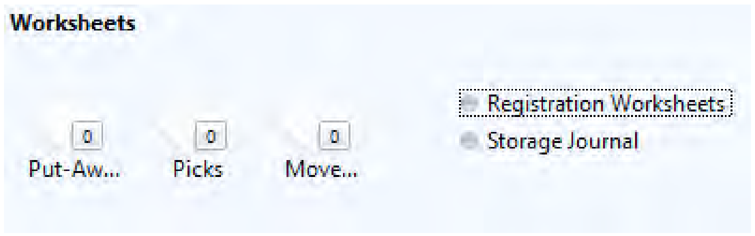
The warehouse process starts with receiving products. To save time when the products arrive on the dock, we ask our customers to register their products in advance. This is done in the storage registration worksheet.

In our application, we have simulated an interface with our customer Cronus International Ltd. We can start the interface from the Role Center directly.

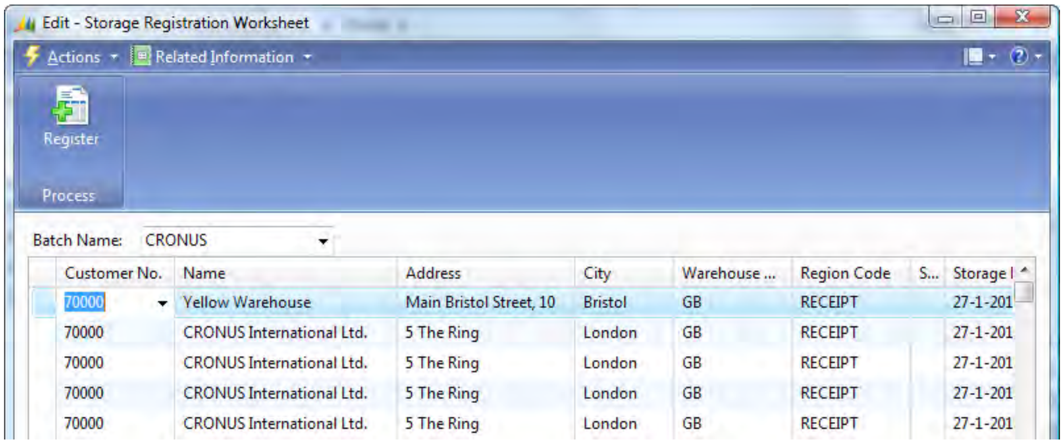
1. We start the report **CRONUS Storage Import Receipt** from the Role Center.


Role Center - Microsoft Dynamics NAV
CRONUS International Ltd. ▸ Home ▸
Search
Microsoft Dynamics NAV
Actions
Reports
Home
Role Cente
Interfaces
CRONUS Storage Import Shipment
CRONUS Storage Import Receipt

2. The system pops up and asks for a **Storage Registration Code**.
3. We will choose CRONUS from the list and start the import process.
4. After this, we open the **Registration Worksheets**.



5. When we now open the registration worksheet, we see what CRONUS will send us today. This allows us to prepare our business, maybe move around some products and schedule resources.



Batch Name: CRONUS

Customer No.	Name	Address	City	Warehouse ...	Region Code	S...	Storage I
70000	Yellow Warehouse	Main Bristol Street, 10	Bristol	GB	RECEIPT		27-1-201
70000	CRONUS International Ltd.	5 The Ring	London	GB	RECEIPT		27-1-201
70000	CRONUS International Ltd.	5 The Ring	London	GB	RECEIPT		27-1-201
70000	CRONUS International Ltd.	5 The Ring	London	GB	RECEIPT		27-1-201
70000	CRONUS International Ltd.	5 The Ring	London	GB	RECEIPT		27-1-201

6. We can now register this worksheet which will create the receipt documents for us.

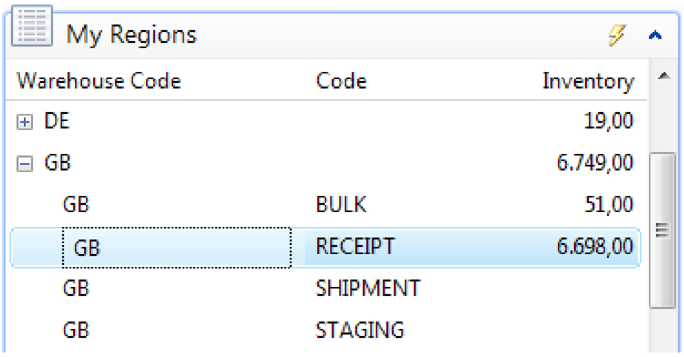
Storage documents

We use documents to determine which product goes where. Creating those documents in the system manually requires a large amount of work, so in our application this is done automatically.

Receipt

By default all products that are received are stored in the **RECEIPT** region. This region does not have shelves. If required, we can change the region code.

After we register the Receipt document, we have inventory in the **RECEIPT** location.



Warehouse Code	Code	Inventory
DE		19,00
GB		6.749,00
GB	BULK	51,00
GB	RECEIPT	6.698,00
GB	SHIPMENT	
GB	STAGING	

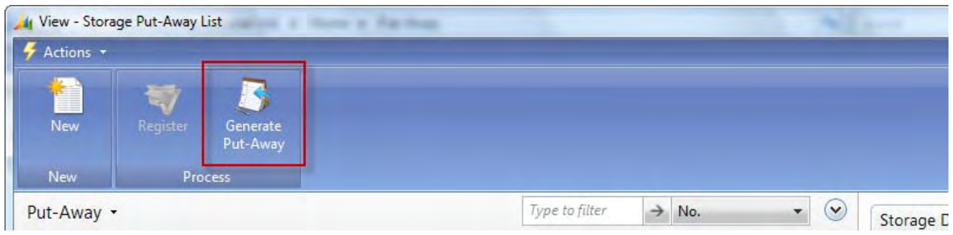
Since this is a relatively small region, we need to move the products to the bulk location as quickly as possible. This is done using a put-away document.

Put-away

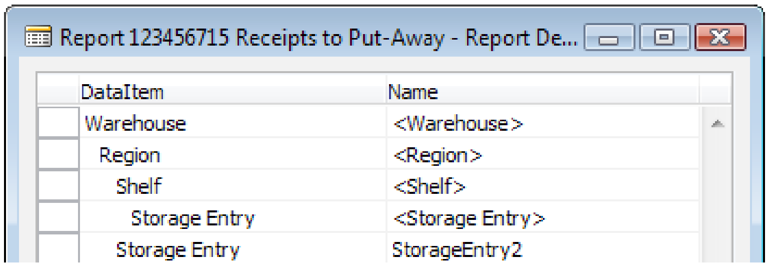
A put-away document is used to move products from the receipt region into the bulk region. The storage entries tell us what is in the receipt region, so we copy that information into a new put-away document. These documents can be created manually, and based on the warehouse information on the document, we can pull the data into the document.

Another requirement is to have an automated process that creates put-away documents based on the entire content of the receipt region.

To provide for this functionality, we have created report **Receipts to Put-Away (123456715)**. This processing-only report reads the storage entries for the receipt region, and creates the put-away documents based on certain predefined rules.



The report filters down the storage entries based regions of type **Receipt** and with inventory.


Report 123456715 Receipts to Put-Away - Report De...

DataItem	Name
Warehouse	<Warehouse>
Region	<Region>
Shelf	<Shelf>
Storage Entry	<Storage Entry>
Storage Entry	StorageEntry2

It creates a put-away document for each warehouse suggesting the first put-away region in the warehouse. For each `Storage Entry`, the function `CreateLine` is started.

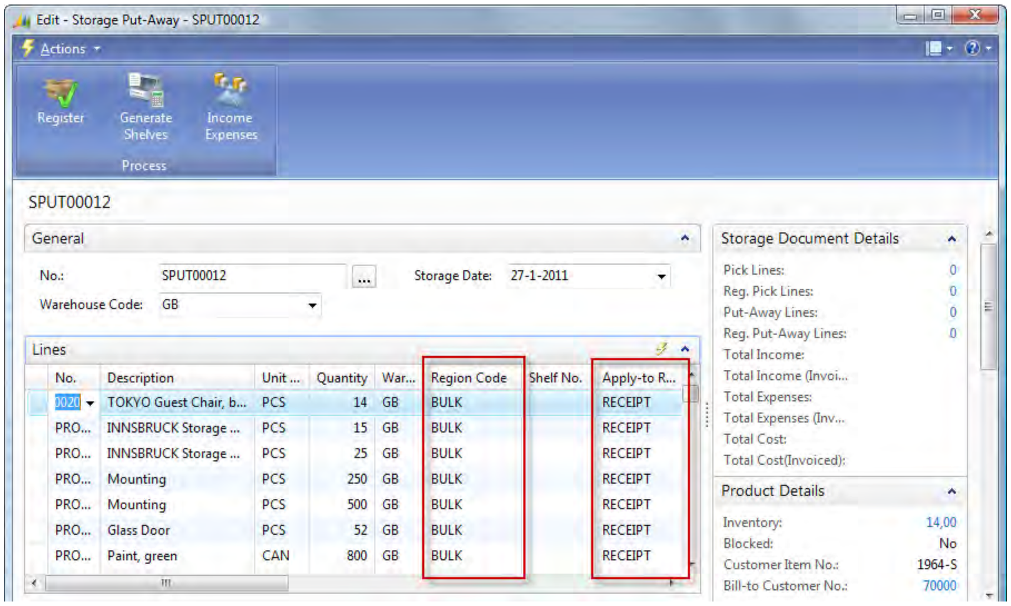
Let's have a look at the C/AL code for this.

```
CreateLineCreateLine()

FindOrCreateStorageHdr;

Region2.SETRANGE("Warehouse Code");
Region2.SETRANGE("Put-Away", TRUE);
Region2.FINDFIRST;

WITH StorageEntry DO BEGIN
  NextLineNo := NextLineNo + 10000;
  StorageLn."Document Type" := StorageHdr."Document Type";
  StorageLn."Document No." := StorageHdr."No.";
  StorageLn."Line No." := NextLineNo;
  StorageLn."No." := "Product No.";
  StorageLn."Warehouse Code" := "Warehouse Code";
  StorageLn."Region Code" := Region2.Code;
```

The first step is to check if it is required to create a new storage document. We create a new document for each `Warehouse` and `Storage Date`.

Then the system filters on the region table to find a Put-Away region. For each `Storage Entry`, a `Storage Line` is created.

After running the report, our put-away document looks like this:



SPUT00012

General

No.: SPUT00012 Storage Date: 27-1-2011
Warehouse Code: GB

Lines

No.	Description	Unit ...	Quantity	War...	Region Code	Shelf No.	Apply-to R...
0020	TOKYO Guest Chair, b...	PCS	14	GB	BULK		RECEIPT
PRO...	INNSBRUCK Storage ...	PCS	15	GB	BULK		RECEIPT
PRO...	INNSBRUCK Storage ...	PCS	25	GB	BULK		RECEIPT
PRO...	Mounting	PCS	250	GB	BULK		RECEIPT
PRO...	Mounting	PCS	500	GB	BULK		RECEIPT
PRO...	Glass Door	PCS	52	GB	BULK		RECEIPT
PRO...	Paint, green	CAN	800	GB	BULK		RECEIPT

Storage Document Details

Pick Lines:	0
Reg. Pick Lines:	0
Put-Away Lines:	0
Reg. Put-Away Lines:	0
Total Income:	
Total Income (Invoi...	
Total Expenses:	
Total Expenses (Inv...	
Total Cost:	
Total Cost(Invoiced):	

Product Details

Inventory:	14,00
Blocked:	No
Customer Item No.:	1964-S
Bill-to Customer No.:	70000

The suggested **Region Code** is **BULK** and the **Apply-to Region Code** is **RECEIPT**.

If we now try to register this document, we should receive an error since we did not enter any **Shelves** because this is mandatory on this region.

[This check is done in the Codeunit Storage Jnl.-Check Line. By moving these checks into this Codeunit, we make sure these rules are mandatory in each posting.]

Since we rely on the system to keep track of our inventory we can also have it suggest available shelves for us. This is also done using batch report 123456716 **Generate Put-Away Shelves**.

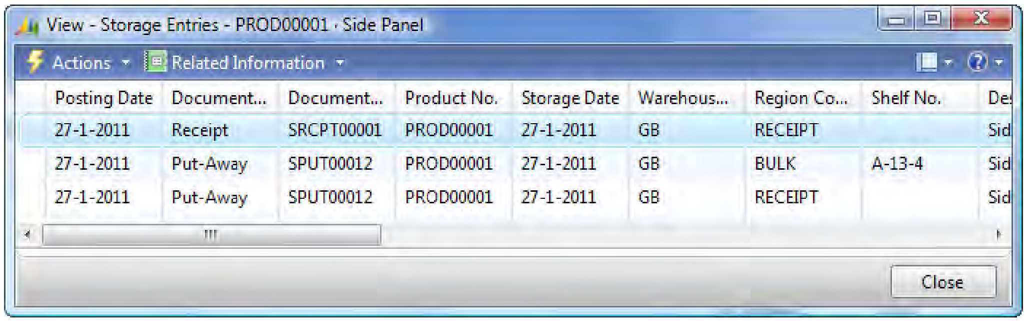
Let's design the report and look at the C/AL code in the `Storage Line DataItem`.

```
Storage Line - OnAfterGetRecord()

Counter := Counter + 1;
Window.UPDATE(1,"Document No.");
Window.UPDATE(2,ROUND(Counter / CounterTotal * 10000,1));

Shelf.SETRANGE("Warehouse Code", "Storage Line"."Warehouse Code");
Shelf.SETRANGE("Region Code", "Region Code");
Shelf.SETRANGE(Inventory, 0);
Shelf.SETRANGE("Blocked by Storage", FALSE);
Shelf.FINDFIRST;

"Shelf No." := Shelf."No.";
MODIFY;
```

For each `Storage Line` in the put-away document, it finds another shelf by filtering on availability based on `Inventory` and `Blocked by Storage`.

The `Blocked by Storage` field is a flow field that returns true if the shelf is used on a Warehouse Document preventing two forklift trucks from stopping at the same shelf.

When this report is executed, we can register this Put-Away document and we can see the Storage Entries that are generated from the Product Card using the **Ledger Entries** Action.


View - Storage Entries - PROD00001 · Side Panel

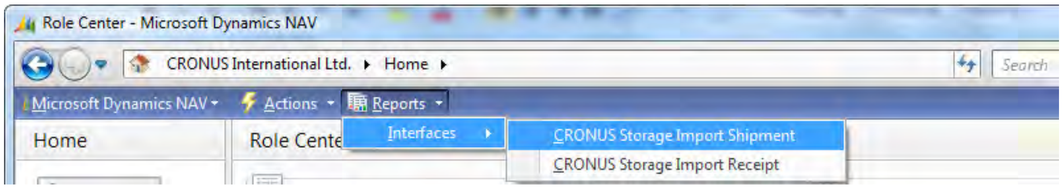
Posting Date	Document...	Document...	Product No.	Storage Date	Warehous...	Region Co...	Shelf No.	De
27-1-2011	Receipt	SRCPT00001	PROD00001	27-1-2011	GB	RECEIPT		Sid
27-1-2011	Put-Away	SPUT00012	PROD00001	27-1-2011	GB	BULK	A-13-4	Sid
27-1-2011	Put-Away	SPUT00012	PROD00001	27-1-2011	GB	RECEIPT		Sid

Here we can see that the Put-Away has applied its entries to the Receipt entries. Since we moved everything, the original entry is closed and the remaining quantity is set to zero.

This functionality is similar to what we created in *Chapter 2, A Sample Application* when applying an invoice entry to a reservation.

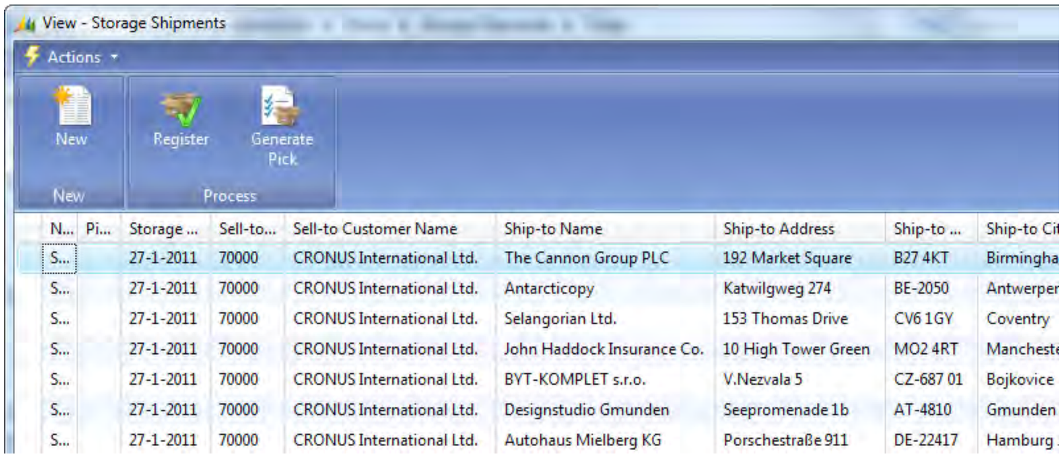
Shipment

After a while, when the products are on inventory, the customer may send a request to ship them. The shipping documents are sent using the same interface as the receipt documents.


Role Center - Microsoft Dynamics NAV
CRONUS International Ltd. ▸ Home ▸
Search
Microsoft Dynamics NAV
Actions
Reports
Interfaces
CRONUS Storage Import Shipment
CRONUS Storage Import Receipt
Home
Role Cente

Running this will create the Storage Registration Worksheet which we can check and register to a shipment document the same way as the Receipt Documents earlier.

The system creates a shipment document for each **Ship-to Address**.



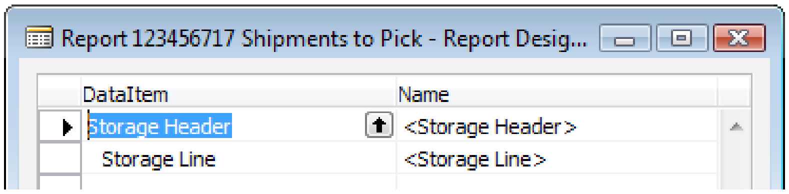
N...	Pi...	Storage ...	Sell-to...	Sell-to Customer Name	Ship-to Name	Ship-to Address	Ship-to ...	Ship-to Cit
S...		27-1-2011	70000	CRONUS International Ltd.	The Cannon Group PLC	192 Market Square	B27 4KT	Birmingha
S...		27-1-2011	70000	CRONUS International Ltd.	Antarcticopy	Katwilgweg 274	BE-2050	Antwerper
S...		27-1-2011	70000	CRONUS International Ltd.	Selangorian Ltd.	153 Thomas Drive	CV6 1GY	Coventry
S...		27-1-2011	70000	CRONUS International Ltd.	John Haddock Insurance Co.	10 High Tower Green	MO2 4RT	Mancheste
S...		27-1-2011	70000	CRONUS International Ltd.	BYT-KOMPLET s.r.o.	V.Nezvala 5	CZ-687 01	Bojkovice
S...		27-1-2011	70000	CRONUS International Ltd.	Designstudio Gmunden	Seepromenade 1b	AT-4810	Gmunden
S...		27-1-2011	70000	CRONUS International Ltd.	Autohaus Mielberg KG	Porschestraße 911	DE-22417	Hamburg

We now have to start the process of moving the products from the storage region to the shipment region.

Picks

The products that will be shipped need to be picked from the bulk or staging region using a Pick Document. As with the Put-Away functionality, our application design provides an automated process that supports this process.

To create the document, we use batch report **123456717 Shipments to Pick**.


Report 123456717 Shipments to Pick - Report Desig...

DataItem	Name
Storage Header	<Storage Header>
Storage Line	<Storage Line>

This report can combine shipments into one or more pick documents.

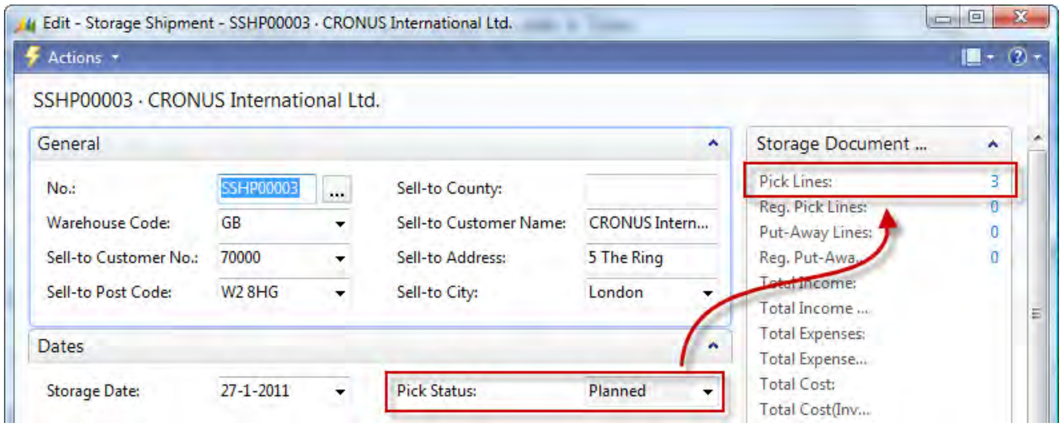
```
Storage Line - OnAfterGetRecord()

Counter := Counter + 1;
Window.UPDATE(1,"Document No.");
Window.UPDATE(2,ROUND(Counter / CounterTotal * 10000,1));

Product.GET("No.");
Product.SETRANGE("Warehouse Filter Code", "Warehouse Code");
Product.CALCFIELDS(Inventory);
```

```
IF Quantity > Product.Inventory THEN
  ERROR(Text001, Quantity, Product.Inventory, "No.");

QtyToPick := Quantity;

StorageEntry.SETCURRENTKEY("Product No.");
StorageEntry.SETRANGE("Warehouse Code", "Storage Header"."Warehouse
Code");
StorageEntry.SETRANGE("Product No.", "No.");
StorageEntry.SETRANGE(Open, TRUE);
IF StorageEntry.FINDSET(TRUE) THEN REPEAT
  StorageEntry.CALCFIELDS("Blocked by Storage");
  IF NOT StorageEntry."Blocked by Storage" THEN BEGIN
    IF QtyToPick >= StorageEntry.Quantity THEN
      QtyToPick := QtyToPick - StorageEntry.Quantity
    ELSE BEGIN
      StorageEntry.Quantity := QtyToPick;
      QtyToPick := 0;
    END;
    CreateLine(StorageEntry);
  END;
UNTIL (StorageEntry.NEXT = 0) OR (QtyToPick = 0);

IF QtyToPick > 0 THEN
  ERROR(Text002, "No.");
```

First the system checks if the products are on inventory in this warehouse. If they are, it starts browsing though the storage entries to look for available shelves. Here we also use the **Blocked by Storage** flow field to avoid two employees fighting over the same products.

One of the functional requirements in our application is not to have half a shipment be picked and block the SHIPMENT region being incomplete. If there is not enough inventory available for the pick, the system should error out.

After the pick is created, we update a field **Pick Status** on the **Shipment**. We can see that there are three **Pick Lines** attached to this shipment.


Edit - Storage Shipment - SSHP00003 · CRONUS International Ltd.

SSHP00003 · CRONUS International Ltd.

General

No.:	SSHP00003	Sell-to County:	
Warehouse Code:	GB	Sell-to Customer Name:	CRONUS Intern...
Sell-to Customer No.:	70000	Sell-to Address:	5 The Ring
Sell-to Post Code:	W2 8HG	Sell-to City:	London

Dates

Storage Date:	27-1-2011	Pick Status:	Planned

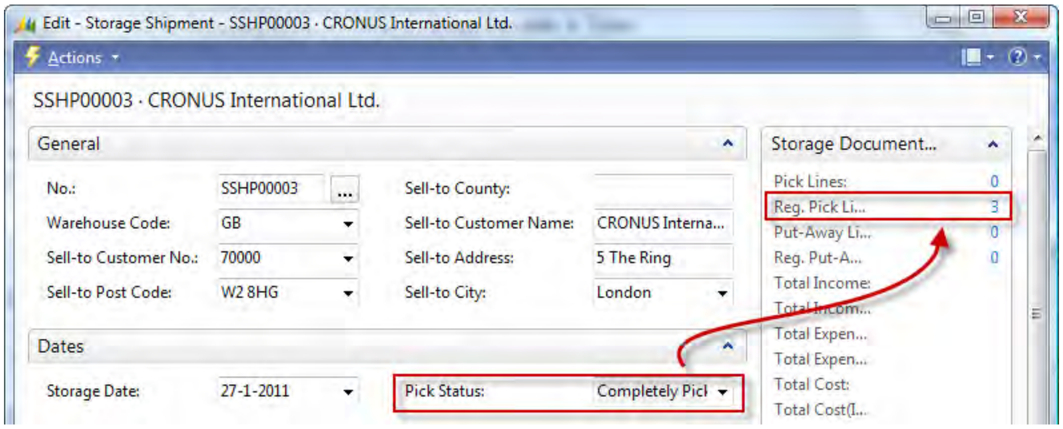
Storage Document ...

Pick Lines:	3
Reg. Pick Lines:	0
Put-Away Lines:	0
Reg. Put-Awa...	0
Total Income:	
Total Income ...	
Total Expenses:	
Total Expense...	
Total Cost:	
Total Cost(Inv...	

When we click on the **3** the system opens the lines. Double clicking on the lines will open the **Pick Document**.

[To influence the double click event, assign the `RETURN` shortcut to one of the actions on a page.]

After registration of the Pick Document, the status of the Shipment moves to **Completely Picked**. We can see that the Pick Lines are registered.


Edit - Storage Shipment - SSHP00003 · CRONUS International Ltd.

SSHP00003 · CRONUS International Ltd.

General

No.:	SSHP00003	Sell-to County:	
Warehouse Code:	GB	Sell-to Customer Name:	CRONUS Interna...
Sell-to Customer No.:	70000	Sell-to Address:	5 The Ring
Sell-to Post Code:	W2 8HG	Sell-to City:	London

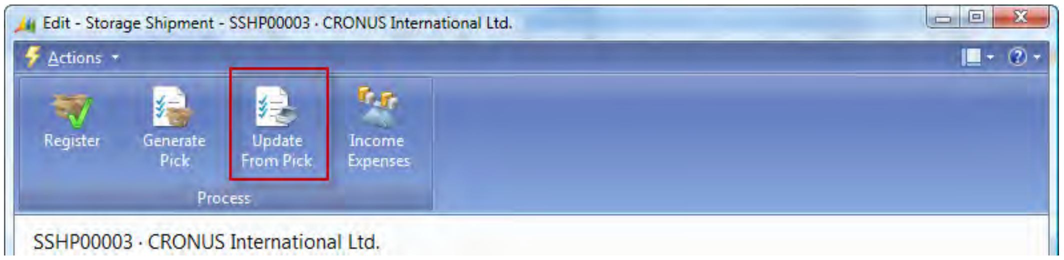
Dates

Storage Date:	27-1-2011	Pick Status:	Completely Pick

Storage Document...

Pick Lines:	0
Reg. Pick Li...	3
Put-Away Li...	0
Reg. Put-A...	0
Total Income:	
Total Incom...	
Total Expen...	
Total Expen...	
Total Cost:	
Total Cost(I...	

The last step before the shipment can be registered is updating the Storage Lines with the Apply-to Storage Entry No. from the Pick document. For this step, we have designed a dedicated report `Update Storage Shipment (123456718)` that can be started from the **Storage Shipment** document.


Edit - Storage Shipment - SSHP00003 · CRONUS International Ltd.
Actions
Register
Generate Pick
Update From Pick
Income Expenses
Process
SSHP00003 · CRONUS International Ltd.

After this, the shipment can be registered. The products have now left our warehouse and are on the road to the customer.

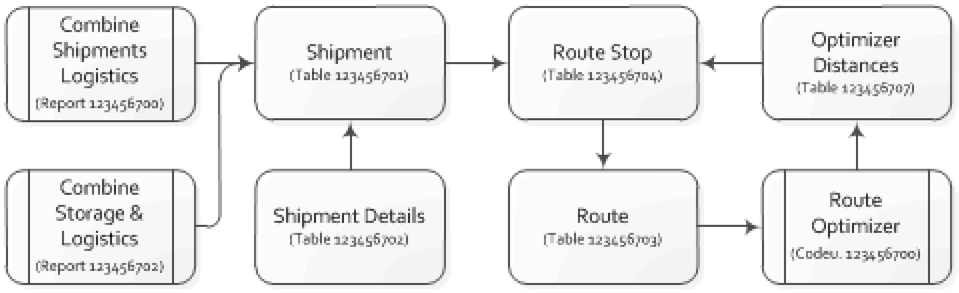
The logistics application

Similar to Production Orders in the standard application, the processes in our logistics application are status driven rather than transaction driven. This is why this part of the application does not have a journal with entries. The tables can have archived copies but they are not part of a normal registering or posting routine.

[For the examples in this part of the chapter, we should change the default Role Center to `Logistics Role Center (123456700)` in the `Profile table (2000000072)`.]

Designing the table and posting schema

If we look at the structure of the logistics application, we can see that the typical posting transactions are missing. The application uses a status-driven workflow based on events which are defined in the triggers of the tables.


Combine Shipments Logistics (Report 123456700)
Shipment (Table 123456701)
Route Stop (Table 123456704)
Optimizer Distances (Table 123456707)
Combine Storage & Logistics (Report 123456702)
Shipment Details (Table 123456702)
Route (Table 123456703)
Route Optimizer (Codeu. 123456700)

The logistics shipment and shipment details have a lot of similarity with the shipments from the warehouse. Still we have chosen to move them into new tables for the following reasons:

- Security: In Microsoft dynamics NAV, the table level is the most important for security. If we would share this table, it would be impossible to set up users to have access to logistics and not to warehouse or vice-versa.
- Locking: If two departments use the same table for different purposes they will most likely have a different locking mechanism. For example, in logistics, shipments are bound to the route object. The warehouse shipments are bound to other shipment documents.
- Filtering the same table in main processes in different ways will significantly increase the probability of blocks and deadlocks.
- Table size: The storage documents are registered shortly after they are created. Most documents are deleted and moved to registered tables the same day that they are created. Logistics shipments have a longer lifecycle.
- It takes longer to take the products from our warehouse to the customer and during this process, many things can go wrong because of outside events. The transport tables may be periodically cleaned up like Manufacturing or Jobs in the standard Microsoft Dynamics NAV product.

Getting started

To start the logistics process, we could create some shipments manually but the application also provides an interface to the Sales shipments and Warehouse shipments.

Let's start the option **Combine Shipments (Sales)** from the **Activities** on the **Logistics Role Center** to generate some data to work with.

Shipments

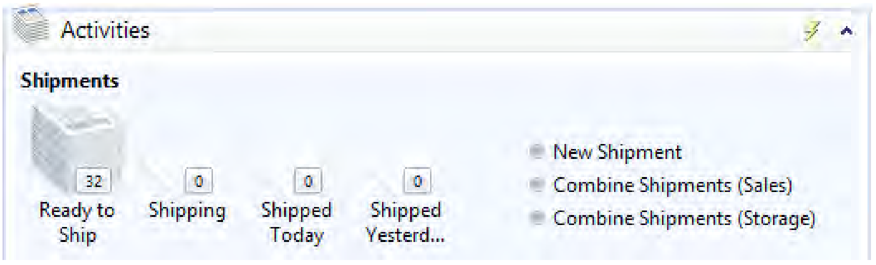
Logistics shipments are products moving from one physical address to another physical address.

In our example, the shipments from our warehouse to the customer are created, but a shipment can also be from another address to a customer.

Tracking the status of a shipment is important for the planners. A shipment starts with the status **Ready to Ship** as soon as all mandatory fields are checked.

When the shipments are combined into routes the shipment moves to the **Shipping** status. During this stage, the products are picked up from the warehouse. When this happens the **Pickup Date Time** is populated. This is done from the route.

After delivery the **Delivery Date Time** is populated and the status is set to **Shipped**.



The planners can follow the shipments from their Role Centers in a workflow.

Routes

Shipments are combined into a route. In order to make a product planning, it is very important that the planners get the shipment details correct. The length, width, height, and weight of the products determine if they can be combined into a truck, ship, airplane, or train.

Our example add-on system has a report to combine shipments into a route. The shipments in a route will be combined into stops if they have the same address information.

Combining shipments

This is done in report `Shipment To Route & Warehouse (123456701)`. The shipments are grouped per warehouse. For each warehouse, a new route is created.

For each shipment, the system creates a route stop. The stops have different types: Pickup, Delivery, Pickup group, and Delivery group.

Each shipment then gets a Pickup and Delivery stop.

Shipment - OnAfterGetRecord()

```
IF Route.Description <> Warehouse.Name THEN BEGIN
  Route."No." := '';
  Route.Description := Warehouse.Name;
  Route."Shipment Date" := WORKDATE;
  Route.Status := Route.Status::Planned;

  Route."Bill-to Customer No." := "Bill-to Customer No.";
  Route."Bill-to Name" := "Bill-to Name";
  Route.INSERT(TRUE);
  i := 0;
END;
i := i + 10000;
RouteStop."Route No." := Route."No.";
RouteStop."Line No." := i;
RouteStop.Type := RouteStop.Type::Pickup;
RouteStop.VALIDATE("Shipment No.", "No.");
RouteStop.INSERT;
i := i + 10000;
RouteStop."Route No." := Route."No.";
RouteStop."Line No." := i;
RouteStop.Type := RouteStop.Type::Delivery;
RouteStop.VALIDATE("Shipment No.", "No.");
RouteStop.INSERT;
```

After the routes are created and the shipments are assigned to a stop, a grouping and optimizing algorithm is started. This is Codeunit `Route Optimizer (123456700)`.

Route optimizer

The algorithm in our example is designed to find the optimal route to deliver the products to the addresses by calculating the distance to each address from the warehouse. The route starts from the address that is closest to our warehouse and ends at the address that is the farthest away.

This is just an example of a simple algorithm. Each company will have its own algorithm that needs to be implemented.

```
RouteStopPickup.SETRANGE("Route No.", Route."No.");
RouteStopPickup.SETRANGE(Type, RouteStopPickup.Type::Pickup);
RouteStopPickup.FINDFIRST;

RouteStopDelivery.SETRANGE("Route No.", Route."No.");
RouteStopDelivery.SETRANGE(Type, RouteStopDelivery.Type::Delivery);
RouteStopDelivery.FINDSET;
REPEAT
  Window.UPDATE(2, RouteStopDelivery."Shipment No.");

  IF NOT Optimizer.GET(RouteStopDelivery.Name) THEN BEGIN
    CLEAR(BingMapMgt);
    BingMapMgt.CalculateRoute('', RouteStopPickup.Latitude,
RouteStopPickup.Longitude,'', RouteStopDelivery.Latitude,
      RouteStopDelivery.Longitude, Optimizer."Distance
(Distance)",Optimizer."Activity Time", Optimize::Distance);

    Optimizer.Name := RouteStopDelivery.Name;
    Optimizer.Latitude := RouteStopDelivery.Latitude;
    Optimizer.Longitude := RouteStopDelivery.Longitude;
    Optimizer.INSERT;
  END;

UNTIL RouteStopDelivery.NEXT = 0;
```

The calculation of the distance is done by calling a web service from Bing Maps. This is explained in the interface chapter.

Each distance is stored as a record into the Optimizer table which is a helper table. This table is a temporary variable in this Codeunit.

Temporary tables have multiple benefits that make them interesting to use. As they are not stored in the database, they have much better performance compared to real tables. This also has a benefit for concurrency since there can be no locking.

Temporary tables are free to use. They are not checked in the license file when used. To create and modify the definition, a valid license is still required.

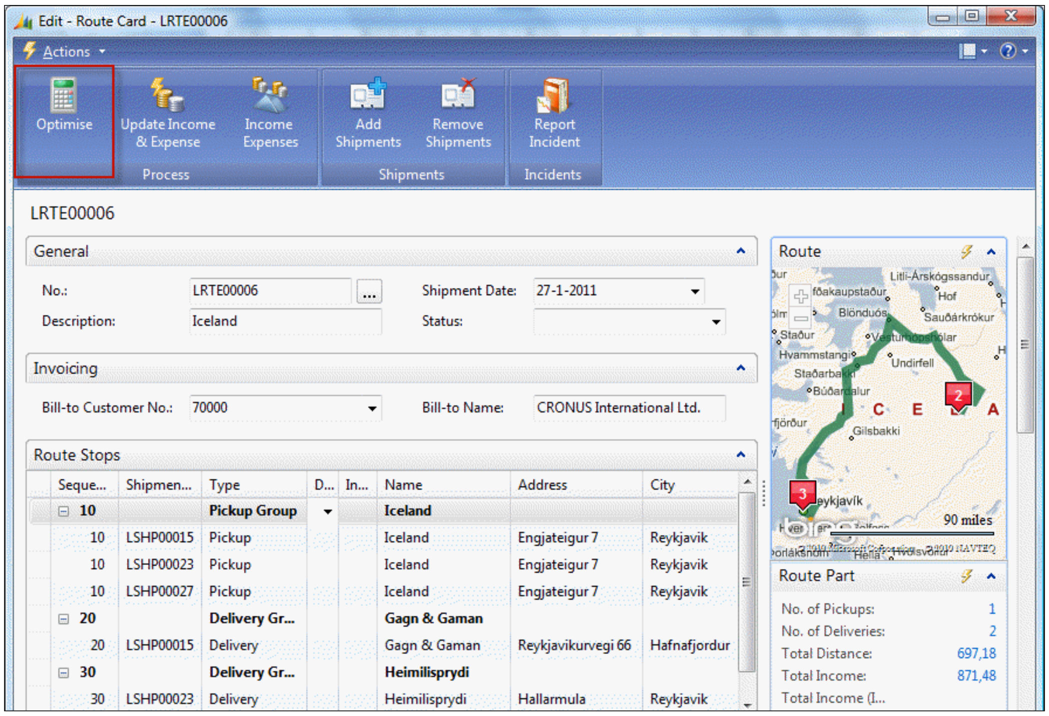
After generating the distances all Pickup shipments are combined into one stop by assigning them all to the same `Sequence No.` value.

```
RouteStopGroup.INIT;
RouteStopGroup."Route No." := Route."No.";
RouteStopGroup."Line No." := 10;
RouteStopGroup.Type := RouteStopGroup.Type::"Pickup Group";
RouteStopGroup."Sequence No." := 10;
RouteStopGroup.Name := RouteStopPickup.Name;
RouteStopGroup.INSERT;
RouteStopPickup.MODIFYALL("Sequence No.", 10);
```

By sorting the distance helper table on distance, we can easily assign the correct `Sequence No.` to the delivery stops. For each `Sequence No.` value, we will also generate a group record in the stop table.

```
Optimizer.SETCURRENTKEY("Distance (Distance)");
Optimizer.ASCENDING(FALSE);
Optimizer.FIND('-');
REPEAT
  RouteStopGroup.INIT;
  RouteStopGroup."Route No." := Route."No.";
  RouteStopGroup."Line No." := Sequence;
  RouteStopGroup.Type :=
    RouteStopGroup.Type::"Delivery Group";
  RouteStopGroup."Sequence No." := Sequence;
  RouteStopGroup.Name := Optimizer.Name;
  RouteStopGroup.INSERT;
  RouteStopDelivery.SETRANGE(Name, Optimizer.Name);
  RouteStopDelivery.MODIFYALL("Sequence No.", Sequence);

  Sequence := Sequence + 10;
  IF (xLongitude <> Optimizer.Longitude) OR
    (xLatitude <> Optimizer.Latitude)
  THEN BEGIN
    IF xLongitude + xLatitude <> 0 THEN BEGIN
      CLEAR(BingMapMgt);
      BingMapMgt.CalculateRoute('', xLatitude, xLongitude,'',
        Optimizer.Latitude, Optimizer.Longitude,
        RouteStopGroup.Distance, RouteStopGroup.Time,
        Optimize::Distance);
      RouteStopGroup.MODIFY;
    END;
    xLongitude := Optimizer.Longitude;
    xLatitude := Optimizer.Latitude;
  END;
UNTIL Optimizer.NEXT = 0;
```

After optimizing the route it should look something like this. We pick up two shipments at the warehouse and drive them to two addresses in the country.



LRTE00006

General

No.:	LRTE00006	Shipment Date:	27-1-2011
Description:	Iceland	Status:	

Invoicing

Bill-to Customer No.:	70000	Bill-to Name:	CRONUS International Ltd.

Route Stops

Seque...	Shipmen...	Type	D...	In...	Name	Address	City
10		Pickup Group			Iceland		
10	LSHP00015	Pickup			Iceland	Engjateigur 7	Reykjavik
10	LSHP00023	Pickup			Iceland	Engjateigur 7	Reykjavik
10	LSHP00027	Pickup			Iceland	Engjateigur 7	Reykjavik
20		Delivery Gr...			Gagn & Gaman		
20	LSHP00015	Delivery			Gagn & Gaman	Reykjavikurvegi 66	Hafnafjordur
30		Delivery Gr...			Heimilisprydi		
30	LSHP00023	Delivery			Heimilisprydi	Hallarmula	Reykjavik

Route Part

No. of Pickups:	1
No. of Deliveries:	2
Total Distance:	697,18
Total Income:	871,48
Total Income (I...	

Route follow up

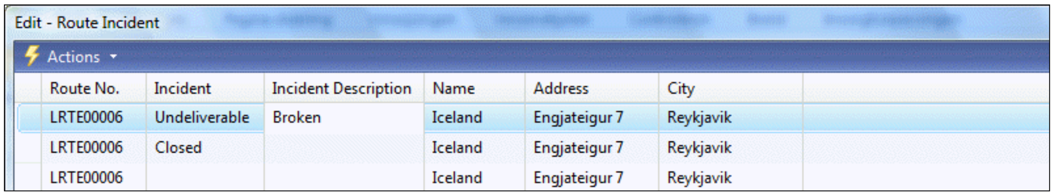
During the route, the planner needs to follow up with the driver. This will result in the status of the shipment being updated.

In our solution, the planner should populate the field Date Time Completed. This field is automatically updated in the shipment using a flow field.

Incidents

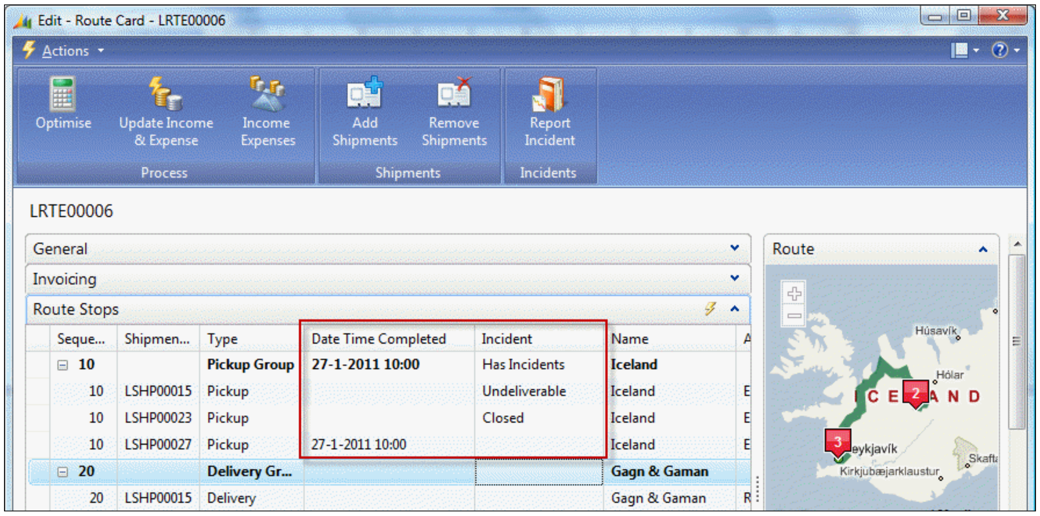
A special status for a Shipment is Incident. If for any reason we cannot deliver the shipment it should be taken back to the warehouse and shipped again. Based on the reason for the incident we might need to invoice extra services.

The incident can be on a stop group or on an individual shipment and can have status Undeliverable, Closed, or Other. The planner can add extra comments.



Route No.	Incident	Incident Description	Name	Address	City
LRTE00006	Undeliverable	Broken	Iceland	Engjateigur 7	Reykjavik
LRTE00006	Closed		Iceland	Engjateigur 7	Reykjavik
LRTE00006			Iceland	Engjateigur 7	Reykjavik

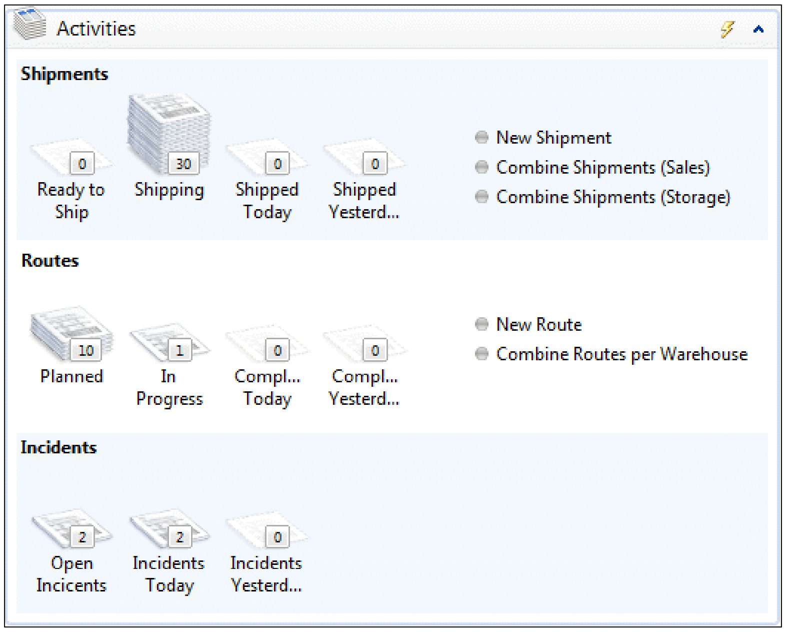
The other shipments that do not have incidents get the new status whilst the incidents move to another place in the Role Center.



