

Paul M. Diffenderfer | Samir El-Assal

# Microsoft Dynamics NAV

Jump Start to Optimization

2nd Edition



**VIEWEG+**  
**TEUBNER**

Paul M. Diffenderfer | Samir El-Assal

Microsoft Dynamics NAV

**Stochastic Petri Nets**

by Falko Bause and Pieter S. Kritzinger

**From Enterprise Architecture to IT Governance**

by Klaus D. Niemann

**ISSE/SECURE 2007 Securing Electronic Business Processes**

by Norbert Pohlmann, Helmut Reimer and Wolfgang Schneider

**Understanding MP3**

by Martin Ruckert

**Process Modeling with ARIS®**

by Heinrich Seidlmeier

**Computing in Russia**

by Georg Trogemann, Alexander Y. Nitussov and Wolfgang Ernst (Eds.)

**Computing Fundamentals**

by J. Stanley Warford

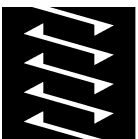
Paul M. Diffenderfer | Samir El-Assal

# Microsoft Dynamics NAV

Jump Start to Optimization

2nd revised Edition

With 209 Illustrations



**VIEWEG+**  
**TEUBNER**

Bibliographic information published by the Deutsche Nationalbibliothek  
The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie;  
detailed bibliographic data are available in the Internet at <http://dnb.d-nb.de>.

This book is the revised 3rd edition of the German book "Profikurs Microsoft Dynamics NAV"  
(© Vieweg+Teubner 2008) now offered to the English speaking audience by the original authors.

1st Edition 2006

The title of this edition was "Microsoft Navision 4.0".

2nd revised Edition 2008

All rights reserved

© Vieweg+Teubner | GWV Fachverlage GmbH, Wiesbaden 2008

Editorial Office: Sybille Thelen | Andrea Broßler

Vieweg+Teubner is part of the specialist publishing group Springer Science+Business Media.  
[www.viewegteubner.de](http://www.viewegteubner.de)



No part of this publication may be reproduced, stored in a retrieval system or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the copyright holder.

Registered and/or industrial names, trade names, trade descriptions etc. cited in this publication are part of the law for trade-mark protection and may not be used free in any form or by any means even if this is not specifically marked.

Cover design: KünkelLopka Medienentwicklung, Heidelberg

Printing company: MercedesDruck, Berlin

Printed on acid-free paper

Printed in Germany

ISBN 978-3-8348-0516-4

## **preface to 2<sup>nd</sup> english edition**

---

What was once simply Navision, became Navision Financials which became Navision Attain, then Microsoft Navision and is today called Microsoft Dynamics NAV. What is important to the loyal owner or user is that the ERP world behind the changing name is the same excellent quality they have come to trust. The newest version, Microsoft Dynamics NAV, is a continuation of this tradition of quality. This book is intended to serve companies and individuals using a wide range of versions within the Navision tradition. Therefore, the authors concentrate on the more essential deep features and concepts that remain largely unchanged despite the uneasy reality of shifting product names.

Following the adage, “Teach a boy to fish and he’ll feed his family for a life time,” the authors have developed examples that strengthen the reader’s intuition, encouraging him or her to think deeply about the system so that they have the confidence and tools to strike out on their own and explore the endless opportunities for optimization made possible through Microsoft Dynamics NAV. Whether or not you master the coding techniques covered in this book, you will become a better judge of the quality of others’ work. As you probably already know, the ERP system is the heart of your enterprise, and therefore, you should not trust just anyone to do open-heart surgery on it. This book will help you develop a vision for successfully administering an ERP implementation even if you do none of the programming and physical implementation yourself.

Readers are invited to contact the authors directly via email at: [pdiffenderfer@yahoo.com](mailto:pdiffenderfer@yahoo.com), if more specific Microsoft Dynamics NAV assistance—not otherwise covered in this book—is needed. Likewise, mention must be made of the best source for Microsoft Dynamics NAV online help, [www.mibuso.com](http://www.mibuso.com). Thanks MIBUSO for getting us all—professionals as well as amateurs—out of the odd, technical quandary!

Special thanks to Jill L. Keehner for the intense editing she did to improve this book—the 2<sup>nd</sup> english edition of a book that has for too long straddled a precarious line of bilingual ambiguity.

Paul Diffenderfer and Samir El-Assal

February 2008, Frankfurt, Germany

## **preface to 1<sup>st</sup> english edition**

---

In the last year and a half, worldwide interest in the ERP solution Microsoft Navision has more than doubled—and with good reason. Microsoft Navision already enjoyed a broad and solid utility when it was purchased by Microsoft in early 2002, and now with the leadership and resources of Microsoft this ERP solution has quickly and dramatically improved in new and promising areas. Microsoft Navision is truly a comprehensive solution to the complex business information management and analysis problems experienced by medium to small-sized firms. Whether you are operating a production or merchandising firm or a purely service-based establishment, Microsoft Navision has tools that can streamline and optimize your information infrastructure.

This is the third edition of the successful book “Microsoft Navision optimiert einsetzen.” In this book the authors attempt to discuss some of these new technologies as well as continue to offer an introduction to Microsoft Navision’s deeper structure and programming conventions, which by and large have remained a stable and unchanged basis for the recent software developments.

As ever, the reader is encouraged to test the examples while studying this book. It suggested that one make use also of the abundant reference resources included on the Microsoft Navision demonstration CD in PDF format as well as the software’s extensive on-line help files. If you do not have a copy of this demonstration CD please contact your Microsoft Business Solutions Center or see [www.mibuso.com](http://www.mibuso.com) and they will be able to supply this for you.

The purpose of this book is to help the reader overcome the danger of becoming overwhelmed by the size and complexity of the software and its resources. Therefore this book will be an aid in building a sense of the important themes, themes that will develop the right intuition in the reader and thus make her or him able to embark on successful self-instruction projects in the future. Mastering Microsoft Navision use and development can be the key to giving your company the quality and efficiency that will win you and your firm a profitable and effective future.

We wish to make a special note of thanks to a few individuals and organizations whose kind and expert help has been necessary to bring this book to the public.

---

Firstly Dr. Reinald Klockenbusch whose excellent guidance and trust have been the origin and prime mover in Vieweg Fachverlag's publication of the first and second edition of this book. Secondly a loud applause is due Sabine Thiele in her lightning fast and expert translation work done for both German editions.

We would like to thank the team at TONACO GmbH for patience and dedication to ever further optimisation of their Microsoft Navision ERP system. The examples in this book are larger taken from a real and tough day in the life of TONACO GmbH. Also we wish to thank B.I.Team Softwareberatung for providing demonstration software as well as high quality and professional support.

Paul Diffenderfer and Samir El-Assal

Washington DC, May 2005





to

**ΟΜΗΡΟΣ**

*“Poet of a democratic religion”*

# table of contents

---

- 1 The ERP Philosophy**..... 1
  - 1.1 Promises of an ERP system..... 1
  - 1.2 The Potential Dangers of Implementing an ERP System ..... 3
  - 1.3 Strategies to Win the Promises and Avoid the Dangers..... 7
  
- 2 Architecture, Login and Test System Creation** ..... 11
  - 2.1 Login..... 11
  - 2.2 Creating a Microsoft Navision Test Database ..... 14
    - 2.2.1 The Benefits of Having a Test Copy ..... 14
    - 2.2.2 Learning About Your Database..... 16
    - 2.2.3 Creating the Backup Packet..... 17
    - 2.2.4 Creating a New Database “Shell” and Installing Your Backup.... 19
    - 2.2.5 Installing Your Microsoft Navision License Data ..... 20
    - 2.2.6 Importing the Database Backup ..... 22
  
- 3 The Microsoft Navision User Environment**..... 25
  - 3.1 Handling Navigation and Graphic Tools ..... 25
  - 3.2 Handling Complexity with Filtering Techniques ..... 29
    - 3.2.1 Basic Types of Information..... 29
    - 3.2.2 Two Types of Filters..... 30
    - 3.2.3 Filtering Example in the Chart of Accounts ..... 32
    - 3.2.4 Filter Options ..... 39
  - 3.3 Sorting ..... 41
  - 3.4 Implementing Menu, Filtering and Sorting Knowledge ..... 45
    - 3.4.1 Searching for the Correct Customer ..... 45
    - 3.4.2 Viewing Fields with Zoom ..... 49
    - 3.4.3 Creating a Sales Order..... 51
    - 3.4.4 Looking Deeper into the Flow Field Inventory ..... 56

<b>4</b>	<b>Introduction to Development Concepts</b> .....	61
4.1	The Challenges in Organizing Information.....	61
4.2	Organizing Information Using Table Relations .....	63
4.2.1	How Not to Organize Data: A Negative Example .....	64
4.2.2	Table Relations: Maintaining the Integrity of Your Information..	66
4.2.3	Relational Data System Example: The Sales Order .....	67
4.3	The Importance of Presentation .....	75
4.3.1	Different Views for Different Purposes and Users .....	76
4.3.2	General Ledger Account Table: Two Distinct Views .....	76
4.4	Object Designer: The Development Environment .....	87
4.4.1	Entering the Inner Structure of Microsoft Navision.....	87
4.4.2	Working With Table Objects.....	88
4.4.3	Data Type.....	90
4.4.4	Creating a New Form .....	94
4.4.5	Adding New Standard Filtering Options to a Report .....	110
<b>5</b>	<b>Creating New Flow Fields</b> .....	115
5.1	Connecting a Variable With Its History.....	115
5.2	Connecting Salespeople To Their Sales .....	115
5.2.1	Clearly and Operationally Defining 'Sales'.....	115
5.2.2	The New Sales Flow Field: Searching For the Correct Fields....	116
5.2.3	Visualizing the Table Relationships Behind the New Sales Flow Field .....	124
5.2.4	Inserting the Total Sales Flow Field in the Salesperson/Purchaser table .....	125
5.2.5	Indexing the Sales History For the New Sales Flow Field.....	132
5.2.6	Presentation of the New Flow Field.....	134
<b>6</b>	<b>Creating a New Report</b> .....	137
6.1	Creating a Hard Copy of New Sales Information .....	137
6.2	Searching For the Sources of Data .....	137
6.3	Diagram of the Table Relationships For the New Report.....	145
6.4	Diagram of Information Flow Behind the Report .....	145

6.5	Introduction to Report Designer.....	148
6.5.1	DataItem Structure .....	149
6.5.2	General Report Properties.....	156
6.5.3	Designing a Report Printout.....	158
6.5.4	Comparing the Section Designer With the Info-Flow Diagram.....	163
6.5.5	Building Output Sections .....	163
6.5.6	Creating Original Information Within the Report.....	185
6.5.7	Viewing the Report.....	214
<b>7</b>	<b>Introduction to Powerful Code Techniques.....</b>	<b>215</b>
7.1	A Practical Approach to Development.....	215
7.2	Syntax and Style.....	217
7.2.1	Dealing With the Rigor of a Programming Language .....	217
7.3	The Sentence: The Most Basic Unit of Syntax .....	220
7.4	Referring to Variables .....	221
7.5	Inserting a Value Into a Variable.....	222
7.6	Implications.....	222
7.7	Looping .....	225
7.8	Globals and C/AL Functions: Establishing Table Relations .....	226
7.8.1	Calling Foreign Table Information .....	227
7.8.2	FIND: Searching For a Specific Record in Tables.....	233
7.8.3	Representation of the Item Relationships .....	239
7.9	C/AL Calculation Functions.....	241
7.9.1	Elementary Operators.....	241
7.9.2	Incompatibility Problems with Data Type .....	242
7.9.3	POWER: Calculating Exponents .....	243
7.9.4	ABS: Using Absolute Numbers .....	244
7.9.5	ROUND: Rounding Decimals.....	244
7.9.6	CALCDATE: Calculating Dates.....	245
7.9.7	CALCFIELDS: Controlling Flow Fields with C/AL Code.....	246
7.10	Option: Special Data Type.....	253
7.11	MESSAGE and ERROR: Sending Information to the User.....	254
7.12	Data Editing Functions.....	254

<b>8 Using Dataports to export &amp; import Navision Data</b> .....	267
8.1 Simple Export from Navision.....	267
8.2 Complex Export from Navision.....	282
8.3 Creating an Import Dataport.....	289
8.3.1 Importing new records into the Item table.....	289
8.3.2 Preparing neighboring tables for Item import.....	296
<b>index</b> .....	301

# 1

## The ERP Philosophy

---

One of the most important decisions your company can make is how it will measure and manage its performance. Without good fortune and hard work—not to mention investment—realizing this goal is difficult regardless of the size of your operation. Fortunately you have purchased Microsoft Navision as well as the designer licenses needed to open and optimize its structure. Armed with Microsoft Navision's development tools such as the Application Builder, Table Designer, and Report Designer you now have a flexible and powerful state-of-the-art business information tool capable of managing and measuring your firm's performance. Next you need only the time and know-how to optimize it, that is to put it into use and adapt it to your firm's specific needs. With patience and a few key skills you can create the ultimate controlling, workflow optimization, and management system that will help make your firm the lean, mean and profit-making machine that you always envisioned.

At the outset of this book we will provide some background about Microsoft Navision as a tool which requires a brief discussion of the philosophy of Enterprise Resource Planning (ERP). Next we will look at some of the promises and dangers of ERP solutions. We discuss what strategies to take to avoid problems and what tools in Microsoft Navision are there to help you execute winning strategies, thus actualizing the promises of ERP.

### 1.1

#### Promises of an ERP system

First and foremost the ERP system is an information source. The information management must be quick, complete and correct. With these things in place, the ERP system promises you the ability to have every fact about your company's past, present and future plans and expectations—in detail and in summary—at your finger tips. Because ERP systems bring the information of various departments together you enjoy the benefit of having a real-time overview of the entire firm.

*Interdepartmental  
Information fu-  
sion*

As the ERP system connects the information of each department it can enforce a single standard throughout your entire firm. This

gives you a unique opportunity to ‘rationalize’ your company’s operations and have your corporate culture and vision reflected in the practices of each department. For example, if one of your business goals is to have same-day shipping you can set up your ERP system so that each department has information about the customer’s order even before it is their turn to work on it. This transparency of information allows each department to prepare for their part of the production process. It also facilitates communication between departments, making it easier to avoid or find solutions to problems before the product has reached their department. In this way departmental barriers can be broken down by giving everyone access to the same relevant information at the same time. This level of interdepartmental, information fusion can bring tremendous efficiency to your shipping, inventory, and customer order entry departments.

*Standardization of Operations*

The ERP system gives you the possibility of standardizing your business processes. For example, you can customize your ERP system to automatically print invoices at the moment the shipping department prints the packaging documents so there is never the question whether the customer received an invoice without the matching shipping documents. Another example is having the system automatically suggest only those ingredients which have a certain critical shelf life to be used in production thus enforcing the uniform quality of your products. This standardization of your processes and operations allows you to measure your firm’s performance more accurately because you are receiving consistent information from your departments who are working in the same manner and with the same information.

*Position training and optimization*

With improved standardization of information and processes the firm can train new workers more quickly. The optimized ERP system not only suggests to workers the next step in their task and informs them when dangerous or tricky actions are being undertaken, but also educates them about working in an orderly manner. By tailoring the system you can guide individual workers within a specific workflow that optimizes the tasks and provides built-in clarity about the expectations of the job they are performing. In this way the whole firm can reach a higher level of performance and efficiency.

*Planning tools*

The ERP system promises to optimize your company’s efficiency and some of the best tools designed towards this end are the planning tools. Imagine that in one table, for example, the buying department could see every product and every product’s components in open orders as well as their availability in the in-

ventory. This information could save workers countless foot-steps, not to mention endless back-and-forth questioning. The ERP system's interdepartmental connections optimize project management and task coordination. Efficiency and costs can be saved as it will no longer be necessary to shuffle paper between departments or make frequent trips to the refer to the archives.

*An overview without losing the details*

Another advantage of the ERP solution—which integrates firm-wide information—is that important aggregate variables (like *Total Sales* or *Total Freight Costs*) are directly linked to the specific details of which the aggregate variables are summations. For example, when looking at the *Liquidity Analysis* in the *General Ledger* matrix you can immediately drill down to individual unpaid invoices that make up the accounts payable and see exactly which are the most expensive purchases contributing to the accounts payable. Here you see that you do not have to sacrifice the everyday details to obtain a big picture of what's happening. This type of capability gives the CEO an eagle's-eye view without losing touch with the individual, everyday decisions being made throughout the firm.

Because of the interconnectedness of information the controller can cross-reference reports to check the consistency of the information recorded by each department. For example, the accounts payable reported in the *General Ledger* must be the same amount as the sum of open invoices reported by the buying department. If it is not the same, there could have been a manual booking into the *General Ledger Accounts Receivables*. Whereas, *General Ledger Accounts Receivables* should only contain system bookings. In this way you can see the inconsistencies that would have remained hidden if not for this integrated information infrastructure. Such things can easily be managed with a good ERP system.

*Results of these advantages*

Finally, all these promises—when actualized—should add up to the improved manageability and performance of your entire enterprise. With an optimized Microsoft Navision ERP system you should be able to do easily with a small team what a few years ago would have required an army of expensive managers.

## **1.2 The Potential Dangers of Implementing an ERP System**

You must keep in mind however, that without the proper marriage between your firm and your ERP solution you may experience results that are quite opposite of the expected promises. It is no overnight task to have a detailed and systematically defined



understanding of how your firm operates day-to-day. You need to understand your business to the extent that you can then strictly structure it in a software environment.

The worst possible result—one that in practice occurs all too often—is that your new ERP system decreases your firm's efficiency, frustrates your workers and gives confusing or simply incorrect information. This is all on top of the fact that the software was an expensive investment. The reality is that an ERP system is extremely complex and sensitive. In the ERP/firm marriage much work is required to adapt the software so that it becomes a mirror image of your ideal business processes.

The more knowledge you have of your firm's daily operations and the capabilities of the software, the fewer compromises will have to be made between your information processing work and your firm's general operation. While it is usually difficult to find both of these knowledge sets within the same person, it is the ideal to shoot for.

*Data entry and care*

As mentioned above, a common problem that ERP users run into is that the ERP system actually increases their workload and gives them false information. This failure occurs most frequently because ERP users do not have "clean" foundation information in the system. As a result, the system uses less than perfect information to produce false results that must be endlessly corrected. When this happens you will hear workers complain about having to continually "feed" this new and expensive tool—work that before was unnecessary. Sadly, this exact situation occurs in many firms. You **MUST** make an effort in the very beginning of your ERP installation to get your "stock" data nearly perfect. Your stock data constitutes the information that you use that rarely changes, such as chemical or product recipes or the address or bank account of a customer. If you utilize incorrect stock data you will, from day one, generate new and automatically false entries with each transaction that utilizes incorrect stock data.

*Negative example*

Let us consider an example that frequently occurs. Suppose there is a production firm that introduces Microsoft Navision as its first ERP system. Let us also suppose that the production manager of this firm has resisted the firm's innovation and has failed to embrace, understand and manage the ERP project. In particular, he has failed to invest the time needed to make 100% sure that basic information has been correctly and completely entered into Microsoft Navision. Let us further suppose that both false and in-

complete product recipes have been entered into the production module and that the ERP system uses these unapproved recipes in creating production orders. When these production orders are processed, the ERP system automatically eliminates false ingredients from the inventory and calculates false production costs along each step of the production process.

Although the production manager has always orchestrated the production without recommendations from an information system and feels no immediate pain from his poor management of the new system, his bad habits will be the source of problems for nearly the entire ERP system and all its users. This bad behavior in one causes bad business decisions to be made, creates false inventories and late shipments. Pretty soon, the entire firm has lost faith in the ERP solution. This chain of problems occurs because: the system will not calculate the correct material costs of production; the system inventory will be incorrect because the system will not be properly accounting for what is being used in the production process; the buying, shipping and the order entry departments will not be able to trust their tools which are dependent on correct inventory information, thus resulting in their inability to plan.

On top of all these problems, someone must spend several hours each week correcting the system inventory (if they want the ERP system to function at all) until the source of the problem—the false recipes in this example—is corrected. The advantage of having an information system which integrates data from each department can quickly become a disadvantage if one department fails to work cleanly.

*Out-sourcing  
programming  
work*

Another danger to be aware of is this: becoming overly-dependent on outsourced programming. Such relationships can become a vicious circle because the programmers frequently write solutions in a manner that make them inflexible in the long run. It is better for the long-term future of your firm to have someone in-house who can accurately fine-tune your system. With a little knowledge of the background development tools and capabilities, you will be able to determine the difficulty of your various development needs and judge the quality of the specialists you hire. Imagine that you need to insert the information of a second bank account so that it appears on your firm's invoice. Such tasks (adding information to reports, documents or masks) are frequently required and are not dangerous or difficult to perform. Therefore, it may not be reasonable to hire an ex-

pensive specialist when you could do the work yourself in minutes and get exactly the results you want.

*Common-practice example*

Let us consider the following example. You ask an outsourced programmer to set a default price group so that it is automatically given for each new customer. Imagine that this default price group should be: STANDARD\_CUSTOMER. To accomplish this simple task they simply program the default code, STANDARD\_CUSTOMER, into the customer table as a static and absolute text that permanently assigns this price group to new customers. The better way would have been for the programmer to create a new field, *Default Price Group*, in the price group table so that the user can—at any time—select a default price group. The first and inferior technique means that in the future, when you want to change the default, you will have to pay the programmer again to do with code what any user could have done themselves. An even worse outcome is when the option, STANDARD\_CUSTOMER, is erased from the price group table list. In this case, an error will occur stopping the creation of new customers. The computer will try to assign the text, STANDARD\_CUSTOMER, even though this option has been erased from the price group table. The following cryptic error message will appear when a user tries to create a new customer: “Customer price group code STANDARD\_CUSTOMER does not exist.” Now it is impossible to create new customers until a “specialist” comes to fix the “problem” unless someone in your company can interpret the error message.

Later we will discuss in detail how you can make these kinds of simple changes yourself and protect yourself from potential emergencies invented by specialists who may be building future work for themselves into your system.

*Price flexibility versus chaos*

An out-of-the-box ERP system like Microsoft Navision can be so flexible that if your firm is not careful, the plethora of options may foster chaos. Consider the following example: There is a standard price for each good in the product catalogue, however, this standard price applies to only the standard form and standard distribution of the good. Because of the tremendous complexity of distribution in today’s firms, an ERP system like Microsoft Navision offers several, complex sales price methods. Below are some pricing options:

- Price is linked to an individual product/customer relationship

- Price is linked to a customer group
- Price is linked to a product group
- Price is determined by discounting the standard price based on information in a third table which contains information about invoice discount

This complexity goes on. If your firm fails to have a clear strategy, it becomes difficult to find out where and on what level false price information has entered into a calculation.

*Reducing the complexity*

The ERP system is there to give the firm choices, but too many choices will lead to poor decision-making. The ERP system's flexibility provides a variety of options for many types of firms, but in practice you should limit the standard functionality to neatly fit the operations of your firm. Therefore, although many possibilities exist in the software it is best to choose one strategy for performing a task and then block or hide (using the software development tools which we will discuss in detail later) the alternatives.

*Adapting ERP to your firm, & not the reverse*

One of the biggest dangers of any software is that the firm will adapt to the software instead of the software adapting to the firm. In practice, the former happens quite often. It is frequently the case that because of a minor function or lack of a minor function in the software, the firm must change its workflow to follow an inefficient alternative just to support the larger interests of the software's operation. Here is an actual example: To stop an order from being partially shipped before it is finished being entered into the system, the shipping department and the order department must work in shifts, which means that products can rarely be shipped in the same day. This absurdity occurs simply because the firm did not have the know-how and development tools to put a key into the software which distinguishes between "finished orders" and "orders not yet completely entered." This book and the development tools within Microsoft Navision will help you to avoid some of these absurd situations.

## 1.3

### **Strategies to Win the Promises and Avoid the Dangers**

Attitude: Never assume there is no better way of doing something. The key to success in optimizing your ERP solution is committing to a plan of continual improvement and fine-tuning. With Microsoft Navision and its development tools you have the perfect ERP platform for adapting an expansion ERP solution to

achieve all the promises previously stated and avoid the dangers that follow.

*Prerequisites*

The following steps are critical for attaining the goals of a successful ERP solution:

- Define your business operations clearly and concretely, in a logical if/then style
- Define standard naming conventions and cataloguing conventions
- Gain control of the programmability of your software

An effective way to begin defining the structure of your business operations is to use a good graphic workflow design tool like Microsoft Visio. With Visio you can quickly create diagrams that guide you along your path of program development to achieve solutions that will be easy to implement when you begin adapting your Microsoft Navision software. This book will suggest concepts and examples of techniques that will help you to realize these workflow diagrams in the software.

*Feedback from your workers*

In addition to mapping your firm's activities and familiarizing yourself with some development techniques, you will need continual support and feedback from your workers who must use the ERP on daily basis. It is helpful to foster a firmwide understanding that improvements require time, testing, feedback and training—a long but profitable process. This is the road to creating a truly comfortable and efficient information infrastructure that reflects the best intelligence of your firm.

Einstein is supposed to have remarked, "Today's problems were yesterday's solutions." Many times what you have used to meet the needs of one user will indirectly create a problem for another. Due to the interconnectedness of a complex ERP software, it is always difficult to see how fixing a problem in one area will affect other seemingly distant functions in the software. This is why it is important to have a complete test version where you can check the effects of your new ideas (How to create a test version will be discussed in detail later). Everyone must work together to create the system that they will all be using together. As you move forward, have patience, because regardless of the promises of the ERP salesperson, no software can immediately—out-of-the-box—optimize your firm.

*Continuous-improvement*

Microsoft Navision is the winning ERP choice for small to mid-sized companies looking to avoid the dangers and achieve the promises of the ERP philosophy. This is largely due to years of real world testing and continual improvement of the base Microsoft Navision software. The software is supported by a large network of experienced specialists who have helped more than 15,000 firms with the installation and adaptation of the Microsoft Navision system. This is a history you can trust. However, it does not mean the software is perfect. As the needs of the business world change, so must the software. A tool as complex and powerful as an ERP system is like a living system—ever growing, changing and always offering new alternatives to outmoded practices.

Microsoft Navision has out-of-the-box features that put tremendously powerful and flexible information-gathering tools like *Flow Filters*—which eliminate the need of literally hundreds of special reports—at the user’s fingertips.

Microsoft Navision’s powerful and intuitive application and function development environments make adapting as well as creating new fields, table views, forms and reports easy. If you are familiar with Microsoft Access, working in the Microsoft Navision environment will not be difficult.

An important and powerful feature of Microsoft Navision is the ease with which you can create a fully operational copy or copies of your Microsoft Navision system. This is critical for testing Microsoft Navision functions or newly programmed applications or reports or tricky new techniques before integrating it into your firm’s precious Microsoft Navision database. A copy can also be a stress-free environment in which to learn about the Microsoft Navision system without the fear of making a mistake.

Finally, with the help of this book, the extensive Microsoft Navision help files, documentation and the network of experienced specialists, you will have everything you need to optimize your installation of Microsoft Navision.

# 2

## Architecture, Login and Test System Creation

---

In this chapter we will shed some light on the background of the Microsoft Navision system layout and show you how to locate, login and copy the Microsoft Navision database.

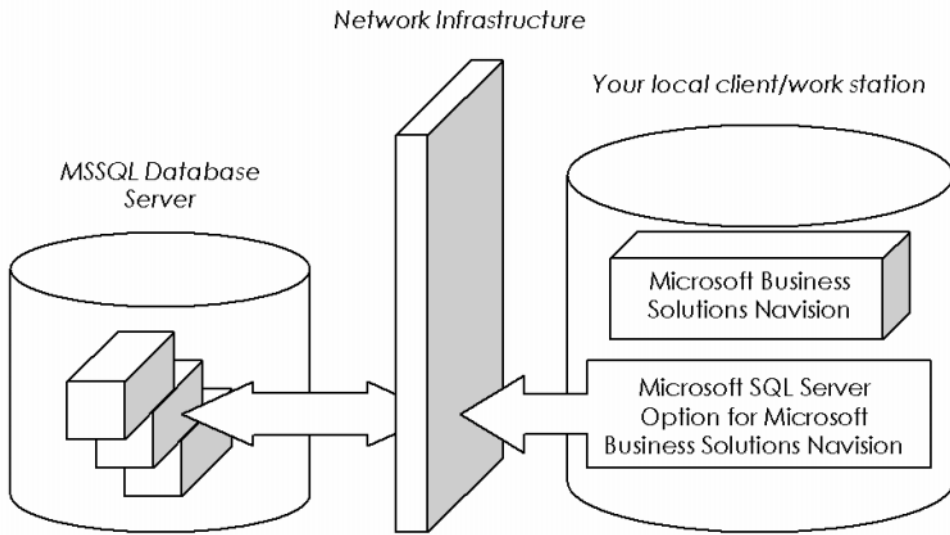
### 2.1

#### Login

In practice, most firms have one server, one Microsoft Navision database on that server and one company on that database. If you are using a single Microsoft Navision company, you can simply use the login window that appears automatically when Microsoft Navision is opened. You can forget about the background complexity of connecting to a specific server, database and Microsoft Navision company. The login window that automatically appears only connects the user to the most recently accessed Microsoft Navision database and company. This is not the environment that you want to work in as you learn to master the Microsoft Navision environment. Therefore, we **strongly** recommend that you create a second database and use it as a test copy of your Microsoft Navision database. For this reason it is best to cancel your automatic login window so that you learn how to establish these connections manually.

#### *Manual Login*

Logging in manually is more complex than the login you may be used to in other simpler software packages. The reason for this is that Microsoft Navision gives the user many options to choose from such as: whether to use a database located on the desktop client machine, use a database on a remote server and the choice of which Microsoft Navision company within each of these various databases to use. Here is a diagram to help you visualize the Server/Client and Database/Company architecture:



**Figure 2.1** Server/Client & Database/Company architecture

There are four steps to complete when you manually log onto your desired Microsoft Navision company:

1. Whether to open Microsoft Navision's Client version **or** Microsoft Navision's Microsoft SQL Server version
2. Choose the server where your desired database and Microsoft Navision company exist (this step is only necessary if you have chosen to use the Microsoft Navision Microsoft SQL Server version in step 1)
3. Enter your username and password
4. Choose which company in the database you want to open

*To work locally or via the network?*

The first step is to choose whether you want to use a database that is contained within your client PC (located on your client local hard-drive) or a database located on a remote sever. Normally, you will use the server edition of Microsoft Navision if you have a central Microsoft Navision database that is used simultaneously by many workers. In this book we discuss the Client and MS SQL server editions of Microsoft Navision as they are

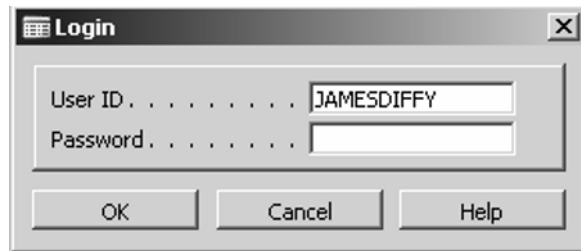


the most common types of Microsoft Navision installations. These versions of the software are located at:

- **Start > Programs > Microsoft Business Solutions-Microsoft Navision > Microsoft Business Solutions-Microsoft Navision** (for the client version)
- **Start > Programs > Microsoft Business Solutions > Microsoft Business Solutions-Microsoft Navision SQL server option** (for the MSSQL version)

*Finding the right Microsoft Navision database*

Next, a *Login* window will appear where you enter your *User ID* and *Password*.



**Figure 2.2** *Login* window

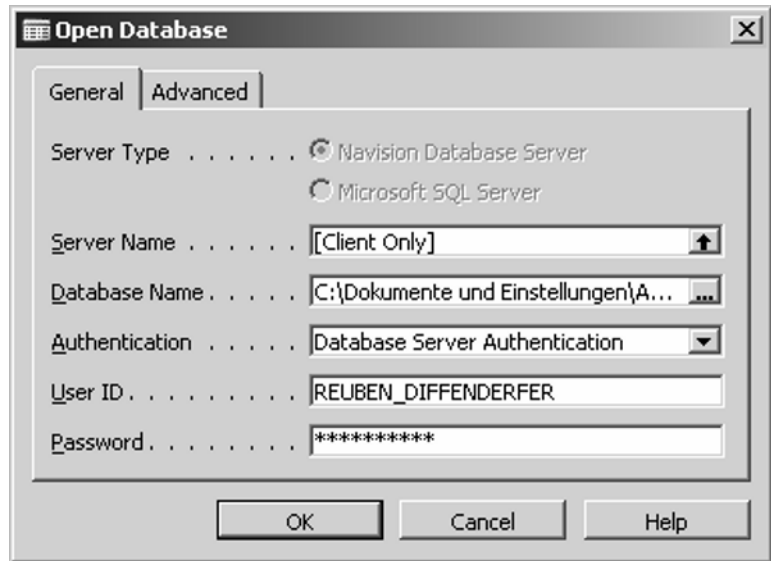
This *Login* window will automatically access the most recently used database and company which are recorded as ZUP-data in the client user information file. If you are using only one database and one company in that database, you may always use this login window and forget the background connection it automatically implies. However, we will be using at least two different databases in this book: your original “live” Microsoft Navision system and a test copy. Therefore, to choose the desired database, first cancel the automatic *Login* window (if one has automatically appeared) and go to:

- **File > Database > Open** (you may have to repeat this step twice)

Next, the following window will appear: (See Figure 2.3).

First, you must choose your *Server Name*. In the client version, [Client Only] will appear. This is fine if your desired database is located locally on the client hard drive. We recommend that the original installation of Microsoft Navision and your Microsoft SQL server be set up by your solution center as these are activities performed only once and may require the experience of a specialist. Next, you must enter your *User ID* and *Password* before the server will allow you to enter anything into the *Database*

Name field. Choose which database you want to use and click OK.



**Figure 2.3** *Open Database* window

Next, enter your *User ID* and *Password* and then click on the *Database Name* field.

A help arrow will appear in the white space of this field. Find the database you want to use and click OK. Next, you must choose which company in the database you want to work with. Go to:

- **File > Company > Open**

## 2.2 Creating a Microsoft Navision Test Database

Now we will show you how to create a fully-operational copy of your Microsoft Navision system.

### 2.2.1 The Benefits of Having a Test Copy

*Risk-minimization*

You can attain the promise of adaptability and flexibility in your Microsoft Navision ERP system because it provides you with tools for copying and securing your firm's active system. As you begin creating the information infrastructure that will literally mirror your firm, you will have to do a lot of testing and proceed by trial and error. These necessary development steps are risky

to test in your active Microsoft Navision system. Therefore, in the next section we will show you how to create a test version of your company's Microsoft Navision ERP and eliminate all risks of trial and error development.

*... in an identical environment*

Although it may seem like an advanced topic, it will be of great use to know at an early stage how to make a complete and working test copy of your active Microsoft Navision ERP system. After you have created a test version you can try the concepts discussed in this book in a familiar and safe Microsoft Navision environment.

With a complete operational copy of your Microsoft Navision system you will enjoy many new conveniences. If you are going on a trip and want to take your company's Microsoft Navision with you, simply install it on your laptop. You can also use the following techniques to create a security backup. A Microsoft Navision backup system for testing, learning and exploring will be critical in achieving your ERP goals, but is sadly a option that many firms fail to take advantage of.

*First test, then implement*

It is a good practice to test all new applications, reports, forms and methods in a fully-functional test copy of your firm's Microsoft Navision system before copying and implementing them into your firm's active system. Likewise, all outsourced programmers who work on your Microsoft Navision system should do so in a test copy before implementing their work into your active system.

We recommend that you create a client-based version test copy of the Microsoft Navision system as opposed to creating a test version on your server. A client-based test system is preferable for the following reasons:

- A server-based copy may use precious server resources, causing too much network traffic which could decrease your network performance
- A server-based copy could be logged onto by accident and used without the worker noticing they are not in the company's "real" or active database
- It may be a security risk while the copy contains all the company's financial, customer, etc. information and may not have the security set up in exactly the way the company's active Microsoft Navision database is set up

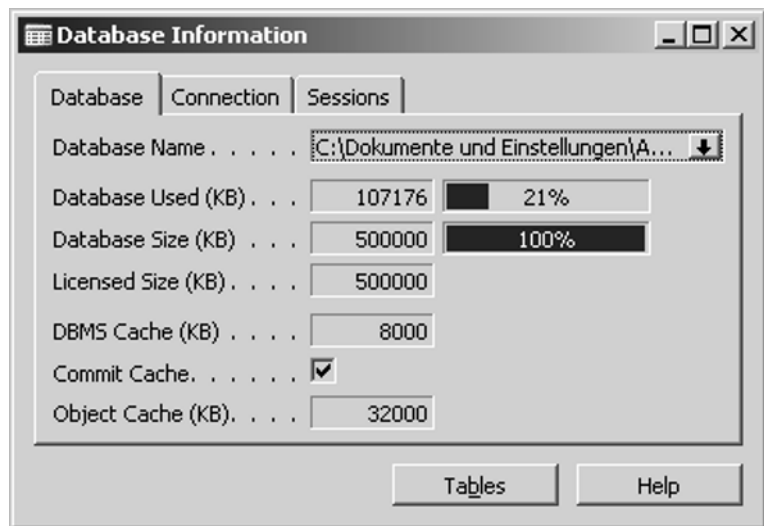
### 2.2.2 Learning About Your Database

To help you visualize the following process, imagine that there are at least two layers to your Microsoft Navision system. There is the database itself which is like a shell within which you insert your company's specific information. The first step is to create this "shell" and secondly to insert your company's information into it. The "shell" must be big enough to hold your company's mass of information.

The first step in creating a copy of your company's information is to assemble information about the database you wish to copy. After you have logged on and opened your desired Microsoft Navision database, go to:

- **File > Database > Information**

Here you will see general information about your Microsoft Navision database.



**Figure 2.4** Database Information window

You will need to make a note concerning the size of the database you wish to copy which can be seen in the *Database Used (KB)* field. You need this information to determine how large to make your new database as well as to know how much disk space is needed for the database backup.

*Kick out other users when backingup!!*

Next, you must make sure that you alone are logged onto your desired database during the time you are creating a backup. You can do this by checking the second page of the Database Information window called *Sessions*.

Here you will find the *Current Sessions* field. The number '1' should appear in this field. If it does not, there are other users in the firm's Microsoft Navision database which could disrupt your backup process. To find out who is "disrupting" the backup, click on the *Current Sessions* field and then on the arrow that appears in it. A detailed list will appear with all the Microsoft Navision activity taking place on your server at that moment.

Make sure the users editing the data are logged off the system that you want to copy. If someone is working in the system while you are making a copy, you may lose the most current information, which means the copy will have errors in it. It is best to make your database backup in the evening when the network is not in use. To create the database backup you will need as much data storage space as the number of kilobytes indicated in the *Database Used (KB)* field. If you do not have enough storage space to work with both the database backup and the new database on your local or server storage disk (2.2 times the size of the database size indicated in the *Database Information* window), then you can create your database backup and save it to a compact disc (CD). After your new database is created, import the database backup into your database from your CD.

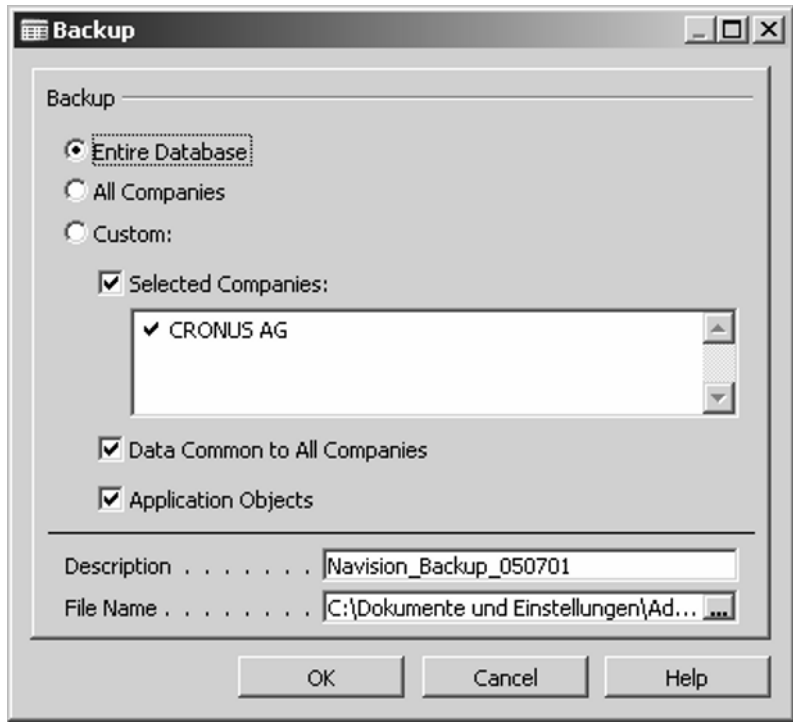
### 2.2.3

#### **Creating the Backup Packet**

Continue by going to:

- **Tools > Backup...**

The following window will appear:



**Figure 2.5** Database *Backup* window

Select the *Entire Database* box as well as the other options shown in Figure 2.6. This ensures that a complete copy is made. Fill in the *Description* field. We suggest that you choose a naming convention for the new database. That said, the same name should then be used for the *File Name* field and later in the new database name. The name you choose should begin with the date on which the database backup was created. This date should be written with the year first, then the month and lastly the day. Using this method ensures that all backups and databases will be sorted in chronological order on any file directory. The next part of your new database name should be: *Microsoft Navision*, then *Test* or *Backup*. Lastly, every word should be separated with an underscore ( ).

It is recommended throughout the book that you avoid the inclusion of empty spaces in the names of all objects and data names. Following this rule will make it easier to organize your data in the long run. Likewise, it will be easier to use your in-

formation with other types of software which often have problems converting empty spaces.

After filling in the *Backup* window click OK and wait until the message appears that tells you your Microsoft Navision backup was successfully created. This could take some time depending on the size of your database.

Once you have created your database file you have a complete and compressed copy of your firm's entire Microsoft Navision system and information, current up to the moment it was created. Now you have a test version or security backup of your system. It is wise to make a database backup periodically and store it offsite—a step that could prove critical if something unforeseen were to happen to your firm or its information systems.

## 2.2.4 Creating a New Database “Shell” and Installing Your Backup

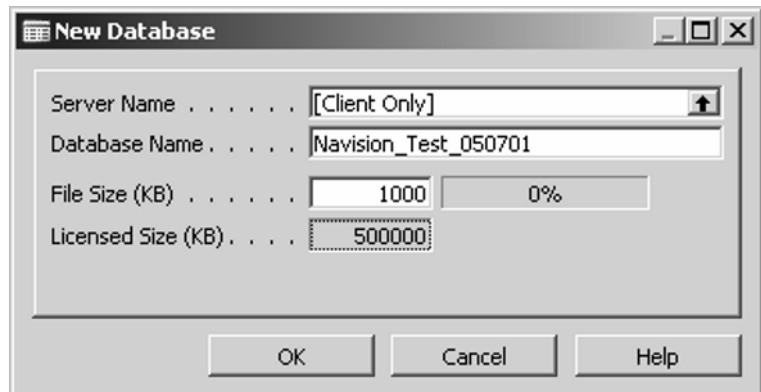
Let us continue creating the new test system. Completely close your network-based Microsoft Navision program and open the local version of Microsoft Navision located at:

- **Start > Programs > Microsoft Business Solutions-Microsoft Navision > Microsoft Business Solutions-Microsoft Navision**

Cancel any login window that comes up automatically, then go to:

- **File > Database > New**

The following window will appear:



**Figure 2.6** *New Database* window

Notice the value written in the field, *Licensed Size (KB)*. Microsoft Navision is designed so that everything the user does is lim-

ited by the rights your company purchased from your Microsoft Business Solutions Center. These various rights are contained in a license file that must be imported into your Microsoft Navision system. Within this license you will find the parameters for the size of database your company is allowed to use. In order to create a copy of your Microsoft Navision system you may need to reimport this license file into Microsoft Navision before you are able to install your database copy into it. Therefore, if the *Licensed Size (KB)* value is smaller than the *Database Used (KB)* size of the database which you want to recreate, then you will need to import your firm's special and unique license into the Microsoft Navision program before you can continue. On the contrary, if the value here is sufficiently large you can skip the next section, "Installing your Microsoft Navision License data" and continue to the next step.

### 2.2.5 Installing Your Microsoft Navision License Data

The Microsoft Navision license file is the key that unlocks the functions of the Microsoft Navision environment. It contains all the rights for the functions of Microsoft Navision that your company has purchased. Your unique Microsoft Navision license will not only allow you to increase the size of your new database, but also give you access to all the various modules and development tools you have purchased from your Microsoft Business Solutions Center.

The license file determines the size of database you are allowed to create. When you first open your database a demonstration license may be in it. This will cause the *License Size (KB)* value to be too small. Therefore, you must upload your firm's special and unique Microsoft Navision license into the software. Importing your license is easy to do. If you do not have this license data you must get a copy of it from your Microsoft Business Solutions Center.

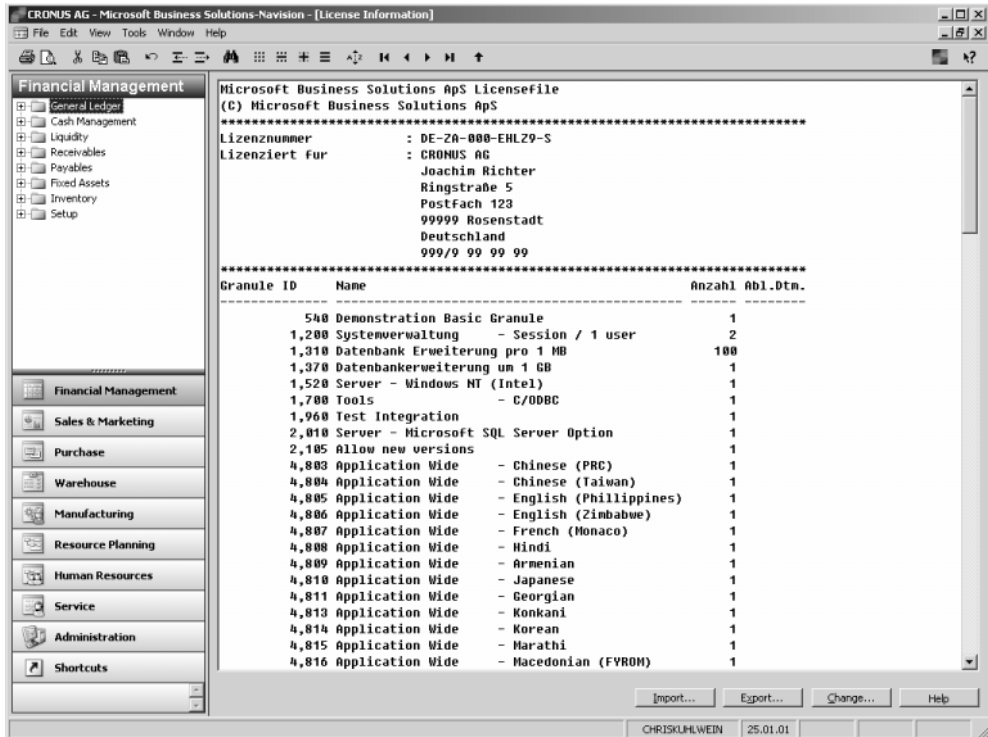
The license data is the key to using your entire Microsoft Navision system, therefore it should be handled with care and securely stored when not in use. If you need a copy of it you may be able to find it on your server by conducting a serverwide search for files whose names end with the file extension, *\*.lff*. If you cannot find it using this method, call your Solution Center and they can easily send it as an attachment in an email. Once you have a copy of your Microsoft Navision license you will



need to import it into your Microsoft Navision program. Proceed to:

- **Tools > License information**

The following window will appear:



**Figure 2.7** License Information window

Next, click on Import to locate and open your firm's license data. The information in the license file should immediately be updated to reflect all specific rights which your firm has purchased from your Microsoft Business Solutions Center. Now you can continue creating a client copy of your complete Microsoft Navision system with all the functionality of your firm's active Microsoft Navision system.

When your license data has been imported and the value in the field *Licensed Size (KB)* is great enough to install your database, enter the number of kilobytes that you will need for your database into the field *File Size (KB)*. Determine this size by the size of the original "live" database, multiplied by a factor of 1.1. Type in the same name that you used for the database backup into the

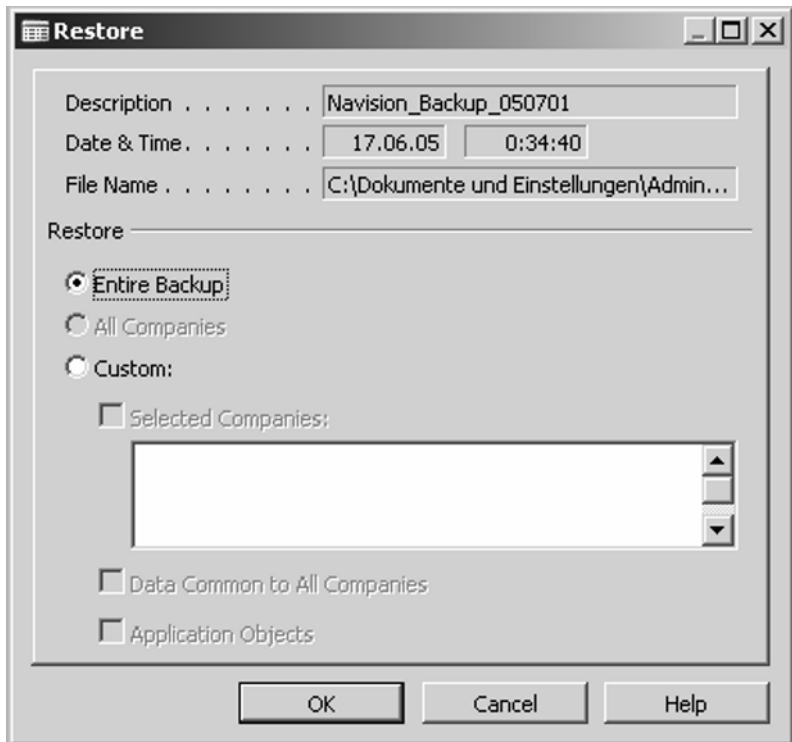
*Database Name* field and press ENTER. You must wait while Microsoft Navision creates the new database, which may take a few minutes to complete depending on the size of your database.

### 2.2.6 Importing the Database Backup

When Microsoft Navision has successfully created the new database all that is left to do is to import your database backup file into you new but empty database. Proceed to:

- **Tools > Restore**

Locate and open your backup file. The following window will appear:



**Figure 2.7.1** *Restore* window

Select the field, *Entire Backup* to be imported. Microsoft Navision will now read into your new test database the copy of your firm’s Microsoft Navision system contained within the database backup. This may take some time depending on the size of your database.

When Microsoft Navision gives you the message that it has successfully imported the database backup, click OK and then open and proceed to:

- **File > Company > Open**

Select your desired company and click OK.

Your new company will have the same name as the original. We recommend that you change this name to minimize confusion between your active Microsoft Navision system and your copy. Proceed to:

- **File > Company > Rename**

Name your new company according to the same naming description referred to above and then click OK. It may take several minutes for Microsoft Navision to rename the Company. When this is completed you will have a perfect copy of your firm's entire Microsoft Navision system. Now you can play in the system and not worry about making mistakes.

# 3

## The Microsoft Navision User Environment

---

In this chapter we discuss the most important tools in the Microsoft Navision end user environment. You will learn how to master the information control tools which includes navigating menus, using graphic tools, filtering and sorting. We will practice using these tools and techniques in numerous examples throughout the book.

### 3.1

*Curiosity is the best teacher*

### Handling Navigation and Graphic Tools

The Microsoft Navision desktop environment uses a standard Windows application layout. The important user interfaces in each module are:

- File cards and table views where information is inserted, viewed and edit information
- Posting tables where transactions are entered and processed
- Reports where information is viewed, processed and printed
- Module setup where Microsoft Navision-wide default values, properties and attitudes are set up and defined

As with learning any complex software, it will take some time for the user to know how and where to get the best information the quickest. In Microsoft Navision there are many places to find the same information and many different views of the same information. We encourage users to be curious and willing to look around every corner. Microsoft Navision is a true ERP solution; it connects almost every various aspect of the business with each other and even seemingly distant parts of the Microsoft Navision environment may be linked. As a result, there are many shortcuts if you are clever enough to find them. It is easy to create shortcuts in the software which make things more convenient for the user. We will show such techniques in more detail later in the book.

*Output types*

You can see information from your Microsoft Navision system as a printout or viewable report, in a file card display, in a table view list format or in output that has been exported to another program like Microsoft Excel. The file cards are helpful if you want to see many different types of information concerning one item all at once. An example would be if you want to see everything related to one of your customers in a single table view. Table view lists are really helpful when you want to see simultaneously how a single variable or category compares between many different items, such as comparing the total sales for all your sales products at the same time. Reports are critical for obtaining information that is too complex to find in a single place or that includes difficult calculations. Most reports are designed to generate physical documents. However, you can also view exported data in Microsoft Excel, leveraging all the graphic and presentation tools that Excel has to offer.

*Double goal in learning environment*

It is important to develop an understanding of the Microsoft Navision end user environment, not only for your use of Microsoft Navision but also as a foundation for your education as a developer. It does not take much to master the Microsoft Navision end user environment since the same handful of powerful tools are applicable throughout the Microsoft Navision. This understanding will be the basis for a powerful intuition that will guide you in learning some simple programming techniques and creating new possibilities within the software. Having this knowledge will help you tailor your Microsoft Navision information infrastructure so that you can begin optimizing your firm.

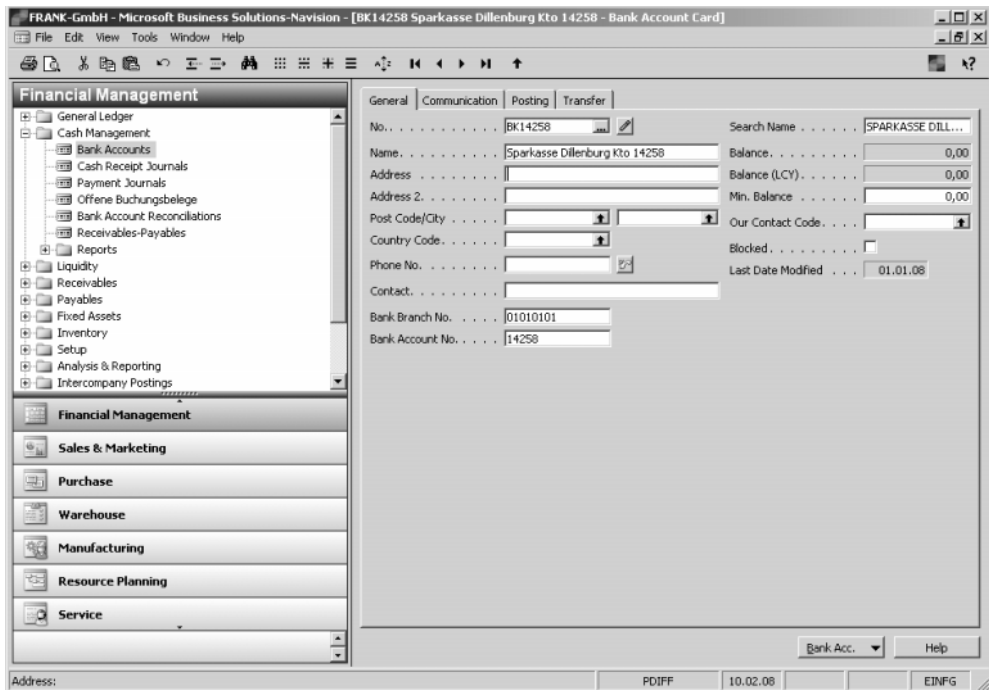
To begin your training in navigating Microsoft Navision, it is good practice to stop and make a mental or visual map of your task before you try anything. It is often tempting to begin by clicking around until you see something you can use, when in reality there is usually one location for the information you need to answer a given question. By forming a mental map you can check your expectations and test to see if you really know what's going on and where you are in your Microsoft Navision environment.

Let us make a quick overview of the basic end user tools in the Microsoft Navision working environment. For an excellent treatment of these overview matters you can also refer to the standard Microsoft Navision introductory documentation which is on the demonstration CD.

We will begin with the background working area in Microsoft Navision. Moving from left to right we will name and discuss the special horizontal Microsoft Navision toolbar located at the top of your Microsoft Navision working area just below the program menu containing File, Edit and so forth.

*Print (Ctrl+P)* This button only functions when you are viewing a printable report.

*Page View* This button functions to execute a viewable report.



**Figure 3.1** Work environment and menus

*Cut (Ctrl+X)* With this function you can remove and save information selected from within fields (or many selected records simultaneously) to the Windows clipboard for pasting into Microsoft Navision or other applications like Microsoft Excel.

*Copy (Ctrl+C)* This functions just like the Cut function except that it does not remove the information it copies to the clipboard.

*Undo (Ctrl+Z)* This function will reverse your action in a field before you have left that field (ESC will do the same).

<i>Paste (Ctrl+V)</i>	This function enters contents to the clipboard into the area you have selected.
<i>Insert (F3)</i>	This function will open a new, empty record into a table.
<i>Delete (F4)</i>	This function deletes a selected record from a table.
<i>Search (Ctrl+F)</i>	This function opens the search, find and replace window.
<i>Field Filter (F7)</i>	With this function you can limit what records you retrieve from the database with Microsoft Navision filter keys. (We will go into great detail about this important tool later).
<i>Table Filter (Ctrl+F7)</i>	With this function you can view, add to and/or remove all the filters in a table that limit the records Microsoft Navision can retrieve.
<i>Flow Filter (Shift+F7)</i>	This function shows which filters determine the values in calculated fields, such as an account balance or total sales.
<i>Sorting (Ctrl+F8)</i>	This button opens the <i>Sorting</i> window and shows you what options are available to sort the records in a table. (Later we will discuss how you can add to this list of options).
<i>First, Previous, Next, Last</i>	These buttons allow you to toggle through the records in a table.
<i>List (F5)</i>	<p>This button displays the table view of the file card you are currently in. If the file card does not have a table view it is possible to create one.</p> <p>Some of these tools we will discuss in great detail, however, many are standard for all Windows applications, such as Cut, Copy and Paste. Several are simply self-explanatory, like the buttons that move you through the records in a table. The most difficult and perhaps most important functions in the whole Microsoft Navision environment are the filtering tools, which are discussed throughout this book. Developing intuition and skill in using these tools will make you a better end user and help you to become a solid developer and Microsoft Navision troubleshooter.</p> <p>It is recommended that you change the data in the setup areas only when you are sure of the implications of such a change or have tested the same idea in your most recent Microsoft Navision test copy as previously discussed.</p>
<i>Help (F1)</i>	When you have a question about a Microsoft Navision feature you can select the object and press the F1 function button on your keyboard. Not every feature contains help in the documentation, however, most are covered.

## 3.2 Handling Complexity with Filtering Techniques

The ERP world promises information control more than anything else. You must be able to get an overview that is connected to the details. You must be able to get to information and get to it quickly.

Attaining the promise of information control requires some skill and experience—largely due to the overall complexity and sheer mass of the information as well as the diversity of information centers within the software.

Great complexity exists in such an expansive tool as Microsoft Navision because it performs so many different interconnected tasks simultaneously. For example, the simple posting of a customer invoice will initiate the following background operations:

- The customer's history is updated
- The inventory is updated
- The Sales and Sales Tax are posted in the *Chart of Accounts*
- The inventory value is adjusted in the *Chart of Accounts*
- The salesperson's commission is posted

### 3.2.1 Basic Types of Information

To deal with the complexity of the information itself you should recognize the different types of information making up the ERP system. There are three basic types of information which are important:

- Stock data
- Transaction data
- Calculated data

#### *Stock data*

Stock data is the type of data that, in principle, should never change. The name and tax ID of your firm for example are types of stock data. Another example would be the names and account numbers of your *Chart of Accounts*. Stock data is the information backbone of the daily business and should rarely change. In Microsoft Navision this type of information is usually presented in the file card type view, like the customer or item cards for example.

#### *Transaction data*

Transaction data is the information type that records the history of your daily business activities. It is called transaction data be-



cause it is continually growing with every movement of your accounts and is usually presented in a table view. The table view is excellent for transaction data because it shows you many records in a chronological list which helps you see the timeline of movements. Examples of transaction data in Microsoft Navision include: the *Chart of Accounts* postings, the inventory postings and customer payments.

#### *Calculated data*

Calculated data is where stock and transaction data types meet—where we find the sum of changes or movements associated with a standard product. For example, in the *Item Card* where the inventory movements of an item are shown in the *Inventory* field. We find another calculation field in the *Customer Card* where the customer's actual account balance (the sum of all payments and charges) is shown. This type of data is always current and actual in Microsoft Navision and can be found everywhere, including table views and file cards. When you click on such a field and the arrow that appears, the information that Microsoft Navision used to arrive at its calculation is immediately opened.

Understanding the depth and breadth of Microsoft Navision's functions requires experience. Knowing where to find things in a quick and efficient manner comes with use and study. Although Microsoft Navision is laid out in a consistent and logical manner, it is difficult to navigate due to its size, which means you will not be able to get an overview of your firm's performance overnight. Therefore, handling the sheer mass of the data in Microsoft Navision is the subject of the next two sections. A firm grasp of the next two sections will give you an excellent foundation in your pursuit to control your systems information.

The filtering tools are designed to help you handle the sheer mass of data in your system. The primary goal of the filters is to define limits on the set of records Microsoft Navision accesses from the database.

### **3.2.2 Two Types of Filters**

Some of the most important tools in Microsoft Navision are the filters. Using filters will be critical whether you are an end user or someone who is interested in adapting Microsoft Navision to your firm's special needs. We will look at many practical examples of filtering techniques in Microsoft Navision which will help you to get the most out of your system and see how Microsoft Navision is working in the background. You will quickly see that

on some level, almost everything that must happen in the software requires some filtering of the tables. Filtering allows you to get the data and connections that you need for your applications and studies.

After we help you visualize setting filters on data in various tables, we will move on and show you how to do what you had been doing manually with programming code. Working in this order means you are less likely to get lost in the abstractness of the programming code because you already have a visualization of doing the same thing manually. Having a knowledge of both methods gives you the chance to cross check the results of your programming with the results you would achieve if you had performed the same task manually.

*Limiting the data flow*

The primary function of filters is to limit data—that is, the records which will be accessed from a table within a database. More simply put, through the function of a filter, we get only what we want out of our data base, such as records within a specific category, records with certain values, or any combination of these conditions.

There are basically two types of filters in Microsoft Navision:

- Table Filter
- Flow Filter

*Table Filter*

The *Table Filter* is simple to understand: it filters on information found in any one given table. A *Table Filter* can be used, for example, to limit the customers shown in the *Customer Card* to include only those customers located in California, and only customers that have greater than \$10,000 in sales in a given year. Both the *Field Filter* and *Table Filter* tools can be used to define a *Table Filter*, the difference is that the *Field Filter* shows only a part of the filtering process—that is, a single field at a time.

**\* It is highly recommended by the authors that you use the *Table Filter* as opposed to the *Field Filter*. If you use the *Table Filter* you will see all of the filters that are set on a table simultaneously. On the contrary, the *Field Filter* will show you one field at a time—the field which your cursor is presently on. Not knowing which filters are acting on a table at any one time can produce incorrect results and undesirable mysteries.**

*Flow Filter*

The second type of filter, the *Flow Filter* is more complex than the *Table Filter*. *Flow Filters* control the evaluation of calculated

fields in Microsoft Navision. These filters limit the set of information taken from a foreign table to be evaluated and then entered into a calculated field. In Microsoft Navision these special calculated fields, *Flow Fields*, are influenced by *Flow Filters*. Examples of *Flow Fields* include: *Inventory*, *Net Change* and *Balance*. The values of such fields are always current, always subject to change and can differ based on what *Flow Filters* have been set on them. Examples of *Flow Filters* that you can use to determine these *Flow Fields* are: *Date Filter*, *Department Filter*, *Project Filter*, *Budget Filter* and the *Dimension Flow Filters*.

### 3.2.3 Filtering Example in the Chart of Accounts

Now let us consider an example that will require the use of both types of filters. Go to:

- **Financial Management > General Ledger > Chart of Accounts**

The following window will appear:

No.	Name	I...	A...	Totaling	G..	G..	G..	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..		P..	N..	M..	147.73	147.73
1130	Decreases during the Year	B..	P..		S..	N..	M..		
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
1200	Operating Equipment	B..	B..						
1210	Operating Equipment	B..	P..					582,872.18	582,872.18
1220	Increases during the Year	B..	P..		P..	N..	M..	25,116.00	25,116.00
1230	Decreases during the Year	B..	P..		S..	N..	M..		
1240	Accum. Depr., Oper. Equip.	B..	P..					-508,176.74	-508,176.74
1290	Operating Equipment, Total	B..	E..	1200..1290				99,811.44	99,811.44
1300	Vehicles	B..	B..						
1310	Vehicles	B..	P..					49,473.91	49,473.91
1320	Increases during the Year	B..	P..		P..	N..	M..	87,000.00	87,000.00
1330	Decreases during the Year	B..	P..		S..	N..	M..		
1340	Accum. Depreciation, Vehicles	B..	P..					-60,603.78	-60,603.78
1390	Vehicles, Total	B..	E..	1300..1390				75,870.13	75,870.13
1395	Tangible Fixed Assets, Total	B..	E..	1005..1395				1,128,689.52	1,128,689.52
1999	Fixed Assets, Total	B..	E..	1003..1999				1,128,689.52	1,128,689.52
2000	Current Assets	B..	B..						

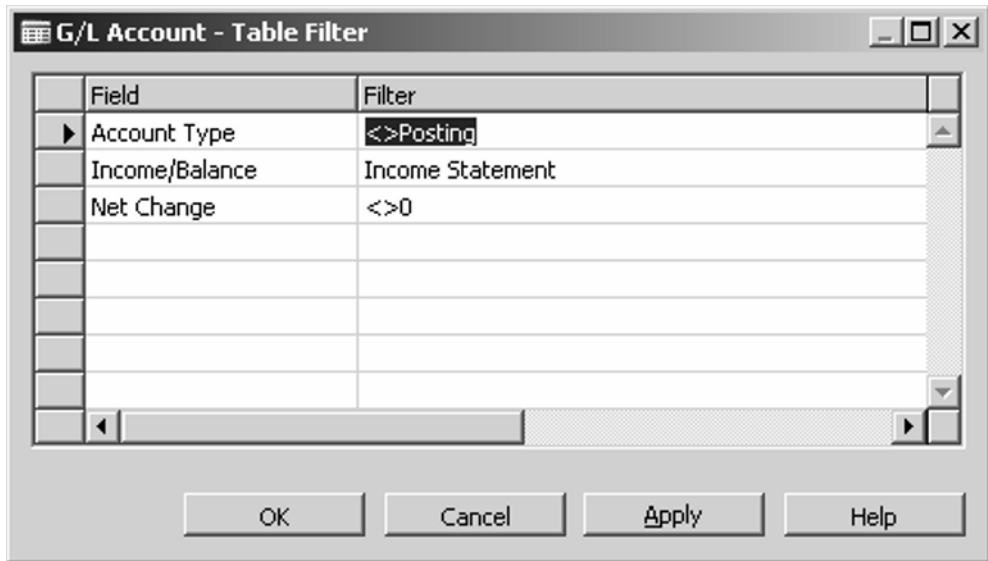
Figure 3.2 Chart of Accounts

Here we see the financial information pertaining to the entire firm. In this view you see every header text, account and account sum. In the columns you see not only all the account numbers and account names but also the financial sums in the *Net Change* and *Balance* fields on the right. This information is presented in a table view form based on the *G/L Account* table (later we will discuss how forms are really just outward presentations of deeper, more fundamental tables stored in the database.)

You may ask the question how these financial sums—figures which are perpetually changing—can be “written” into the database? Must Microsoft Navision rewrite the data in the *G/L Account* table every time these sums change? The answer is that these sums, *Net Change* and *Account Balance*, are merely displayed and not actually written into the database (this is why you cannot sort a list by such a calculated field). Rather, these values are calculated from scratch every time the user calls them. Such fields are examples of *Flow Fields*, which are built from a formula written into the definition of the *Flow Field* itself and their value is determined by summing the set of information from another table. In this example the *Net Change* is a *Flow-Field* in the *G/L Account* table which sums the *Amount* column from a set of records in the *G/L Entry* table. Here is where the *Flow Filters* come into use: they determine exactly what set of G/L Entries will be summed. Now to continue to our example, let us consider that following task.

#### *Example*

Suppose your boss comes to your desk and wants a quick answer concerning the company’s profit in January 2001. You need to be able to explain the results utilizing data from individual transaction. To do this, you will need to see a short summarized list of accounts showing only the income/cost accounts. Likewise, you’ll need to be able to go into these sums and see exactly what details are behind them as the boss believes there may be some very implausible values. Here is what you can do:



**Figure 3.3** *G/L Account - Table Filter* window

First, you must control what Microsoft Navision accesses in the *G/L Account* table: account names, codes and types, for example. To limit Microsoft Navision's access into a table, you must use a *Table Filter*. Go to:

- **View > Table Filter** (Ctrl+F7)

Enter the texts that are shown above (Figure 3.3) in the *Filter* column.

The characters < > signify "unequal to" within the filter function. What we have done now is to give Microsoft Navision a command that it: Only access accounts in the *G/L Account* table where *Account Type* is not equal to *Posting*; only access accounts that are defined as *Income Statement* (that is the *Income/Balance* option is only equal to *Income Statement*); and lastly, only access accounts where *Net Change* is **not** equal to zero. This tremendously shortens the list of *Chart of Accounts* and gives you only the accounts which are sums and have some activity. Next, click OK. The following window will appear:

No.	Name	I...	A...	Totaling	G...	G...	G...	Net Change	Balance
6195	Total Sales of Retail	I...	E..	6105..6195				-983,351.00	-983,351.00
6295	Total Sales of Raw Mater...	I...	E..	6205..6295				-5,848,247.47	-5,848,247.47
6495	Total Sales of Resources	I...	E..	6405..6495				-24,837.00	-24,837.00
6995	Total Revenue	I...	E..	6100..6995				-7,056,468.52	-7,056,468.52
7195	Total Cost of Retail	I...	E..	7105..7195				837,430.70	837,430.70
7295	Total Cost of Raw Materi...	I...	E..	7205..7295				2,994,812.39	2,994,812.39
7995	Total Cost	I...	E..	7100..7995				3,832,243.09	3,832,243.09
8190	Total Bldg. Maint. Expens...	I...	E..	8100..8190				296,202.58	296,202.58
8290	Total Administrative Exp...	I...	E..	8200..8290				76,692.54	76,692.54
8390	Total Computer Expenses	I...	E..	8300..8390				68,834.68	68,834.68
8490	Total Selling Expenses	I...	E..	8400..8490				153,550.20	153,550.20
8590	Total Vehicle Expenses	I...	E..	8500..8590				29,739.63	29,739.63
8690	Other Operating Exp., Total	I...	E..	8600..8690				59,504.21	59,504.21
8695	Total Operating Expenses	I...	E..	8000..8695				684,523.84	684,523.84
8790	Total Personnel Expenses	I...	E..	8700..8790				1,949,747.16	1,949,747.16
8890	Total Fixed Asset Deprecia...	I...	E..	8800..8890				304,852.13	304,852.13
8995	Net Operating Income	I...	T..	6000..8995				-282,606.66	-282,606.66
9190	Total Interest Income	I...	E..	9100..9190				-1,897.65	-1,897.65
9290	Total Interest Expenses	I...	E..	9200..9290				214,846.70	214,846.70
9395	NI BEFORE EXTR. ITEMS & T...	I...	T..	6000..9395				-68,153.73	-68,153.73
9495	NET INCOME BEFORE TAXES	I...	T..	6000..9495				-67,389.85	-67,389.85
9999	NET INCOME	I...	T..	6000..9999				-11,442.65	-11,442.65
*▶		I...	P..						

Figure 3.4 Chart of Accounts after filtering

As you can see this list is considerably shorter and contains only those cost accounts that have had activity. Only one problem remains: Is this the profit from January 2001? To see what information is being used to generate the financial sums, click on the position in the column, *Net Change*, and then click on the arrow that appears (In the future we will refer to this activity as drilling down to the details.) The following window appears:

Posting ...	D..	Docume...	G/L Acco...	Description	G..	G..	G..	Amount	B..
01/01/00		2000-1	6110	Entries, January 2000	S..	N..	R..	-66,834.51	G..
02/01/00		2000-2	6110	Entries, February 2000	S..	N..	R..	-69,271.59	G..
03/01/00		2000-3	6110	Entries, March 2000	S..	N..	R..	-74,007.80	G..
04/01/00		2000-4	6110	Entries, April 2000	S..	N..	R..	-73,587.88	G..
05/01/00		2000-5	6110	Entries, May 2000	S..	N..	R..	-62,127.37	G..
06/01/00		2000-6	6110	Entries, June 2000	S..	N..	R..	-43,359.79	G..
07/01/00		2000-7	6110	Entries, July 2000	S..	N..	R..	-40,660.82	G..
08/01/00		2000-8	6110	Entries, August 2000	S..	N..	R..	-42,916.14	G..
09/01/00		2000-9	6110	Entries, September 2000	S..	N..	R..	-76,539.72	G..
10/01/00		2000-10	6110	Entries, October 2000	S..	N..	R..	-68,940.90	G..
11/01/00		2000-11	6110	Entries, November 2000	S..	N..	R..	-74,679.20	G..
12/01/00		2000-12	6110	Entries, December 2000	S..	N..	R..	-56,592.11	G..
12/03/00	I..	103020	6110	Invoice 1002	S..	N..	R..	-533.40	G..
12/11/00	I..	103019	6110	Invoice 1001	S..	N..	R..	-1,063.10	G..
01/07/01	I..	103005	6110	Order 101001	S..	N..	R..	-6,963.40	G..
01/14/01	I..	103008	6110	Order 101004	S..	N..	R..	-649.40	G..
01/14/01	C..	104001	6110	Credit Memo 104001	S..	N..	R..	246.60	G..
01/15/01	I..	103021	6110	Invoice 1003	S..	N..	R..	-688.90	G..
01/16/01	I..	103009	6110	Order 101012	S..	N..	R..	-178.00	G..
01/16/01	C..	104002	6110	Credit Memo 104002	S..	N..	R..	649.40	G..
01/17/01	I..	103018	6110	Order 6005	S..	N..	R..	-2,920.00	G..
01/19/01	C..	104003	6110	Credit Memo 104003	S..	N..	R..	944.60	G..
01/22/01	I..	103014	6110	Order 101007	S..	N..	R..	-944.60	G..
01/23/01	I..	103004	6110	Invoice 103004	S..	N..	R..	-21,814.00	G..

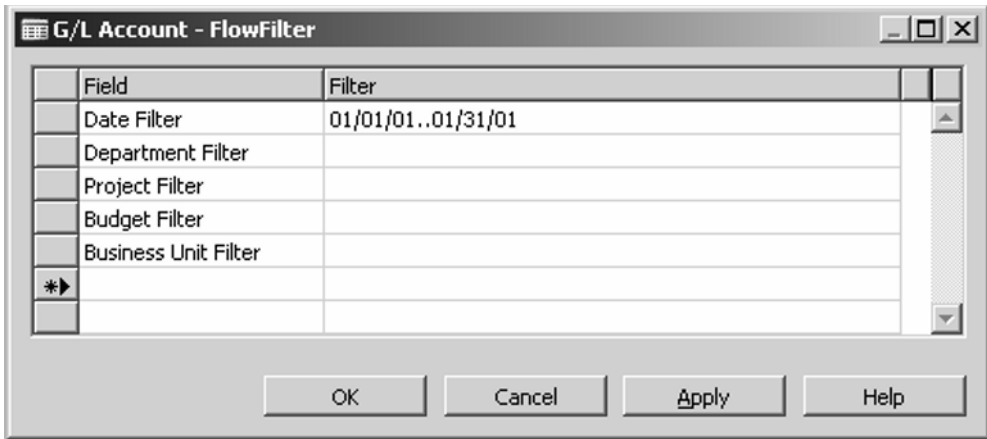
**Figure 3.5** Account 6110 entries

Immediately we can see that there are records in the list that have a posting date in the year 2000. These records cannot be used to calculate the firm's January 2001 profit and therefore must be eliminated. How do we make sure that the sums in the *Chart of Accounts* represent only records posted between 01/01/01 and 01/31/01? Leave the drill down list by pressing the ESC key on your keyboard and return to the *Chart of Accounts*.

To control the information being used to calculate the values in the *Net Change Flow Fields*, you must define a *Flow Filter* on the *Chart of Accounts*. Go to:

- **View > Flow Filter**

The following window will appear:



**Figure 3.6** *G/L Account - Flow Filter window*

Here is a list of *Flow Filters*. These special filters control the information that the *Chart of Accounts* accesses to calculate the values Microsoft Navision places into the *Flow Field*, *Net Change*. Enter the text 010101..013101 into the second column, *Filter*, in the *Date Filter* row. This filter commands Microsoft Navision to access only those records in its *Flow Field* calculation that have a posting date between the 1<sup>st</sup> and 31<sup>st</sup> of January 2001. After entering this click OK. Next, the shortened *Chart of Accounts* window will return as follows:



No.	Name	I... A.	Totaling	G.. G.. G.	Net Change	Balance
6195	Total Sales of Retail	I... E..	6105..6195		-65,737.68	-983,351.00
6295	Total Sales of Raw Mater...	I... E..	6205..6295		-433.79	-5,848,247.47
6495	Total Sales of Resources	I... E..	6405..6495		-24,837.00	-24,837.00
6995	Total Revenue	I... E..	6100..6995		-89,284.53	-7,056,468.52
7195	Total Cost of Retail	I... E..	7105..7195		25,099.85	837,430.70
7295	Total Cost of Raw Materi...	I... E..	7205..7295		85,509.93	2,994,812.39
7995	Total Cost	I... E..	7100..7995		110,609.78	3,832,243.09
8190	Total Bldg. Maint. Expens...	I... E..	8100..8190		456.69	296,202.58
8290	Total Administrative Exp...	I... E..	8200..8290		193.59	76,692.54
8390	Total Computer Expenses	I... E..	8300..8390		110.52	68,834.68
8490	Total Selling Expenses	I... E..	8400..8490		166.79	153,550.20
8590	Total Vehicle Expenses	I... E..	8500..8590		85.47	29,739.63
8695	Total Operating Expenses	I... E..	8000..8695		1,013.06	684,523.84
8790	Total Personnel Expenses	I... E..	8700..8790		1,000.27	1,949,747.16
8995	Net Operating Income	I... T..	6000..8995		23,450.91	-282,606.66
9395	NI BEFORE EXTR. ITEMS & T...	I... T..	6000..9395		23,450.91	-68,153.73
9495	NET INCOME BEFORE TAXES	I... T..	6000..9495		23,450.91	-67,389.85
9999	NET INCOME	I... T..	6000..9999		23,450.91	-11,442.65

**Figure 3.7** Chart of Accounts after Table & Flow Filters are defined

Once again drill down to some details behind the *Flow Field*, *Net Change* by clicking on the first position in the column, *Net Change*. The following window—as seen in Figure 3.8—appears:

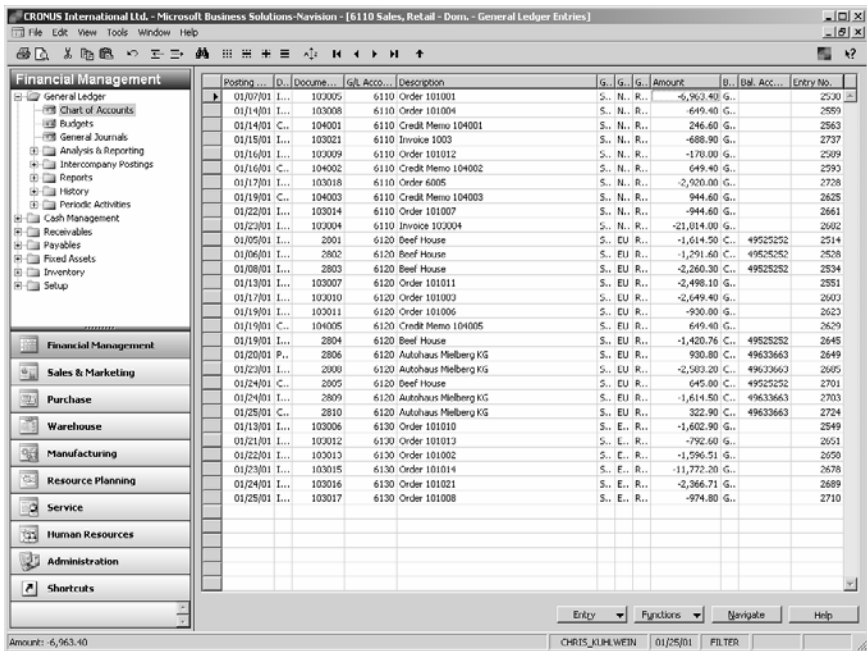


Figure 3.8 G/L Entries window

As you can see the *Net Change* value is only a sum of the *Amount* column for records posted between the dates 01/01/01 and 01/31/01. Press ESC and return to the *Chart of Accounts*.

This is how to set up a window to get a short and powerful overview of all the details that makeup the financial posting for January 2001. It is accomplished using both *Table Filters* and *Flow Filters*. Now you can answer the question posed by your boss concerning how much profit was made in January 2001: -23,450.91. If she asks any questions about how that is possible, you have all the details at your fingertips. The same filtering techniques must be used throughout Microsoft Navision whether you are running a report or using the dimensions or sales analysis matrixes. Understanding *Table Filters* and *Flow Filters* will help you to answer a million questions.

### 3.2.4

## Filter Options

Here is a first glimpse at recognizing the power of filters and understanding how to set them so that get what you want and only what you want. Below is a catalogue of all the filtering operators you can use in Microsoft Navision, their properties and how to use them together.

## Microsoft Navision: Filtering

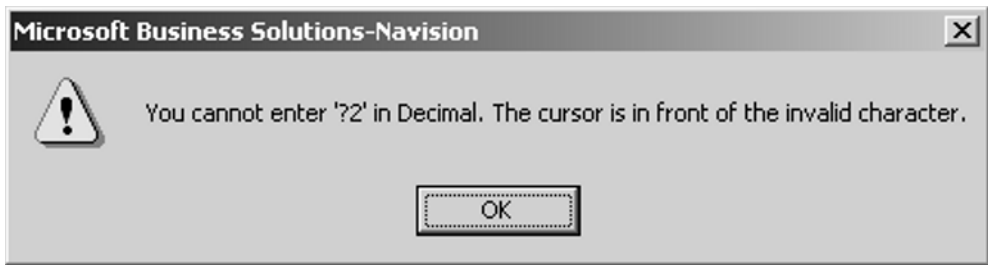
Function	Operator	Filter	All records...
Equal to		<b>80</b>	with values equal to 80.
		<b>Profit</b>	identical to the word 'Profit.'
Empty	''	''	that are empty.
Not Empty	<>' '	<>''	that are not empty.
Interval	..	<b>.80</b>	with values less and equal to 80
	..	<b>20..80</b>	with values from 20 to 80.
	..	<b>20..</b>	with values greater than and equal to 20.
	..	<b>..S</b>	with letter before, but not including capital S.
	..	<b>P..S</b>	Capital P up to, but not including capital S.
	..	<b>P..</b>	begining with letter capital P and up.
	..	<b>..01/02/01</b>	dated up to and including 01/02/01
Either/or		<b>20 80</b>	with values equal to 20 or 80.
		<b>Profit Loss</b>	identical to the word 'Profit' or the word 'Loss.'
		<b>12/31/00 02/01/01</b>	dated 12/31/00 or 02/01/01.
Inequality	<>	<b>&lt;&gt;20</b>	with values not equal to 20.
	<>	<b>&lt;&gt;Loss</b>	not identical to the word 'Loss.'
	<>	<b>&lt;&gt;01/02/01</b>	not dated 01/02/01.
Greater than	>	<b>&gt;20</b>	greater than, and not equal to 20.
	>=	<b>&gt;=20</b>	greater than and equal to 20.
	<	<b>&lt;80</b>	less than and not equal to 80.
	<=	<b>&lt;=80</b>	less than and equal to 80.
Unknown Te:	?	<b>Prof?t</b>	like Profit, Profet, where ? Is substituted.
Characters	*	<b>*S</b>	ending with the letter capital S.
	*	<b>*S*</b>	containing the letter capital S.
	*	<b>S*</b>	begining with capital S.
Capital or lower case	@	<b>@s</b>	contianing capital or lower case S.
<b>Compound Functions</b>			
And	&	<b>&gt;20&amp;&lt;80</b>	between but not including 20 and 80.
Parentheses	()	<b>10 (&gt;=20&amp;&lt;=80)</b>	containing 10 or between and including the
Misc.	.. ..	<b>..12/31/00 02/01/01..</b>	dated up to 12/31/00 or all after 02/01/01.
	*@*&*@*	<b>*@S*&amp;*@P*</b>	containing a captial or lower case S and P.

Figure 3.9 Filtering techniques

Note that each type of filtering operator will work in only certain types of fields. For example, if your cursor is within a quantity field like *Sales Amount* and you click on the filter tool and write the value:

?2

Microsoft Navision will give you the following error message:



**Figure 3.10** False filter value type entered into filter

This message means that you have tried to enter a text or string value into a quantitative field. Instead, you should enter:

..2

if you want to view all the records up to and equal to 2.

Filtering is the ability to control the information that Microsoft Navision selects out of the database for viewing, editing, deleting and processing. Becoming an expert user of Microsoft Navision means you will need to understand filtering which is almost always the key to getting the right information. This is true whether you want to trace a specific order history or calculate the correct product performance report. As you can imagine, a user will be lost in Microsoft Navision without a firm grasp of filtering.

### 3.3

### Sorting

To achieve 100% predictability of your information we must take a moment to look at an issue concerning the behavior of databases.

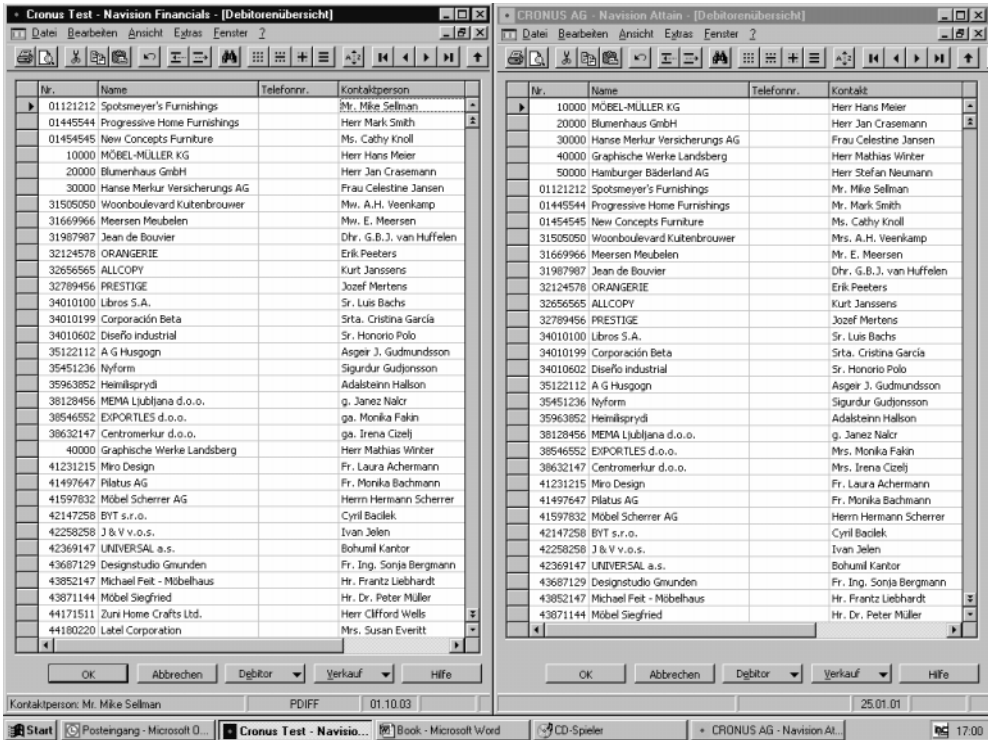
Various Microsoft Navision databases sort records differently. Depending on which Microsoft Navision database type you are using (Microsoft SQL, Oracle or the Microsoft Navision client database), the order of the same records might be different. Some databases sort numbers before letters and others vice versa.

*Difference between Client and SQL versions*

Other databases may ignore the zeros that are put before a number while others do not.

The way the records in a database are sorted can determine which filters you must use. In this book we are using examples based on the Microsoft Navision Client version and the Microsoft Navision MSSQL Server version. These two Microsoft Navision versions use different database solutions; there is a tricky sorting difference between these two systems.

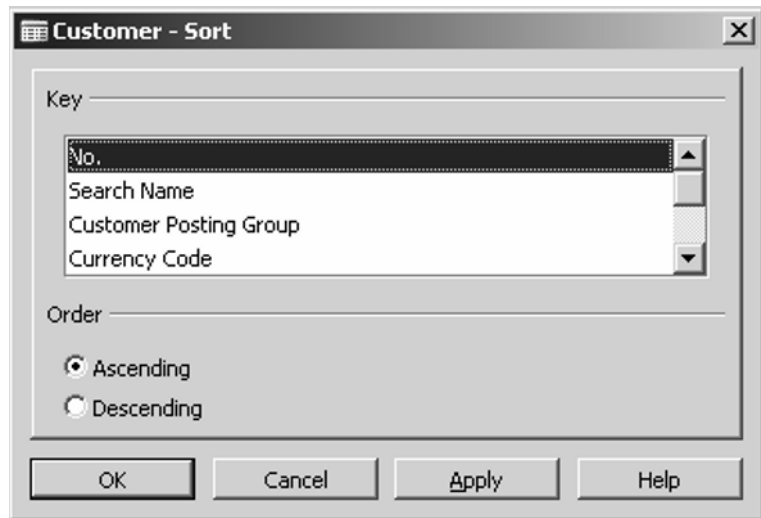
Consider the following windows: The window on the left is the Cronus database *Customer List* view in the Microsoft SQL Server edition of Microsoft Navision, while the window on the right is the Microsoft Navision Client version. Both databases contain the same customers, but sort records differently.



**Figure 3.11** Same customer list seen in two different databases

As you can see, the sorting of the customer records differs greatly between the two databases. This is the case even though both *Customer List* views are sorted by the same key. If you click

the sorting tool in either database you will see that the same sorting option is chosen:

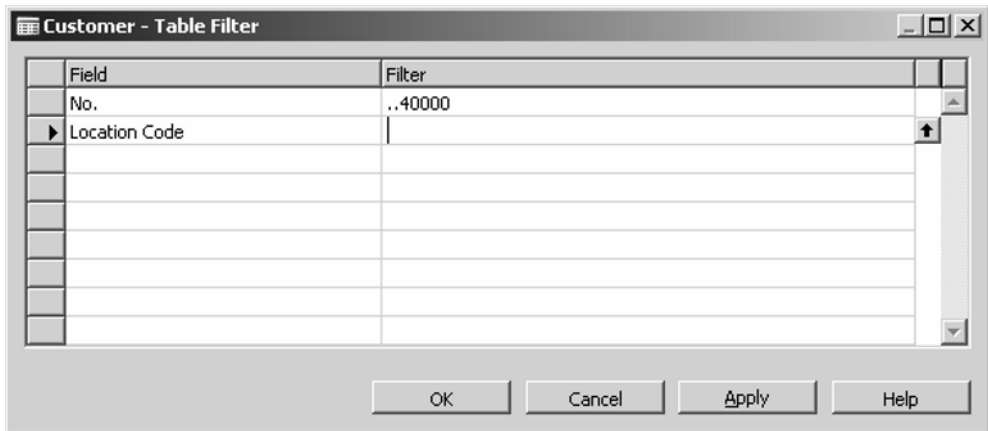


**Figure 3.12** *Customer - Sort* options window

To understand the confusion this sorting difference could create, consider the following example:

*Example of sorting ambiguity*

In both Cronus databases the following *Table Filter* is set in the *Customer Table Filter* window:



**Figure 3.13** *Customer - Table Filter* window

The following records will be returned by the two different database-sorting systems:

Nr.	Name	Telefonnr.	Kontaktperson
01121212	Spotsmeyer's Furnishings		Mr. Mike Sellman
01445544	Progressive Home Furnishings		Herr Mark Smith
01454545	New Concepts Furniture		Ms. Cathy Knoll
10000	MOBEL-MULLER KG		Herr Hans Meier
20000	Blumenhaus GmbH		Herr Jan Crasemann
30000	Hanse Merkur Versicherungs AG		Frau Celestine Jansen
31505050	Woonboulevard Kutenbrouwer		Mw. A.H. Veenkamp
31669966	Meerssen Meubelen		Mw. E. Meerssen
31987987	Jean de Bouvier		Dhr. G.B.J. van Huffelen
32124578	ORANGERIE		Erik Peeters
32656565	ALLCOPY		Kurt Janssens
32789456	PRESTIGE		Jozef Mertens
34010100	Libros S.A.		Sr. Luis Bachs
34010199	Corporación Beta		Srta. Cristina Garcia
34010602	Diseño industrial		Sr. Honorio Polo
35122112	A G Husgogn		Asgeir J. Gudmundsson
35451236	Nyform		Sigurdur Gudjonsson
35963852	Heimilspryd		Adalsteinn Hallson
38128456	MEMA Ljubljana d.o.o.		g. Janez Nalor
38546552	EXPORTLES d.o.o.		ga. Monika Falin
38632147	Centromerkur d.o.o.		ga. Irena Cizelj
40000	Graphische Werke Landsberg		Herr Mathias Winter

Nr.	Name	Telefonnr.	Kontakt
10000	MOBEL-MULLER KG		Herr Hans Meier
20000	Blumenhaus GmbH		Herr Jan Crasemann
30000	Hanse Merkur Versicherungs AG		Frau Celestine Jansen
40000	Graphische Werke Landsberg		Herr Mathias Winter

**Figure 3.14** Customer lists from different database types

Each Microsoft Navision edition returns a very different set of customer lists. The Microsoft SQL edition on the left considers a zero in the beginning of a code to be significant while the client database version on the right does not.

You must be careful when you define a filter that this difference does not invalidate your research results. This sorting question becomes extremely problematic when an application is designed with weak record sorting assumptions and is then used in databases with different sorting habits.

### *Clever coding*

The way around this problem is to assign account IDs in a more clever way than has been used here in the Cronus database. For example, if the *Account ID* field is the type that can contain both letters and numbers (that is either a text field or a code field), then you need to:

- Decide how many place values you want to use in the *Account ID* and always use the same number of places.
- Decide whether the *Account ID* will begin with a letter or number and then always follow your decision.
- Never begin an ID with a zero, if possible. Doing so can cause import and export peculiarities, such as the foreign program considering the *Account ID* to be a value rather than text.

If you follow these conventions you should have no problem with sorting ambiguities. However, you may not have control over the *Account ID* naming conventions, so keep this in mind.

## 3.4 **Implementing Menu, Filtering and Sorting Knowledge**

Let us now work through a concrete example to understand how Microsoft Navision handles information with filtering techniques, sorting and other general tools.

### 3.4.1 **Searching for the Correct Customer**

As a way to introduce the Microsoft Navision end user working environment, we will work through a task together using some of the tools already discussed. You will see by way of a concrete example the twists and turns taken to accomplish a task. We will discuss aspects of the Microsoft Navision environment as we happen upon them in the example.

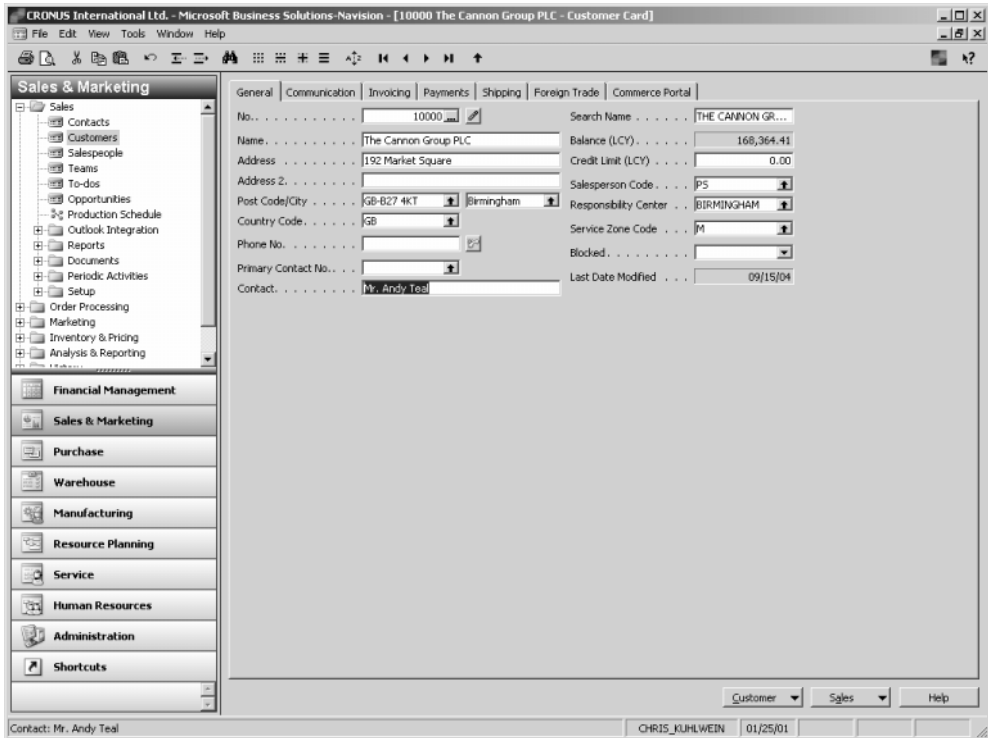
Suppose you have just found a sales order beside the fax machine. You do not know how long it has been there so you want to process it as fast as possible. The sales order is difficult to read because it was quickly handwritten by one of your salespersons and then faxed. The order is from an existing customer but the only identification you can make out on the poorly written order are a name, *Möbel Sieg...*, the country name, *Austria*, and a street address, *Raxstraße 47*.

Let us walk through this process step-by-step and discuss some working environment details as we go. To search for the correct customer go to:

- **Sales & Marketing > Sales > Customers**

You will now see the *Customer Card* as follows:





**Figure 3.15** Customer Card window

Now search for the customer *Möbel*. It is recommended that when you search for a customer or an item that you develop a habit of using a table view. If you search from the card view it is easy to overwrite information in the card without realizing it.

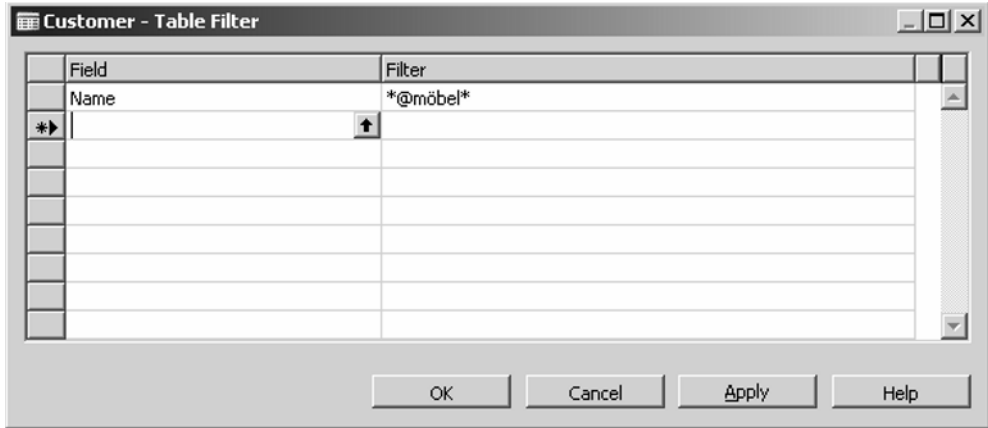
*Searching with the list view*

Working with table views also enables you to more easily see errors in your firm's information. For example, if your firm has a large customer database it may be the case that two or more customers have similar or identical names, or it may even be the case that the same customer has been mistakenly entered into the database more than once. This type of customer duplication can cause all sorts of problems that may cost you the customer's account in the end. When you are searching through customer's names with clever filters in the *Customer List* view you will immediately see when a customer may have been entered more than once or ambiguously named.