Seque...	Shipmen...	Type	Date Time Completed	Incident	Name	A
⊟ **10**		**Pickup Group**	**27-1-2011 10:00**	Has Incidents	**Iceland**	
10	LSHP00015	Pickup		Undeliverable	Iceland	E
10	LSHP00023	Pickup		Closed	Iceland	E
10	LSHP00027	Pickup	27-1-2011 10:00		Iceland	E
⊟ **20**		**Delivery Gr...**			**Gagn & Gaman**	
20	LSHP00015	Delivery			Gagn & Gaman	R

Follow up

The incidents can be followed up by the planner via the Role Center. Incidents that are not handled keep the status open until someone decides what to do with it.



The invoicing application

In *Chapter 2*, *A Sample Application*, we introduced invoicing for an add-on solution. For the solution in this chapter, we'll take this one step further.

Our company is invoicing different logistics services such as:

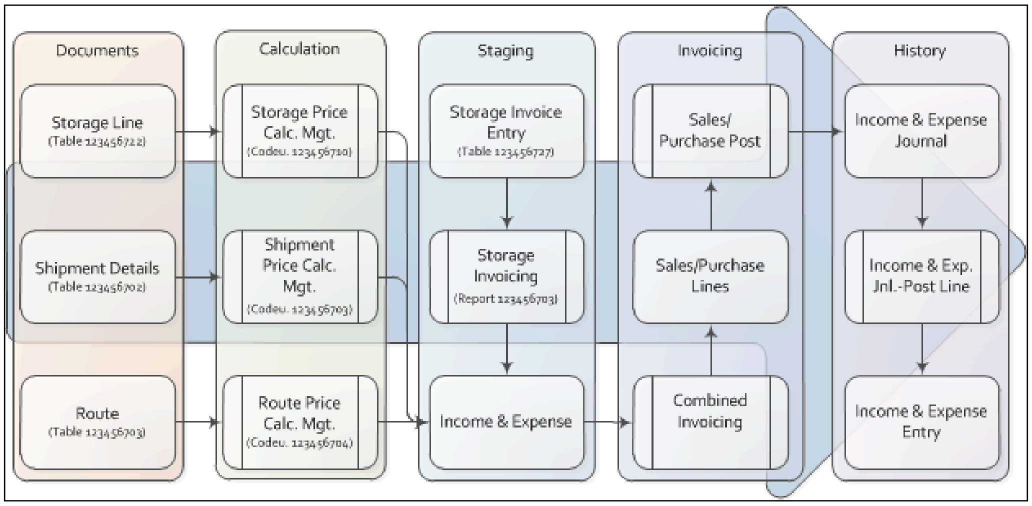
- Handling costs for storage receipt and shipments.
- Storage costs for the period we keep the inventory.
- Costs for transporting the products to the end consumer.

All these costs need to be combined on one invoice. Some customers may require monthly invoicing, some weekly, and for incidental customers we invoice directly. This requires a special module to handle the invoicing.

For the examples in this part of the chapter the Default Role Center in the Profile table (2000000072) should be changed to Income & Expenses Role Center (123456761).

Process

Let's have a look at the process to see where invoicing is required.

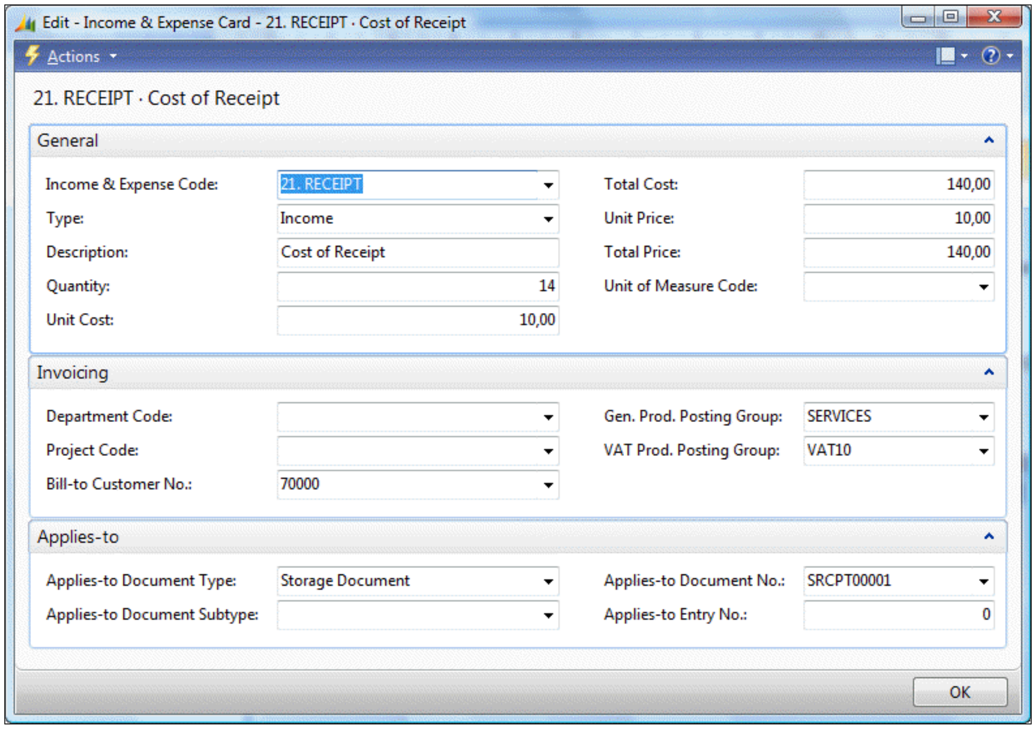

Documents
Calculation
Staging
Invoicing
History
Storage Line (Table 123456722)
Storage Price Calc. Mgt. (Codeu. 123456710)
Storage Invoice Entry (Table 123456727)
Sales/ Purchase Post
Income & Expense Journal
Shipment Details (Table 123456702)
Shipment Price Calc. Mgt. (Codeu. 123456703)
Storage Invoicing (Report 123456703)
Sales/Purchase Lines
Income & Exp. Jnl.-Post Line
Route (Table 123456703)
Route Price Calc. Mgt. (Codeu. 123456704)
Income & Expense
Combined Invoicing
Income & Expense Entry

Income and expense

Everything that we want to invoice at one time to a customer, we store in a new table we will call **Income & Expense**. This is a container where they will be kept until the periodical invoicing is done for this customer.

Income & Expense records can be created manually by end users or automatically by the system.

Let's have a look at this.


Edit - Income & Expense Card - 21. RECEIPT · Cost of Receipt
Actions
OK

21. RECEIPT · Cost of Receipt

General

Income & Expense Code:	21. RECEIPT	Total Cost:	140,00
Type:	Income	Unit Price:	10,00
Description:	Cost of Receipt	Total Price:	140,00
Quantity:	14	Unit of Measure Code:	
Unit Cost:	10,00		

Invoicing

Department Code:		Gen. Prod. Posting Group:	SERVICES
Project Code:		VAT Prod. Posting Group:	VAT10
Bill-to Customer No.:	70000		

Applies-to

Applies-to Document Type:	Storage Document	Applies-to Document No.:	SRCPT00001
Applies-to Document Subtype:		Applies-to Entry No.:	0

To create a new Income & Expense record we need to fill in these fields.

- **Income & Expense Code**: This is a reference to the group of Income & Expense.
- **Type**: This can be either Income or Expense. The former will be used on Sales Invoices; the latter is reserved for future use on Purchase Invoices if we decide to hire other companies to handle our logistics.
- **Description**: This is the description that will be printed on the Sales Invoice.
- **Quantity**: The number of services we have done. For example, the number of storage days or number of kilometers or miles in a route.
- **Unit Cost/Total Cost**: This can be used to calculate the profit of a service.
- **Unit Price/Total Price**: This is the price the customer will see on the Sales Invoice.
- **Unit of Measure Code**: A reference to the calculation method like `BOX`, `KM`, `MILES`, or `DAY`.
- **Applies-to Document (Sub)Type**: A reference to Storage Header, Registered Storage Header, Logistics Shipment, or Logistics Route. If necessary this can be expanded to accommodate other add-ons.

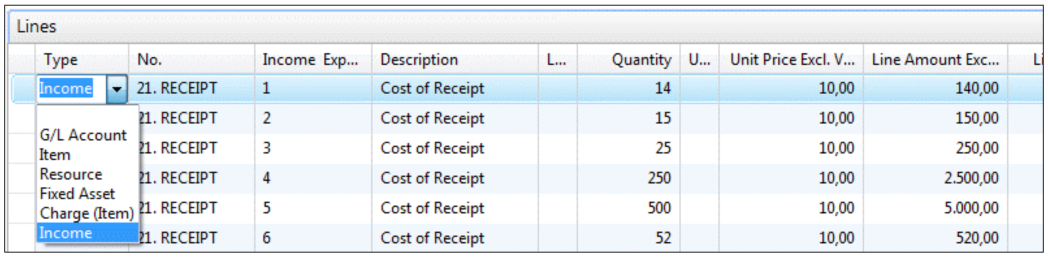
- **Applies-to Document (Line)No.**: A reference to the Storage and Logistics documents that this Income & Expense record belongs to.
- **Applies-to Entry No.**: A reference to the Storage Invoice Entry

Invoicing

After the Income & Expenses are created, we can start the invoicing process. To support this, some minor changes are done in the invoicing part of Microsoft Dynamics NAV and as an example we choose a slightly different approach compared to Chapter 2.

Sales Line

The Sales Line table (37) has gotten some minor modifications. We have added an extra type for **Income** and implemented a table relation for the No. field.



Type	No.	Income Exp...	Description	L...	Quantity	U...	Unit Price Excl. V...	Line Amount Exc...
Income	21. RECEIPT	1	Cost of Receipt		14		10,00	140,00
	21. RECEIPT	2	Cost of Receipt		15		10,00	150,00
	21. RECEIPT	3	Cost of Receipt		25		10,00	250,00
	21. RECEIPT	4	Cost of Receipt		250		10,00	2.500,00
	21. RECEIPT	5	Cost of Receipt		500		10,00	5.000,00
	21. RECEIPT	6	Cost of Receipt		52		10,00	520,00

This enables us to also create new entries on a sales invoice without having to create an Income & Expense first.

The Sales Line also has a reference to the Income & Expense Entry No. and the Apply-to fields. This enables us to create the Income & Expense Journal Lines in the Sales Post Code Unit.

Codeunit Sales-Post (80)

The Sales Post Code Unit has only one change to populate the Income & Expense Journal.

```
OnRun()
...
//* Chapter 7 >>>
SalesLine.Type::Income:
  IF SalesLine."Qty. to Invoice" <> 0 THEN BEGIN
    IncExpJnlLn.INIT;
    IncExpJnlLn."Posting Date" := "Posting Date";
```

```
        IncExpJnlLn."Document Date" := "Document Date";
        IncExpJnlLn."Reason Code" := "Reason Code";
        IncExpJnlLn."Income & Expense Type" :=
          IncExpJnlLn."Income & Expense Type"::Income;
        IncExpJnlLn."Income & Expense No." := SalesLine."No.";
        IncExpJnlLn.Description := SalesLine.Description;
        IncExpJnlLn."Bill-to Customer No." :=
          SalesLine."Bill-to Customer No.";
        IncExpJnlLn."Applies-to Document Type" :=
          SalesLine."Applies-to Document Type";
        IncExpJnlLn."Applies-to Document Subtype" :=
          SalesLine."Applies-to Document Subtype";
        IncExpJnlLn."Applies-to Document No." :=
          SalesLine."Applies-to Document No.";
        IncExpJnlLn."Applies-to Document Line No." :=
          SalesLine."Applies-to Document Line No.";
        IncExpJnlLn."Income & Expense Entry No." :=
          SalesLine."Income & Expense Entry No.";
        IncExpJnlLn."Shortcut Dimension 1 Code" :=
          SalesLine."Shortcut Dimension 1 Code";
        IncExpJnlLn."Shortcut Dimension 2 Code" :=
          SalesLine."Shortcut Dimension 2 Code";
        IncExpJnlLn."Gen. Bus. Posting Group" :=
          SalesLine."Gen. Bus. Posting Group";
        IncExpJnlLn."Gen. Prod. Posting Group" :=
          SalesLine."Gen. Prod. Posting Group";
```

This is done in the same way as the Resource Journal.

As the Sales Line has all the Posting Group and Amount fields populated the General Ledger Entries, VAT Entries and Customer Ledger Entries are automatically generated by the standard application.

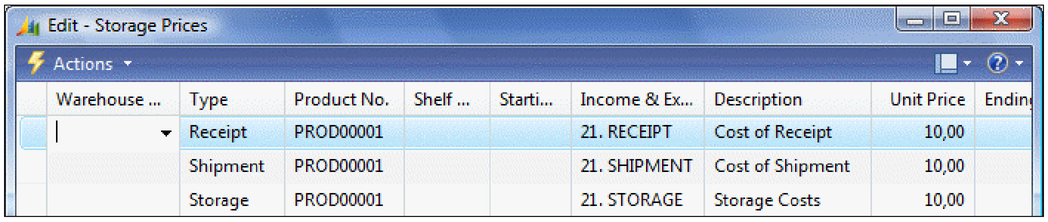
Pricing methodology

Our add-on solution has three levels of automatic price calculation that are more or less identical. We can calculate prices for Storage Documents, Logistics Shipments, and Routes.

Let's look at the Storage Prices as an example of how this is done.

Storage prices

In the Storage Price table, we can register prices for different storage activities.



Warehouse ...	Type	Product No.	Shelf ...	Starti...	Income & Ex...	Description	Unit Price	Endin
	Receipt	PROD00001			21. RECEIPT	Cost of Receipt	10,00	
	Shipment	PROD00001			21. SHIPMENT	Cost of Shipment	10,00	
	Storage	PROD00001			21. STORAGE	Storage Costs	10,00	

When the price is calculated, the system will filter down in this table to find the price that matches best. For example, if a Product has a price for Receipt without a Warehouse Code, this price is used in all warehouses, but if one Warehouse Code is populated, this Warehouse has a special price.

Prices can be differentiated to Receipt, Shipment, Pick, Put-Away, Movement and Storage. The first options are used on the Storage documents, the latter when calculating Storage Cost.

The Income & Expense Code determines which type of Income & Expense will be created for this combination. A Storage Document can have more than one Income & Expense; for example, a normal receipt line and a customs surplus.

Calculation

The Income & Expenses are created using a Price Calc. Mgt. Codeunit we are familiar with from Chapter 2, only this time we will not update the Unit Price but create the Income & Expenses.

The calculation for Storage is done in Codeunit 123456710.

```
FindStorageLinePrice

WITH StorageLine DO BEGIN
  Product.GET("No.");
  StorageLinePriceExists(StorageHeader, StorageLine);
  CreateIncExp(StorageHeader,StorageLine,TempStoragePrice);

END;
```

The `FindStorageLinePrice` function will call the standard function `StorageLinePriceExists` to find the Storage Prices that match the criteria.

For all the Storage Prices in the filter, it calls the function `CreateIncExp`.

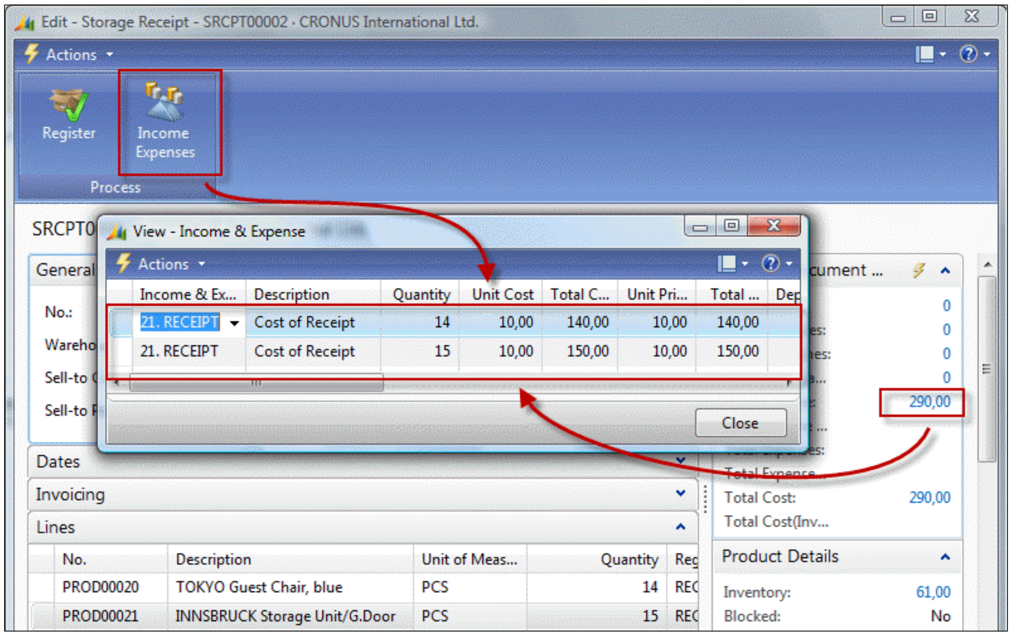
```
CreateIncExp()

IncExp.SETRANGE("Applies-to Document Type", IncExp."Applies-to
Document Type"::"Storage Header");
IncExp.SETRANGE("Applies-to Document No.", StorageHeader."No.");
IncExp.SETRANGE("Applies-to Document Line No.", StorageLine."Line
No.");
IncExp.DELETEALL;

WITH StoragePrice DO BEGIN
  FoundStoragePrice := FINDSET;
  IF FoundStoragePrice THEN BEGIN
    REPEAT
      IncExpCode.GET(StoragePrice."Income & Expense Code");
      IncExp.INIT;
      IncExp."Entry No." := 0;           //* For Autoincrement
      IncExp.Type := IncExpCode.Type;
      IncExp."Income & Expense Code" :=
        "Income & Expense Code";
      IncExp.Description := Description;
      IncExp.Quantity := StorageLine.Quantity;
      IncExp."Unit Cost" := IncExpCode."Unit Cost";
      IncExp."Total Cost" := IncExp.Quantity *
        IncExp."Unit Cost";
      IncExp."Unit Price" := StoragePrice."Unit Price";
      IncExp."Total Price" := IncExp.Quantity *
        IncExp."Unit Price";
      IncExp."Applies-to Document Type" :=
        IncExp."Applies-to Document Type"::"Storage Header";
      IncExp."Applies-to Document No." := StorageHeader."No.";
      IncExp."Applies-to Document Line No." :=
        StorageLine."Line No.";
      IncExp."Bill-to Customer No." :=
        StorageHeader."Bill-to Customer No.";
      IncExp."Gen. Prod. Posting Group" :=
        IncExpCode."Gen. Prod. Posting Group";
      IncExp."VAT Prod. Posting Group" :=
        IncExpCode."VAT Prod. Posting Group";
      IncExp.INSERT;
    UNTIL NEXT = 0;
  END;
END;
```

Each price will create a separate Income & Expense record.

[The Income & Expense table is set to **Auto Increment**. This means that the SQL Server will generate the entry number for us. This enables multiple users to generate entries in this table at the same time without blocking each other.]

Result

When new documents are generated by the system or end users, the prices are automatically calculated. The user can see the total cost and price on the Fact Box and change, remove, or add records if necessary.



Income & Ex...	Description	Quantity	Unit Cost	Total C...	Unit Pri...	Total ...	Dep
21. RECEIPT	Cost of Receipt	14	10,00	140,00	10,00	140,00	
21. RECEIPT	Cost of Receipt	15	10,00	150,00	10,00	150,00	

No.	Description	Unit of Meas...	Quantity	Req
PROD00020	TOKYO Guest Chair, blue	PCS	14	REC
PROD00021	INNSBRUCK Storage Unit/G.Door	PCS	15	REC

	0
	0
	0
	0
	290,00
Total Cost:	290,00
Total Cost(Inv...	
Inventory:	61,00
Blocked:	No

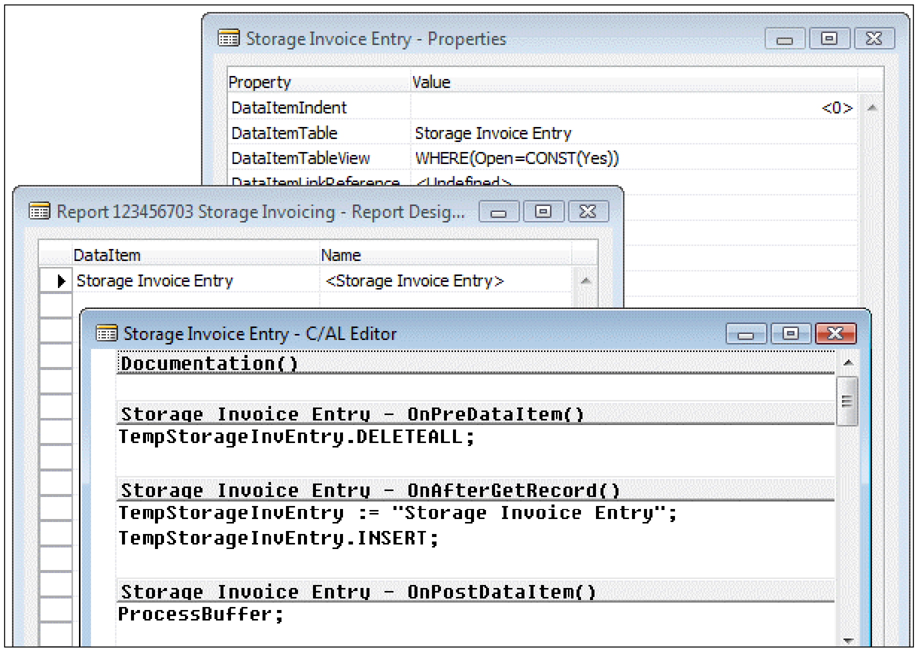
Periodic invoicing

One of the services we are providing is storage. This means that sometimes products can be in our warehouse for several days, weeks, or even months. Our customers will be invoiced for the time they use our warehouse space.

Each time we receive a product in our warehouse or move a product to another region or shelf, a Storage Entry is created to keep track. For invoicing, we also create a Storage Invoice Entry. This is mainly because the inventory handling and invoicing are done on different moments by different persons. The products can be shipped to the customer when we start the invoicing process.

The **Storage Invoice Entry** is created with a **From Storage Date** that is inherited from the **Storage Date** of the **Storage Entry**. The **Storage Invoice Entry** also has a **To Storage Date** that maintains blank until the product leaves the warehouse or moves to another location that might have another price. The **Income & Expense Code** determines which price will be invoiced and is determined when posting a **Storage Document**.

The batch report **Storage Invoicing (123456703)** is used for the creation of the Income & Expenses. Let's have a look at how this is done.


Storage Invoice Entry - Properties
Report 123456703 Storage Invoicing - Report Desig...
Storage Invoice Entry - C/AL Editor

Property	Value
DataItemIndent	<0>
DataItemTable	Storage Invoice Entry
DataItemTableView	WHERE(Open=CONST(Yes))
DataItemLinkReference	<Undefined>

DataItem	Name
Storage Invoice Entry	<Storage Invoice Entry>

```
Documentation()

Storage Invoice Entry - OnPreDataItem()
TempStorageInvEntry.DELETEALL;

Storage Invoice Entry - OnAfterGetRecord()
TempStorageInvEntry := "Storage Invoice Entry";
TempStorageInvEntry.INSERT;

Storage Invoice Entry - OnPostDataItem()
ProcessBuffer;
```

The report only has one Storage Invoice Entry DataItem which is filtered on Open=Yes.

In the report, all the Storage Invoice Entries are moved to a buffer table first and handled later.

There are two important reasons for implementing a solution like this:

- Changing Record Set: This report filters on Storage Invoice Entries which are open for invoicing. When the Storage Invoice Entry is completely invoiced, we want to change this value. This means that the record set we use is changing during the process. This is something the SQL Server backend cannot handle and this will result in a very poor performance.
- By first moving all records to a buffer table the filtering will be done on a virtual table that is maintained on the Service Tier rather than SQL Server.
- Locking: If we were to filter on open entries and modify our dataset it would result in locking more records than necessary. Filtering on a non-clustered index will result in SQL Server moving to Range Locks rather than Row Locks.
- By reading the actual Storage Invoice Entry one by one using the clustered index, we will make sure that SQL Server only locks the records we use for this process allowing other users to keep creating new records at the end of this table.

Processing the buffer

When processing the buffer we first check if this entry has been invoiced before. If this is the case, we start invoicing from the previous date, if not we use the From Storage Date.

Then we check if the products have already left the warehouse or have been moved. If this is the case, we can close this entry by invoicing until this date, otherwise we will invoice until the Workdate.

> Users can change the systems Workdate and influence the system's behavior this way and invoice until another date.

```
ProcessBuffer()

StorageInvEntry.LOCKTABLE;

WITH TempStorageInvEntry DO
  IF FIND('-') THEN REPEAT
    StorageInvEntry.GET("Entry No.");

    IF "Last Invoice Date" <> 0D THEN
      FromDate := "Last Invoice Date"
    ELSE
      FromDate := "From Storage Date";

    IF "To Storage Date" <> 0D THEN
```

```
          StorageInvEntry."Last Invoice Date" := "To Storage Date"
        ELSE
          StorageInvEntry."Last Invoice Date" := WORKDATE;

        Date.SETRANGE("Period Type", Date."Period Type"::Datum);
        Date.SETRANGE("Period No.", 1, 5);
        Date.SETRANGE("Period Start", FromDate,
          StorageInvEntry."Last Invoice Date");

        IncExp."Entry No." := 0;
        IncExp."Income & Expense Code" := "Income & Expense Code";
        IncExp.Type := IncExp.Type::Income;
        IncExp.Description := STRSUBSTNO(Text000, FromDate,
          StorageInvEntry."Last Invoice Date");
        IncExp.Quantity := Date.COUNT;
        IncExp."Unit Cost" := "Unit Cost";
        IncExp."Total Cost" := IncExp.Quantity * "Total Cost";
        IncExp."Unit Price" := "Unit Price";
        IncExp."Total Price" := IncExp.Quantity * "Unit Price";
        IncExp."Global Dimension 1 Code" :=
          "Global Dimension 1 Code";
        IncExp."Global Dimension 2 Code" :=
          "Global Dimension 2 Code";
        IncExp."Bill-to Customer No." := "Bill-to Customer No.";
        IncExpCode.GET(IncExp."Income & Expense Code");
        IncExp."Gen. Prod. Posting Group" :=
          IncExpCode."Gen. Prod. Posting Group";
        IncExp."VAT Prod. Posting Group" :=
          IncExpCode."VAT Prod. Posting Group";
        IncExp."Unit of Measure Code" :=
           IncExpCode."Unit of Measure Code";
        IncExp."Applies-to Entry No." := "Entry No.";
        IncExp.INSERT;

        StorageInvEntry.Open := "To Storage Date" <> 0D;
        StorageInvEntry.MODIFY;
      UNTIL NEXT = 0;
```

The next step in our code is to calculate the number of workdays between the two dates. This will prevent our customer from paying for storage on Saturday and Sunday.

We do this by using the virtual Date table. This table contains all dates, weeks, months, quarters, and years between January 1st 0000 and December 31st 9999 and can be very useful in date calculations.

With this result, we can now create the Income & Expense records that will be invoiced later.

If the To Storage Date is populated, we close the Storage Invoice Entry.

Combined invoicing

The Data Model we use allows us to combine invoicing on all the services we provide for our customers. We can create one invoice that contains handling, storage, and transportation costs for our customers.

This is done by batch report 123456704 Combine Storage & Logistics which works exactly the same as the report in Chapter 2.

Add-on flexibility

The add-on we have created in this chapter is definitely not ready to be used by a real company, but it demonstrates how to create a flexible solution that can be expanded by others easily.

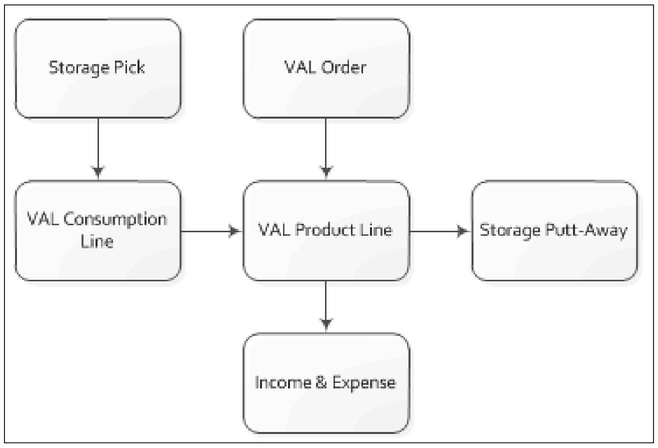
Most modern logistic service providers offer other services to customers like Value Added Logistics, Item Tracking, and third and fourth party logistics.

Value added logistics

When a company offers Value Added Logistics services, they not only keep products on inventory but they also offer services around this like display packaging.

This can be best compared with Manufacturing in Microsoft Dynamics NAV. A list of items called a Bill of Materials is combined into a new product. This new product is then shipped to the customer.

When the displays are no longer necessary, for example, when a marketing campaign is finished, the displays need to be picked up from the customer and disassembled into the original products.

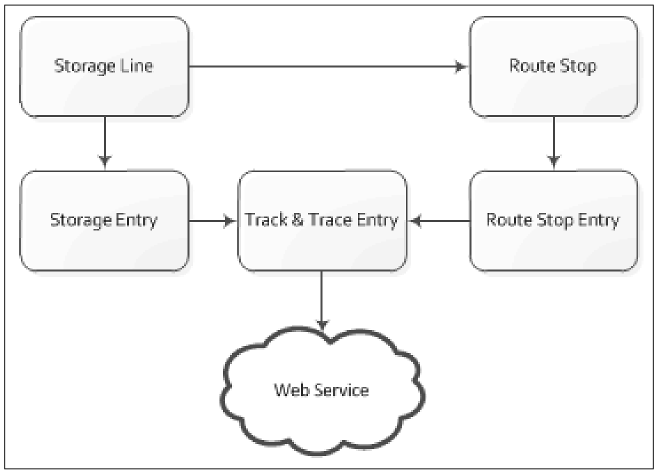


In our solution, this could be implemented by creating a VAL region where the products are moved to.

Item tracking

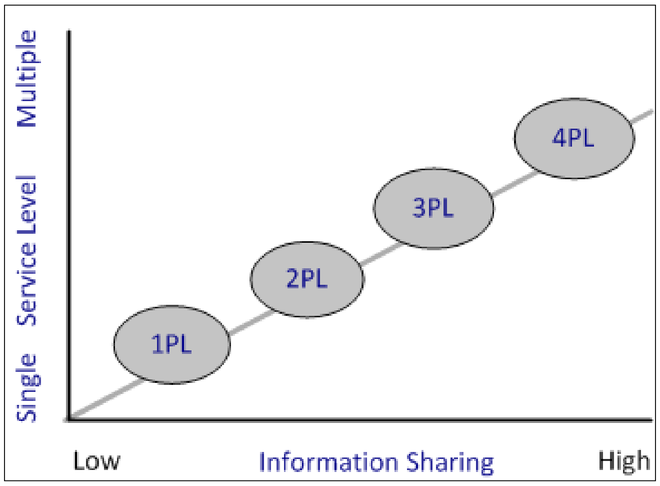
Our customers also want to know the whereabouts of their products—which warehouse are they in and which product was shipped to which customer. It is especially important in the food and medicine industry to be able to call back a lot if something goes wrong.

To implement this in our solution requires some changes. First, we need to implement a Tracking Code in the Storage Entries, second, we need to implement some kind of Tracking Entries when we ship a product outside our warehouse since currently our logistics solution does not have any entries, only status fields.



Third and fourth party logistics

In our example database, we plan shipments on routes and drive them to the end customer with our own trucks. This is called second party logistics. First party logistics would be if we were to handle our own products with our own trucks.



If we were to provide third party logistics, we would use other companies to offer parts of the services to our customers. We will then tell them which part of the service to handle and report back to us when it is finished. The third party involved does not know the details of the complete transaction.

If we were to offer fourth party logistics we would outsource a complete warehouse or route to another company. We would only tell them which product should be moved where and they would handle it without us knowing the details.

It is common for third and fourth party logistics to be mixed but almost always they are handled by interfaces between different companies.

Summary

In this chapter, we looked at the Microsoft Dynamics NAV product from a completely different viewpoint compared to the previous chapters.

The goal was not to design a rock solid Storage and Logistics add-on solution for Microsoft Dynamics NAV as this would require much more than one chapter. The information in this chapter is intended to demonstrate how to integrate new functionality *on top of* Microsoft Dynamics NAV.

We analyzed business processes and designed new data and transaction models to handle them in the product and implemented this.

For our solution, we designed two new document structures and two new journal and entry structures. We stayed close to the standard methodology of Microsoft Dynamics NAV by creating a framework that can easily be expanded.

We also spent some time looking at how to prevent unnecessary locking in the database and how to avoid changing a filtered dataset.

At the end, we looked at some examples of how our add-on solution could be enhanced to better suit other demands in the market.

This chapter does not end here. The C/AL objects provided with this chapter can be studied in order to understand even better how the pieces are put together.

In the next chapter, we will design an application *inside* Microsoft Dynamics NAV. We will look at how it can be used for a consultancy company using the Jobs module and how to extend this with new functionality to meet specific requirements.

8 Consulting

In this chapter we will learn how Microsoft Dynamics NAV fits a consultancy company. Most consultancy companies have project related processes. They take on larger projects that take a certain time to complete. Some consultancy companies also purchase and sell items.

For each of the projects the consultancy company needs to keep track of used resources and items. Sometimes they can invoice the resource hours they spend one-on-one but most of the time they will also take a risk in doing fixed price projects. In this case it is even more important to know if the project was budgeted well and ensure money is not lost on the way.

There are many types of companies working this way. Some examples include accountancy firms and lawyers, but also many companies in the construction business work like this.

For this chapter we will use an example company we are all very familiar with, either through being an employee, a customer or maybe even an owner. We will look at the business process of a Microsoft Dynamics NAV Partner.

The Partner in our case sells Microsoft Dynamics NAV licenses for new projects. They also help existing customers in upgrades and support. Lastly, they are sell infrastructure solutions; assemble servers and desktop systems in house.

We will discuss four different project scenarios and see how Microsoft Dynamics NAV can be used to support those. To do this we will create some modifications along the way.

With this chapter objects are required. The Appendix describes how to import and activate them.

After reading this chapter you will have a good understanding of the possibilities and limitations of the Job Module in Microsoft Dynamics NAV, how it fits in with the rest of the product and how it can be expanded safely.

The process

The two main processes for Microsoft Dynamics NAV partners are implementing new projects and providing services such as support and upgrades to existing customers. A third process is selling infrastructure and assembling computer systems but this is an extra service, not the core business.

To support the projects (jobs) the company needs people, software licenses, and hardware. The people (resources) need to be carefully planned on the projects as they are the least flexible part of the company. Hardware (items) and software licenses (G/L accounts) will be purchased from vendors such as Microsoft.

The projects can be divided into large and small projects. The larger projects are new implementations and upgrades. Smaller projects include implementing small features and helping users with regular support issues.

Invoicing can be done in various ways. New implementations and small projects can be invoiced per billable hour while upgrades are sold fixed price. For hardware we will use items. Licenses are invoiced directly to the General Ledger.

Large projects also have budgets and a plan that need to be maintained. If the budget is fully used and the planning milestones have not been reached there should be a new budget created in order to complete the project.

To support this process we will use the Jobs functionality with some customizations. Projects are called Jobs in Microsoft Dynamics NAV so we will use that term from now on.

The Jobs module has been completely redesigned by Microsoft for version 5. In this chapter we will use a lot of the new functionality where we would have done customizations in the older versions of Microsoft Dynamics NAV.

Fits

The registration of the Jobs can be done using the standard functionality of Microsoft Dynamics NAV as well as the budgeting and planning.

The standard software also allows us to invoice Jobs both fixed price and on time and materials. We can also purchase items for our Jobs.

Gaps

The Jobs module in Microsoft Dynamics NAV is often referenced as a framework that almost always needs some changes. Fortunately, it is designed to be easily changed and we will do so to support our processes.

Resource groups

Although many companies work this way; budgeting on Resource Groups is not possible. We will create a solution for that. We will also make it possible to see the total number of planned, used, and invoiced hours.

Time registration

The standard software allows us to register hours but it does not have a real-time sheet application. We will create one.

Item calculation

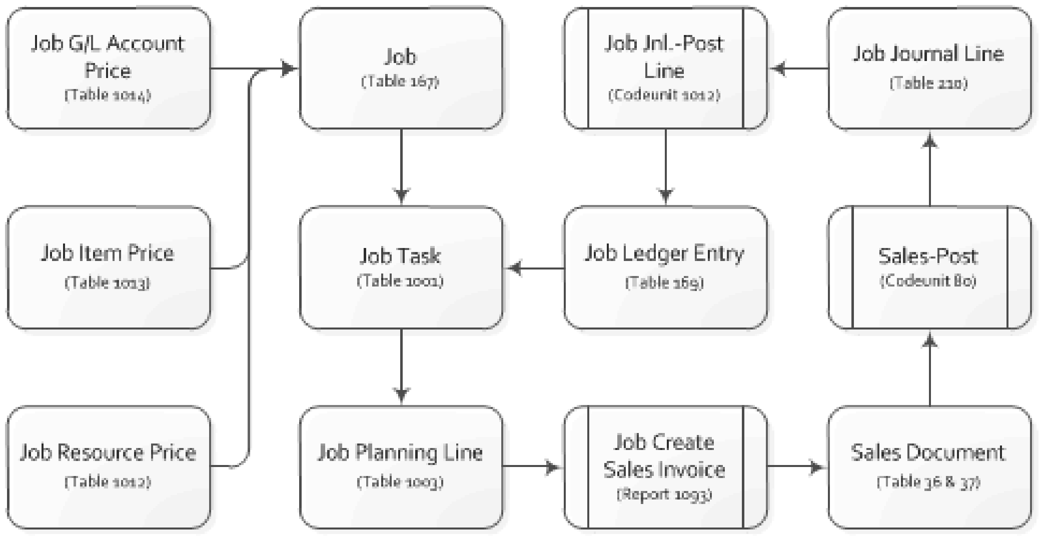
We will create a solution calculating the system assembling. As hardware specifications are changing rapidly we do not want to create a new item for each system when we may only sell that particular configuration once or twice.

Issue registration

Our support team needs a single point for registration of all support issues for all customers and follow up their workflow. For this we will also create the functionality to register and follow up issues.

Getting started

Before we start creating any new jobs, we should have a look at the data and posting model of the Microsoft Dynamics NAV Jobs module.


Job G/L Account Price (Table 1014)
Job (Table 167)
Job Jnl.-Post Line (Codeunit 1012)
Job Journal Line (Table 210)
Job Item Price (Table 1013)
Job Task (Table 1001)
Job Ledger Entry (Table 169)
Sales-Post (Codeunit 80)
Job Resource Price (Table 1012)
Job Planning Line (Table 1003)
Job Create Sales Invoice (Report 1093)
Sales Document (Table 36 & 37)

The starting point is the Job table, which has Job Tasks and Job Planning Lines we can use for budgeting and planning. Each job can have its own prices.

The Job Planning Lines get invoiced through the standard Microsoft Dynamics NAV Sales functionality which then creates Job Ledger Entries.

How many jobs

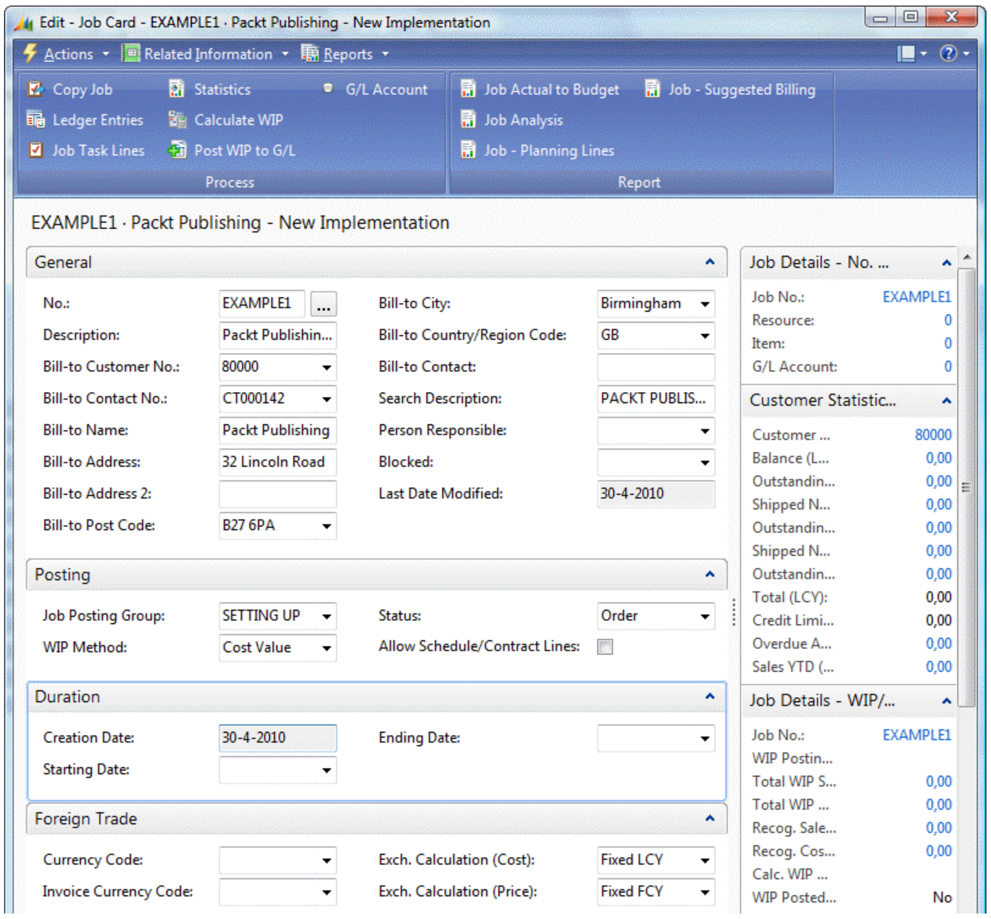
The first step is setting up a new job. There can be different angles on setting up jobs. This depends on how we want to work with the system. The minimum requirement is to have at least one job per Bill-to Customer. This enables us to do the invoicing. Some companies use jobs this way to use it as a pre-invoice engine.

Another angle can be to set up new jobs nicely for each project we do for the customer. In our case this starts with the basic Dynamics NAV implementation. When this is finished we close the job. If the customer has any new requirements we will need to start a new job. This way we can keep better track of what issues we have outstanding with each customer. The downside of this methodology is that it requires some work to set up a new job every time.

Most companies end up with a solution in the middle. It is common to set up a new job for larger jobs and to have a job for support issues. This also allows us to set up different invoicing strategies for each job. We will use this strategy.

Job card

Let's have a look at the **Job Card** and the important fields there.



EXAMPLE1 · Packt Publishing - New Implementation

General

No.:	EXAMPLE1	Bill-to City:	Birmingham
Description:	Packt Publishin...	Bill-to Country/Region Code:	GB
Bill-to Customer No.:	80000	Bill-to Contact:	
Bill-to Contact No.:	CT000142	Search Description:	PACKT PUBLIS...
Bill-to Name:	Packt Publishing	Person Responsible:	
Bill-to Address:	32 Lincoln Road	Blocked:	
Bill-to Address 2:		Last Date Modified:	30-4-2010
Bill-to Post Code:	B27 6PA		

Posting

Job Posting Group:	SETTING UP	Status:	Order
WIP Method:	Cost Value	Allow Schedule/Contract Lines:	☐

Duration

Creation Date:	30-4-2010	Ending Date:	
Starting Date:			

Foreign Trade

Currency Code:		Exch. Calculation (Cost):	Fixed LCY
Invoice Currency Code:		Exch. Calculation (Price):	Fixed FCY

Job Details - No. ...

Job No.:	EXAMPLE1
Resource:	0
Item:	0
G/L Account:	0

Customer Statistic...

Customer ...	80000
Balance (L...	0,00
Outstandin...	0,00
Shipped N...	0,00
Outstandin...	0,00
Shipped N...	0,00
Outstandin...	0,00
Total (LCY):	0,00
Credit Limi...	0,00
Overdue A...	0,00
Sales YTD (...	0,00

Job Details - WIP/...

Job No.:	EXAMPLE1
WIP Postin...	
Total WIP S...	0,00
Total WIP ...	0,00
Recog. Sale...	0,00
Recog. Cos...	0,00
Calc. WIP ...	
WIP Posted...	No

Let's see these fields in more detail:

- **No.**: This is the unique number of a job. We can use different number series strategies for this, from simple sequential numbering to linked number series for different job types or manual numbering.