Now click the first icon tool from the left—the upward pointing arrow—to open the *Customer List* view. When the *Customer List* view is open, click on the *Name* column and then click on the

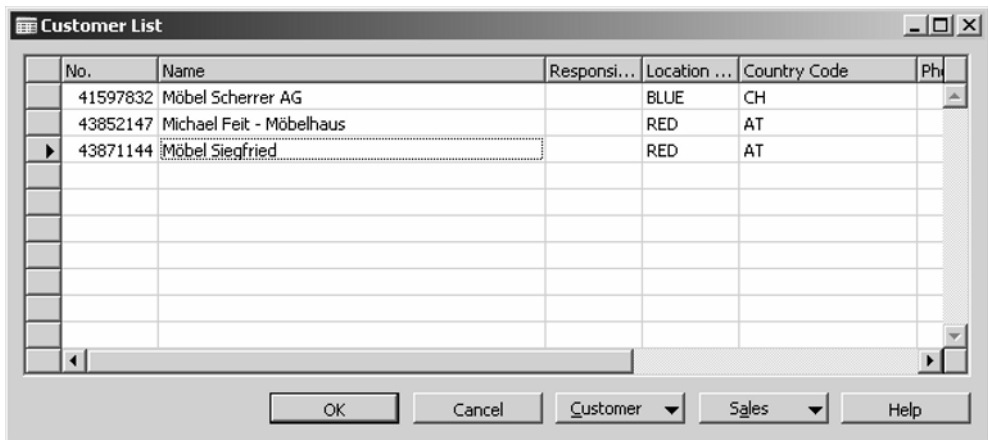
icon tool ninth from the left (the Table Filter tool). Next, type the following text into the second column, *Filter*:

\*@möbel\*



**Figure 3.16** Customer Table Filter

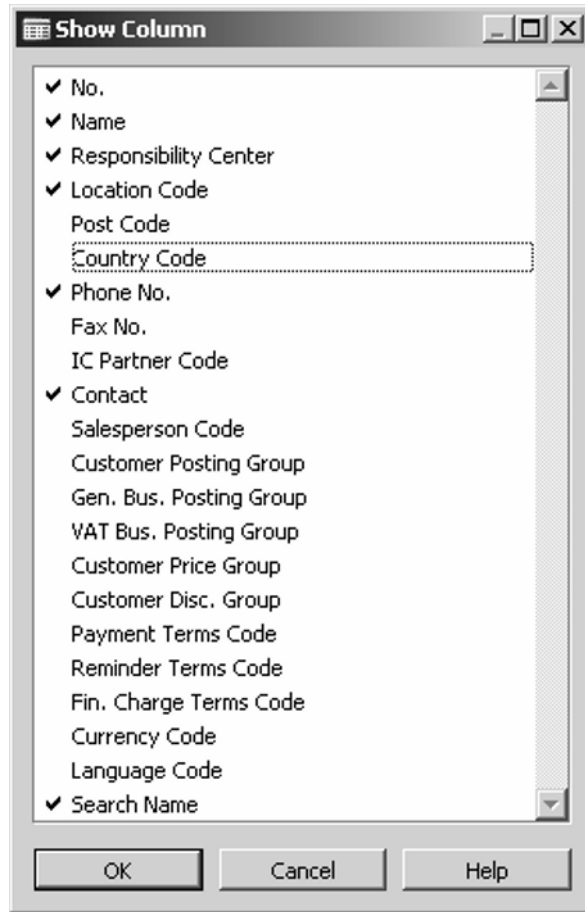
Click OK. The following window will appear:



**Figure 3.17** Filtered Customer List

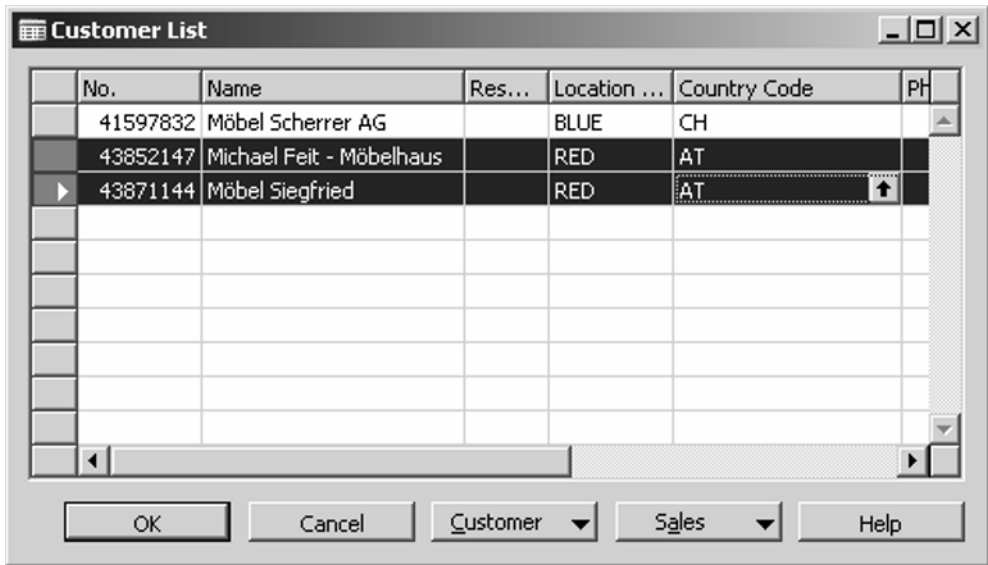
It is not quite clear that customer number 4387114 is the correct customer, but just to make sure you may want to check the Country Code which indicates that the customer is located in Austria. To do this, place your cursor on any column heading

and click the right button on your mouse. The following window will appear:



**Figure 3.18** Activating hidden columns

This is the list of all the fields that can be displayed in the *Customer List* view. Place your cursor just before the textline, *Country Code*, and then click the left button on your mouse—placing a check mark before the text. Next, click OK. The following window will appear:



**Figure 3.19** *Customer List* with *Country Code* column

As you can see a new column, *Country Code*, has appeared. Two customers—both located in Austria (*Country Code* is equal to *AT*)—are shown. Therefore, this new column cannot tell you 100% which customer is correct. The last piece of information is the street address. However, as you can see, there is no street address shown in the *Customer List* view nor is a street address in the list of hidden columns where we found *Country Code*. If you return to the card, you lose the filter because the *Customer List* view is reset. So, where is the street address information for the customers in the table view list?

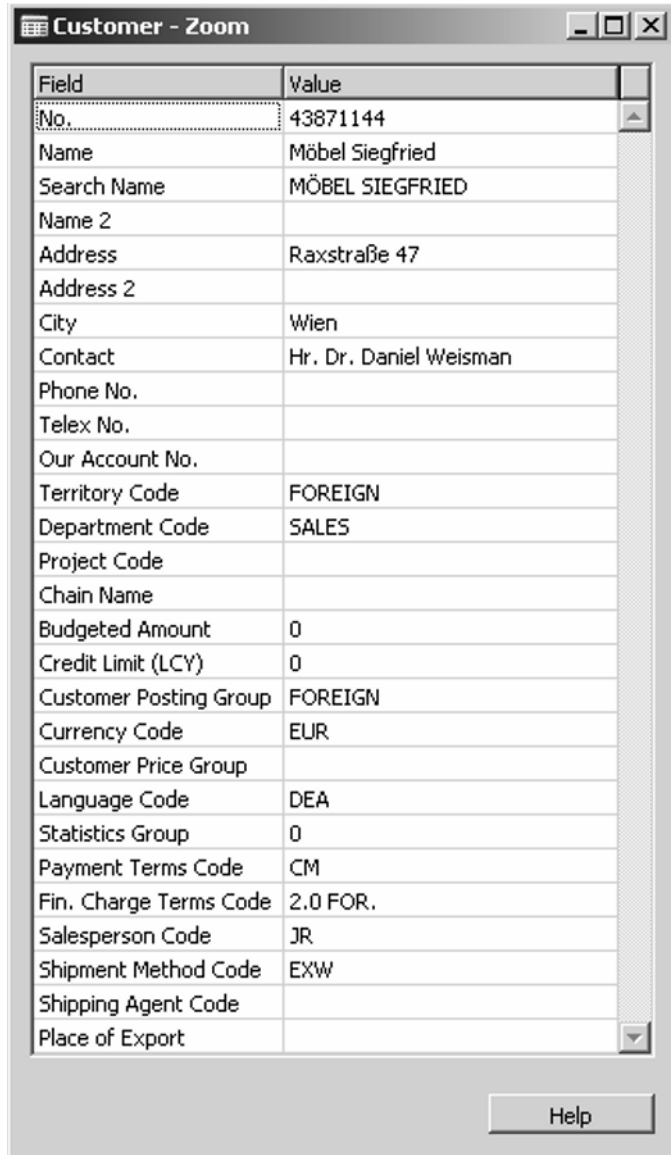
### 3.4.2

#### Viewing Fields with Zoom

Although you can be pretty sure that the address will be in the *Customer Card*, no doubt you prefer to see the street address from your present position. You can open a window where everything that exists in Microsoft Navision concerning this one customer is put into a single list. You can do this easily by using the Microsoft Navision Zoom tool. Simply click anywhere on the record you wish to see (in this case you want to consider the record *Möbel Siegfried*), and then go to:

- **Tools > Zoom**

The following window will appear:



**Figure 3.20** Customer - Zoom view

Here you will see a complete list of every field that is possible in the customer stock dataset even if they are fields not found in the *Customer List* view or the *Customer Card*. This is the complete list of all the fields in the underlying customer table deep within the Microsoft Navision database.

Zoom will help you gain a general as well as a deeper understanding of Microsoft Navision and is therefore an important tool for general users and developers. Zoom is important because it can reveal information available nowhere else to the end user. This “hidden” information may include things like *Flow Fields* or counting fields, such as *Entry No.*, which keeps the posting records sorted in Microsoft Navision.

In the Zoom window we see the street address in the *Address* field is *Raxstraße 47*. Now we are 100% sure that customer number, 4387114, is the correct customer for this sales order. Next, close the Zoom window by pressing the ESC button. Double-click on customer 4387114 in the *Customer List* view. You are once more in the *Customer Card*.

### 3.4.3 Creating a Sales Order

The next step in entering the sales order is to make sure it has not been entered by another employee. In the *Customer Card* select the Sales button at the bottom of the window and then select the menu item, *Orders*. The following window will appear (See Figure 4.21):

When you enter the *Sales Order* from the *Customer Card* Microsoft Navision automatically sets a filter on the open *Sales Orders* so that only orders from the customer in the *Customer Card* are shown. Because the *Sales Order* form that appears now is empty, we know with certainty that there are no open orders for customer, *Möbel Siegfried*—or at least at this moment. On the contrary, if there had been existing open orders, then you should look through them to make sure you are not entering an order which has already been created in the system.

Click into the sales header or on the sales lines. Microsoft Navision automatically fills the sales header with information on *Möbel Siegfried*, supplied from the customer table.

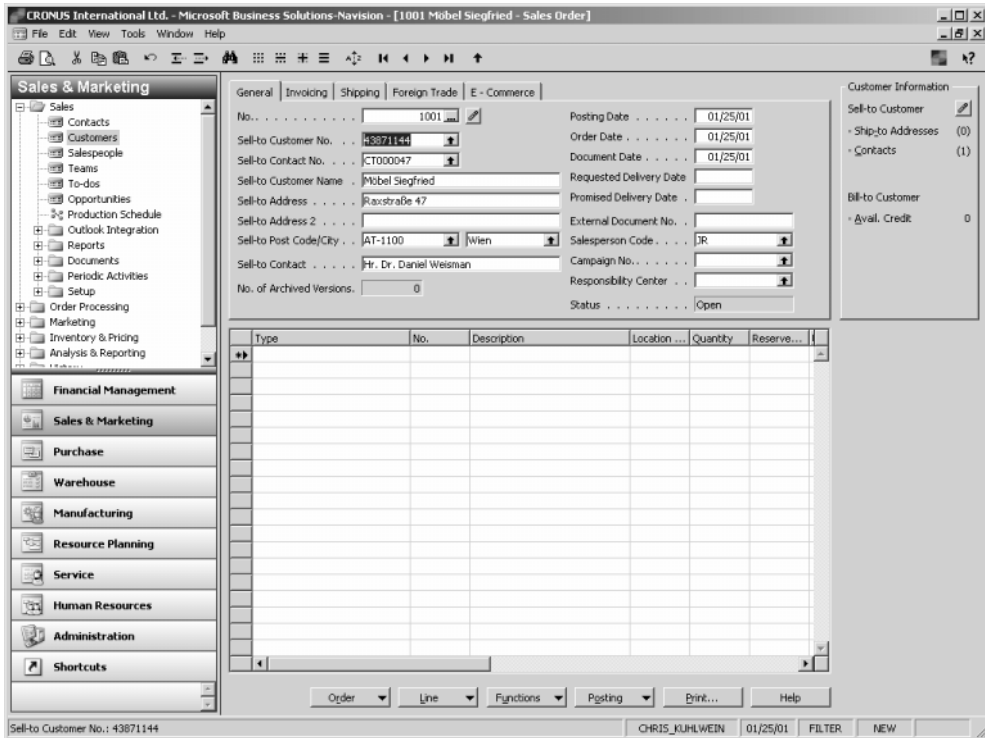


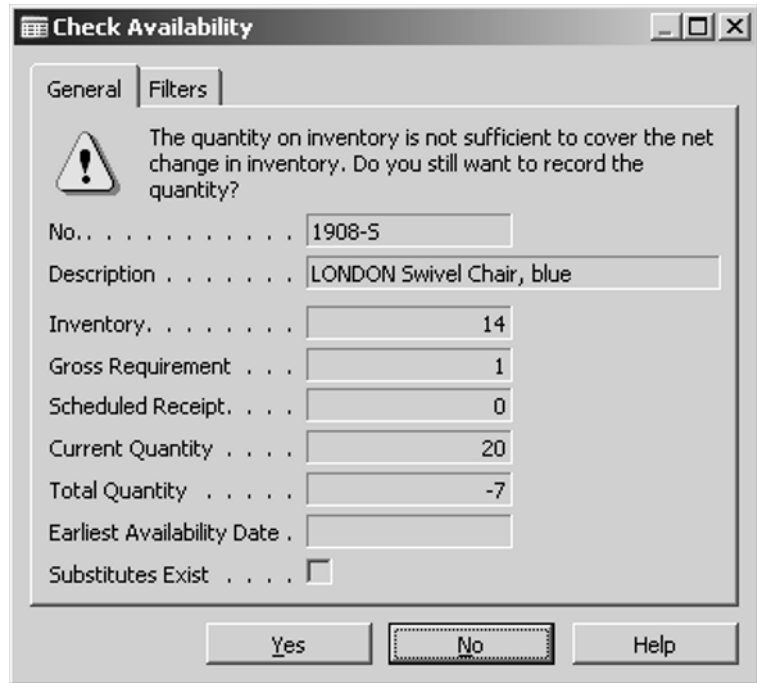
Figure 3.21 Sales Order window

You can begin entering the items to be purchased by this customer. When you click in the first column (*Type*) of the first sales line, an option list will appear. Based on the option you choose, Microsoft Navision automatically configures the sales line so that other columns display information and links that pertain to the *Type* you have chosen. Next, drill down in the *No.* field by clicking on the *No.* column and then click on the arrow that appears. When you click on an arrow within a field you will be shown a list of all the items pertaining to the *Type* you selected.

Let us enter an item for demonstration purposes. Select the *Item* option from the option menu which appears after clicking on the *Type* column. Next, drill down in the *No.* field; the *Item List* view will appear. Select the item which is the second from the top of the list: *Item number 1908-S, LONDON Swivel Chair, blue*.

Suppose you have talked with the buying department today and they told you that a shipment of 20 *LONDON Swivel Chair, blue* has just arrived today and are available to be sold. This just happens to be the quantity needed by the customer in your sales

order. Next, enter 20 in the *Quantity* column and press ENTER. The following warning message window will appear:



**Figure 3.22** *Check Availability* warning

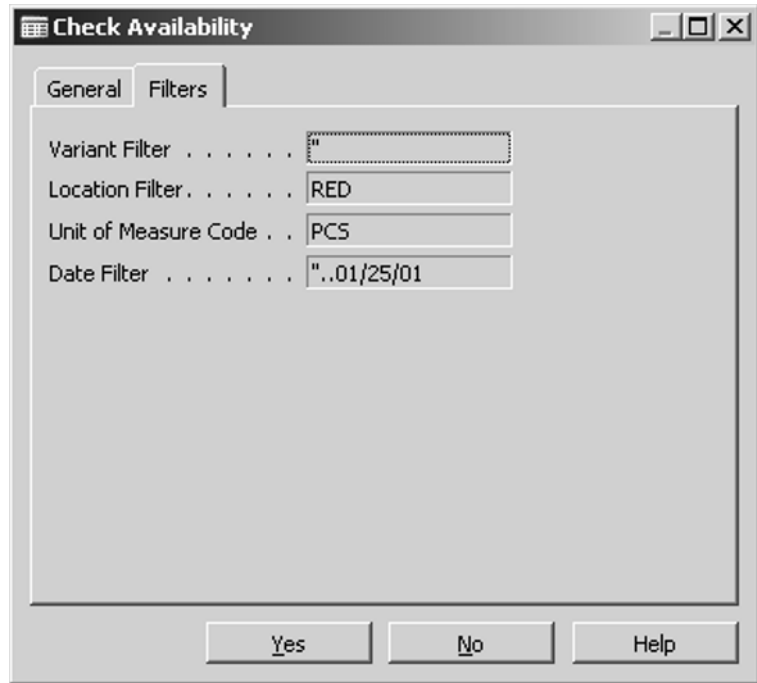
This message informs you that the item and quantity that you have entered are not currently available in stock. Because you know 20 of this item arrived today, you are confused by Microsoft Navision's calculation which indicates that only 14 are available. Before you jump to the conclusion that someone in the buying department is lying, look deeper to see how Microsoft Navision came up with this calculation. The *Inventory* field is one of the *Flow Fields* that Microsoft Navision calculates each time it is used. As you know from the previous section, *Flow Fields* are controlled by two things:

- A calculation formula that is written into the field itself
- A *Flow Filter*

These are your clues to understanding why Microsoft Navision is telling you there are only 14 in stock. The calculation formula within the *Flow Field* inventory tells Microsoft Navision to make a sum of the *Remaining Quantity* for an item in the *Item Ledger*

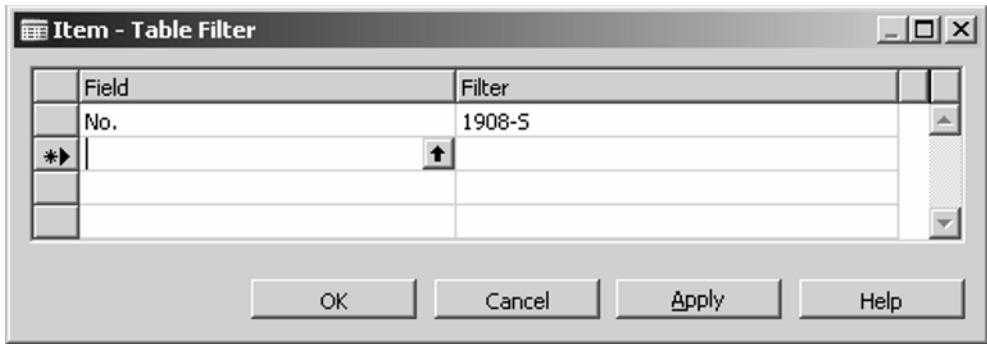


*Entry* table—a table which records every movement that an item makes. Now try to see what *Flow Filters* may be acting on the inventory calculation at this moment. Click on the second tab in this message window. The following window will appear:

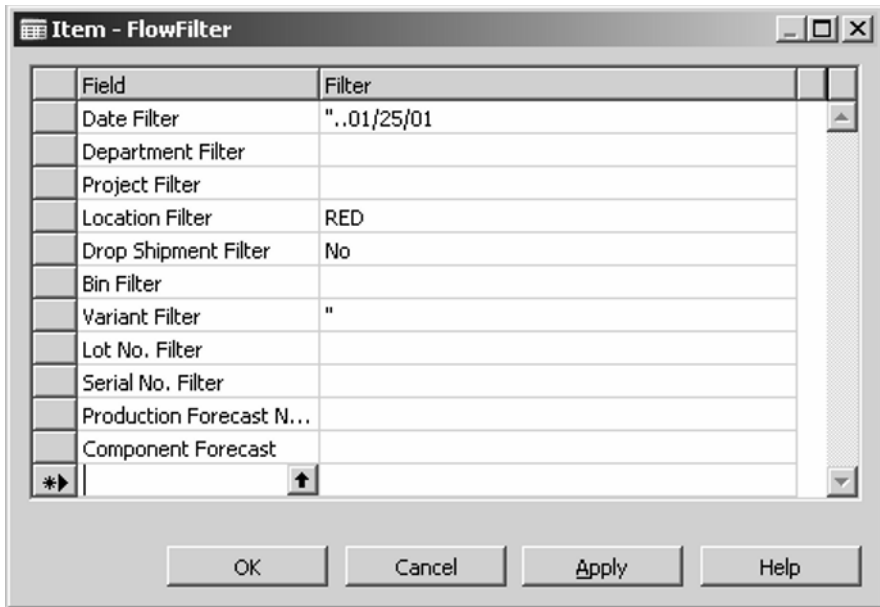


**Figure 3.23** Filter factors behind inventory calculation

This window shows us exactly what conditions are determining Microsoft Navision's calculation of the inventory; Microsoft Navision is using these *Flow Filters* to determine the Inventory. Here Microsoft Navision has automatically defined a few such *Flow Filters* on things like *Location Code*, which may lead us to us to why we expected 14 pieces in stock. You can do some background research on this item's inventory to understand how Microsoft Navision arrived at this calculation of the inventory. Make a note of the various filters that determine this inventory calculation. To do so, first check the *Table Filters* by clicking on the Table Filter icon tool.



**Figure 3.24** The *Table Filter* conditions for *Item*



**Figure 3.25** *Flow Filter* conditions on inventory calculations

This *Table Filter* shows the number of the item we have used. Check the *Flow Filter* by pressing ESC and clicking on the Flow Filter icon tool. Make a note of the information in the above window (See Figure 3.25).

These are all the filters that Microsoft Navision is using to calculate the *Flow Field* inventory.

### 3.4.4 Looking Deeper into the Flow Field Inventory

Press ESC to leave the *Flow Filter* window. To go deeper into the history of this item and its inventory, go to the item itself in the *Item Card*. Select “No” and then press ESC three times to leave the *Sales Order* and return to the main menu. Next, go to:

- **Warehouse > Planning & Execution > Item**

Click on the *No.* field and then click on the Search icon tool (CTRL+F or the ninth icon from the left in the Toolbox). In the search window type *1908-S* and click on First. Now close the search window by pressing ESC. The following window will appear:

1908-S LONDON Swivel Chair, blue - Item Card	
General   Invoicing   Replenishment   Planning   Foreign Trade   Item Tracking   E - Commerce   Warehouse	
No. . . . . 1908-S	Search Description . . . LONDON SWIVEL ...
Description . . . . . LONDON Swivel Chair, blue	Inventory . . . . . 305 ↓
Base Unit of Measure . . . PCS	Qty. on Purch. Order . . . 50
Bill of Materials . . . . . <input type="checkbox"/>	Qty. on Prod. Order . . . 0
Shelf No. . . . . D5	Qty. on Component Lines . . . 0
Automatic Ext. Texts . . . <input type="checkbox"/>	Qty. on Sales Order . . . 18
Created From Nonstoc... <input type="checkbox"/>	Qty. on Service Order . . . 0
Item Category Code . . .	Service Item Group . . .
Product Group Code . . .	Blocked . . . . . <input type="checkbox"/>
	Last Date Modified . . . 09/15/04
Item ▼ Sales ▼ Purchases ▼ Functions ▼ Help	

**Figure 3.26** *Item Card* window

In the second column you see that the *Inventory* field contains a value of 305 and that the *Qty. on Sales Order* is only 18. How is it possible that Microsoft Navision tells you there are not enough available for your order of 20? The answer lies in understanding that *Inventory* is a *Flow Field* and it can have a different value depending on what *Flow Filters* are determining it. To see what information the *Flow Field* inventory in the *Item Card* is calculated from, drill down to its components by clicking on the *Inventory* field and then click on the arrow that appears. The following window will appear:

Posting Date	Entry Type	D...	Item No.	Description	Location Code	Quantity	Remaining Quantity	Sales Amount (Actual)
12/31/00	Positive ...	S...	1908-5		BLUE	234	234	0
12/31/00	Positive ...	S...	1908-5		GREEN	47	37	0
01/22/01	Purchase	1...	1908-5	LONDON Kontorstol, blå	GREEN	20	20	0
01/25/01	Transfer	1...	1908-5		GREEN	-10	0	0
01/25/01	Transfer	1...	1908-5		OWN LOG.	-10	0	0
01/25/01	Transfer	1...	1908-5		OWN LOG.	-10	0	0
12/31/00	Positive ...	S...	1908-5		RED	5	4	0
01/22/01	Sale	1...	1908-5		RED	-1	0	0
01/25/01	Transfer	1...	1908-5		RED	10	10	0

**Figure 3.27** *Item Ledger Entries* window

Above you see the *Item Ledger Entries*. The *Flow Field* Inventory is a *Flow Field* in the *Item* table that sums the *Remaining Quantity* in the *Item Ledger Entry* table according to a set of *Flow Filters*. If you sum the *Remaining Quantity* column, you get 305, which is exactly what Microsoft Navision does to arrive at the same number displayed on the *Item Card*.

*Manually simulating Microsoft Navision's inventory calculation*

Let us try simulating manually what Microsoft Navision does programmatically and automatically. This is the first step to understanding the structure of Microsoft Navision. The importance of this step cannot be underestimated because it helps you achieve the skills of a practical developer with a minimum of abstraction. If you can repeat manually what Microsoft Navision does programmatically, then later you will be able to do the opposite: write a program to do what you can do manually. You might ask: If I can do something manually why would I write a program to do the very same thing? The answer is this: Programmatically means automation, speed and no human calculation errors! We recommend that you become an expert in using Microsoft Navision's filtering and navigation tools manually. This is the quickest and surest path to developing an imagination for what may be useful to later automate using programming code.

Next, look in the *Item Ledger Entries* for the posting of the 20 *LONDON Swivel Chair, blue* that you heard about from the buyer today. In this window you see there has been a purchase transaction on 01/22/01 of exactly 20 pieces; these are probably the

20 mentioned by the buying department. What you should notice is that these 20 pieces were all entered into the *GREEN Location Code*.

Now set the same *Flow Filters* on the *Item Card* that you noted were behind the mysterious message that stated there were only 14 *LONDON Swivel Chair, blue* available. You are trying to simulate manually what Microsoft Navision is doing programmatically.

First, press ESC to leave the drill down list and return to the *Item Card*. Next, click the Flow Filter icon tool and define the filters in exactly the way we have seen them defined in the *Flow Filters* of our warning message. After you have typed into the *Filter* column exactly what we noted in the *Flow Filters* of the *Sales Order* message, press OK. Now the value of the *Inventory* will be exactly 14. Next, drill down into the *Inventory* field. The following table view will appear:

Posting Date	Entry Type	D... No.	Item No.	Description	Location Code	Quantity	Remaining Quantity	Sales Amount (Actual)
12/31/00	Positive ...	5...	1908-5		RED	5	4	0
01/22/01	Sale	1...	1908-5		RED	-1	0	0
01/25/01	Transfer	1...	1908-5		RED	10	10	0

**Figure 3.28** Item Ledger Entries after Flow Filter activation

Sum the values in the *Remaining Quantity* field and you will calculate 14.

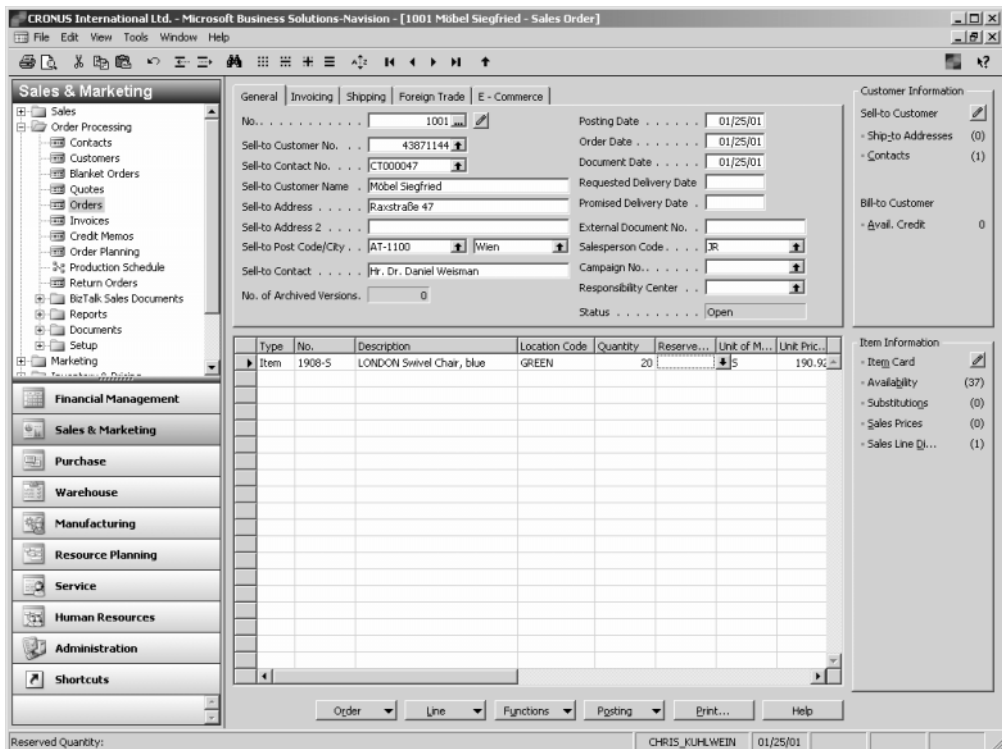
Only one last question remains: Why did Microsoft Navision choose its set of filters in the *Sales Order* to see what was available in the inventory? The answer is simple: These filters were based on the information entered—either by the user or automatically as a default—into the sales line itself.

Return one last time to the *Sales Order* and change the *Location Code* to *GREEN* since this is the *Location Code* where the newest shipment of 20 pieces has been posted. Go to:

- **Sales & Marketing > Order Processing > Orders**

Find the sales order for *Möbel Siegfried*.

You can see in the fourth column of the sales line, *Location Code*, that *RED* is written. Previously, we saw in the *Item Card* that the shipment of 20 pieces was posted in the *GREEN Location Code*. Therefore, in the fourth column of the sales line remove the *Location Code RED* and select *GREEN* from the look-up list. Now, enter *20* in the *Quantity* field and press ENTER. As you might expect, Microsoft Navision gives you no inventory availability warning because exactly 20 have come in today.



**Figure 3.29** *Sales Order* window

*Automatic filter conditions*

This is exactly the process that Microsoft Navision uses to check the availability of items in the inventory. Behind most of the processes in Microsoft Navision something like a filtering process is determining what actions Microsoft Navision will take. If you

understand this example you are well on your way to getting a feeling for what Microsoft Navision is doing in the background. You will be able to simulate the same processes yourself, manually.

All the features we have discussed are general features throughout the whole end user environment in Microsoft Navision. Filtering, list view, Zoom and drill down can be utilized in every module. Becoming proficient with filtering, drilling down and Zooming will make you an efficient Microsoft Navision navigator and is the groundwork needed to develop a programmer's intuition.

# 4

## Introduction to Development Concepts

---

In this chapter we will lay the groundwork for understanding the inner structure of Microsoft Navision. We will discuss the database principles upon which Microsoft Navision is built. These include table-relations and object hierarchy.

### 4.1 The Challenges in Organizing Information

For the Microsoft Navision ERP tool to help you optimize your firm, it must be able to adapt to the ideal structure of your company. To achieve this, it is essential that the software be flexible, programmable and expandable. Likewise, it must have powerful tools for updating, analyzing and restructuring your firm's business processes and information. Finally, it must have user-friendly tools to help you accomplish your goals. In the following section we will move from the purely manual end user's point-of-view to the developer's point-of-view.

*Basics to help you communicate with developers*

The first hurdle in developing a new solution or adapting an existing one is to get the developer to understand what you want—no small task. If you never use the development practices in this book, at the very least, you will gain the skills needed to communicate with a developer. Knowing a little about how Microsoft Navision works will help you to:

- Get what you need from a developer
- Judge the quality of the developer's work
- Better estimate the time/money costs of specific development proposals

These three skills can save you prodigious amounts of time, money and frustration. In practice, the business mind and the programming mind are rarely in the same head. Sadly, it is often the case that much time and money are spent without the firm getting what it needs while the programmer is unsure of what he or she did wrong.



If you or someone within your firm is doing the ERP development work, then many of these problems are minimized. However, knowing what kind of solution your firm needs is not easily answered. A significant improvement in the software—one that is good for your firm and that you and your colleagues are happy with—will likely only come about following some trial and error use.

You may be worried about the dangers of doing “open heart surgery” on your firm’s information system. If you have little or no understanding of the background structure of your system, you may be afraid that if you try to adapt the system, you will destroy it. “Leave it to the specialists” is the usual resolution. Unfortunately, this strategy tends to create a mistrust of the software and later a mistrust of the specialist. If you do not want to know anything about basic program development concepts in Microsoft Navision, you run a great risk of communication breakdown between your firm and your outsourced programmer. Without this basic software knowledge you may not be able to communicate clearly to the specialists. What can happen is that you find yourself unable to tell the specialists what you need which means the developer cannot tell you if your development wishes are even possible with the given resources.

Without a foundation of knowledge of Microsoft, you may still be able develop your own in-house solutions, however, you may not know if the results are valid. For example, you might worry that if you create a new sales report, you might still lack the knowledge needed to determine whether or not the sales report is supplying 100% accurate information.

These problems are all symptoms of the overall complexity of such a tremendous tool as Microsoft Navision. The sheer size of the software and the apparent abstractness of the programming world and development concepts tends to overwhelm newcomers.

These are some of the “traps” that firms—who are never happy with their IT investment—have fallen into and which can keep you from optimizing you firm’s Microsoft Navision.

*Solution*

To avoid these problems:

- Gain a minimum of in-house development skill
- Have a test copy of your Microsoft Navision system so you can learn, test and develop without the risk of damaging your active system

- Realize upfront that the system will be in a state of continual improvement and that you will always need feedback from your users

The fear of making mistakes in your development will become a non-issue if you do thorough testing in your test system. You should realize however, that often times the only “real test” occurs when your staff uses the new solution in their everyday work. Therefore, it is sometimes impossible to know whether your solution will be 100% successful until it has been tested by the users in the live system. So long as you do not overwrite or erase data, almost every change you make is reversible. If you have something you cannot fix you can always contact your Microsoft Navision Business Solutions Center, but do not be afraid to try yourself. If your work in the software is well-documented, it should always be easy to return the system to its standard functionality.

*Method of cross-checking*

The way to be 100% sure of the results of your application is to cross-check the results with a second rigorous method. One of the reasons for covering the inventory calculation in such detail earlier was to show you how a value that is produced by the program can be checked manually. Likewise, when you create applications you should first have an idea of how the same results could be arrived at manually. It is good practice to come up with at least two different methods of arriving at the same calculation before you begin constructing your solution. This method allows you to validate the results of your programming. Follow this motto: Plan A without a plan B is no plan at all.

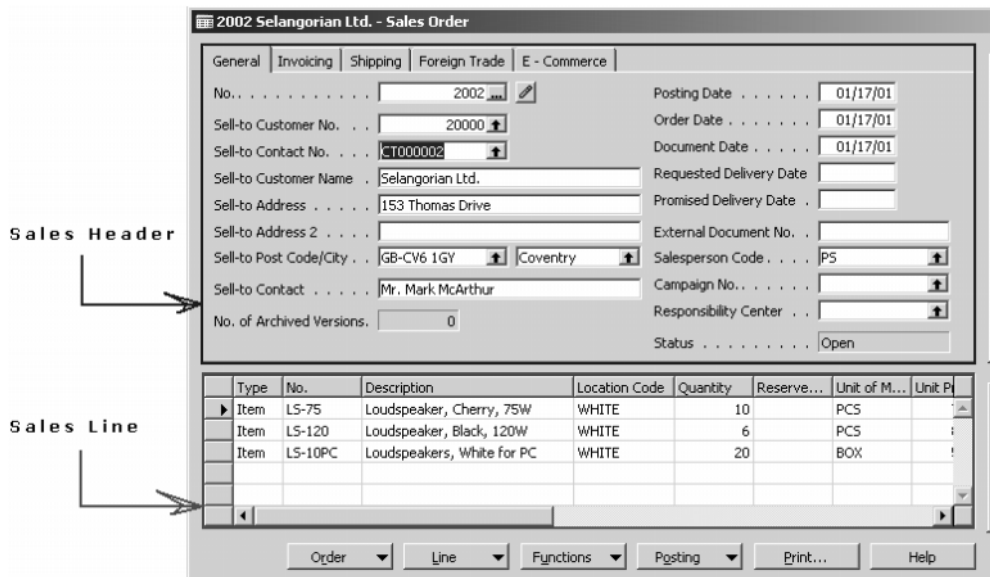
Everything seems strange when it is new, but with time, practice and understanding you will come to master the flexibility of your ERP system.

## 4.2 Organizing Information Using Table Relations

In this section we are going to look at how Microsoft Navision organizes the complex relationships between various types of information into a highly ordered system. Understanding these relationships will give you the ability to visualize the backbone of the database system. This understanding will make you a better end user of Microsoft Navision—or any modern database system for that matter—and develop your intuition and resulting in your ability to tailor Microsoft Navision to your firm’s particular needs.

### 4.2.1 How Not to Organize Data: A Negative Example

You might ask: Why is the structure of the Microsoft Navision database so complicated? And, why seeming related information is often put into completely different tables? You might think it would be more efficient to have the information about the buyer of products in a *Sales Order* in the same table with the purchased products. In Microsoft Navision, however, this is not the case. Instead, the table with information concerning the customer's purchased products is located in the *Sales Line* table while the customer's name and address are located in the *Sales Header* table. This is apparent in the layout of the *Sales Order* in Microsoft Navision. Therefore, *Sales Order* is made of two linked tables: one holding the sales header information and the other the sales line information.



**Figure 4.1** Two-part *Sales Order*

*Fundamentals of a relational data system*

The reason Microsoft Navision uses several different tables to do a single function—like describing a *Sales Order*—has to do with a handful of ingenious principles that are the basis for all relational databases. You must get a handle on the basic ideas behind a relational database before the inner world of Microsoft Navision will make sense. With this knowledge, you will under-

stand how to link the information in one table to the information in another.

Below are the four most important development concepts to be covered in this book:

- Setting filters on a set of tables
- Linking these tables
- Searching through these tables
- Making decisions based on what is found in the tables

Microsoft Navision promises quick, accurate and stable information, which it can deliver because it follows the principles of a relational table layout.

*Without table relationships – Ouch!*

The best way to appreciate the beauty of a relational table system is through a negative example. Let us consider what it would be like if you tried to handle your company's information by using only large, self-contained tables. Specifically, imagine handling your firm's sales orders with one large and complete table. Ignoring the relational principles, you think it would be a good idea to put everything that has to do with your customers and the products they order into a single sales order table.

Order Type	Customer Name	Address	Product	Packing Unit	Quantity	Price
Promo	Jill Keehner Ltd.	Ashburn, VA	Wheat Beer	carton	100	20
Normal	Reuben's Brewery	Columbus, Ohio	Smoked Hopps	case	1000	20
Normal	Porsche	Stuttgart, Deut.	Smoked hopps	Bottles	200	2
Promotion	Reuben Brewery	Columbus, OH	smoked-hopps	glass bottles	300	2
Norm	Porsch	Stutgard, Germany	Wheat-beer	box	24	20
Offer	Jill Keehner inc.	Ashburn, Virginia	weiss beer	container	56	20
Normal	Porsche AG	Stuttgart	Hopps-smoked	rack	123	20
Promo	Reuben Brew Ltd.	Columbus OH	smoke-hopps	bottle	400	2
Standard	J. Keehner Co.	Ashburn VA	Wheat-bier	bunch	1000	20

**Figure 4.2** Sales order information entered into a single table

Suppose that each time a sales order was created a worker entered the order number, customer information, product information and price for each product purchased. By working in this manner you only need one instead of several tables and have the advantage of having all of the information in one place. You

might think that using a single table will make searching for information a snap.

Let us now consider some of the disadvantages of organizing data in this manner. The only readily apparent disadvantage is that for each line you must enter all information manually. This means you must reenter all the customer information again even if there is a repeated customer, as seen in the above example. At a glance, this may not seem like that much extra work, however, consider the following question: If the same customer name must be written into your table several times, how can you be 100% certain that it will be spelled consistently—that is, exactly the same each time? Furthermore, if customer names are not spelled consistently, how can you automate a search which calculates, for example, the sales of this customer over a several years?

If you cannot be certain that the spelling is consistent, you must do the sales calculation by hand or hope for the best from the automatic process which will return an answer that is at best an approximation. As you look more closely at the above example, you will be able to imagine other plausible problems. This kind of uncertainty in an ERP system is disastrous.

What if your customer changes their address in a few years? That's right. You will have to go through each position where you find this customer and change it manually. You cannot trust the computer to do this because you cannot be sure how the customer's name is spelled throughout the table.

*Correct data organization*

As a rule, if it is possible for an error to occur, it will; it is merely a question of when and where. Therefore, you must maintain a very high standard of structure and clarity in everything you do with respect to your ERP system. Unfortunately, this means that your massive table above would require too much care to maintain and would quickly become untrustworthy. You could never hope to balance accounts to the penny for thousands of customers, buying thousands of products, over many years, if you used the system above. You could not even manage a small firm with such poorly organized data. So, how can we get around these problems that arise due to ambiguity and human error?

## 4.2.2

### **Table Relations: Maintaining the Integrity of Your Information**

The solution is to break your information into pieces. Next, you must force a strict relationship between these pieces using a table hierarchy, wherein some tables control the information in

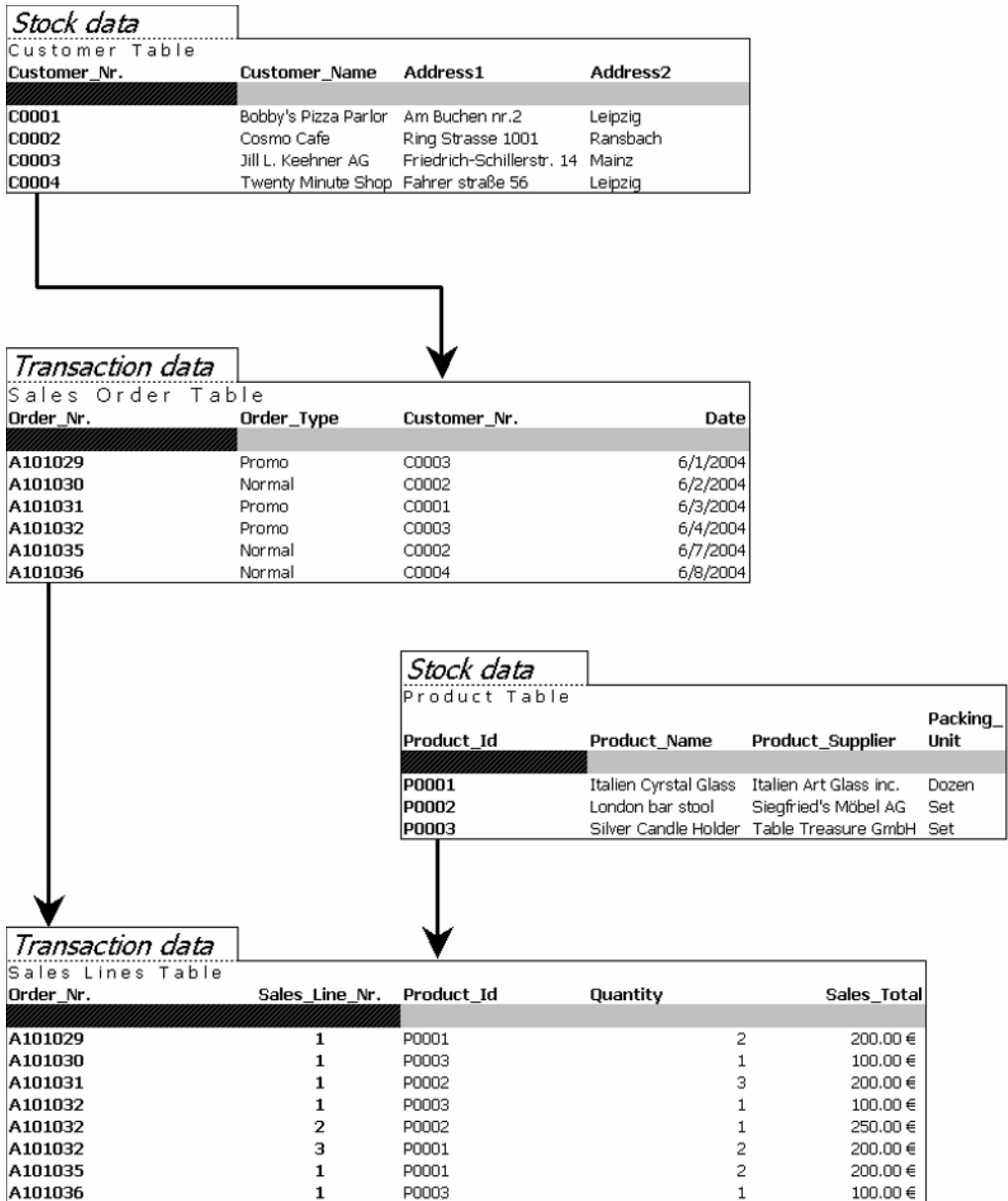
other tables. Using this method, you need only input a piece of information once and in one table. Everywhere else this information is needed, you will simply create a link to the original entry in the parent table. Through a system of links between tables, an automatic copy of the single original entry is present in other tables. This eliminates the possibility for ambiguity or misspellings because there is one authoritative description of a fact. It is true you will have more complexity due to table diversity, but the data in these tables will be 100% consistent. This type of table layout minimizes the storage space needed for your database because information is stored once instead of being repeated over and over again. If you need to update a fact, such as a customer's address, you can do it in one place and the links to the original will automatically be updated.

It is reasonable to allow your workers to be human and therefore make mistakes, however, with a good relational data table structure, you can still expect to have perfect data.

### 4.2.3 Relational Data System Example: The Sales Order

First, let us consider a visual example of a relational approach to the sales order problem before discussing the logic behind it (See figure 4.3).

These tables contain exactly the same stock and transaction information as our single table above. You will notice in this layout each unique piece of information is entered once and is furthermore handled by links between table line IDs such as *Customer\_No.* or *Order\_No.* These IDs are like addresses in the tables and keep things in order.



**Figure 4.3** Sales order information organized with table relationships

Suppose you would like Microsoft Navision to give you a perfect sum of the open sales for the customer, Cosmo Café. Microsoft Navision can, through an automated application, come up with a perfect result, quickly. The system arrives at this result simply by opening each line of the *Sales Order* table where it finds the customer number (C0002) for *Cosmo Café*. For each *Sales Order* it finds, it will open each *Sales Line* having a matching *Order\_No.* and keep a running total of the *Sales\_Total* variable. Such a process is quick, and more importantly correct, thus eliminating any ambiguity or reservations you may have about the results. As you can see, the advantages of the relational database over the one-table layout are tremendous.

Let us walk through your *Cosmo Café* example following the same process as Microsoft Navision (See figure 4.4).

To design a relational structure you will need to think about what types of information you are working with. You must compare each of the fields in the table with each other to see if they belong in the same table. By comparing the variables to one another you will develop an idea of the relationships between the variables. You will be able to see what variables belong together and what variables do not. Here are some questions to consider which will help you design an effective relational table superstructure:

*Questions you  
can ask to im-  
prove your table  
relations*

- Is one variable a stock variable, like *Customer\_Name* and the other a transaction information type, like *Sales\_Order\_Date*? If so, they should be in separate tables.
- Is there a causal relationship between our variables? For example, does the existence of a sales order depend first on the existence of a customer? If so, then the sales orders should be in a separate table which is partly determined by the customer table.
- How many of one variable can there be for every one of the other variable and vice versa? For example, how many customers are there for any single sales order and vice versa?

The third question may seem very general and abstract, however, it is the driving concept behind the other questions and therefore very important.



Chooses "Cosmo Cafe" for your sales enquiry by entering "Cosmo Cafe" in the Customer\_Name Table filter.

Customer Table			
Customer_Nr.	Customer_Name	Address1	Address2
C0001	Bobby's Pizza Parlor	Am Buchen nr.2	Leipzig
C0002	Cosmo Cafe	Ring Strasse 1001	Ransbach
C0003	Jill L. Keehner AG	Friedrich-Schillerstr. 14	Mainz
C0004	Twenty Minute Shop	Fahrer straÙe 56	Leipzig

Customer\_Nr. "C0002" is the key to search for "Cosmo Cafe" orders

Sales Order Table			
Order_Nr.	Order_Type	Customer_Nr.	Date
A101029	Promo	C0003	6/1/2004
A101030	Normal	C0002	6/2/2004
A101031	Promo	C0001	6/3/2004
A101032	Promo	C0003	6/4/2004
A101035	Normal	C0002	6/7/2004
A101036	Normal	C0004	6/8/2004

Order\_Nr.'s from the Sales Lines Table are searched in Sales Lines Table.

Sales Lines Table				
Order_Nr.	Sales_Line_Nr.	Product_Id	Quantity	Sales_Total
A101029	1	P0001	2	\$200.00
A101030	1	P0003	1	\$100.00
A101031	1	P0002	3	\$200.00
A101032	1	P0003	1	\$100.00
A101032	2	P0002	1	\$250.00
A101032	3	P0001	2	\$200.00
A101035	1	P0001	2	\$200.00
A101036	1	P0003	1	\$100.00

\$100,00

+

\$200,00

---

**C0002: Cosmo Cafe**      **Σ**      **\$300.00**

**Figure 4.4** Summing the history of a customer's sales

It may help to think of your table layout like a family tree where every child has only one set of parents while every set of parents may have any number of children. Similarly, for every sales order there must be a single customer who ordered it while for every

customer there can be many sales orders. When you clearly see a quantitative relation between two sets of information you should place these two sets of information into two separate but linked tables.

Perceiving a one-to-many relationship between two sets of information is the most important skill you can learn to help you build a high-quality information structure. In practice, you do this all the time in your everyday thinking and decision-making without perhaps realizing it. For example, you know that it is possible to receive many paychecks but that it is not possible that a single paycheck you receive belongs to many other people. Learning to become a good developer means being conscious of the rules of order, relationships and decision-making which govern the real world.

*Keeping order  
with primary keys*

Another important aspect of a relational table system is that it requires tables to have certain orderly properties. Each table in a relational table system must be linkable to its parent and children tables. As you have already seen, this linking can be done by finding the line or record ID of the parent table in the records of the child table. In order for this system to be orderly and expandable, each table must have an *ID* column set that serves as a unique address for each record in that table. The conventional name for such a table record ID is the primary key of that table. For example, the primary key of the customer table is the *Customer\_No.* and the primary key for the *Sales Order* table is the *Order\_No.*

Therefore, in a relational database—a system of related tables—there must be a unique address for every record. This address can be made from a single field, like *Product\_No.* or it can be a set of fields, such as *Order\_No.* and *Sales\_Line\_No.* All that is required is that the information in the primary key be unique in the entire table.

*Definition of  
secondary keys*

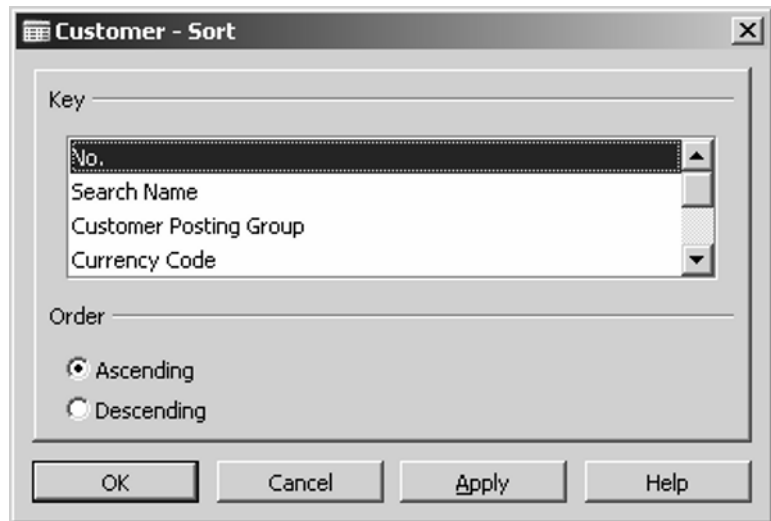
Once the primary key of a table is established, it can be copied into other children tables—that is tables dependent on it—and used as a reference back to the parent table (in the same way that, for example, you receive a name from your parents as opposed to them receiving a name from you). Conventionally, the primary key of any table is called a “foreign” or “secondary key” when it appears in a child or dependant table. For example, the *Customer\_No.* is a foreign or secondary key when we find it in the *Sales Order* table.

The rules governing a foreign key are less strict than those governing a primary key. Just like a single set of parents can have many children, so an ID can be repeated many times in the child table where it is a foreign key. Because primary keys are used and referred to in other tables, you can set up a perfect one-to-many relationship between tables. These one-to-many table relationships allow you to create a stable and trustworthy data organization system.

In Microsoft Navision you will see that the primary key of every table is always the first sorting option available in the *Sorting* option window. Go to:

- **Sales & Marketing > Order Processing > Customers**

and click on the Sorting tool icon (Shift+F8 or the sixth icon tool from the left). The following window appears:

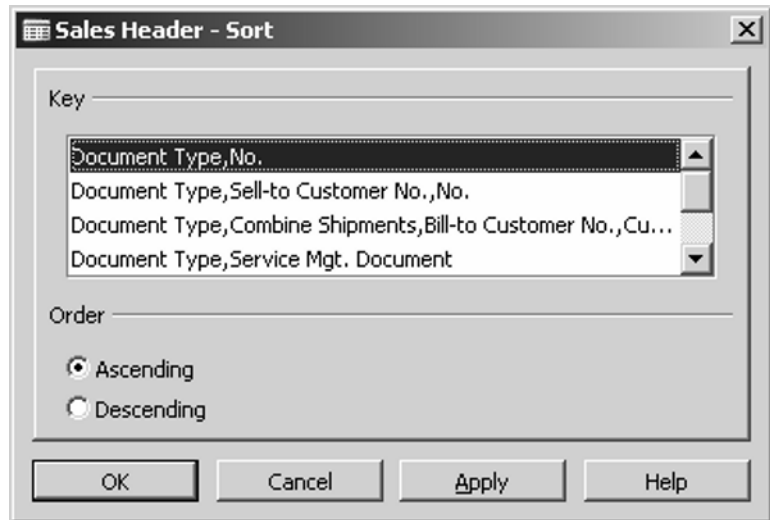


**Figure 4.5** Customer - Sort options

As expected, the field, *No.*, is listed as the first sorting option which tells you that this is Microsoft Navision's primary key for the customer table.

Look at the actual sales order in Microsoft Navision to see if it is using similar primary key strategies as in the *Sales Order* table organization above. Go to:

- **Sales & Marketing > Order Processing > Orders**



**Figure 4.6** *Sales Header - Sort* options

and click on the *Sales Order Header* and then on the Sorting tool icon. The following window appears: (See Figure 5.6 above)

Here you see that the primary key for the *Sales Order Header* in Microsoft Navision consists of two fields: *Document Type* and *No.* The *No.* field is to be expected from the example above but why does Microsoft Navision use a second piece of information in its primary key? To answer this question, use the Zoom tool to see what information is hidden in the *Document Type* field. Go to:

- **Tools > Zoom**

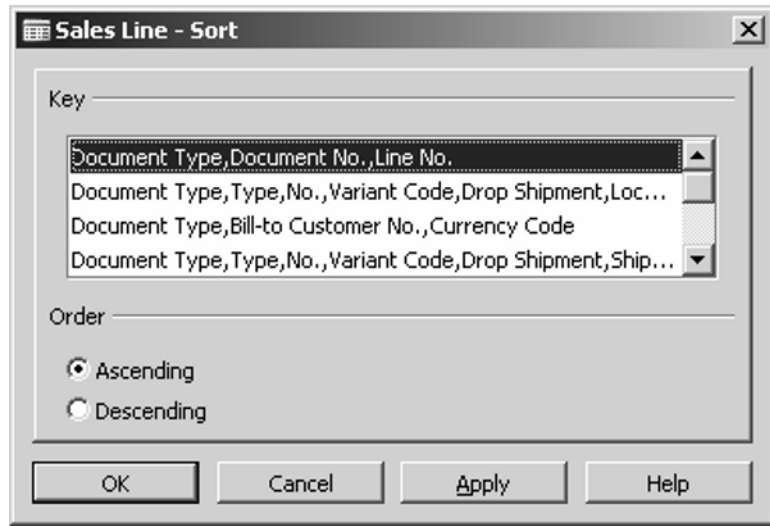
The following window will appear:

Field	Value
Document Type	Order
Sell-to Customer No.	20000
No.	2002
Bill-to Customer No.	20000
Bill-to Name	Selangorian Ltd.
Bill-to Name 2	
Bill-to Address	153 Thomas Drive
Bill-to Address 2	
Bill-to City	Coventry
Bill-to Contact	Mr. Mark McArthur
Your Reference	
Ship-to Code	
Ship-to Name	Selangorian Ltd.
Ship-to Name 2	
Ship-to Address	153 Thomas Drive
Ship-to Address 2	
Ship-to City	Coventry
Ship-to Contact	Mr. Mark McArthur
Order Date	01/17/01
Posting Date	01/17/01
Print Date	01/17/01

**Figure 4.7** Sales Header - Zoom

Here you can see that the information in *Document Type* tells Microsoft Navision that the *Sales Header No.*, 2002, is an *Order*. The fact that the primary key of the *Sales Header* table is made of two parts tells us that the same number or code in the *No.* field can be used more than once, but never for the same type of document or *Document Type*. In practice this means you could have a sales *Quote* with the number 2002 and also an *Order* with the number 2002 but never two *Quotes* or two *Orders* with the same number.

Next, look at what Microsoft Navision uses in the sales lines for a primary key. Press ESC to leave the Zoom function, then click on a sales line and then on the Sorting tool icon. The following window will appear:



**Figure 4.8** *Sales Line - Sort* options

As you would expect, Microsoft Navision uses the primary key of the *Sales Header* plus the field, *Line No.* as a counter of the sales lines themselves. This is similar to the example above where the *Order\_No.* and *Sales\_Line\_No.* were used to organize the *Sales Lines* table.

*Behind every drill down a secondary key is operating*

Every field with a drill down function in Microsoft Navision is a foreign or secondary key. For example, if, when in the *Sales Header*, you click on the customer number field, Microsoft Navision opens the customer table and allows you to choose a customer from the list. When you have selected a customer, Microsoft Navision inserts the ID from that customer into the *Sales Order Header*.

Thus, you can see very clearly the principles of a relational table system working throughout Microsoft Navision. Microsoft Navision is powerful and flexible, yet always maintains order by enforcing the principles of a relational table system.

## 4.3

### The Importance of Presentation

In this section we will discuss the complexity and the advantages of Microsoft Navision's layered structure. We will also look at an example of this layered structure.

### 4.3.1 Different Views for Different Purposes and Users

To master the Microsoft Navision working environment you must understand that information is layered and that objects exist in a hierarchy. Under all Microsoft Navision forms, table views and reports are the tables themselves where the raw data is stored. Data which is not in the tables neither exists to be viewed or used in the forms, table views or the reports. The end user never sees these tables and therefore many things will remain forever mysterious to them.

*Tree-like structure of Microsoft Navision*

A metaphor for understanding the layout of Microsoft Navision is to think of the system as a layered hierarchy or tree. The tables are the roots of the system, and like roots, the tables are hidden from the sight of normal viewers. The forms and masks are the leaves and branches which form the surface of the system. The reports are the final fruits of the system—the output. It is important to learn how to visualize this layered structure. For example, you cannot create a report based on the information displayed in a mask without first knowing the table source on which the mask is based. At first glance, this complex organization may seem strange, but once you understand it, you will realize that it is worth a million spreadsheet solutions.

One of the primary advantages of this layered, tree-like structure is the ability to choose how the end user will see the information and how they can work with it without having to change the basic structure of the system. For example, you could choose to have a view in the *Sales Order* where you see all the contents of your open sales in a single list. A different view of the sales lines might involve the user actually entering articles into a *Sales Order*. These are two quite different views, but in reality they are in the same table; in this example, the underlying table is our *Sales Line* table. The end user may never discover that the difference between these two views is merely one of presentation, however, if you want to achieve a deeper understanding of the application, you must be able to discover these hidden relationships.

### 4.3.2 General Ledger Account Table: Two Distinct Views

Let us consider another example of two different presentations of the same table as a way to introduce the Microsoft Navision development environment. The *G/L Account* table is found in the following two places:

- **Financial Management > General Ledger > Chart of Accounts**
- **Sales & Marketing > Order Processing > Order**

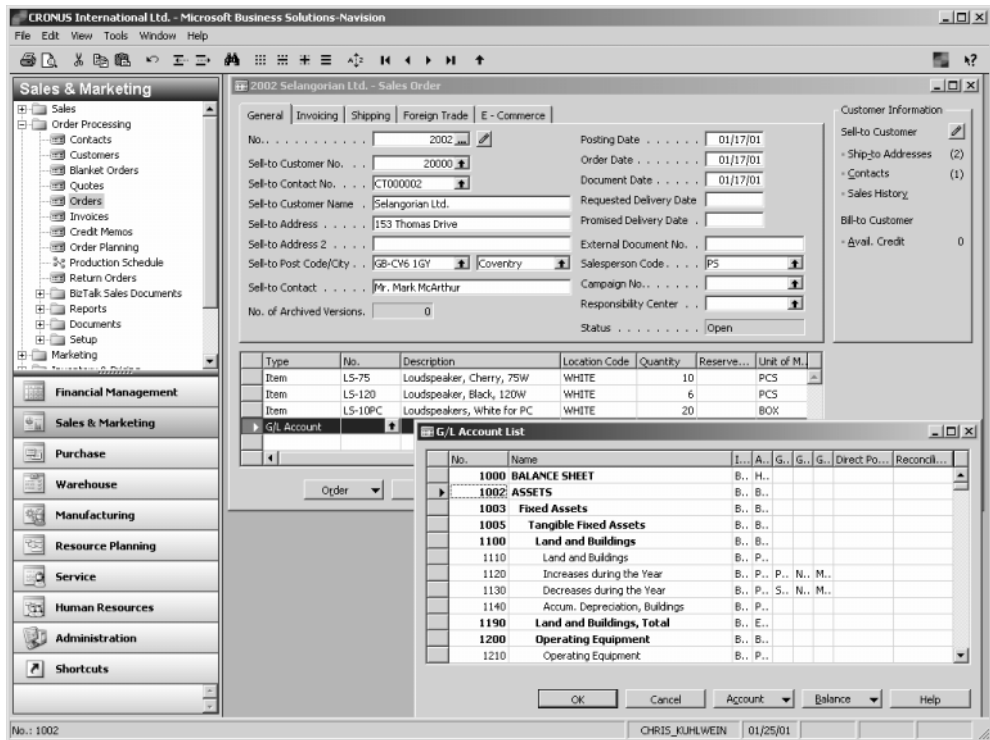
Click into a sales line in the *Sales Order*. Next, drill down into the *Type* column within the sales line and select the *G/L Account* option. Next, drill down into the *No.* column.

Here are the two views:

No.	Name	I..	A..	Totaling	G..	G..	G..	Net Change	Balance
▶ 1000	<b>BALANCE SHEET</b>	B..	H..						
1002	<b>ASSETS</b>	B..	B..						
1003	<b>Fixed Assets</b>	B..	B..						
1005	<b>Tangible Fixed Assets</b>	B..	B..						
1100	<b>Land and Buildings</b>	B..	B..						
1110	Land and Buildings	B..	P..					1,479,480.60	1,479,480.60
1120	Increases during the Year	B..	P..		P..	N..	M..	147.73	147.73
1130	Decreases during the Year	B..	P..		S..	N..	M..		
1140	Accum. Depreciation, B...	B..	P..					-526,620.38	-526,620.38
1190	<b>Land and Buildings, To...</b>	B..	E..	1100..1190				953,007.95	953,007.95
1200	<b>Operating Equipment</b>	B..	B..						
1210	Operating Equipment	B..	P..					582,872.18	582,872.18
1220	Increases during the Year	B..	P..		P..	N..	M..	25,116.00	25,116.00
1230	Decreases during the Year	B..	P..		S..	N..	M..		
1240	Accum. Depr., Oper. E...	B..	P..					-508,176.74	-508,176.74
1290	<b>Operating Equipment, ...</b>	B..	E..	1200..1290				99,811.44	99,811.44
1300	<b>Vehicles</b>	B..	B..						
1310	Vehicles	B..	P..					49,473.91	49,473.91

**Figure 4.9** *G/L Account* table as seen from *Chart of Accounts*





**Figure 4.10** Drop-down list of the sales line field, *No.*, when *Type* is equal to *G/L Account*

You should notice immediately that there are no financial sums in the second view.

As is generally the rule, employees in the order entry department should not see the firm finances. If the end user tries to find the hidden financial information by adding hidden columns with the

- **View > Show Column**

function or by looking into the

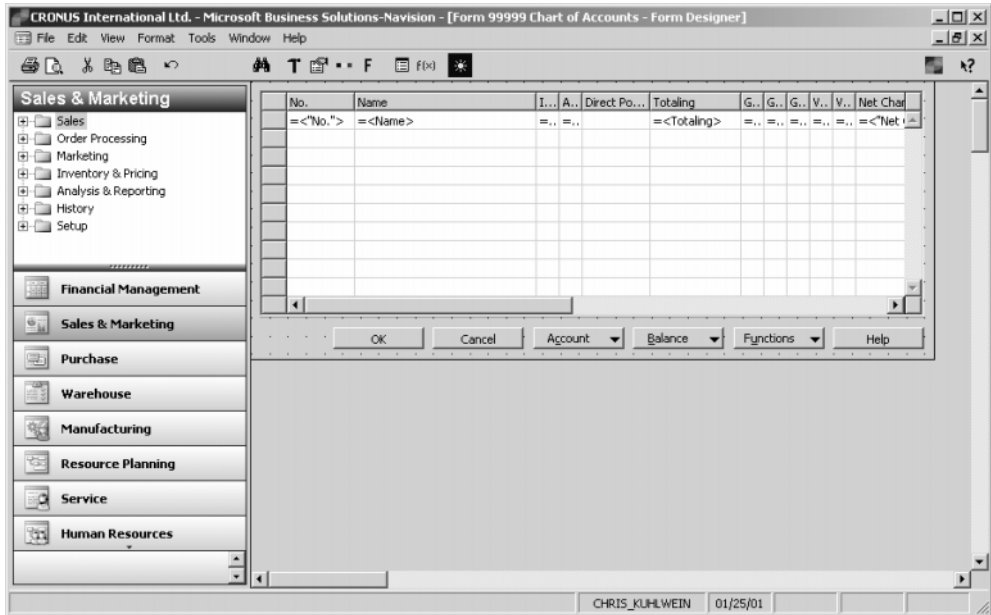
- **Tools > Zoom**

function they will not see this financial information. One list view automatically shows the finances of the entire firm and the other shows only the names and numbers of the financial accounts. However, these objects are merely two different list views of what in reality is the same *G/L Account* table.

Let us begin with the *Financial Management, Chart of Accounts*. To open the inner structure of the *Chart of Accounts*, press CTRL+F2 or go to:

- **Tools > Designer**

The following view will appear:



**Figure 4.11** Inner structure of the *Financial Management* view of the *G/L Accounts* table

### *FormDesigner*

This is the Microsoft Navision *FormDesigner* environment. Move horizontally through the columns until you see the *Net Change* column. This is one of the fields that shows the financial sums in the finished object. Next, click outside of the form in the empty gray area (this will automatically select the properties of the object as a whole) and then go to:

- **View > Properties**

The following window will appear:

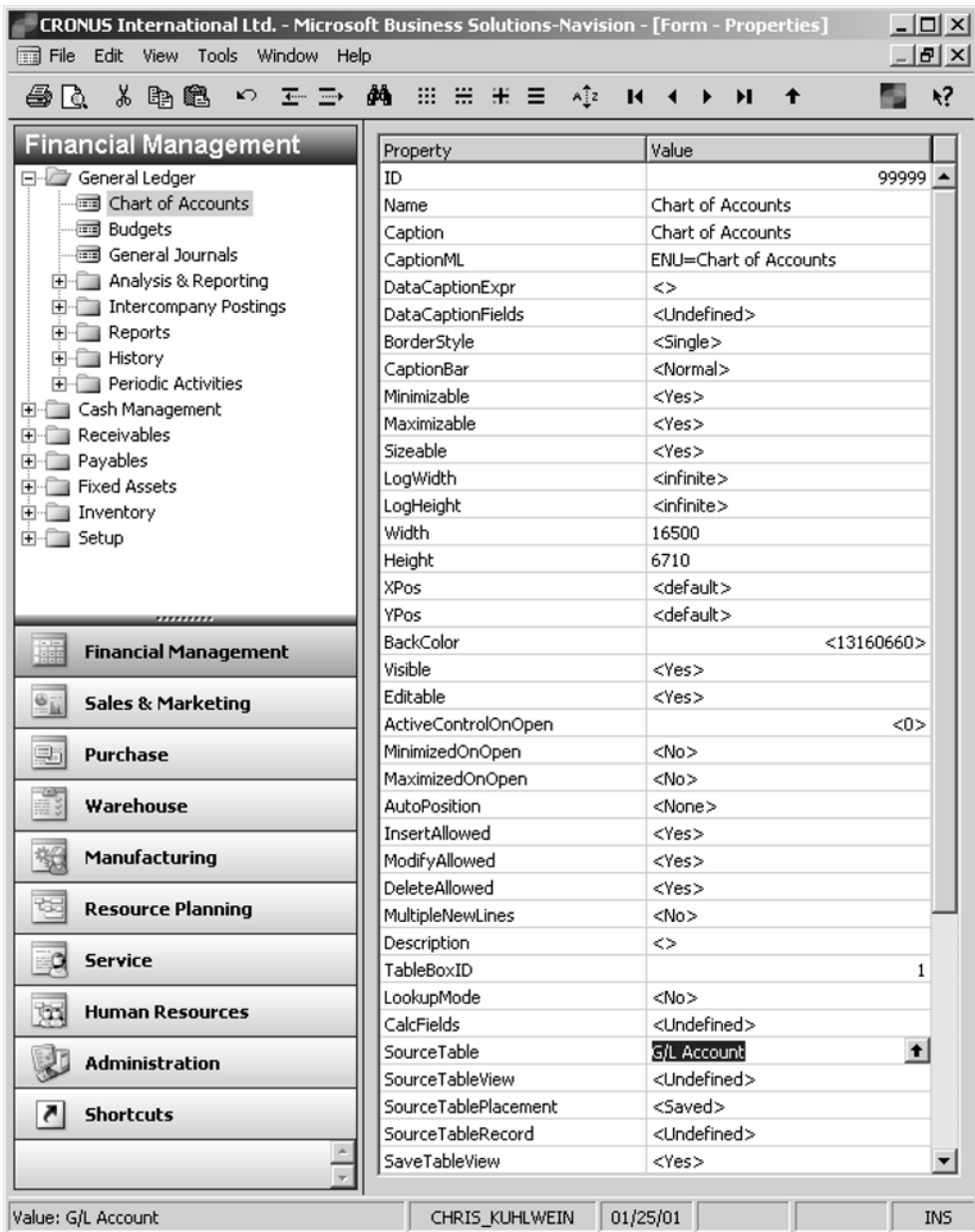


Figure 4.12 Form - Properties window

These are the many functions that you can use to control the form. The most important one, which we will consider now, is the *SourceTable*.

*SourceTable* –  
determines the  
form object

You can move through the list by repeatedly pressing the first letter of the name of the function or name which you wish to find. In this case, click on the left column and press the S key on your keyboard until you come to the control, *SourceTable*.

As you can see the table name that has been selected after *SourceTable* is *G/L Account*. This is the table on which the *Chart of Accounts* form is built. The *G/L Account* table is never seen directly by the end user. The *Chart of Accounts* form object is a mere presentation of information contained within the *G/L Account* table. In the form you can control the end user's access and view of the *G/L Account* table. While the form is a mere presentation of the *G/L Account* table, changing its controls will not effect the properties in the original *G/L Account* table object on which this object is based. You need not worry about damaging your *General Ledger Account* information through changes and tests you make in the form.

Do not worry about understanding all of these control types; many are obscure and infrequently used. As you read through this book you will gain a better understanding about which types are important.

As you can, see some of the property controls, such as *Editable* <Yes>, are defined in a way that makes sense for a *G/L Account* presentation in the *Financial Management* area. In the *Financial Management* area it is necessary for the user to have the rights and ability to change, add and delete accounts. Therefore, the *Editable* property is defined as <Yes>.

Now, if the *Editable* control in the *G/L Account* table is defined as <No>, then regardless of what is defined in the form, the end user will not be able to edit the contents of the *G/L Account* table. That is to say, the controls in the tables have a priority over objects like forms and list views which are based on them. Referring back to the tree diagram, we can expect that what is not available in the table or roots will not be available either to the rest of the system.

Now consider the second list view of the *G/L Account* table. To get to this second view of the *G/L Account* table, press the ESC key. When the window that asks you if you want to save any changes appears, reply, "No." Next, go to:

- **Sales & Marketing > Order Processing > Order**

Click into a sales line in the *Sales Order*. Next, drill down into the *Type* column within the sales line and select the *G/L Account* option. Next, drill down into the *No.* column.

Now the following *G/L Account List* view appears:

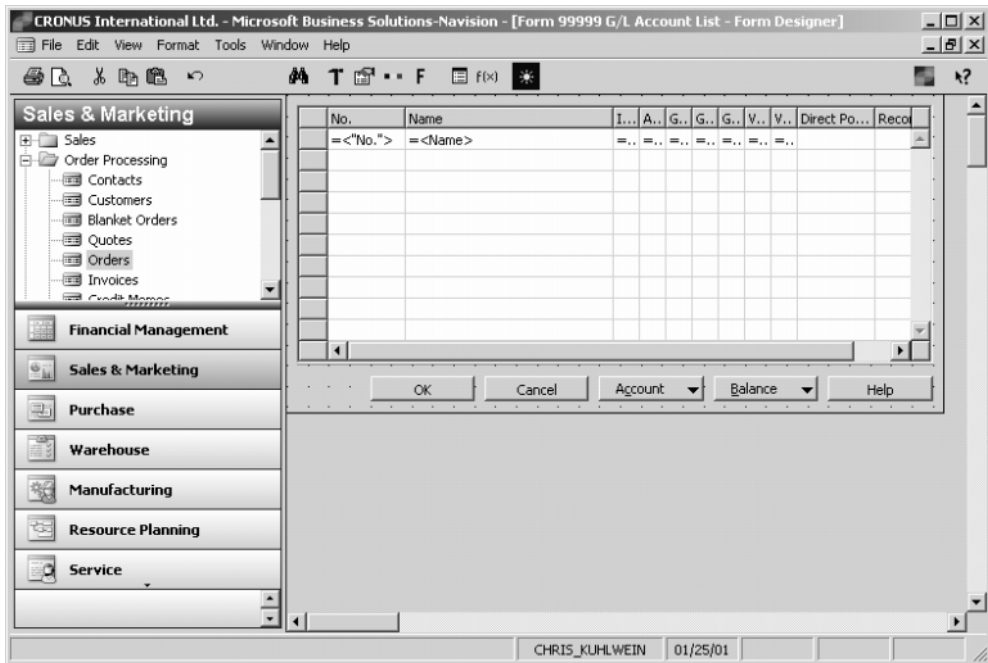
No.	Name	I..	A..	G..	G..	G..	Direct Po...	Reconcili...
1000	<b>BALANCE SHEET</b>	B..	H..					
1002	<b>ASSETS</b>	B..	B..					
1003	<b>Fixed Assets</b>	B..	B..					
▶ 1005	<b>Tangible Fixed Assets</b>	B..	B..					
1100	<b>Land and Buildings</b>	B..	B..					
1110	Land and Buildings	B..	P..					
1120	Increases during the Year	B..	P..	P..	N..	M..		
1130	Decreases during the Year	B..	P..	S..	N..	M..		
1140	Accum. Depreciation, Buildings	B..	P..					
1190	<b>Land and Buildings, Total</b>	B..	E..					
1200	<b>Operating Equipment</b>	B..	B..					
1210	Operating Equipment	B..	P..					
1220	Increases during the Year	B..	P..	P..	N..	M..		
1230	Decreases during the Year	B..	P..	S..	N..	M..		
1240	Accum. Depr., Oper. Equip.	B..	P..					
1290	<b>Operating Equipment, Total</b>	B..	E..					
1300	<b>Vehicles</b>	B..	B..					
1310	Vehicles	B..	P..					
1320	Increases during the Year	B..	P..	P..	N..	M..		
1330	Decreases during the Year	B..	P..	S..	N..	M..		
1340	Accum. Depreciation, Vehicles	B..	P..					
1390	<b>Vehicles, Total</b>	B..	E..					
1395	<b>Tangible Fixed Assets, Total</b>	B..	E..					
1999	<b>Fixed Assets, Total</b>	B..	E..					
2000	<b>Current Assets</b>	B..	B..					
2100	<b>Inventory</b>	B..	B..					
2110	Resale Items	B..	P..					
2111	Resale Items (Interim)	B..	P..					
2112	Cost of Resale Sold (Interim)	B..	P..					

**Figure 4.13** Drop-down list of the sales line field *No.* when *Type* is equal to *G/L Account*

This above view shows no financial sums and cannot be edited by the end user. Press CTRL+F2 on your keyboard or go to:

- **Tools > Designer**

Seen below, you have opened the Microsoft Navision *Form Designer* within the development environment. The *Form Designer* is used to create and edit forms and list views.



**Figure 4.14** Inner structure of the *Sales Order* view of the *G/L Account* table

Move through the object window—toward the left. You will notice there are no *Net Change* or *Account Balance* columns. Next, click out into the empty, gray area around the form object. Microsoft Navision selects the object as a whole. Next, go to:

- **View > Properties**

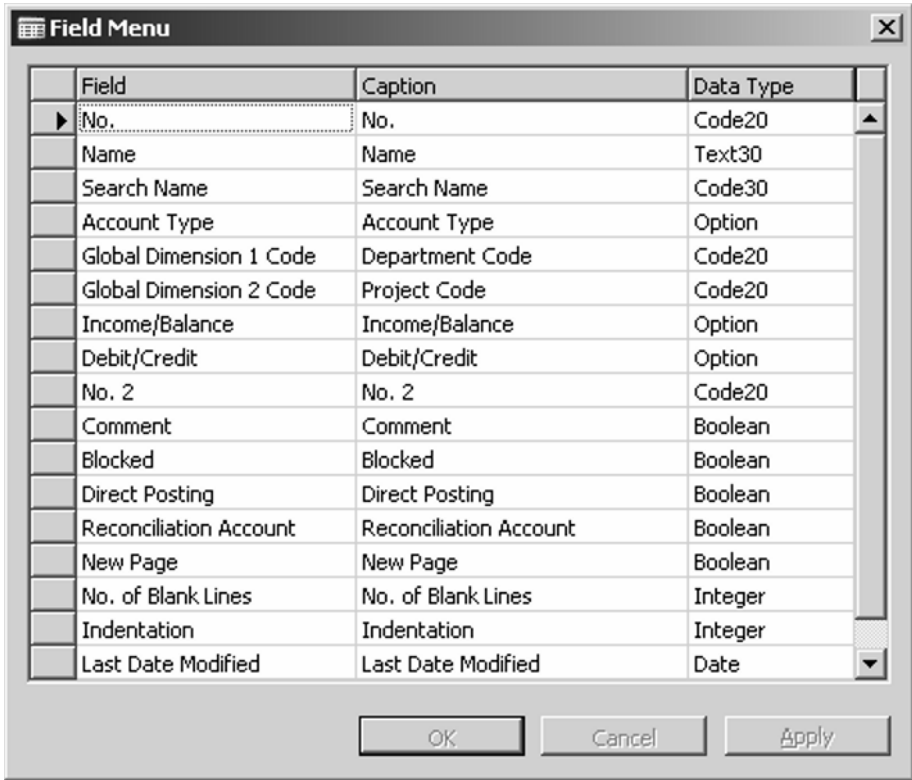
Find the *SourceTable* property towards the bottom of the properties list.

Again we see that our source table is *G/L Account*. The same table that we found under the *Chart of Accounts* view. However, in the property list of this object we find that the property, *Editable*, is set to *<No>*. In this way the *G/L Account* view will be closed for changes by the end user.

Now press ESC on your keyboard to return to the designer window. Go to either

- **View > Field Menu**

or click on the third icon from the right side of the icon menu located directly above the object editing area. The following window will appear:

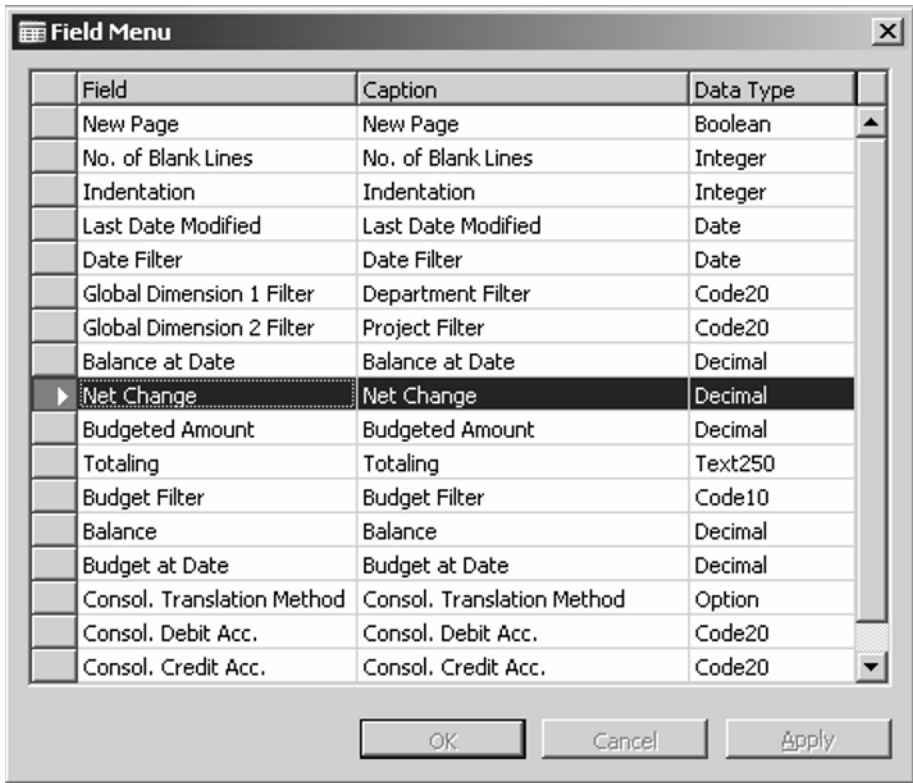


**Figure 4.15** *Field Menu* window

The *Field Menu* shows you the complete list of all the fields that exist in the table on which this object is built. In this list you will see the field names with their field type and length in the adjacent column. In the *Field Menu* you will see all the possible fields that you can place in your form or list view object. There are many fields which do not exist in your form.

*Making changes  
in forms or list  
views*

To add a column into the list view you must select an entire row in the *Field Menu*. To do this, simply click on the gray bar directly to the left of the field you desire until the entire row is highlighted in blue. See the following example:



**Figure 4.16** *Field Menu* window

To delete a column, simply click on it and press DELETE on your keyboard. It is very often the case that you must add or remove fields from your list views so you should get comfortable with this function. For example, there are times when you will like to have as few fields as possible so that your list view is easy to work with, however, at other times you may want to see a rarely used field.

#### *Showing & Hiding columns*

The best way to accomplish a happy medium between efficiency and completeness in your list views is to place the most important fields first. Less important fields can be automatically hidden when the table view is opened. To hide fields, click on the column you wish to automatically hide and then go to:

- **View > Properties**

Scroll down until you find the control, *Visible*. Set the *Visible* control to *No*. When the end user opens the list view this field will be hidden automatically but is available via menu:



*SourceExpression* determines the column's field

- **View > Show Column**

Next, click on an individual field in your object and then go to:

- **View > Properties**

You will see a slightly different list of control possibilities. You may notice that we are in a property list where there is no *SourceTable* control. This is because you have entered the properties function after selecting an individual field which means that these properties apply only to this specific field. Within a specific field you cannot choose a table but you can choose a field from a table that has been defined for the object as a whole. Here, instead of finding *SourceTable*, we find *SourceExpression*. A nice feature of Microsoft Navision is that it allows you to enter a formula into this *SourceExpression* control and not just a field. For example you could write here:

“Net Change” - “Balance”

Microsoft Navision will display the results of this formula in your finished form or list view. (Quotation marks are required around the field names when the field names contain spaces such as, *Net Change*, for example.)

The fact that within the field properties we cannot control the tables behind the form and that with the form we cannot control properties set within the underlying tables, demonstrates a hierarchy of presentation in Microsoft Navision. This brings us back to our tree metaphor. You will see in the field properties that we are one more layer removed from our source object, the *G/L Account* table. What this means is that the properties here in the single field are determined not only by the table properties but also by the general card or list view properties.

The *SourceTable* has control over the card or list view and the card or list view has control over the individual fields and columns in the object. This means, in practice, you can make all the columns in the entire card or list view uneditable—all at once—by setting the *Editable* control to <No> in the object as a whole or by selecting individual field columns and setting the *Editable* within them to <No>.

Presentation is important because it gives you the ability to take the same table and customize the face, the contents as well as the levels of access to underlying tables.

## 4.4 Object Designer: The Development Environment

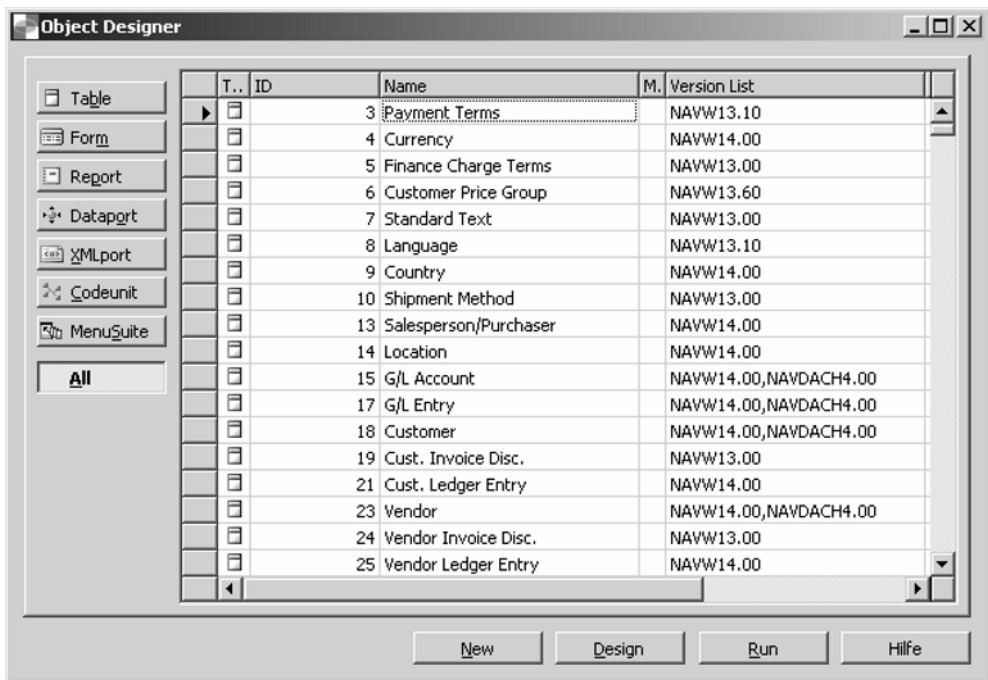
Now let us take a look at the menu and objects used in the development environment.

### 4.4.1 Entering the Inner Structure of Microsoft Navision

As we move ever further from the surface of Microsoft Navision—where the end user works—we begin to enter the developers environment. The main menu in the developer environment is called *Object Designer*. To open this window, go to:

- **Tools > Object Designer**

The following window will appear:



**Figure 4.17** *Object Designer* window

In the previous section we described how Microsoft Navision is created in layers. Here we see these layers represented as different types of objects. These object types are listed on the left side of the *Object Designer* window and include:

- Table

- Form
- Report
- Dataport
- XMLport
- Codeunit
- MenuSuite

The most important objects are the tables—the roots of the system. They contain all the data and fields that the other objects edit or present.

*Object numbering* The objects here are organized not only by object type but also by *ID* number. The *ID* numbers are organized in ranges. For example, the main applications are usually stored in the low number range and optional applications are listed later. These *ID* numbers are important; depending on your Microsoft Navision license, you will have access only to certain object *ID* number ranges. If you have purchased the right from your Microsoft Navision Solutions Center to add new tables, you will only be allowed to insert them in a specific object *ID* number range. It is most likely that you will only be able to insert new tables beginning with the *ID* number, 50000. You must be careful when you are inserting a new object that you do not accidentally overwrite an existing object. You should always observe an unused *ID* number before you try to insert a new object.

From the *Object Designer* window you can quickly move between application objects. Depending on your task, it might be optimal to first locate the object you wish to edit in the end user environment. Once you have found the object in the end user environment, press CTRL+F2 or go to:

- **Tools > Designer**

This action opens the internal structure of the object. If you look top of your screen you will see the object *ID* number. Next, close the object without saving so you can find the object again in the *Object Designer* window.

#### 4.4.2 Working With Table Objects

As you have seen, the foundation of the Microsoft Navision information system is a nexus of highly organized tables linked to each other in a relational organization. In this organization each

table may determine or be determined by other tables. Within each table, every record or line of data is uniquely specified by an absolutely unique address, called a primary key. Within each of these tables, a field that contains a reference to the primary key of another table can be inserted. The reference in any table to the primary key of another table allows Microsoft Navision to link these two tables. This system of links allows, for example, all the information about a customer to remain in the customer table while the customer's primary key is used to mark each of their sales orders located in another table. In this section we are going to put these concepts into practice by looking at useful examples within Microsoft Navision.

*Creating new variables*

Many times you will find that you need to create new variables in Microsoft Navision. This requires you to open a table in Microsoft Navision and add new fields. If you are trying to organize records with your new variable, it is likely that you will also need to create a new table. In the new table you can create a master list that can then be used as a secondary key in the table that you are trying to organize. Let us consider the following example.

*Example*

Suppose your company sells its articles in many types of markets and that certain articles are designed only for specific sales markets. Now, suppose that your boss comes to you and tells you that she wants a report every Friday detailing the best-selling articles in each of the sales markets.

Microsoft Navision has an excellent report which ranks the sales of the various articles called, "Inventory - Top 10 List". This report would be perfect for your sales study, unfortunately, it does not distinguish between articles and the various sales markets for which they are designed. What you need for your study is the ability to run this report for specific articles within a certain sales market while filtering others out.

You need a key on each item that tells Microsoft Navision each item's sales market. This key must be something you can trust, as opposed to merely text which is manually entered into the *Item Card*. If it is manually entered into each item, then you can never be sure that the same sales market category is globally given the exact same name, spelling, and so forth. What we need here is a new table where we can write each sales market category once. Then, this authoritative list can be linked to our *Item Card* where the user can select only from our predeter-

mined list. Let us create a new table where the sales market categories can be stored.

*Creating a table*

To create our new table go to:

- **Tools > Object Designer**

Click on the Table icon on the left of your screen. To add a new table in Microsoft Navision you must have purchased the right to create extra tables from your Microsoft Business Solutions Center. Normally, you can buy the rights to have at least ten tables at a given time. Once you have purchased the rights to these new tables, the Microsoft Business Solutions Center updates your license which you must then upload into your Microsoft Navision system. (See Chap. 3: “Installing your Microsoft Navision license data.”) When you purchase the right to create new tables, you are allowed to insert new tables only within a certain table *ID* number range—which you must obtain from your Microsoft Business Solutions Center.

Suppose that you have the right to create a new table with the table *ID* number 50000. Let us create a new table, *Item/SalesMarket*. First, click the New button at the bottom of your *Object Designer* window. An empty list will appear into which you can enter new table fields. The first field in any table should also be the primary key for the table. Type *SalesMarket* into the *Field Name*, located in the first row in the table field list. The next column is *Data Type*. Next, set the *Data Type* to *Code*. In Microsoft Navision it is a good convention to use the *Data Type Code* for the primary key of a table that stores stock information. In the next column, set the *Length* field to 20.

### 4.4.3

### Data Type

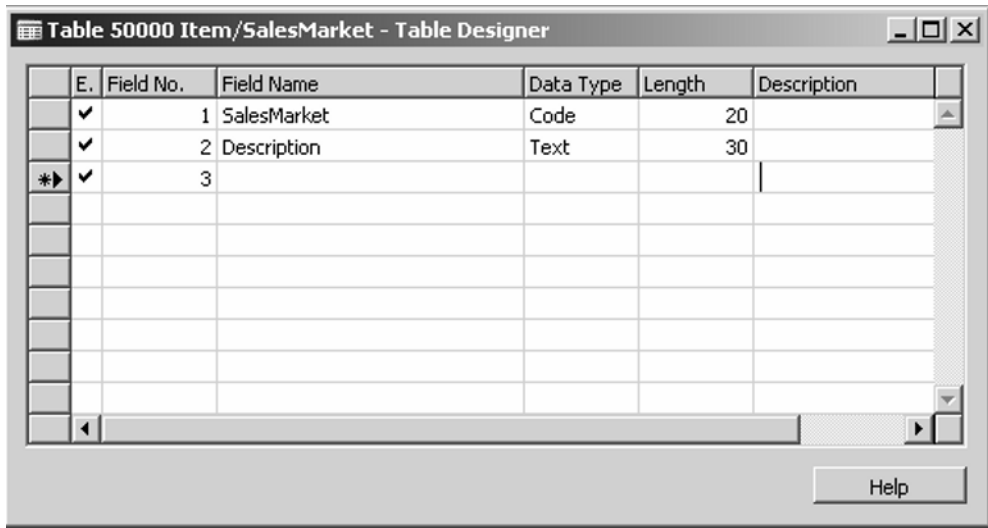
The data type is an important option in Microsoft Navision and in any database. *Data Type* tells Microsoft Navision and your database what kind of information is to be stored in this field, which is useful for many purposes. For example, the data type can tell Microsoft Navision how to format the information which the user enters into the field, thus controlling the user’s input. For example, if we wish to create a field which records the date that a new sales market came into existence, we could create a field called *Market Start date* with *Data Type* equal to *Date*. Then, if the user tries to write something such as “Cat” into this field, they will immediately receive an error message telling them that *Cat* is an invalid date.

The data type is an important option in Microsoft Navision and in any database. *Data Type* tells Microsoft Navision and your database what kind of information is to be stored in this field, which is useful for many purposes. For example, *Data Type* can tell Microsoft Navision how to format the information the user enters into the field, thus controlling the user's input. If you wish to create a field that records the date when a new sales market comes into existence, you could create a field, *Field Name, Market Start date* and enter the *Data Type, Date*. Now, if the user tries to write, for example, *Cat* into this field, they will immediately receive an error telling them that *Cat* is an invalid date. Therefore, *Data Type* limits the type of information allowed in a field.

<i>Integer</i>	Whole numbers between -2,147,483,647 and +2,147,483,647
<i>Decimal</i>	Decimal numbers between -10+E63 and +10+E63
<i>Boolean</i>	Two values TRUE or FALSE
<i>Option</i>	A special numeric field that is stored in Microsoft Navision as an integer but converted to a string upon use. The string is determined by an option string that the user defines in the fields property called <i>OptionString</i> .
<i>Date</i>	Stores a date
<i>Time</i>	Stores a time of day
<i>Text</i>	Stores a text string
<i>Code</i>	A special text string where Microsoft Navision converts the entered text to all uppercase letters and eliminates empty spaces at the beginning and end of the input

These are the basic and most important *Data Types* used to define the contents of a field in a table. Later we will discuss other complex *Data Types* that can be used to create other objects, such as reports and forms. For a more detailed discussion of *Data Types*, please refer to the standard Microsoft Navision documentation.

Now let us create a second field, *Description*, with the *Data Type* equal to *Text* and the *Length* equal to 30. In this field we can enter text descriptions of the *SalesMarket* type. After you have entered these two fields, your new table should look like the following:



**Figure 4.18** Table Designer window

Next, go to:

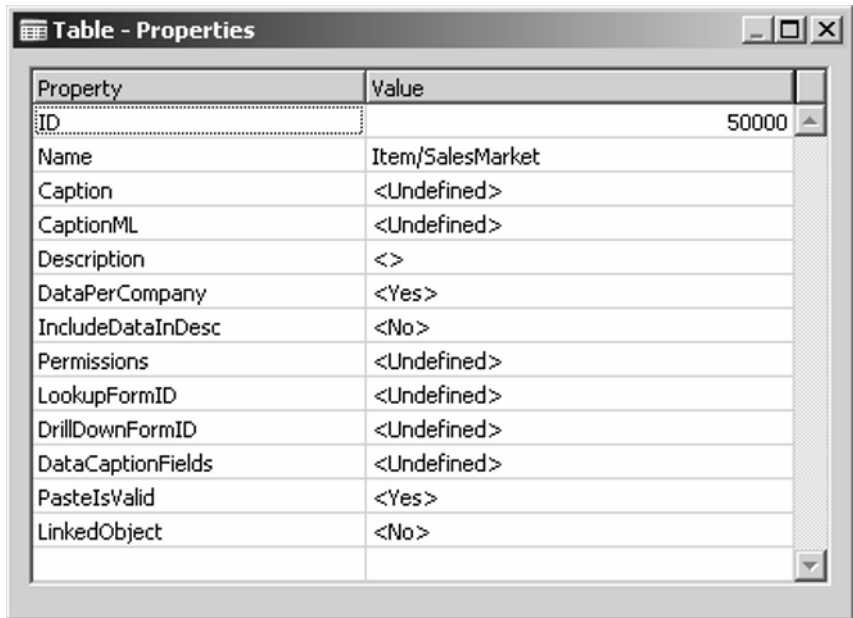
- **View > Keys**

As you can see, Microsoft Navision has automatically entered the first field in your table as the primary key and primary sorting option.

Press ESC and now let us look at the properties of the table as a whole—that is the global properties of the table. To look at the properties of the table as a whole, you must click into an empty field so that you are not selecting the properties of any specific field. To do so, click into an empty field below the two you have already created and then press Shift+F4 or go to:

- **View > Properties**

The following window will appear:



**Figure 4.19** Table's global properties

The following is a list of some general table properties:

<i>ID</i>	The ID number of the table that will be listed in the <i>Object Designer</i> menu
<i>Name</i>	The text name of the table
<i>Description</i>	Space where you describe the contents of your table
<i>DataPerCompany</i>	Here you define whether or not Microsoft Navision should store data separately for each company in the database
<i>IncludeDataInDesc</i>	Here you define whether or not Microsoft Navision includes table data in an import or export of the table object
<i>RemoteServerNo</i>	Used to redirect database inquiries in a C/PLEX server, multiplexer environment
<i>Permissions</i>	Defines extended security permissions for the table
<i>LookupFormID</i>	This gives Microsoft Navision the name of the form to use when the user clicks into a field which is linked to this table
<i>DrillDownFormID</i>	This gives Microsoft Navision the name of the form to use when the user clicks into a <i>Flow Field</i> linked to this table



*DataCaption-Fields*

Tells Microsoft Navision what fields to show in the title of an object, such as a card form

*PasteIsValid*

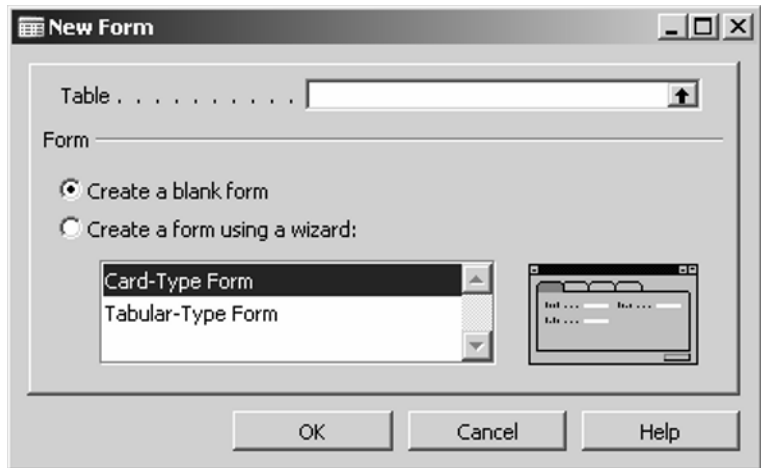
Determines if it is possible to insert data into the table using the Insert or Paste tool

For the purposes of our *Item/SalesMarket* table and for the bulk of Microsoft Navision tables, we will use only a few of these properties. The remainder can be left as they are automatically defined.

The *Item/SalesMarket* table will be a list that opens within the *Item Card*. We will therefore need to create a form object which Microsoft Navision will use to present this table. This will be the form that opens when the user clicks into *Item Card Sales Market*. After we create a form object for the end user to view the *Item/SalesMarket* table we will write the name into the general property of the table, *LookupFormID*.

#### 4.4.4 Creating a New Form

To create a form object of the table, *Item/SalesMarket*, first save and leave this table by pressing ESC. After saving, enter the table name, *Item/SalesMarket*, and *50000* as the table *ID*. Next, go to the *Object Designer* main menu. Click on the Form button on the left side of the *Object Designer* window. Find an available *ID* number in the list of form objects; assume you can use *ID* number 50000 in the form objects. Next, click the New button at the bottom of the screen. The following window will open:



**Figure 4.20** *New Form Wizard*

Above is the *New Form* creation window. Here you can quickly create simple form types, however, for our purposes we will not use the wizard option because it is necessary to learn what is happening at the basic object-building level. This will be of greater benefit in the long run. Ensure that the line, “Create a blank form,” is selected and then click on the OK button.