- **Description**: This should be a logical description of the job for internal use. Most people will search on this field so make sure to have certain rules for naming. This will make searching for old jobs easier in the future.
- **Bill-to Customer No.**: Each job has one Bill-to Customer. If we want to invoice multiple customers for one job we will need to customize the application.
- **Search Description**: By default this will be populated with the value of the description field but can be changed to another value if required.
- **Person Responsible**: This is an informative field indicating who is responsible for this job.
- **Blocked**: If this field is checked, it is not possible to make new entries for this job. Use this for closed jobs.
- **Job Posting Group**: This refers to the G/L Accounts that are used for the Work In Progress postings (WIP). There can be different G/L Accounts for different types of jobs or WIP methods.
- **WIP Method**: Each job can have one Work in Progress method. We will discuss this briefly later in this chapter.
- **Status**: The jobs have a limited set of status fields. The only available status values are **Planning**, **Quote**, **Order**, and **Completed**.

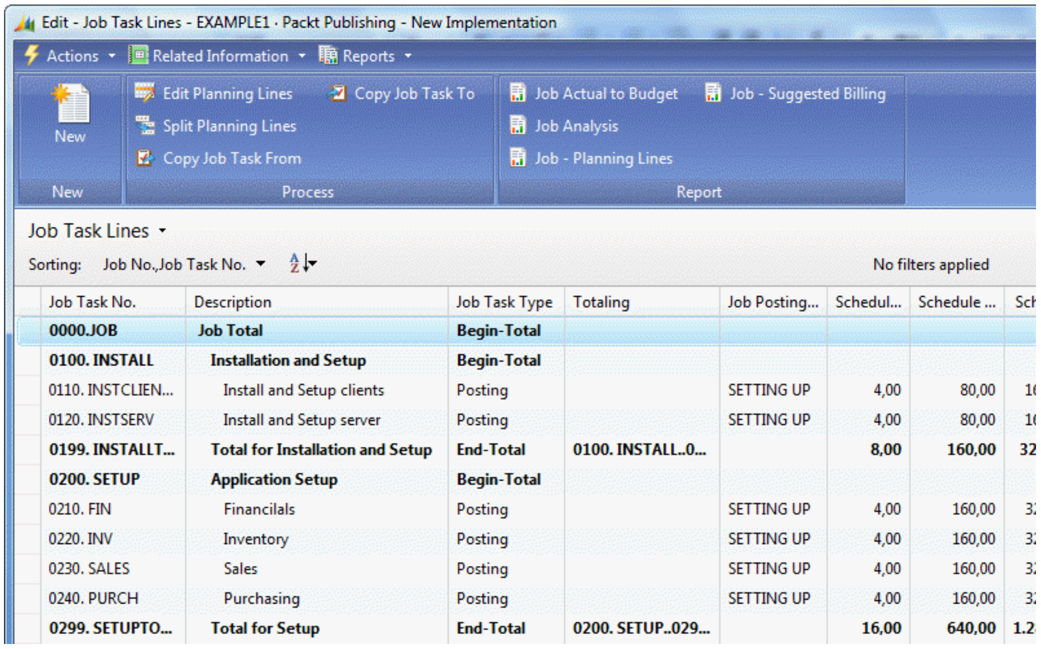
Most companies want to have more sub statuses for the order phase. The best approach for this is to add a new status field that maps with the standard status field. This requires minimum changes to the application while creating new workflow possibilities.

- **Allow Schedule/Contract Lines**: If this field is not checked it is not possible to create planning lines, which have both schedule and contract options. When planning lines are created they will be split into a schedule and a contract line.
- **Starting** and **Ending Date**: These are informative fields that are only used to calculate the currency exchange rates for the job.
- **Foreign Trade**: In the jobs module it is possible to send calculate and create invoices in a currency other than the local currency. This will increase the complexity of the implementation and should be used carefully.

Job task and planning lines

When the Job is created, the next step is to create Job Tasks and Planning Lines. These can be used in different ways.

Using job task lines we can cut the job into smaller pieces, which we can then schedule and invoice. The more detailed the job tasks are, the better we can measure the progress of the job, but the more work they require to maintain. Balance is the key to success here.



Job Task No.	Description	Job Task Type	Totaling	Job Posting...	Schedul...	Schedule ...	Sch
0000.JOB	**Job Total**	**Begin-Total**					
0100. INSTALL	**Installation and Setup**	**Begin-Total**					
0110. INSTCLIEN...	Install and Setup clients	Posting		SETTING UP	4,00	80,00	1
0120. INSTSERV	Install and Setup server	Posting		SETTING UP	4,00	80,00	1
0199. INSTALLT...	**Total for Installation and Setup**	**End-Total**	**0100. INSTALL..0...**		**8,00**	**160,00**	**32**
0200. SETUP	**Application Setup**	**Begin-Total**					
0210. FIN	Financilals	Posting		SETTING UP	4,00	160,00	3
0220. INV	Inventory	Posting		SETTING UP	4,00	160,00	3
0230. SALES	Sales	Posting		SETTING UP	4,00	160,00	3
0240. PURCH	Purchasing	Posting		SETTING UP	4,00	160,00	3
0299. SETUPTO...	**Total for Setup**	**End-Total**	**0200. SETUP..029...**		**16,00**	**640,00**	**1.2**

The Job Tasks can be created with the same structure as the Chart of Accounts, meaning the actual Task Lines can be grouped using Begin and End Total lines. Each level can be indented for better readability.

The Job Planning Lines are the detail lines of each job task. This defines what we will do and how this will be invoiced. A job planning line can be linked to the master data types Resource, Item, G/L Account, or Text.

[Job Tasks and Job Planning Lines can be copied very easily from other jobs. This allows us to reuse them and even create template jobs for frequently used combinations.]

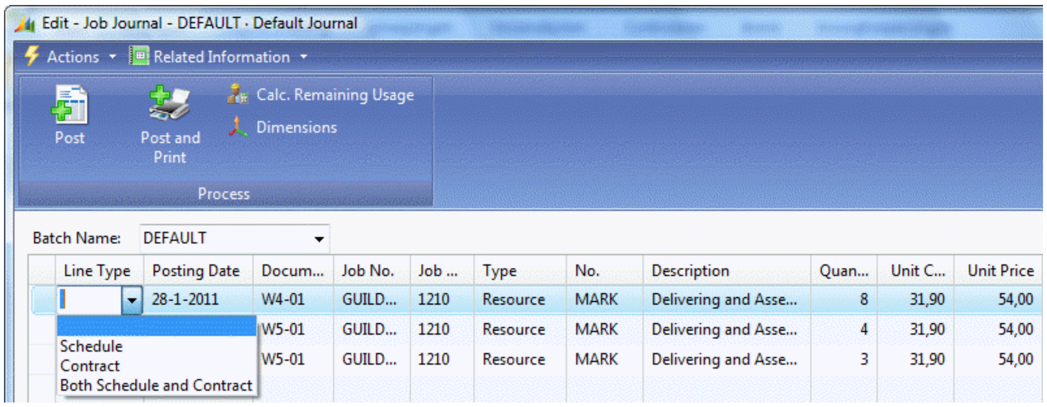
The line type in the job planning line defines how it will be invoiced. There are three types:

- **Schedule**: The amounts on this line will only be used in for budgeting purposes. When invoicing we need to post one or more job journal lines that will be invoiced or we can create another job planning line with the invoice amount. Schedule lines should be used when billing on time and materials.
- **Contract**: This line will be invoiced with the exact amounts. However the amounts do not show up in the budget. This can be used when invoicing fixed price jobs in a schedule, for example: 50% when signing the contract and 50% on job completion.
- **Both Schedule and Contract**: This line will be invoiced exactly the same way as the contract lines but the amount will also show up in the budget.

Job journal

When the job tasks and job planning lines are set up we can start the job. During the job we will consume resources and items from our company. This should be registered using the **Job Journal**. The Job Journal is the lowest level of the Journal Posting diagram we drafted in *Chapter 1, Introduction to Microsoft Dynamics NAV*, and uses the other journals to create the Resource, Item, and General Ledger Entries.

When creating a job journal line a few fields are particularly important for the process:



Batch Name: DEFAULT

Line Type	Posting Date	Docum...	Job No.	Job ...	Type	No.	Description	Quan...	Unit C...	Unit Price
	28-1-2011	W4-01	GUILD...	1210	Resource	MARK	Delivering and Asse...	8	31,90	54,00
		W5-01	GUILD...	1210	Resource	MARK	Delivering and Asse...	4	31,90	54,00
		W5-01	GUILD...	1210	Resource	MARK	Delivering and Asse...	3	31,90	54,00

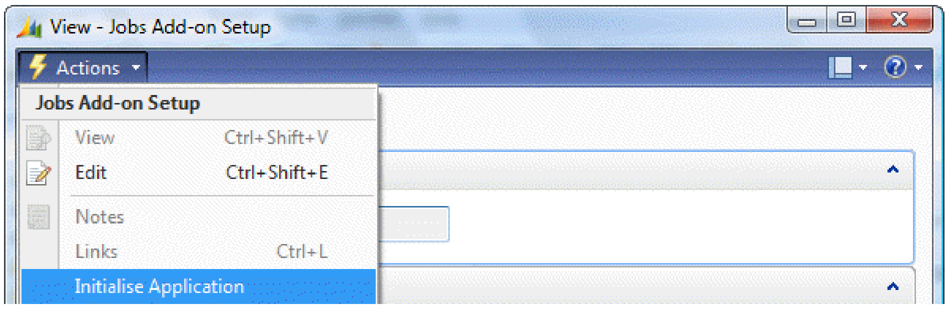
- **Line Type**: This has the same options as the job planning line, **Schedule**, **Contract**, and **Both Schedule and Contract**. When the job journal line should be invoiced, the type should be Contract. When the job journal line is part of a fixed price the line type should be left blank. When the line type is Schedule, the system will create additional Job Planning Lines of this type which may corrupt our budget for the customer as they are already created.
- **Unit Cost** and **Unit Price**: These fields will determine the cost of the job and the price that will be invoiced to the customer if the line type is Contract. This information is also used in the calculation of the Work in Progress.

Job examples

Let's go through some different job scenarios to see how we can use this functionality.

Chapter objects

The chapter objects contain both the changes we discuss in this chapter and as the example jobs we will use. After importing `chapter 8.job` as described in the *Appendix*, run page 123.456.700 **Jobs Add-on Setup** and then run **Initialise Application**.



When this completes, restart the Role Tailored Client. You should now see the Project Manager Role Center.

1 | The new implementation

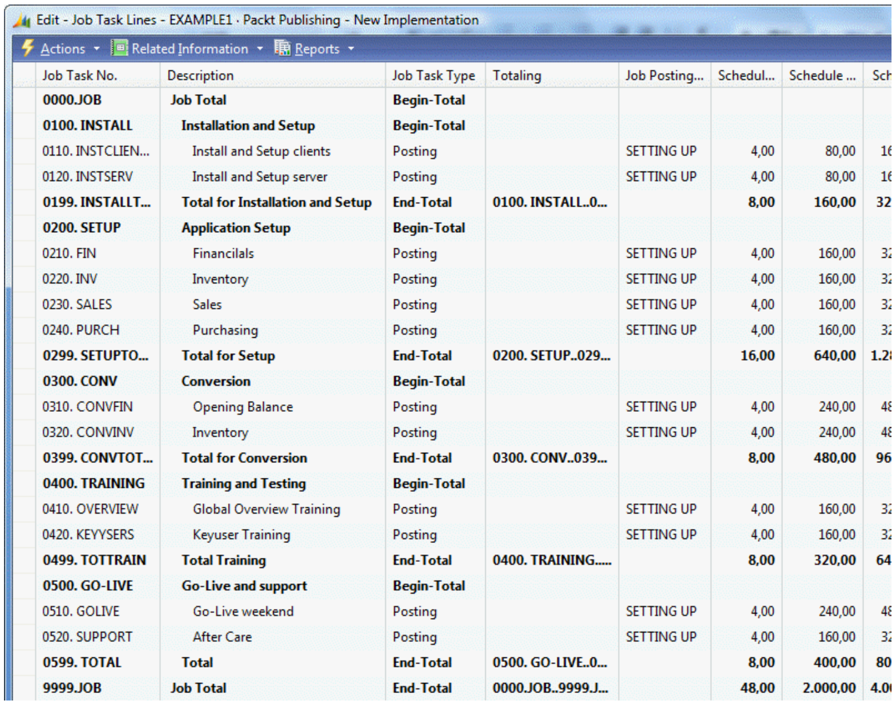
Implementing Microsoft Dynamics NAV 2009 is not an easy task and many things need to be taken care of before we can use the product. We will implement Microsoft Dynamics NAV for Packt Publishing. The job for this example is **EXAMPLE1**.

For the implementation we will create various job task groups. Each part of the implementation gets a code. Because the sorting is done on this field we will create codes using numbers and a logical name. For example, **0200. SETUP** and **0210. FIN**.

Leave enough space in the numbers to add additional lines if required. This will avoid renaming which is an expensive task for the database engine and users will have to wait until it is completed.

Our consultants will help the customer to install the system, help with the setup, and convert the data from the old system. When this is done we will help them with testing and train them for using Microsoft Dynamics NAV. The consultants will be set up in the system as Resources, which are in turn entered into the job planning lines.

When everything is working as expected we can schedule a go-live weekend and help them in the first period using the system.


Edit - Job Task Lines - EXAMPLE1 · Packt Publishing - New Implementation
Actions
Related Information
Reports

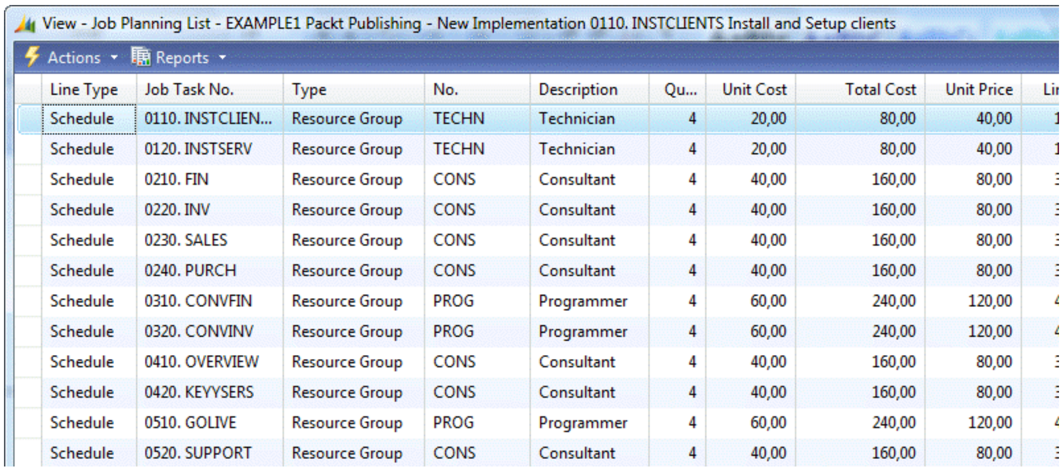
Job Task No.	Description	Job Task Type	Totaling	Job Posting...	Schedul...	Schedule ...	Sch
0000.JOB	**Job Total**	**Begin-Total**					
0100. INSTALL	**Installation and Setup**	**Begin-Total**					
0110. INSTCLIEN...	Install and Setup clients	Posting		SETTING UP	4,00	80,00	16
0120. INSTSERV	Install and Setup server	Posting		SETTING UP	4,00	80,00	16
0199. INSTALLT...	**Total for Installation and Setup**	**End-Total**	**0100. INSTALL..0...**		**8,00**	**160,00**	**32**
0200. SETUP	**Application Setup**	**Begin-Total**					
0210. FIN	Financilals	Posting		SETTING UP	4,00	160,00	32
0220. INV	Inventory	Posting		SETTING UP	4,00	160,00	32
0230. SALES	Sales	Posting		SETTING UP	4,00	160,00	32
0240. PURCH	Purchasing	Posting		SETTING UP	4,00	160,00	32
0299. SETUPTO...	**Total for Setup**	**End-Total**	**0200. SETUP..029...**		**16,00**	**640,00**	**1.2**
0300. CONV	**Conversion**	**Begin-Total**					
0310. CONVFIN	Opening Balance	Posting		SETTING UP	4,00	240,00	48
0320. CONVINV	Inventory	Posting		SETTING UP	4,00	240,00	48
0399. CONVTOT...	**Total for Conversion**	**End-Total**	**0300. CONV..039...**		**8,00**	**480,00**	**96**
0400. TRAINING	**Training and Testing**	**Begin-Total**					
0410. OVERVIEW	Global Overview Training	Posting		SETTING UP	4,00	160,00	32
0420. KEYYSERS	Keyuser Training	Posting		SETTING UP	4,00	160,00	32
0499. TOTTRAIN	**Total Training**	**End-Total**	**0400. TRAINING.....**		**8,00**	**320,00**	**64**
0500. GO-LIVE	**Go-Live and support**	**Begin-Total**					
0510. GOLIVE	Go-Live weekend	Posting		SETTING UP	4,00	240,00	48
0520. SUPPORT	After Care	Posting		SETTING UP	4,00	160,00	32
0599. TOTAL	**Total**	**End-Total**	**0500. GO-LIVE..0...**		**8,00**	**400,00**	**80**
9999.JOB	**Job Total**	**End-Total**	**0000.JOB..9999.J...**		**48,00**	**2.000,00**	**4.0**

Invoicing a job like this is done using a budget. We will make a precalculation of the number of hours we think are necessary and start with that. During the job we need to measure the used budget and compare it with the progress.

Budgeting

The budget is created using the Job Planning Lines. During this phase of the job we do not yet know which resource will be used for the job tasks and it might even be done by more than one resource. This is why we want to use Resource Groups in our budget.

> This is not possible in the standard application so we have created a modification which we will discuss at the end of this chapter.



View - Job Planning List - EXAMPLE1 Packt Publishing - New Implementation 0110. INSTCLIENTS Install and Setup clients

Line Type	Job Task No.	Type	No.	Description	Qu...	Unit Cost	Total Cost	Unit Price	Li
Schedule	0110. INSTCLIEN...	Resource Group	TECHN	Technician	4	20,00	80,00	40,00	1
Schedule	0120. INSTSERV	Resource Group	TECHN	Technician	4	20,00	80,00	40,00	1
Schedule	0210. FIN	Resource Group	CONS	Consultant	4	40,00	160,00	80,00	3
Schedule	0220. INV	Resource Group	CONS	Consultant	4	40,00	160,00	80,00	3
Schedule	0230. SALES	Resource Group	CONS	Consultant	4	40,00	160,00	80,00	3
Schedule	0240. PURCH	Resource Group	CONS	Consultant	4	40,00	160,00	80,00	3
Schedule	0310. CONVFIN	Resource Group	PROG	Programmer	4	60,00	240,00	120,00	4
Schedule	0320. CONVINV	Resource Group	PROG	Programmer	4	60,00	240,00	120,00	4
Schedule	0410. OVERVIEW	Resource Group	CONS	Consultant	4	40,00	160,00	80,00	3
Schedule	0420. KEYYSERS	Resource Group	CONS	Consultant	4	40,00	160,00	80,00	3
Schedule	0510. GOLIVE	Resource Group	PROG	Programmer	4	60,00	240,00	120,00	4
Schedule	0520. SUPPORT	Resource Group	CONS	Consultant	4	40,00	160,00	80,00	3

The Line Type of these job planning lines is **Schedule**. This means that these lines are just for budgeting and schedule purposes. The system will invoice the actual consumption posted in the Job Journal.

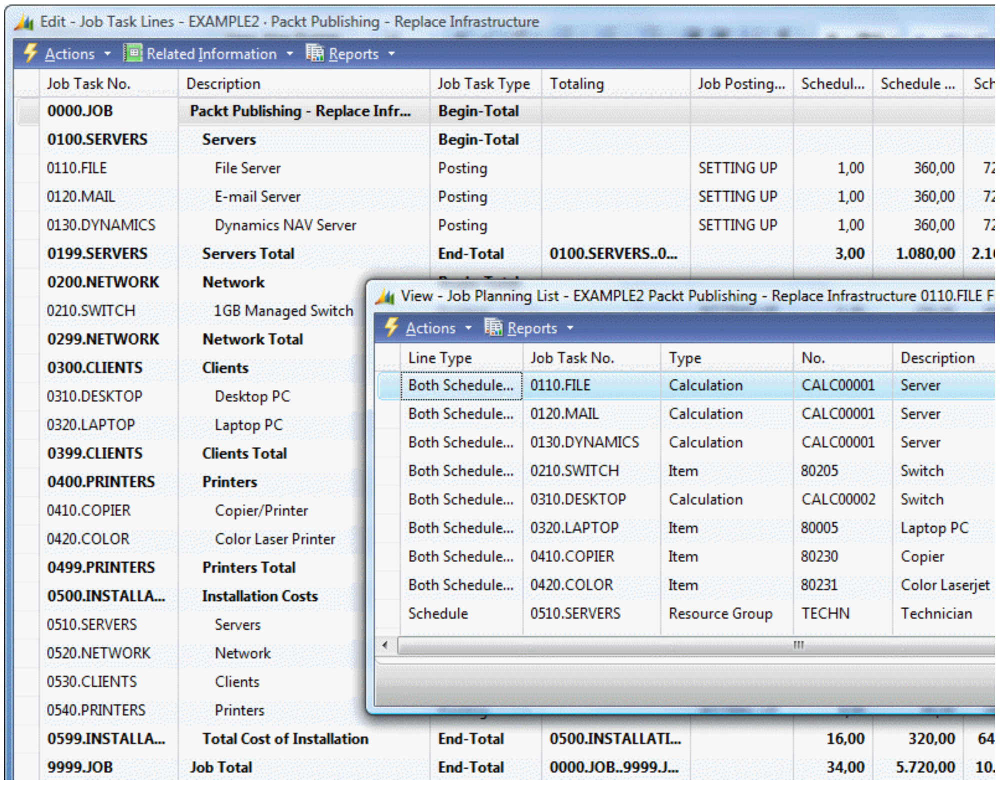
2 | The infrastructure

To use Microsoft Dynamics NAV 2009 Packt Publishing needs new infrastructure. Their current systems do not meet the requirements for Microsoft Dynamics NAV 2009.

For this job we could create new **Job Task Lines** in the implementation job, but for a clearer overview we will create a new job, **EXAMPLE2**.

Our company builds and sells its own computer systems. We can build both servers and desktop systems. Because none of the systems are exactly the same and available components switch regularly we do not want to create an item and a bill of materials for each system. Instead we use a **Calculation** system that allows us to determine a price for a system. For other products like switches, routers, printers, and laptops we use items which we purchase from vendors.

The Job Tasks and Job Planning Lines for this job look like this:


Edit - Job Task Lines - EXAMPLE2 · Packt Publishing - Replace Infrastructure
Actions
Related Information
Reports

Job Task No.	Description	Job Task Type	Totaling	Job Posting...	Schedul...	Schedule ...	Sch
0000.JOB	**Packt Publishing - Replace Infr...**	**Begin-Total**					
0100.SERVERS	**Servers**	**Begin-Total**					
0110.FILE	File Server	Posting		SETTING UP	1,00	360,00	72
0120.MAIL	E-mail Server	Posting		SETTING UP	1,00	360,00	72
0130.DYNAMICS	Dynamics NAV Server	Posting		SETTING UP	1,00	360,00	72
0199.SERVERS	**Servers Total**	**End-Total**	**0100.SERVERS..0...**		**3,00**	**1.080,00**	**2.1**
0200.NETWORK	**Network**						
0210.SWITCH	1GB Managed Switch						
0299.NETWORK	**Network Total**						
0300.CLIENTS	**Clients**						
0310.DESKTOP	Desktop PC						
0320.LAPTOP	Laptop PC						
0399.CLIENTS	**Clients Total**						
0400.PRINTERS	**Printers**						
0410.COPIER	Copier/Printer						
0420.COLOR	Color Laser Printer						
0499.PRINTERS	**Printers Total**						
0500.INSTALLA...	**Installation Costs**						
0510.SERVERS	Servers						
0520.NETWORK	Network						
0530.CLIENTS	Clients						
0540.PRINTERS	Printers						
0599.INSTALLA...	**Total Cost of Installation**	**End-Total**	**0500.INSTALLATI...**		**16,00**	**320,00**	**64**
9999.JOB	**Job Total**	**End-Total**	**0000.JOB..9999.J...**		**34,00**	**5.720,00**	**10.**

View - Job Planning List - EXAMPLE2 Packt Publishing - Replace Infrastructure 0110.FILE F
Actions
Reports

Line Type	Job Task No.	Type	No.	Description
Both Schedule...	0110.FILE	Calculation	CALC00001	Server
Both Schedule...	0120.MAIL	Calculation	CALC00001	Server
Both Schedule...	0130.DYNAMICS	Calculation	CALC00001	Server
Both Schedule...	0210.SWITCH	Item	80205	Switch
Both Schedule...	0310.DESKTOP	Calculation	CALC00002	Switch
Both Schedule...	0320.LAPTOP	Item	80005	Laptop PC
Both Schedule...	0410.COPIER	Item	80230	Copier
Both Schedule...	0420.COLOR	Item	80231	Color Laserjet
Schedule	0510.SERVERS	Resource Group	TECHN	Technician

The installation costs in this job are **Resource Groups** with line type **Schedule**, just as in the previous job, so we invoice actual hours spent on the job.

The other lines are of type **Both Schedule and Contract**. This means we will invoice exactly what is in the budget. The job journal lines for these tasks should be posted with a blank line type.

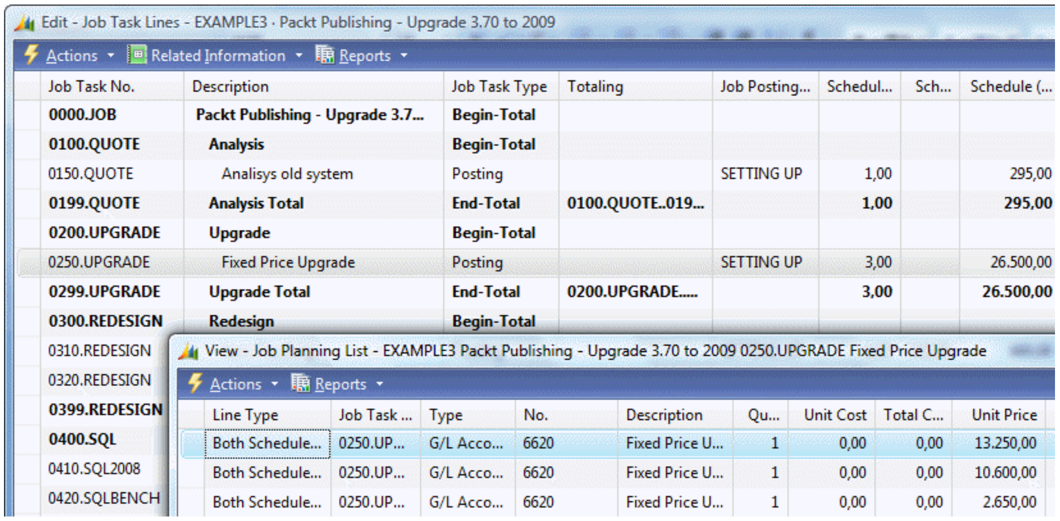
3 | The upgrade

Our customer requests an upgrade from Navision version 3.70 to Microsoft Dynamics NAV 2009. We can do this for a fixed price but we require a fee for analyzing the system.

For this job, **EXAMPLE3**, we can start with a limited number of **Job Task Lines**, just for the quote. When the customer agrees to do the upgrade we can add new job task lines.

Both the **Quote** and the **Upgrade** are fixed price and posted directly to the general ledger. This does not mean we cannot have our resources to register the actual hours using the job journal but the line type should be blank.

Another part of the upgrade is not done fixed price. The systems needs some redesign, a conversion to SQL Server 2008, and the customer wants additional training and support.


Edit - Job Task Lines - EXAMPLE3 · Packt Publishing - Upgrade 3.70 to 2009
Actions
Related Information
Reports

Job Task No.	Description	Job Task Type	Totaling	Job Posting...	Schedul...	Sch...	Schedule (...
0000.JOB	**Packt Publishing - Upgrade 3.7...**	**Begin-Total**					
0100.QUOTE	**Analysis**	**Begin-Total**					
0150.QUOTE	Analisys old system	Posting		SETTING UP	1,00		295,00
0199.QUOTE	**Analysis Total**	**End-Total**	**0100.QUOTE..019...**		**1,00**		**295,00**
0200.UPGRADE	**Upgrade**	**Begin-Total**					
0250.UPGRADE	Fixed Price Upgrade	Posting		SETTING UP	3,00		26.500,00
0299.UPGRADE	**Upgrade Total**	**End-Total**	**0200.UPGRADE.....**		**3,00**		**26.500,00**
0300.REDESIGN	**Redesign**	**Begin-Total**					
0310.REDESIGN							
0320.REDESIGN							
0399.REDESIGN							
0400.SQL							
0410.SQL2008							
0420.SQLBENCH							

View - Job Planning List - EXAMPLE3 Packt Publishing - Upgrade 3.70 to 2009 0250.UPGRADE Fixed Price Upgrade
Actions
Reports

Line Type	Job Task ...	Type	No.	Description	Qu...	Unit Cost	Total C...	Unit Price
Both Schedule...	0250.UP...	G/L Acco...	6620	Fixed Price U...	1	0,00	0,00	13.250,00
Both Schedule...	0250.UP...	G/L Acco...	6620	Fixed Price U...	1	0,00	0,00	10.600,00
Both Schedule...	0250.UP...	G/L Acco...	6620	Fixed Price U...	1	0,00	0,00	2.650,00

The fixed price part of the upgrade is invoiced in three phases. When the job starts we invoice 50%, when we deliver the test system we invoice 40% and 10% is invoiced three months after go-live.

This is done using lines of **Both Schedule and Contract** line type.

4 | The support team

For the support team, our policy is to create one job per fiscal year per Customer. We will use this job, **EXAMPLE4**, for invoicing the maintenance of the license and all support issues.

The support issues can be both little questions customers call us for, such as changing a report or a page, or implementing new features that requires only a few days work.

Each issue and new feature will be created as a job task line. The new features will be created by the account manager who sells the feature. We can then decide if the invoicing is done fixed price, using contract lines, or on time and materials using schedule lines.

Our support team also needs to use the job system, but we do not want them to manually create a new job task line for each support call, and we also want them to view all outstanding issues for all customers easily. For this we have created a new issue registration system which we discuss at the end of this chapter.

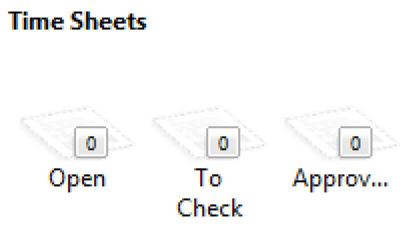
Each issue in the system is linked to a job task. When Support Engineers create a new issue, the job task line is automatically generated for them and they can use it in our time and billing system.

Time sheets

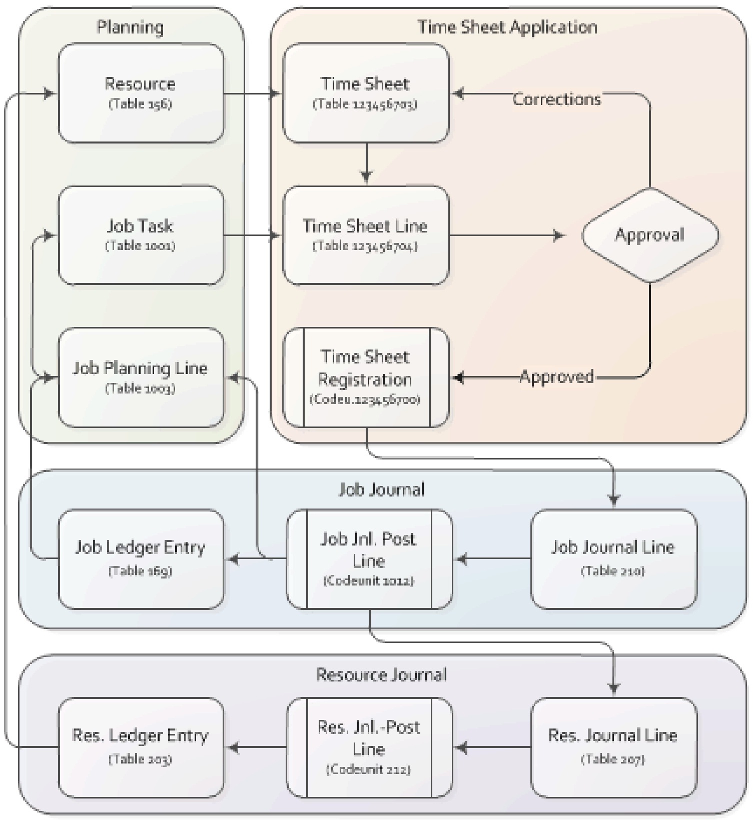
For all the jobs in our examples it is critical to have a solid registration of resource hours. In the standard Microsoft Dynamics NAV job application resources need to post a job journal for each combination of Job, Job Task, and Posting Date. This is not the way most people want to register their hours; therefore we have created a Time Sheet application.

Data and transaction model

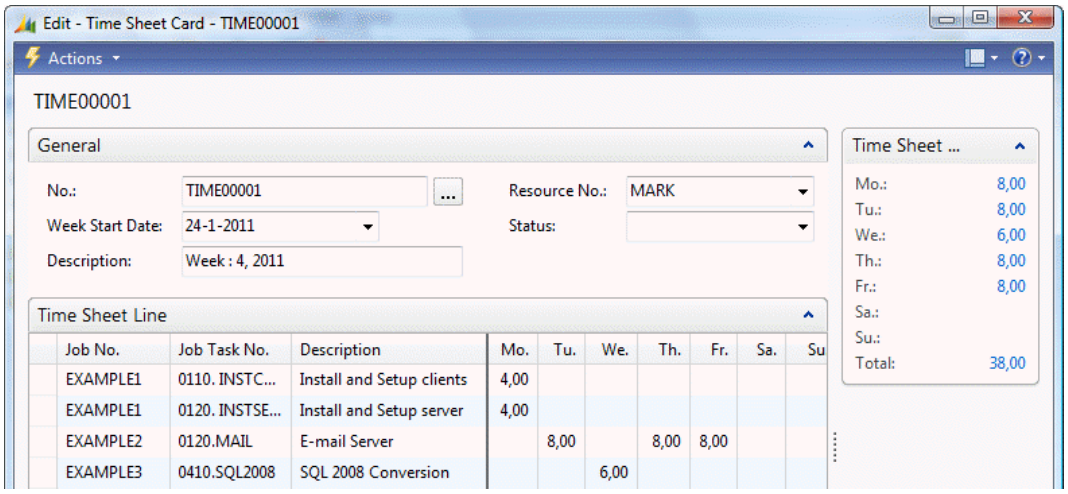
The Time Sheet application is layered above the job journal line and is created using resources and job tasks.



There is an approval process for the person responsible for the job allowing them to make corrections.



The time sheet is designed to be created for each week. The system automatically generates the week starting date and creates the description. After that the resource can create time sheet lines for each job task line, and populate the number of hours each day of the week.



No.:	TIME00001
Week Start Date:	24-1-2011
Description:	Week : 4, 2011
Resource No.:	MARK
Status:	

Job No.	Job Task No.	Description	Mo.	Tu.	We.	Th.	Fr.	Sa.	Su
EXAMPLE1	0110. INSTC...	Install and Setup clients	4,00						
EXAMPLE1	0120. INSTSE...	Install and Setup server	4,00						
EXAMPLE2	0120.MAIL	E-mail Server		8,00		8,00	8,00		
EXAMPLE3	0410.SQL2008	SQL 2008 Conversion			6,00				

Mo.:	8,00
Tu.:	8,00
We.:	6,00
Th.:	8,00
Fr.:	8,00
Sa.:	
Su.:	
Total:	38,00

If we look at this time sheet we can see, after it is updated, that Wednesday is missing two hours.

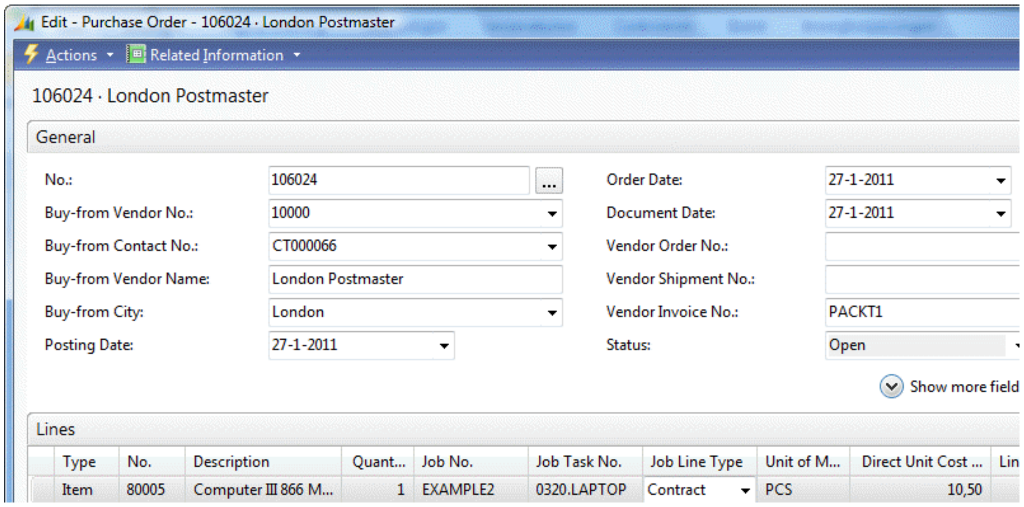
Purchasing

For some jobs it might be necessary to purchase items specifically for that job. Unfortunately job planning lines are not used in the requisition worksheet we looked at in *Chapter 5*, *Production*, or *Chapter 6*, *Trade*, so we need to create the purchase orders manually.

Expanding the requisition worksheet business logic for job planning lines is not recommended for two important reasons. First reason is that the code is very complex and cuts deep in the item reservation process. The second, more obvious reason, is that most companies that use jobs do not use the requisition worksheet as it is designed for production and trade businesses.

A better way of automatically creating the purchase orders for jobs would be to copy the drop shipment functionality from sales.

For this example we will create a manual **Purchase Order** looking like this:



Edit - Purchase Order - 106024 · London Postmaster

106024 · London Postmaster

General

No.:	106024	Order Date:	27-1-2011
Buy-from Vendor No.:	10000	Document Date:	27-1-2011
Buy-from Contact No.:	CT000066	Vendor Order No.:	
Buy-from Vendor Name:	London Postmaster	Vendor Shipment No.:	
Buy-from City:	London	Vendor Invoice No.:	PACKT1
Posting Date:	27-1-2011	Status:	Open

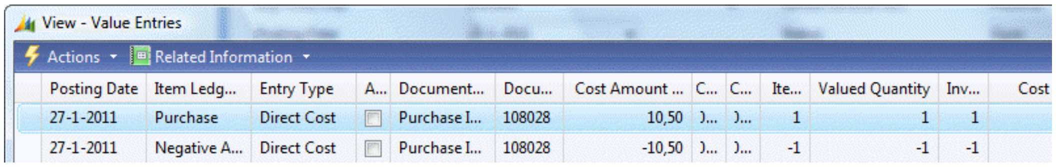
Lines

Type	No.	Description	Quant...	Job No.	Job Task No.	Job Line Type	Unit of M...	Direct Unit Cost ...	Lin
Item	80005	Computer III 866 M...	1	EXAMPLE2	0320.LAPTOP	Contract	PCS	10,50	

When purchasing for a job, the **Job No.** and **Job Task No.** fields should be used. If we set the **Job Line Type** to contract this item will be invoiced to the customer. Usually this is not required since it should have been in the Job Planning Lines already.

Item costing versus work in progress

After we post this purchase document and navigate from the purchase invoice we can see that the system has created two **Value Entries** for this item (as shown in the following screenshot):



View - Value Entries

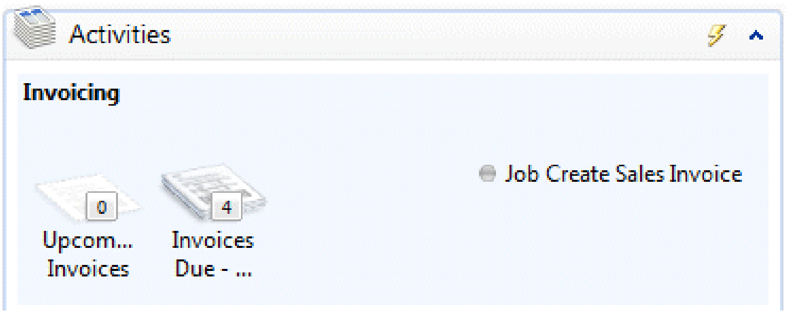
Posting Date	Item Ledg...	Entry Type	A...	Document...	Docu...	Cost Amount ...	C...	C...	Ite...	Valued Quantity	Inv...	Cost
27-1-2011	Purchase	Direct Cost	☐	Purchase I...	108028	10,50	)...	)...	1	1	1	
27-1-2011	Negative A...	Direct Cost	☐	Purchase I...	108028	-10,50	)...	)...	-1	-1	-1	

This is very important for the costing as we discussed in the previous chapters. Purchased items for jobs are not calculated as inventory but used for the Work in Progress calculation.

Invoicing

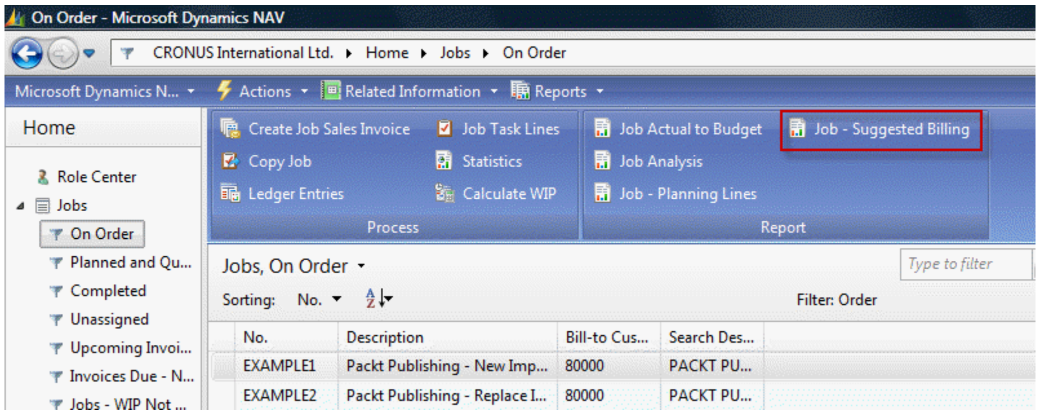
When everything in our jobs is set up as required and the job journal is used to post the usage, creating the invoices is a simple task.

In the Job Manager Role Center we can see if a job is due to be invoiced. This is done using a flow filter on the Planning Date field of the Job Planning Lines.



Each job planning line has a planning date. This can be used to schedule our resources but is far more useful for invoice scheduling. Each job planning line that is ready to be invoiced should get the invoice date in the planning date field.

The invoices can then be created using the batch report **Job Create Sales Invoice** (1093) but we can also preview the invoice by using report **Job Suggested Billing** (1011). This report can be started from each job.



No.	Description	Bill-to Cus...	Search Des...
EXAMPLE1	Packt Publishing - New Imp...	80000	PACKT PU...
EXAMPLE2	Packt Publishing - Replace I...	80000	PACKT PU...

The sales invoice lines are created using the same description as the job planning lines. To clarify information for the customer we can use extra text lines to add information to the invoice.

When the invoice is created and posted, the job task is updated with the actual invoicing information.

Calculating Work in Progress

As most jobs are not completed in a day or a week it is important to be able to determine the status of each job at any given point in time. This can be measured both in quantity and financially. In quantities we can see how much of the budget is being used by looking at the job task page. For financial progress we can calculate the Work in Progress or WIP.

WIP calculates the costs we incurred and the sales we invoiced on the job and creates job WIP entries for this. This can then be posted to the general ledger if required. The WIP amounts depend on the WIP method. Microsoft Dynamics NAV has five possible WIP calculation methods:

- **Cost Value**: The Cost Value is leading for WIP calculation. The WIP is the total percentage of job completion multiplied by the cost value. Job completion is calculated using `((Real Cost/Scheduled Cost) – (Really Invoiced/Scheduled Price)).`
- **Sales Value**: The Sales Value is leading for WIP Calculation. The WIP is the total percentage of job invoicing multiplied by the cost sales. Invoiced % is a field in the job task.
- **Cost of Sales**: The total cost is calculated for the WIP calculation.
- **Percentage of Completion**: The scheduled costs are multiplied by the job completion percentage.
- **Completed Contract**: All completed contract entries are used for WIP.

The online help shows some good explanations of the WIP methods.

Example

Let's create an example and calculate the WIP for each method.

We created a job with a total price of 1000 and total cost of 500. We used four resource hours worth 500 and cost 250. We invoiced nothing.

	Recog. Cost	Recog. Sales	WIP Cost	WIP Sales
Cost Value	125	0	125	0
Sales Value	250	250	0	250
Cost of Sales			250	
Percentage of Completion	250	250	0	250
Completed Contract			250	

Now we send an invoice to the customer for the hours spent. We invoice 500.