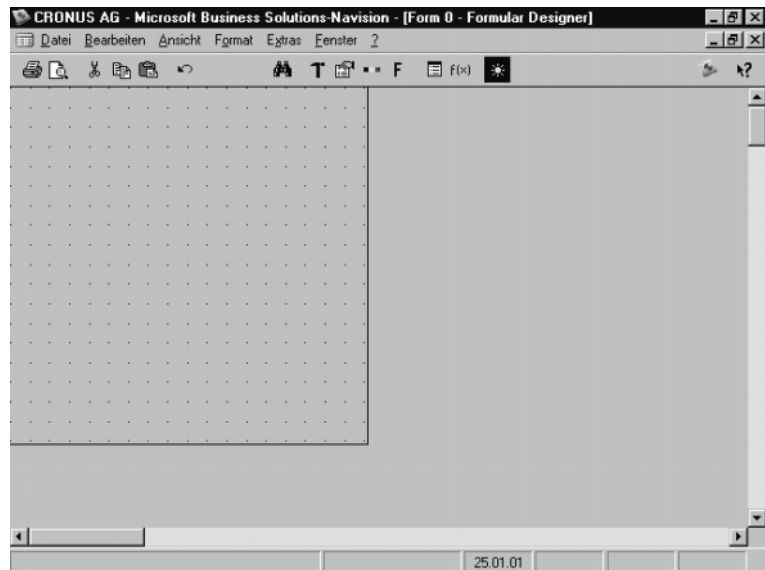
The following window will appear:

*SourceTable*

First we must set the general properties of the form. Go to:

- **View > Properties**

The most important property is *SourceTable*. Here you can write in the name of the table you wish to present: *Item/SalesMarket*. You can also click on the *Value* column of the *SourceTable* and then click on the assist arrow that appears in the left of the *Value* column. Next, Microsoft Navision will open a list of all the potential tables and you can make a selection. After entering the table name, *Item/SalesMarket*, press ENTER and then ESC to return to the surface of the *Form Designer*.



**Figure 4.21** *Form Designer*

*Toolbox*

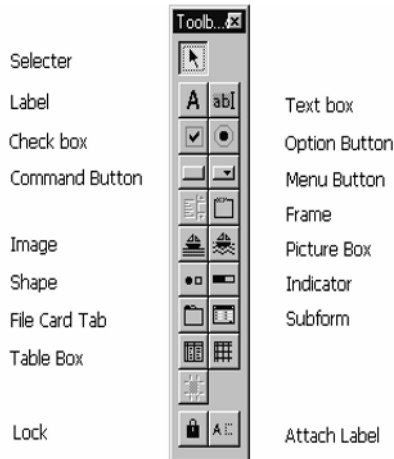
Now we need a template in which to insert the table information. It is best to create a list presentation of the table so that it is easy for the user to choose a sales category. To insert a form template open the Toolbox. Go to:

▪ **View > Toolbox**

The following graphic menu will appear: (See Figure 4.22)

Below is a brief list and definition of the buttons:

<i>Selector</i>	Use this to select objects for moving, copying, cutting and pasting
<i>Label</i>	Use this to create a static text window
<i>Check box</i>	This tool can be linked to a two valued decision variable
<i>Command Box</i>	Multipurpose control button used to call other forms or reports
<i>Image</i>	Use this tool to import and display a single bitmap graphic (graphic size < 32KB)



**Figure 4.22** Toolbox for form objects

<i>Shape</i>	Use this tool to display shapes and lines
<i>File Card Tab</i>	Use this tool to insert a file card window
<i>Table Box</i>	Use this tool to insert a table record list window
<i>Lock</i>	This tool fixes a specific tool to the cursor function
<i>TextBox</i>	This tool links to table data and calculates an expression
<i>Option Button</i>	This tool can be used to select between possibilities of an option field
<i>Menu Button</i>	This tool allows you to create a list of command controls
<i>Frame</i>	This tool creates an empty frame
<i>Picture Box</i>	This tool displays a graphic from a list of user-imported graphics. Displayed graphic is determined by the <i>SourceExpression</i> value
<i>Indicator</i>	This tool displays a status bar to show the percent that a task is finished
<i>Subform</i>	This tool creates a form that can link a separate table to the main form table. It is particularly useful when one record in the main form table exists many times in another table. For example, the <i>Sales Line</i> is a Subform linked under the <i>Sales Header</i> in the <i>Sales Order</i>

To insert a table list into your form object you must select the Table Box tool and then move the cursor into the grid area of your otherwise empty form. Click into the grid space. Next, insert the fields from the table into the table list. Go to:

- **View > Field Menu**

A list will appear showing the two fields contained in the table. You can select the top left corner of the *Field Menu* list across from the *Field* column, thus selecting all the fields at once. Next, move your cursor into the center of the table list. Click into the center of the table list object; Microsoft Navision will insert the table fields into the object.

Now you will need a Command Button for the user to confirm their record selection and return to the table from which they came. Click on the Command Button tool in the Toolbox. Drag your cursor below the table list object and click the left button on your mouse. Next, open the properties of the Command Button. To do this, click on the Command Button and then go to:

- **View > Properties**

The following list will appear: (See Figure 4.23)

*Properties of the  
Command Button*

These are properties for the Command Button. Here you must define what you want to happen when the user clicks on this button. Go to the *PushAction* property and click on the *Value* column and then the assist arrow that appears to the right. Here a list of options will appear. Select *LookupOK* from the *PushAction* list. This tells Microsoft Navision to confirm the user's selection from the list and enter it into the linked table. Lastly, you must enter the name of the button the user will see. Go to the *Caption* property and write OK in the *Value* column.

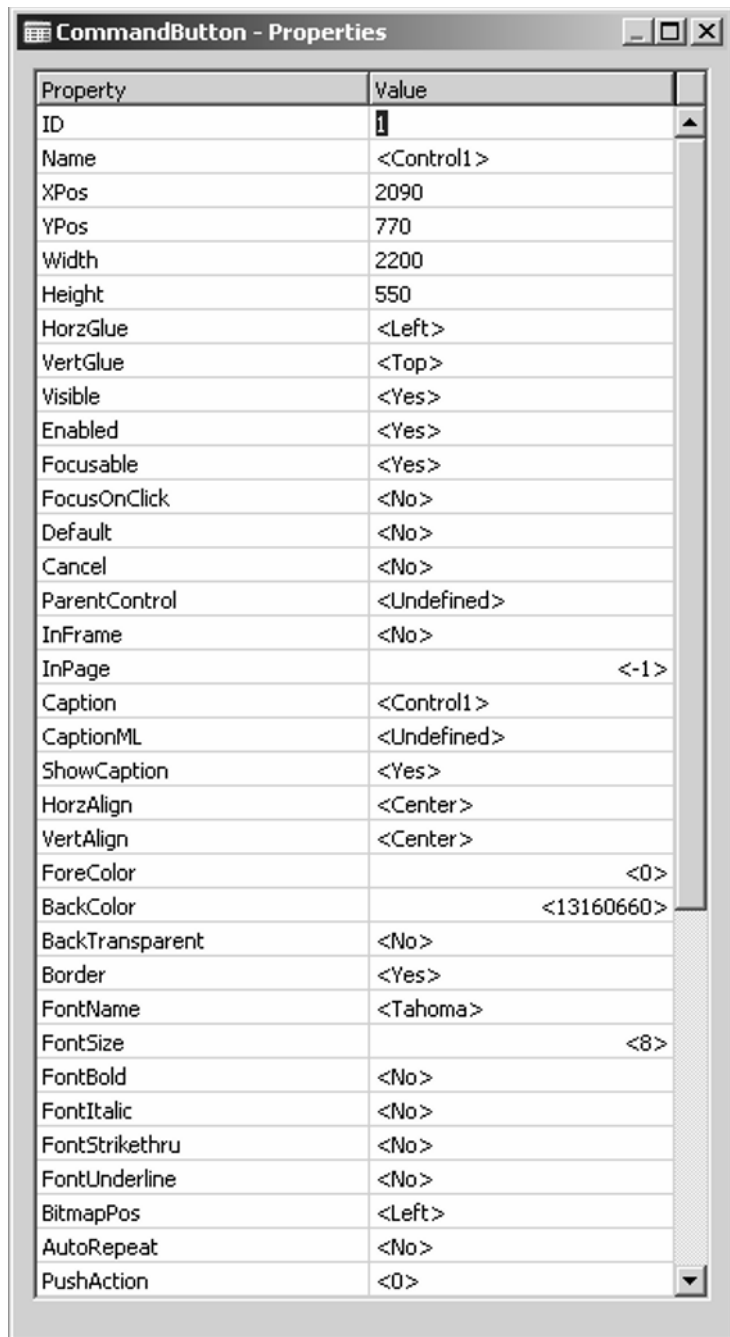
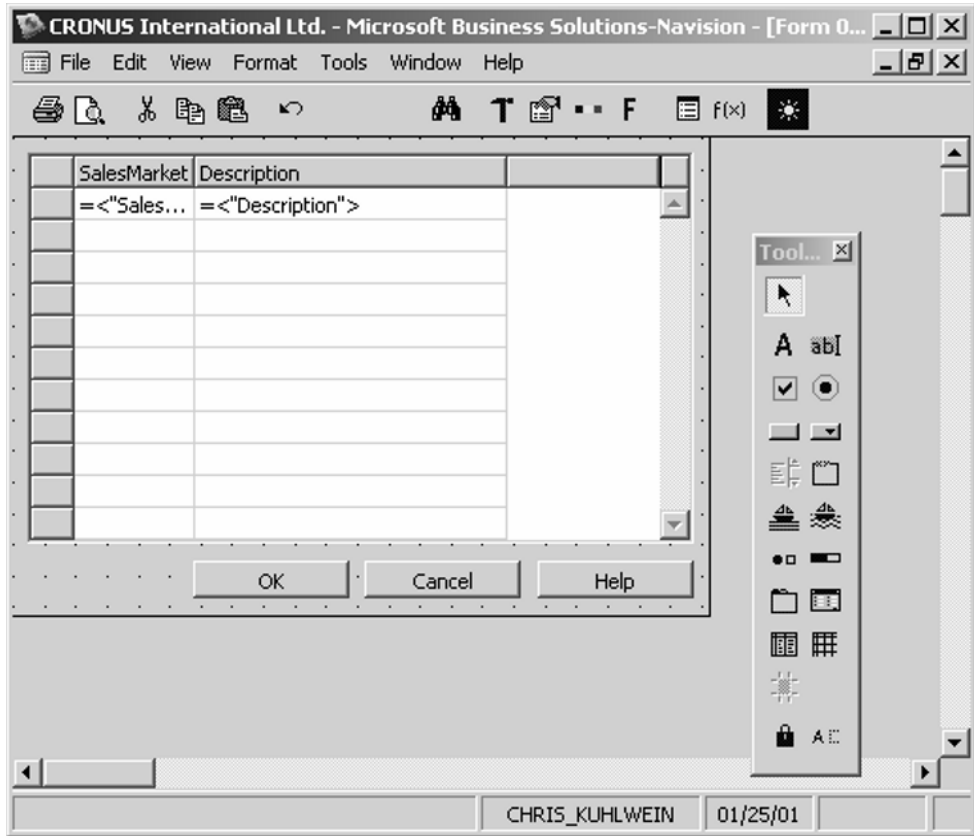


Figure 4.23 Command Button - Properties window

It is standard in Microsoft Navision to have a Help and Cancel button at the bottom of every form. Select the Command Button tool and insert two additional command buttons. Select from the Command Button property, *PushAction* option list, *LookupCancel*, and type *Cancel* into the *Caption* property. For the Help button, select *FormHelp* from *PushAction* property option list and type *Help* into the *Caption* property. The *Form Object* should look like the following:



**Figure 4.24** Form Object Designer window

Press ESC and save the form as *ID 50000* or as any available *ID* number and name the new form, “SalesMarketForm.”

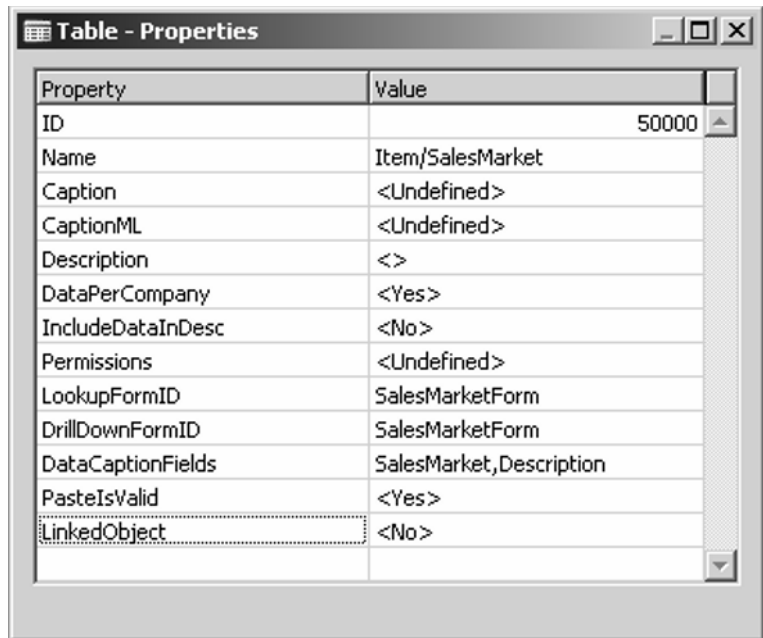
Now you have a form to refer to in the general table property, *LookupFormID*, in the *Item/SalesMarket* table. Go to:

- **Tools > Object Designer > Item/SalesMarket**

Select an empty line after the last field in the table and then go to:

- **View > Properties**

Next, fill in the *LookupFormID* with the value, *SalesMarketForm*. You can also enter this form name into the *DrillDownFormID* although it is not necessary at the moment. It is good practice to insert the name of the primary key and its description field into the *DataCaptionFields*. Doing so tells Microsoft Navision what information to display in the title of the record which is displayed in file card type forms.

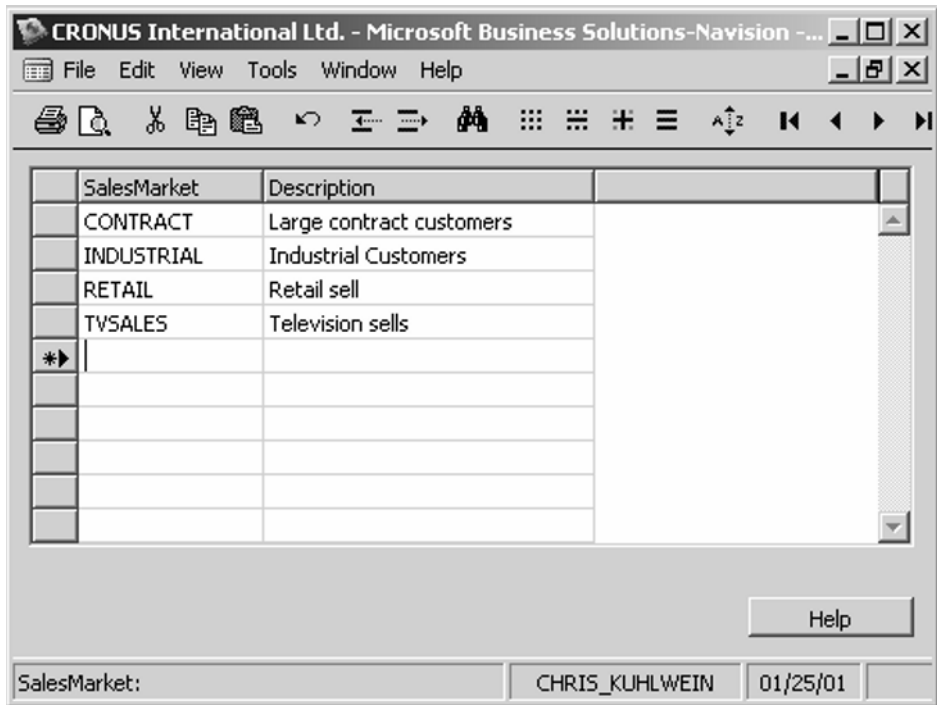


Property	Value
ID	50000
Name	Item/SalesMarket
Caption	<Undefined>
CaptionML	<Undefined>
Description	<>
DataPerCompany	<Yes>
IncludeDataInDesc	<No>
Permissions	<Undefined>
LookupFormID	SalesMarketForm
DrillDownFormID	SalesMarketForm
DataCaptionFields	SalesMarket,Description
PasteIsValid	<Yes>
LinkedObject	<No>

**Figure 4.25** *Table - Properties* window

Close and save the table by pressing the ESC key twice and answer “Yes” to the save dialogue box. Next, click Run at the bottom of the *Object Designer* window to open the new table for data entry. Enter the following information into your table:





**Figure 4.26** *Item/SalesMarket* table

*Connecting view table with the Item Card*

To make use of this table you must link it to the information in the *Item Card*. The *Item Card* must have a new field that contains this new variable for the items. To insert this new variable into the *Item Card* you must first create the new field in the table upon which the *Item Card* is based. You can call this new field, *SalesMarketcode*. After you create this variable it will be linked from the *Item Card* to the new table you just created—the *Item/SalesMarket* table.

Press ESC twice and return to the end user environment. Next, go to:

- **Warehouse > Planning & Execution > Item**

The *Item Card* should now be open. Next, you need to determine which table the *Item Card* is based upon. To do so, press CTRL+F2 to open the internal structure of the *Item Card*. Once the internal structure of the *Item Card* is open, click into the empty gray area surrounding the object and then go to:

- **View > Properties**

Find the *SourceTable* property. You will see here that the table upon which this form is built is, *Item*.

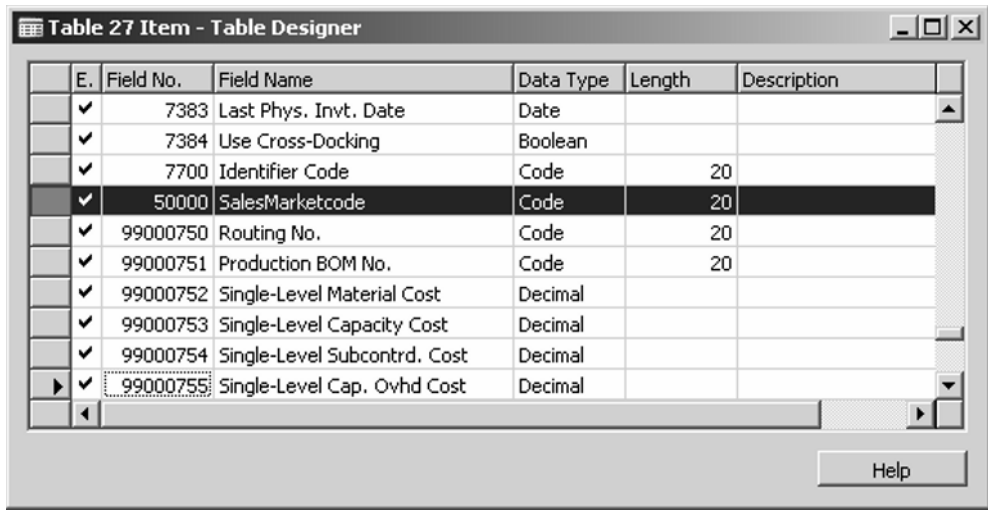
Now that you know what table underlies the *Item Card*, you must go into the *Object Designer* and find it. Locate the *Item* table and click on the Design button at the bottom of your window. Here you must find a free field *ID* number which you have the right to use. Normally, you can enter new fields in the range of 50000 to 50999.

Enter the name, *SalesMarketcode*, into the field name. You want Microsoft Navision to use in this field only that information which has been predetermined in the new *Item/SalesMarket* table. You must be careful to make the new field the same *Data Type* and *Length* as the field with which you want to link it; two fields that are linked must be the same type and length so that information from one can fill the other.

We must choose a field from the foreign table that we know is unique in the foreign table. This is the only way to avoid confusion regarding which record is being referred to in the foreign table. It has to be a field with an address that will never be confused with another record. The field you link to from the *Item Card* must therefore be the primary key within the foreign table. This means that the new field, *SalesMarketcode*, in the *Item* table must be linked to the primary key field of the *Item/SalesMarket* table which is called, *SalesMarket*.

The *Data Type* of *SalesMarket* must be the same as that of *SalesMarketcode* in the *Item* table. We can link two fields only if their *Data Types* are equal. Also the length of both linked fields must be the same—in this case 20.

Therefore, in the *Item* table you should type in the following highlighted line:



**Figure 4.27** Item - Table Designer window

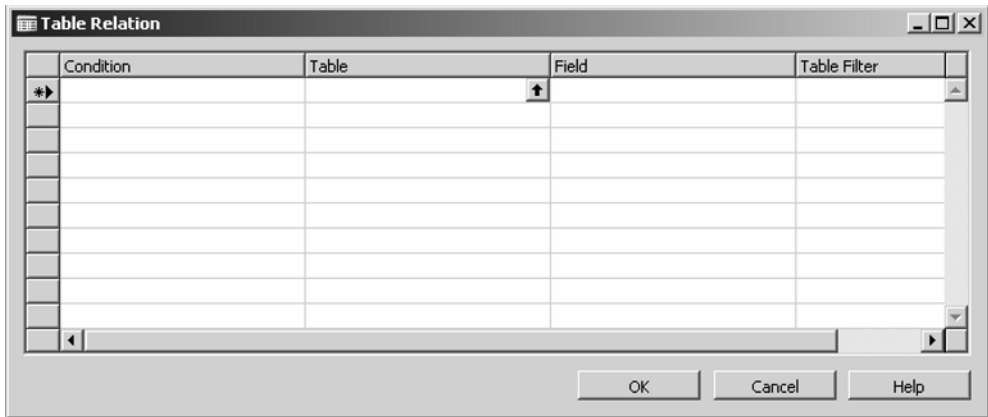
Now the table link must be established. After selecting the new field, *SalesMarketcode*, go to:

- **View > Properties**

Open the properties of this field and look for the important *TableRelation* property. Click on this property and you will see an assist button appear to the right of the row. Click on this assist. The following window will appear: (See Figure 5.28)

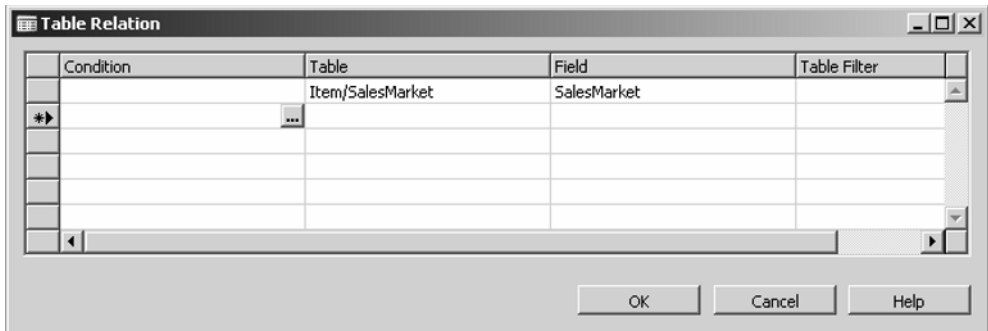
*Table relation*

In the *Table* column click on the assist arrow that appears in the top right of the *Table* column or write, *Item/SalesMarket*, into the *Table* column. When you enter a valid table name here, Microsoft Navision automatically assumes that you will be using the primary key of this table.



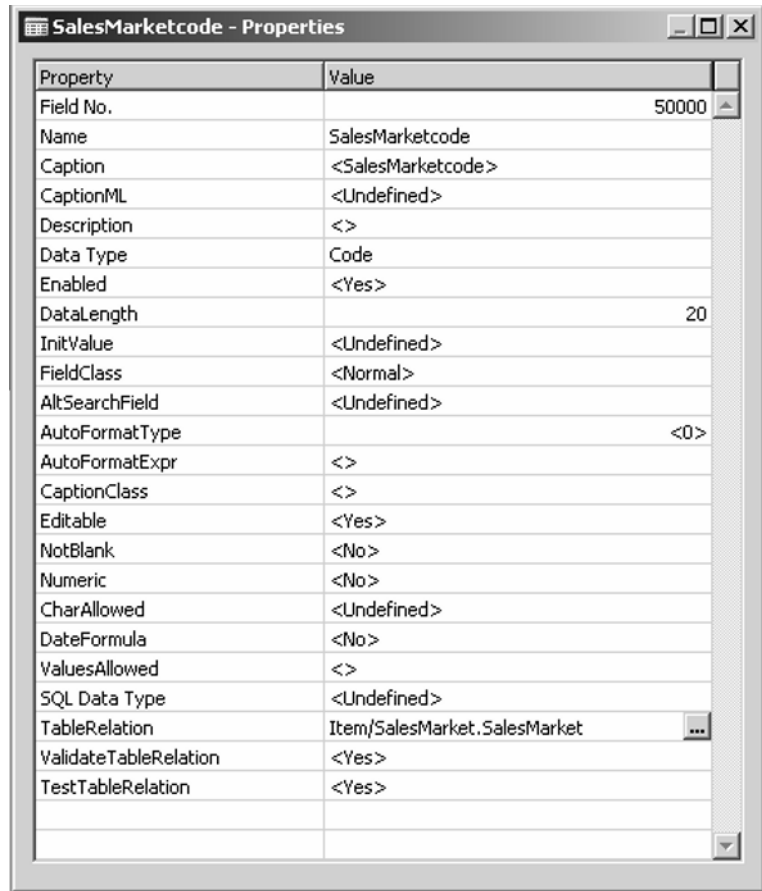
**Figure 4.28** *Table Relation* window

The *Field* column exists here even though Microsoft Navision automatically takes the primary key; it is often the case that the primary key of a table consists of more than one field. An example would be when a table has an order number and an order line which together define the address of each table record. Write *SalesMarket* into the *Field* column or click into the *Field* column and select the *SalesMarket* from the list that appears. Your *Table Relation* window should now look like the following:



**Figure 4.29** *Table Relation* window

Click OK at the bottom of the *Table Relation* window. The following property list will appear:



**Figure 4.30** SalesMarketcode - Properties window

As you can see, the *Data Type* is Code, the *Length* is 20 and the *TableRelation* is *Item/SalesMarket.SalesMarket*.

When Microsoft Navision refers to a table and one of its fields, it always states this in the form:

#### Table.Field

That is, the table name should be followed by a period (.) and then the name of the field. If there are any spaces or special symbols in the name of a table or field, Microsoft Navision will use quotation marks (“ ”) around the name to indicate that it is one name unit rather than two or more. In this instance, the table and field names contain no spaces, which is a good practice to follow. Microsoft Navision refers to the table and field link as:

## Item/SalesMarket.SalesMarket

Next, press ESC three times and answer “Yes” when Microsoft Navision asks if you want to save the table. The next step is to enter the new field into the *Item Card* now that you are finished creating it in the underlying *Item* table.

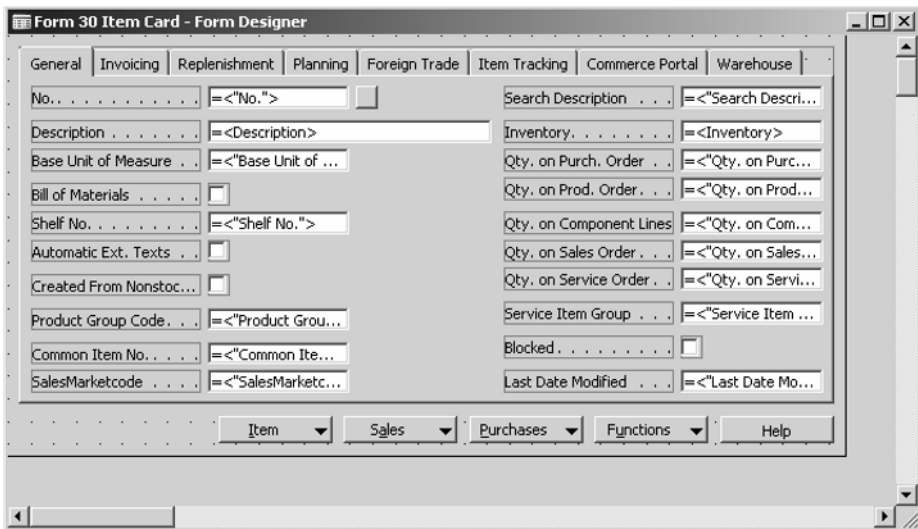
*Inserting a new field in the Item Card*

Go to:

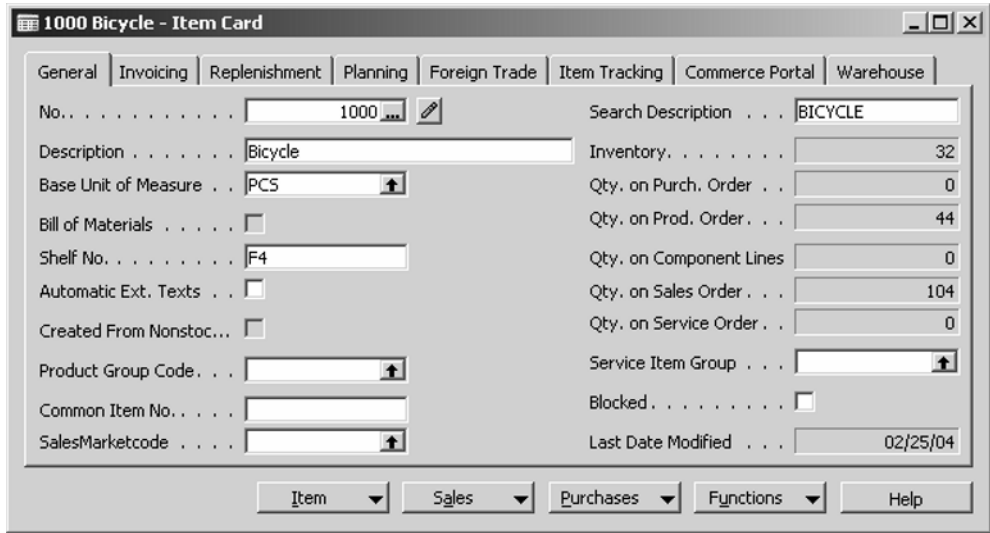
- **Warehouse > Planning & Execution > Item**

Press CTRL+F2 to open the internal structure of the *Item Card*. Open the *Field Menu* by clicking on the third icon tool from the right. This menu shows you all the potential fields in the underlying table. Click into the table and press the letter S on your keyboard until you come to our new field, *SalesMarketcode*. Highlight this field and then drag and drop it into the form object. Place it just below the *Common Item No.* field on the first tab *General* of the *Item Card*. You should now see the following window: (See Figure 4.29)

Press ESC twice and answer “Yes” when Microsoft Navision asks you if you want to save your changes. Next, press ESC again and then reenter the *Item Card*—you must always reload the object after you have made changes to it. The following window will appear: (See Figure 4.32)



**Figure 4.31** *Item Card - Form Designer* window



**Figure 4.32** Item Card with new field, SalesMarketcode

Click into the new field, SalesMarketcode. Now the following list will appear:



**Figure 4.33** New Item/SalesMarketing list

This is even the form that we created. Next, select *CONTRACT* and click the OK button at the bottom of the window; Microsoft Navision automatically returns you to the *Item Card*. Paste your selection into the *Item Card*. For testing purposes, suppose we enter into each sales item of the CRONUS Database the following *SalesMarketcode* categories:

No.	SalesMarketcode
1900-S	CONTRACT
1906-S	INDUSTRIAL
1908-S	RETAIL
1920-S	TVSALES
1924-W	CONTRACT
1928-S	INDUSTRIAL
1928-W	RETAIL
1936-S	TVSALES
1952-W	CONTRACT
1960-S	INDUSTRIAL
1964-S	RETAIL
1964-W	TVSALES
1968-S	CONTRACT
1968-W	INDUSTRIAL
1972-S	RETAIL
1972-W	TVSALES
1976-W	CONTRACT
1980-S	INDUSTRIAL
1984-W	RETAIL
1988-S	TVSALES
1988-W	CONTRACT
1992-W	INDUSTRIAL
1996-S	RETAIL
2000-S	TVSALES



Now we are ready to run the Microsoft Navision standard “Inventory - Top 10 List” report while filtering it on the new *Item* field, *SalesMarketcode*. Go to:

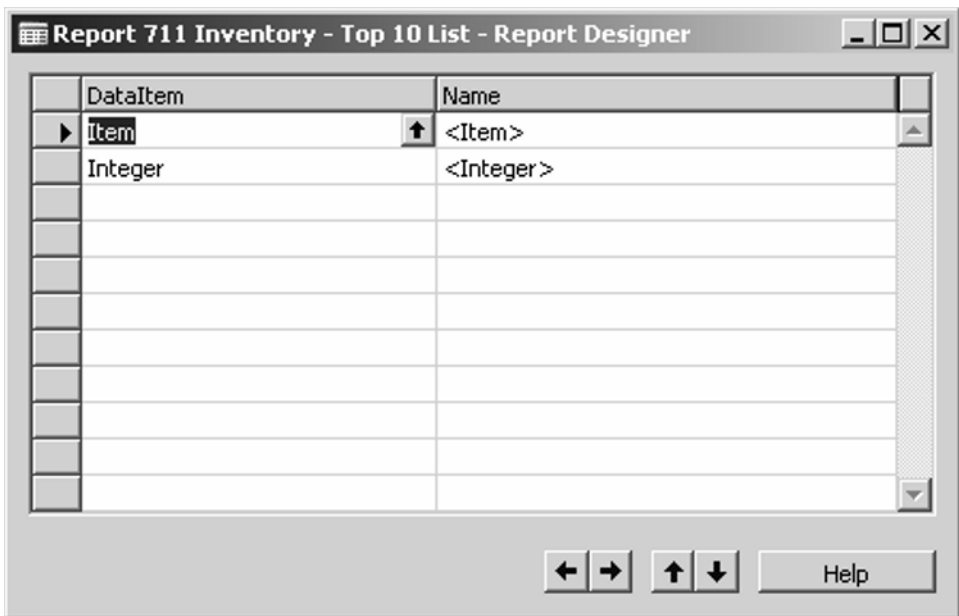
- **Sales & Marketing > Inventory & Pricing > Reports > Inventory - Top 10 List**

Click Print at the bottom of the screen.

#### 4.4.5 Adding New Standard Filtering Options to a Report

The “Inventory - Top 10 List” report is now open. It is nice if the new field automatically appears as a filter option like the other fields, such as *No.* or *Inventory Posting Group*. You can manually enter the new field, *SalesMarketcode*, to the list of filters by clicking into the empty row immediately following the last filter and then by clicking on the arrow that appears. Next, select the *SalesMarketcode* row from the list that appears. If this is done manually then it must be done every time this report is used. We will change the report so that the new filtering option appears automatically each time the report is opened.

Next, open the inner structure of the report by pressing CTRL+F2. The following window will appear:



**Figure 4.34** *Inventory - Top 10 List - Report Designer* window

*Automatic filter choices*

This is the first level of the inner structure of a Microsoft Navigation report. To change the automatic filtering selection in a report we must open the properties of the *DataItem* that contain the table where the desired field exists. As you remember, you created *SalesMarketcode* as a new field in the *Item* table. Therefore, click on the first row where the table *Item* is a *DataItem*. After clicking on the *Item DataItem*, go to:

- **View > Properties**

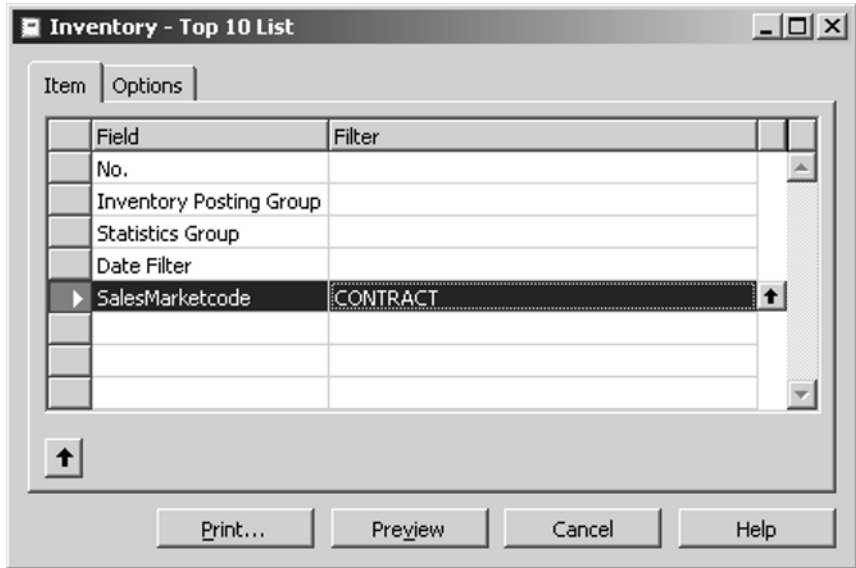
The following list of report *DataItems* will appear:

Property	Value
DataItemIndent	<0>
DataItemTable	Item
DataItemTableView	SORTING(No.)
DataItemLinkReference	<Undefined>
DataItemLink	<Undefined>
NewPagePerGroup	<No>
NewPagePerRecord	<No>
ReqFilterHeading	<>
ReqFilterHeadingML	<>
ReqFilterFields	No.,Inventory Posting Group,Statistics Group,Date Filter
TotalFields	<Undefined>
GroupTotalFields	<Undefined>
CalcFields	<Undefined>
MaxIteration	<0>
DataItemVarName	<Item>
PrintOnlyIfDetail	<No>

**Figure 4.35** *Item - Properties* window

Here we need to enter a field into the *ReqFilterFields* property. You can define default filter options here for *DataItems*. These filter options are seen by the user when they run the report. Type a comma (,) after the last field listed in *ReqFilterFields* and then type, *SalesMarketcode*. You can also click on the *Value* column of the *ReqFilterFields* line and then on the assist box that appears at the right of the column. This opens a list of all the fields in the *Item* table, including the *SalesMarketcode* field. After you have added to this *ReqFilterFields* list, press ESC twice and

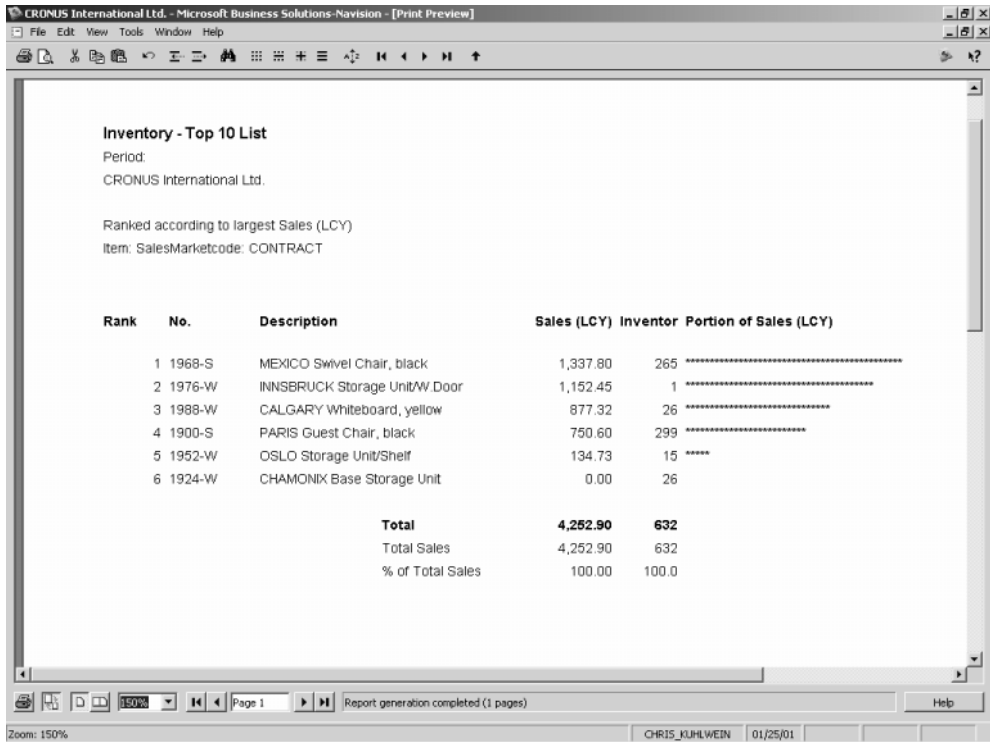
answer “Yes” to Microsoft Navision’s request to save changes. Next, press ESC again to exit the report and then select the report again from the *Item Reports - Print* list to reopen the newer version of the same report. The following window will appear:



**Figure 4.36** *Inventory - Top 10 List* with new filtering option automatically given

As you can see, the new filtering key has automatically appeared. Next, click into the *Filter* column in the *SalesMarketcode* row and then click on the assist arrow that appears at the right of the *Filter* column.

The same form you created earlier—the *SalesMarketForm*—will appear. Select the first *SalesMarket* in the list, *CONTRACT*, and click OK. Look at the *Options* file card tab of the report to see if there are any other important conditions that must be set. In this case, leave the *Options* window as it is and click on the *Preview* button at the page bottom. The following report output will appear:



**Figure 4.37** "Inventory - Top 10 List" report filtered by *SalesMarketcode*, *CONTRACT*

Compare this list with the former list to see which items in the *Item Card* have the *SalesMarketcode* equal to *CONTRACT*.

No.	SalesMarketcode
1900-S	CONTRACT
1924-W	CONTRACT
1952-W	CONTRACT
1968-S	CONTRACT
1976-W	CONTRACT
1988-W	CONTRACT

As you can see, the “Inventory - Top 10 List” report provides a clean ranking analysis of only those items that are defined to be sold in the *SalesMarket, Contract*.

Now, every week, at the push of a button, you can tell the boss exactly which items are performing the best and in which sales distribution areas. This information is instantaneously available, correct and always actual. The changes we have made in Microsoft Navision were not difficult to accomplish and are of such a quality that they cannot be distinguished from those existing in the standard Microsoft Navision application.

Although it may take a beginning developer some time to go through all the required steps, with practice such Microsoft Navision optimization tasks can be accomplished in less than five or ten minutes. The time saved and overall informational benefits save hundreds of hours in a year. This is the power of a flexible ERP system with which you can truly optimize your firm.

**Note:**

***Once again, it is highly recommend by the authors that you first create such development solution in a test copy of your Microsoft Navision database, especially if you are making a change at the table level. After thoroughly testing your new solution, cut the updated or new objects out of your test database and paste them into the live Microsoft Navision system. This cut and paste operation can be done by highlighting the new or updated objects and then using the Edit > Copy and Paste tools. The more recent your test copy database is, the better. Please refer to the earlier section, “Creating a Test/Learning Microsoft Navision Database,” for information on test databases.***

# 5

## Creating New Flow Fields

---

A step-by-step presentation of how to create new *Flow Fields* will be given in this chapter. We will base our discussion on the concepts and procedures developed in previous chapters and use object like tables and forms in our examples.

### 5.1 Connecting a Variable With Its History

As you have seen when looking at *Flow Field* variables like *Inventory* and *Net Change*, a *Flow Field* provides a real time, calculated value. This calculated value is connected directly to the details that Navision uses to arrive at the value. The true merit of *Flow Fields* is that they give you a flexible overview without sacrificing the everyday details.

*Flow Fields* accomplish this work according to the basic principles of relational data organization. The *Flow Field* finds every line in a foreign table by selecting a match to its primary key within the lines of the foreign table and then sums these lines. Therefore, to create a *Flow Field*, you must know that the primary key for the *Flow Field* table exists in the foreign table where you want to sum the lines. Navision uses a special indexing of the foreign table to speed the summation operation.

A good way to think of the structure of a *Flow Field* is that it connects a variable with the history of its movements.

### 5.2 Connecting Salespeople To Their Sales

Again we will return to our discussion of the relational table system as it relates to the current theme—connecting salespeople to their sales.

#### 5.2.1 Clearly and Operationally Defining ‘Sales’

Let us use a real world example to create a *Flow Field* that provides a view of the performance of your firm’s sales force. Suppose your boss comes to you with a question about the performance of the sales staff. She wants to be able to see, at the

click of a button, their sales within a limited sales period and look at what customers have purchased.

As always, you must begin with an understanding of your firm, its definitions and needs. What is the best definition of *sales* in your firm? This is neither an easy or obvious question.

If your firm pays commission on sales, it is a question you must get right. Are sales defined as the total number of customer orders entered into Navision, or are sales the total of everything that your company has shipped to the customer? Should you wait until a customer has paid an invoice to include the invoice amount in the salesperson's sales? What if something is entered into an order but is later returned? Must you remove this from a salesperson's sales? These are questions you must answer based on your knowledge of your firm and how it operates. If you are unclear about such definitions, Microsoft Navision will not be able to help.

For this example, we will choose a safe definition of sales. That is, we will define sales as the sum of customer orders that have been shipped and invoiced, minus anything that is credited to the customer due to a return. Now we will need only track two Navision documents to measure our sales: customer invoices and credit memos. We will not need to worry about the shipping of products as the standard Navision system does not allow an invoice to be created for products without finished shipping documents. Therefore, if your shipping department sends everything for which it posts a shipping document then you will not need to consider shipping.

What you will need to do to satisfy the boss is to create a sales *Flow Field* that includes the logic of this sales definition and sums customer sales sorted by the salesperson responsible for each order.

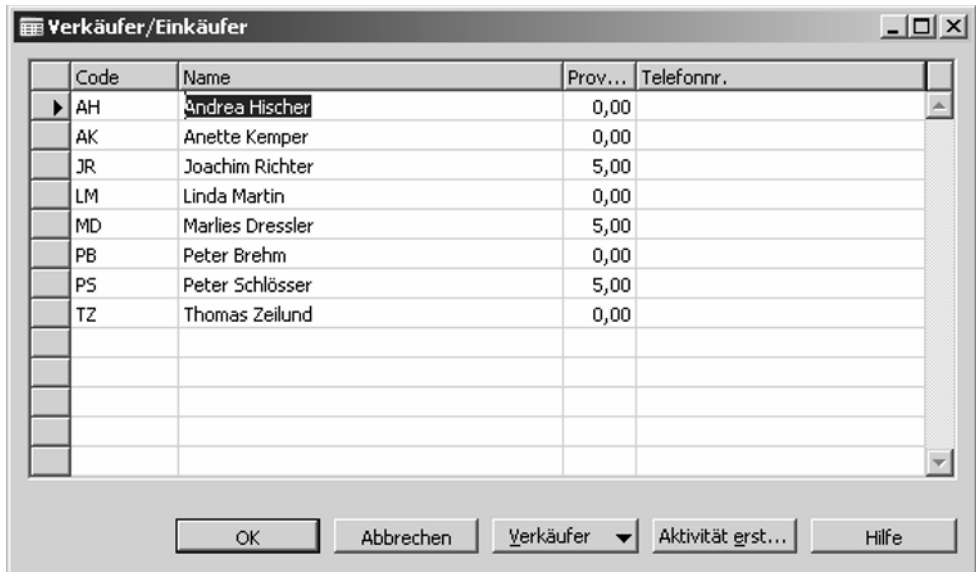
### **5.2.2 The New Sales Flow Field: Searching For the Correct Fields**

We must now locate the salespeople and sales information and determine how to link them. You can begin looking in Navision for a list of the salespersons in the *Sales & Marketing* menu. Go to:

- **Sales & Marketing > Sales > Salespeople**

Imagine that your boss would like to compare the salespeople. Therefore, a table list view would be a better presentation than

the file card form display. Press F5 to open the *Salesperson/Purchaser* List view.



The screenshot shows a window titled 'Verkäufer/Einkäufer' with a table of sales staff. The table has four columns: Code, Name, Prov..., and Telefonnr. The data is as follows:

Code	Name	Prov...	Telefonnr.
AH	Andrea Hischer	0,00	
AK	Anette Kemper	0,00	
JR	Joachim Richter	5,00	
LM	Linda Martin	0,00	
MD	Marlies Dressler	5,00	
PB	Peter Brehm	0,00	
PS	Peter Schlösser	5,00	
TZ	Thomas Zeilund	0,00	

At the bottom of the window, there are buttons for 'OK', 'Abbrechen', a dropdown menu currently showing 'Verkäufer', 'Aktivität erst...', and 'Hilfe'.

**Figure 5.1** *Salesperson/Purchaser* list view

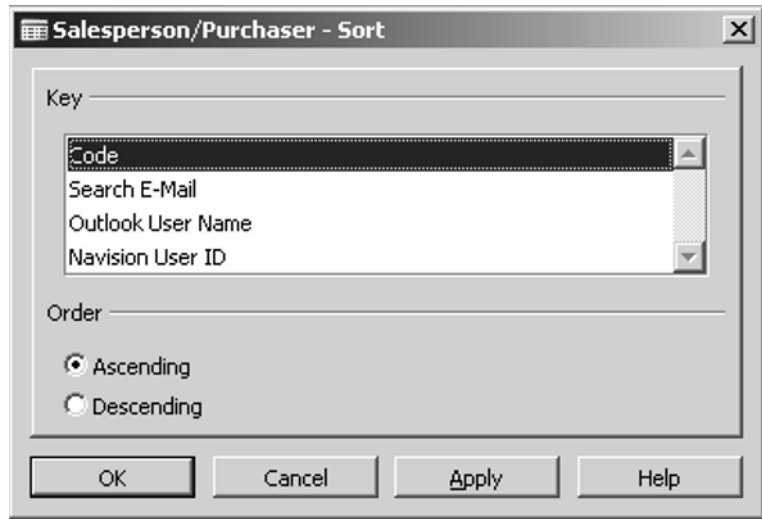
Here is a complete list of the company's sales staff. This is a stock information table because its primary information will not change, such as the names of the salespersons. It does not contain information concerning individual transactions. Here is an ideal place for a column with a sum of each salesperson's sales. These total sales can also be connected to individual customer invoice transactions.

Now that we know where our stock information is, we need to answer three questions about the source of this stock information that will help us later in making our table connections:

- What variables make up the primary key of the stock information table?
- What are the field names of the variables that make up the primary key?
- What is the table name upon which this form is built?

We can answer the first question by using the Sorting tool. Click on the icon tool which is located sixth from the left at the top of your screen. The following window will appear:





**Figure 5.2** *Salesperson/Purchaser - Sort* options window

The first entry in the sorting option list is always the primary key of the underlying table. The primary key here is *Code* and it contains the code of the salesperson.