	Recog. Cost	Recog. Sales	WIP Cost	WIP Sales
Cost Value	375	500	-125	0
Sales Value	250	250	0	-250
Cost of Sales	500	500	-250	
Percentage of Completion	250	250	0	-250
Completed Contract			250	-500

In the last example we will use an item with cost of 250 which we cannot invoice. We now have 500 costs and 500 sales.

	Recog. Cost	Recog. Sales	WIP Cost	WIP Sales
Cost Value	500	500	0	0
Sales Value	250	250	0	-250
Cost of Sales	500	500	0	
Percentage of Completion	500	500	0	0
Completed Contract			500	-500

When the WIP is positive it means that we have done more than we have invoiced, when the WIP is negative we have invoiced more than we have done.

Each company that uses Microsoft Dynamics NAV should make their own decision on what WIP method to use. WIP methods can change for each job and even change during a job.

WIP post to general ledger

Some accountants want to post the WIP amounts to the general ledger. The benefit of doing this is to have all the financial information in one place for easier reporting.

The G/L accounts for the WIP posting are set up in the job posting group.

When posting WIP to the general ledger there is always a reversal posting. When a company does monthly reporting the WIP is posted on the last day of the month and reversed on the first day of the next month.

Changing jobs

In this chapter we have used some changes to the job functionality in order to make it work for Cronus International Ltd. to sell Microsoft Dynamics NAV.

Quantity budgeting

For some companies it is important to know the total number of hours required for a job and the number of hours used rather than the exact amounts.

For this we have created new flow fields in the **Job Task** table:

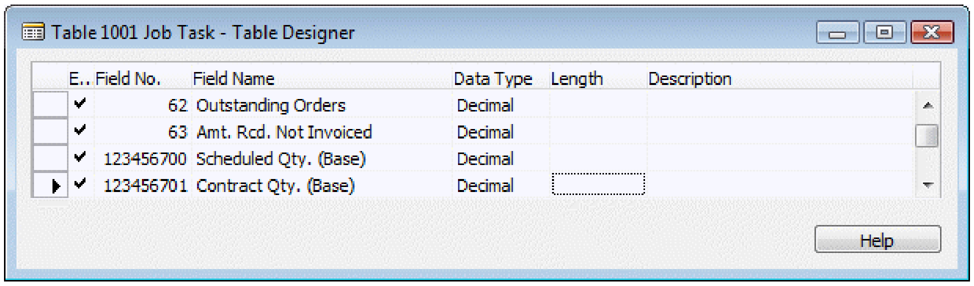


Table 1001 Job Task - Table Designer

E..	Field No.	Field Name	Data Type	Length	Description
✓	62	Outstanding Orders	Decimal		
✓	63	Amt. Rcd. Not Invoiced	Decimal		
✓	123456700	Scheduled Qty. (Base)	Decimal		
✓	123456701	Contract Qty. (Base)	Decimal		

The flow field definition is special.

```
Sum("Job Planning Line"."Quantity (Base)"
  WHERE (Job No. = FIELD(Job No.),
         Job Task No. = FIELD(Job Task No.),
         Job Task No. = FIELD(FILTER(Totaling)),
         Contract Line = CONST(Yes),
         Planning Date = FIELD(Planning Date Filter)))
```

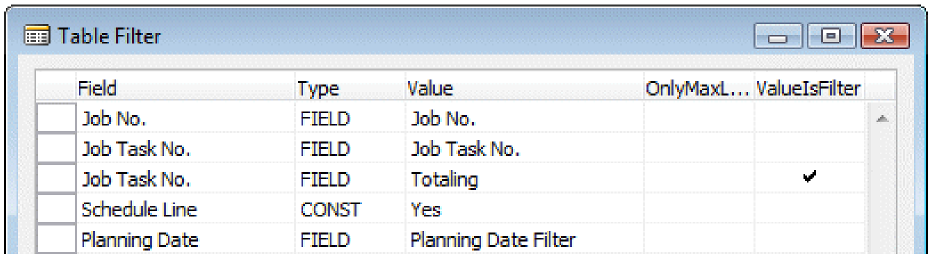
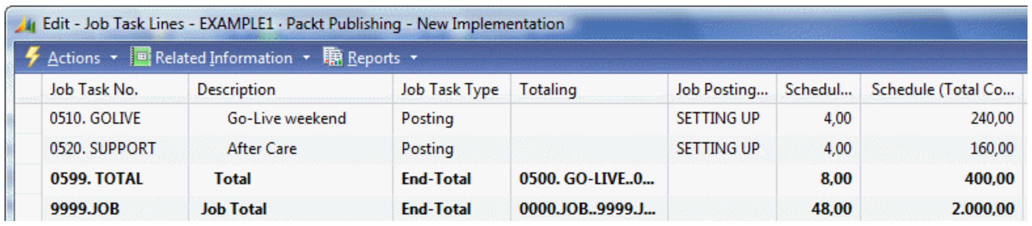


Table Filter

Field	Type	Value	OnlyMaxL...	ValueIsFilter
Job No.	FIELD	Job No.		
Job Task No.	FIELD	Job Task No.		
Job Task No.	FIELD	Totaling		✓
Schedule Line	CONST	Yes		
Planning Date	FIELD	Planning Date Filter		

The **Totaling** field is for the lines of type **End-Total**. The **ValueIsFilter** property ensures that the field will be interpreted as filter instead of a value.

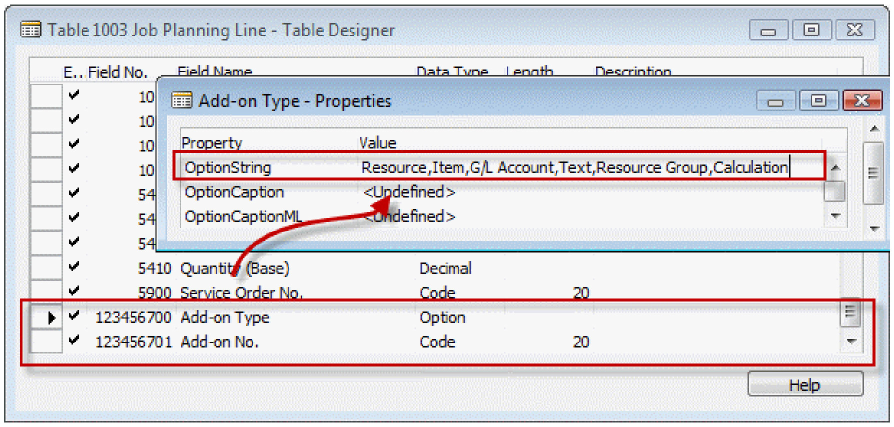

Edit - Job Task Lines - EXAMPLE1 · Packt Publishing - New Implementation
Actions
Related Information
Reports

Job Task No.	Description	Job Task Type	Totaling	Job Posting...	Schedul...	Schedule (Total Co...
0510. GOLIVE	Go-Live weekend	Posting		SETTING UP	4,00	240,00
0520. SUPPORT	After Care	Posting		SETTING UP	4,00	160,00
0599. TOTAL	**Total**	**End-Total**	**0500. GO-LIVE..0...**		**8,00**	**400,00**
9999.JOB	**Job Total**	**End-Total**	**0000.JOB..9999.J...**		**48,00**	**2.000,00**

The result is visible in the Job Task page (1002).

Resource groups

For scheduling we have implemented the possibility of using Resource Groups in the Job Planning Lines as well as Calculations. This is done by adding two new fields: **Add-on Type** and **Add-on No.**.



Property	Value
OptionString	Resource,Item,G/L Account,Text,Resource Group,Calculation
OptionCaption	<Undefined>
OptionCaptionML	<Undefined>

E..	Field No.	Field Name	Data Type	Length	Description
✓	10				
✓	10				
✓	10				
✓	10				
✓	54				
✓	54				
✓	54				
✓	5410	Quantity (Base)	Decimal		
✓	5900	Service Order No.	Code	20	
✓	123456700	Add-on Type	Option		
✓	123456701	Add-on No.	Code	20	

These fields replace the standard Type and No. fields on the pages allowing users to select these new options. The caption of the new fields matches the replacement fields.

```
Add-on No. - OnValidate()
CASE "Add-on Type" OF
  "Add-on Type"::Resource, "Add-on Type"::Item, "Add-on Type"::"G/L
Account", "Add-on Type"::Text:
    BEGIN
      VALIDATE(Type, "Add-on Type");
      VALIDATE("No.", "Add-on No.");
    END;
  "Add-on Type"::"Resource Group":
    BEGIN
      TESTFIELD("Line Type", "Line Type"::Schedule);
      VALIDATE(Type, Type::Text);
      VALIDATE("No.", '');
      ResGroup.GET("Add-on No.");
      Description := ResGroup.Name;
      "Resource Group No." := ResGroup."No.";
      GetJob;
      ResCost.SETRANGE(Type,
        ResPrice.Type::"Group(Resource)");
      ResCost.SETRANGE(Code, ResGroup."No.");
      IF ResCost.FINDFIRST THEN BEGIN
        "Unit Cost" := ROUND(
            CurrExchRate.ExchangeAmtLCYToFCY(
              "Currency Date","Currency Code",
              ResCost."Unit Cost","Currency Factor"),
            UnitAmountRoundingPrecision);
```

In the C/AL code we can make sure that when users select the values available in the standard product, the normal code is executed. If a user selects a resource group we execute our own business logic.

To make sure everything works as expected we use the type `Text` in the background. The `Line Type` is mandatory set to `Schedule` because we do not want to invoice resource groups, we just want them to be budgeted.

The Unit Cost and Unit Price are calculated using the Resource Cost and Resource Price tables, which support the use of Resource Groups. This is an inheritance from the previous Job functionality prior to version 5.0.

The pages Job Planning List (1007) and Job Planning Line Sub form (1022) are changed to show our add-on fields instead of the normal fields.

To completely finish this functionality we would also need to change the reports that show the Job Planning Lines and the C/AL code that creates the Job Planning Lines, when posting a Job Journal Line. This is not done in the example code for this chapter.

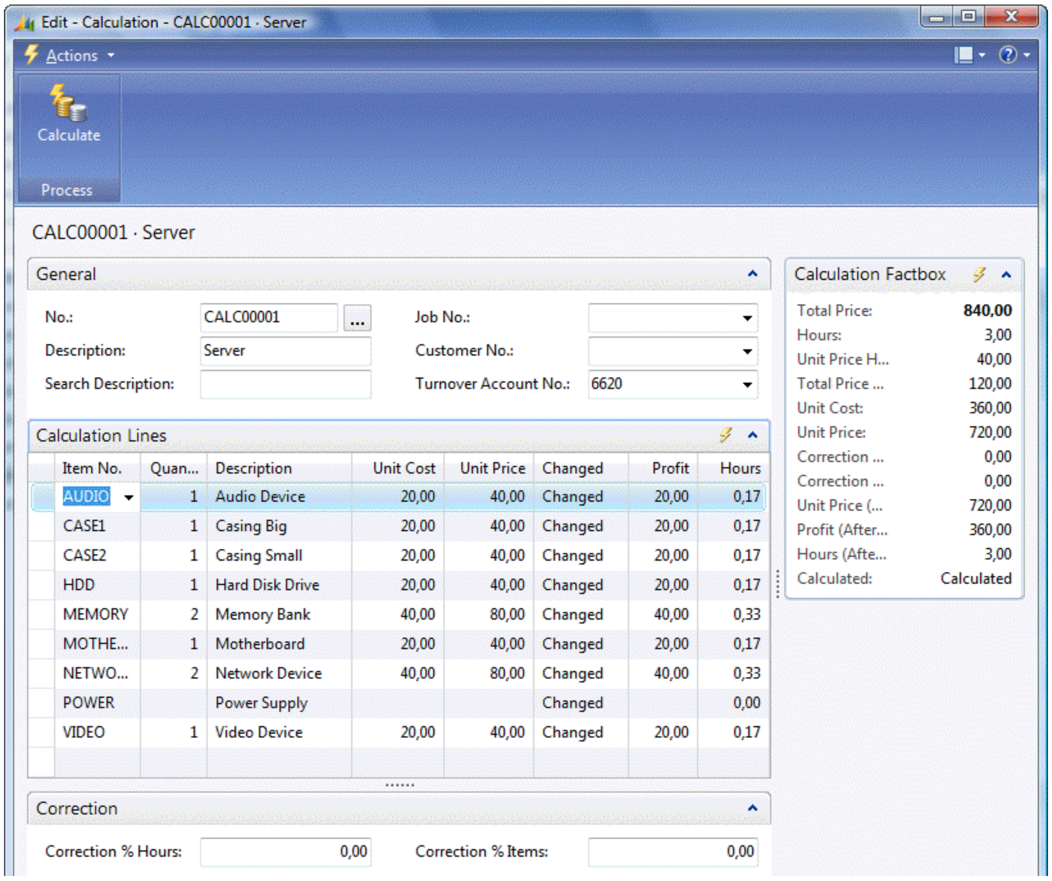
Calculations

Some companies using the Job functionality have a need for flexible calculations. In our example we use it to calculate the price of a computer system but other examples are book publishers or construction companies.

They want to know what it costs to create a product without exactly knowing which screws, hinges, or color of chipboard is used.

For these companies we designed a simple but effective calculation module.

In our database there are two example calculations, a server and a desktop system.



CALC00001 · Server

General

No.:	CALC00001	Job No.:	
Description:	Server	Customer No.:	
Search Description:		Turnover Account No.:	6620

Calculation Lines

Item No.	Quan...	Description	Unit Cost	Unit Price	Changed	Profit	Hours
AUDIO	1	Audio Device	20,00	40,00	Changed	20,00	0,17
CASE1	1	Casing Big	20,00	40,00	Changed	20,00	0,17
CASE2	1	Casing Small	20,00	40,00	Changed	20,00	0,17
HDD	1	Hard Disk Drive	20,00	40,00	Changed	20,00	0,17
MEMORY	2	Memory Bank	40,00	80,00	Changed	40,00	0,33
MOTHE...	1	Motherboard	20,00	40,00	Changed	20,00	0,17
NETWO...	2	Network Device	40,00	80,00	Changed	40,00	0,33
POWER		Power Supply			Changed		0,00
VIDEO	1	Video Device	20,00	40,00	Changed	20,00	0,17

Correction

Correction % Hours:	0,00	Correction % Items:	0,00

Calculation Factbox

Total Price:	**840,00**
Hours:	3,00
Unit Price H...	40,00
Total Price ...	120,00
Unit Cost:	360,00
Unit Price:	720,00
Correction ...	0,00
Correction ...	0,00
Unit Price (...	720,00
Profit (After...	360,00
Hours (Afte...	3,00
Calculated:	Calculated

The Calculation is designed using a header/line construction with a Number Series and a Line Number. The calculation lines are Items.

When a new Calculation is created some lines are automatically inserted. This is done in a C/AL function that is called from the `OnInsert` trigger.

The `OnInsert` trigger will also copy the default Unit Price for Hours from our setup table.

```
OnInsert()
CalcSetup.GET;

IF "No." = '' THEN BEGIN
  CalcSetup.TESTFIELD("Calculation Nos.");
  NoSeriesMgt.InitSeries(CalcSetup."Calculation Nos.",xRec."No.
Series",0D,"No.","No. Series");
END;

"Unit Price Hours (LCY)" := CalcSetup."Unit Price Hours";
InitLines;
```

The `InitLines` function creates a Calculation Line for each Item marked as **Calculation Item**. This is a new field we added to the Item table.

```
InitLines()
CalcLn.RESET;

i := 0;
Item.SETRANGE("Calculation Item", TRUE);
IF Item.FINDSET THEN REPEAT
  i += 10000;
  CalcLn."Calculation No." := "No.";
  CalcLn."Line No." := i;
  CalcLn.VALIDATE("Item No.", Item."No.");
  CalcLn.INSERT;
UNTIL Item.NEXT = 0;
```

In the Calculation we can choose how many we will use from each item, and the system will not only calculate the cost and price, but also the required number of hours that is required. The Unit Cost and Unit Price are used from the Item table. Hours is calculated from a new field, Minutes, which we added to the Item table.

```
Calculate()
CalcLn.RESET;
CalcLn.SETRANGE("Calculation No.","No.");
CalcLn.CALCSUMS("Unit Cost", "Unit Price", Profit, Hours);
CalcLn.FIND('-');
CalcLn.MODIFYALL(Changed,Calculated::Calculated);

CalcLn.CALCSUMS("Unit Cost", "Unit Price", Hours);

"Unit Cost" := CalcLn."Unit Cost";
```

```
"Unit Price" := CalcLn."Unit Price";
Profit := "Unit Price" - "Unit Cost";
Hours := CalcLn.Hours;

Correct;

"Total Price Hours (LCY)" := "Hours (After Correction)" * "Unit Price
Hours (LCY)";
"Total Price" := "Total Price Hours (LCY)" +
  "Unit Price (After Correction)";
Calculated := Calculated::Calculated;
MODIFY;

Correct()
"Unit Price (After Correction)" := "Unit Price" + ("Unit Price" *
("Correction % Items" / 100));
"Profit (After Correction)" :=
  "Unit Price (After Correction)" - "Unit Cost";
"Hours (After Correction)" :=
  Hours + (Hours * ("Correction % Hours" / 100));
```

When we now use the `Calculate` function, the system will generate a total `Unit Cost`, `Unit Price`, and `Hours` for this product to be created. Flexibility is added to the system by allowing users to correct hours and usage with a percentage.

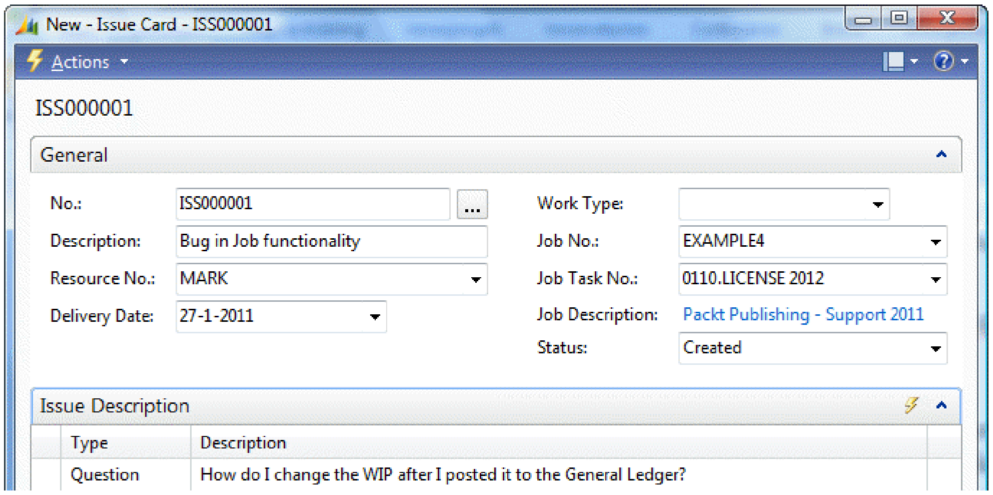
The Calculation can be used in a Job Planning Line the same way as the Resource Groups earlier, the only difference is that we use the G/L Account type on the background to invoice a calculation fixed price. Let's look at the C/AL code in the `OnValidate` trigger of the `Add-On No.` field in the Job Planning Line:

```
Add-on No. - OnValidate()
CASE "Add-on Type" OF
  "Add-on Type"::Resource ... "Add-on Type"::Text:
      ...
  "Add-on Type"::"Resource Group":
      ...
  "Add-on Type"::Calculation:
    BEGIN
      Calc.GET("Add-on No.");
      IF Calc."Turnover Account No." = '' THEN BEGIN
        TESTFIELD("Line Type", "Line Type"::Schedule);
        VALIDATE(Type, Type::Text);
        VALIDATE("No.", '');
      END ELSE BEGIN
        TESTFIELD("Line Type",
          "Line Type"::"Both Schedule and Contract");
        VALIDATE(Type, Type::"G/L Account");
        VALIDATE("No.", Calc."Turnover Account No.");
      END;
      Description := Calc.Description;
      GetJob;
```

To complete this functionality we would create a method to use the hours in the calculation for the Resource planning. This can be done using job planning lines of line type Schedule with no unit cost and unit price.

Issue registration

For our support team we have implemented an issue registration solution. This allows them to have a simple application where they can register issues for all customers, and keep track of their status without going in and out of each Job.



No.:	ISS000001	Work Type:	
Description:	Bug in Job functionality	Job No.:	EXAMPLE4
Resource No.:	MARK	Job Task No.:	0110.LICENSE 2012
Delivery Date:	27-1-2011	Job Description:	Packt Publishing - Support 2011
		Status:	Created

Type	Description
Question	How do I change the WIP after I posted it to the General Ledger?

The Issue registration is a header/line construction with a number series and a line number. The lines can be used to phrase questions and answers.

When a support engineer creates a new issue, the system will create the Job Task automatically. Let's have a look at the C/AL code that does that:

```
CreateJobTask()
TESTFIELD("Job No.");
TESTFIELD("Job Task No.", '');

OldJobTask.SETRANGE("Job No.", "Job No.");
OldJobTask.SETRANGE("Job Task Type",
  OldJobTask."Job Task Type"::Posting);
IF OldJobTask.ISEMPTY THEN
```

```
    OldJobTask.SETRANGE("Job Task Type",
      OldJobTask."Job Task Type"::"Begin-Total");
  OldJobTask.FINDLAST;
  JobTask."Job No." := "Job No.";
  JobTask."Job Task No." := INCSTR(OldJobTask."Job Task No.");
  JobTask.Description := Description;
  JobTask."Job Task Type" := JobTask."Job Task Type"::Posting;
  JobTask.INSERT(TRUE);
  CODEUNIT.RUN(CODEUNIT::"Job Task-Indent Direct", JobTask);
  "Job Task No." := JobTask."Job Task No.";
```

The system searches for the last `Job Task` of the type `Posting` in the Job. If that cannot be found, it searches for the last `Begin-Total` line.

Assuming this line exists we create a new Job Task line using the `INCSTR` function to increment the number. The description is copied to the job task.

The support engineers can now register their hours on this job lask.

This piece of C/AL code is very simple but shows how effective a small solution can be without even touching any of the standard Microsoft Dynamics NAV objects. This is a very safe method of development.

Time sheet

We already discussed the functionality of the Time Sheet application in this chapter. Let's have a look at how it was created.

The Time Sheet application is also a header/line construction with a number series and a line number. It is designed to be used weekly and therefore automatically creates the description based on the date. It calculates the week number and the year.

```
Week Start Date - OnValidate()
IF DATE2DWY("Week Start Date", 1) <> 1 THEN
  FIELDERROR("Week Start Date");

Description := Text000 + FORMAT("Week Start Date", 0, '<Week>,
  <Year4>');
```

The user can change the `Description` if required.

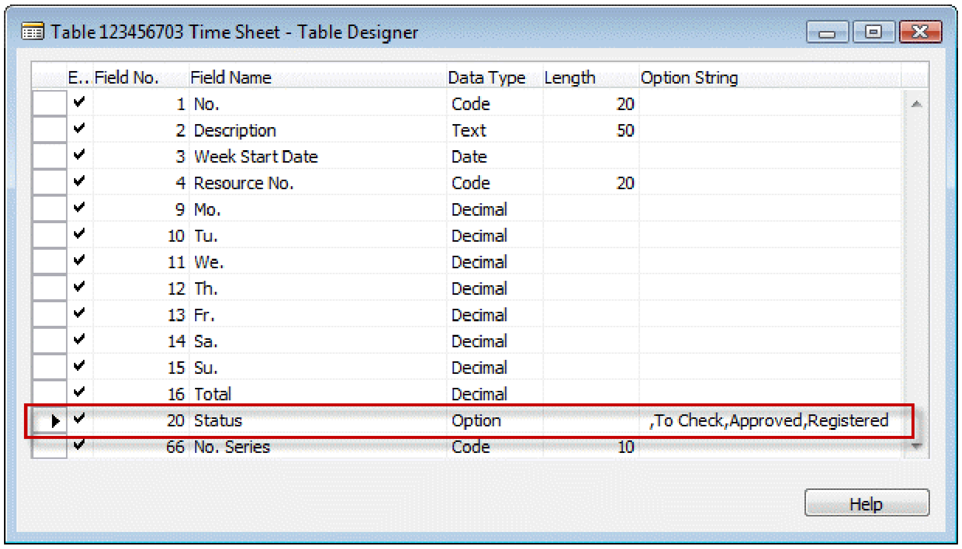


Table 123456703 Time Sheet - Table Designer

E..	Field No.	Field Name	Data Type	Length	Option String
✓	1	No.	Code	20	
✓	2	Description	Text	50	
✓	3	Week Start Date	Date		
✓	4	Resource No.	Code	20	
✓	9	Mo.	Decimal		
✓	10	Tu.	Decimal		
✓	11	We.	Decimal		
✓	12	Th.	Decimal		
✓	13	Fr.	Decimal		
✓	14	Sa.	Decimal		
✓	15	Su.	Decimal		
✓	16	Total	Decimal		
✓	20	Status	Option		,To Check,Approved,Registered
✓	66	No. Series	Code	10	

To support a workflow the **Time Sheet** table has a **Status** field. During the work week a resource can keep track of the things he did and change everything. When the week is finished they can change the status to **To Check**. Then the person responsible for the Job can check whether everything is valid, and change the status to **Approved**. The Time Sheet can then be **Registered**, this is done using a codeunit we will discuss later.

The **Mo.**, **Tu.**, **We.**, **Th.**, **Fr.**, **Sa.**, and **Su.** fields are Flowfields used in the Fact Box.

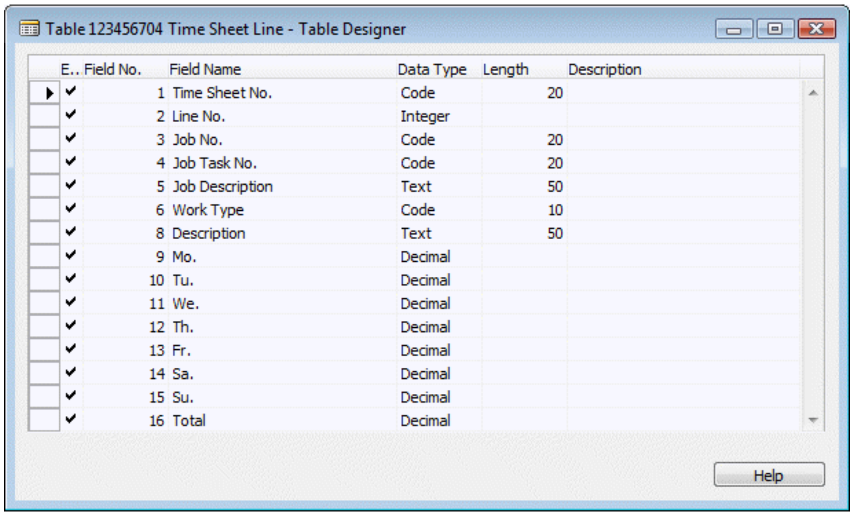


Table 123456704 Time Sheet Line - Table Designer

E..	Field No.	Field Name	Data Type	Length	Description
✓	1	Time Sheet No.	Code	20	
✓	2	Line No.	Integer		
✓	3	Job No.	Code	20	
✓	4	Job Task No.	Code	20	
✓	5	Job Description	Text	50	
✓	6	Work Type	Code	10	
✓	8	Description	Text	50	
✓	9	Mo.	Decimal		
✓	10	Tu.	Decimal		
✓	11	We.	Decimal		
✓	12	Th.	Decimal		
✓	13	Fr.	Decimal		
✓	14	Sa.	Decimal		
✓	15	Su.	Decimal		
✓	16	Total	Decimal		

The **Time Sheet Line** allows users to create a line for each Job Task we worked on during the week. Each day of the week has its own column for easy registration. After validating each day the total is calculated using a function.

```
CalcTotal()
TESTFIELD("Job Task No.");

Total :=
  "Mo." + "Tu." + "We." + "Th." + "Fr." + "Sa." + "Su.";
```

The **Total** field is used to calculate the total hours per week in a flow field.

Registration

The Time Sheet application has a registration codeunit. As the Time Sheet application is a document covering the Job Journal, this checks the status of the time sheet, and then creates and posts a Job Journal Line for each day if required.

```
OnRun()
TESTFIELD(Status, Status::Approved);

TimeSheet := Rec;

TimeSheetLn.SETRANGE("Time Sheet No.", "No.");
IF TimeSheetLn.FINDSET(TRUE) THEN REPEAT
  Job.GET(TimeSheetLn."Job No.");
  IF TimeSheetLn."Mo." <> 0 THEN
    CreateJnlLn("Week Start Date", TimeSheetLn."Mo.");
  IF TimeSheetLn."Tu." <> 0 THEN
    CreateJnlLn("Week Start Date" + 1, TimeSheetLn."Tu.");
  IF TimeSheetLn."We." <> 0 THEN
    CreateJnlLn("Week Start Date" + 2, TimeSheetLn."We.");
  IF TimeSheetLn."Th." <> 0 THEN
    CreateJnlLn("Week Start Date" + 3, TimeSheetLn."Th.");
  IF TimeSheetLn."Fr." <> 0 THEN
    CreateJnlLn("Week Start Date" + 4, TimeSheetLn."Fr.");
  IF TimeSheetLn."Sa." <> 0 THEN
    CreateJnlLn("Week Start Date" + 5, TimeSheetLn."Sa.");
  IF TimeSheetLn."Su." <> 0 THEN
    CreateJnlLn("Week Start Date" + 6, TimeSheetLn."Su.");
UNTIL TimeSheetLn.NEXT = 0;

Status := Status::Registered;
MODIFY;
```

At the end of the routine we change the `Status` to `Registered`.

```
CreateJnlLn()
JobJnlLn.INIT;
JobJnlLn."Job No." := TimeSheetLn."Job No.";
JobJnlLn."Job Task No." := TimeSheetLn."Job Task No.";
JobJnlLn.VALIDATE("Line Type", JobJnlLn."Line Type"::Contract);
JobJnlLn."Posting Date" := PostingDate;
JobJnlLn."Document Date" := PostingDate;
JobJnlLn."Document No." := TimeSheetLn."Time Sheet No.";
JobJnlLn.Type := JobJnlLn.Type::Resource;
JobJnlLn.VALIDATE("No.", TimeSheet."Resource No.");
JobJnlLn."Work Type Code" := TimeSheetLn."Work Type";
JobJnlLn.Description := TimeSheetLn.Description;
JobJnlLn.VALIDATE(Quantity, Qty);
JobJnlPostLine.RunWithCheck(JobJnlLn,TempJnlLineDim);
```

The Job Journal Line records are of type `Contract` to allow invoicing.

Summary

In this chapter we learned how to implement the Job functionality of Microsoft Dynamics NAV. We discussed different strategies of setting up Jobs and Job Tasks.

We created several examples with different invoicing methods using the Job Planning Lines in a creative way.

The Job Journal should be used to register usage on Jobs, but we discussed how to enhance this by creating a simple but effective Time Sheet application.

When Purchasing Items for Jobs the Items are not used for Costing, but in the Work in Progress calculation we discussed in detail.

Invoicing is done automatically when everything happens as it should.

Lastly, we designed some small enhancements for the Job module without making big changes in the standard application.

This was the last chapter about the functionality of Microsoft Dynamics NAV. We discussed all possibilities of the application, and how they should be changed without risking breaking anything.

In the next chapter we will look at how Microsoft Dynamics NAV can interface with other applications.

9 Interfacing

When the first version of Microsoft Dynamics NAV for Windows was released in 1995 the system was very closed. It was possible to import and export data using flat text files and that was basically it. These flat text files would be placed on a floppy disk and sent by postal mail. Internet and e-mail were just coming and large USB sticks were a dream.

Since then the world has changed tremendously. Internet, e-mail, ActiveX, SQL Server and .NET changed the way we think about interfacing with applications.

Today Microsoft Dynamics NAV 2009 has a completely open database and supports a wide range of interfacing possibilities which we will learn in this chapter.

In this chapter, we will first discuss the available interfacing technologies and the interfaces available in the standard product. Then we will talk about interfacing methodology and how to create reliable interfaces.

At the end of the chapter, we will create some sample interfaces and see how the future will further improve interfacing.

After reading this chapter, you will have a good understanding of what interfaces the product supports out of the box, what interfacing technologies to use, and how to design a solid business-to-business interface.

Interface types

When discussing an interface we usually start with the technology, but before that, some other basic questions need answering such as the following:

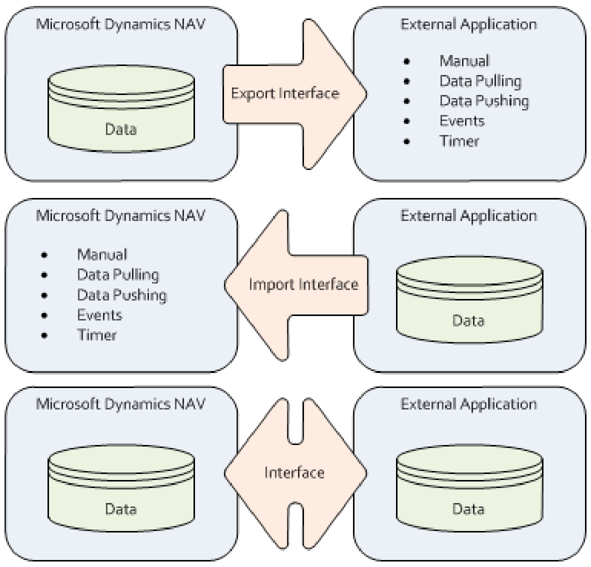
- Does it need to import, export, or both?
- Is it started manually or automatically?
- Is the interface timer or event driven?

Let's discuss these questions.

Import and export

The first question is if the interface should only export data from Microsoft Dynamics NAV or if it would also import data to the system which then needs to be processed.

When importing and exporting, the data process can be started manually by an end-user using data pulling or data pushing. The interface can also be event or timer driven.



Manual

When an interface is manual, the first application has an export process and another application has an import process. The end-user first manually starts the export process and then manually starts the import process in another application, usually saving the data to a flat-file. This is a very classical approach to interfacing.

An example of manual interfacing is exporting tele-banking information from Microsoft Dynamics NAV or sending XBRL files to your accountant.

Data pulling

When using data pulling to export data, the interface is started from an external application. This application will read the data from the database and process it.

When using data pulling to import data, the interface is started from the application which reads and processes data from another application.

Data pushing

If an interface uses data pushing, the exporting application writes the data to the other data source. This method is used when the data in the other application does not need further processing. A typical example is exporting data from Microsoft Dynamics NAV to Microsoft Office applications such as Word or Excel.

Event driven versus timer driven

When data pushing or data pulling is combined with the use of events or timers, there is no longer any need for end-user interference. The interface will then run automatically.

We will discuss these methods in detail later in this chapter when we discuss interface methodologies.

Interfacing technologies

In Microsoft Dynamics NAV there are a wide range of methods to interface. Each method is useful for certain types of interfacing and less useful for other types. We will discuss all available methods in the C/SIDE development platform.

File

Both Flat files and XML files are supported by Microsoft Dynamics NAV.

Flat files have been available since the introduction of the product in 1995 using Data Ports for the classic Clients.

XML support was introduced in version 3.60 as an extra option for data ports.

Version 4.0 introduced the XMLPort object that replaced the data port for importing and exporting XML files.

In the Role Tailored Client for Microsoft Dynamics NAV 2009, data ports are no longer supported. The XMLPort has taken over the task by having a flat file option.

Automation Control and OCX

The implementation for Microsoft COM and ActiveX in Microsoft Dynamics NAV is referred to as Automation Control or OCX.

OCX

OCX is short for **Ole Control Extension**, which was the predecessor of ActiveX. OLE was introduced in 1990 and ActiveX in 1996. Although OCX is a 20 year old technology it is loved by the Microsoft Dynamics NAV developers. Most commonly implemented OCX types are Microsoft Common Dialog and Microsoft COM control.

Although supported in Microsoft Dynamics NAV 2009 it is very likely that this old technology will no longer be supported in future versions of the product.

Automation Control

Automation Control or ActiveX allows software applications to be reused as embedded parts of another application. Most Microsoft applications support being used in such a way. Examples are Microsoft office, Windows Scripting Host, and **ActiveX Data Objects** (**ADO**).

Microsoft Dynamics NAV has support for Automation Control. The application can be consumed by other applications using the **C/FRONT.dll** which is shipped with the product. We will discuss **C/FRONT** later on in this chapter.

Consuming Automation Control is done using interface exposing methods and properties.

The most commonly used and generic interface is iUnknown. This is also the only Automation Control interface supported by Microsoft Dynamics NAV. If the Automation Control uses other interfaces, a wrapper should be created in Visual Studio transforming the interface to iUnknown. We should also create a wrapper when the Automation Controls needs to be embedded using a form control.

More information about the iUnknown interface and COM technology can be found here : `http://en.wikipedia.org/wiki/IUnknown`.

Events

Most Automation Controls allow data to be pushed. Using events for automation control, it is also possible to start business logic in Microsoft Dynamics NAV when something happens in the other applications.

A classic example used by many Microsoft Dynamics NAV developers is using the `ntimer.dll` that is shipped with the product. We will discuss this example later on in this chapter in the section "Example Interfaces".

.NET

Service Pack 1 for Microsoft Dynamics NAV 2009 introduced using embedded .NET controls in the Role Tailored Client. This technology is called Client Add-ins. We will discuss this technology later in this chapter.

Consuming .NET controls is only supported in the current version of Microsoft Dynamics NAV when it is wrapped in a COM control.

Automation wrappers

To work around the issues with embedded Automation Control, the iUnknown interface, and .NET support, many developers use COM Wrappers. This converts the methods and properties to the format that Microsoft Dynamics NAV supports.

An example of how to create a com wrapper for a .NET control can be found on `https://blogs.msdn.com/freddyk/archive/2009/03/18/integration-to-virtual-earth-part-1-out-of-4.aspx`

ODBC/ADO

Open Database Connectivity was developed in 1992 with the goal of allowing all types of databases to exchange data in a unified way.

ODBC for Microsoft Dynamics NAV allows both reading and writing in the application database as well as reading and writing to other databases.

When using the SQL Server database, we can use all SQL Server interfacing features including ODBC. The Native database supports ODBC using the C/ODBC driver.

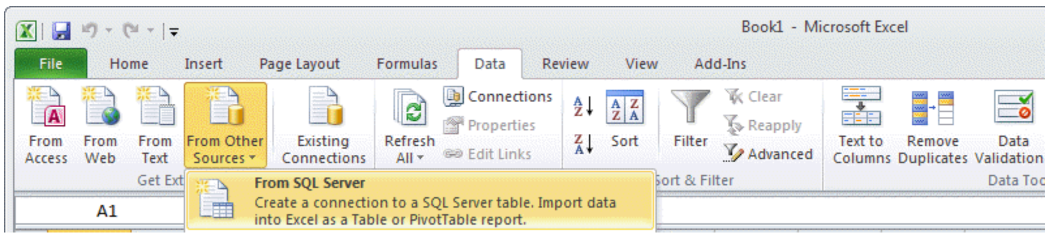
Using ODBC is more advanced and requires basic knowledge of T-SQL Statements.

Reading from Microsoft Dynamics NAV

To read data from the database it is only required to have a valid ODBC driver installed on the Windows Machine you are using and credentials to log in to the database.

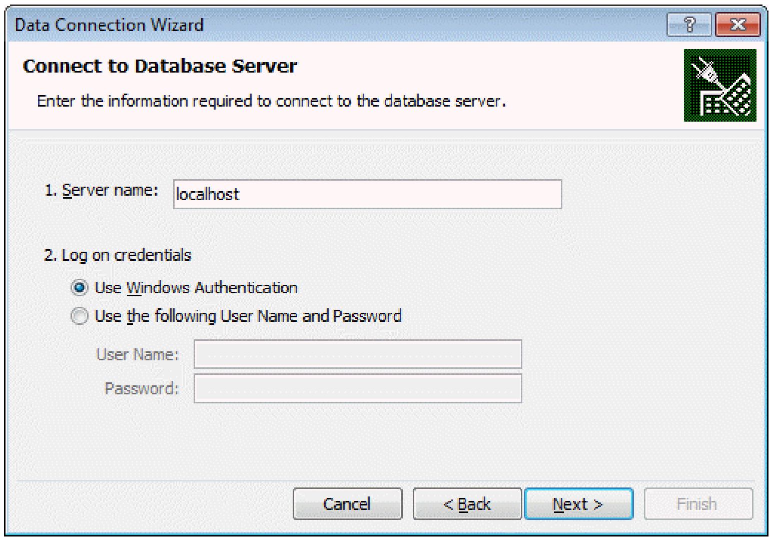
Let's create an example to import data from Microsoft Dynamics NAV using Excel:

1. Open Microsoft Excel and select **Data**.



From SQL Server
Create a connection to a SQL Server table. Import data into Excel as a Table or PivotTable report.

2. Select a **Server name** and valid credentials.


Data Connection Wizard

Connect to Database Server

Enter the information required to connect to the database server.

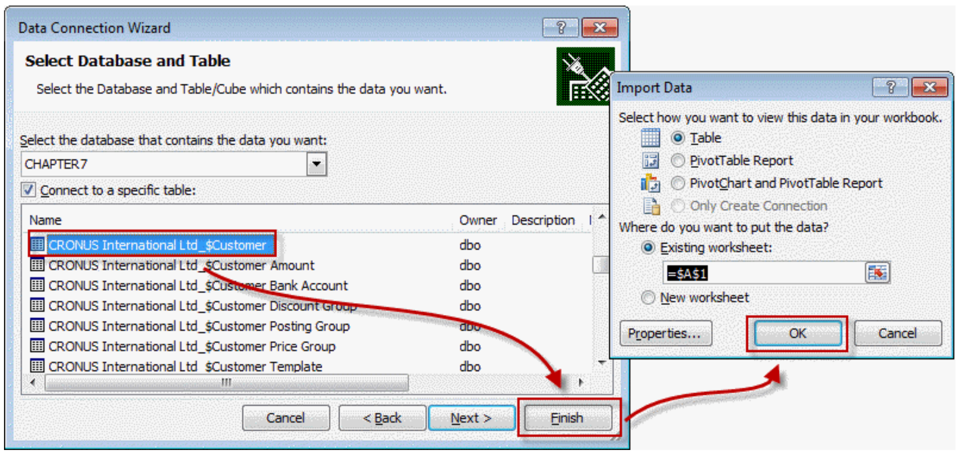
1. Server name: localhost

2. Log on credentials

[✓ Use Windows Authentication]
Use the following User Name and Password

User Name:
Password:

3. Select a Database and the Table you want to view. In our example, we will select the Customer Table. Then select **Finish** and **OK**.



Data Connection Wizard

Select Database and Table

Select the Database and Table/Cube which contains the data you want.

Select the database that contains the data you want: CHAPTER7

☑ Connect to a specific table:

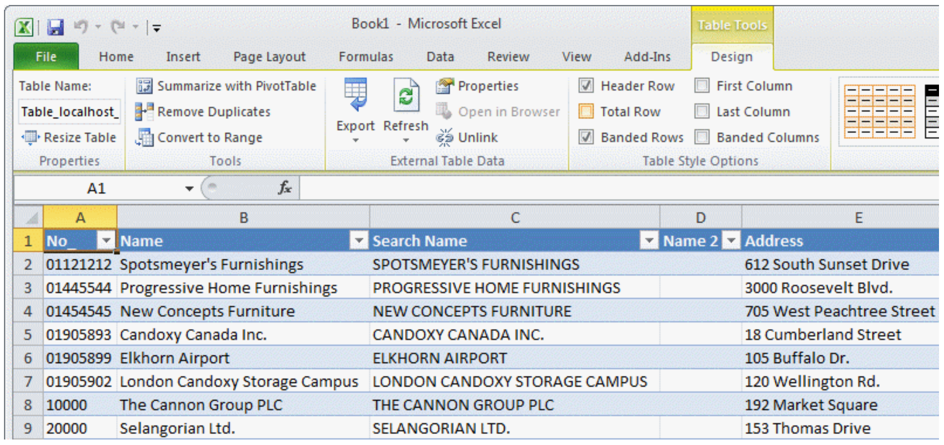
Name	Owner	Description
CRONUS International Ltd_$Customer	dbo	
CRONUS International Ltd_$Customer Amount	dbo	
CRONUS International Ltd_$Customer Bank Account	dbo	
CRONUS International Ltd_$Customer Discount Group	dbo	
CRONUS International Ltd_$Customer Posting Group	dbo	
CRONUS International Ltd_$Customer Price Group	dbo	
CRONUS International Ltd_$Customer Template	dbo	

Import Data

Select how you want to view this data in your workbook.
[✓ Table] PivotTable Report / PivotChart and PivotTable Report / Only Create Connection

Where do you want to put the data?
[✓ Existing worksheet:] =A1 / New worksheet

4. We now have the Microsoft Dynamics NAV data in Excel.



No_	Name	Search Name	Name 2	Address
01121212	Spotsmeyer's Furnishings	SPOTSMEYER'S FURNISHINGS		612 South Sunset Drive
01445544	Progressive Home Furnishings	PROGRESSIVE HOME FURNISHINGS		3000 Roosevelt Blvd.
01454545	New Concepts Furniture	NEW CONCEPTS FURNITURE		705 West Peachtree Street
01905893	Candoxy Canada Inc.	CANDOXY CANADA INC.		18 Cumberland Street
01905899	Elkhorn Airport	ELKHORN AIRPORT		105 Buffalo Dr.
01905902	London Candoxy Storage Campus	LONDON CANDOXY STORAGE CAMPUS		120 Wellington Rd.
10000	The Cannon Group PLC	THE CANNON GROUP PLC		192 Market Square
20000	Selangorian Ltd.	SELANGORIAN LTD.		153 Thomas Drive

[Since Flow Fields are not actual fields in the SQL Server database, we cannot use them in ODBC.]

Writing to Microsoft Dynamics NAV

Directly writing data to the Microsoft Dynamics NAV database using ODBC is not recommended as a best practice. The reason for this is the missing business logic at this interface level.

When writing via ODBC, we directly address SQL Server without allowing the C/AL Business Logic to validate the data we create. The C/AL data normally ensures data integrity for the business rules we develop. The same applies when using the C/ODBC driver for the Native database.

To work around this issue, the data can be saved in a special interface buffer table and processed by a C/AL transaction using an Application Server or can be started from the User Interface.

Talking to other databases

To use ODBC to read and write data from Microsoft Dynamics NAV to other databases, it is recommended to use ActiveX Data Objects (ADO). ADO is a Microsoft Technology that allows using an ActiveX interface to connect using ODBC. Using ADO allows us to read and write to the database on the other end.

We could even use ADO to connect to the Microsoft Dynamics NAV SQL Server database and run SQL Statements from C/AL code.

We will use ADO in the "Interface methodology" section of this chapter.

SQL Server interfacing

If Microsoft Dynamics NAV runs on top of a SQL Server database we can use all available technologies in SQL Server to get data in and out. This offers a wide range of options that goes beyond the scope of this book, but let's briefly discuss some of them:

- **Linked servers**: In SQL Server it is possible to set up linked servers. This allows us to send queries to other databases such as other SQL Servers or MS Access or Oracle and create views based on this data.
- **Views**: A view in SQL Server is a saved query with a fixed result set that can be interpreted as a table. In C/Side we can use a view as a data source for a table using the Linked Object property and create a Page or Report based on this data source.
- **DTS Packages and Integration Services**: **SQL Server Integration Services** (**SSIS**) replaces DTS Packages as the primary component for SQL Server to integrate with other databases. Using SSIS requires good knowledge and skills of both SQL Server and Microsoft Dynamics NAV.

- **Reporting Services**: **SQL Server Reporting Services** or **SSRS** is a server based reporting platform that can be integrated with SharePoint allowing users to design RDL reports based on T-SQL queries.
- **Analysis Services**: **SQL Server Analysis Services** or **SSAS** is Microsoft's answer to the OLAP, BI, and Data mining requirements of their customers.

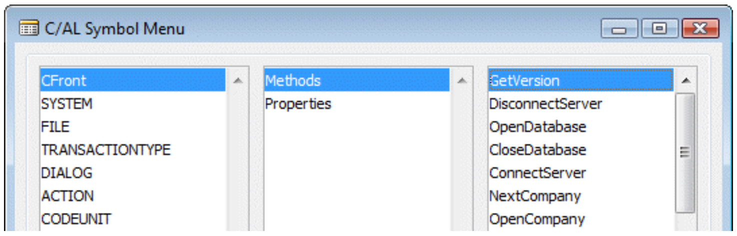
[Another SQL Server component we can use is the SQL Server Agent. This component allows us to schedule interface tasks that run directly on the database.]

C/FRONT

C/FRONT is an ActiveX component that can be used to call Microsoft Dynamics NAV from other applications. C/Front uses both OCX and .NET technology.

Before Web services were introduced in Microsoft Dynamics NAV 2009, C/Front was the only technology available to interface directly into the Microsoft Dynamics NAV Application from other systems.

The `C/Front.dll` allows a range of properties and methods to be used and can also be used from within C/AL.



[More information on using C/FRONT can be found on MDSN: `http://msdn.microsoft.com/en-us/library/dd338953.aspx`.]

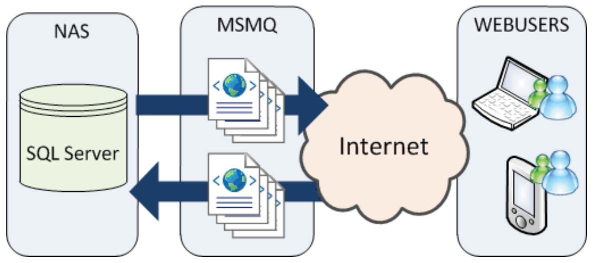
Microsoft Message Queue

Microsoft Message Queue or **MSMQ** allows applications to integrate that run asynchronously with an unreliable connection. This interfacing technology is very popular for websites that use information from Microsoft Dynamics NAV and send information back to the database.

To use MSMQ from Microsoft Dynamics NAV, it is required to use Automation Control. An example in the standard application that uses MSMQ is Employee Portal.

NAS

MSMQ is always combined with an Application Server to handle the requests sent back by the website.



The web users can be employees from the company using a web solution for timesheet registration or a PDA or customers using a web shop.

Web services

With the success of the internet grew the need for an API that is designed for communication using this platform. Web services were introduced in the beginning of the 21st century using XML and standard communication protocols such as HTTP.

Microsoft Dynamics NAV is capable of both consuming and exposing web services. To expose web services, it is a requirement to use the Service Tier and SQL Server.

The most important difference between Web services and Microsoft Message Queue is that Web services is real time and only runs when the connection between both applications is active whilst MSMQ will queue the requests until the connection has been re-established.

Consuming web services in NAV

To consume a web service in C/AL we use an ActiveX control Microsoft XMLHTTP. This can send an XML file to a web service listener and wait for the answer.

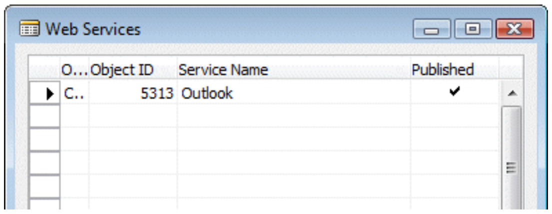
Examples of web services are Address Verification, Shipping Rates and Services (UPS, FedEx), Currency Exchange Rates, or Credit Card number validation.

A very good example of consuming a web service using C/AL can be found at `http://mibuso.com/blogs/ara3n/2008/03/06/`.

When developing a web service interface, make sure to have good documentation of the web service format and to have a valid test account.

Exposing a NAV web service

In Microsoft Dynamics NAV 2009 every Page object and most Codeunits can be exposed as a web service. This can be done using the Web Service Table (2000000076).


Web Services

O...	Object ID	Service Name	Published
C..	5313	Outlook	✓

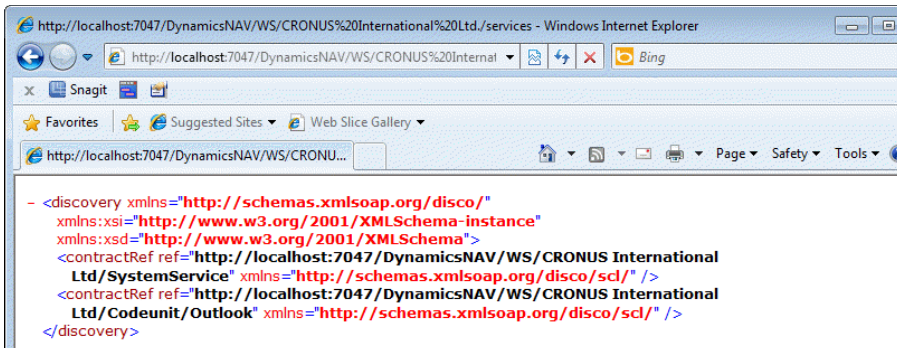
To publish a web service, select the **Object type** and **Object ID** and find a unique service name. Then select the **Published** checkmark.

The only valid web service object in the standard product is Codeunit Outlook Synch. Dispatcher (5313) which is used for Outlook Synchronization which we will discuss later in this chapter.

Consuming a Microsoft Dynamics NAV web service

To consume the web service, an address is generated that is called from the other application.

```
http://<Server>:<WebServicePort>/<ServerInstance>/WS/<CompanyName>/
```


http://localhost:7047/DynamicsNAV/WS/CRONUS%20International%20Ltd./services - Windows Internet Explorer
Snagit
Favorites
Suggested Sites
Web Slice Gallery
Page
Safety
Tools
Bing

```
- <discovery xmlns="http://schemas.xmlsoap.org/disco/"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <contractRef ref="http://localhost:7047/DynamicsNAV/WS/CRONUS International
      Ltd/SystemService" xmlns="http://schemas.xmlsoap.org/disco/scl/" />
    <contractRef ref="http://localhost:7047/DynamicsNAV/WS/CRONUS International
      Ltd/Codeunit/Outlook" xmlns="http://schemas.xmlsoap.org/disco/scl/" />
  </discovery>
```

A very good source of information on web services for Microsoft Dynamics NAV is `http://blogs.msdn.com/freddyk/`.

Client add-ins

The latest interfacing technology for Microsoft Dynamics NAV is called Client add-ins.

This allows the possibility to hand over a page control to a .NET dll. Using this technology requires some basic .NET skills.

Standard application interfaces

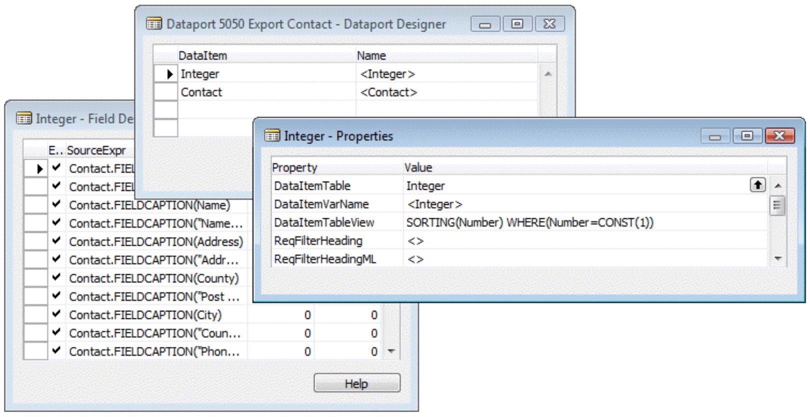
We discussed all the available interface technologies for Microsoft Dynamics NAV. Let's have a look at how this has been implemented in the standard product.

In this book, we will not go into details for each interface since that would almost require another book. We will just briefly discuss where to find all technologies we discussed in the standard application and indicate where a white paper or website can be found.

Exporting Contacts Microsoft Dynamics NAV allows us to export our contacts both from the classic client as from the Role Tailored Client. As both interfaces have a different object type, we will discuss both the Dataport as well as the XMLPort.

Dataport

The Dataport that handles this is Export Contact (5050). The Dataport has two DataItems; Integer and Contact.



Dataport 5050 Export Contact - Dataport Designer

DataItem	Name
Integer	<Integer>
Contact	<Contact>

Integer - Field De

E..	SourceExpr		
✓	Contact.FIEL		
✓	Contact.FIEL		
✓	Contact.FIELDCAPTION(Name)		
✓	Contact.FIELDCAPTION("Name…		
✓	Contact.FIELDCAPTION(Address)		
✓	Contact.FIELDCAPTION("Addr…		
✓	Contact.FIELDCAPTION(County)		
✓	Contact.FIELDCAPTION("Post …		
✓	Contact.FIELDCAPTION(City)	0	0
✓	Contact.FIELDCAPTION("Coun…	0	0
✓	Contact.FIELDCAPTION("Phon…	0	0

Integer - Properties

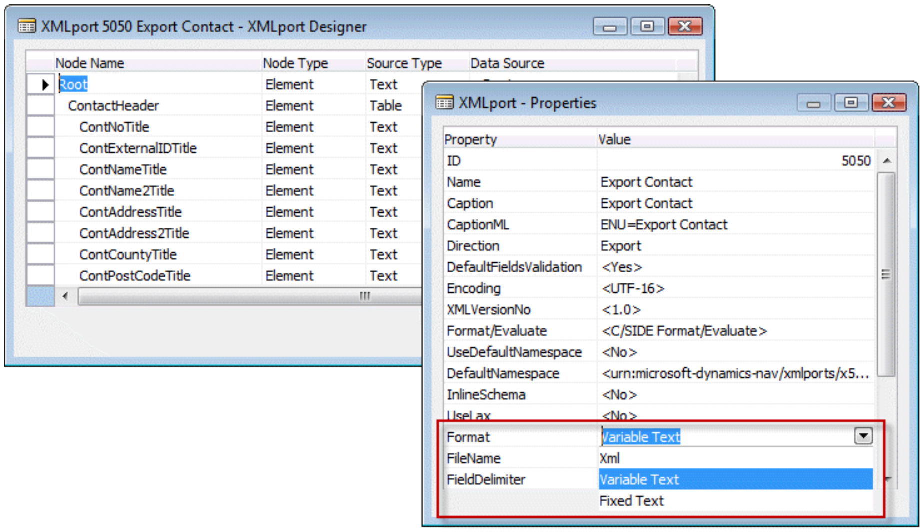
Property	Value
DataItemTable	Integer
DataItemVarName	<Integer>
DataItemTableView	SORTING(Number) WHERE(Number=CONST(1))
ReqFilterHeading	<>
ReqFilterHeadingML	<>

The Integer DataItem is used to export the column headers. The Contact DataItem exports the real information.

Using the virtual Integer table as a DataItem for Dataports and Reports can solve many issues where iteration is required.

XMLPort

The corresponding XMLPort for this functionality has the same number (5050) and uses the Format **Variable Text**. Other options are **XML** and **Fixed Text**.



XMLport 5050 Export Contact - XMLport Designer

Node Name	Node Type	Source Type	Data Source
Root	Element	Text	
ContactHeader	Element	Table	
ContNoTitle	Element	Text	
ContExternalIDTitle	Element	Text	
ContNameTitle	Element	Text	
ContName2Title	Element	Text	
ContAddressTitle	Element	Text	
ContAddress2Title	Element	Text	
ContCountyTitle	Element	Text	
ContPostCodeTitle	Element	Text	

XMLport - Properties

Property	Value
ID	5050
Name	Export Contact
Caption	Export Contact
CaptionML	ENU=Export Contact
Direction	Export
DefaultFieldsValidation	<Yes>
Encoding	<UTF-16>
XMLVersionNo	<1.0>
Format/Evaluate	<C/SIDE Format/Evaluate>
UseDefaultNamespace	<No>
DefaultNamespace	<urn:microsoft-dynamics-nav/xmlports/x5...
InlineSchema	<No>
UseLax	<No>
Format	Variable Text
FileName	
FieldDelimiter	

Where Dataports have a separate window to define the fields, the XMLPorts have a node structure like pages. The XMLPort also starts with Integer table as the first data type followed by the Contact table.

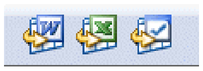
More information about programming Data Ports and XMLPorts can be found in Programming Dynamics NAV by David Studebaker, published by Packt.

Office integration

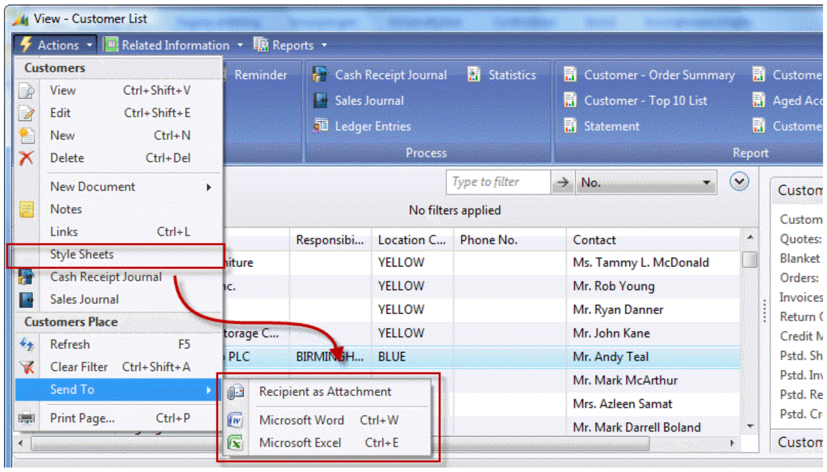
Microsoft Dynamics NAV and Microsoft Office are integrated to use with Word, Excel, and Outlook. We will first discuss the standard Word and Excel integration and later discuss alternatives. Lastly, we will briefly discuss the possibilities for Outlook integration.

Word and Excel integration

In Microsoft Dynamics NAV, each form or page can be exported to Word and Excel using style sheets. This built-in technology is automatically provided by the user interface and requires no effort from developers.



There is a difference in functionality when using the Role Tailored Client compared to the Classic Client. The classic client supports multiple style sheets and executes them using C/AL whilst the Role Tailored Client only supports one style sheet per page object. This is solved in the Style Sheet Tool version 2.0.



The style sheets are standard XML office style sheets.

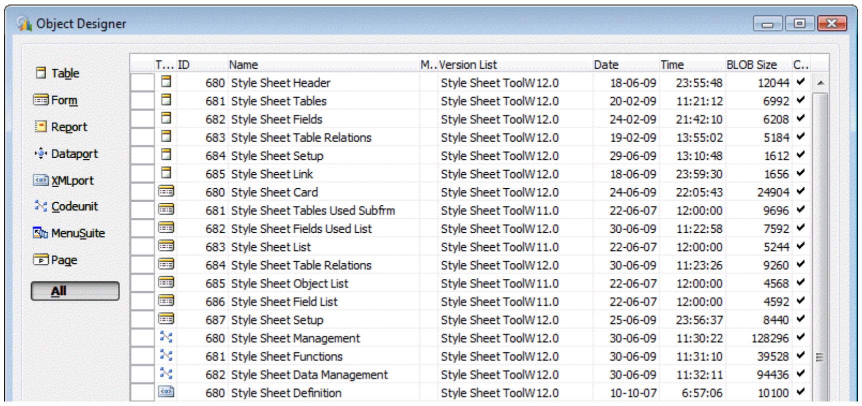
Style sheet tool

Since Office XML style sheet syntax is not very user-friendly, Microsoft has released Style sheet tool for Microsoft Dynamics NAV and Word. This tool allows users to easily generate style sheets.

The Style Sheet tool version 2.0 can be downloaded from this website: `http://www.mibuso.com/dlinfo.asp?FileID=1111`.

The tool can only be started from the Classic Client. After importing the `.fob` file the first step is to create the setup by opening the form `Style Sheet Setup (687)`. When this is done we can start creating style sheets by using form `Style Sheet Card (680)`.

The manual provided with the style sheet tool gives a good description about how to create the style sheets.



T...	ID	Name	M..	Version List	Date	Time	BLOB Size	C..
	680	Style Sheet Header		Style Sheet ToolW12.0	18-06-09	23:55:48	12044	✓
	681	Style Sheet Tables		Style Sheet ToolW12.0	20-02-09	11:21:12	6992	✓
	682	Style Sheet Fields		Style Sheet ToolW12.0	24-02-09	21:42:10	6208	✓
	683	Style Sheet Table Relations		Style Sheet ToolW12.0	19-02-09	13:55:02	5184	✓
	684	Style Sheet Setup		Style Sheet ToolW12.0	29-06-09	13:10:48	1612	✓
	685	Style Sheet Link		Style Sheet ToolW12.0	18-06-09	23:59:30	1656	✓
	680	Style Sheet Card		Style Sheet ToolW12.0	24-06-09	22:05:43	24904	✓
	681	Style Sheet Tables Used Subfrm		Style Sheet ToolW11.0	22-06-07	12:00:00	9696	✓
	682	Style Sheet Fields Used List		Style Sheet ToolW12.0	30-06-09	11:22:58	7592	✓
	683	Style Sheet List		Style Sheet ToolW11.0	22-06-07	12:00:00	5244	✓
	684	Style Sheet Table Relations		Style Sheet ToolW12.0	30-06-09	11:23:26	9260	✓
	685	Style Sheet Object List		Style Sheet ToolW11.0	22-06-07	12:00:00	4568	✓
	686	Style Sheet Field List		Style Sheet ToolW11.0	22-06-07	12:00:00	4592	✓
	687	Style Sheet Setup		Style Sheet ToolW12.0	25-06-09	23:56:37	8440	✓
	680	Style Sheet Management		Style Sheet ToolW12.0	30-06-09	11:30:22	128296	✓
	681	Style Sheet Functions		Style Sheet ToolW12.0	30-06-09	11:31:10	39528	✓
	682	Style Sheet Data Management		Style Sheet ToolW12.0	30-06-09	11:32:11	94436	✓
	680	Style Sheet Definition		Style Sheet ToolW12.0	10-10-07	6:57:06	10100	✓

Using style sheets in the RTC

To enable using the style sheets in the Role Tailored Client, all pages need to be changed. This can be easily done using a tool that comes provided with the style sheet application and requires the following steps:

1. Export the pages you want to use the style sheets for to the new `.xml` file format.
2. Start Codeunit 50000 and select the generated `.xml` file. The tool will generate a new file with the same name adding the letters –SST.
3. Import the `-sst.xml` file and compile the pages.
4. The pages have now the style sheet action.

Word Automation

The built-in style sheet functionality was introduced by Microsoft in version 5.0. Before that, the general solution to integrate with Microsoft Word was to use Automation Control. An example of how this can be done can be found in Codeunit WordManagement (5054).

Advanced Excel integration

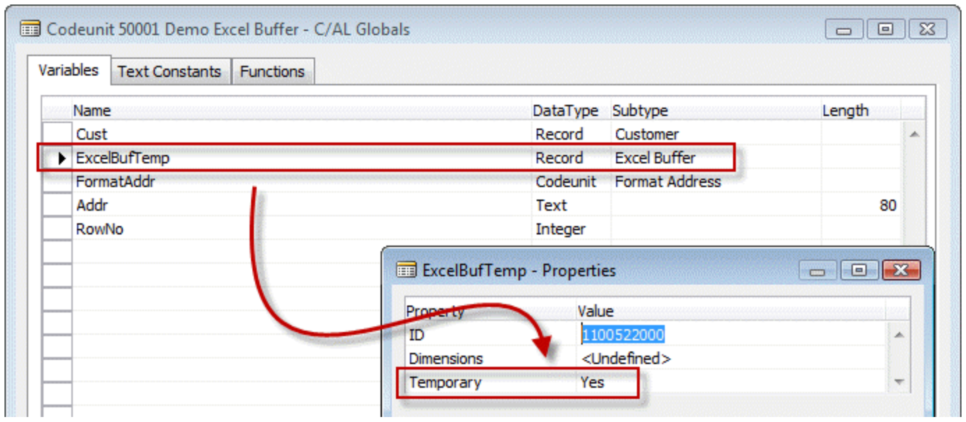
When exporting information to Excel that needs to be combined from different parts of the application, using style sheets is not the ideal way.

To support this, the Excel Buffer table (370) can be used. This table can be populated with data and then sent to Excel using a simple C/AL command.

This is used in several parts of the application, for example, to import and export the budgets we discussed in Chapter 3.

Let's create a sample Codeunit that exports data to Excel using the Excel Buffer table.

1. Create a new Codeunit and define a global variable of type record **Excel Buffer**. This needs to be a temporary variable. Also, define the other variables as displayed in this screenshot:



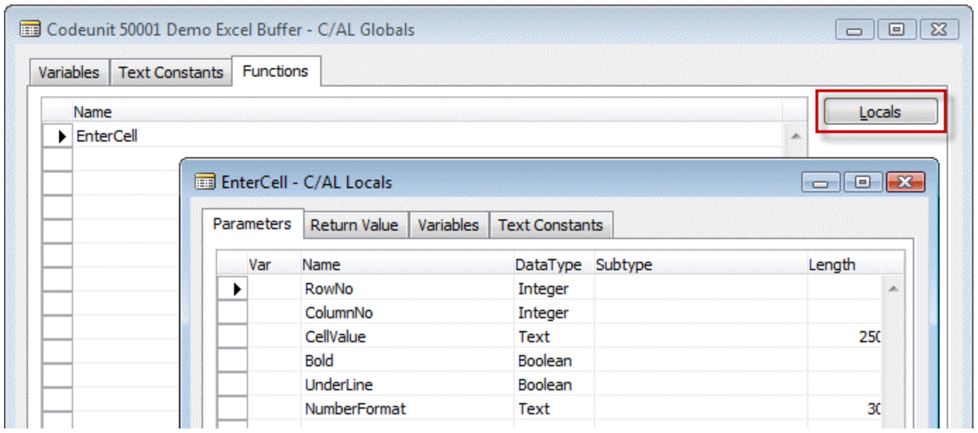
Name	DataType	Subtype	Length
Cust	Record	Customer	
ExcelBufTemp	Record	Excel Buffer	
FormatAddr	Codeunit	Format Address	
Addr	Text		80
RowNo	Integer		

ExcelBufTemp - Properties

Property	Value
ID	1100522000
Dimensions	<Undefined>
Temporary	Yes

Temporary record variables are not stored in the database but in client memory. This allows multiple users to create the same records without blocking each other. It is also faster since all handling is done without the network and database.

2. Create a new Function `EnterCell` with the parameters displayed in this screenshot:



Codeunit 50001 Demo Excel Buffer - C/AL Globals

EnterCell - C/AL Locals

Var	Name	DataType	Subtype	Length
	RowNo	Integer		
	ColumnNo	Integer		
	CellValue	Text		250
	Bold	Boolean		
	UnderLine	Boolean		
	NumberFormat	Text		30

3. Put the C/AL Code in place that will handle the interface.

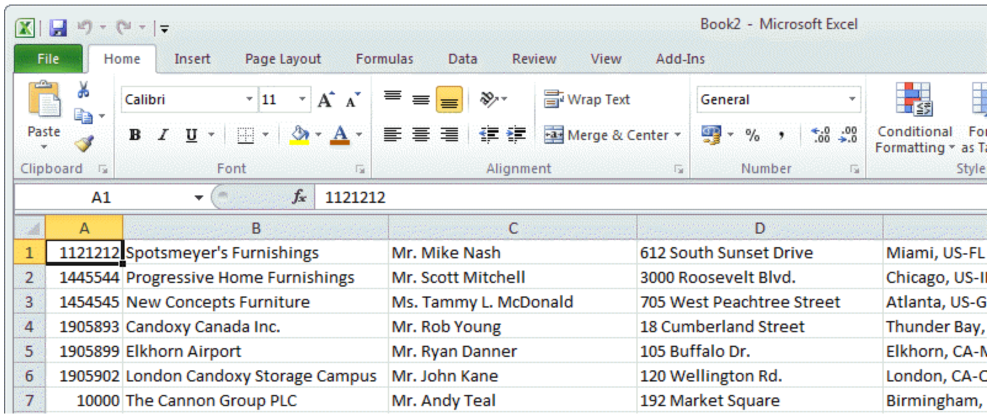
```
OnRun()
ExcelBufTemp.CreateBook;

Cust.FIND('-');
REPEAT
  RowNo := RowNo + 1;
  EnterCell(RowNo, 1, Cust."No.", FALSE, FALSE, '');

  FormatAddr.Customer(Addr, Cust);
  EnterCell(RowNo, 2, Addr[1], FALSE, FALSE, '');
  EnterCell(RowNo, 3, Addr[2], FALSE, FALSE, '');
  EnterCell(RowNo, 4, Addr[3], FALSE, FALSE, '');
  EnterCell(RowNo, 5, Addr[4], FALSE, FALSE, '');
  EnterCell(RowNo, 6, Addr[5], FALSE, FALSE, '');
  EnterCell(RowNo, 7, Addr[6], FALSE, FALSE, '');
  EnterCell(RowNo, 8, Addr[7], FALSE, FALSE, '');
  EnterCell(RowNo, 9, Addr[8], FALSE, FALSE, '');

UNTIL Cust.NEXT = 0;

ExcelBufTemp.CreateSheet(Cust.TABLECAPTION,Cust.TABLECAPTION,
  COMPANYNAME,USERID);
ExcelBufTemp.GiveUserControl;

EnterCell()
ExcelBufTemp.INIT;
ExcelBufTemp.VALIDATE("Row No.",RowNo);
```

```
        ExcelBufTemp.VALIDATE("Column No.",ColumnNo);
        ExcelBufTemp."Cell Value as Text" := CellValue;
        ExcelBufTemp.Formula := '';
        ExcelBufTemp.Bold := Bold;
        ExcelBufTemp.Underline := UnderLine;
        ExcelBufTemp.NumberFormat := NumberFormat;
        ExcelBufTemp.INSERT;
```

This C/AL code will browse the customers in the database and format the addresses using the Address Format (365) Codeunit.

The `Customer No.` field and the result array `Addr[]` are saved in the Excel Buffer table.

Lastly, we start the C/AL functions to generate the Excel spreadsheet based on the data.



A1 1121212

	A	B	C	D	
1	1121212	Spotsmeyer's Furnishings	Mr. Mike Nash	612 South Sunset Drive	Miami, US-FL
2	1445544	Progressive Home Furnishings	Mr. Scott Mitchell	3000 Roosevelt Blvd.	Chicago, US-II
3	1454545	New Concepts Furniture	Ms. Tammy L. McDonald	705 West Peachtree Street	Atlanta, US-G
4	1905893	Candoxy Canada Inc.	Mr. Rob Young	18 Cumberland Street	Thunder Bay,
5	1905899	Elkhorn Airport	Mr. Ryan Danner	105 Buffalo Dr.	Elkhorn, CA-M
6	1905902	London Candoxy Storage Campus	Mr. John Kane	120 Wellington Rd.	London, CA-C
7	10000	The Cannon Group PLC	Mr. Andy Teal	192 Market Square	Birmingham,

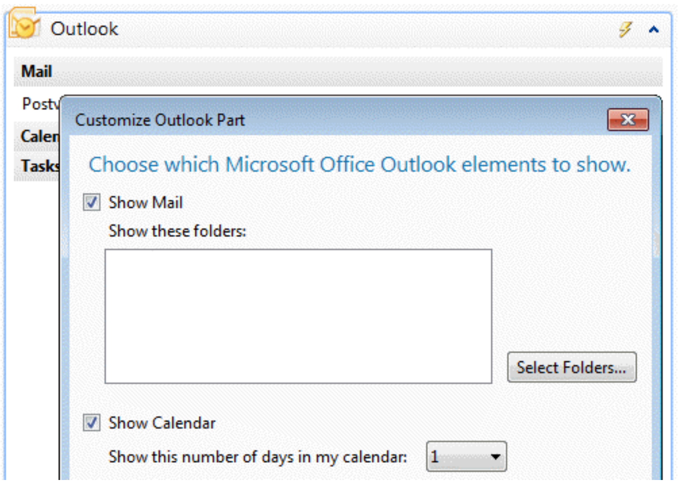
Outlook integration

Microsoft Dynamics NAV 2009 allows different levels of interfacing with Microsoft Outlook. They are as follows:

1. The Outlook Part on the Role Center.
2. Sending e-mails from pages using the `ExtendedDatatype` property.
3. Using the Mail (397) or SMTP Mail (400) Codeunits to send e-mails.
4. Synchronize Contacts and To-Do's using the Outlook integration Web Service.
5. Reading e-mail from Exchange using the E-Mail – Logging functionality.

Outlook part

On a Role Center, it is possible to activate the Outlook System Part. This allows users to see their E-mail, Agenda, and Tasks directly on the Role Center.



Outlook

Mail

Postv

Caler

Tasks

Customize Outlook Part

Choose which Microsoft Office Outlook elements to show.

☑ Show Mail

Show these folders:

Select Folders...

☑ Show Calendar

Show this number of days in my calendar: 1

This functionality is built in the Role Tailored Client and cannot be changed using C/AL Code.

ExtendedDatatype property

When a **Text** field in a table uses the **ExtendedDatatype** property **E-Mail**, the Role Tailored Client will automatically allow the users to directly send an e-mail to the address specified in the field.

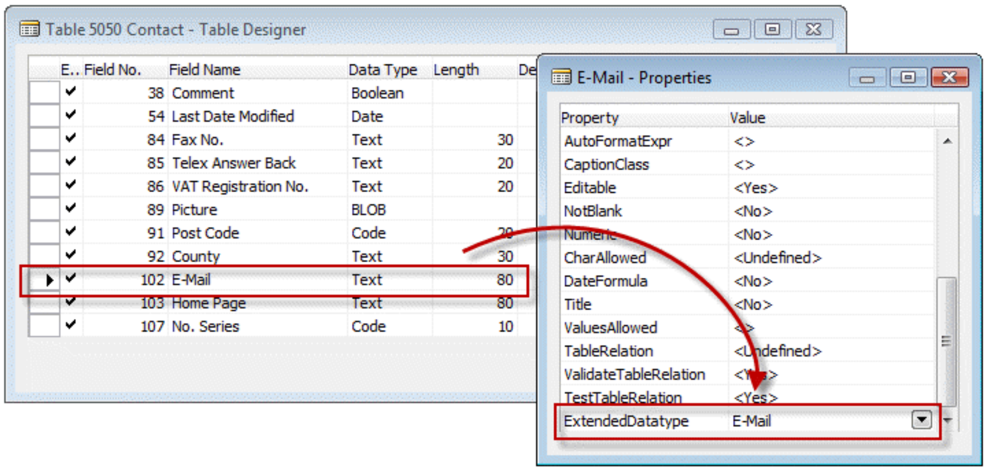


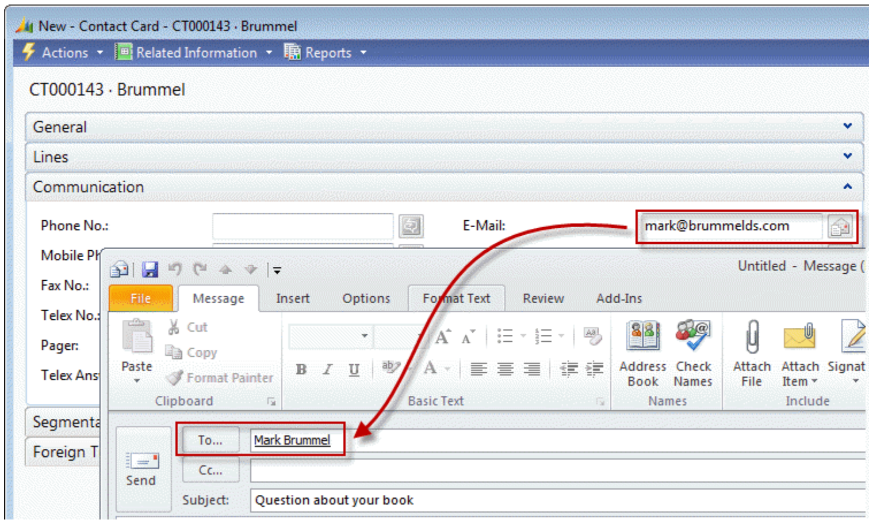
Table 5050 Contact - Table Designer

E..	Field No.	Field Name	Data Type	Length	De
✓	38	Comment	Boolean		
✓	54	Last Date Modified	Date		
✓	84	Fax No.	Text	30	
✓	85	Telex Answer Back	Text	20	
✓	86	VAT Registration No.	Text	20	
✓	89	Picture	BLOB		
✓	91	Post Code	Code	20	
✓	92	County	Text	30	
✓	102	E-Mail	Text	80	
✓	103	Home Page	Text	80	
✓	107	No. Series	Code	10	

E-Mail - Properties

Property	Value
AutoFormatExpr	<>
CaptionClass	<>
Editable	<Yes>
NotBlank	<No>
Numeric	<No>
CharAllowed	<Undefined>
DateFormula	<No>
Title	<No>
ValuesAllowed	<>
TableRelation	<Undefined>
ValidateTableRelation	<Yes>
TestTableRelation	<Yes>
ExtendedDatatype	E-Mail

This is also built in functionality in the Role Tailored Client that cannot be influenced by C/AL code.



mark@brummelds.com

To... Mark Brummel

Subject: Question about your book

Mail and SMTP mail Codeunits

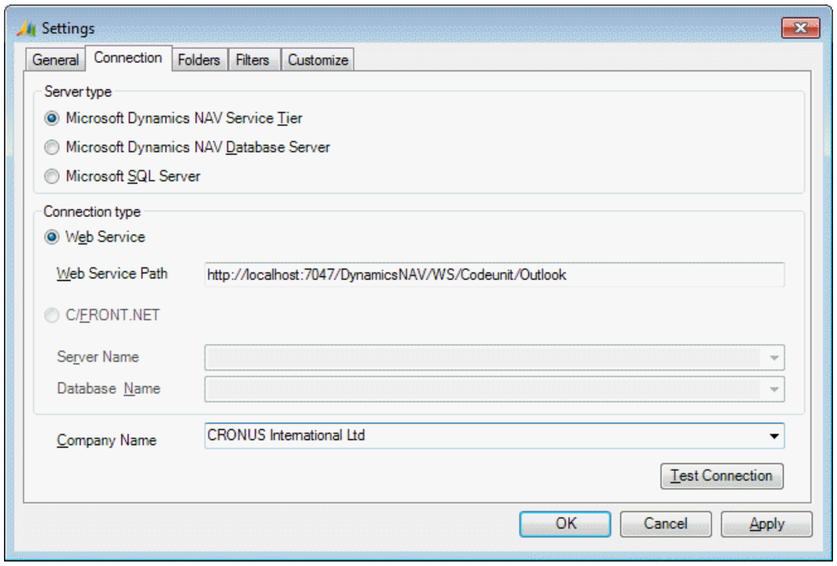
Before the introduction of the `ExtendedDatatype` property, the e-mails from Microsoft Dynamics NAV were sent using an Automation Control wrapper DLL to Microsoft Outlook. This is handled in Codeunit 397 and can still be used to send e-mails directly from C/AL code.

Codeunit SMTP Mail (400) allows us to send e-mails directly to an SMTP server.

Outlook synchronization

Microsoft Outlook can be used as an offline client for Microsoft Dynamics NAV. Every table can be synchronized to Microsoft Outlook when a connection with both systems is available. Using the offline functionality in Outlook, users can view the data when they are on the road and even change the information or create new data.

This is done using the Outlook Synchronization Web Service we discussed earlier in this chapter or in previous versions using a C/FRONT interface.


Settings

General | Connection | Folders | Filters | Customize

Server type
- [✓ Microsoft Dynamics NAV Service Tier]
- Microsoft Dynamics NAV Database Server
- Microsoft SQL Server

Connection type
- [✓ Web Service]

Web Service Path: http://localhost:7047/DynamicsNAV/WS/Codeunit/Outlook

- C/FRONT.NET

Server Name:

Database Name:

Company Name: CRONUS International Ltd

The functionality is very well documented by Microsoft.

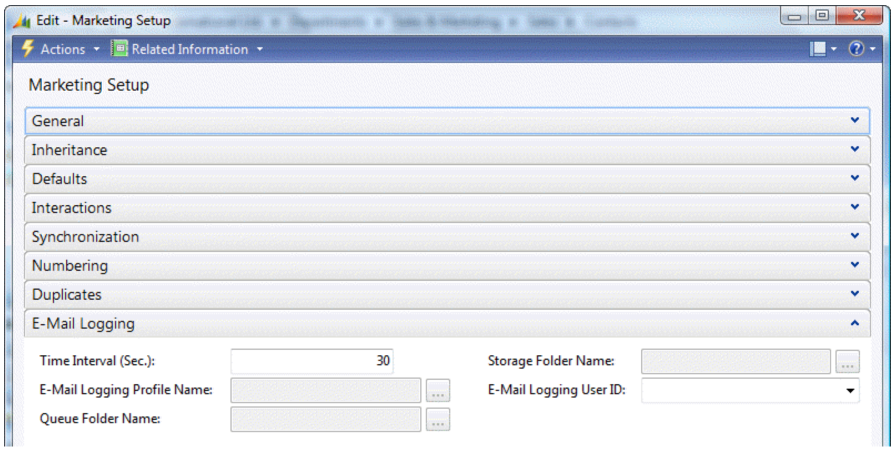
More information can be found on MSDN `http://msdn.microsoft.com/en-us/library/dd338966.aspx` or on Partner Source `https://mbs.microsoft.com/partnersource/deployment/documentation/howtoarticles/howtonavoutlooksynchronization.htm`

Exchange integration

To read incoming e-mails, Microsoft Dynamics NAV offers integration with Exchange Public folders. Information in these mailboxes can be read and used in Microsoft Dynamics NAV.

The handling of the interface is done using the Job Queue and the Application Server (NAS).

In the **Marketing Setup** that we discussed in *Chapter 4, Relationship Management*, we can set up the parameters for the Exchange integration.



Time Interval (Sec.): 30
E-Mail Logging Profile Name:
Queue Folder Name:
Storage Folder Name:
E-Mail Logging User ID:

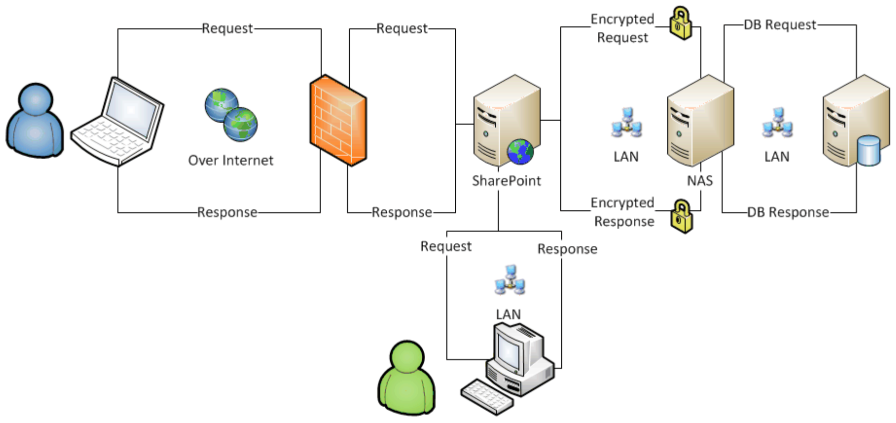
Interaction log entries

Each e-mail read from Microsoft Exchange is displayed in Microsoft Dynamics NAV as an interaction log entry.

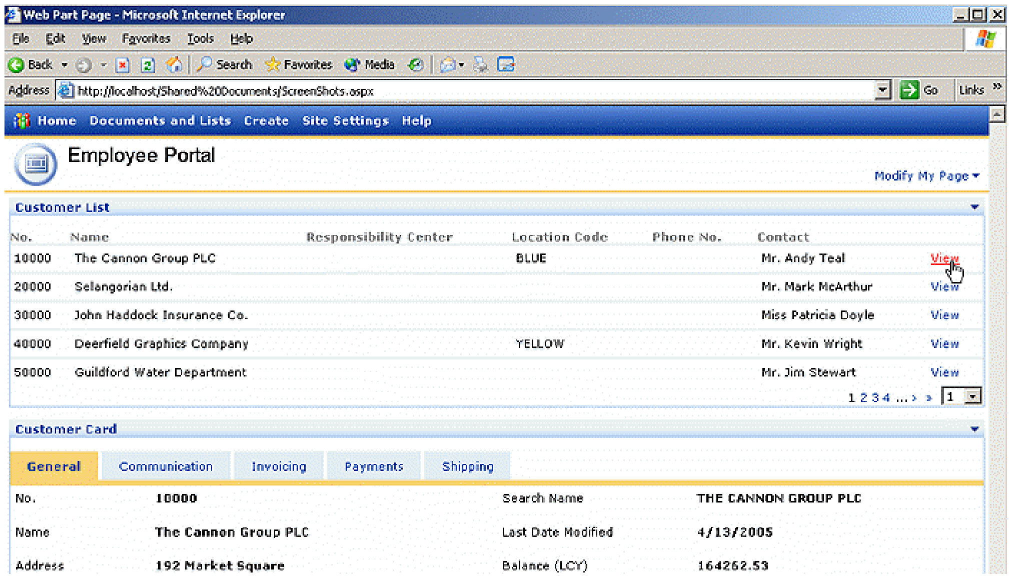
[Since Microsoft Exchange 2010 no longer supports public folders this functionality can no longer be used.]

SharePoint

Microsoft Dynamics NAV is integrated with SharePoint using a product called Employee Portal.



Most forms can be exposed to SharePoint. Using Message Queue and one of more Application Servers, users can read and write data the same way as in the normal user interfaces.


Web Part Page - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost/Shared%20Documents/ScreenShots.aspx
Go Links
Home Documents and Lists Create Site Settings Help
Employee Portal
Modify My Page
Customer List
1 2 3 4 ... › »
Customer Card
General Communication Invoicing Payments Shipping

No.	Name	Responsibility Center	Location Code	Phone No.	Contact	
10000	The Cannon Group PLC		BLUE		Mr. Andy Teal	View
20000	Selangorian Ltd.				Mr. Mark McArthur	View
30000	John Haddock Insurance Co.				Miss Patricia Doyle	View
40000	Deerfield Graphics Company		YELLOW		Mr. Kevin Wright	View
50000	Guildford Water Department				Mr. Jim Stewart	View

No.	10000	Search Name	THE CANNON GROUP PLC
Name	The Cannon Group PLC	Last Date Modified	4/13/2005
Address	192 Market Square	Balance (LCY)	164262.53

More information about Employee Portal can be found on Partner Source: `https://mbs.microsoft.com/partnersource/downloads/releases/employeeportalnav50.htm`

BizTalk

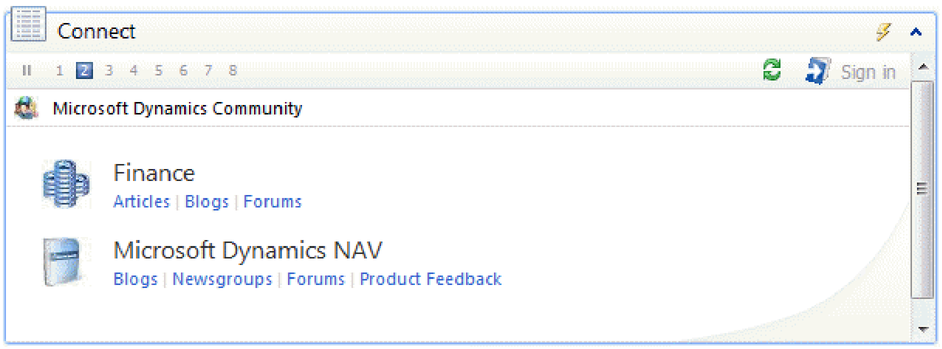
Electronic Data Interchange or **EDI** is the most commonly used standardized interface between companies.