Now you must find the name of table upon which this form is built. This is necessary because the *Flow Field* you wish to create must be inserted into the table upon which this form is built. To find the table name, press CTRL+F2. Next, click into the empty gray area surrounding the form object and go to:

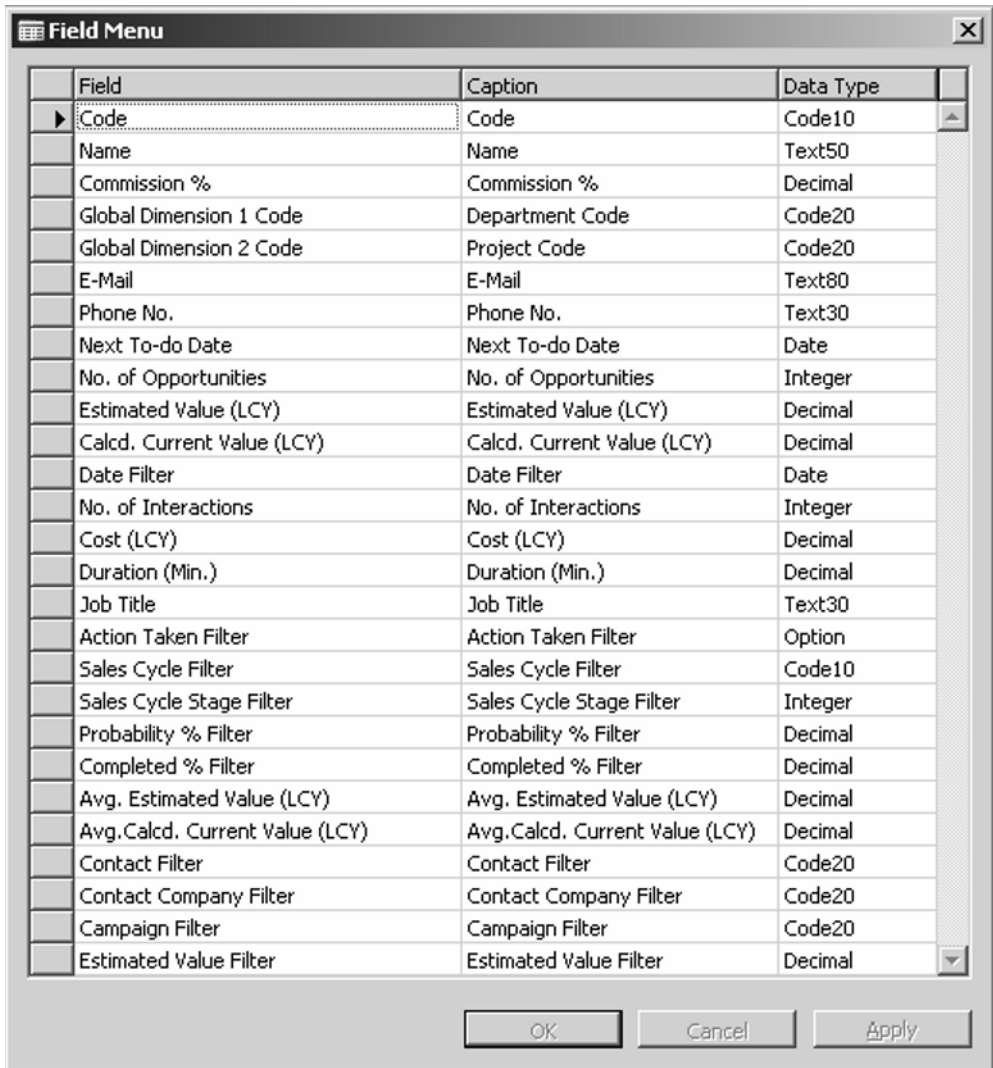
- **View > Properties**

All of the properties of the form will open. Look down through the property list for the *SourceTable* property. The value in this property is, *Salesperson/Purchaser*, which is the name of the table underlying our *Salesperson/Purchaser* List view.

Next, check the name of the variable *Code* in the table itself by pressing ESC once to leave the form property list. Next, go to:

- **View > Field Menu**

The following window will appear:



**Figure 5.3** *Field Menu* window for *Salesperson/Purchaser*

Here you can see that the *Caption* in the form is the same as the variable *Field* name, *Code*.

Now that you have the necessary information concerning the stock information table, you must find the table where the sales and customer transactions are located. This table must contain the primary key of the stock table, including the salesperson

*Code* and variables where the sales invoice amounts and customers are recorded.

You can look for such a table starting at the *Customer Card*. Go to:

- **Sales & Marketing > Sales > Customers**

Next, open the posting list that contains the history of the customer transactions. Go to the button at the bottom of the *Customer Card* and the click on *Ledger Entries*. The following window will appear:

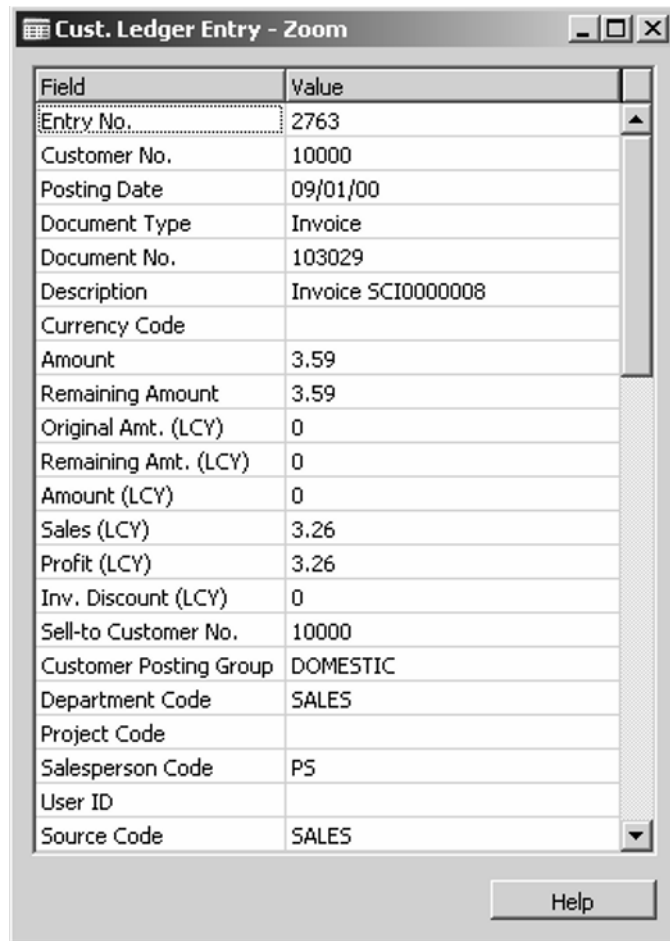
Document Type	Docume...	Customer No.	Description	Original Amount	Amount	Remaining A...
00 Invoice	103022	10000	Invoice SC10000001	0.12	0.12	0.12
00 Invoice	103023	10000	Invoice SC10000002	0.11	0.11	0.11
00 Invoice	103027	10000	Invoice SC10000006	3.59	3.59	3.59
00 Invoice	103034	10000	Invoice SC10000013	3.37	3.37	3.37
00 Invoice	103028	10000	Invoice SC10000007	3.59	3.59	3.59
00 Invoice	103035	10000	Invoice SC10000014	3.37	3.37	3.37
00 Invoice	103029	10000	Invoice SC10000008	3.59	3.59	3.59
00 Invoice	103036	10000	Invoice SC10000015	3.37	3.37	3.37
00 Invoice	103030	10000	Invoice SC10000009	3.59	3.59	3.59
00 Invoice	103037	10000	Invoice SC10000016	3.37	3.37	3.37
00 Invoice	103031	10000	Invoice SC10000010	3.59	3.59	3.59
00 Invoice	103038	10000	Invoice SC10000017	3.37	3.37	3.37
00 Invoice	103032	10000	Invoice SC10000011	3.59	3.59	3.59
00 Invoice	103039	10000	Invoice SC10000018	3.37	3.37	3.37
00 Invoice	00-1	10000	Opening Entries, Customers	25,389.25	25,389.25	0.00
00 Invoice	00-11	10000	Opening Entries, Customers	63,473.13	63,473.13	63,473.13
00 Invoice	00-16	10000	Opening Entries, Customers	33,852.35	33,852.35	33,852.35
00 Invoice	00-3	10000	Opening Entries, Customers	50,778.50	50,778.50	0.00
00 Invoice	00-6	10000	Opening Entries, Customers	67,704.67	67,704.67	0.00

**Figure 5.4** *Customer Ledger Entries* list view

Here you can see the customer code in the field, *Customer No.*, the amount of each transaction in the *Amount* field and the type of the transaction in the *Document Type* field. The only necessary variable that is missing here is a variable containing the code of the salesperson—the variable by which the sales will be grouped. It is also the primary key of the stock information table and therefore, absolutely necessary. Look at every variable in the table behind this *Customer Ledger Entries* list view. Go to:

- **Extras > Zoom**

The following list will appear:



Field	Value
Entry No.	2763
Customer No.	10000
Posting Date	09/01/00
Document Type	Invoice
Document No.	103029
Description	Invoice SCI0000008
Currency Code	
Amount	3.59
Remaining Amount	3.59
Original Amt. (LCY)	0
Remaining Amt. (LCY)	0
Amount (LCY)	0
Sales (LCY)	3.26
Profit (LCY)	3.26
Inv. Discount (LCY)	0
Sell-to Customer No.	10000
Customer Posting Group	DOMESTIC
Department Code	SALES
Project Code	
Salesperson Code	PS
User ID	
Source Code	SALES

**Figure 5.5** Customer Ledger Entries - Zoom view

Here you see the variable, *Salesperson Code*, which contains the salesperson ID. With this variable you can link the two tables. You also see the field, *Sales (LCY)*, which shows the sales amount without the sales tax included. This is a perfect variable to sum in the sales *Flow Field*.

Next, find the name of the table that the form, *Customer Ledger Entries*, is built on. You must also find the field names of the important variables within the table. Press ESC once and then CTRL+F2. Next, select the empty gray area outside the form object then go to:

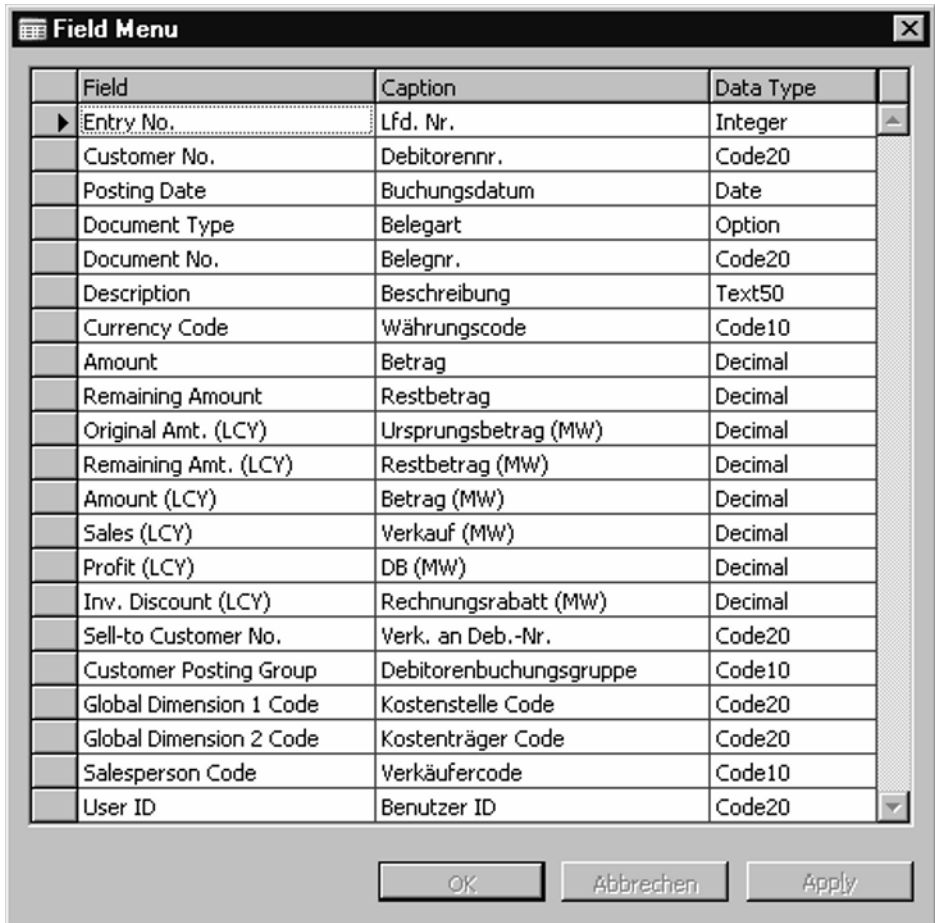
- **View > Properties**

Look for the *SourceTable* property. You will see that the *Customer Ledger Entries* form is built on the *Cust. Ledger Entry* table.

Now look for the names of the important variables as they exist in the table. Leave the form property list by pressing ESC. Go to:

- **View > Field Menu**

The following window will open:



Field	Caption	Data Type
▶ Entry No.	Lfd. Nr.	Integer
Customer No.	Debitorennr.	Code20
Posting Date	Buchungsdatum	Date
Document Type	Belegart	Option
Document No.	Belegnr.	Code20
Description	Beschreibung	Text50
Currency Code	Währungscode	Code10
Amount	Betrag	Decimal
Remaining Amount	Restbetrag	Decimal
Original Amt. (LCY)	Ursprungsbetrag (MW)	Decimal
Remaining Amt. (LCY)	Restbetrag (MW)	Decimal
Amount (LCY)	Betrag (MW)	Decimal
Sales (LCY)	Verkauf (MW)	Decimal
Profit (LCY)	DB (MW)	Decimal
Inv. Discount (LCY)	Rechnungsrabatt (MW)	Decimal
Sell-to Customer No.	Verk. an Deb.-Nr.	Code20
Customer Posting Group	Debitorenbuchungsgruppe	Code10
Global Dimension 1 Code	Kostenstelle Code	Code20
Global Dimension 2 Code	Kostenträger Code	Code20
Salesperson Code	Verkäufercode	Code10
User ID	Benutzer ID	Code20

Buttons: OK, Abbrechen, Apply

**Figure 5.6** *Customer Ledger Entries Field Menu*

Within the table the variables have the field names: *Customer No.*, *Document Type*, *Sales (LCY)* and *Salesperson Code*.

The variable *Document Type* is a special *Data Type* called, *Option*. *Option* data types contain a fixed list of choices for the user to choose from. The *Option* list is not found in a separate table, like *Sales Market* was in our previous example, but rather is contained in the definition of the field within the table of the field. There are a few special field properties for this type of variable. The developer can define within an *Option* field type a fixed set of options that the end user will be able to choose from.

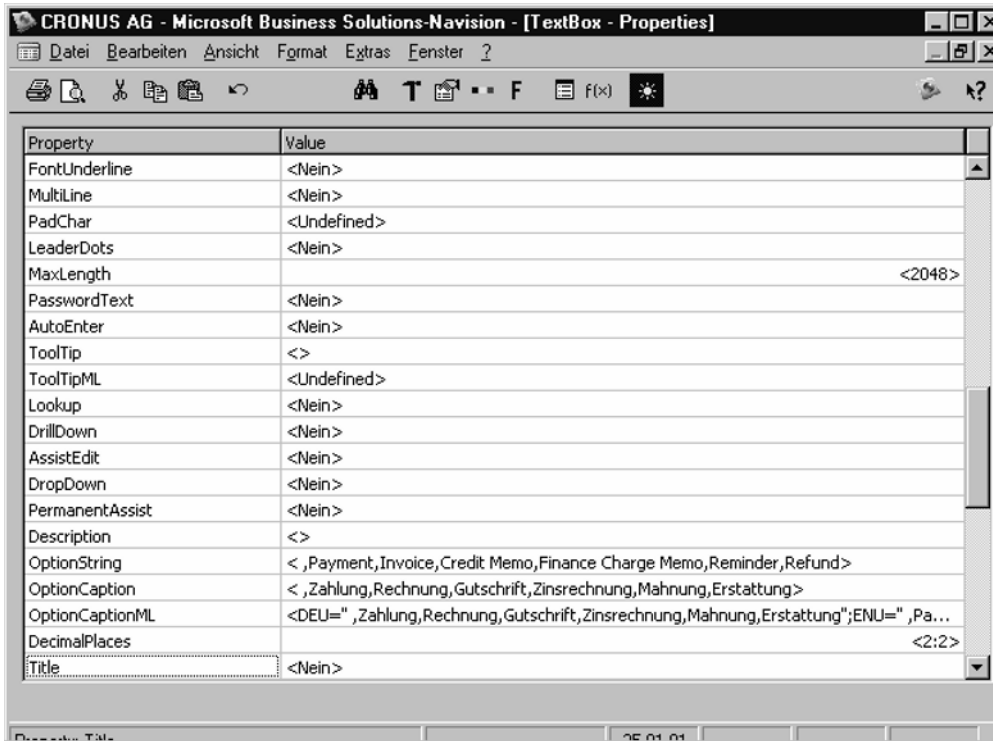
Because of Navision's language flexibility, you can define two levels of names in the *Option* list: one in a standard language and one that will be used in the presentation of the data in forms, reports and list view. To work with this *Option* variable in the development environment, you must find the standard *Option* list. This list is normally written in English because this is the standard language within the development environment. The standard list exists within the property list of the *Option* field. Once you find the standard list of options for this field then you can control this field within the development environment.

Press ESC to leave the form, *Field Menu*. Next, find the *Document Type* column—the second column from the left in the *Customer Ledger Entries* form. Next, click on the *Document Type* column and go to:

- **View > Properties**

Scroll down to the *OptionCaptionML* property. The following window will appear: (See Figure 5.7)

There are three properties that define the list of options for the *Document Type* field. The first is *OptionString* and it is the fundamental list that must be used when referring to an *Option* in the development environment. The next two fields determine how Navision will present this *Option* list to the end user. In the property, *OptionCaptionML*, you can describe the *Option* list with a reference to the language that your company is operating in. For example, the text GER= indicates that the standard language of the users is German. Please note the *Option* list as it appears in the property *OptionString* and note which of the German options in the *CaptionString* these correspond to.



**Figure 5.7** Option Language possibilities

In the present example you must instruct Navision to consider only *Customer Ledger Entries* transactions that correspond to invoices and credit memos. In the standard *Option* list language these are the options *Invoice* and *Credit Memo*.

### 5.2.3

#### Visualizing the Table Relationships Behind the New Sales Flow Field

Now let us construct a quick diagram of the tables, fields and table relations that are needed to create the sales *Flow Field*. We will include some values only for illustration purposes in the diagram and use the names of the variables, tables and options as they appear within the standard Navision development environment. Assign the name, *Total Sales*, to the new *Flow Field* that you will create within the *Salesperson/Purchaser* table. Consider the following diagram:

"Total Sales" Flow Field Diagramm

#### Stock Information

Table name = "Salesperson/Purchaser"			
Primary Key			
Code	Name	Total Sales	...
HK	Hans-Joachim Klee	11,000.00	...
JK	Jill L. Keehner	7,000.00	...
RS	Reiner Schulz	12,515.00	...

#### Transaction Information

Table name = "Cust. Ledger Entry"			
Customer No.	Document Type	Secondary Key	
		Salesperson Code	Sales (LCY)
D94773	Invoice	HK	1,000.00
D54763	Payment	JK	215.00
D97652	Invoice	HK	7,000.00
D85633	Invoice	HK	3,000.00
D10203	Invoice	JK	5,000.00
D63749	Credit Memo	RS	485.00
D53429	Refund	HK	985.00
D56463	Invoice	JK	2,000.00
D84535	Credit Memo	JK	3,000.00
D85732	Invoice	RS	3,000.00
D74653	Invoice	RS	10,000.00

**Figure 5.8** Total Sales Flow Field diagram

### 5.2.4 Inserting the Total Sales Flow Field in the Salesperson/Purchaser table

Now let us insert the new *Flow Field Sales Total* into the *Salesperson/Purchaser* table. Go to:

- **Extras > Object Designer**

Next, click on the Table button at the right and look through the list for the table object, *Salesperson/Purchaser*. When you find and select this table, click on the Design button at the bottom of the window. Next, go to the end of the field list and insert a new field. The new field must be given an ID number that is unique and acceptable according to the parameters of your Navision license. Use the ID number 50000 in the present example and name the field, *Sales Total*.

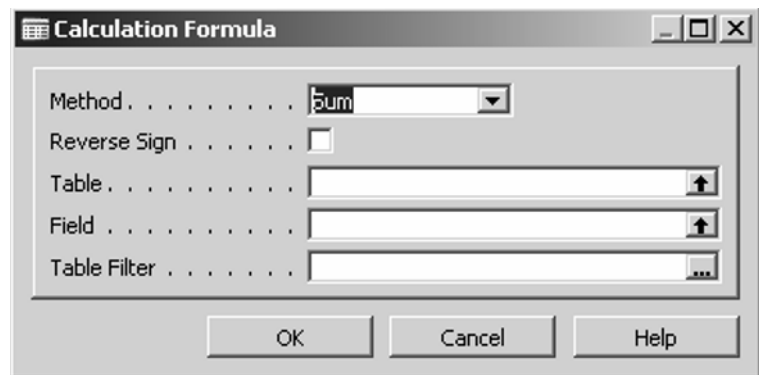


While this new field will contain a sum of values from the *Sales (LCY)* field in the *Customer Ledger Entry* table, it must be of the same *Data Type* and *Length* as the *Sales (LCY)* field. As seen in the *Field Menu* for the form *Customer Ledger Entries*, the *Field Type* of the *Sales (LCY)* variable is *Decimal*. This *Data Type* has no user-determined length so you can leave this column empty.

Now open the property list for the new variable. Go to:

- **View > Properties**

Next, you must tell Navision that this variable is of the special *Flow Field* type. Change the property, *FieldClass*, from *Normal* to *Flow Field*. When you change this *FieldClass*, a change in the variable property list occurs and a new property, *CalcFormula*, appears. This is the property where you will define the formula that Navision will use when it calculates the value for the *Flow Field*. Next, click on the *CalcFormula* property and then on the assist that appears in the right side of the column. The following window will appear:



**Figure 5.9** *Flow Field Calculation Formula* window

In the *Calculation Formula* window you can construct the process that Navision will use to fill the *Flow Field*. The first variable, *Method*, is used to determine what type of mathematical or logical operation Navision should apply in its calculation. The *Method* type *Sum* is by far the most frequently used type of *Flow Field* operation and it is the operation needed to add all the sales amounts together for the *Sales Total* variable.

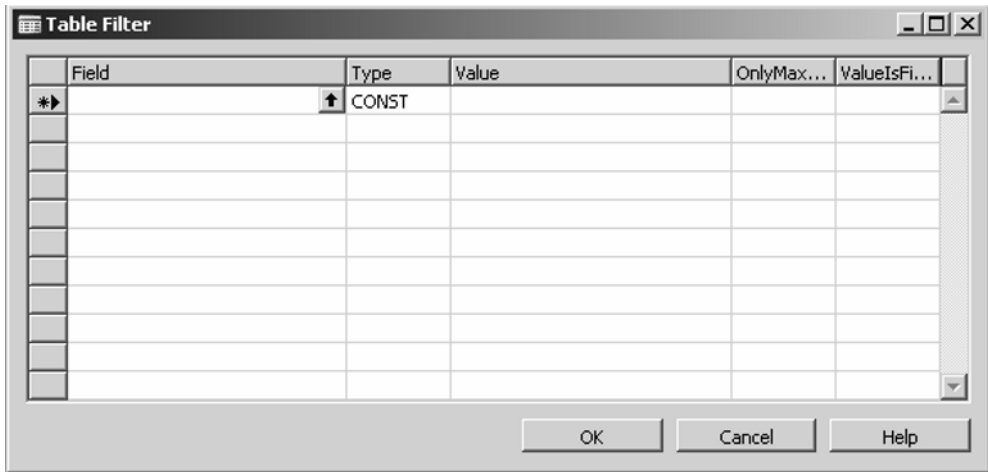
The *Reverse Sign* variable is useful when you want Navision to change the sign of the *Flow Field* calculation.

In the variable, *Table*, you must enter the table name you want connected to the *Flow Field*. This is the table that contains the

flow information to be summed. In this example, connect to the table, *Customer Ledger Entry*, where the sales transactions are located. Type in this name or click on the assist to select from the table list that appears.

Now that you have selected the table, enter the variable from the connecting table that you want Navision to sum. In this example, write in or select from the field, *Sales (LCY)*.

Now you must tell Navision which records within the table are to be summed and which are to be ignored. To do this you need to define a *TableFilter* that Navision will automatically set on the connecting table. You must set a filter so that Navision only opens records in the linked table that match the salesperson of the *Flow Field* table. You must also set a filter so that only *Invoices* and *Credit Memos* are summed. Lastly, you must set a filter so the user can control the date of the records to be included in the sales sum. Click on the assist in the variable, *TableFilter*. The following window will appear:



**Figure 5.10** *Flow Field* filtering window

This is where the important details of the *Flow Field* will be defined. In the leftmost column, enter the fields you wish to set filters on. In the middle column, define what type of filter you wish to set and in the third column, enter the filter itself.

Normally, when setting a filter on a table you may only enter static text which Navision compares against every record in the table. However, here in the *Flow Field TableFilter* system you

have the ability not only to enter one static text but also a variable filter—a field as the filter condition. To do this you must select *Field* in the *Type* column. The *Type* option, *Filter*, is for entering a normal static text filter. The *Type*, *Constant*, is for entering the option from the option list of a special *Option* data type variable, such as the variable with the *Caption*, *Document Type*.

To link these tables you must be sure that they have at least one variable in common. This common variable must be the primary key of the *Flow Field* table and a foreign key in the linked table. You must create a filter that allows Navision to find records in the foreign table that match the value in the primary key of the *Flow Field* table. Enter this filter first as it is the most important.

Next, either type *Salesperson Code* into the leftmost column or click on the column and then on the assist that appears and select *Salesperson Code* from the field list. Now select the *Option Field*, in the second column, *Type*. Lastly, type *Code* into the *Value* column or select it from the field list that appears if you click on the arrow in the *Value* column.

Next, you must tell Navision that you want only *Invoices* and *Credit Memos* included in the *Flow Field* sum. This information is contained in the variable with the *Caption* name, *Document Type* which corresponds to the *Field* name, *Document Type*. You must use the *Field* name for the variable in the development environment, therefore, select *Document Type* from the *Field* list in the leftmost column or simply type it into the column. Next, select the *Filter*, *Type* from the second column. Now, enter the text, *Invoice|Credit Memo*, into the *Value* column. Navision now has access only to the records within the *Customer Ledger Entries* where the *Document Type* equals either *Invoice* or *Credit Memo*.

You will need one last filter to control the sales period. You should not enter a static text filter here. Rather, you will want the option of changing this time period from the end user *Flow Filter* window. To accomplish this you will need another special *Flow Field* filtering function.

In addition to the existence of *Flow Fields*, Navision has a special field type, *Flow Filter*. We mentioned this type of field earlier in the book when we discussed the operation of the field, *Inventory*. Just as we used a *Flow Filter* in the end user environment to determine the way Navision calculated the *Inventory*, so we will insert a *Flow Filter* into the sales *Flow Field*. As a result, the end user will be able to control the *Sales Total* calculation using the same method shown to control *Inventory*.

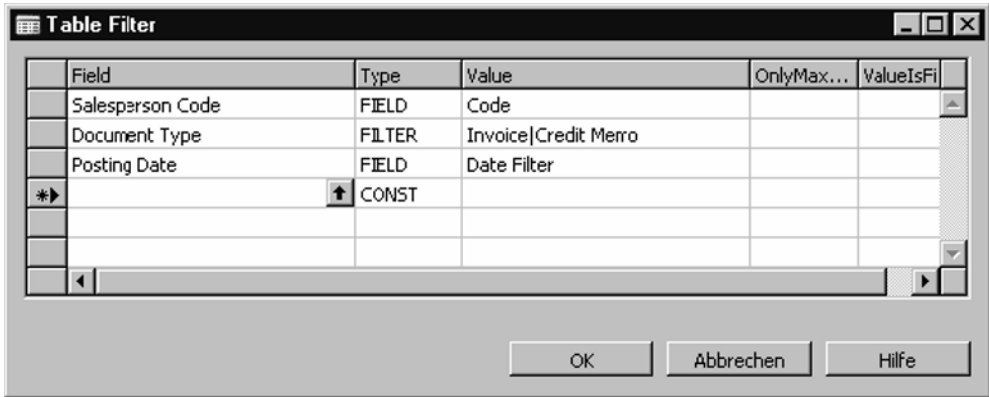
Next, we will make this new *Sales Total Flow Field* sensitive to the date of the transactions it sums. From the end user environment you have already seen that you can control the time period of summed transactions with the *Flow Filter* that has the *Caption, Date Filter*. An example would be if you were in the *Chart of Accounts* and wanted to see the sales totals from a previous year in the *Net Change* column. To accomplish this, set the *Date Filter, 01/01/2001..12/31/2001*, into the *Flow Filter Date Filter* located in the *Flow Filter* window. After setting this *Flow Filter*, the variable, *Net Change*, shows you the sum of transactions occurring since January 1, 2001. Therefore, the field, *Net Change*, is sensitive to the *Flow Field Date Filter*. To make the *Sales Total* field sensitive to the *Date Filter*, you must first find the true field name of *Date Filter* and enter it into the *Filter Table* window of the *Sales Total* field.

You can find the true field name of *Date Filter* in the *Salesperson/Purchaser* table. You must also find the name of the date field it controls in the *Cust. Ledger Entry* table.

Find the field name of *Date Filter* in the *Field Menu* of the *Salesperson/Purchaser* form window as shown above. Here you see the variable with *Caption* name, *Date Filter*.

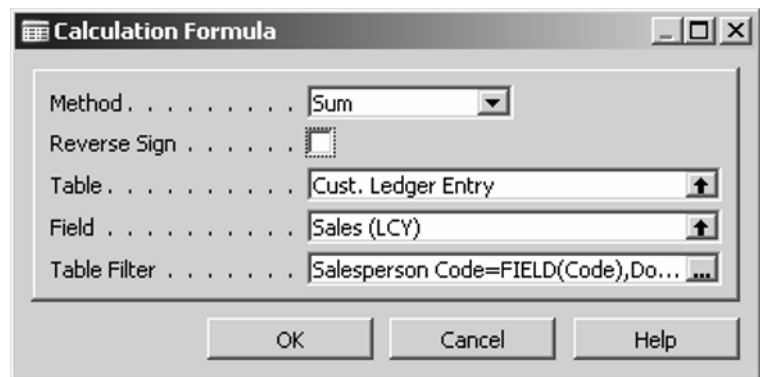
Now look for the name of the field in the *Cust. Ledger Entry* table that contains the posting date. You will see this in the *Field Menu* window from the *Customer Ledger Entries* form above.

Now you can return to the details of the new *Flow Field Sales Total* and define the time period controlling filter. In the leftmost column of the *TableFilters* window enter, *Posting Date*. Next, select *Field* from the second column *Type* list. Lastly, in the third column, *Value*, enter the field name: *Date Filter*. The window *TableFilter* should now look like the following:



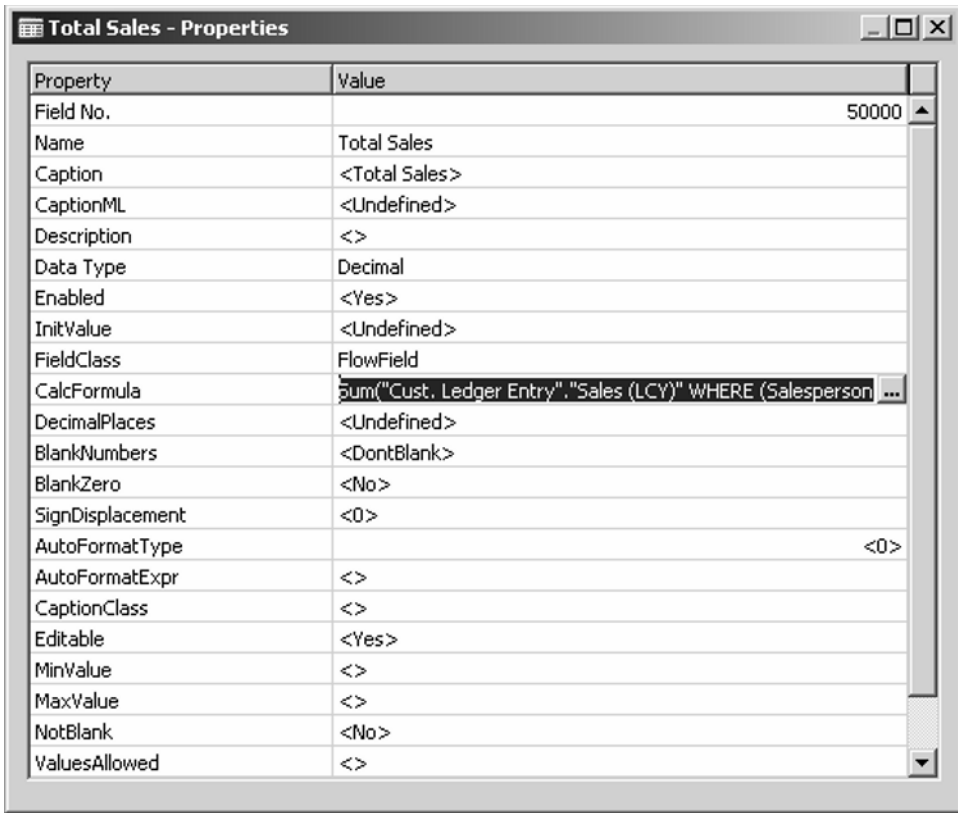
**Figure 5.11** Flow Field Table Filter window

Click OK and the *Calculation Formula* window will reappear. You will see the following window:



**Figure 5.12** Flow Field Calculation Formula window

Click OK which returns you to the *Total Sales - Properties* window. The following window appears:



**Figure 5.13** Total Sales - Properties window

In the property, *CalcFormula*, Navision automatically creates a formula based on the parameters you entered into the *Calculation Formula* and *TableFilters* windows. This formula is written in an SQL syntax that the database can quickly evaluate.

Give the new field a *Caption* name which the end user will see in their forms, reports and views. Enter into the *Caption* property, the text, *Sales*. As you will see, Navision automatically entered this text into the *CaptionML* property with *DEU*, the language key, before it, which signifies that this is the German equivalent for the field, *Sales Total*. If you are operating in US English then the *CaptionML* will display *ENU*.

Next, change the property, *Editable*, to the value, *No*. This is important because a *Flow Field* stores no data and it is therefore nonsense for the user to try to enter information into it.

Press ESC once to leave the field properties window and then again to close the *Salesperson/Purchaser* table. When Navision prompts you to save your changes, click on OK.

### 5.2.5 Indexing the Sales History For the New Sales Flow Field

Now you are finished entering and defining the new *Flow Field* in the *Salesperson/Purchaser* table. You must now create a new index in the transaction table so that Navision can organize a sum of the records for the *Flow Field*. Therefore, open the table you have linked to the *Flow Field*. Go to:

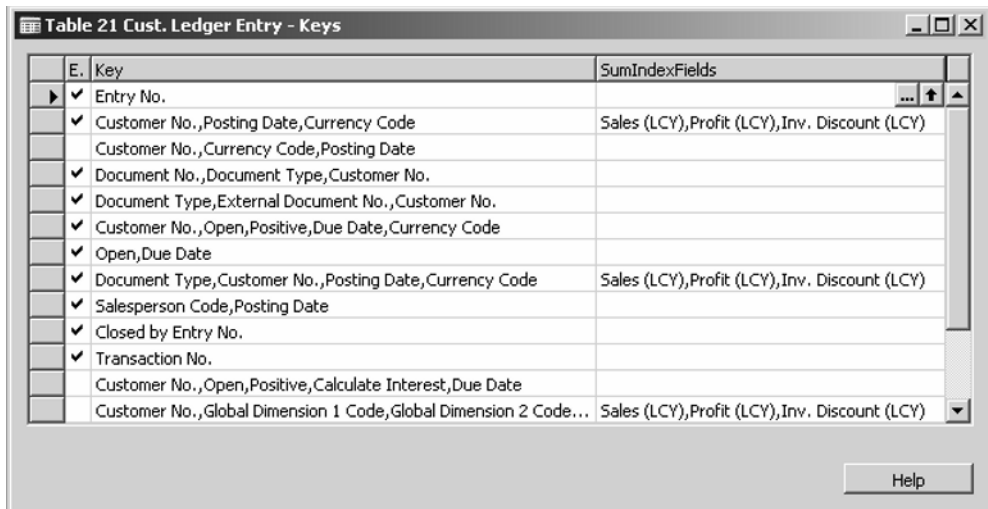
- **Extras > Object Designer**

Click on the Table button. Find the *Cust. Ledger Entry* table and click on the Design button at the bottom of your screen.

Now that you are within the *Cust. Ledger Entry* table, you must enter a new key and index so that your *Flow Field* organizes the records in this table. Go to:

- **View > Keys**

The following window will open:



**Figure 5.14** *Customer Ledger Entry - Keys*

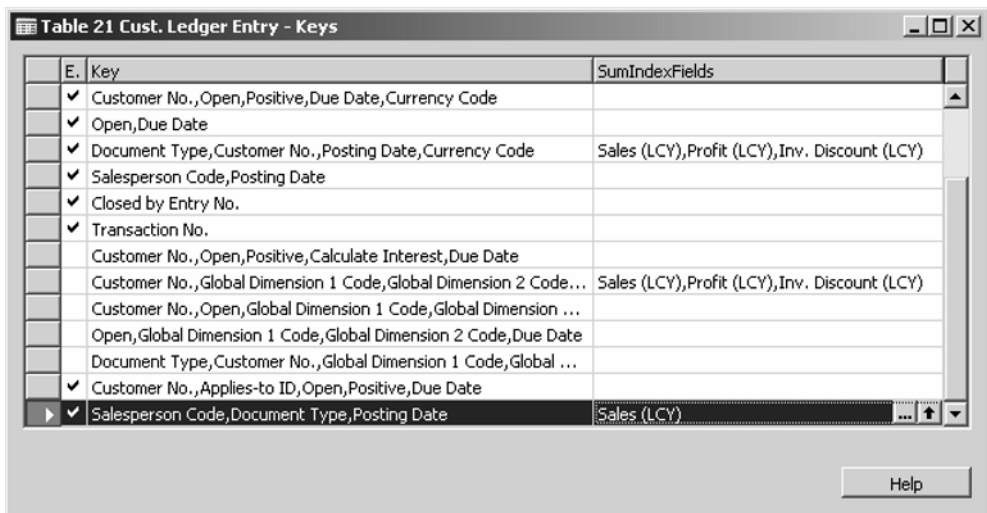
The new key you create must begins with the primary key of the *Flow Field* table so that Navision can group the information in this table according to the information in the stock information table. In this example you will group the *Customer Ledger Entries*

first by the *Salesperson Code*. Click on the *Key* column and then on the first assist. The following window will appear:



**Figure 5.15** Key *Field List* window

In this window you must enter the variables Navision needs to organize the *Sales Total Flow Field*. Either type in or select from the field list that appears when you click on the assist, the following fields: *Salesperson Code*, *Document Type* and *Posting Date*. Next, click OK. The following window will reappear:



**Figure 5.16** *Customer Ledger Entry - Keys* index for the new *Total Sales Flow Field*



In the *SumIndexFields* column, enter the field name: *Sales (LCY)*. This new key allows Navision to sort the *Customer Ledger Entries* first by *Salesperson Code*, then by *Document Type*, next by posting date and lastly sum the field, *Sales (LCY)*, within these records. Next, press ESC twice and answer “Yes” when prompted to save changes.

### 5.2.6 Presentation of the New Flow Field

You have now completed the fundamental work in creating your *Flow Field*. All that remains is the presentation of this new information. Display the new *Flow Field* in the *Salesperson/Purchaser Card* and the *Salesperson/Purchaser* list view forms. Press ESC once to return to the end user environment and then go to:

- **Sales & Marketing > Sales > Salespeople**

Next, open the inner structure of the *Salesperson/Purchaser Card* by pressing CTRL+F2. Now go to:

- **View > Field Menu**

Search for the new *Flow Field* by the field name, *Sales Total*, or the *Caption* name, *Sales Total*. Double-click on the *Sales Total* field and then drag and drop it on the file card. For this example insert this field into the *Invoicing* tab—the second tab in the file card. Now press ESC and save your changes. Press ESC again and then reenter the *Salesperson/Purchaser Card* to reflect the changes.

Next, click on the upward pointing arrow graphic work tool or press F5 to open the list of salespersons. Open the inner structure of this form by pressing CTRL+F2 and then insert the field, *Sales Total*, from the *Field Menu* into the form. Press ESC, save your changes and then reopen the salesperson list. The following list will appear:

The screenshot shows a window titled "Salespeople/Purchasers" with a table containing the following data:

Code	Name	Total Sales	Commission %	Phone No.
AH	Annette Hill	0.00	0.00	
BD	Bart Duncan	0.00	0.00	
DC	Debra L. Core	0.00	0.00	
JR	John Roberts	246,052.99	5.00	
LM	Linda Martin	0.00	0.00	
MD	Mary A. Dempsey	0.00	5.00	
PS	Peter Saddow	904,544.54	5.00	
RL	Richard Lum	0.00	0.00	
*▶		0.00	0.00	

At the bottom of the window, there are buttons for "OK", "Cancel", "Salesperson" (with a dropdown arrow), "Create Inter...", and "Help".

**Figure 5.17** New *Total Sales Flow Field* in the *Salespeople/Purchasers* window

The screenshot shows a window titled "Customer Ledger Entries" with a table of transactions. The last row is highlighted:

Posting ...	Document Type	Docume...	Custome...	Description	Original Amount	Amount	Remaining A
01/15/01	Invoice	103021	30000	Invoice 1003	861.13	861.13	86
01/15/01	Invoice	103026	10000	Invoice SCI0000005	0.58	0.58	
01/16/01	Invoice	103009	20000	Order 101012	215.83	215.83	21
01/17/01	Invoice	103018	10000	Order 6005	4,101.88	4,101.88	4,10
01/22/01	Invoice	103014	20000	Order 101007	1,145.33	1,145.33	1,14
01/22/01	Invoice	103001	10000	Invoice 103001	8,182.35	8,182.35	8,18
01/22/01	Invoice	103002	20000	Invoice 103002	6,971.78	6,971.78	6,97
01/22/01	Invoice	103003	30000	Invoice 103003	5,999.40	5,999.40	5,99
01/23/01	Invoice	103004	50000	Invoice 103004	32,788.40	32,788.40	32,78
01/25/01	Invoice	103025	50000	Invoice SCI0000004	0.73	0.73	
01/14/01	Credit Memo	104001	10000	Credit Memo 104001	-292.84	-292.84	-29
01/16/01	Credit Memo	104002	20000	Credit Memo 104002	-787.40	-787.40	
▶ 01/19/01	Credit Memo	104003	20000	Credit Memo 104003	-1,145.33	-1,145.33	

At the bottom of the window, there are buttons for "Entry" (with a dropdown arrow), "Application" (with a dropdown arrow), "Navigate", and "Help".

**Figure 5.18** *Customer Ledger Entries* from the *Total Sales* drill down function

When you click on the new *Flow Field Total Sales* and then on the assist that appears, the individual sales transactions will appear. Click into the *Total Sales* value for the salesperson, *Peter*

*Saddow*, for example. The following window will appear: (See Figure 5.18 above) As you may notice, Navision shows only the transactions for *Invoices* and *Credit Memos* exactly as requested.

Now your boss can have, in an instant, 100% actual information about the sales performance of the entire sales force and this information is also connected to the details of each customer order.

# 6

## Creating a New Report

---

In this chapter we will consider the inner structure of a report. Step-by-step we will create an example report by practicing the concepts covered in former chapters.

### 6.1 Creating a Hard Copy of New Sales Information

The most powerful and flexible tool for manipulating and presenting information in Navision is the report. The report object can be used to accomplish anything from an automated price change in the item catalogue to creating a complex financial analysis printout. Report programs can run the gamut of very simple to very complex. In the following section we will create a simple report with some additional features to help you understand the general structure of the report object.

Consider this problem scenario:

Suppose your boss comes to you and is very happy with the new *Total Sales Flow Field* that you created. She wants you to create a report that shows the sales results by product as opposed to being connected to customer invoices as with the *Flow Field*. Likewise, she wants to see the profit margins and commission amounts per article in the same report. She needs a printable form to give to the salespeople so she can encourage them to concentrate on products with high margins.

Solution:

You will need to use a report object to create a printed analysis. This program will rest on the basic principles of relational data organization, therefore you will need to construct a linked table hierarchy, filter on the records in these tables and finally, make decisions about the information that is generated.

### 6.2 Searching For the Sources of Data

Let us begin by finding the necessary tables, table names, keys, fields and field names.

*locating tables, fields and visualizing the table relations for our new reports inner structure*

After determining what kind of information you need and where it is located, create a data model or diagram to help you visualize the inner workings of the report you must generate.

The boss wants product sales by salesperson so you will need to base your program on the salesperson table. You have just used this information in the previous *Flow Field* creation example, so simply restate this information from the previous section.

The salespeople are located in the *Salesperson/Purchaser* table. The field name of the primary key is, *Code*, and contains the code of the salesperson. You will also need an additional variable from the *Salesperson/Purchaser* table for this project. That variable has the caption, *Commission %*, and the field name of this variable is, *Commission %*. You will need this variable to calculate the provision the salesperson receives from each product sold. Please note that this commission is recorded as a whole percent—a range from 1 to 100. You must, therefore, divide the *Commission %* by 100 before multiplying it with the sales to find the correct sales commission.

Now you must find a table that contains a field for the salesperson code, product information and sales information. It is possible that no one table exists in Navision with all this information. When the information you need is not in any single table, your report must then connect to this information indirectly, which makes the report more complex. What makes a report exceptionally useful is its ability to bring together information from separate Navision tables. Fortunately, in this example, there is a single Navision table that contains all the information needed for your product sales transactions.

First look in the *Item Card* where the product information is most likely to be. Go to:

- **Warehouse > Planning & Execution > Items**

You need historical information about the products, which you can find in *ItemLedger Entries*. Find the *ItemLedger Entries* by clicking on the Item button at the bottom of the screen, then on:

- **Entries > Ledger Entries**

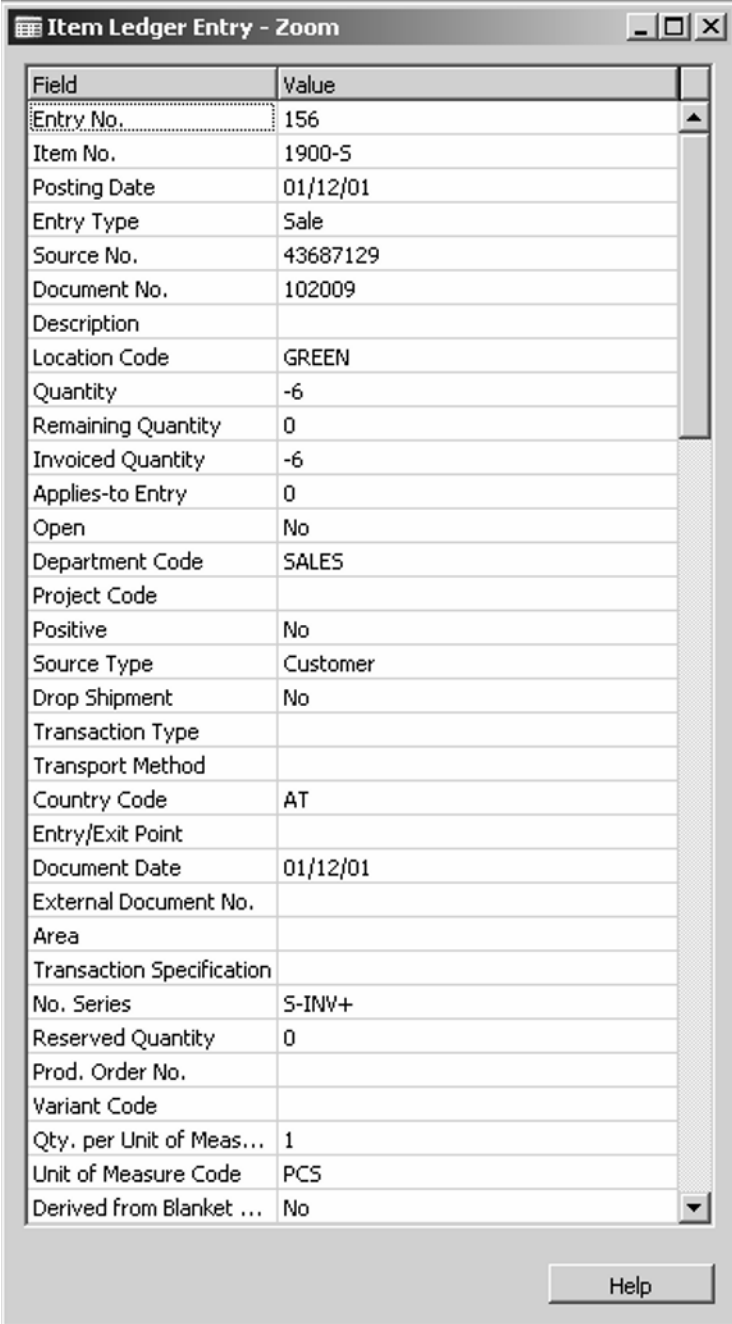
The follow window will appear:

Posting ...	E..	Docume...	Item No.	Location ...	Quantity	Invoiced...	Remainin...	Sales Amoun...	Cost Amo
12/31/00	P..	START	1896-5	GREEN	49	49	48	0.00	23,5
01/10/01	S..	102005	1896-5	GREEN	-1	-1	0	629.92	-5
01/16/01	S..	107002	1896-5	GREEN	1	1	1	-629.92	5
01/25/01	T..	108005	1896-5	OUT. LOG.	25	25	25	0.00	12,0
12/31/00	P..	START	1896-5	RED	52	52	19	0.00	25,0
01/13/01	S..	102010	1896-5	RED	-1	-1	0	649.40	-5
01/19/01	S..	107004	1896-5	RED	1	1	1	-649.40	5
01/21/01	S..	102021	1896-5	RED	-6	0	0	0.00	
01/22/01	S..	102022	1896-5	RED	-1	0	0	0.00	
01/25/01	T..	108005	1896-5	RED	-25	-25	0	0.00	-12,0
12/31/00	P..	START	1896-5	YELLOW	160	160	160	0.00	77,0

**Figure 6.1a** *Item Ledger Entries* window

Look to see if you can locate the field that you need to connect to your *Salesperson/Purchaser* table by using the Zoom function. Go to:

- **Tools > Zoom**



The screenshot shows a window titled "Item Ledger Entry - Zoom" with a table of fields and values. The table has two columns: "Field" and "Value". The "Entry No." field is highlighted with a dotted border. A vertical scrollbar is on the right side of the table. A "Help" button is located at the bottom right of the window.

Field	Value
Entry No.	156
Item No.	1900-5
Posting Date	01/12/01
Entry Type	Sale
Source No.	43687129
Document No.	102009
Description	
Location Code	GREEN
Quantity	-6
Remaining Quantity	0
Invoiced Quantity	-6
Applies-to Entry	0
Open	No
Department Code	SALES
Project Code	
Positive	No
Source Type	Customer
Drop Shipment	No
Transaction Type	
Transport Method	
Country Code	AT
Entry/Exit Point	
Document Date	01/12/01
External Document No.	
Area	
Transaction Specification	
No. Series	S-INV+
Reserved Quantity	0
Prod. Order No.	
Variant Code	
Qty. per Unit of Meas...	1
Unit of Measure Code	PCS
Derived from Blanket ...	No

**Figure 6.1b** *Item Ledger Entry - Zoom* window

There is no *Salesperson Code* here, therefore, you cannot connect the *Item Ledger Entries* directly to the *Salesperson/Purchaser* table.