Microsoft Dynamics NAV supports sending and receiving EDI Sales and Purchase orders using Microsoft BizTalk server.

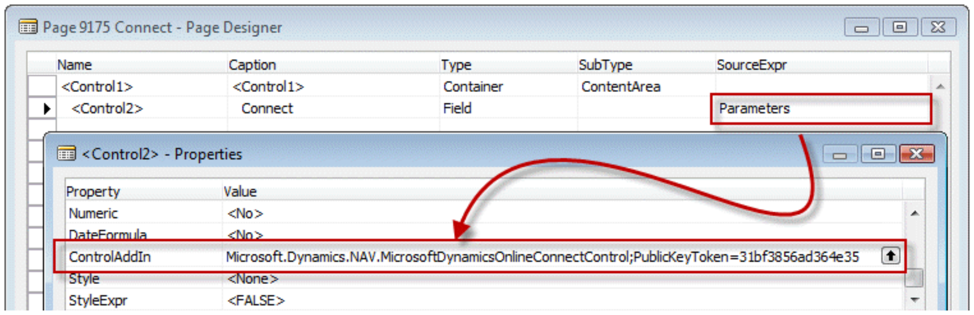
To implement BizTalk requires knowledge of the application and should be done by specialized consultants.

Client Add-ins

Microsoft Dynamics NAV 2009 SP1 ships with one Client add-in for Microsoft Connect. The page object that uses the Connect control is Connect (9175).


Connect
1 2 3 4 5 6 7 8
Sign in
Microsoft Dynamics Community
Finance
Articles | Blogs | Forums
Microsoft Dynamics NAV
Blogs | Newsgroups | Forums | Product Feedback

Let's have a look at how this done:



Name	Caption	Type	SubType	SourceExpr
<Control1>	<Control1>	Container	ContentArea	
<Control2>	Connect	Field		Parameters

Property	Value
Numeric	<No>
DateFormula	<No>
ControlAddIn	Microsoft.Dynamics.NAV.MicrosoftDynamicsOnlineConnectControl;PublicKeyToken=31bf3856ad364e35
Style	<None>
StyleExpr	<FALSE>

The page type of this page is **CardPart** and it has no Source Table. The only control on the page is `Parameters` which is a function with a Text(350) return value.

The property `ControlAddIn` points to the Add-in that will be used when this page is started. This Add-in will replace the original control on the page.

In the `Parameters` function, a string is created to feed information into the Connect add-in enabling it to show information that is interesting for the current role. This is done using a combination of other C/AL functions.

```
Parameters()
InitCurrentRoleValues;

EXIT(Add(Version) + Add(Locale) + Add(Role) + Add(RoleID) +
  Add(Serial));

Add()
```

```
EXIT(Parameter + Separator);
```

Version()

```
EXIT('version=' + FORMAT(ApplicationManagement.ApplicationVersion +
':' +
  ApplicationManagement.ApplicationBuild,0,XMLFormat));
```

Locale()

```
// Windows Language ID
EXIT('locale=' + FORMAT(CurrentLanguageID,0,XMLFormat));
```

Role()

```
// Profile ID (Any text entered in Profile ID)
EXIT('role=' + FORMAT(DELCHR(CurrentRole,'=',Separator),0,
      XMLFormat));
```

RoleID()

```
// Role Center ID (Page ID)
EXIT('roleid=' + FORMAT(CurrentRoleID,0,XMLFormat));
```

Serial()

```
// License ID
EXIT('serial=' + FORMAT(SERIALNUMBER,0,XMLFormat));
```

Separator()

```
EXIT(';');
```

XMLFormat()

```
EXIT(9);
```

InitCurrentRoleValues()

```
CurrentLanguageID := GLOBALLANGUAGE;
CurrentRoleID := ApplicationManagement.DefaultRoleCenter;
CurrentRole := FORMAT(CurrentRoleID);
...
```

Chapter 7, Storage and Logistics, uses Client Extensibility and Bing Maps to show the stops of a route on a map.

More information on using Client Extensibility can be found on `http://blogs.msdn.com/cabeln/`. More information on how to integrate with Bing Maps can be found on `https://blogs.msdn.com/freddyk/archive/2009/03/18/integration-to-virtual-earth-part-1-out-of-4.aspx`

The available libraries are stored in the Client Add-in table (2000000069).

Interface methodologies

So now we discussed interface types, interface technologies and the built in interfaces in Microsoft Dynamics NAV.

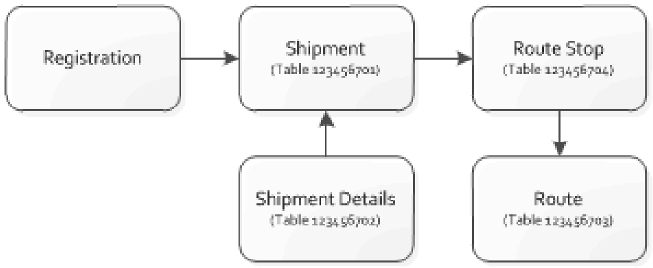
Let's design and develop a new business to business interface. We will use the objects from *Chapter 7, Storage and Logistics* to create the interface.

The scenario

One of our customers wants to e-mail the shipments from now on instead of faxing. The e-mail will contain an Excel file in a predefined format.

The design

Let's bring back the Data Model we designed for the Logistics part of the solution in Chapter 7.



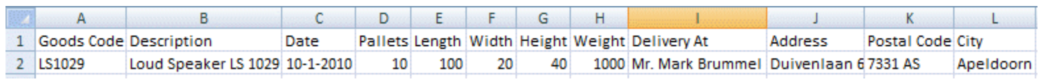
The process starts in the Registration table. From a registration, we generate shipments and shipments are combined into a Route with stops.

So we need to move the data from the Excel sheet to the Registration table.

The mapping

When a customer delivers us an Excel sheet with information, it seldom happens that they exactly use the same fields as our table. Therefore, we need to create a mapping. Each field in the Excel sheet needs to be mapped to a field and missing fields need to be identified and discussed.

The Excel Sheet we get from the customer looks like this.



	A	B	C	D	E	F	G	H	I	J	K	L
1	Goods Code	Description	Date	Pallets	Length	Width	Height	Weight	Delivery At	Address	Postal Code	City
2	LS1029	Loud Speaker LS 1029	10-1-2010	10	100	20	40	1000	Mr. Mark Brummel	Duivenlaan 6	7331 AS	Apeldoorn

Let's try to map this information to our Logistics Registration Worksheet table.

Field No.	Field Name	Data Type	Length	Mapped Field
1	Registration Batch	Code	10	-
2	Line No.	Integer		-
6	Shipment Date	Date		Date
8	Product No.	Code	20	Goods Code
10	Description	Text	50	Description
12	Unit of Measure	Text	10	-
16	Quantity	Decimal		Pallets
20	Length	Decimal		Length
21	Width	Decimal		Width
22	Height	Decimal		Height
31	Gross Weight	Decimal		-
32	Net Weight	Decimal		Weight
36	Units per Parcel	Decimal		-
37	Unit Volume	Decimal		-
53	Ship-to Name	Text	50	Delivery At
55	Ship-to Address	Text	50	Address
57	Ship-to City	Text	30	City
58	Ship-to Contact	Text	50	-
59	Ship-to Post Code	Code	20	Postal Code
60	Ship-to County	Text	30	-
61	Ship-to Country/Region Code	Code	10	-

Most of the fields in the Excel sheet can be mapped to a field in our table.

The gaps

Some fields that are needed in NAV are not populated by the Excel sheet. For some fields this is okay, for example the Registration Batch and Line No. fields are determined by the import.

Some other fields are more difficult. Unit of Measure, Gross Weight, Units per Parcel, and Unit Volume are left blank in the Excel sheet, but they are all needed in NAV.

For these fields, we need to come to an agreement with the customer. They need to either specify these fields or tell us if they have default values. Let's look at our gaps and fill them in:

- **Unit of Measure**: For this customer it is always "PALLET".
- **Volume**: This can be calculated using `Length x Width x Height`.
- **Gross Weight**: We agree that this is equal to Net Weight.
- **Units per Parcel**: This is always 1.

What if it does not work

Reading the external data into the database is just one step in creating a reliable interface.

But what happens if the customer contacts us and says: "We sent you a file with 10 lines and the shipment document shows nine lines." When we check our database, the shipment does show nine lines, but there is no way to check if we imported the original ten lines. At this stage, the imported Registration lines are deleted and the shipments are generated.

If this happens we need traceability.

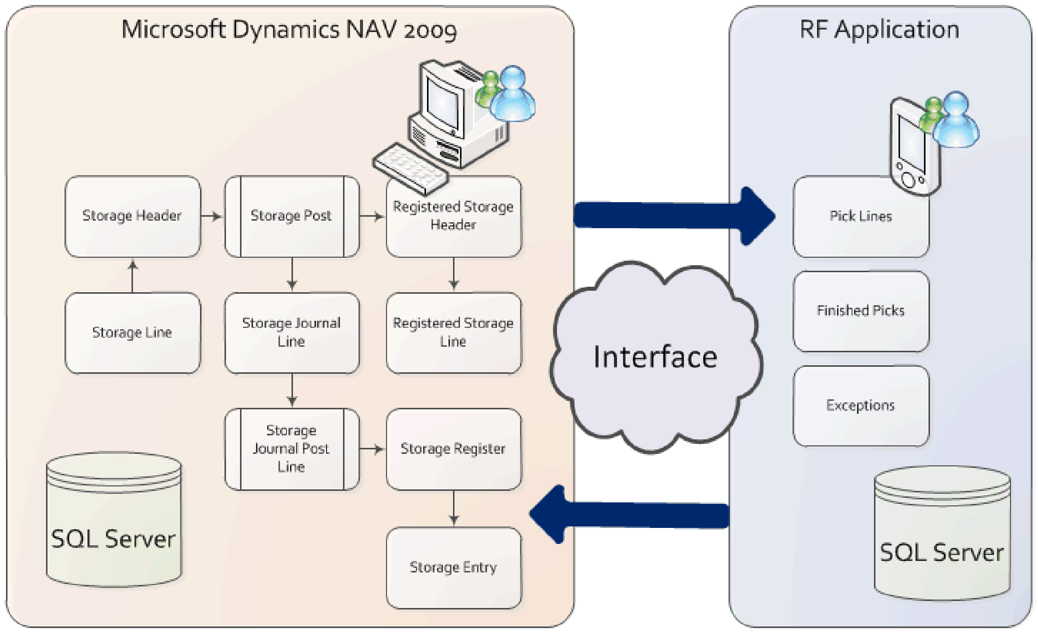
In a well designed interface, we should always create a table that exactly matches the imported data. This allows us to first check if everything matches.

The data from this table can be processed but should not be deleted from the database but periodically cleaned up. This allows us to check if things go wrong.

We will demonstrate this in a more advanced example.

The scenario

The implementation of our storage and logistics add-on requires a real time interface with a Radio Frequency application. The RF scanners are used for the pick process. The RF application uses its own database system with tables that we should populate and read afterwards.



The RF application has three tables. Our interface needs to export data to the **Pick Lines** table, and it needs to import data from the two remaining tables, **Finished Picks** and **Exceptions**.

The interface type

This is an import and export interface that will use data pushing for the Pick Lines and data pulling for the Finished Picks and Exceptions.

The interface will be timer driven. Every minute we will poll for new data.

The interface technology

For this interface, we will use a combination of technologies that we discussed in this chapter.

The main technology is Automation Control. We will use two of them.

Active Data Objects

The Picking database runs on SQL Server, so we will use ADO to connect to the database and send T-SQL Statements to read and write data.

Reliability

For the timer feature we can use two technologies. The obvious choice seems to use the Job Queue to execute our business logic. However, the interface we are creating here can be defined as business critical. If it stops working, the picking process in the warehouse will quickly stop. When designing a business critical interface always remember the more components we use, the more can go wrong. For this reason, it might be better to create our own timer mechanism.

NTimer.dll

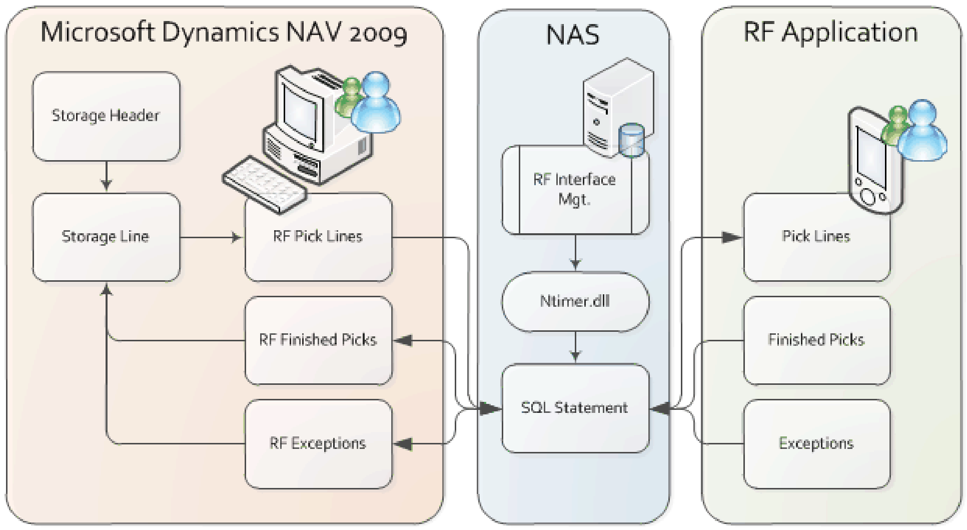
We will use the `ntimer.dll` for this and it comes shipped with each Microsoft Dynamics NAV installation.

Logging

In this interface, we will enable two types of logging. The first log will be to duplicate the RF tables in Microsoft Dynamics NAV and use them as a buffer. A second log will be maintained where we will save a copy of all T-SQL statements we generate. This will enable us to see what we generated if something goes wrong.

The design

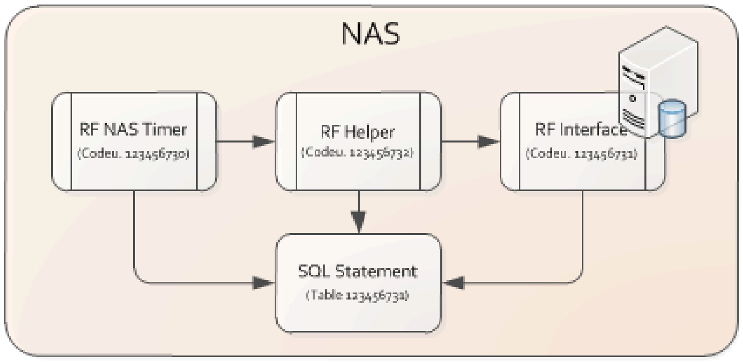
Let's look at the design of the interface we will be developing for this project.



The interface will be controlled from an Application Server. Each minute it will execute a Codeunit that checks if there are new Storage Lines that need to be exported. These lines will first be moved to the RF Pick Lines buffer table and then moved to the RF database using ADO and T-SQL. New Finished Picks and Exceptions from the RF database will be moved to Microsoft Dynamics NAV using the same technology and can then be processed.

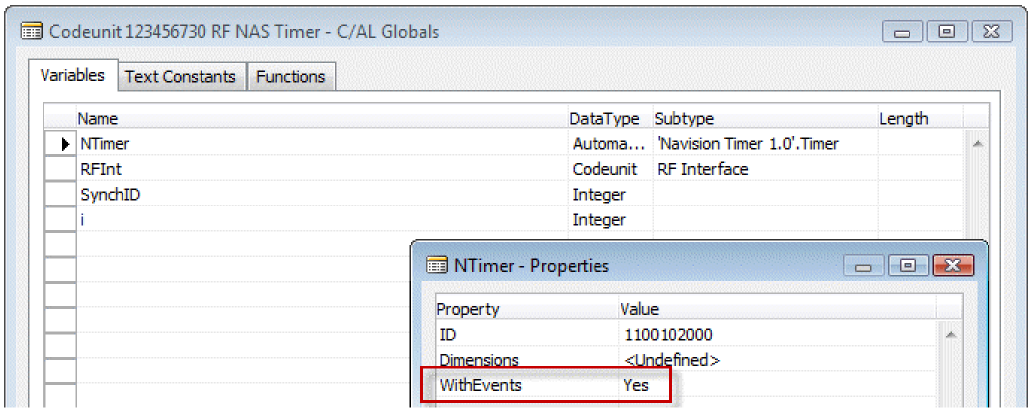
The solution

To run the interface we have created three Codeunits and a table. The **SQL Statement** table is used to log each interface session.


NAS
RF NAS Timer (Codeu. 123456730)
RF Helper (Codeu. 123456732)
RF Interface (Codeu. 123456731)
SQL Statement (Table 123456731)

1. **RF NAS Timer (123.456.730)**

   This Codeunit is started from the `NASHandler` function in Codeunit ApplicationManagement 1. It uses the `ntimer.dll` as an automation control with **Events** and the `SingleInstance` property.


Codeunit 123456730 RF NAS Timer - C/AL Globals

Name	DataType	Subtype	Length
NTimer	Automa...	'Navision Timer 1.0'.Timer	
RFInt	Codeunit	RF Interface	
SynchID	Integer		
i	Integer		

NTimer - Properties

Property	Value
ID	1100102000
Dimensions	<Undefined>
WithEvents	Yes

`Ntimer.dll` is an automation control that can be used to handle timer events in C/AL. The Event `NTimer::Timer` can be used to execute C/AL code.

Enabling events on an Automation Control can be done in the properties of the variable.

```
OnRun()
IF ISCLEAR(NTimer) THEN
  CREATE(NTimer);

NTimer.Interval := 1000;
NTimer.Enabled := TRUE;

RFInt.CreateConnectionString;

ParseError()
SELECTLATESTVERSION;
RFIntSetup.GET;
SynchID := RFIntSetup."Synchronisation ID";

SQLStat.INIT;
SQLStat."SQL Statement 1" := 'ERROR : ' + GETLASTERRORTEXT;

SQLStat.Bold := TRUE;
SQLStat.Color := 16711680;
SQLStat.SessionID := SynchID;
SQLStat.Type := SQLStat.Type::Error;
SQLStat.INSERT(TRUE);
COMMIT;

NTimer::Timer(Milliseconds : Integer)
IF NOT CODEUNIT.RUN(CODEUNIT::"RF Helper") THEN
  ParseError;
```

`SingleInstance` is a property of a Codeunit that changes the behavior variables in the object. Normally all variables are destroyed when the transaction is completed. In `SingleInstance`, Codeunit variables remain active during the lifetime of the application.

This means that the `NTimer::Timer` event will keep coming back during the lifetime of the NAS which is always.

From the `NTimer::Timer` trigger, we execute Codeunit RF Helper. If this Codeunit returns an `ERROR` we save the error in the SQL Statement table. This enables traceability of the interface.

`GETLASTERRORTEXT` is a C/AL function that returns the last error message that was generated by the system. It can be used in combination with `IF CODEUNIT.RUN` syntax to catch runtime errors.

2. **RF Helper (123.456.732)**

   The RF Helper Codeunit is a wrapper Codeunit that is used for error catching and maintaining readability.

   Each run of the interface we create a new SQL Statement ID which we can filter on to trace any errors.

```
OnRun()
SELECTLATESTVERSION;
RFIntSetup.GET;
RFIntSetup."Synchronisation ID" := RFIntSetup."Synchronisation ID"
+ 1;
RFIntSetup.MODIFY;
SynchID := RFIntSetup."Synchronisation ID";

SQLStat.INIT;
SQLStat."SQL Statement 1" :=
  '-SYNCHRONISATION STARTED- ID = ' + FORMAT(SynchID) + ' -';
SQLStat.Bold := TRUE;
SQLStat.Color := 16711935;
SQLStat.SessionID := SynchID;
SQLStat.Type := SQLStat.Type::StartStop;
SQLStat.INSERT(TRUE);

COMMIT;

CLEAR(RFInterface);
RFInterface.SetSynchID(SynchID);

StorageLn.LOCKTABLE;
IF StorageLn.FINDSET THEN REPEAT
  RFInterface.CreatePickLines(StorageLn);
UNTIL StorageLn.NEXT = 0;

COMMIT;

CLEAR(RFInterface);
RFInterface.SetSynchID(SynchID);
RFInterface.ReadFinishedPicks;

COMMIT;

CLEAR(RFInterface);
RFInterface.SetSynchID(SynchID);
RFInterface.ReadExceptions;

COMMIT;
```

```
SQLStat.INIT;
SQLStat."SQL Statement 1" :=
  '-SYNCHRONISATION STOPPED- ID = ' + FORMAT(SynchID) + ' -';
SQLStat.Bold := TRUE;
SQLStat.Color := 16711935;
SQLStat.SessionID := SynchID;
SQLStat.Type := SQLStat.Type::StartStop;
SQLStat.INSERT(TRUE);

COMMIT;
```

Then the three interface functions are triggered to synchronize the three required tables.

COMMIT

After each command we execute the `COMMIT` statement. This will make sure that everything in the database is stored up to that point. This is necessary since the ADO statements we create are outside our transaction. If our interface run would roll back it might synchronize data that is already synchronized.

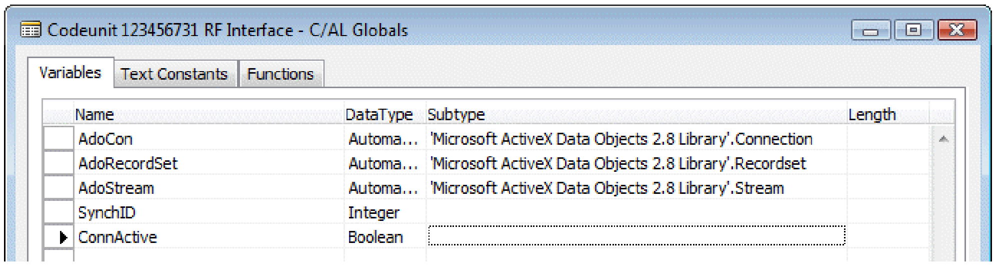
RF Interface (123.456.731)

The actual ADO synchronization is done in this Codeunit. This Codeunit is also `SingleInstance`. This will keep the ADO connection alive during the NAS session.

```
CreateConnectionString()
IF ConnActive THEN EXIT;

IF ISCLEAR(AdoCon) THEN CREATE(AdoCon);
IF ISCLEAR(AdoRecordSet) THEN CREATE(AdoRecordSet);
IF ISCLEAR(AdoStream) THEN CREATE(AdoStream);

RFIntSetup.GET;
Database := RFIntSetup."Database Name";
Server := RFIntSetup."Server Name";

ConnString := 'Provider=SQLOLEDB;Data Source=' + Server +
  ';Initial Catalog=' + Database + ';Trusted_Connection=yes';

SaveReadSQL('Connection ' + ConnString + ' opened on ' +
  FORMAT(CURRENTDATETIME),TRUE,0,0,0, '');

AdoCon.ConnectionString := ConnString;
AdoCon.Open;

ConnActive := TRUE;

CloseConnectionString()
AdoCon.Close;
```

```
CLEAR(AdoCon);
CLEAR(AdoStream);
CLEAR(AdoRecordSet);
SaveReadSQL('Connection closed on ' +
  FORMAT(CURRENTDATETIME),TRUE,0,0,1, '');
ConnActive := FALSE;
```

For the interface we use three Automation Variables.


Codeunit 123456731 RF Interface - C/AL Globals

Name	DataType	Subtype	Length
AdoCon	Automa...	'Microsoft ActiveX Data Objects 2.8 Library'.Connection	
AdoRecordSet	Automa...	'Microsoft ActiveX Data Objects 2.8 Library'.Recordset	
AdoStream	Automa...	'Microsoft ActiveX Data Objects 2.8 Library'.Stream	
SynchID	Integer		
ConnActive	Boolean		

- ADO.Connection: This is used for the connection with the database and to execute the T-SQL statements.
- ADO.Recordset: The result sets of a SELECT statement can be read using this.
- ADO.Stream: The streams are used to convert data types between ODBC and C/Side.

Writing data

The RF application needs data from the Storage Line table. We first create a mapping to the RF application as we did with the Excel interface earlier in this chapter.

This mapping is saved in a buffer table for traceability.

```
CreatePickLines()
CreateConnectionString;
SaveReadSQL('CreatePickLines',TRUE, 1, 8388608, 3, '');
PickID := COPYSTR(StorageLn."Document No." + FORMAT(StorageLn."Line
No."), 1, 20);
SaveReadSQL('Pick Document : '+PickID,TRUE,3,16711680,7,'');
WITH RFPickLines DO BEGIN
  "Pick Code" := PickID;
  Quantity := StorageLn.Quantity;
  "Terminal ID" := 1;
  "Display 1" := StorageLn.Description;
  "Display 2" := 'Warehouse ' + StorageLn."Warehouse Code";
```

```
    "Display 3" := 'Region ' + StorageLn."Region Code";
    "Display 4" := 'Shelf ' + StorageLn."Shelf No.";
    INSERT;

    SQLStatement := 'INSERT INTO [RF Pick Lines]' +
                    '([Pick Code],'+
                     '[Quantity],'+
                     '[Terminal ID],'+
                     '[Display 1],'+
                     '[Display 2],'+
                     '[Display 3],'+
                     '[Display 4])'+
                     'VALUES('+
                     Quote + PickID + Quote      +','+
                     FORMAT(Quantity)            +','+
                     '1'                         +','+
                     Quote + "Display 1" + Quote +','+
                     Quote + "Display 2" + Quote +','+
                     Quote + "Display 3" + Quote +','+
                     Quote + "Display 4" + Quote + ')';
  END;

  ExecuteSQL(SQLStatement);

  StorageLn.Exported := CURRENTDATETIME;
  StorageLn.MODIFY;
```

The actual data is moved to the RF database using an `INSERT` command.

> To avoid exporting the same data twice we need to keep track of what we exported. The simplest way to do this is to create a new field called Exported. Making this field a **DateTime** also enables the traceability of the application.

Reading data

When reading data from the RF database, we also send a T-SQL `SELECT` query for the data. We use the `ADORecordSet` to browse through the records that are in the result set.

For each record in the result set, we create a record in our buffer table which we can later use to update the information in the Storage Lines.

When reading data, we do not want to import the same data twice. To avoid this, we need to store a unique identifier in a table that enables us to remember where we left in the last run.

```
ReadFinishedPicks()
CreateConnectionString;

SaveReadSQL('ReadFinishedPicks',TRUE, 1, 8388608, 3, '');

RFIntSetup.GET;
LastSync := RFIntSetup."Last Finished Pick";

AdoRecordSet := AdoCon.Execute(
  SaveReadSQL('SELECT ' +
              '[Reference Entry No],' +
              '[Terminal ID],' +
              '[Duration],' +
              '[Ready Date Time]' +
    ' FROM [RF Finished Pick] WHERE [Reference Entry No] > ' +
      LastSync,FALSE,2,0,2, ''));

IF AdoRecordSet.BOF AND AdoRecordSet.EOF THEN
  EXIT;

AdoRecordSet.MoveFirst;
REPEAT
  RFFinishedPick.INIT;
  RFFinishedPick."Reference Entry No." :=
    ReadInteger('Reference Entry No');
  RFFinishedPick."Terminal ID" := ReadInteger('Terminal ID');
  RFFinishedPick.Duration := ReadInteger('Duration');
  RFFinishedPick."Ready Date Time" :=
    ReadDateTime('Ready Date Time');
  RFFinishedPick.INSERT;
  AdoRecordSet.MoveNext;
UNTIL AdoRecordSet.EOF;

RFIntSetup."Last Finished Pick" := Quote + FORMAT(RFFinishedPick."Refe
rence Entry No.") + Quote;
RFIntSetup.MODIFY;

AdoRecordSet.Close;
```

In our example, this unique identifier is the `Reference Entry No`.

Log, log, and log more

Although much of the logging is done using the buffer tables, we also want to store the general process of the interface each time it runs. This is done using the SQL Statement table. Both the SQL Statements as well as the other events are stored there.

By using the `COMMIT` functionality we can exactly see where it stopped by looking at the last record in this table. We can solve the problem that caused the interface to stop and restart the interface without losing data.

[Never use the `COMMIT` statement unless there is a very good reason for it. C/SIDE will normally handle the transactions for you by enabling a full role back when things go wrong. Creating a `COMMIT` in a normal C/SIDE transaction will prevent C/SIDE from rolling back.]

Testing

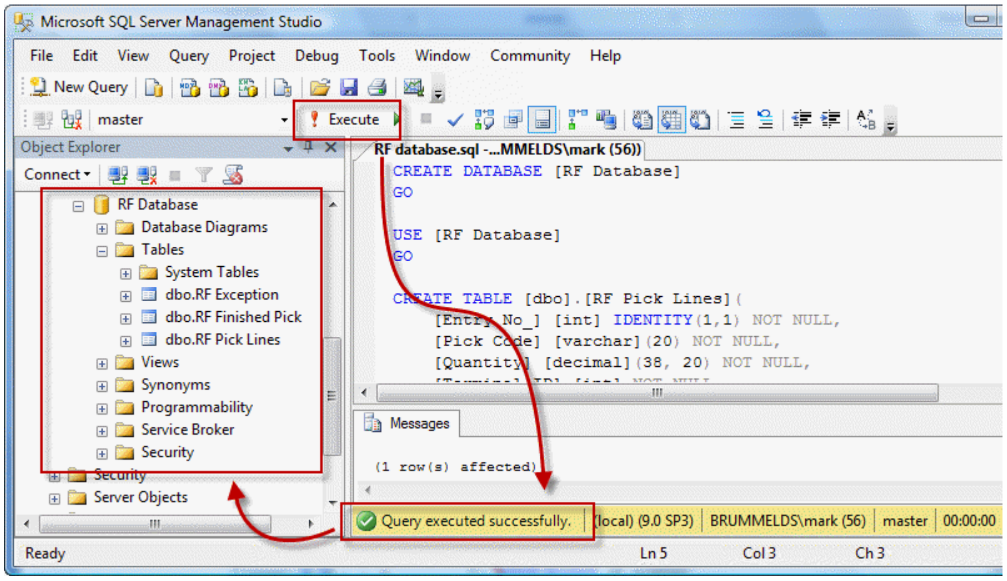
Let's test the interface we have just designed and developed. In order to do this, we need to have records in the Storage Line table and the RF database needs to exist somewhere.

The RF database

To test the objects we have created for this solution, the RF database should exist on your system. This database can be created using a T-SQL script and should be executed on a Microsoft SQL Server machine.

[The script **RF database.sql** is part of the object files downloaded for this book.]

Open the script in SQL Server Management Studio and push **Execute**.



```
CREATE DATABASE [RF Database]
GO

USE [RF Database]
GO

CREATE TABLE [dbo].[RF Pick Lines](
    [Entry No_] [int] IDENTITY(1,1) NOT NULL,
    [Pick Code] [varchar](20) NOT NULL,
    [Quantity] [decimal](38, 20) NOT NULL,
```

```
(1 row(s) affected)
```

Query executed successfully. | (local) (9.0 SP3) | BRUMMELDS\mark (56) | master | 00:00:00

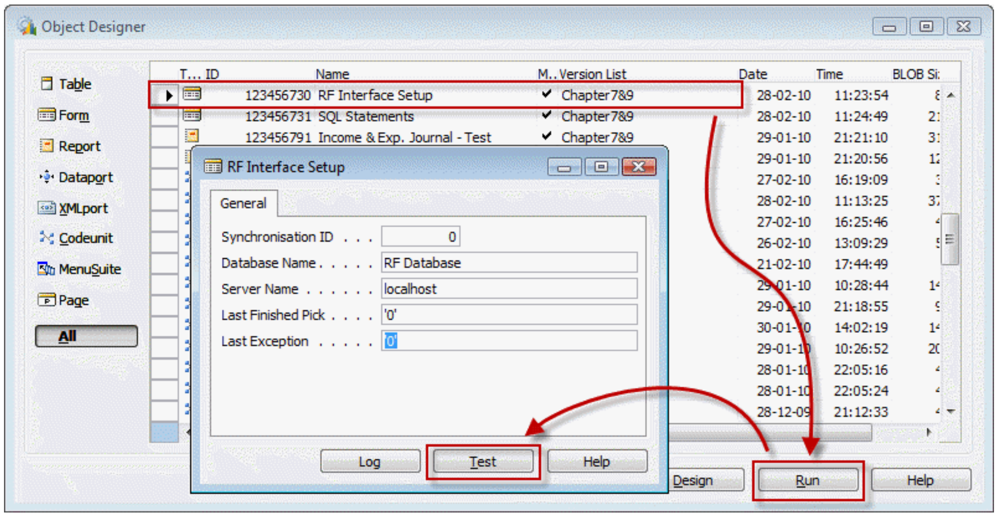
The test

Even though the C/AL code could run in the Role Tailored Client, we will run the test in the Classic Client. The reason for this is that the interface will run in the NAS which will execute the C/AL code the same way as the Classic Client. Another reason to use the Classic Client is that this is the interface for the DBA to perform all their tasks.

To start a test run, open form **RF Interface Setup** (123.456.730) from the Object Designer.

Make sure that the **Database** and **Server** are correct. The server should be the SQL Server instance where the SQL Script was executed.

The ADO connection uses the current Windows Account with Trusted connection. This user should have enough rights to insert and read data from the RF database.



T... ID	Name	M.. Version List	Date	Time	BLOB Si:
123456730	RF Interface Setup	✓ Chapter7&9	28-02-10	11:23:54	8
123456731	SQL Statements	✓ Chapter7&9	28-02-10	11:24:49	21
123456791	Income & Exp. Journal - Test	✓ Chapter7&9	29-01-10	21:21:10	31
			29-01-10	21:20:56	12
			27-02-10	16:19:09	3
			28-02-10	11:13:25	37
			27-02-10	16:25:46	4
			26-02-10	13:09:29	5
			21-02-10	17:44:49	
			29-01-10	10:28:44	14
			29-01-10	21:18:55	9
			30-01-10	14:02:19	14
			29-01-10	10:26:52	20
			28-01-10	22:05:16	4
			28-01-10	22:05:24	4
			28-12-09	21:12:33	4

RF Interface Setup

General

Synchronisation ID . . .	0
Database Name	RF Database
Server Name	localhost
Last Finished Pick	'0'
Last Exception	'0'

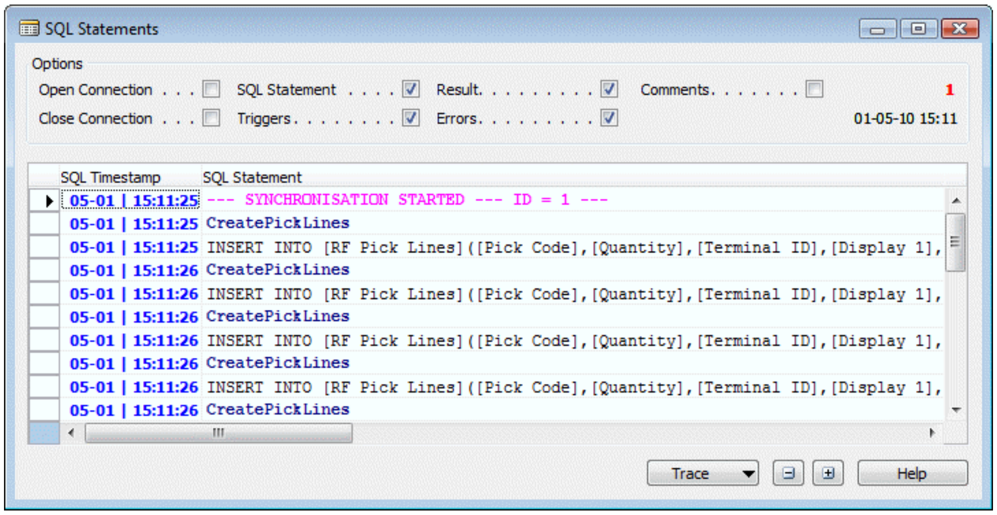
To start a Test run, push the **Test** button.

Viewing the results

If everything went well, the results should show both in the log and in the buffer tables and the RF Database. Let's check them all.

SQL statements

The SQL Statement log can be opened by either pushing the **Log** button on the RF Interface Setup form or opening form **SQL Statements** (123.456.731) from the Object Designer.


SQL Statements

Options

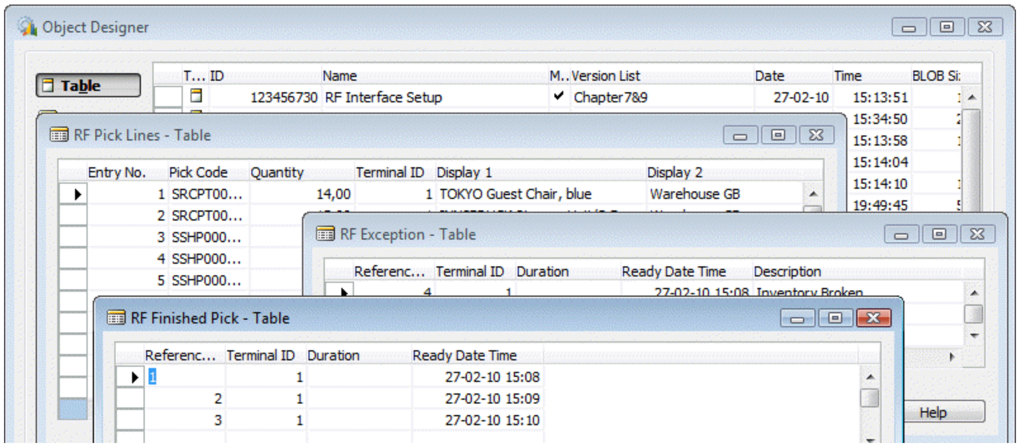
Open Connection	☐	SQL Statement	☑	Result.	☑	Comments.	☐	1
Close Connection	☐	Triggers.	☑	Errors.	☑			01-05-10 15:11

```
SQL Timestamp       SQL Statement
05-01 | 15:11:25    --- SYNCHRONISATION STARTED --- ID = 1 ---
05-01 | 15:11:25    CreatePickLines
05-01 | 15:11:25    INSERT INTO [RF Pick Lines]([Pick Code],[Quantity],[Terminal ID],[Display 1],
05-01 | 15:11:26    CreatePickLines
05-01 | 15:11:26    INSERT INTO [RF Pick Lines]([Pick Code],[Quantity],[Terminal ID],[Display 1],
05-01 | 15:11:26    CreatePickLines
05-01 | 15:11:26    INSERT INTO [RF Pick Lines]([Pick Code],[Quantity],[Terminal ID],[Display 1],
05-01 | 15:11:26    CreatePickLines
05-01 | 15:11:26    INSERT INTO [RF Pick Lines]([Pick Code],[Quantity],[Terminal ID],[Display 1],
05-01 | 15:11:26    CreatePickLines
```

The information on the form shows us exactly what the interface did during this run.

The buffer tables

When we open the buffer tables from the object designer, we can see that the interface moved the data from the Storage Line table into the RF Pick Lines table.


Object Designer

T...	ID	Name	M..	Version List	Date	Time	BLOB Si
	123456730	RF Interface Setup	✓	Chapter7&9	27-02-10	15:13:51	
						15:34:50	
						15:13:58	
						15:14:04	
						15:14:10	
						19:49:45	

RF Pick Lines - Table

Entry No.	Pick Code	Quantity	Terminal ID	Display 1	Display 2
1	SRCPT00...	14,00	1	TOKYO Guest Chair, blue	Warehouse GB
2	SRCPT00...				
3	SSHP000...				
4	SSHP000...				
5	SSHP000...				

RF Exception - Table

Referenc...	Terminal ID	Duration	Ready Date Time	Description
4	1		27-02-10 15:08	Inventory Broken

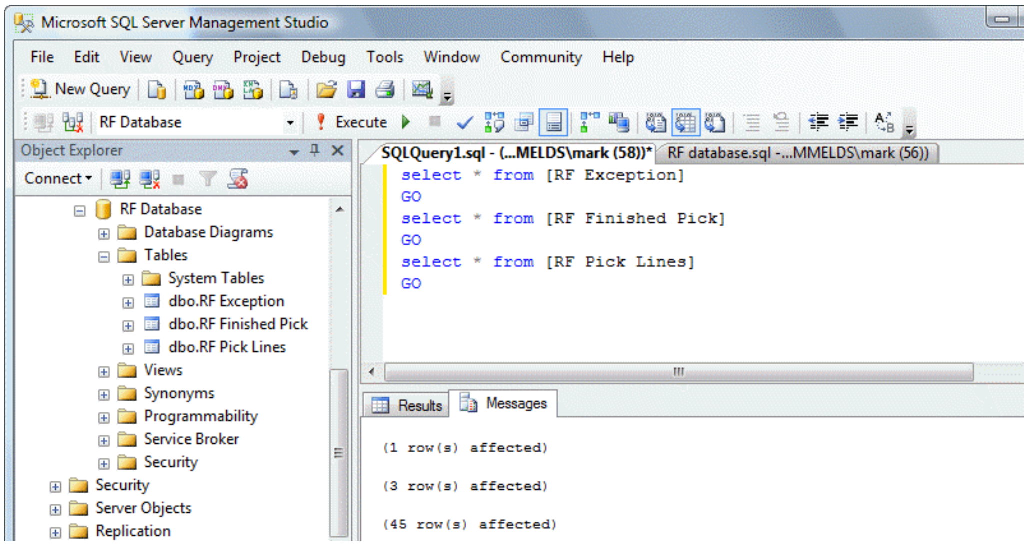
RF Finished Pick - Table

Referenc...	Terminal ID	Duration	Ready Date Time
1	1		27-02-10 15:08
2	1		27-02-10 15:09
3	1		27-02-10 15:10

The **RF Finished Pick** and **RF Exceptions** are also populated with the records from the RF Database.

The RF database

Last thing to check is the data in the RF Database. The data in both databases should now be exactly the same.



```
select * from [RF Exception]
GO
select * from [RF Finished Pick]
GO
select * from [RF Pick Lines]
GO
```

```
(1 row(s) affected)

(3 row(s) affected)

(45 row(s) affected)
```

This can be checked from the SQL Server Management Studio.

Interfacing into the future

Interfacing will become more and more important in the future as technology evolves. Newer technologies and faster internet connections will allow us to integrate our applications better but will also make it more accessible for end users.

Let's briefly discuss some of the integrations we will see in the near future with Microsoft Dynamics NAV.

SharePoint client in Microsoft Dynamics NAV "7"

The next major release of Microsoft Dynamics NAV will be shipped with a new SharePoint Client that will replace the Employee Portal product discussed in this chapter.

The new product will allow us to reuse the Pages we have already defined for the Role Tailored Client and Web services, and connects to the Service Tier.

Microsoft Dynamics CRM

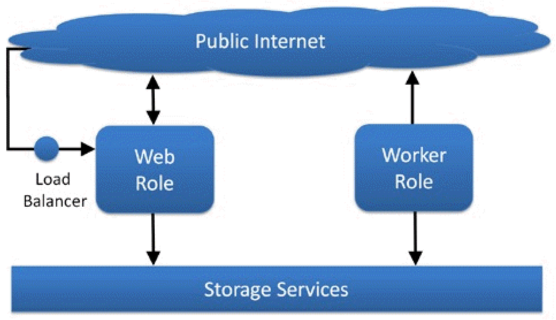
Another important new interface will be with Microsoft Dynamics CRM. This will allow us to use the strong integration that Microsoft Dynamics CRM has with Microsoft Outlook and use the analysis and remote possibilities of this product.

Windows Azure

Cloud Computing is the latest technology where companies like Microsoft and Google are really pushing us into, so what does that mean for Microsoft Dynamics NAV and interfacing.

We discussed earlier in this chapter that using Web Services requires a real time connection between the systems using Windows Domain Credentials.

Windows Azure allows us to implement a remote mechanism that will solve these issues. We can publish information to the cloud and users can subscribe to this information.



More information on Windows Azure can be found on this website:
`http://geekswithblogs.net/claeyskurt/Default.aspx`

Summary

In this chapter, we have looked at how Microsoft Dynamics NAV can interface with other applications.

We discussed the basics of interfacing, import versus export, and data pulling versus data pushing. An interface can be executed manually or by a timer or event.

Microsoft Dynamics NAV supports a wide range of interfacing technologies such as Files, Automation Control, OCX, .NET, ODBC, ADO, and Web Services.

It is also possible to integrate using SQL Server technologies.

The Application Server (NAS) is often used for interfacing with other systems for example using Microsoft Message Queuing or Active Data Objects (ADO).

A wide range of interfaces that comes with the product have been discussed including all interfaces with Microsoft Office, Exchange, SharePoint, and BizTalk.

We designed and developed two business to business interfaces. One is to import data manually from Microsoft Excel and the other to automatically import and export data to another database using ADO and a timer.

When designing an interface, reliability and traceability are the key elements.

In the next chapter, we will talk about Application Design methodologies and principles.

10
Application Design

In Microsoft Dynamics NAV, technology and functionality go hand in hand. It is impossible to design an effective change or enhancement to the application without thorough knowledge of how the standard pieces fit together. With this knowledge now available we can start designing our own applications.

In this book we talked about Application Design for Microsoft Dynamics NAV. We discussed the data and the transaction model; how it works and why it works that way.

We designed several small and large changes to the system both in detailed examples and on a conceptual level.

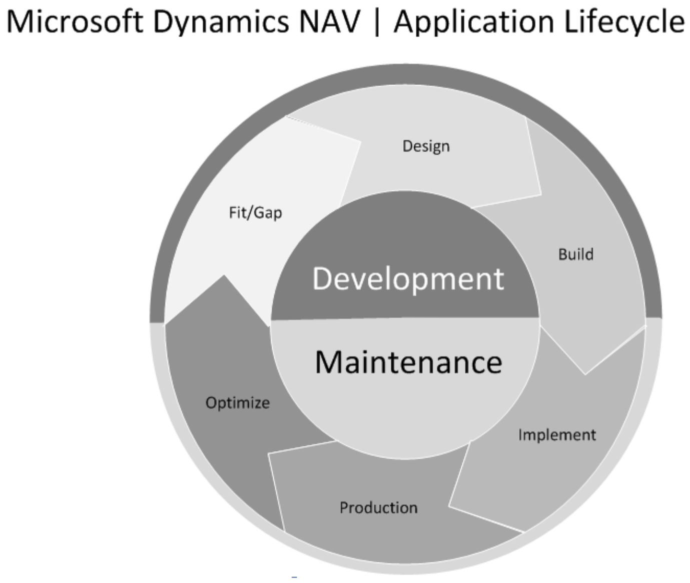
In this chapter we will fit together all the pieces we have learned in this book, and turn them into concepts for good application design.

We will also discuss how to approach a Microsoft Dynamics NAV implementation project, and how to maintain the application. This requires a different approach depending on the level of customization the project contains.

Application lifecycle

Designing an application is more than just analyzing processes and developing new objects. These phases are just the tip of the iceberg.

Once your application has been designed and developed it is most likely that one or more companies will start using it. When this happens your software will start a new phase in its lifecycle. Let's have a look at the lifecycle of a Microsoft Dynamics NAV application.



The **Development** phase of the application starts with the **Fit/Gap** analysis, followed by the **Design** and **Build** steps we saw in the earlier chapters of this book. When those steps have been completed, the maintenance phase of your application will start.

The **Maintenance** phase starts with the implementation and taking the software into production. The first time this will happen it will be the Microsoft Dynamics NAV implementation in your company. Once this is done your system will enter the real lifecycle where constant improvements will be made to the application.

With the flexibility of the Microsoft Dynamics NAV product, this is a special procedure where it is easy to step into the many pitfalls there are along the way.

We will discuss some guidelines that are important to follow. There are six categories: Design to use, Maintain, Support, Upgrade, Perform, and Analyze.

Design to use

Designing software is not a goal; it's a way to support companies doing their business. This makes usability one of the most important focus areas when designing your application.

The first thing that pops into mind when talking about usability is the user interface. Microsoft Dynamics NAV 2009 has two interfaces that are commonly used, the Classic Client and the new Role Tailored Client.

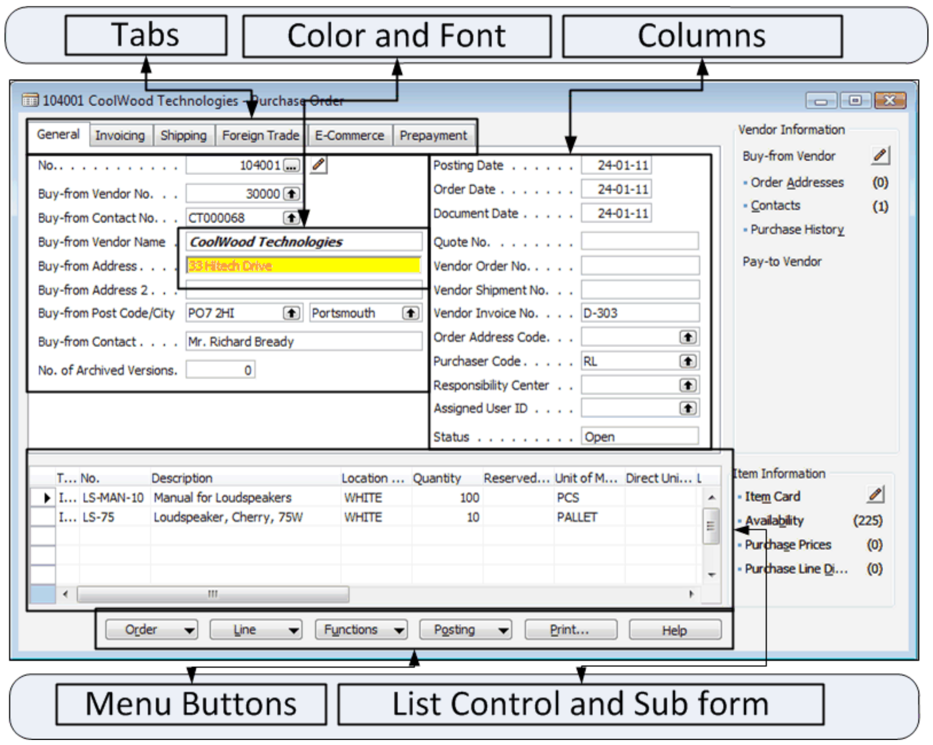
Both interfaces have a very different approach, where the Classic Client is very flexible in design using a limited number of WYSIWYG (What You See Is What You Get) controls, and the Role Tailored Client for Microsoft Dynamics NAV 2009 is less flexible rendering the page definition at runtime.

We will discuss both object types, focusing on the issues which require the most design effort.

Forms

The form object is available for backwards compatibility with the Classic Client and Add-on solutions that are not yet transformed to the Page objects.

- **Tabs**: Forms are tabbed horizontally. This means thaty only the fields on the first tab are visible by default. It is tempting to move important fields for an implementation to the first tab, and this might increase productivity which is good. However, there is a downside to this, which we will discuss when talking about upgrades.



104001 CoolWood Technologies - Purchase Order

No.	104001
Buy-from Vendor No.	30000
Buy-from Contact No.	CT000068
Buy-from Vendor Name	CoolWood Technologies
Buy-from Address	33 Hitech Drive
Buy-from Address 2	
Buy-from Post Code/City	PO7 2HI Portsmouth
Buy-from Contact	Mr. Richard Bready
No. of Archived Versions	0
Posting Date	24-01-11
Order Date	24-01-11
Document Date	24-01-11
Quote No.	
Vendor Order No.	
Vendor Shipment No.	
Vendor Invoice No.	D-303
Order Address Code	
Purchaser Code	RL
Responsibility Center	
Assigned User ID	
Status	Open

Vendor Information
Buy-from Vendor
Order Addresses (0)
Contacts (1)
Purchase History
Pay-to Vendor

T...	No.	Description	Location ...	Quantity	Reserved...	Unit of M...	Direct Uni...
I...	LS-MAN-10	Manual for Loudspeakers	WHITE	100		PCS	
I...	LS-75	Loudspeaker, Cherry, 75W	WHITE	10		PALLET	

Item Information
Item Card
Availability (225)
Purchase Prices (0)
Purchase Line Di... (0)

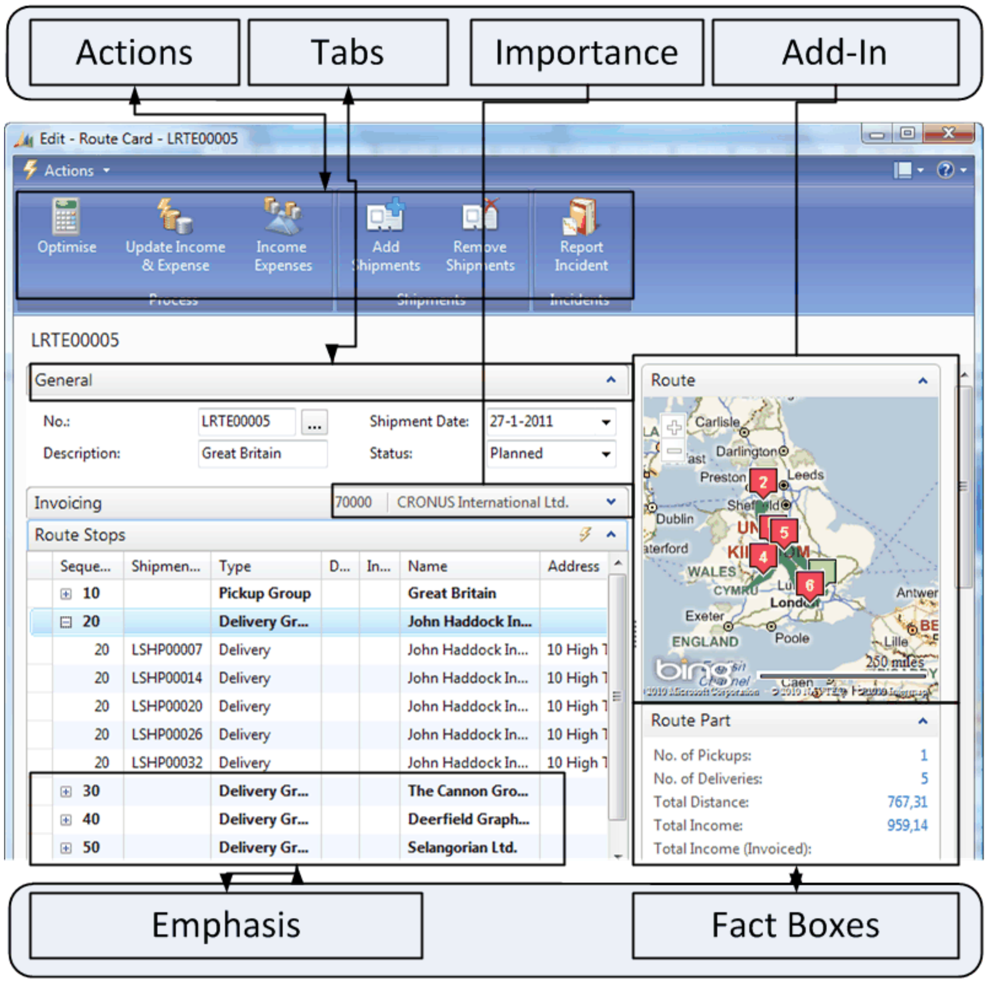
- **Columns**: By default most forms have two columns, where the first is wider and the latter is narrower. This is mainly for screen sizing. A third column is often added, to allow us to display more information on the first tab. This is not a bad thing. Try to keep the columns vertically aligned. This will improve the readability of the form.
- **Fields**: Most field types have a standard font, font size, width, and height when they are placed on a form. Try to maintain them as much as possible. This will improve the similarity of the forms throughout the application. Exceptions are forms for Point of Sale, Shop floor, and Warehouse employees. In these cases the users often have only one or few screens they use throughout the day, and they won't see the rest of the application.
- **Colors**: Colors are something all users love. It makes it easier to manage exceptions and make decisions quickly. Be careful with implementing too much color, this will have a counter effect and lead to confusion.
- **Menu Buttons**: By default the menu buttons of many standard application forms give access to all possible application features supported. Most companies do not use them all and the end users have even less need for them. This makes it tempting to go and change them. However this will only have a very short term productivity effect. After a while most users will have learned where to click. A better improvement for productivity is teaching end users to use the available shortcuts.
- **List Controls**: The list controls are the most flexible controls in the classic client. Users can change the visibility and size of the columns themselves.
- **Sub Forms**: Most document forms have one sub form displaying the lines. Sometimes there might be a need to add more than one sub form. This should be done only as a last resort. Having more than one sub form is an indication of possible bad design. Exceptions to this are forms that are used to analyze data by managers. For those forms it might be very useful to have all the information in one place.

Pages

Where forms are very flexible in design, pages are fixed. However they have a lot of advantages over forms that make them easier to use, although they require slightly more work to design.

- **Tabs**: Pages have vertical tabs that can be opened at the same time making it less desirable to move fields to the first tab.
- **Embedded Lists**: Another advantage of pages is that the users always gets to see a embedded list page first, and then continues to the card that opens in a new Windows control after selecting a record. This is something that many developers have done in the classic client as a modification.

- **Importance**: On pages it is possible to promote controls to be displayed when the tab is closed or made additional so the end users have to specifically make them visible. Use this functionality carefully when designing your application.
- **Personalization**: All pages can be personalized by end users, also card pages. This makes it easier to customize pages during an implementation for a company, department, or end user. Personalization does not change the object definition and does not require a developer.



LRTE00005

General

No.:	LRTE00005	Shipment Date:	27-1-2011
Description:	Great Britain	Status:	Planned

Invoicing 70000 CRONUS International Ltd.

Route Stops

Seque...	Shipmen...	Type	D...	In...	Name	Address
10		Pickup Group			Great Britain	
20		Delivery Gr...			John Haddock In...	
20	LSHP00007	Delivery			John Haddock In...	10 High T
20	LSHP00014	Delivery			John Haddock In...	10 High T
20	LSHP00020	Delivery			John Haddock In...	10 High T
20	LSHP00026	Delivery			John Haddock In...	10 High T
20	LSHP00032	Delivery			John Haddock In...	10 High T
30		Delivery Gr...			The Cannon Gro...	
40		Delivery Gr...			Deerfield Graph...	
50		Delivery Gr...			Selangorian Ltd.	

No. of Pickups:	1
No. of Deliveries:	5
Total Distance:	767,31
Total Income:	959,14
Total Income (Invoiced):	

- **Actions**: On pages Actions have replaced the menu buttons used on Forms. Users can select which actions they want to emphasize making it easier for them to get started with the application.

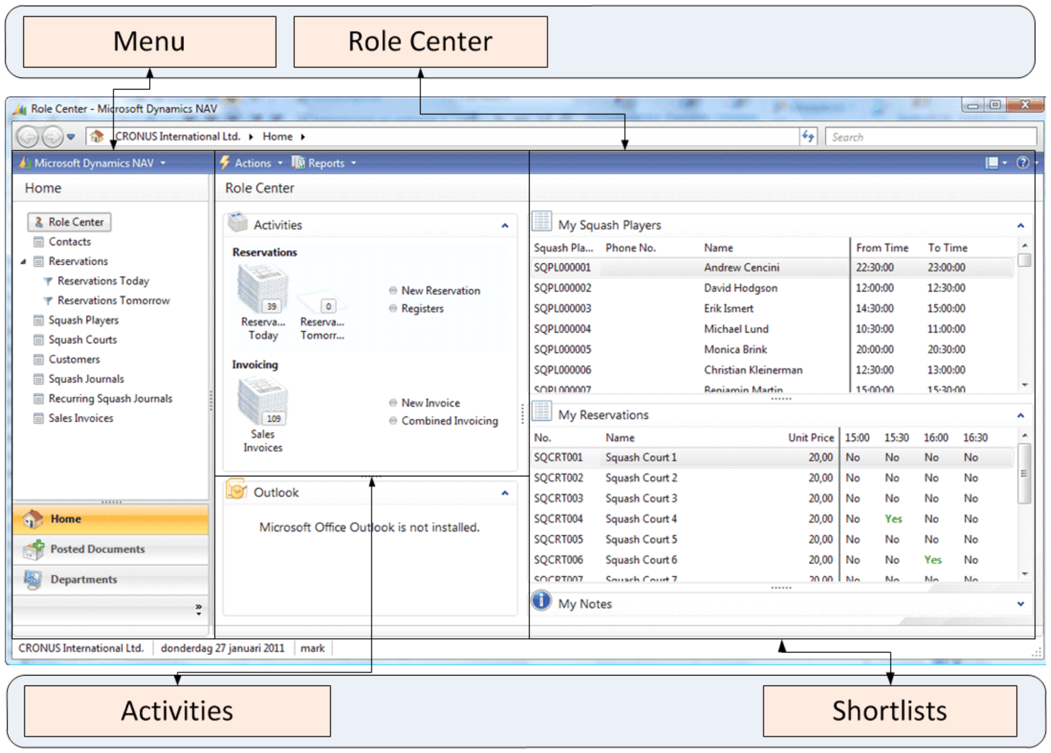
- **Fact Boxes**: This is a new feature which was not available in the classic client. Each page can have an unlimited number of fact boxes attached. Fact boxes can be used to show detailed information about a record. The Route page in *Chapter 7*, *Storage and Logistics*, is a good example where we can see the route in Bing maps and the details of the stops.
- **Emphasis**: A control on a page can be emphasized with Bold, Red, Green, or Red Italics. A combination of these colors in one control is not possible.
- **Client extensibility**: A control on a page can be taken over by a `.net dll`. The `.net` control will use the content of the `dll` and render the information. We have discussed Client extensibility in *Chapter 9, Interfacing*.
- **Web Services**: All pages can be exposed as a Web Service. This makes it possible to create your own user interface in Visual Studio, Borland Delphi, or another development tool that can consume Web Services.

Role centers

When it comes to usability in Microsoft Dynamics NAV 2009, the Role Centers are the heart of the application. The Role Center is the place where the end users starts their working day and returns to regularly. Let's discuss the Role Centers we created in this book.

Squash application

The Squash Court Role Center was created for *Chapter 2*, *A Sample Application*, and looks like this:



My Squash Players

Squash Pla...	Phone No.	Name	From Time	To Time
SQPL000001		Andrew Cencini	22:30:00	23:00:00
SQPL000002		David Hodgson	12:00:00	12:30:00
SQPL000003		Erik Ismert	14:30:00	15:00:00
SQPL000004		Michael Lund	10:30:00	11:00:00
SQPL000005		Monica Brink	20:00:00	20:30:00
SQPL000006		Christian Kleinerman	12:30:00	13:00:00
SQPL000007		Benjamin Martin	15:00:00	15:30:00

My Reservations

No.	Name	Unit Price	15:00	15:30	16:00	16:30
SQCRT001	Squash Court 1	20,00	No	No	No	No
SQCRT002	Squash Court 2	20,00	No	No	No	No
SQCRT003	Squash Court 3	20,00	No	No	No	No
SQCRT004	Squash Court 4	20,00	No	Yes	No	No
SQCRT005	Squash Court 5	20,00	No	No	No	No
SQCRT006	Squash Court 6	20,00	No	No	Yes	No
SQCRT007	Squash Court 7	20,00	No	No	No	No

The Application screen has two sections, the Menu section and the Role Center section.

The Menu section is created by merging Actions from the Role Center with the main menu. When clicking on **Departments** an end user can access the entire application depending on the security setup.

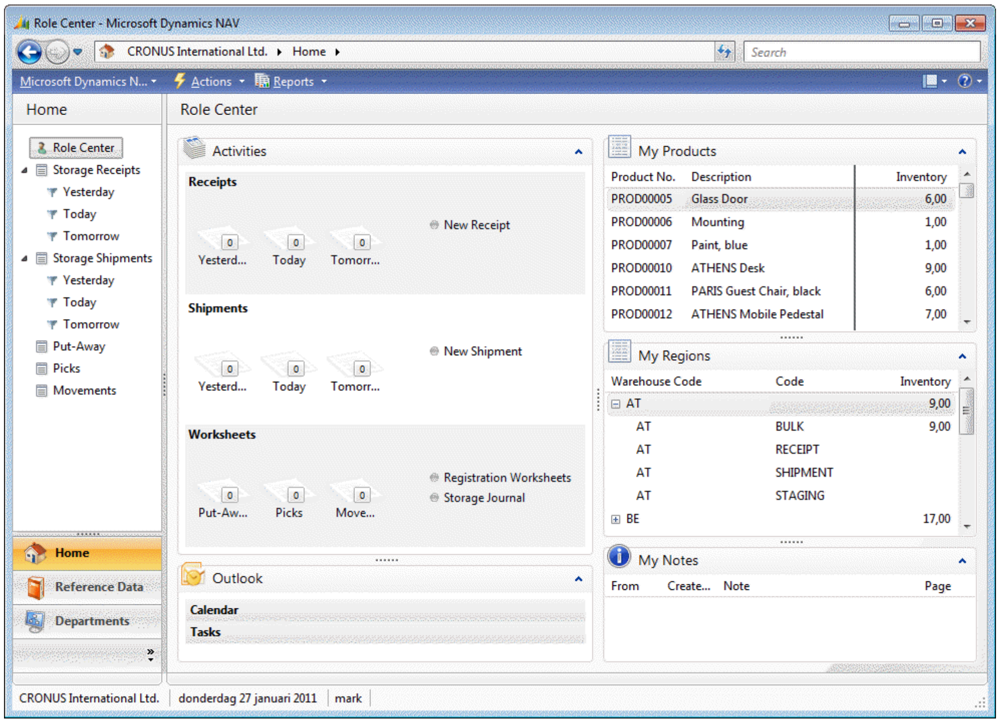
> More information about the **Home** items can be found on this blog: `http://dynamicsuser.net/blogs/mark_brummel/archive/2009/12/16/tip-26-grouping-in-the-homeitems.aspx`.

The Role Center has a left and right part. The left part usually contains the activities and a shortcut to Microsoft Outlook. The right part contains shortlists to the **My List** pages that show frequently used records and notes. An end user can customize the Role Center and move the parts around.

A Role Center can also show graphs. Graphs are defined using an XML structure which is stored in the Chart table (2000000078).

Storage and logistics

This application has four different Role Centers. We will discuss the Storage Role Center (123.456.726). Other Role Centers are Logistics Role Center (123.456.700), Manager Log. and St. Role Center (123.456.756), and Income and Expenses Role Center (123.456.761).

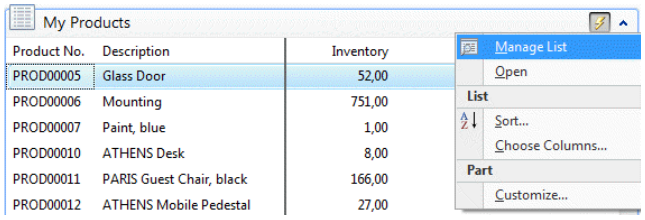


Product No.	Description	Inventory
PROD00005	Glass Door	6,00
PROD00006	Mounting	1,00
PROD00007	Paint, blue	1,00
PROD00010	ATHENS Desk	9,00
PROD00011	PARIS Guest Chair, black	6,00
PROD00012	ATHENS Mobile Pedestal	7,00

Warehouse Code	Code	Inventory
⊟ AT		9,00
AT	BULK	9,00
AT	RECEIPT	
AT	SHIPMENT	
AT	STAGING	
⊞ BE		17,00

On the Activities storage page employees can directly go to the documents filtered on dates from the stacks. From the menu options users can create new documents or open worksheets and journals.

We have designed two shortlist pages, **My Products** and **My Regions**. My Products can be changed by the user by clicking on the small lightning button, and select **Manage List**.



Product No.	Description	Inventory
PROD00005	Glass Door	52,00
PROD00006	Mounting	751,00
PROD00007	Paint, blue	1,00
PROD00010	ATHENS Desk	8,00
PROD00011	PARIS Guest Chair, black	166,00
PROD00012	ATHENS Mobile Pedestal	27,00

The My Region page is built on the Region table. Users cannot change this list. The page uses the `SourceTableTemporary` and `ShowAsTree` properties. This allows users to expand and collapse warehouses.

Reports

The reports in the standard Microsoft Dynamics NAV application are typical ERP reports that show the required information and that's it.

Designing reports requires special skills and is not as easy as it seems. When changing a report layout from the standard application it is best practice to leave the original report as it is, and modify the saved copy.

We will discuss more of reports in the section *Design to analyze*.

Design to maintain

It seldom happens that software is designed and developed, never to be changed. Objects are usually changed many times in the lifetime of the application.

The changes to an existing object may be done quite a while after the object's original development. At this time, even if the changes are done by the original developer, it will be difficult to remember how and why some choices were made.

This is why it is important to develop in a unified way. This will make it easier for developers to read each other's code or to understand their own code after months or years.

Written external documentation is a no brainer at this point but we should realize that this is not always done, and focus on more obvious and easier ways of explaining our code. A well designed and built application should be self documenting. This is done by following some simple guidelines.

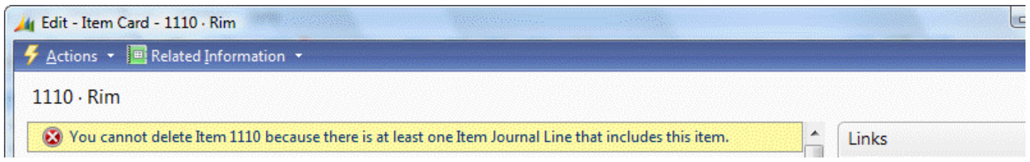
Naming

When creating new objects it is important to follow the naming guidelines of the product. Field and variable names should explain themselves.

Singular and plural

Table names should be singular. This will make the C/AL command `TABLECAPTION` return a usable value. Let's look at an example in the Item table (27).

```
OnDelete()
...
ItemJnlLine.SETRANGE("Item No.","No.");
IF ItemJnlLine.FIND('-') THEN
  ERROR(Text023,TABLECAPTION,"No.",ItemJnlLine.TABLECAPTION);
```



You cannot delete Item 1110 because there is at least one Item Journal Line that includes this item.

List forms and pages should be plural as they contain more than one record, whilst card forms and pages are singular.

Reserved words

Reserved words should not be used in objects such as name for fields, variables, and functions.

> Microsoft has published a list of reserved words on: `http://msdn.microsoft.com/en-us/library/ee414230.aspx`.

One very important Reserved word, which is missing in that list is Action. This is reserved for using `IF Page.RUNMODAL = ACTION::OK then`.

Names and abbreviations

Using standard naming and abbreviations has been one of the strong points of the application that makes it easy to learn for new developers.

Some examples:

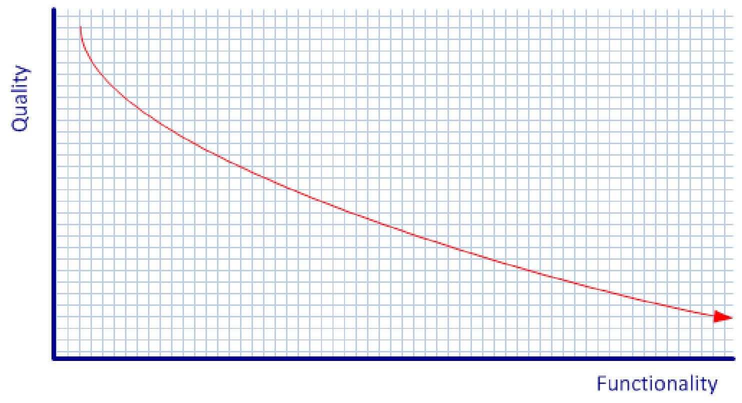
- **<<Table name>> No.**: This is the standard reference to field in a table relation. If the field has a relation with the Customer the field is called Customer No., and if the relation is with Vendor we use Vendor No. In our example application we have used Product No., Squash Player No., and so on.
- **Line No.**: This fieldname is always used in the popular Header/Line and Journal constructions. This field always uses the auto split key property in forms and pages.
- **Entry No.**: This fieldname is always used for entry and register tables such as G/L Entry and Customer Ledger Entry.

- **Name and Description**: Standard naming for persons or products.
- **Quantity/Qty.**: The standard name and abbreviation for measuring quantity.
- **(LCY)**: Abbreviation for Local Currency.
- **Duty Due %**: When the field represents a % this should be in the field name.

A list of naming conventions can be found on MSDN: `http://msdn.microsoft.com/en-us/library/ee414213.aspx`.

Quantity versus quality

There is a general rule that can be applied to the quantity and quality of software that states that, when more functionality is added to a product it is difficult to maintain a certain level of quality.



To avoid this in your solution, make sure to not just add all requirements from your prospects into the product in one release, but use a release policy that ensures small pieces of functionality are developed, tested, and implemented each time.

The framework concept

When developing an add-on product it is important to divide it into smaller parts. This will make it easier to have several developers work on the application and release parts of the application.

Each part of the add-on has its own framework that interacts with other pieces of the add-on or the standard product.

This is exactly what we have done in the Storage and Logistics example add-on application. The add-on has three main functional areas; Storage, Logistics, and Income and Expenses. These three areas share the same master data.

Each area interacts with other parts of the application using mini interfaces.

Using this concept will also have great benefits for upgrading to newer versions which we will discuss in the section *Design to upgrade*.

Transformation tool

Microsoft Dynamics NAV Version 2009 introduced a new object type for user interface, the **Page**. This object is used in the Role Tailored Client and not supported by the Classic Client. The Role Tailored Client does not support forms. This has great new possibilities but is difficult from a maintenance perspective.

Each form object has a corresponding page object with the same number. When a change is made to the form object, the page object should also be changed if both the Classic Client and the Role Tailored Client are used.

To solve this problem Microsoft has released the Transformation tool. This tool can create a page object from a form object, and add to, or remove additional metadata from it.

More information about the transformation tool can be found on MSDN: `http://msdn.microsoft.com/en-us/library/dd338789.aspx`.

Transform or design

Many Microsoft Dynamics NAV partners have difficulties answering this question. Should we keep developing forms and transform them to pages, or drop using forms and move completely to pages?

Let's discuss the pros and cons of Form Transformation:

- **Con 1**: Pages are very strict in their UI behavior. Editable pages have a limitation of only using two columns. Many developers have changed their forms to have three columns or controls that do not align nicely. These forms will have to be redesigned in order to move them to a page.
- **Con 2**: Pages offer the ability to have fact boxes. This was not available in the forms so many of the fact boxes need to be designed from scratch. When using the transformation tool the fact box need first to be designed as a form and then converted to a page. The additional meta data required to attach the fact box to a page needs to be manually updated in the TIF editor.

- **Con 3**: Role centers are also new object types not available before. They too need to be designed as a form and then transformed to a page.
- **Con 4**: Maintaining the additional meta data in the TIF tool is a lot of work that will increase the cost of development and make developing less flexible. Each change to a form should be considered if it can be transformed into a page and if it needs additional Meta data.
- **Pro 1**: Many end users still use forms and have learned to work with them and love them. When implementing new features they do not have to move to the RTC. New customers however will not have issues with moving to the RTC directly.

As you can see there are more cons than pro's. As the next release of Microsoft Dynamics NAV will be shipped without forms, the general advice is to move completely to pages as quickly as possible.

In the TIF tool it is possible to create a page out of a form with a different number. This allows you to have a creative form when this is necessary without losing the possibility to use the transformation tool. Examples to this feature are the matrix forms in the standard product.

Design to support

There are different levels of support. First level support is usually done by someone at the customer site that works in the IT department or someone that has feeling for IT. The general first line support questions are about filters, missing data, and so on.

Second level support is usually a small bug in the software or something missing in set up or master data. Depending on the customer this will be solved by the internal IT department or escalated to the partner.

As a developer you are most likely going to do the third level support where something needs to be debugged or reverse engineered in order find the bug.

So before a bug reaches the developer other people have already spent time analyzing the issue without success. The development of the software should be done in such a way that third line support has a minimum chance of being required.

When the guidelines discussed in *Design to use* and *Design to maintain* are used, it will already be easier for second level support to analyze the issue.

Second level support

Most problems in support occur in the second level. The first level support engineers are often very familiar with the system, and third level support engineers are often the original developers of the software.

Second level support people need to be able to go into a database and analyze the issue without having to change their way of thinking.

Let's briefly summarize the general guidelines for this specific topic:

- **Shortcuts**: Use standard shortcuts as much as possible. Remember that many of the shortcuts have changed when moving to the RTC. For example, use *F9* for posting and registering, *Ctrl+F7* for ledger entries. Avoid using reserved shortcuts such as *F8* (Copy previous) and *Alt+F3* (Filter to this value).
- **Screen Layout**: Avoid screen layouts that are too creative. Too much information on a screen is often an indication of bad design and will be difficult to support. Typical examples are multiple sub forms and hiding elements based on business logic such as menu buttons and sub forms. An example of bad design in the standard application is form 6510 Item Tracking Lines where there is a hidden menu button behind a sub form.
- **Variable Naming**: As discussed in the section *Design to maintain*, good naming conventions will make a huge difference when looking at someone else's design. This starts with trying to use the naming conventions that Microsoft uses for the standard application.
- **C/AL Placement**: Microsoft Dynamics NAV is very flexible when it comes to placing C/AL code in objects. Both forms and pages support using C/AL code to an extent that it is possible to write an entire posting routine there. C/AL coding should be done in tables or codeunits unless it is not possible.
- **Using Functions**: When your C/AL code exceeds the size of your screen it is best practice to create a function. This will make the original code more readable for others. Use a name for your function that makes sense so that the code will document itself. An example for this can be found in the codeunit Register Time Sheet (75000) we discussed in *Chapter 8, Consulting*.
- **Global vs. Local Variables**: Variables can be both global and local in C/AL. Microsoft does not have strict guidelines in which to use when. The general rule when looking at the standard application is to use global variables unless the variable is only used in a function. Then it can be local.

[The compiler does not give a warning when using a local variable with the same name as the global variable. The system will always use the local variable first.]

Design to upgrade

It might not be the first thing you think of when designing your application but there will come a time when it needs to be upgraded to a newer version.

When upgrading your application we can split the process into two parts. Part one is the part of the add-on that is written on top of the standard application, new tables, pages, and codeunits that are loosely coupled with the standard application. This part is often easily upgraded. The other part is the changes made in the base application. These changes are often more difficult to move to a newer version.

Has Microsoft changed my (referenced) object

This is the question it comes down to when analyzing the upgrade task. If the object you modified has not been changed by Microsoft, the upgrade is easy. If Microsoft has changed the object slightly we might need to analyze the changes to see if we need to change something as well.

With each release Microsoft tends to redesign a part of the application. If your solution is integrated with the part Microsoft has redesigned it will be a bigger task to bring the add-on forward.

To see the design changes done by Microsoft in a new release, analyze the upgrade toolkit objects to see what it hits.

Some redesign examples

Let's have a look on some redesign examples.

CRM (version 2.0)

In version 2.0 Navision introduced the current CRM application we discussed in *Chapter 4, Relationship Management*. The most important change was to merge company contacts and persons into one table as well as implementing new functionality.

Dimensions (version 3.x)

In version 3.0 Navision introduced the Dimension solution we know today. Before this the current Global Dimensions 1 and 2 were called **Department Code** and **Project Code**.

Bin code (version 3.x)

With the introduction of WMS the usage of the Bin Code field changed. The Bin Code used to be a field in the Item Ledger Entry table (32) and moved to the Warehouse Entries.

Inventory valuation (version 3.x)

No single piece of code in Microsoft Dynamics NAV has changed as many times as the Inventory Valuation solution. Try to avoid changing this in your add-on application.

Item tracking (version 3.6 and 4.0)

As with Inventory Valuation, item tracking has been changed many times. Where older versions had Item Tracking Entries and Item Ledger Entries they are merged into one table in newer versions.

MenuSuite (version 4.0)

Although it is not a functional change, the introduction of MenuSuites in version 4.0 caused a lot of work to upgrade to.

MenuSuites do not support C/AL code. This means that all journals need to be changed for this version.

Jobs (version 5.0)

As discussed in *Chapter 8*, *Consulting*, the Jobs functionality has been changed in version 5.0. The budgeting in the previous version was done differently using Budget Entries and Phase, Task and Step tables.

The Job Journal Line and Job Ledger entries have not changed, but the new job task table has become a mandatory field when posting on a job.

When there is no other way, it is possible to take out the Job Objects and renumber them to be customized tables. This allows you to upgrade to a newer version with minimum impact. After the upgrade, a new project can be started to move to the new Job functionality completely.

Role tailored client (version 2009)

Version 2009 introduced a new user interface called the Role Tailored Client. This was the first real change in the UI since the introduction of the Windows version in 1995.

Item costing (almost all versions)

The Item Costing has been improved in almost every new versions of Microsoft Dynamics NAV. Changes in Item Costing are difficult to upgrade to newer versions and almost always need to be redesigned.

Documentation

While many parts of the application will have no issue in the upgrade it is useful to have external documentation when there is a need to redesign.

This documentation should contain information about the business reason the feature has been implemented. With that information it is possible to do a new fit/gap analysis.

External documentation such as Microsoft Word and Visio files can be linked to C/Side objects. This way it is easy to find the documentation when a developer needs it.

Split operational and financial information

In the Storage and Logistics application we have chosen a data and transaction model that can be easily upgraded to a newer version of Microsoft Dynamics NAV.

This is achieved by creating separate modules that move data to each other.

Design to perform

All good applications are useless if the performance is not adequate. It is important to keep performance in mind when designing your application.

When talking about performance there are two typical issues. The first issue is an application with an overall slow performance, the latter is an application with good performance but users block each other or create deadlocks.

Both issues have their own approach to be analyzed and solved. We will not talk in detail about this process but rather explain how to avoid these situations in general.

OLTP versus OLAP

In any ERP system it is important to balance **Online Transaction Processing** (**OLTP**) with **Online Analytical Processing** (**OLAP**).

This is especially important when working with Microsoft Dynamics NAV. The reason for this is its unique data and posting model, which creates the Analytical data while processing the transaction.

Creating this analytical information in real time can have advantages but when posting transactions take too much time it may not be worth it.

Examples of analytical information are Dimension Ledger Entries and Analysis View Entries but also VAT Entries and Value Entries. Although they give us important information about the business we do not always need them instantly when processing the transaction.

Other examples of Analytical information are Secondary Keys and SumIndexFields. All this information will be created when creating the master record. If a Ledger Entry table has 32 secondary keys and 15 SumIndexFields it will take a considerable amount of time to write this information to the database.

Fast transaction posting

Good performance starts with fast transactions. There are several ways to achieve this.

Cleanup unused indexes

Each secondary SumIndexFields field in the database needs to be maintained whether they are used or not. Microsoft Dynamics NAV allows end users to create their own schedule to maintain this.

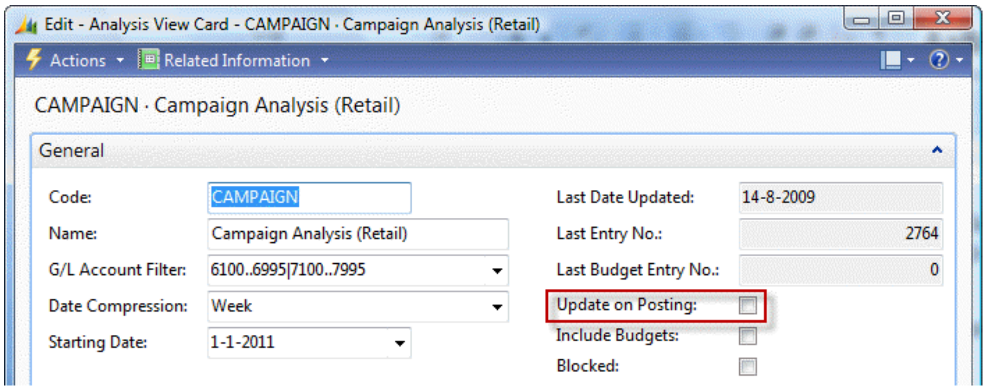
Creating such an index schedule is quite a complex task that should be done by experienced functional developers.

> In versions prior to Microsoft Dynamics 5 SP 1 the overhead of unused SumIndexFields was substantially higher compared to newer versions.

Application setup

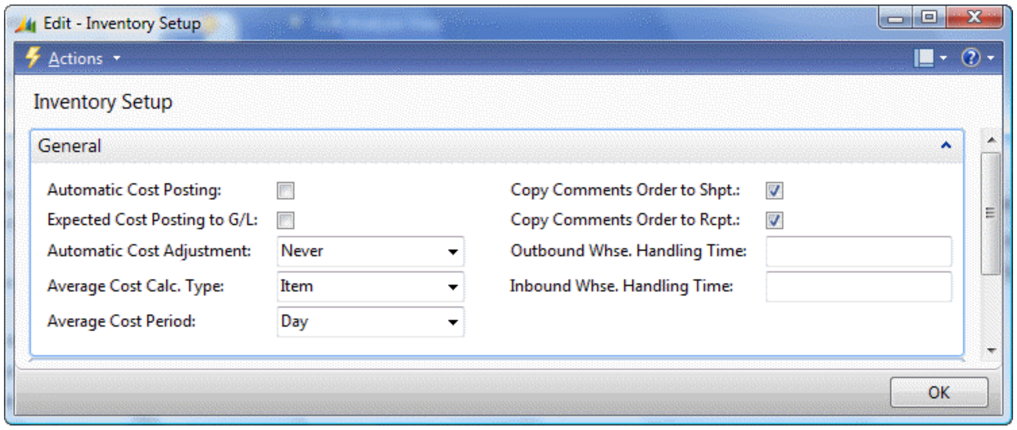
It all starts with a solid application setup. Some setup features in Microsoft Dynamics NAV will cause the system to create more analytical information when posting transactions.

An example is **Update On Posting** for Analysis Views (as shown in the following screenshot). This feature will update the Analysis View Entries at the same time the General Ledger Entries are created:


Edit - Analysis View Card - CAMPAIGN · Campaign Analysis (Retail)
Actions
Related Information
CAMPAIGN · Campaign Analysis (Retail)
General

Code:	CAMPAIGN	Last Date Updated:	14-8-2009
Name:	Campaign Analysis (Retail)	Last Entry No.:	2764
G/L Account Filter:	6100..6995|7100..7995	Last Budget Entry No.:	0
Date Compression:	Week	Update on Posting:	☐
Starting Date:	1-1-2011	Include Budgets:	☐
		Blocked:	☐

Other examples are the **Automatic Cost** options in the **Inventory Setup**. When they are activated the cost is adjusted each time an Item Ledger Entry is created.



Automatic Cost Posting:	☐	Copy Comments Order to Shpt.:	☑
Expected Cost Posting to G/L:	☐	Copy Comments Order to Rcpt.:	☑
Automatic Cost Adjustment:	Never	Outbound Whse. Handling Time:	
Average Cost Calc. Type:	Item	Inbound Whse. Handling Time:	
Average Cost Period:	Day		

Job queue

Microsoft Dynamics NAV is shipped with an excellent Job Queue system. For each company in the database it is possible to set up a Windows service that handles scheduled tasks.

This process is called a **Navision Application Server** or **NAS**. A NAS can execute C/AL code in report and codeunit objects.

Examples of Job Queue tasks are, creating the Analysis View Entries, Posting the Adjust Cost for Inventory Valuation or even Posting Sales, and Purchase documents.

Background posting

Microsoft Dynamics NAV only allows posting documents directly. This means that the user that starts the posting routine needs to wait until this process has been completed before starting a new task.

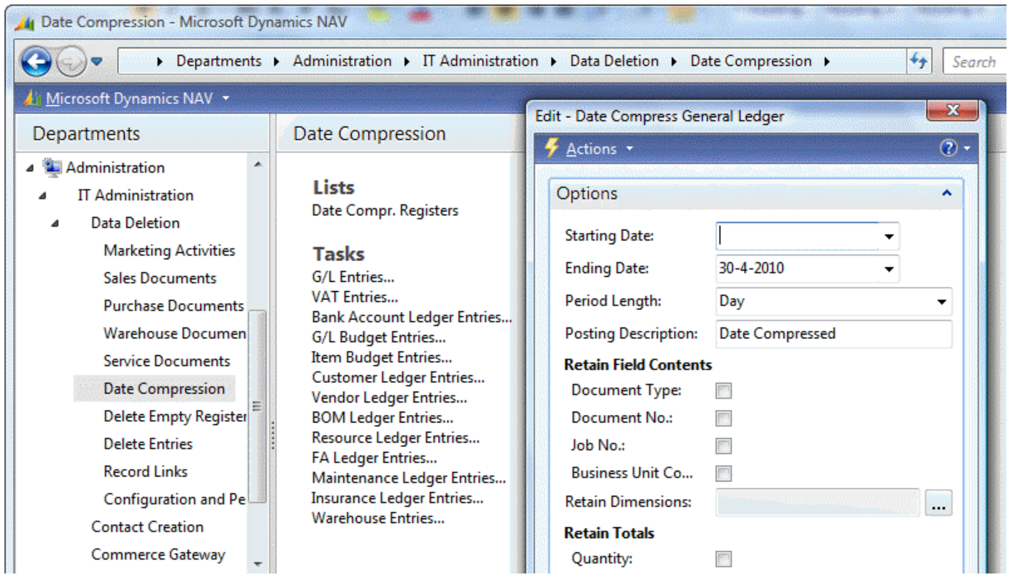
This task can also be handled by the Job Queue. The user marks a document as ready to post and the Job Queue polls for documents with this status.

Date compression and cleanup

When the number of records in a table exceed normal proportions it might be useful to start thinking about doing data maintenance. This is a normal procedure in all ERP systems and Microsoft Dynamics NAV has some possibilities to do that.

Date compression

Most Entry tables in Microsoft Dynamics NAV can be compressed by date. This means that all entries with the same values will be replaced by one new entry. The detailed information is lost afterwards.



Options

Starting Date:	
Ending Date:	30-4-2010
Period Length:	Day
Posting Description:	Date Compressed

Retain Field Contents

Document Type:	☐
Document No.:	☐
Job No.:	☐
Business Unit Co...	☐
Retain Dimensions:	

Retain Totals

Quantity:	☐

Saving the detailed information can be easily implemented by changing the compression report. The detailed information can be saved in a copy of the original table.