Look a little deeper in the *Item Ledger Entries*, particularly at the column, *Sales Amount (Actual)*. When you click on the value in this column, an assist appears. This automatically reveals that the variable, *Sales Amount (Actual)*, is a *Flow Field* that is merely displaying a sum calculated from a linked table. Look in the table linked to this *Flow Field*. Click on the assist in the *Sales Amount (Actual)* column. Remove the *TableFilter* on this view by clicking on the graphic tool symbol, Show All. The following window will appear:

Posting Date	Item Ledger Entry Type	Entry Type	Document No.	Sales Amount (Actual)	Cost Amount (Actual)	Cost Amount (Non-Invtbl.)	Cc G/
09/09/01	Consumption	Dir...	1011003	0.00	-156.24	0.00	
09/09/01	Consumption	Dir...	1011003	0.00	-33.92	0.00	
09/09/01	Consumption	Dir...	1011003	0.00	-115.20	0.00	
09/09/01	Consumption	Dir...	1011003	0.00	-251.20	0.00	
09/07/01	Output	Dir...	1011001	0.00	0.00	0.00	
09/08/01	Output	Dir...	1011002	0.00	0.00	0.00	
12/11/00	Sale	Dir...	103019	342.60	0.00	0.00	
12/11/00	Sale	Dir...	103019	374.20	0.00	0.00	
12/11/00	Sale	Dir...	103019	346.30	0.00	0.00	
12/03/00	Sale	Dir...	103020	187.10	0.00	0.00	
12/03/00	Sale	Dir...	103020	346.30	0.00	0.00	
01/15/01	Sale	Dir...	103021	346.30	0.00	0.00	

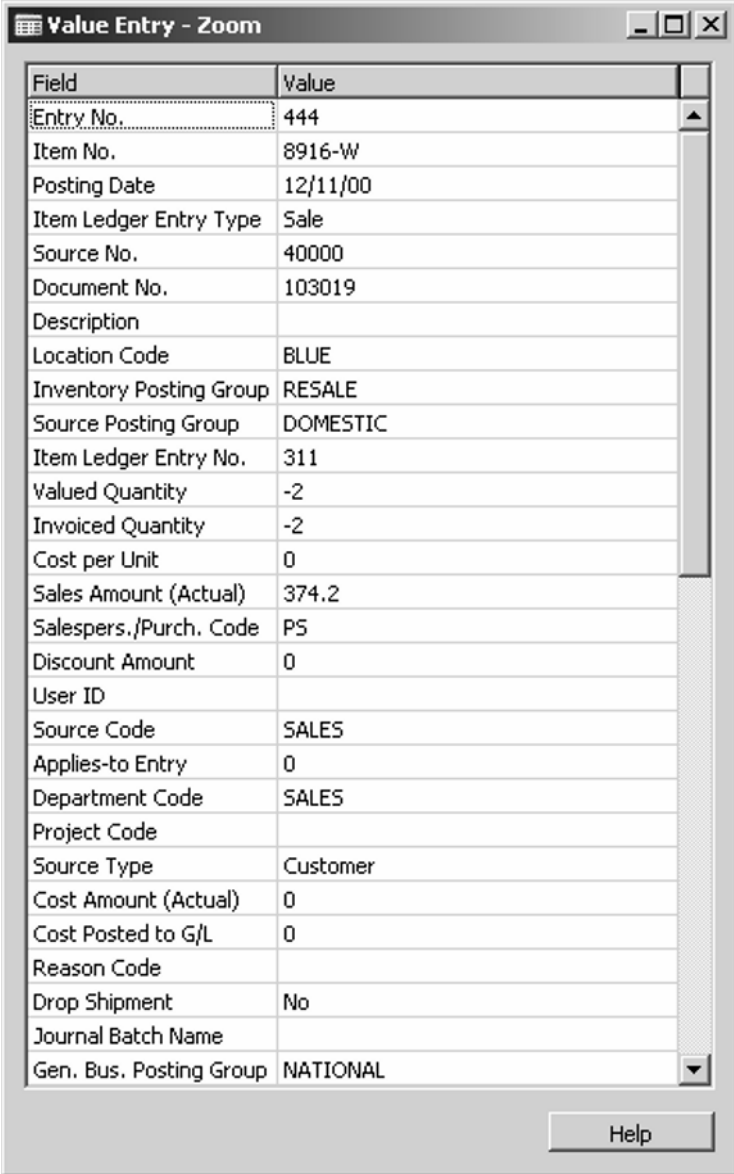
**Figure 6.2** *Value Entries* table

Above is the table view, *Value Entries*. In this table view the variable, *Sales Amount (Actual)*, is not a *Flow Field* but rather a fixed amount written into a table. Look to see if the primary key of the *Salesperson/Purchaser* table is included in the table underlying this table view. Go to:

- **Tools > Zoom**

This following window will appear where you will see all of the variables in the table underlying the *Value Entries* table view:





The screenshot shows a window titled "Value Entry - Zoom" with a table of fields and their corresponding values. The table has two columns: "Field" and "Value". The "Entry No." field is selected with a dotted border. A vertical scrollbar is on the right side of the table. A "Help" button is located at the bottom right of the window.

Field	Value
Entry No.	444
Item No.	8916-W
Posting Date	12/11/00
Item Ledger Entry Type	Sale
Source No.	40000
Document No.	103019
Description	
Location Code	BLUE
Inventory Posting Group	RESALE
Source Posting Group	DOMESTIC
Item Ledger Entry No.	311
Valued Quantity	-2
Invoiced Quantity	-2
Cost per Unit	0
Sales Amount (Actual)	374.2
Salespers./Purch. Code	P5
Discount Amount	0
User ID	
Source Code	SALES
Applies-to Entry	0
Department Code	SALES
Project Code	
Source Type	Customer
Cost Amount (Actual)	0
Cost Posted to G/L	0
Reason Code	
Drop Shipment	No
Journal Batch Name	
Gen. Bus. Posting Group	NATIONAL

**Figure 6.3** *Value Entry - Zoom* view

Here you see all the variables necessary for your current project. They do not have the same names shown in Figure 7.2. Likewise, some of the *Value* fields are empty. The variable, *Salespers./Purch. Code*, is equivalent to the primary key field, *Code*, within the salesperson table. With the *Salespers./Purch. Code*

field you can create a link to the primary key of the *Salesperson/Purchaser* table.

You can also find the *Item No.* variable here which you can use to track the product in each transaction, as well as the cost variable, *Cost Amount (Actual)*, necessary in calculating the *Profit* of each product sold. Note that the cost value, *Cost Amount (Actual)*, is already recorded with a negative sign. For this example, assume that the *Profit* equals the sales amount plus the negative cost amount. Lastly, we see here the variable, *Sales Amount (Actual)*, which contains the necessary sales information.

The next question to consider is how these variables can be absolutely empty—including numeric fields which have *Value, 0*? More importantly, will this cause a problem when both tables are linked? The answer to this question is, "No." You must be careful to link the tables only where there is a match between the variables, *Salespers./Purch. Code*, and the field, *Code*, within the *Salesperson/Purchaser* table. These variables are empty because the *Zoom* window currently shows a record that involves the internal movement of a product. That is, the movement is the result of a production transaction and does not involve a salesperson. However, when there is a movement of a product as a result of a sale, this table will record it, including the salesperson code. You can *Zoom* into other *Value Entries* where there is a sales transaction to observe this.

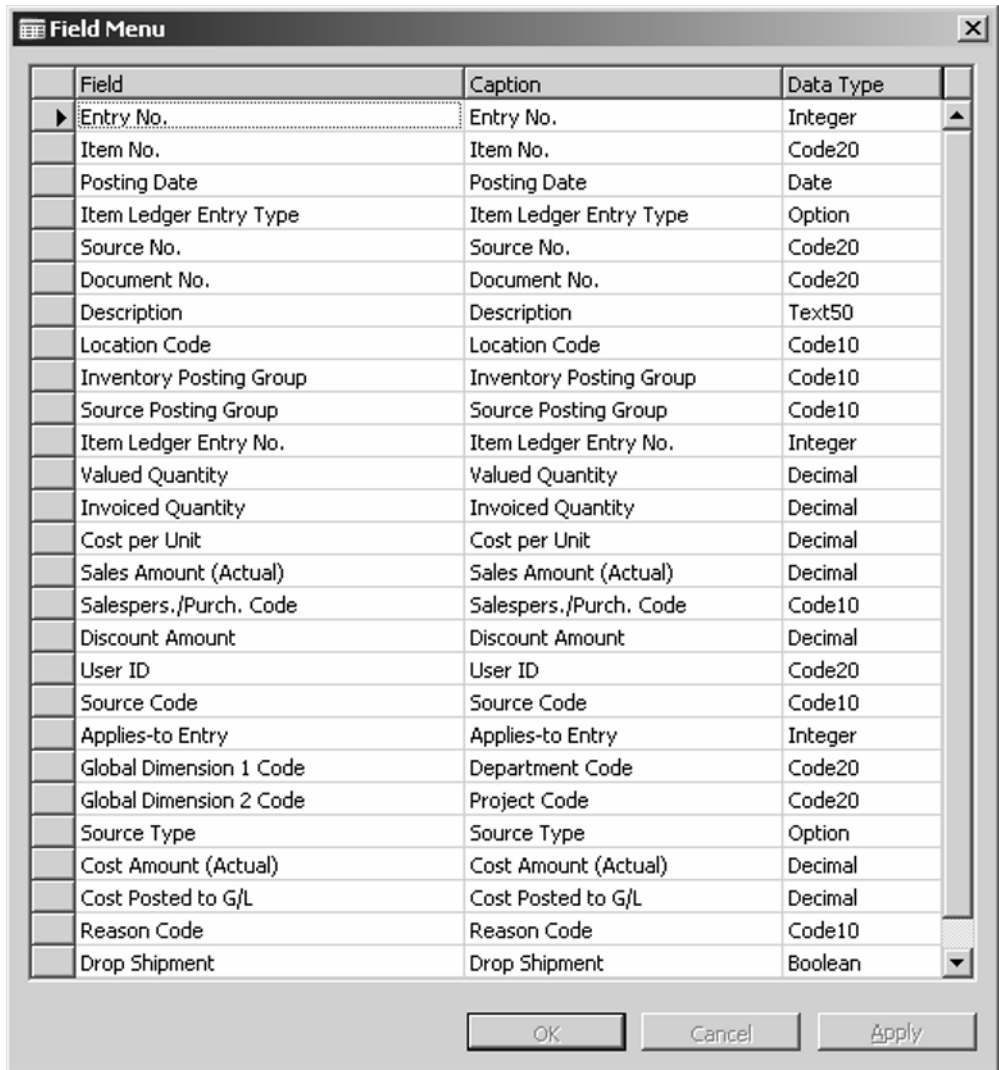
Now that you have found a useful transactions table, you must determine its table name and the field names of the variables you need from it. Press CTRL+F2 to open the inner structure of the form. Next, click into the empty gray area surrounding the form object and go to:

- **View > Properties**

The name of the underlying table, *Value Entry*, is written in the property *SourceTable*. Next, leave the properties by pressing ESC and then open the *Field Menu* by going to:

- **View > Field Menu**

The following window will appear:



**Figure 6.4** *Field Menu* window

Here you will see that the important variables have the field names: *Salespers./Purch. Code* for *Salespers./Purch. Code*, *Item No.* for *Item No.*, *Sales Amount (Actual)* for *Sales Amount (Actual)* and finally *Cost Amount (Actual)* for *Cost Amount (Actual)*.

**\*Note:**

***As the inner structure of Microsoft Navision is normally written in English, the field name and caption will usually agree in the English version of the software.***

### 6.3 Diagram of the Table Relationships For the New Report

Now let us construct a quick diagram of the tables, fields and table relations that are needed to create the report, which you can name, *Salesperson Commission by Item*. Include in your diagram some values merely for illustration purposes. You can use the names of the variables, tables and options as they appear within the Navision standard development environment. Consider the following diagram: (See Figure 6.5)

### 6.4 Diagram of Information Flow Behind the Report

Quickly review the information to be provided by the report: You need a list of each salesperson. For each salesperson you want to see the sales, margins and commissions per product that they have sold, as well as a sum of all the products sold. Lastly, you want to see a sum of these variables for all the salespeople together. Now you must consider the order of the steps Navision must go through to return the desired information. This information flow diagram will help you visualize what it is you need the report program to accomplish.

This flow of information basically encompasses three processes:

1. The movement through tables
2. Performing calculations on information where a match is found between the table keys
3. Displaying and printing this table and calculated information

These are the basic steps the report must go through to bring together and display the product sales and commission of every salesperson. This seems like a complex development, but in reality, this is the layout of a simple report. After some practice this data processing procedure will seem very obvious and natural. The key, as always, is to see that you are merely linking tables by a primary key, searching through the records of the tables looking for matching keys and finally, making decisions based on the values found.

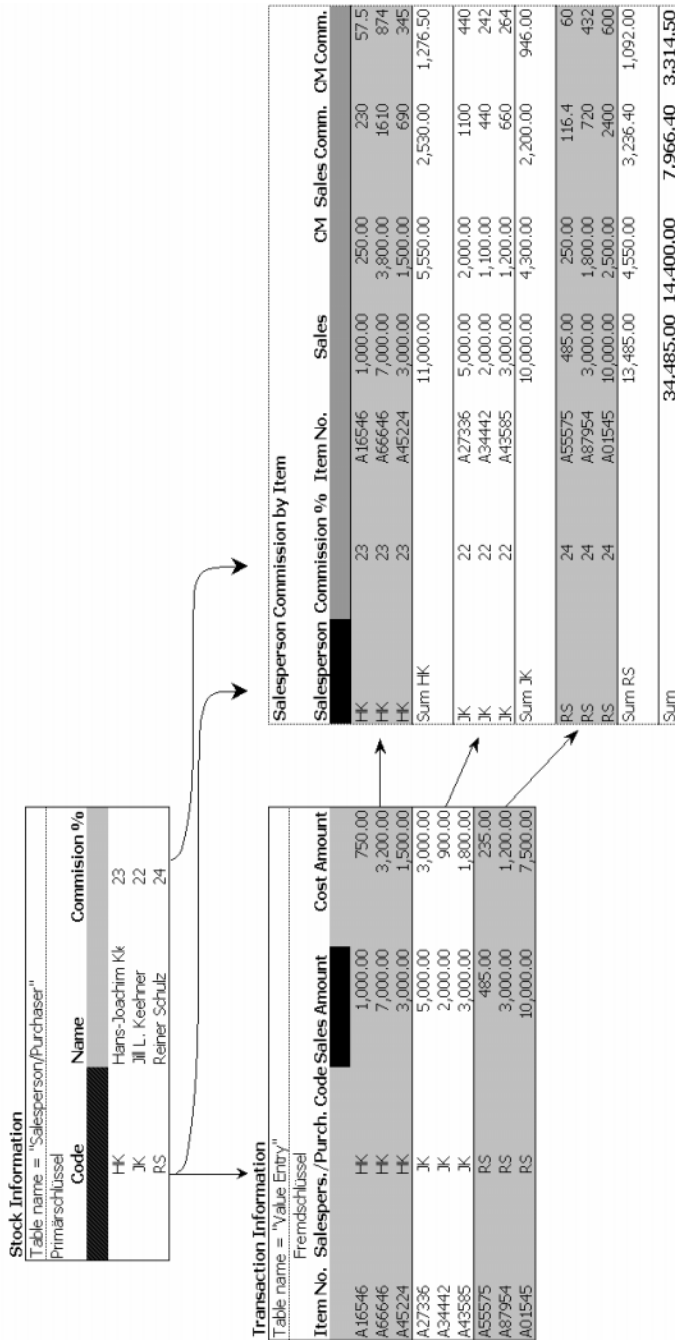
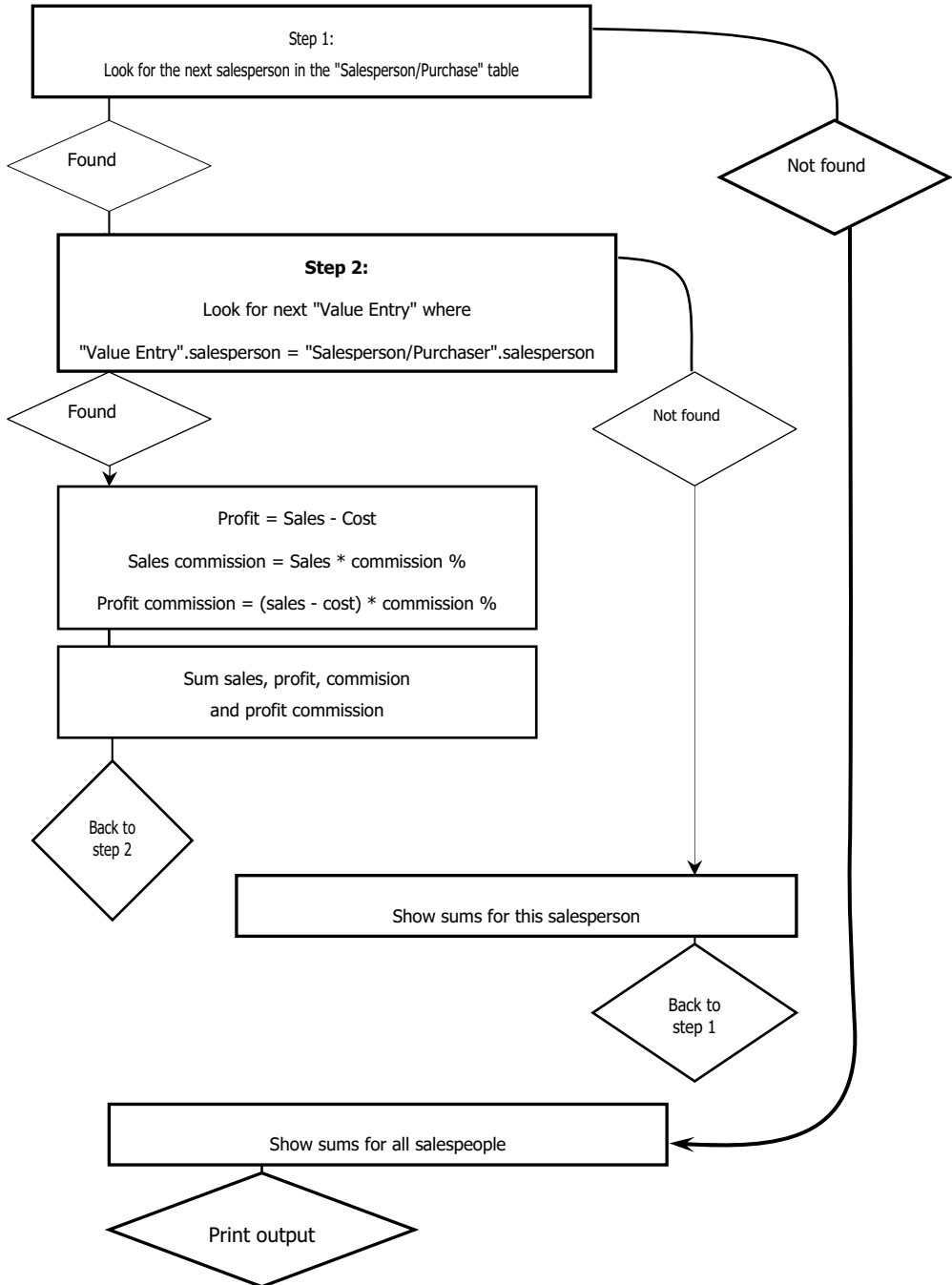


Figure 6.5 Diagram of the table relationships for the report

6.4 Diagram of Information Flow Behind the Report



## 6.5 Introduction to Report Designer

Now you are ready to step into the structure of the Navision report development environment—the *Report Designer*. Most parts of the inner report structure can be roughly divided into parts corresponding to the three basic steps noted above.

Now go to the *Object Designer* by selecting:

- **Tools > Object Designer**

Click on the graphic symbol tool, Report, on the right of the window. Now you can begin to create a new report. You must have a report *Object ID* number in mind. This ID must be unique among reports and permissible according to the parameters of your Navision license. For this example, use the report ID number, 50000. Click on the New button at the bottom of the window. The following window will appear:

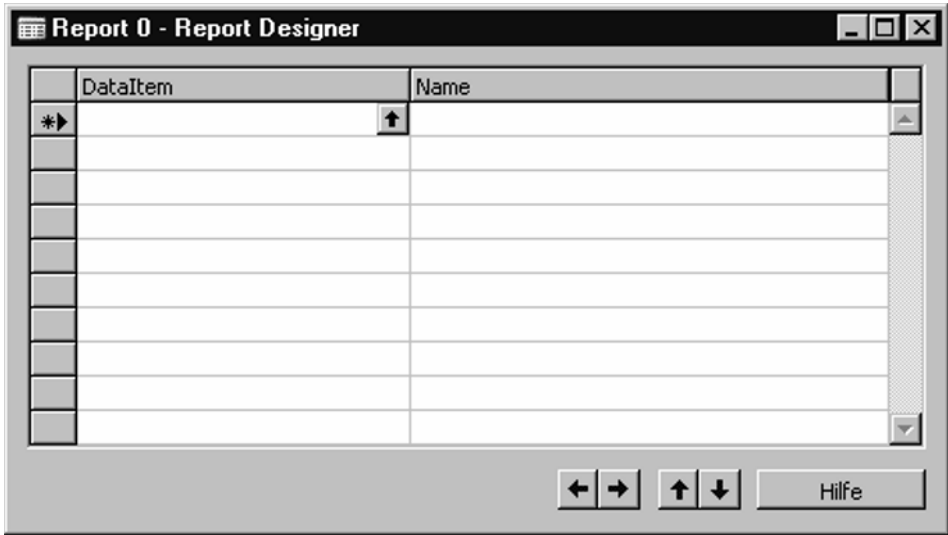


**Figure 6.6** *New Report Wizard* window

The *New Report Wizard* program helps you create extremely simple reports, consisting of a list of the contents of a single table. Normally, a report is needed when you wish to bring together information located in more than one table. In this case, *Report Wizard* is not very useful which means you will need to learn more sophisticated tools.

### 6.5.1 Dataltem Structure

Select the Create a blank report button and then click OK at the bottom of the window. The following window will appear:



**Figure 6.7** Report Designer window

This is the first and most important level of the report structure. Here you will enter the table names in the column, *DataItem*, and define the relations between the tables. From here you will also access the general properties of the report.

Now we will look in more detail at the steps Navision must execute when moving through the records of the linked tables. Then we will show how this process can be constructed in the report using the *DataItem* list and the *DataItem* properties. When Navision moves through two linked tables it does the following:

1. Sorts the parent table according to the user's sorting instructions. If the user defines no special sorting, Navision automatically sorts the table by its primary key in ascending order.
2. If the user has defined a filter for the parent table, Navision limits the records that the database can access according to the user-defined table filter.
3. Opens the parent table and searches for the first record in that table. When and if it finds the first record, it reads



the contents of the primary key field/fields of this first record.

4. Sorts the linked child table according to user instructions or if user has no instructions Navision sorts the child table by its primary key in ascending order.
5. Sets a table filter on the child table if the user has defined a filter for the child table.
6. Opens the child table and searches through the field/fields in the child table—now linked to the parent primary key field/fields—until it finds a record where the contents of the key field/fields in both linked tables are identical.
7. Continues to step through the records of the child table reading each record where it finds a match in the contents between the linked field/fields.
8. Returns to the parent table when it has processed the last match in the child table.
9. Reads the primary key of the next record in the parent table.
10. Repeat steps 7 - 9 until it has matched the primary key contents of every accessed record in the parent table to the contents of every match it finds in the child table.
11. Ends the report when no additional records are found in the parent table.

The goal of designing a report is to open, sort, filter and link tables so that when matches occur between tables information can be calculated and output. This is the process you must simulate within the report program structure.

Now look at the *Report Designer* window to see how you can build this process within Navision. When you wish to construct this process for a report, you must enter a *DataItem* into the *DataItem* list and then enter another *DataItem* in the list below it. The *DataItem* will probably be a table you wish to use but could also be a virtual table. The *DataItem* in the bottom position must be indented one position to the right more than the *DataItem* it is linked to. Indenting means entering at least one empty space before the *DataItem* in your child table. In the properties window of the indented or child *DataItem* you must define the proper sorting, filtering and linking information. For example, if you want to link a salesperson *DataItem* to a cus-

tomers order *DataItem*, you must list the salesperson item first and then list and indent the customer order item. Next, you must go into the customer order item and define the table linking conditions. Navision then goes through every record in the customer order table for each salesperson found in the salesperson parent table.

The *DataItem* list is structured vertically so that which comes after is always subordinate to what proceeds it. This design reflects the hierarchical relational data organization strategy. Therefore, you can link a parent table to a child table below it. Using the arrow buttons at the bottom of the window, you can move a table *DataItem* around to indent it.

Type the table name, *Salesperson/Purchaser*, into the *DataItem* column or click on the assist in this column and select this table from the list that appears. Now enter the table you desire to be linked and controlled by the *Salesperson/Purchaser* table. Enter *Value Entry* into the second row of the *DataItem* window. Next, click on the arrow pointing right, moving the *DataItem Value Entry* one space to the right. Indenting automatically subordinates the *Value Entry* table to records in the *Salesperson/Purchaser* table. Now, for every record in the *Salesperson/Purchaser* table Navision will read all the records in the *Value Entry* table and look for matches between the linked field/fields.

Navision will automatically enter the table names into the second column, *Name*. Within the report you must refer to the text written into the *Name* column of a *DataItem* for Navision to recognize the correct *DataItem*. Occasionally, you need to use the same table more than once within a report, therefore, you need to rewrite the *DataItem Name* to distinguish between them. For example, you could write the same table into two *DataItem* positions with the name of the first being, *Table Xa*, and the second, *Table Xb*.

Now save this report before forgetting the report ID number reserved for it. Press CTRL+S or go to: **File > Save** to save. Enter the name, *Salesperson Commission by Item*, into the report name field and give the report ID number 50000, or another that is appropriate to your specific system and license. Leave the check mark in the field, *Compiled*. This saves the report in an interpreted form which Navision can execute immediately.

Now you must define the important table relations. Click on the *Value Entry DataItem* and then go to:

- **View > Properties**

The following window will appear:

Property	Value
<i>DataItemIndent</i>	1
DataItemTable	Value Entry
DataItemTableView	<Undefined>
DataItemLinkReference	<Salesperson/Purchaser>
DataItemLink	<Undefined>
NewPagePerGroup	<No>
NewPagePerRecord	<No>
ReqFilterHeading	<>
ReqFilterHeadingML	<>
ReqFilterFields	<Undefined>
TotalFields	<Undefined>
GroupTotalFields	<Undefined>
CalcFields	<Undefined>
MaxIteration	<0>
DataItemVarName	<Value Entry>
PrintOnlyIfDetail	<No>

**Figure 6.8** *Value Entry - Properties* window

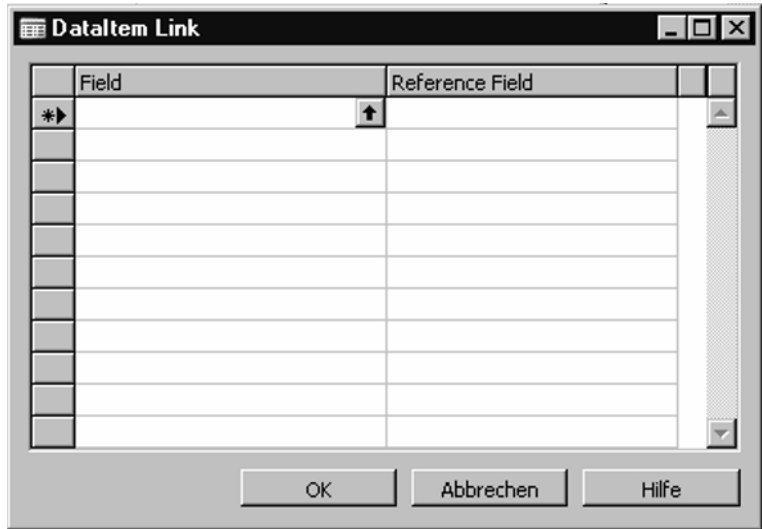
Below is a listing and description of the most important properties of *Value Entry*:

*DataItemIndent* Tells you if this table is linked to a table above it. The integer indicates how many levels of linkages this table is built upon. In this example, the *Value* is 1, indicating that the entire *DataItem* is run for every single record in the original *DataItem* of the report.

<i>DataItemTable</i>	This is the table name where records will be searched in this <i>DataItem</i> .
<i>DataItem-TableView</i>	Define the sorting of the table here or set filters on the table to control which records Navision will have access to when searching through this table.
<i>DataItemLink-Reference</i>	Define the parent table you wish to link to. Enter the name of a <i>DataItem</i> that precedes the <i>DataItem</i> list in your report.
<i>DataItemLink</i>	Create a link between this table and that listed in the <i>DataItem-Link</i> property. Define which field/fields must match between this table and the <i>DataItemLink</i> table.
<i>TotalFields</i>	Enter the fields, the values of which Navision should be accumulating as Navision goes through the records of the <i>DataItem</i> table.
<i>GroupTotalFields</i>	Indicate field totals that should be displayed in the output each time the field value changes. For example, if you enter a salesperson here, Navision displays a total each time the salesperson changes. It is very important to define the sorting of the table so that Navision will not randomly find a change in the values of the <i>GroupTotalFields</i> , but instead each time a new group of them occurs. Therefore, the fields you enter in <i>GroupTotalFields</i> should be defined in the <i>DataItemTableView</i> sorting option. For example, you may wish to have the list of salespersons grouped by job function and print a group total for each job function. To accomplish this, enter <i>Job Title</i> as the table sorting option in <i>DataItemTableView</i> and in the <i>GroupTotalFields</i> property. Navision can now print output for each group of <i>Job Title</i> , using the special output sections, <i>GroupHeader</i> and <i>GroupFooter</i> , which are activated when the value in the <i>GroupTotalFields</i> field changes.

We will discuss the other properties as needed.

You can see that Navision has automatically entered the name, *Value Entry*, into the *DataItemTable* property because it was entered in the *DataItem* list. Now you can link the two *DataItems*. Enter *Salesperson/Purchaser* into the *DataItemLinkReference* property. This is the name of the *DataItem* for the *Salesperson/Purchaser* table entered into the *DataItem* list. Now that Navision knows what two objects to link, you must enter the specific conditions under which they link. This is entered into the property, *DataItemLink*. Click onto this property and then onto the assist that appears in the right of the *Value* column. The following window will appear:



**Figure 6.9** *Dataltem Link* window

In this list you must enter those fields which must be identical between the two linked tables. The column in the left is the under-ordered or child table in the table relation you are building and the column on the right is the over-ordered or parent table in your table relation. Enter the field, *Salespers./Purch. Code*, from the *Value Entry* table in the left column and the field, *Code*, from the *Salesperson/Purchaser* table in the right column. You can simply type in these field names or select them from the field lists that appear when you click on the assists in either column. Now the *Dataltem Link* window appears as follows:

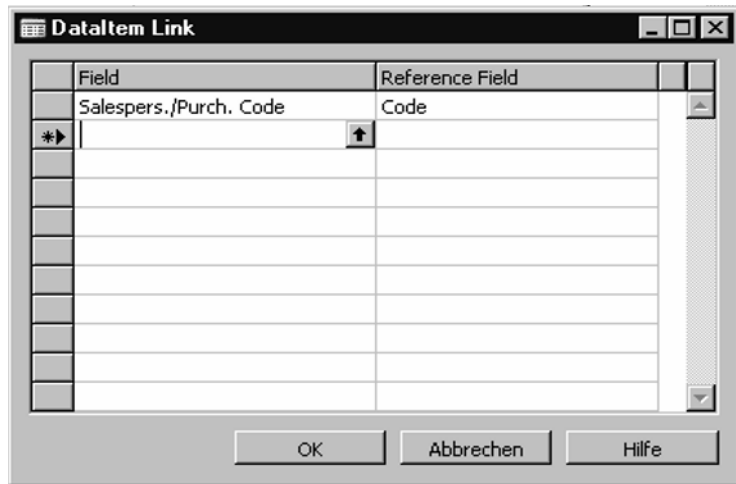


Figure 6.10 DataItem Link window

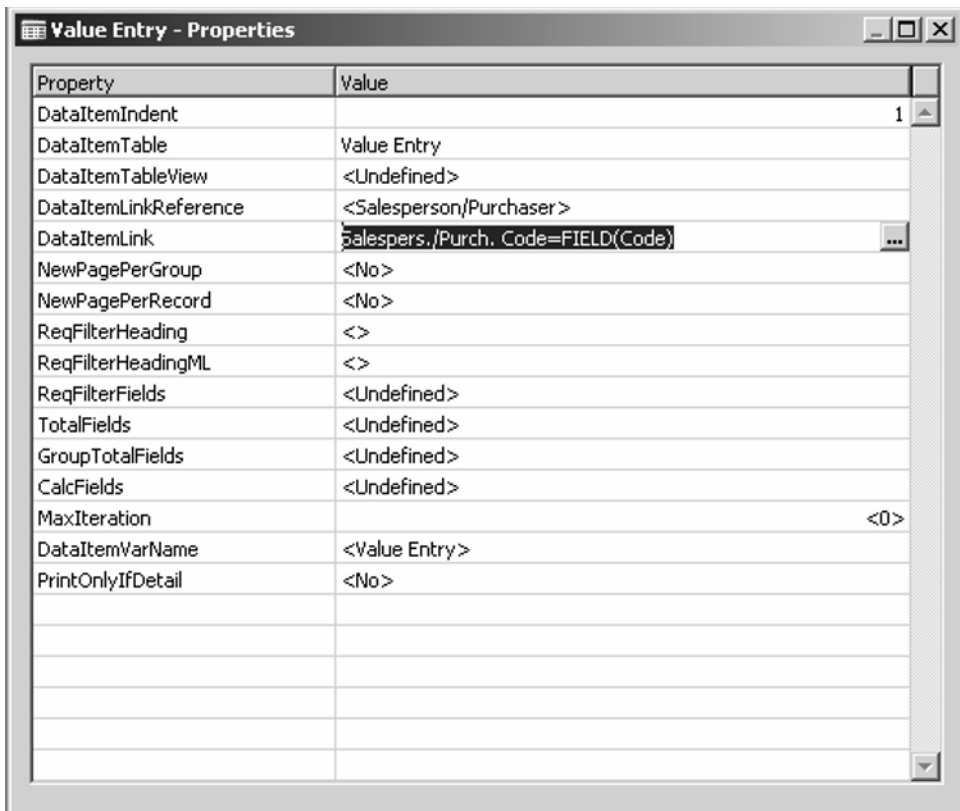


Figure 6.11 Value Entry - Properties window

Click OK to return to the *Value Entry - Properties* window. The following window will appear: (See Figure 6.11 above)

Navision has automatically created the SQL syntax for the table relation and entered it into the *DataItemLink* property. This syntax tells the database that “only records where the salespersons are equal will be considered in the table linking, while all others will be ignored.”

With this simple information entered, you have created the backbone of the information flow in your entire report. Creating a table relation in the report, as you just did, sets up the information flow that occurs when Navision executes the report program. Now, Navision will open each record in the *Salesperson/Purchaser* table and then search for and open any *Value Entry* records that have the same salesperson. All you have to do now is tell Navision what to do when it finds this information and how to present the results.

## 6.5.2 **General Report Properties**

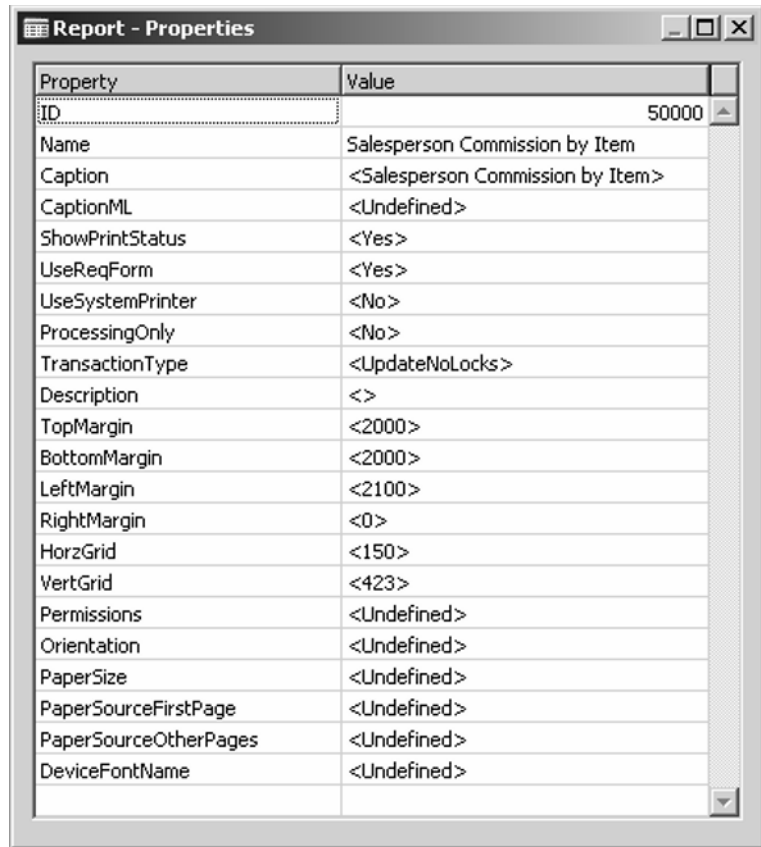
Now take a look at the properties of the report as a whole. Press ESC to leave the *Value Entry - Properties* and return to the *DataItem* list. Place your cursor beyond the last entry in the *DataItem* list in an empty position and then go to:

- **View > Properties**

The following window will appear: (See Figure 7.12) Here are the properties that apply to the report as a whole. The ID and name were already determined when you saved the report. Let us enter a German name into the property, *Caption*. Enter the text: *Sales Commission by Item*. Here is a brief description of the other important properties.

### *UseReqForm*

Allows or disables a page in a report where the user can enter special options and report functions. This user-input page is called the *Request Form* and is displayed always as the last tab—called *Options* in the pre-run report.



**Figure 6.12** General *Report - Properties* window

### *ProcessingOnly*

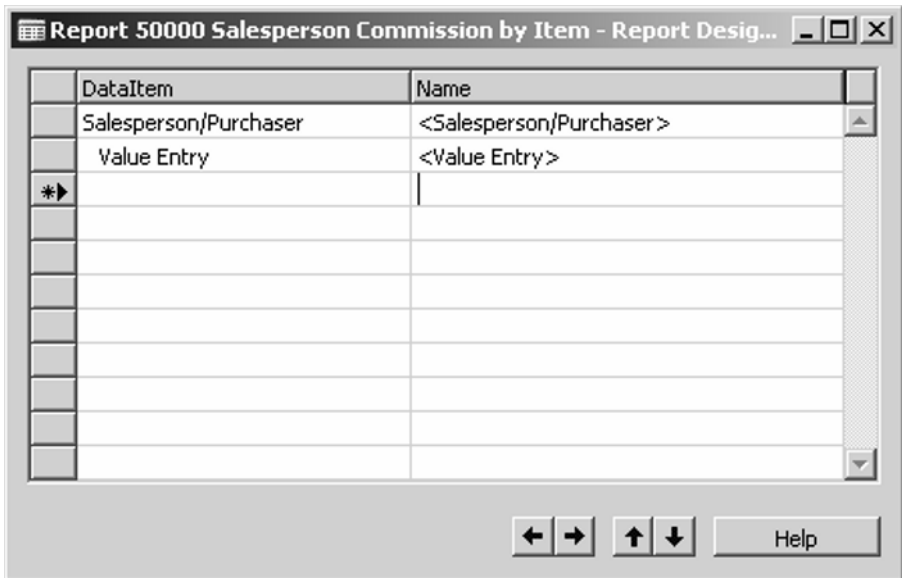
Allows or disables the report output for display or for printout. When set to *Yes*, Navision will not show a report view window or print any report results. This is useful when the report is designed merely to perform data editing since displaying output would only slow down processing.

Other properties will be discussed as needed.

Before printing the output of this report for your boss to distribute to the salesforce, you must make sure that the property, *ProcessingOnly*, is set to, *No*.

Now return to your *DataItem* list which now looks like the following:





**Figure 6.13** *DataItem* window

It is a good practice to compile your program every few minutes or each time you want to check errors in a feature you have created. You can do this by going to:

- **Tools > Compile**

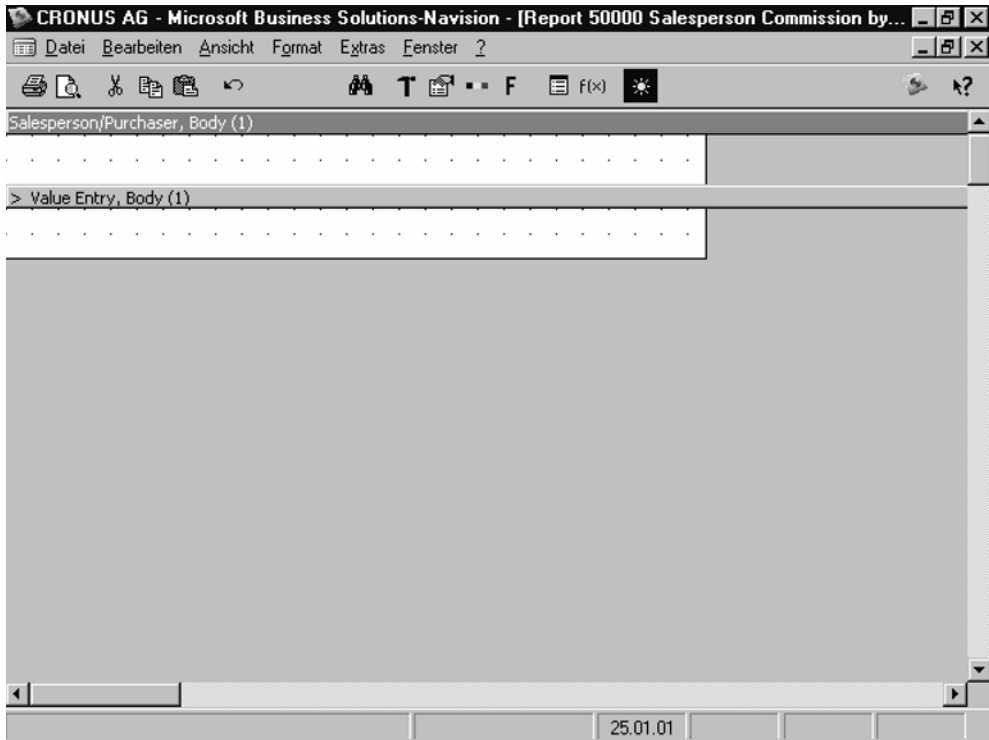
The same can be accomplished by pressing F11. Compiling the program translates it into a form that Navision can execute. If you make any syntax or structural errors the compiler will stop and give you an error message. This is the first level of debugging your program. It is useful to perform this compiling procedure often as it will stop you from building syntax mistakes upon other syntax mistakes.

### 6.5.3 Designing a Report Printout

The next step in your information flow is to perform the calculations necessary for your output. Because you are working with relatively simple output, you can perform most of the calculations within the output display designer. Go to the report output display designer area by selecting:

- **View > Sections**

The following window will appear:



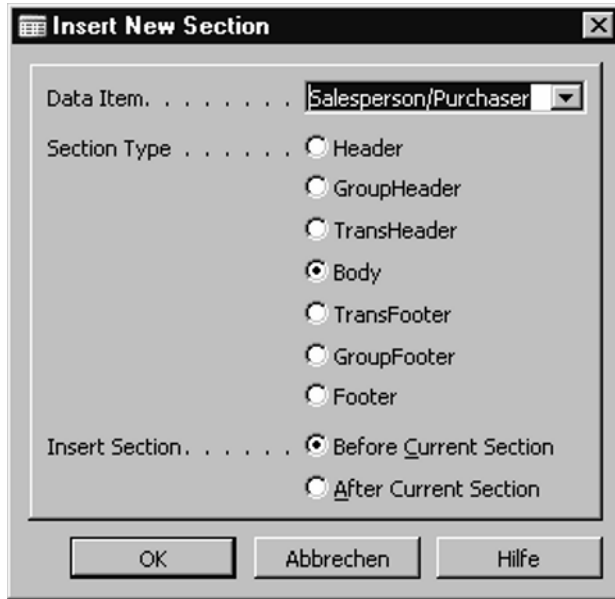
**Figure 6.14** Section Designer

In this environment you will learn to control the display and printout of your report. Each horizontal gray bar symbolizes a data processing step. These various steps correspond to the same steps shown earlier in the information flow diagram. These horizontal gray bars are referred to as *sections*, thus the display design environment is called the *Section Designer*. The basic principle of the designer layout allows you to create a section that corresponds to each and every record and event that will occur in the *DataItems* which you have already defined. For example, you could create a section wherein each time a new salesperson is found in the *Salesperson/Purchaser* table, a section containing the name of that salesperson is displayed. The *Section Designer* is so powerful that you can control exactly what will be printed and when. You will need to learn which sections types will be active and when.

Now we will briefly discuss the various types of sections that can be used. Go to:

- **Edit > Insert New**

to view the *Insert New Section* window. The following window will appear:



**Figure 6.15** *Insert New Section* window

Here you can choose the various section types and the table or *DataItem* that you desire the section to be activated by. Below is a list and description of the section types:

*Header*

The section that is displayed each time Navision enters or re-enters a *DataItem* and finds a record. It is activated before Navision reads and processes the first record in the newly entered table. It can also be defined so that it is displayed for each new page of the report. This section appears each time, for example, that Navision reenters a child table from the parent table in a hierarchical table relation.

*GroupHeader*

A special type of header is activated when Navision determines that the value has changed in a predefined field in a table. The field with the value that activates such a header must be specified in the property of the section *DataItem*, *GroupTotalFields*.

*TransHeader*

An intermediate header that can be determined to print at the top of pages following the first page. Useful when you do not wish to repeat all the header information of the first page throughout the entire report. Using the *TransHeader* you can re-

<i>Body</i>	Activated each time Navision has finished reading and processing a record within a <i>DataItem</i> .
<i>TransFooter</i>	An intermediate footer that can be used to print information at the bottom of a page before the end of a <i>DataItem</i> has been reached. Useful for printing an in-between sum at the bottom of each page before the final sum comes on the final page.
<i>GroupFooter</i>	Activated when Navision determines that a group in a <i>DataItem</i> has ended. It is useful to print the totals for a group of records in a <i>DataItem</i> such the total sales of salespersons in each job function group (See also <i>GroupTotalFields</i> and <i>GroupHeader</i> ). It is activated by a change in the value of the field/fields written into the <i>DataItem</i> property, <i>GroupTotalFields</i> . Be careful to sort the <i>DataItem</i> in the <i>DataItemTableView</i> according to the same field/fields you want grouped in the <i>GroupTotalFields</i> property.
<i>Footer</i>	Activated after the last record of a <i>DataItem</i> has been processed. Useful for printing report totals.

In our present example you do not need to use all of the section types. You do need a header to display the report title, date, user ID and page number. Select *Salesperson/Purchaser* as the *DataItem* and then select *Header*. Click OK.

Navision has already inserted two *Body* sections—one for each table in the *DataItem* list. Navision does this automatically but that does not mean every report requires *Body* sections. When you need only the report title and the complete totals from a report, include only the *Header* and *Footer* sections for your output; the other section can be erased.

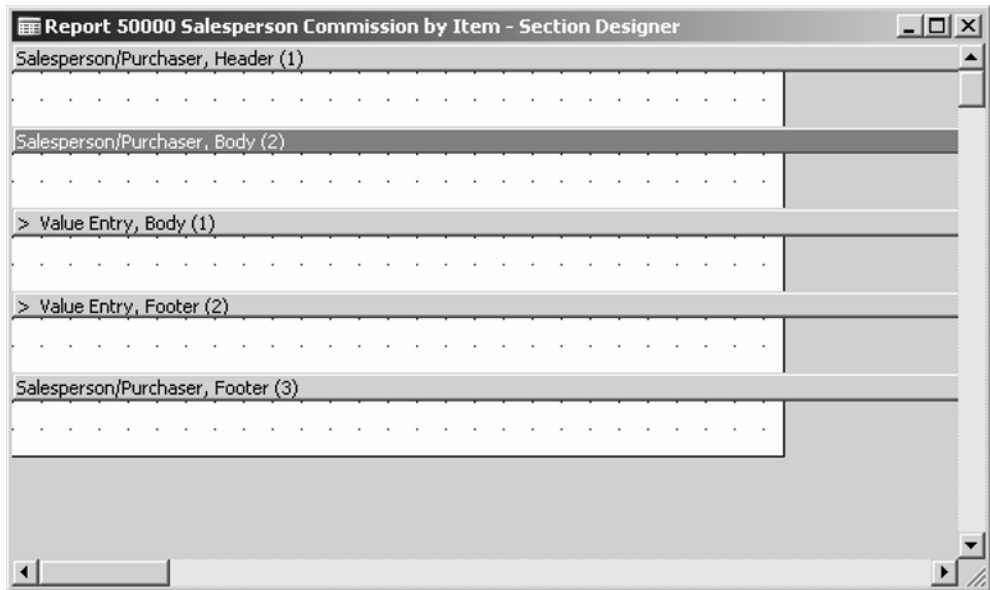
You will also need footers for both of your tables in this example. The first footer is necessary for displaying totals for each salesperson. A *Value Entry, Footer* is activated each time Navision comes to the final record in the *Value Entry* table for a given salesperson. The *Value Entry, Footer (2)* is therefore the proper section for a salesperson total. Go to the *Insert New Section* window by going to:

- **Edit > Insert New**

Select *Value Entry* from the list in the *DataItem* option. Now select *Footer* and click OK.

A second footer is necessary for displaying the total of the entire sales department. Select the *DataItem Salesperson/Purchaser* and then the section type, *Footer*. Click OK.

Your *Section Designer* window will now look like the following:



**Figure 6.16** *Section Designer* window

Each section is activated by a step in the data flow diagram. Think of each section as an event triggered output. With these sections you can control the print output at every step in the reporting process.

To get a feel for what Navision will do, you must always ask these questions when programming: “If X happens, what follows?” and “What will happen if ... ?” This is the basic logic Navision will always use. One way or the other, everything in the computer environment boils down to the statement, “If this happens then that must happen.” For example, try answering the following question: When a *Value Entry* footer has been activated, then what has happened? If you can answer this question for each section in the *Section Designer*, you have understood the results the sections will provide in the final report printout. The answer to the question is: A *Value Entry* header will be activated after the last match between a given *Salesperson/Purchaser* key

and any matching records in the *Value Entry* table have been processed.

If you understand the above information flow diagram you should now recognize a strong similarity between its basic layout and the logic behind it and the layout of what you have defined in the *Section Designer*. When you compare these two layouts you will see that they contain the same basic event flow.