The total size of the database has minimal impact on the performance. More important is the size of the tables we are writing to during a transaction.

Data cleanup

Microsoft Dynamics NAV allows most data to be deleted when the fiscal year it was created in is closed.

Examples of data that can be deleted are Sales Shipments and Purchase Receipts. They can be either deleted or moved to copy tables.

Cleaning up data will prevent the transactions from slowing down if your company uses Microsoft Dynamics NAV for a long time. Data cleanup generally starts after using the product for five years and when the database exceeds 100 gigabytes in size.

Locks, blocks, and deadlocks

The Microsoft Dynamics NAV product is very sensitive when it comes to blocking and deadlocks. This has everything to do with the posting model, the inheritance of the Native database, and the numbering used in entry tables.

Blocks and deadlocks are caused primarily by Locks in the database. Locking is a mechanism databases use to ensure consistency of the data.

Native server versus SQL server

Originally Microsoft Dynamics NAV had a proprietary (Native) database. This database did not support row level locking, only table locking.

The Role Tailored Client no longer supports this database and only runs on SQL Server, which does support row level locks. However the current data and transaction model is designed for table locking. The classic client still supports the Native database and the application has one codebase.

The benefit of row level locking on your SQL Server is best experienced in systems with many users creating documents in the same database. Most posting transactions in the database are isolated, meaning only one user at a time can post a document from anywhere in the application.

Locking is always done for a single company, both in the Native and the SQL Database. Unless tables are shared as explained in *Chapter 4*, a user from company A cannot lock a user in company B.

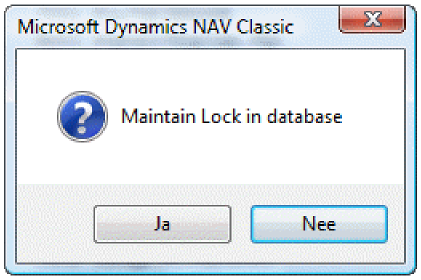
Locking principles

In Microsoft Dynamics NAV locking starts with the `LOCKTABLE` command. Using this command will generate the T-SQL statements that are generated by the application to issue a `UPDLOCK` hint where without the statement `READUNCOMMITED` is issued.

Lets create an example that shows how locking is done. For this example we create a new codeunit Locking A (60000). The codeunit has a global variable `Cust` of type Record 18.

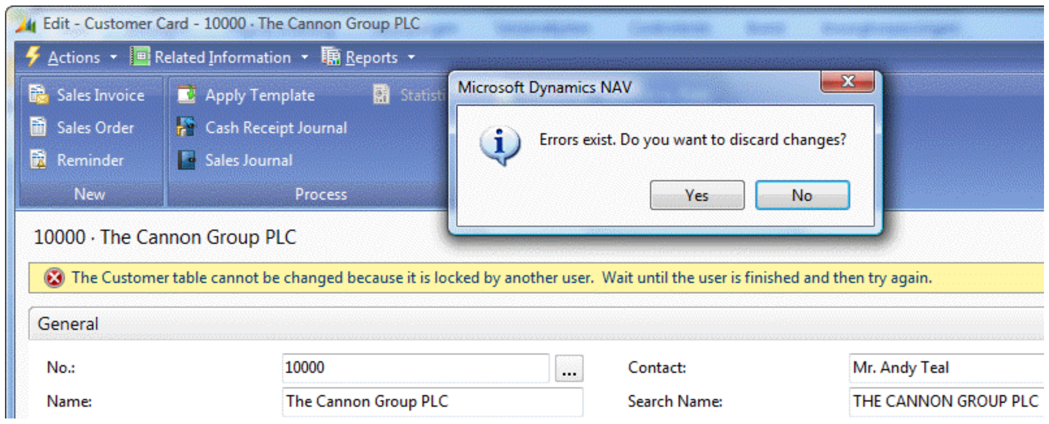
```
OnRun()
Cust.LOCKTABLE;
Cust.GET('10000');
IF CONFIRM('Maintain Lock in database') THEN;
```

We start this codeunit and leave the confirm window open:



Maintain Lock in database

Now we go to the Role Tailored Client, open the Customer Card for Customer 10000 and try to change the name. After 10 seconds we will get this message:



Errors exist. Do you want to discard changes?

10000 · The Cannon Group PLC

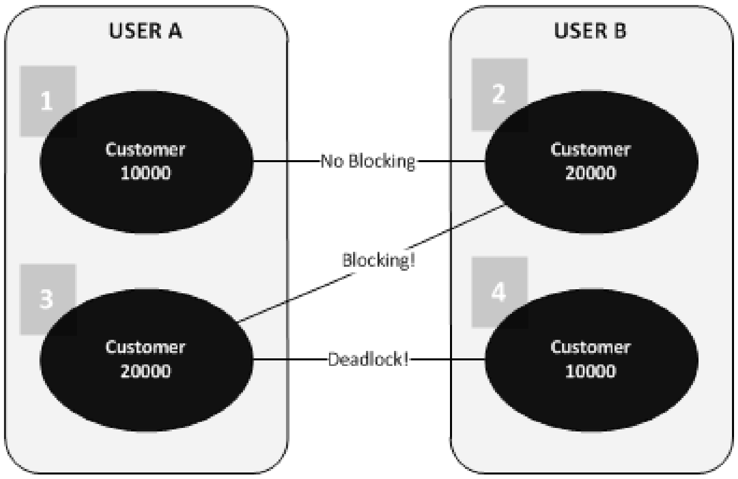
The Customer table cannot be changed because it is locked by another user. Wait until the user is finished and then try again.

No.:	10000	Contact:	Mr. Andy Teal
Name:	The Cannon Group PLC	Search Name:	THE CANNON GROUP PLC

The reason for this error message pop ping up is that the user in the classic client issued an exclusive lock on the record. If we move to Customer 20000 which is the next record in the database we can safely change the name. This record is not locked.

Deadlocks

Let's take this example one step further and simulate a deadlock. Deadlocks happen if users try to lock each other's record in different order.



Lets see this in more detail:

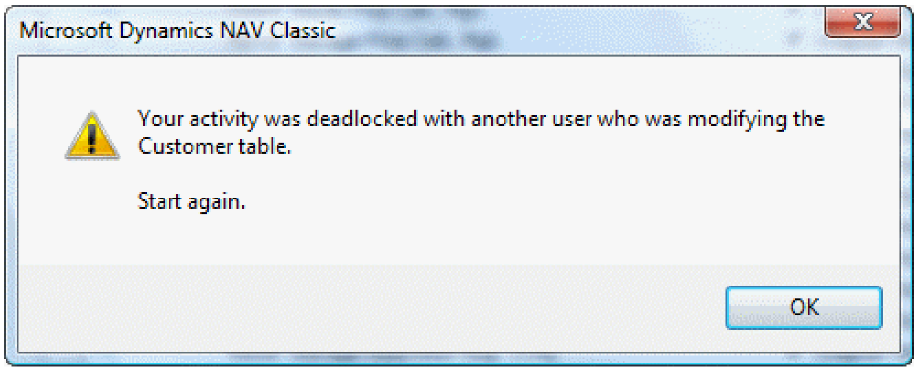
1. User A reads and locks Customer 10000.
2. User B reads and locks Customer 20000.
3. User A tries to read and lock Customer 20000. A blocking event starts...
4. When user B now tries to lock Customer 10000 a deadlock occurs...

To demonstrate a deadlock we have created two codeunits, Deadlock A (60001) and Deadlock B (60002). We need two Classic Clients on the same SQL Server database to do this. Start Deadlock A on one client and Deadlock B on the other.

```
OnRun()
Cust.LOCKTABLE;
Cust.GET('10000');
IF CONFIRM('Start another client and run codeunit 60002') THEN
  LockOtherCust;

LockOtherCust()
Cust2.GET('20000');
IF CONFIRM('Maintain Lock') THEN;

OnRun()
Cust.LOCKTABLE;
Cust.GET('20000');
IF CONFIRM('Select Yes on the other client') THEN
  LockOtherCust;

LockOtherCust()
Cust2.GET('10000');
IF CONFIRM('Did the deadlock happen?') THEN;
```

Then select **Yes** on both confirmation boxes. One of the Clients should now deadlock.

SQL Server checks for deadlocks every 5 seconds and kills the transaction that has the lowest roll back impact on the database. This is why users will experience deadlocks as slow sometimes and fast other times.



Your activity was deadlocked with another user who was modifying the Customer table.

Start again.

The error message is confusing as it leads us to believe that we have locked the entire table but this is not true.

The function `LockOtherCust` reads a record from the Customer table with another variable. This new variable `Cust2` does not explicitly issues a `LOCKTABLE` command. This proves that `LOCKTABLE` is a transaction command that is valid for ALL variables of this type.

Blocking and deadlocks in Microsoft Dynamics NAV

The standard application has several built-in blocking events by design. This is to ensure database integrity and to avoid deadlocks.

The two main isolating tables in Microsoft Dynamics NAV are the G/L Entry table (17) and the Item Ledger Entry table (32).

Both codeunits, Sales-Post 80 and Purch.-Post 90, we discussed earlier isolate on the G/L Entry table. Codeunits 12 has a similar construction.

```
OnRun()
...
  IF RECORDLEVELLOCKING THEN BEGIN
    DocDim.LOCKTABLE;
    SalesLine.LOCKTABLE;
    ItemChargeAssgntSales.LOCKTABLE;
    PurchOrderLine.LOCKTABLE;
    PurchOrderHeader.LOCKTABLE;
    GLEntry.LOCKTABLE;
    IF GLEntry.FINDLAST THEN;
  END;
```

In real life this means that no one in a company can post a Sales Document together with a Purchase Document or a General Journal.

Taking away this isolation will most likely cause deadlocks as the locking order of each transaction in Microsoft Dynamics NAV is different.

This emphasizes the importance of fast transactions and generating analysis data in separate batches.

Impact on development

If we summarize the impact of all this knowledge on your development it emphasizes the importance of designing your own application structures that interface with the Standard Application.

When changing and implementing the standard application, try to reduce the overhead during posting as much as possible.

Create compression routines and allow end users to periodically clean up records.

In the next section we will talk about how to design to analyze and allow end users to generate analysis data in batches separate from the posting transactions.

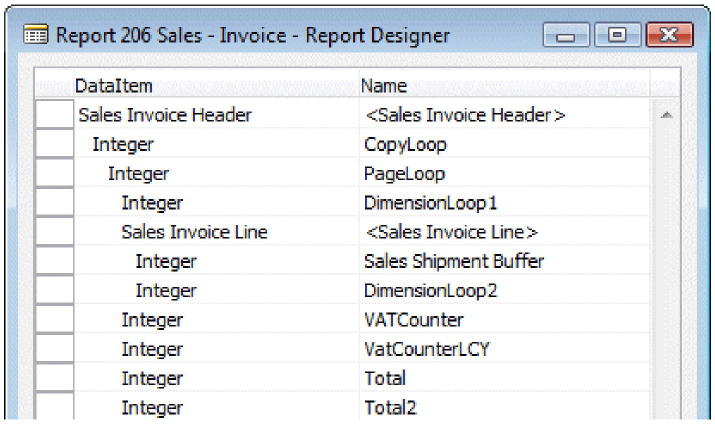
Design to analyze

Analysis in Microsoft Dynamics NAV should always be done on (ledger) entry records. There are many types of entry records that are either created during a transaction or in batches.

Avoid building analysis on document tables. It should always be possible to delete old data in the database without losing information which is essential for data analysis.

Report design

Designing a report in Microsoft Dynamics NAV starts with generating a Data Set. This is built using table relations and can get quite complex.



Report 206 Sales - Invoice - Report Designer

DataItem	Name
Sales Invoice Header	<Sales Invoice Header>
Integer	CopyLoop
Integer	PageLoop
Integer	DimensionLoop1
Sales Invoice Line	<Sales Invoice Line>
Integer	Sales Shipment Buffer
Integer	DimensionLoop2
Integer	VATCounter
Integer	VatCounterLCY
Integer	Total
Integer	Total2

When the Data Set is defined the second step is to define the Layout. This is done differently in the Classic Client and the Role Tailored Client. Creating report layouts is beyond the scope of this book.

Reports with a large Data Set are complex to maintain and have a risk in being slow in performance as the database engine needs to read all the information before combining the information into a view.

This can be solved by preparing the data first and running the report afterwards. This approach is quite common in data warehousing. The preparation of the data can be done in scheduled batches running in the Job Queue.

Version and object management

When doing software development, discussing version management is unavoidable. Microsoft Dynamics NAV is flexible in this and allows developers to make their own decisions on this subject rather than forcing them to one way of versioning.

What is a version

In Microsoft Dynamics NAV there are two ways of determining what a version is. The first and easiest approach is to change the version of an object each time it changes. The initial released version is 1.00 and each change increments to 1.01, 1.02, and so on. A big change will lead to version 2.00.

Another more common approach in Microsoft Dynamics NAV is to group version numbers in releases of a group of objects together. When this is applied the application gets a version number that is incremented each time we release. This means that an object with version number 1.01 can jump to 1.04 if it was not changed in releases 1.02 and 1.03.

Version numbering

There are rules in Microsoft Dynamics NAV for version numbering although the rules have changed over the years.

The current version principle allows using letters and digits. The letters indicate the product and country code, the digits, the version, subversion, and service pack number.

Let's look at an example object to clarify this. Codeunit Sales Tax Calculate (398).

NAV	W1	3.	70.	01
				The last service pack this object was changed
			The last subversion this object was changed	
		The last version this object was changed		
	The Localization Version			
The Product Name				

If Microsoft would change this object in Service pack 1 for 2009 the new version number would be NAVW16.00.01.

Combining versions

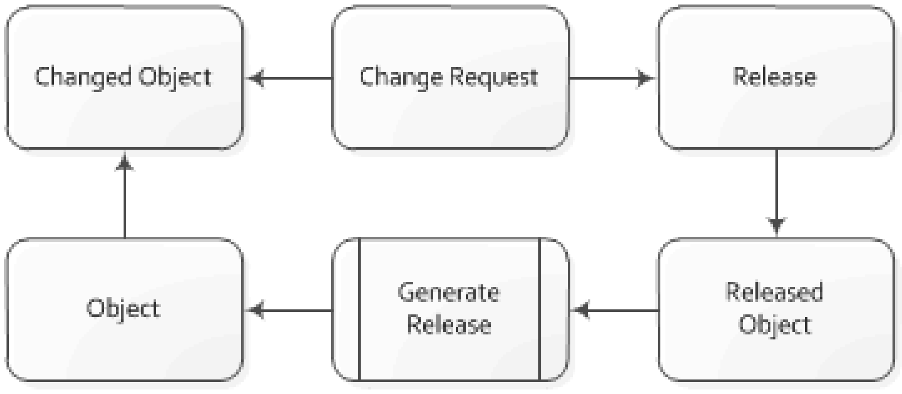
An object can have multiple versions but only one version for each product or country version. A localized object gets version NAVW13.70.01, NAVNL6.00.01. This means that although the global product team has not changed the object, it has been changed by the Dutch localization team.

Creating a version

Versioning in Microsoft Dynamics NAV is done manually. The version number is an editable field in the Object table (2000000001) that can be freely changed.

Developing a tool to do this is easy and has been done by many partners in the channel.

The Data and Transaction model of such a solution should look something like this.



The process starts with a change request. This can be fixing a small bug or creating new functionality. For this change request objects need to be modified.

Each modified object is attached to the change request. We can release several change requests at the same time. All objects in the release will get the version number from the release which can be automatically updated in the Object table.

By saving the change request and release information in the database we will also generate documentation that will help future developers to find information on why objects were changed.

Tracking object changes

Object changes can be tracked using triggers in SQL Server. All the C/Side objects are stored in the Object table (2000000001).

To connect an object change to a change request the developer should tell the system which request they are currently working on. This will enable us to have a failsafe tracking mechanism to do version management.

To view the complete solution for tracking object changes visit: `http://dynamicsuser.net/blogs/stryk/archive/2009/05/18/object-auditing.aspx`.

Development methodology

For developing software there are many methodologies such as Prince2, Extreme Programming or the Microsoft Solutions Framework.

Most of these methodologies are suitable to be applied to Microsoft Dynamics NAV but they should be used properly. Because of the flexibility of the product it is easy to leave out steps in the process that should be there.

In Microsoft Dynamics NAV it is extremely easy to quickly create and modify business software. This is by far the strongest selling point for the solution but also the biggest pitfall.

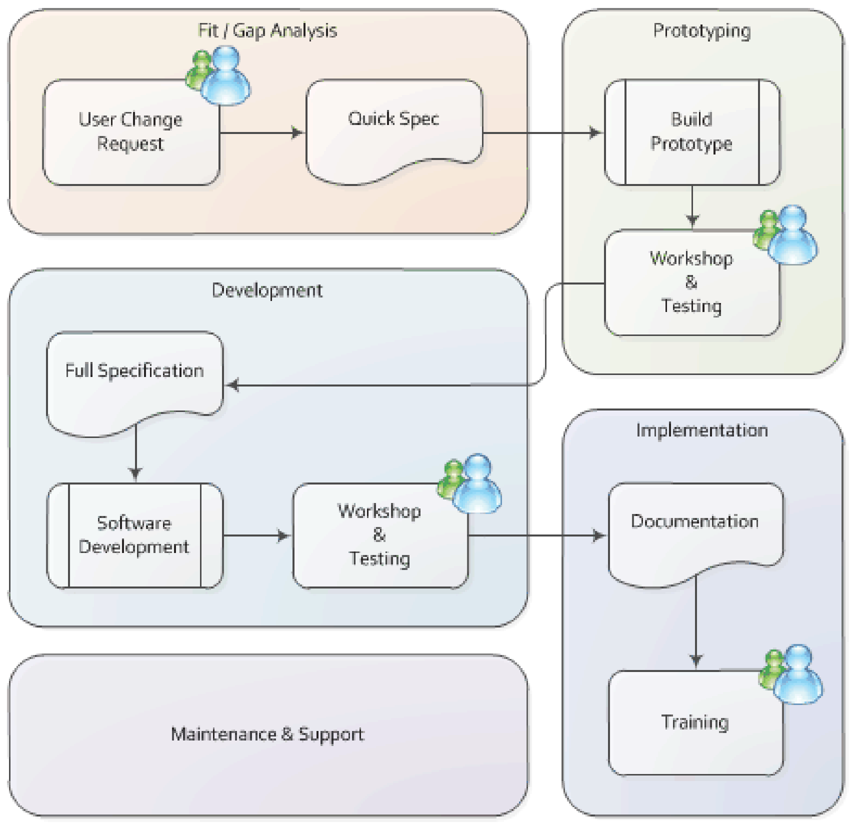
A sample approach

When an end user requests a change to the application it is tempting for most experienced developers to go into the application and create it, preferably, in the production database without documentation. This is not the desired way of software development.

However, Microsoft Dynamics NAV is a suitable design environment for prototyping and Rapid Application Design. All the example applications in this book are first built with prototyping and later finalized using testing.

If we design a suitable Development Methodology for Microsoft Dynamics NAV we see that the Application Lifecycle perfectly fits our methodology.

When implementing Microsoft Dynamics NAV it is important to involve the end users in each step of the development process.



Fit/gap analysis

At this phase a quick specification is usually enough to describe what the user would like the system to do and a possible solution in the application. This document should not count more than two or three pages. During the prototyping phase it is normal to come across advanced understanding. It would be a waste of valuable time to find this during the initial analysis with the risk of not finding them anyway.

Prototyping

With the quick specification a developer creates the solution as a draft without going into too much detail. This should be enough to show the end user what the solution will look like when it is finished. This will often lead to new questions and ideas that should be carefully considered, and put into the full specification or a new prototype, should be built first.

Development

Depending on the amount of changes after the prototype development can often start with the work done already. In this stage all the details should be worked out and tested.

There is no complete checklist for developers to use when developing in Microsoft Dynamics NAV but here is a partial list of things to consider:

- **Captions/Translations**: Make sure all objects have the required captions and translations populated.
- **Table relations**: Make sure all Table Relations are in place, check the Ledger Entry and Line tables as well, they are frequently forgotten.
- **Modify and delete triggers**: What happens if a user modifies or deletes the record. Make sure that everything is nicely handled in the `OnModify` and `OnDelete` C/AL triggers. OnRename should be automatically handled by C/Side. Renaming a table with many table relations may cause severe locking in the database. If users should not rename a record, this can be blocked by placing an `ERROR` command in the OnRename trigger.
- **LookupFormID and DrillDownFormID**: Even when running the Role Tailored it is important to assign a Lookup and Drilldown form ID. Lookup forms or pages are used for table relations and when using the Lookup button in the Classic Client. Drilldown forms or pages are used when drilling down from a SUM flow field.
- **CardFormID**: The Role Tailored Client always starts a list page when a user selects a menu item or a cue. Double clicking a row will open the associated card page. This is controlled by the CardFormID property on a list page.
- **Field Groups**: To show records from a table relation when entering values the Role Tailored Client does not directly use the LookupFormID but first shows a **DropDown** list. The fields in this list are defined in the associated tables Field Groups. Each table can have only one Field Group called **DropDown**.

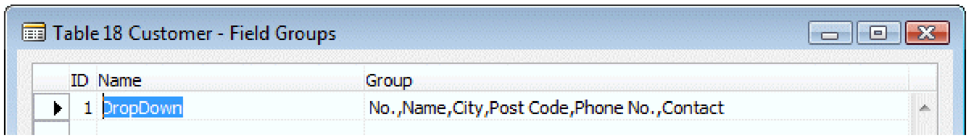


Table 18 Customer - Field Groups

ID	Name	Group
1	DropDown	No.,Name,City,Post Code,Phone No.,Contact

- **Menu Buttons and Actions**: Menu Buttons and Actions should make sense to end users. Make sure to use logical names, and avoid creating menu buttons and actions that are solely for super users and confuse end users. Actions should be placed in the correct container. Only promote actions that will be frequently used by all end users.
- **Shortcuts**: Always assign ampersand (&) shortcuts and avoid double shortcuts. When using function keys such as *F3* and *F9* follow the Microsoft standard.
- **Compression and Posting**: If your solution will generate a potentially large about of data, be sure to provide compression, posting or cleaning up routines so end users can periodically maintain the data.
- **Permissions**: Does your solution require additional permissions to be set up in the system? Make sure to document this when delivering the solution.
- **Unused variables**: Make sure not to leave unused variables in the C/AL objects. Although they won't break the functionality it will make future maintenance of the software more complex.
- **FIND Commands and Locking**: Double check the usage of the correct find commands before you ship the software. Using the wrong commands and leaving locking to the database engine may cause extra performance overhead.

More detailed information about these features is explained in the book Programming Microsoft Dynamics NAV 2009, written by David Studebaker and published by Packt.

There is a free tool available that will perform over 40 checks on your objects to see if the coding is done correctly. This can be downloaded from: `http://www.mibuso.com/dlinfo.asp?FileID=826`.

A good explanation about using the different FIND commands in C/AL can be found on: `http://dynamicsuser.net/blogs/waldo/archive/2008/02/01/what-impact-does-my-c-al-have-on-sql.aspx`.

Testing

Testing is probably one of the most important but under-rated tasks of application design.

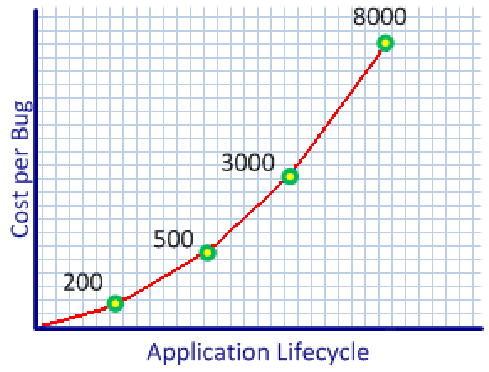
Testing involves three main steps:

1. Does the software meet the original requirements? If this is not the case it does not make sense to continue testing.

2. Does it work as expected? This includes trying to deliberately break the solution. If the software is not monkey proof things will certainly go wrong when using it. Here Murphy's Law is applicable: "What ever can go wrong, will go wrong".
3. Does it fit the rest of the application? Is the software usable and intuitive? A solution that is bug free but difficult to use will be expensive to maintain.

The testing should be performed by someone who likes doing it and has enough available time. If someone is asked to test the software who is buried under normal work the change of bugs slipping in is quite high.

The cost of fixing a bug increases as the software evolves. The sooner a bug is fixed the better.



Implementation

When the changes are developed and tested the documentation should be finalized. This can be either a manual for end users, or a technical reference for future developers and support engineers.

The end users should be trained to use the software.

Maintenance and support

After the software is implemented and users are trained the solution goes into the maintenance and support stage. During this stage the application manager needs to take care of the data generated by the solution, analyze it and cleanup the data periodically.

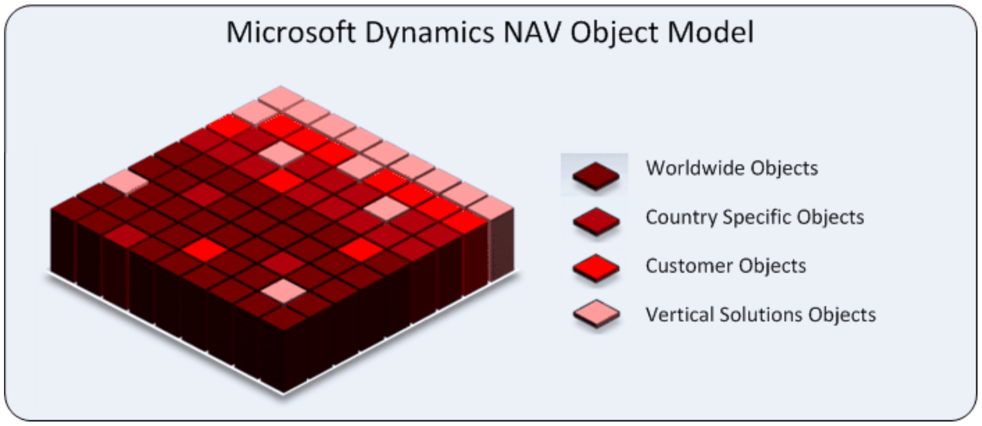
If the end users request a change on the solution the cycle starts again.

The project

Implementing an ERP product such as Microsoft Dynamics NAV is not just installing in a software package and start using it. Each part of your company will have to make decisions how to integrate their work with the software. This often leads to an interesting new look at your companies way of work.

Standard, customized, or both

There are several ways of implementing Dynamics NAV. It is important to make a decision what kind of implementation you do, and adjust the implementation accordingly.


Microsoft Dynamics NAV Object Model
Worldwide Objects
Country Specific Objects
Customer Objects
Vertical Solutions Objects

Compared to when it was introduced in 1995, Microsoft Dynamics NAV 2009 is a mature ERP package with all the built-in functionalities we discussed in this book. On top of this standard product resellers have built horizontal and vertical solutions called add-on products. These two combined offer powerful solutions for companies that cannot work with the standard product but are flexible enough to use a vertical solution.

Add-on products

Vertical solutions have often started years ago as a customised solution for a company who decided to implement Microsoft Dynamics NAV. Together with the implementation partner these companies have customized the product to meet their requirements.

Many of these add-on products are now grown up software solutions that fit a vertical industry.

When buying an add-on solution it is good to ask the reseller some questions.

- What is the release procedure?

  A solid add-on solution has a release procedure. Most resellers have periodical release each half year or maybe sooner. If a bug is found in the software there should be a hot fix. Most resellers have releases they support. Make sure to know what versions are still supported.
- How do I upgrade to a new version of the vertical solution?

  If a new version of the vertical solutions is released there should be an upgrade procedure. This should be clearly documented and tested by the reseller.
- Am I allowed to make changes to the software?

  Most add-on resellers do not recommend their customers to change the software. The reason for this is the increased complexity of bug fixing and upgrading.
- What if I do change the software?

  If an add-on solution is customized anyway it is basically downgraded from being a supported add-on solution to a customized database. For most resellers it is difficult to support these customized solutions.

Customizing

Although customizing an add-on solution is not always recommended, customizing Microsoft Dynamics NAV should not be considered bad practice.

The impact of customization on Microsoft Dynamics NAV can make a difference that can be compared to a suit that is confection or tailored to fit. The benefits of having an ERP package that exactly fits the organization can be more important than the increased cost of ownership of the solution.

Total cost of ownership

The Total Cost of Ownership of Microsoft Dynamics NAV depends highly on the level of customizations. A non-customized implementation with one or two good add-on products created by experienced consultants will have a low impact on your company and will be easy to maintain and support.

The higher the level of customizations, the more it will cost to keep the application running. This is not per se a bad thing. If your company has a unique way of doing business it might need an ERP package that supports this unique way.

Roadmap to success

Designing a solid application in Microsoft Dynamics NAV starts with a thorough knowledge of the standard application functionality, and its design philosophy.

Secondly, we need to carefully analyze the business process we want to support, and implement new functionality step by step to ensure good quality as the solution grows bigger and mature.

Use data and posting models that are similar to Microsoft Dynamics NAV and try to maintain a similar user interface. This will make it easier for end users to adopt your solution and more likely for the software to be easy to maintain and support.

Last but not least do good housekeeping in your database, compress and cleanup data periodically to guarantee stable performance of the system now and in the future.

Summary

In this book we covered functional and technical design of both standard Microsoft Dynamics and how to extend the application to succeed.

This book is not finished. After the publication we will periodically publish new articles, tips and tricks based on the information in this book on `http://www.brummelds.com`.

Any questions or comments regarding the information published in this book can be posted there for discussion as well.

Installation Guide

With this book we provided development examples that can be installed using the demo version of Microsoft Dynamics NAV 2009 SP1 W1.

This demo version can be downloaded from `msdn.microsoft.com`.

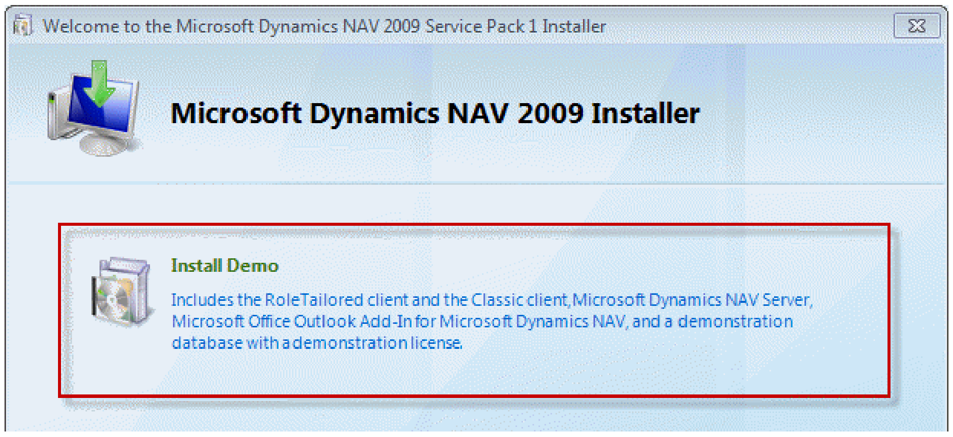
Licensing

Microsoft has very strict licensing regulations for using and developing in Microsoft Dynamics NAV.

For educational purposes, you may use the MSDN license to develop new objects with numbers 123.456.700 to 123.456.799.

Installing Microsoft Dynamics NAV

After downloading the product CD from the MSDN website run the `setup.exe` file.


Welcome to the Microsoft Dynamics NAV 2009 Service Pack 1 Installer
Microsoft Dynamics NAV 2009 Installer
Install Demo

Includes the RoleTailored client and the Classic client, Microsoft Dynamics NAV Server, Microsoft Office Outlook Add-In for Microsoft Dynamics NAV, and a demonstration database with a demonstration license.

From the installation options select **Install Demo** (as shown in the previous screenshot).

Changing the license

After the installation completes we can use both the Classic Client and the Role Tailored Client. We use the Classic Client for administration purposes and development. The user interface is done using the Role Tailored Client.

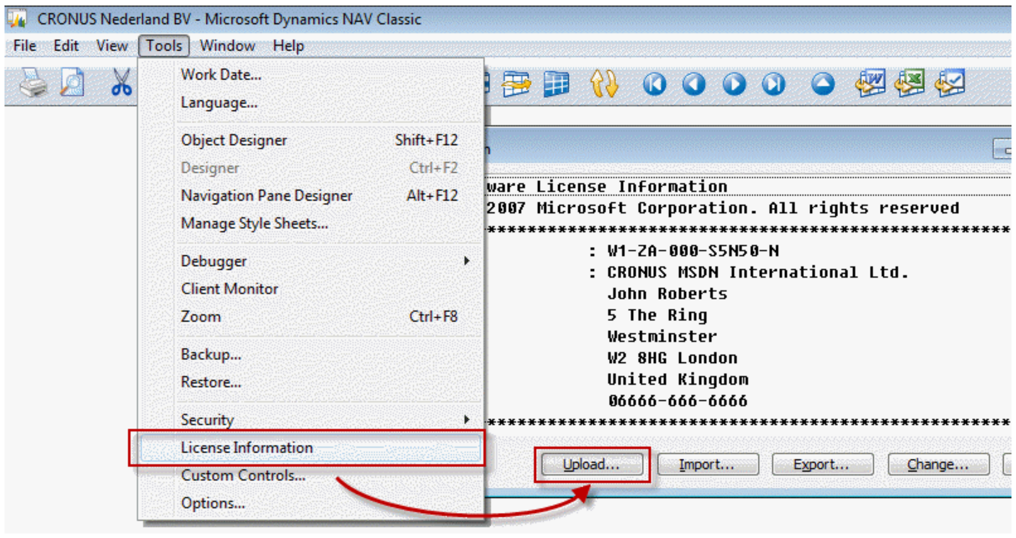
> The example objects can be used in the Role Tailored Client only. This means there are no Form objects, only Pages.

Each server instance of Microsoft Dynamics NAV runs on a license file. This file determines what access we have to the system. The demo license that is installed allows us to access al functionality but not the C/AL code.

To access the C/AL code we would need an official partner development license. To get this license we would have to register as a partner and start being a reseller. If this is not what we want to do we can use the MSDN license.

The MSDN license will allow access to all the new objects developed for the book. Access to the base application change examples is not possible with this license.

To change the license, open the Classic Client, then open the **Tools** menu and select **License Information** (as shown in the following screenshot):



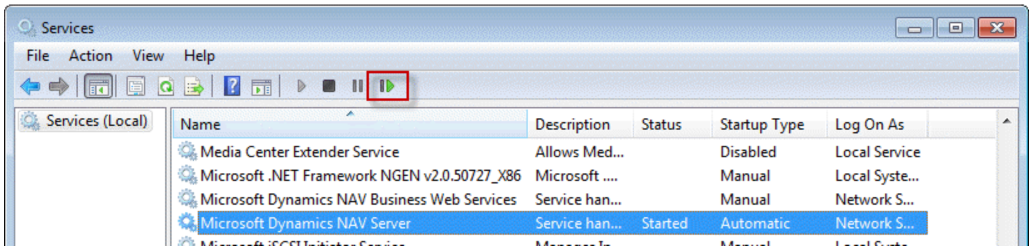
```
ware License Information
2007 Microsoft Corporation. All rights reserved
**************************************************
          : W1-ZA-000-S5N50-N
          : CRONUS MSDN International Ltd.
            John Roberts
            5 The Ring
            Westminster
            W2 8HG London
            United Kingdom
            06666-666-6666
**************************************************
```

This opens the **License Information** screen where we can select **Upload**, which opens a file dialog where we can select the MSDN license.

[To enable the license file on the Classic Client restart the application.]

Restart service tier

To enable the license file on the Role Tailored Client we need to restart the Service Tier. This can be done from the **Services** window in the Windows Control Panel:



Name	Description	Status	Startup Type	Log On As
Media Center Extender Service	Allows Med...		Disabled	Local Service
Microsoft .NET Framework NGEN v2.0.50727_X86	Microsoft		Manual	Local Syste...
Microsoft Dynamics NAV Business Web Services	Service han...		Manual	Network S...
Microsoft Dynamics NAV Server	Service han...	Started	Automatic	Network S...

Installing the objects

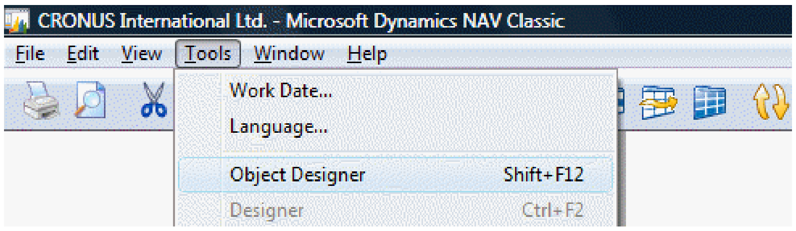
This book has three Microsoft Dynamics NAV object files, two DLL files, a SQL Server script, and some helper files for the installation.

- `Chapter2-4.fob`: This file contains the Squash Court examples used in *Chapters 2*, *A Sample Application*, *Chapter 3*, *Financial Managemenet*, and *Chapter 4*, *Relationship Management*.
- `Chapter7-9.fob`: This file contains the storage and logistics application used in *Chapter 7*, *Storage and Logistics*, and the sample interfaces for *Chapter 9*, *Interfacing*. We need the additional SQL Server scripts to get the ADO examples to run.
- `Chapter8.fob`: This file contains the Job extensions for *Chapter 8*, *Consulting*. This chapter also requires the additional `DLL` files to be installed.
- `RF database.sql`: This is the SQL Server script used in *Chapter 9* to create the RD Database and create the demo data.
- `MSDN.flf`: This is the MSDN License we can use to access the custom objects numbered from 123.456.700 to 123.456.799.
- `NavMaps.dll` and `VEControl.dll`: These are the Dynamic Link Library files we need for *Chapter 7*.

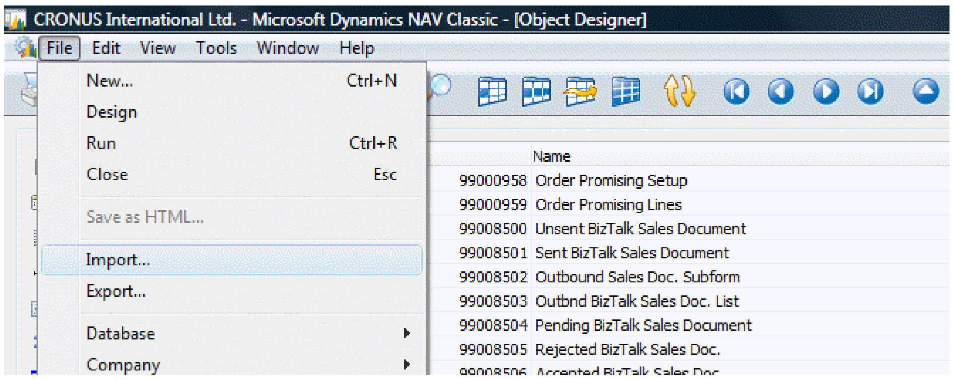
- `Pin1.gif` and `Pin5.gif`: These are the icons displayed on the Bing Map.
- `AddInImporter.msi`: This enables us to register add in applications.

Importing a FOB file

To install the objects first open the Object Designer in the Classic Client by selecting the **Object Designer** *[Shift + F12]* option from the **Tools** menu as shown in the following screenshot:


CRONUS International Ltd. - Microsoft Dynamics NAV Classic
File
Edit
View
Tools
Window
Help
Work Date...
Language...
Object Designer Shift+F12
Designer Ctrl+F2

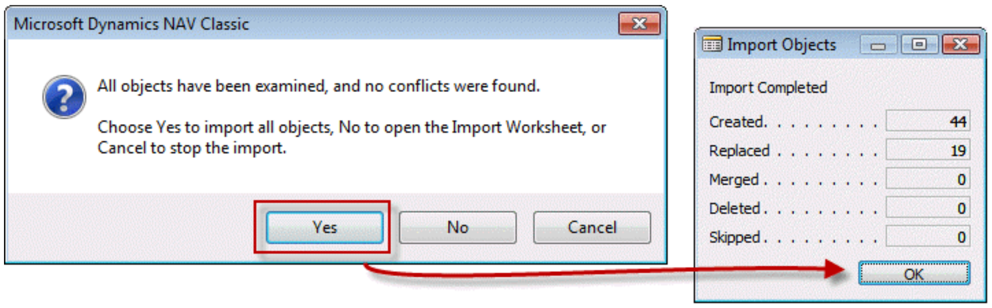
When the Object Designer is active, the **File** menu shows some additional options:



	Name
99000958	Order Promising Setup
99000959	Order Promising Lines
99008500	Unsent BizTalk Sales Document
99008501	Sent BizTalk Sales Document
99008502	Outbound Sales Doc. Subform
99008503	Outbnd BizTalk Sales Doc. List
99008504	Pending BizTalk Sales Document
99008505	Rejected BizTalk Sales Doc.
99008506	Accepted BizTalk Sales Doc.

Select **Import**, this will open a file dialog window, now select the `.fob` file you want to import.

If everything is as it should be this dialog should appear:


Microsoft Dynamics NAV Classic

All objects have been examined, and no conflicts were found.

Choose Yes to import all objects, No to open the Import Worksheet, or Cancel to stop the import.

Import Objects

Import Completed

Created.	44
Replaced	19
Merged	0
Deleted.	0
Skipped	0

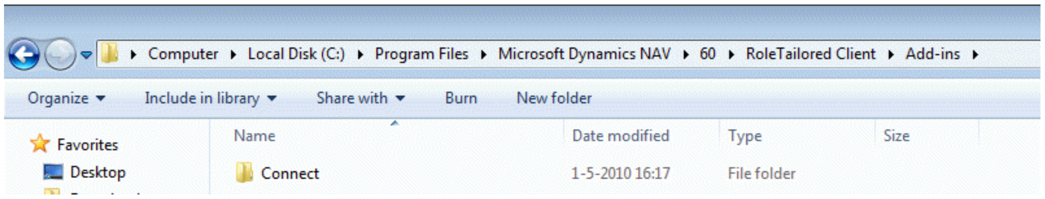
Select **Yes** and this dialog box will confirm the import.

Installing the dynamic link library files

To support the Bing Maps Client Add-in, Geocoding and distance calculation, we ship two `DLL` files and five `.gif` files, along with some supporting files for the installation.

- `NavMaps.dll`
- `VEControl.dll`
- `pin1.gif`
- `pin2.gif`
- `pin3.gif`
- `pin4.gif`
- `pin5.gif`
- `RegisterDll.bat`

These files should be placed in this folder:


Computer ▸ Local Disk (C:) ▸ Program Files ▸ Microsoft Dynamics NAV ▸ 60 ▸ RoleTailored Client ▸ Add-ins ▸

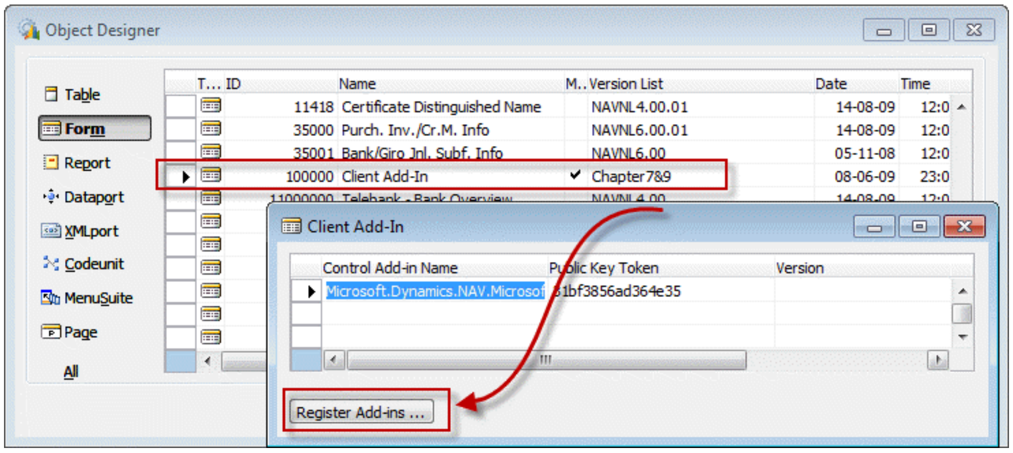
Name	Date modified	Type	Size
Connect	1-5-2010 16:17	File folder	

Register NavMaps.dll

To register this `DLL` we use `RegAsm`. The command is predefined in the `RegisterDll.bat` file that we can execute.

Register VEControl.dll

To register the visual map control we use the **Client Add-In** form in the classic client:



T...	ID	Name	M..	Version List	Date	Time
	11418	Certificate Distinguished Name		NAVNL4.00.01	14-08-09	12:0
	35000	Purch. Inv./Cr.M. Info		NAVNL6.00.01	14-08-09	12:0
	35001	Bank/Giro Jnl. Subf. Info		NAVNL6.00	05-11-08	12:0
	100000	Client Add-In	✓	Chapter7&9	08-06-09	23:0
	11000000	Telebank - Bank Overview		NAVNL4.00	14-08-09	12:0

Control Add-in Name	Public Key Token	Version
Microsoft.Dynamics.NAV.Microsof	31bf3856ad364e35	

This action requires that the AddinImporter is installed. This tool is distributed by Microsoft and can be installed using the `AddInImporter.msi` file.

In the file dialog we select the `VEControl.dll` file.

Index