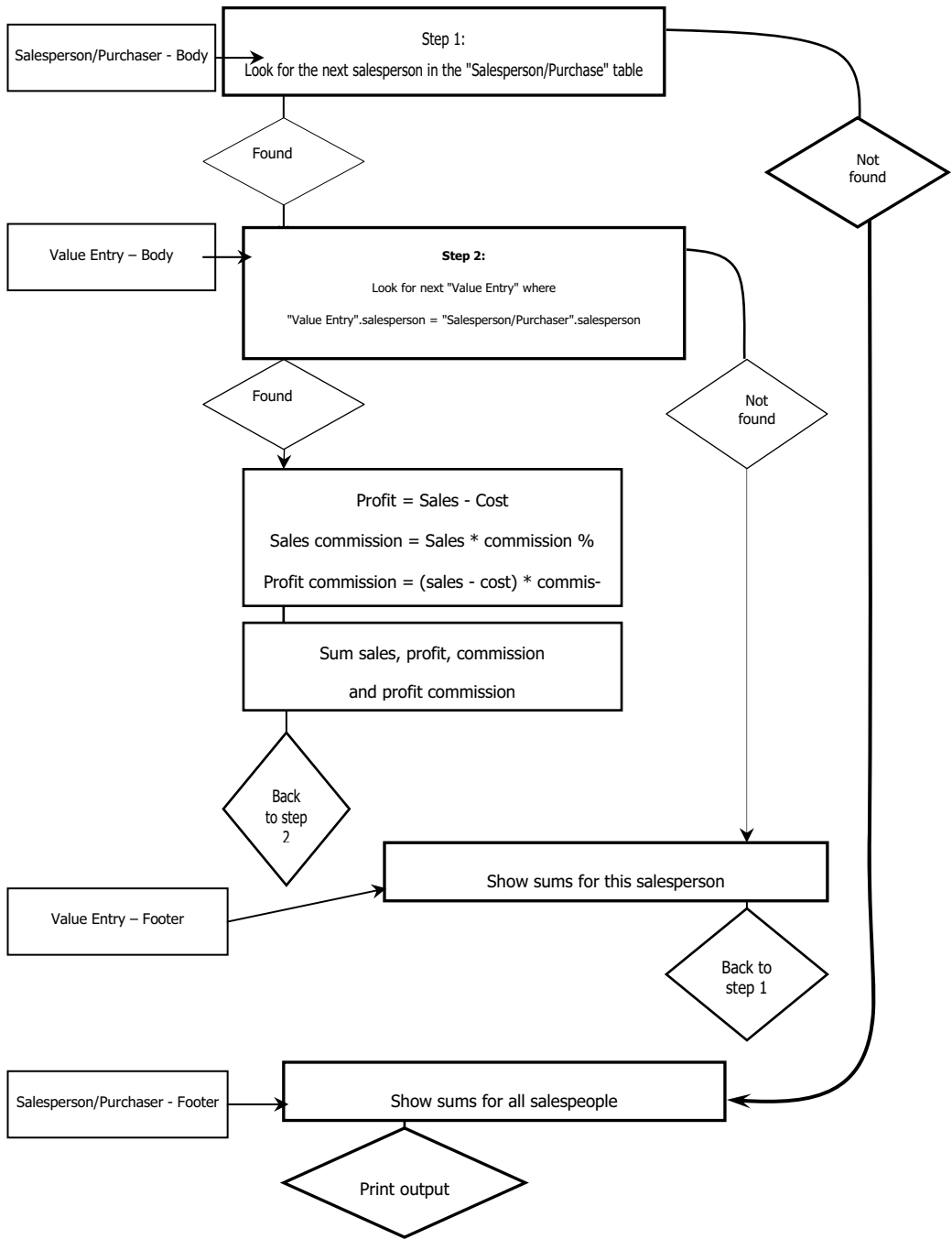
#### **6.5.4 Comparing the Section Designer With the Info-Flow Diagram**

By now you should see some similarities between the layout of the *Section Designer* and the structure of the information flow diagram discussed earlier. We will now compare the events of the two in the diagram found on the next page.

What is left out of the *Section Designer* layout are the found and not found events and the return events. These events relate to whether or not a match is found between the keys of the linked tables. These events, moving through table events, are managed automatically by Navision based on the table relations which we have already set up in the *DataItem* list and *DataItem properties* windows. This is the beauty of the Navision report designer: If you understand how to link tables in the *DataItem* windows, Navision automatically organizes the flow of information between the tables.

#### **6.5.5 Building Output Sections**

Now we will discuss how to construct the form of the report output—how the printout or print view will look.



## 6.5.5.1

**Salesperson Header**

Now we will begin entering some fields and text for Navision to display. In the *Section Designer* each section is associated with a *DataItem* and can display any information that exists within the records of the *DataItem* or parent *DataItem* which it is linked to. For example, within the *Body* section of the *Value Entry DataItem* you can display the field, *Sales Amount (actual)*, which is a part of it and you can also display the field, *Name*, from the *Salesperson/Purchaser DataItem*. However, you cannot do the opposite—display a field from the *Value Entry* table within a *Salesperson/Purchaser* section. This is because the *Salesperson/Purchaser* table is not a part of the *Value Entry DataItem*. With some practice and trial and error you will soon get the feel for these relations.

In the *Section Designer* you have access to the same design tools described in Ch. 4, “Creating a New Form.” Therefore, please review that section to see a list of all the tools that are available from the Toolbox. Remember that for every type of object you can insert into a form or report, there is a special set of properties. These properties can always be reached by selecting the object and then going to:

- **View > Properties**

First, fill out the *Salesperson/Purchaser, Header (1)* section. Enter a report title which can be a static text display. Open the Toolbox by going to:

- **View > Toolbox**

Now find the graphic symbol tool, *Label*, and click on it. Next, move your cursor into the top left area of the *Salesperson/Purchaser, Header (1)* section and double-click the left mouse button to drop the object into the report section. Now click on the *Label* object and go to:

- **View > Properties**

In the *Label* object properties list, enter the following values into the following properties:

Xpos	0
Ypos	0
Width	6000
Height	1692



...	
Caption	Sales Commission by Item
...	
HorzAlign	Left
VertAlign	Center
...	
FontName	Tahoma
FontSize	12

Next, you can create fields in the header that will have variable content as well as fields that show the ID of the user logged onto the Navision program that generated the report. You can display the date on which the report was processed as well as the page number of the report. To create fields that accommodate variable content, you must use the special object, *TextBox*.

Press ESC to return to the *Section Designer*. Click on the graphic tool, *TextBox*, from the Toolbox menu. Click into the *Salesperson/Purchaser, Header (1)* section and place the *TextBox* there. Click on the first *TextBox* and open its properties by going to:

- **View > Properties**

Go down through the list and enter the following values into the properties listed below:

Xpos	12600
Ypos	1269
Width	2400
Height	423
...	
HorzAlign	Left
VertAlign	Bottom
...	
FontName	Tahoma
FontSize	5
...	
SourceExpr	USERID

This object will now display the ID of the user who has logged into Navision and printed the report.

The values within the properties position and object size are measurements in increments of 1/100 of a millimeter.

Now add another *TextBox* object. In the *TextBox* object properties list enter the following values:

Xpos	15000
Ypos	1269
Width	900
Height	423
...	
HorzAlign	Center
VertAlign	Bottom
...	
FontName	Tahoma
FontSize	5
...	
SourceExpr	TODAY

This object will now display the system date when the report is printed.

Now add a third *TextBox* object. In the *TextBox* object properties list enter the following values:

Xpos	16800
Ypos	1269
Width	380
Height	423
...	
HorzAlign	Right
VertAlign	Bottom
...	
FontName	Tahoma
FontSize	5
...	
SourceExpr	CurrReport.PAGENO

This object will now display the page number on each page of the report printout. You will need a label for this object to clarify this page number. Go to the graphic symbol toolbox:

- **View > Toolbox**

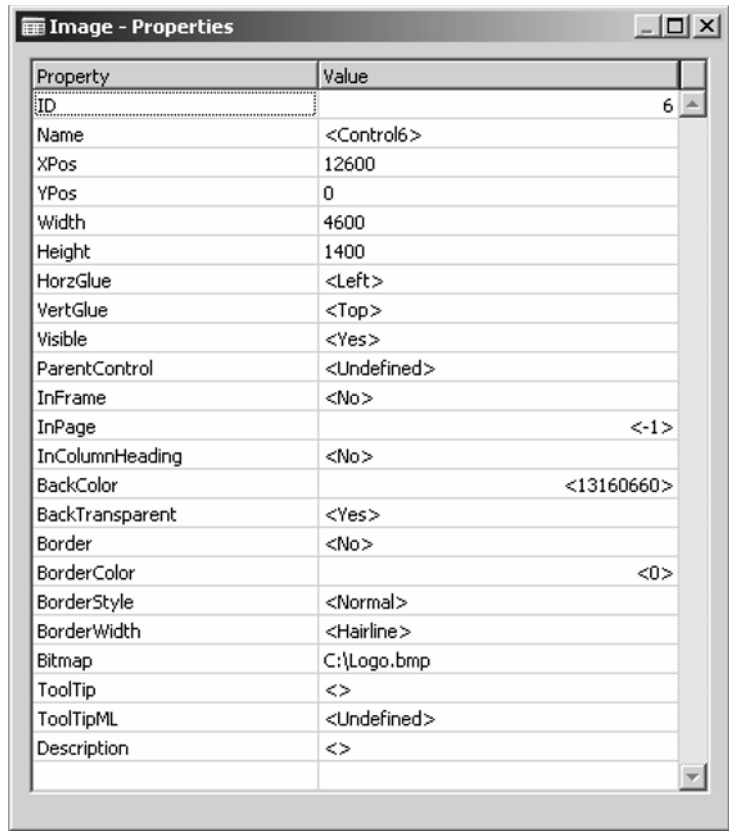
Click on the Label tool and enter the following values into this list of the *Label* properties:

Xpos	15900
Ypos	1269
Width	900
Height	423
...	
HorzAlign	Right
VertAlign	Bottom
...	
Caption	Page
...	
FontName	Tahoma
FontSize	5

This will display the word “Page” before the page number.

Now you can add a company logo to the report header. Open the Toolbox again and this time select the Image tool. Click into the header area and open the properties of this object. The following window will appear: (See Figure 6.17 below)

These are the properties for the graphic display object—the most important of which is the property, *Bitmap*. Here you can enter the location of a bitmap file. The moment you write a file location and press ENTER Navision will attempt to load a bitmap from the location you have specified. The graphic file must be no greater than 32 kilobytes and must be saved in the bitmap data format. 32 kilobytes is not a large storage space so you must either choose a small version of your logo or use several *Image* objects together to form a composite display. Normally, you can load a bitmap object that is less than 750 square millimeters.



**Figure 6.17** Image - Properties window

For this example we will load a black and white logo that is 584 square millimetres. The following values and location will be entered in the following properties for this example:

Xpos	12600
Ypos	0
Width	4600
Height	1400
...	
Bitmap	C:\Logo.bmp

Now press ESC once to return to the *Section Designer* window. Insert a horizontal line to separate the header visually from the rest of the report. Now open the Toolbox and click on the Shape

tool. Click into the header section to drop the *Shape* object. Next, open the *Shape* object properties by going to:

- **View > Properties**

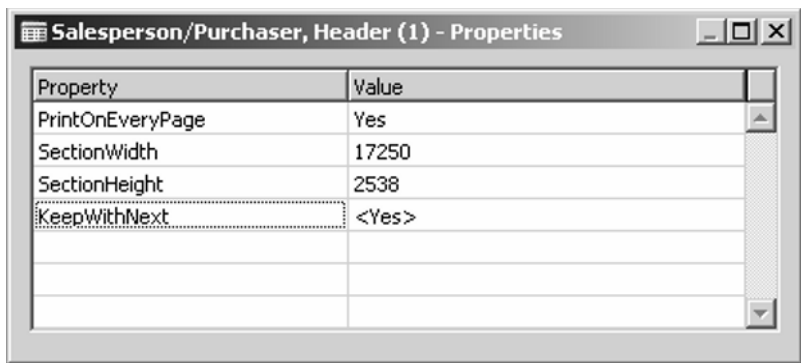
Enter the following values into the following properties:

Xpos	0
Ypos	2115
Width	17250
Height	423
...	
BorderWidth	Hairline
ShapeStyle	Horzline

The last thing to accomplish in this report header example is to instruct Navision to print this header at the top of every page. To do this, click on the horizontal gray, bar with the text, *Salesperson/Purchaser, Header (1)*, written on it. This selects the entire section as a whole. Next go to:

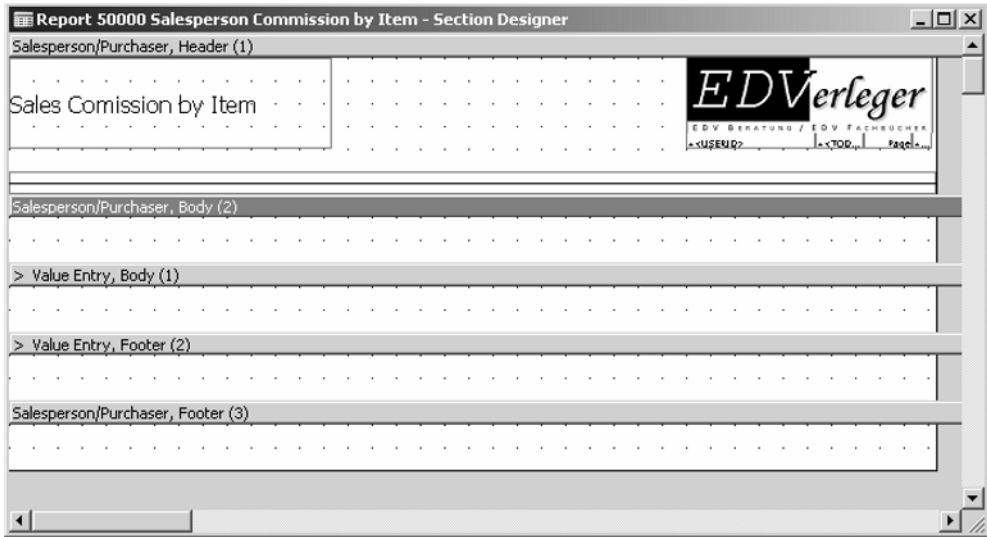
- **View > Properties**

Enter the following values into the following properties:



**Figure 6.18** *Salesperson/Purchaser Header (1) - Properties* window

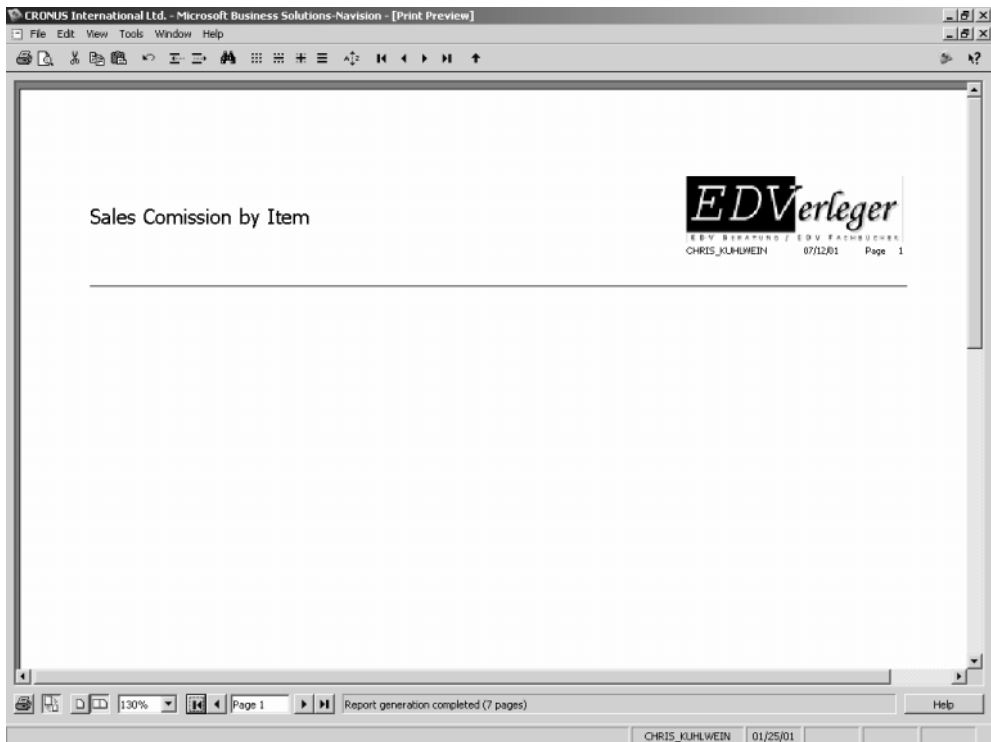
Press ESC to return to the *Section Designer* window. Now you will see the following window:



**Figure 6.19** Section Designer window

Now you can test the header. Press ESC twice and answer "Yes" when Navision asks if you want to save and compile the report.

You are now back in the *Object Designer* window. Click on the *Salesperson Commission by Item* report and then click on the Run button at the page bottom. Next, click the Preview button at the window bottom. The report will now create pages. When the report is finished and *Report generation completed* appears at the bottom of the window, scroll through the various pages of the report. This view of the report will appear:



**Figure 6.20** Complete report header, “Sales Commission by Item”

Note that this header appears at the top of every page. Each page contains the information that you defined as well as a page number, beginning with Page 1 and then so forth.

### 6.5.5.2 Salesperson/Purchaser Report: The Body Section

Now you can add information about each salesperson in your report. You can do this by entering variables into the *Salesperson/Purchaser* section which is activated each time a new salesperson record is read.

Press ESC to leave the report view and then click on the Design button for the *Salesperson Commission by Item* report. Enter the *Section Designer* by going to:

- **View > Sections**

Now you must enter fields that display relevant information about the salesperson. This information can either be displayed in a section that is part of the *Salesperson/Purchaser DataItem* or

a section that is part of linked *DataItems* that have a child relationship to the *Salesperson/Purchaser DataItem*.

If you place variables from the *Salesperson/Purchaser DataItem* into a *Value Entry Body* section, the salesperson information repeats for every line in the *Value Entry DataItem*. This repetition is not particularly useful, so instead use the *Salesperson/Purchaser Body* section to display salesperson information.

Now open the *Field Menu* by going to:

- **View > Field Menu**

This window contains all the fields that exist in the table of the currently selected *DataItem*. Click into the section, *Salesperson/Purchaser, Body (2)*, and then double-click the following variables with field names in the *Field Menu*: *Code*, *Name*, *Commission %*. Now when you double-click into the section, Navision automatically places two objects into the section. The top object is a *Label* object which displays the name of the field you have selected. The second object is a *TextBox* object containing the variable that you have selected. The *TextBox* object and its *Label* object are connected. Therefore, when you move the *TextBox*, the *Label* is automatically dragged with it.

Display the *Label* object in the *Header* section. It is desirable to repeat the *Label* objects only once per page as opposed to each time there is a new salesperson. Place the *Label* objects into the header section, *Salesperson/Purchaser, Header (1)* and delete the *Label, Name*. One title, with the text, *Salesperson*, is clear enough for understanding both *Code* and *Name*.

Next, open the properties of the *Code TextBox* object and change the following properties to the following values:

Xpos	0
Ypos	0
Width	1200
Height	423
...	
Caption	Salesperson
...	
HorzAlign	Left
...	
FontName	Tahoma



FontSize	7
FontBold	Yes

Now open the property list for the *Name TextBox* and enter the following values for the following properties:

Xpos	1300
Ypos	0
Width	
Height	423
...	
HorzAlign	Left
...	
FontName	Tahoma
FontSize	7
FontBold	Yes

Next, open the property list for the *Commission % TextBox* and enter the following values for the following properties:

Xpos	5400
Ypos	0
Width	1800
Height	423
...	
Caption	Commission %
...	
HorzAlign	Center
...	
FontName	Tahoma
FontSize	7
FontBold	Yes
...	
DecimalPlaces	0:2

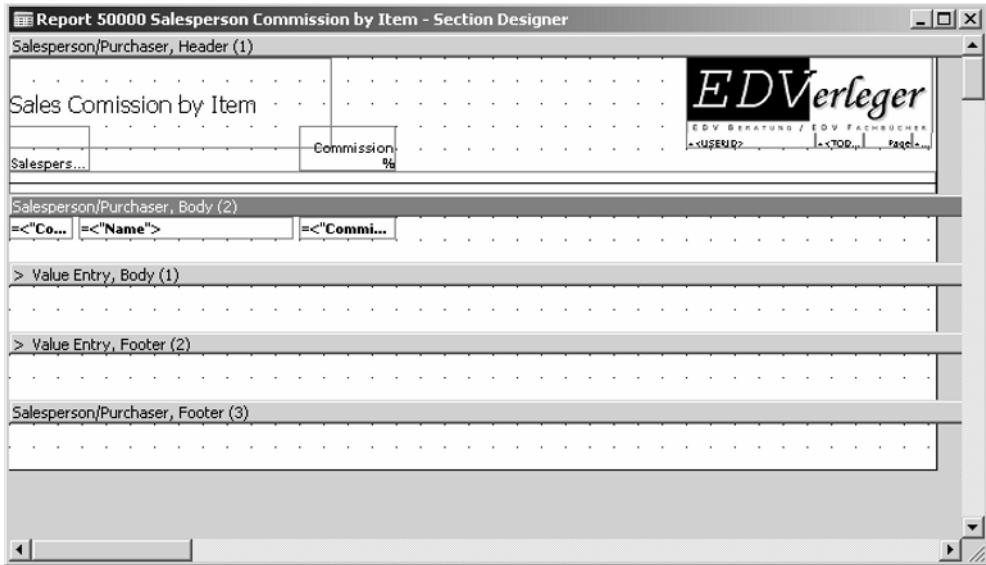
The property, *DecimalPlaces*, controls the values format. The first number tells Navision how many places must be shown after the decimal point and the second number after the colon tells Navision how many places after the decimal point are allowed if the value extends beyond the decimal point. For example, consider setting this property to 2:2 when displaying money amounts. Navision will, therefore, always show exactly two places after the decimal point even if there are no cents in the money amount.

Creating a report is about making it easy to view information. Likewise, this information should be useful in making business decisions. Therefore, simplicity and clarity are the principles to follow when designing a report. If a piece of information is not absolutely necessary, it should be left out. “Less is more” should be the motto of every report designer. Remember that your report should answer questions rather than create them. Your report should inspire the reader to take action and make decisions. The art of business information design is not an art for which programmers are known, however, the best programmers know and understand the needs of their audience.

Now click on the Font graphic tool symbol or go to:

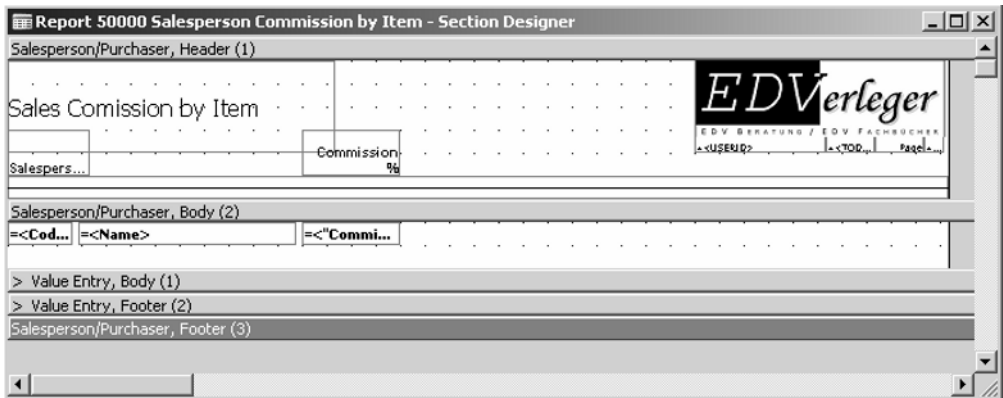
- **View > Font**

Click on each of the *Label* and *TextBox* objects and change the font to **Tahoma**. Next, align them so that they appear as in the window below:



**Figure 6.21a** *Section Designer* window

To minimize the space of the printout, reduce the size of the unused sections before test printing. To do so, put your cursor directly on the horizontal gray bars of the sections not yet used, such as the *Value Entry, Footer (2)* section, and move it upward to reduce the white space below it. (Note Figure 7.21b below)



**Figure 6.21b** *Section Designer* with collapsed, unused sections

Now compile, save and test your report. Return to the *DataItem* page by pressing ESC and then press F11. If no error message appears it means there are no syntax errors in the report program. Next, save the report by pressing CTRL+S. Now click the Run button and then Preview. The following report view window will appear: (See Figure 6.22)

The screenshot shows a report preview window for 'Sales Commission by Item'. The report header includes the title 'Sales Commission by Item' and the logo for 'EDV Verleger' with contact information: 'EDV BERATERS / EDV FACHBUCHER', 'CHRIS\_KUHLWEIN', '07/12/01', and 'Page 1'. The main body of the report is a table with two columns: 'Salesperson' and 'Commission %'. The table lists eight salespersons with their respective commission percentages. The footer of the report indicates 'Report generation completed (1 pages)' and 'Page 1'.

Salesperson	Commission %
AH Annette Hill	0
BD Bart Duncan	0
DC Debra L. Core	0
JR John Roberts	5
LM Linda Martin	0
MD Mary A. Dempsey	5
PS Peter Sadow	5
RL Richard Lum	0

**Figure 6.22** “Sales Commission by Item” header and completed body sections

Here is the display of information from the first *DataItem* in the report which is printed after Navision has processed each record in this *DataItem*.

### 6.5.5.3

#### Value Entry Body Section

Now we will construct output that Navision can print each time it finds information about one of these salespersons within the second *DataItem*, the *Value Entry DataItem*, which has a child relation to the *Salesperson/Purchaser DataItem*.

Press ESC and then click the Design button to reenter the internal structure of the report. Go to:

- **View > Sections**

Open the report *Section Designer*. Open the *Field Menu* by going to:

- **View > Field Menu**

Click on the section, *Value Entry, Body (1)*. Select the fields *Item No.* and *Sales Amount (Actual)* from the *Field Menu*. Drag these fields into the *Value Entry* section and place them just right and below the fields that are in the *Salesperson/Purchaser, Body (2)* section. Double-click to insert the objects. Now move the *Label* objects up to the *Salesperson/Purchaser, Header (1)* section.

Now open the property list for the *Item No. TextBox* and enter the following values into the following properties:

Xpos	7350
Ypos	0
Width	1500
Height	423
...	
Caption	Item
...	
HorzAlign	Right
...	
FontName	Tahoma
FontSize	7
FontBold	No

Now open the *Item No. Label* object and set the following properties to these values:

Xpos	7350
Ypos	1269
Width	1500
Height	846
...	
HorzAlign	Right

...	
FontName	Tahoma
FontSize	7
FontBold	Yes

Next, open the *Sales Amount (Actual) TextBox* and enter the following values into the following properties:

Xpos	9000
Ypos	0
Width	1800
Height	423
...	
Caption	Sales Amount
...	
HorzAlign	Right
...	
FontName	Tahoma
FontSize	7
FontBold	No
...	
DecimalPlaces	2:2

Now open the property list of the *Sales Amount Label* object and change the following properties to the following values:

Xpos	9000
Ypos	1269
Width	1800
Height	846
...	
HorzAlign	Right
...	
FontName	Tahoma
FontSize	7
FontBold	Yes

Open the next *TextBox*, *Cost Amount (Actual)* and enter:

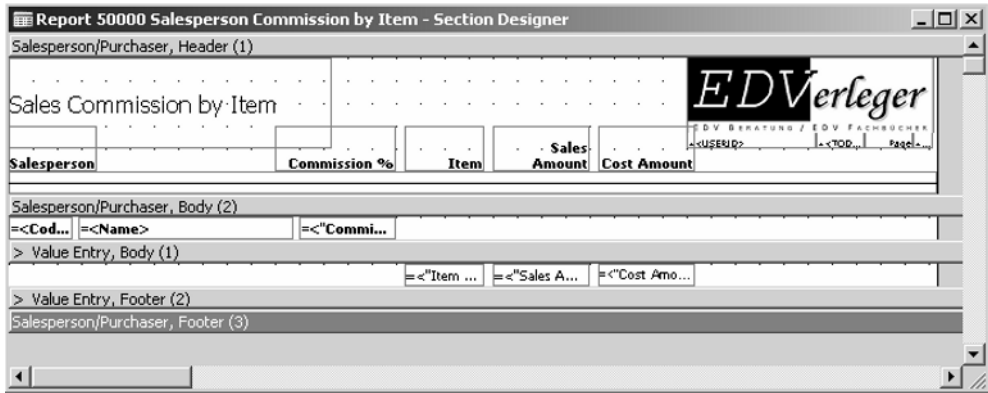
Xpos	10950
Ypos	0
Width	1800
Height	423
...	
Caption	Cost Amount
...	
HorzAlign	Right
...	
FontName	Tahoma
FontSize	7
FontBold	No
...	
DecimalPlaces	2:2

Now open the *Label* object, *Cost Amount*, and modify the values to:

Xpos	10950
Ypos	1269
Width	1800
Height	846
...	
HorzAlign	Right
...	
FontName	Tahoma
FontSize	7
FontBold	Yes

Press ESC to return to the *Section Designer*. The *Section Designer* window should look like the following:

Before making a test print, push the sections together to minimize the unused white space as seen below.



**Figure 6.23** Section Designer window

Now compile, save and test your report. Be careful not to compile your report while in the *Section Designer* window as this may cause a Report Designer failure and cause you to lose your changes. Instead, leave the *Section Designer* window and return to the *DataItem* window before pressing F11 to compile and/or save.

Press ESC twice to return to the *DataItem* window and answer "Yes" to compile and save the report. Now click the Run button in the *Object Designer* window after selecting the *Salesperson Commission by Item* report. Click on the Preview button. The following report view window will appear:



CRONUS International Ltd. - Microsoft Business Solutions-Navision - [Print Preview]

File Edit View Tools Window Help

**Sales Commission by Item**

EDV Verlag  
EDV BEATRICH J. EDV FACHBUCHER  
CHRIS\_KUHLWEIN 07/12/01 Page 1

Salesperson	Commission %	Item	Sales Amount	Cost Amount
AH Annette Hill	0			
BD Bart Duncan	0			
DC Debra L. Core	0			
JR John Roberts	5			
		1968-S	0.00	0.00
		1960-S	0.00	0.00
		1976-W	0.00	0.00
		70011	0.00	0.00
		1968-S	0.00	0.00
		1960-S	0.00	0.00
		1976-W	0.00	0.00
		1972-S	0.00	0.00
		1968-S	0.00	0.00
		1980-S	0.00	0.00
		1920-S	0.00	0.00
		1900-S	0.00	0.00
		1996-S	0.00	0.00
		1896-S	0.00	0.00
		1906-S	0.00	0.00
		1972-S	739.80	-576.60
		1968-S	493.20	-384.40
		1980-S	369.90	-288.30
		1920-S	840.80	-658.00

Report generation completed (4 pages)

Report generation status  
CHRIS\_KUHLWEIN 07/12/01

**Figure 6.24** Report *Print Preview*

As you can see the report includes a lot of information that is unnecessary for the purposes of this report. For example, it is unnecessary to display salespersons who have had no sales, nor is it necessary to show product movements where there are no sales or where sales equal zero.

#### 6.5.5.4

#### Eliminating Unwanted Report Lines

To eliminate irrelevant information you must control which records Navision accesses in the *DataItems*. You could improve the report by setting a filter on the product movements in the *Value Entry DataItem* so that Navision only accesses positions that are sales type and not zero. You could also instruct Navision that when no information is found in the child table, it should not print the information from the parent table. These two controls should clean up the report considerably.

Press ESC to leave the report view and then click on the Design button to reenter the internal structure of the report. Now select

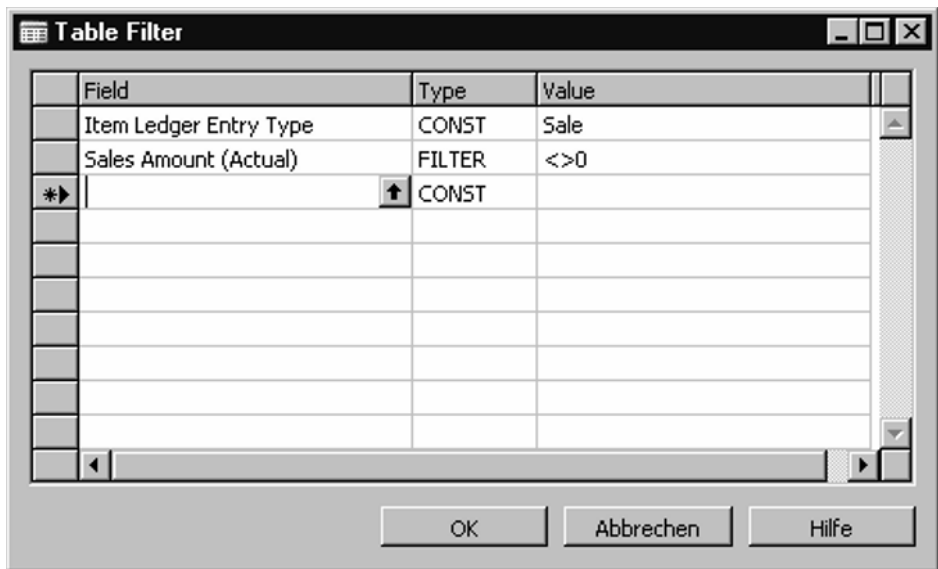
the first *DataItem*, *Salesperson/Purchaser*, and then open its properties by going to:

- **View > Properties**

Now enter the following value into the following property:

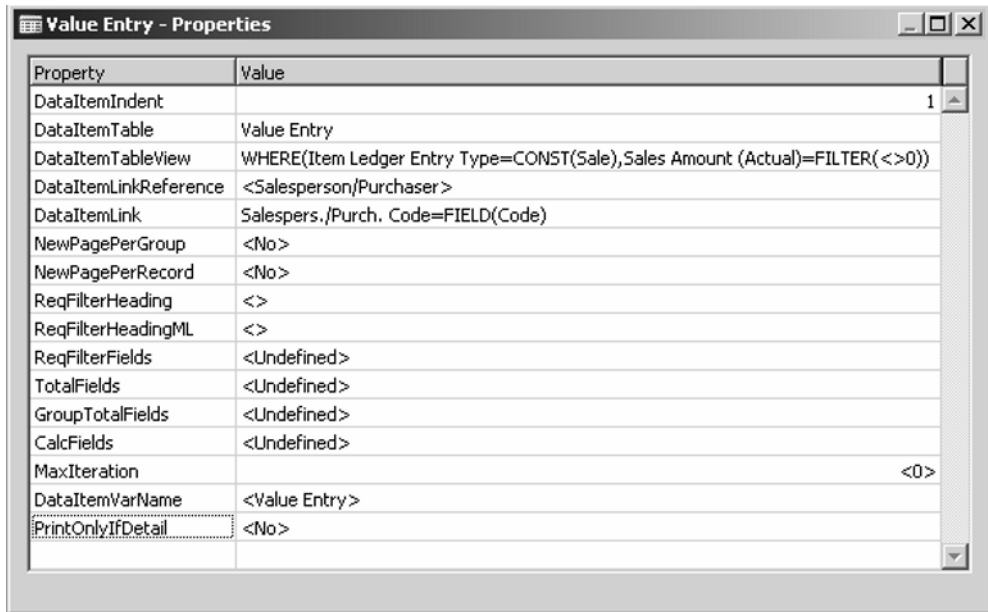
...	
PrintOnlyIfDetail	Yes

Next, press ESC and then open the properties of the second *DataItem*, *Value Entry*. Click onto the property, *DataItem-TableView*, then click on the assist symbol that appears in the right of the column. Next, click on the assist in the variable, *TableFilter*, opening the *TableFilter* window. Fill in the following filters in the following fields as shown in the window below:



**Figure 6.25** *Table Filter* window

These *Table Filters* will stop Navision from accessing records that are not sales movements and that are zero. Click OK and then OK again. The *Value Entry* properties window will now look like the following:



**Figure 6.26** *Value Entry - Properties DataItem* window

Press ESC twice to save and leave the report. Now click on the Run button in the *Object Designer* after selecting the *Salesperson Commission by Item* report. Now click on the Preview button. The following report view window will appear:

CRONUS International Ltd. - Microsoft Business Solutions-Navision - [Print Preview]

File Edit View Tools Window Help

Sales Commission by Item

EDV Verlag  
EDV BERATUNG / EDV FACHSCHAFT  
CHRIS\_KUHLWEIN 07/12/01 Page 1

Salesperson	Commission %	Item	Sales Amount	Cost Amount
JR	John Roberts			5
		1972-S	739.80	-578.60
		1968-S	493.20	-384.40
		1980-S	369.90	-288.30
		1920-S	840.80	-656.00
		1900-S	750.60	-585.00
		1996-S	906.70	-707.20
		1968-S	246.60	-192.20
		1968-S	123.30	-96.10
		1968-S	123.30	-96.10
		1960-S	375.30	-292.50
		1960-S	250.20	-195.00
		1960-S	250.20	-195.00
		1976-W	460.98	-301.20
		1976-W	230.49	-150.60
		1976-W	460.98	-301.20
		70011	61.45	-38.90
		1896-S	649.40	-506.60
		1906-S	281.40	-219.50

Report generation completed (2 pages)

CHRIS\_KUHLWEIN 01/25/01

**Figure 6.27** Print Preview of cleanedup version of report

As you can see the report has improved; it shows only information that is relevant to sales.

An additional advantage to this technique is that it may stop the possibility of an intrusion of a nasty program bug because you have blocked Navision's access to records where the sales amount is zero. Doing so eliminates the possibility of a division by zero error in any formulas where the sales are used as a divisor.

## 6.5.6 Creating Original Information Within the Report

Next, you must learn to create information in the report even when it does not exist in the tables. Here is where you begin to see the real merit of the report object.

### 6.5.6.1 Creating Totals and Calculated Variables Within the Report

Next, you must work on the output. It does not exist in any table data, but must be calculated within the report.

The first calculated variable you should construct is the contribution margin (Profit). In this report it is defined as “the sales amount plus the negative cost amount.” Now perform this calculation for each line of the *Value Entry DataItem*.

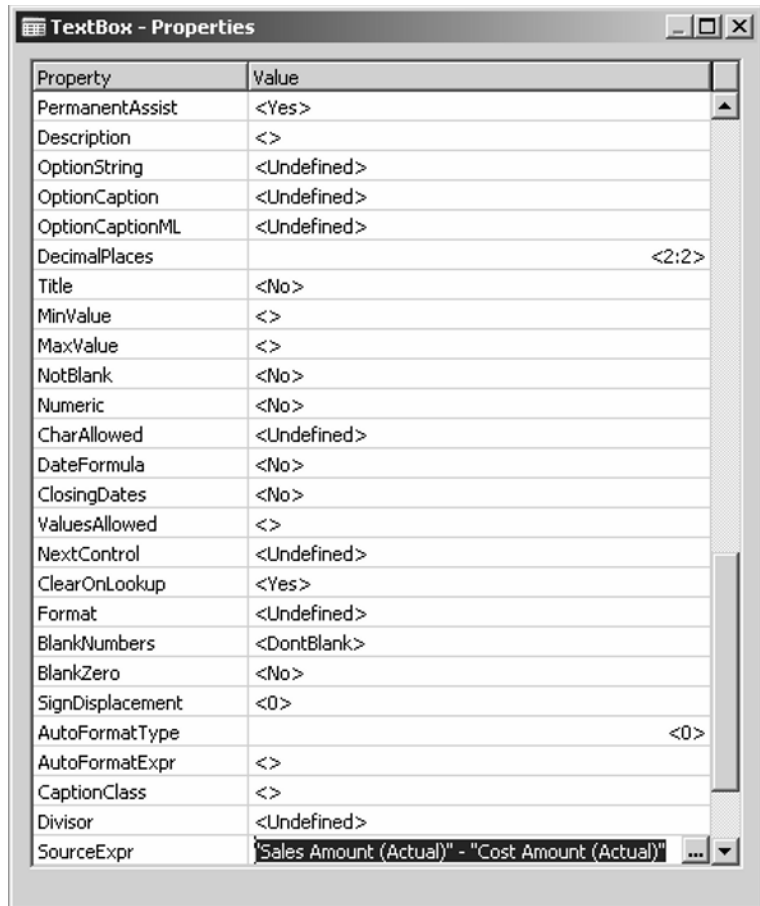
Navision allows you to write a variable or field into the *SourceExpr* property of a *TextBox* object, or define a simple formula there. Using a *TextBox* object, write the formula, “sales plus the negative cost” within it. Click on the *TextBox*, *Cost Amount (Actual)* and go into its properties window by going to:

- **View > Properties**

Now scroll down to the *SourceExpr* property. You will see the following window (See Figure 6.28 below):

Within the following properties, enter the following values:

Xpos	10950
Ypos	0
Width	1800
Height	423
...	
Caption	Profit
...	
HorzAlign	Right
...	
FontName	Tahoma
...	
FontSize	7
FontBold	No
...	
DecimalPlaces	2:2
...	
SourceExpr	"Sales Amount (Actual)" + "Cost Amount (Actual)"



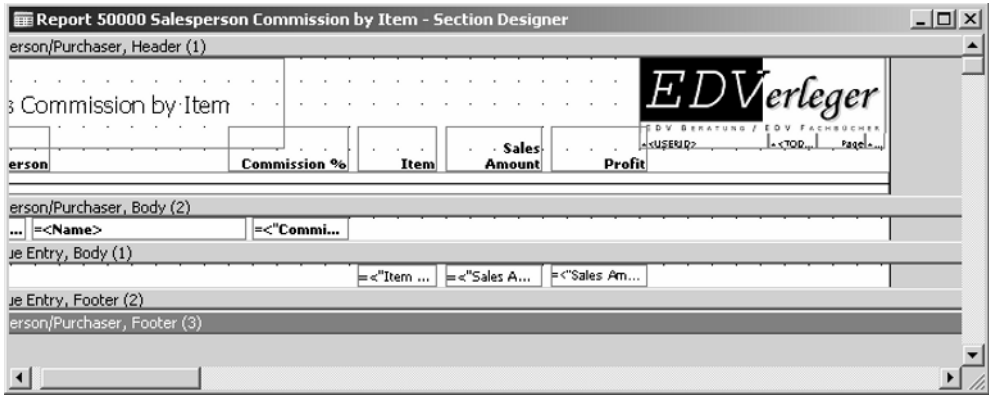
**Figure 6.28** *TextBox - Properties* window

Next, open the property list for the *Profit Label* object and enter the following values in the following properties:

Xpos	10950
Ypos	1269
Width	1800
Height	846
...	
Caption	Profit
HorzAlign	Right
...	

FontName	Tahoma
FontSize	7
FontBold	Yes

Press ENTER and then ESC. The *Section Designer* window will now look like the following:



**Figure 6.29** *Section Designer* window

Following the principle that a report should answer questions rather than create them, you should include helpful information for the reader. This report should clearly indicate to the salespeople which are the best products to sell. Therefore, you should include a factor that compares the performance of the various products. Create a column that shows the percentage of profit contribution for each product. To do this, enter a new *TextBox* that contains the following formula in the *SourceExpr* property:

...	
SourceExpr	("Sales Amount (Actual)" + "Cost Amount (Actual)") / "Sales Amount (Actual)" * 100

Open the Toolbox by going to:

- **View > Toolbox**

Click on the *TextBox* tool and immediately click on the *Add Label* tool. Now insert these objects into the Value *Entry, Body (1)*

section with a double-click. Move the *Label* object upward into the *Salesperson/Purchaser, Header (1)* section. Now open the property list of the new *TextBox* object and enter the following values into the following properties:

Xpos	11850
Ypos	0
Width	1500
Height	423
...	
Caption	Profit %
...	
HorzAlign	Center
...	
FontName	Tahoma
...	
FontSize	7
FontBold	No
...	
DecimalPlaces	0:0
...	
SourceExpr	("Sales Amount (Actual)" + "Cost Amount (Actual)") / "Sales Amount (Actual)" * 100

Open the property list of the *Profit % Label* box and set the following properties to the following values:

Xpos	11850
Ypos	1269
Width	1500
Height	846
...	
HorzAlign	Center
...	
FontName	Tahoma
FontSize	7
FontBold	Yes



Press ESC three times to return to the *DataItem* window and activate the save window. Click “Yes” to compile and save the report. Now click on the Run button and then the Preview button to view the report output. The following report *Print Preview* window will appear:

CRONUS International Ltd. - Microsoft Business Solutions-Navision - [Print Preview]

File Edit View Tools Window Help

**Sales Commission by Item**

EDV Verlag  
EDV BEATRICE / EDV FACHBUCHER  
CHRIS\_KUHLWEIN 07/12/01 Page 1

Salesperson	Commission %	Item	Sales Amount	Profit	Profit %
JR	John Roberts	5			
		1972-S	739.80	163.20	22
		1968-S	493.20	106.80	22
		1980-S	369.90	81.60	22
		1920-S	840.80	184.80	22
		1900-S	750.60	165.60	22
		1996-S	906.70	199.50	22
		1968-S	246.60	54.40	22
		1968-S	123.30	27.20	22
		1968-S	123.30	27.20	22
		1960-S	375.30	82.80	22
		1960-S	250.20	55.20	22
		1960-S	250.20	55.20	22
		1976-W	460.98	159.78	35
		1976-W	230.49	79.89	35
		1976-W	460.98	159.78	35
		70011	61.45	24.55	40
		1896-S	649.40	142.80	22
		1906-S	281.40	61.90	22
		1896-S	-649.40	-142.80	22
		1952-W	134.73	41.13	31

Report generation completed (2 pages)

CHRIS\_KUHLWEIN 01/2/...

**Figure 6.30** Report *Print Preview* with profit information

Now add more calculated *TextBox* objects with *Label* objects attached to them. The variables you should create next include *Sales Com.* and *Profit Com.* These variables do not exist in any table within Navision so you cannot add them from the *Field Menu*. Instead you must create them from scratch.

Open the Toolbox with:

- **View > Toolbox**

First click on the *TextBox* tool and then immediately click on the *Add Label* tool. This will create an empty but grouped *TextBox* and *Label* object. Place these in the *Value Entry, Body (1)* section and double-click. Now move the *Label* object up into the *Salesperson/Purchaser, Header (1)* section. Next, open the properties

of the new but empty *TextBox*. Enter the following values into the following properties:

Xpos	13800
Ypos	0
Width	1650
Height	423
...	
Caption	Sales Com.
...	
HorzAlign	Right
...	
FontName	Tahoma
...	
SourceExpr	"Sales Amount (Actual)" * ("Salesperson/Purchaser"."Commission %"/100)

Enter the properties of this *TextBox* object *Label* object and enter the following values to the following properties:

Xpos	13800
Ypos	1269
Width	1650
Height	423
...	
Caption	Sales Com.
...	
HorzAlign	Right
...	
FontName	Tahoma
FontSize	7
FontBold	Yes

Now you must create another *TextBox* and connected *Label* object for the *Profit Com.* variable. Click again on the *TextBox* and then on Add Label from within the Toolbox and place these in

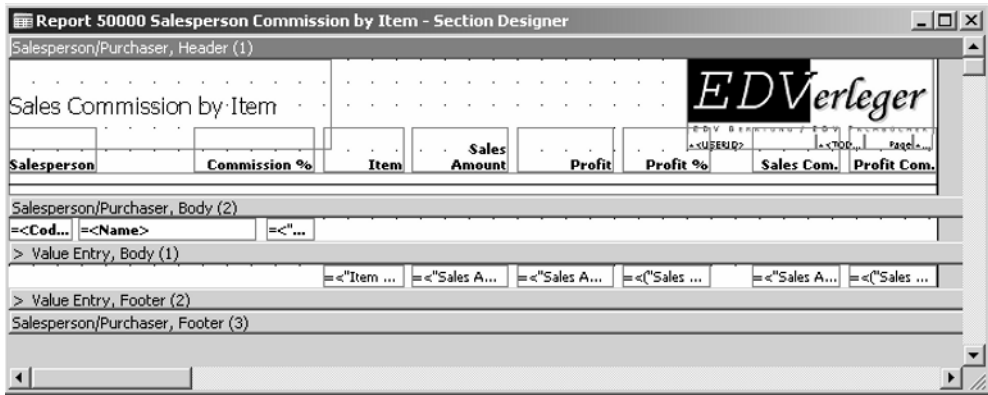
the *Value Entry, Body (1)* section. Open the properties of the *TextBox* and enter the following values into the following properties:

Xpos	15600
Ypos	0
Width	1650
Height	423
...	
Caption	Profit Com.
...	
HorzAlign	Right
...	
FontName	Tahoma
...	
SourceExpr	("Sales Amount (Actual)" + "Cost Amount (Actual)") * ("Salesperson/Purchaser"."Commission %"/100)

Now open the properties of the *Label* object and enter these values:

Xpos	15600
Ypos	0
Width	1650
Height	423
...	
Caption	Profit Com.
...	
HorzAlign	Right
...	
FontName	Tahoma
FontSize	7
FontBold	Yes

Now leave the *Label* object properties. The *Section Designer* window will now look like the following:



**Figure 6.31** *Section Designer* window

As you can see, it is possible to define complex expressions in the *SourceExpr* property. Normally, you only need to use a *TextBox* object for displaying a variable or a table field, however, at other times you may need to perform calculations or output formatting here. Complex expressions—like those above—must follow the syntax conventions of what is known as C/AL Code. C/AL Code is the development programming language for Navision, which we will discuss in more detail in the next chapter. For the moment, we will make only a few points that directly relate to the use of C/AL Code in the *SourceExpr* property.

When defining a complex expression within the *SourceExpr* property of a *TextBox* object, you must obey the following conventions:

1. Variable names should be enclosed within quotation marks (“ ”) if they contain special characters or spaces. For example:

...	
SourceExpr	"Strange Amount (Actual)"

2. *TextBox* objects can display static text. Text must be enclosed by apostrophes ( ' ). If you wish to display sev-

eral separate components of text, each must be separated with a plus symbol (+). For example:

...	
SourceExpr	"Your Profit ' + 'or' + ' or our Loss'

3. Mathematical operations can be done on variables and constants. Use the following operators for your calculations:

\* **Multiplication**

/ **Division**

- **Substraction**

+ **Addition**

( ) **Grouping**

Be careful when you using division that your dividend is not zero. Division by zero will always cause a program to fail. Note the following example:

...	
SourceExpr	("Sales Amount" + Cost) * ("Commission %" / 100)

4. If using fields in your *SourceExpr* that do not exist within the *DataItem* of the section, you must always write not only the field name but also the *DataItem* name followed by a period (.). For example:

...	
SourceExpr	"Salesperson/Purchaser"."Name"

You may only use fields that are either within the sections *DataItem*, fields or part of a parent linked *DataItem* or so-called *Global* fields which we will discuss later.

5. The *Data Type* of expression components must be compatible with one another. For example, an expression like Sales + TODAY will return an error because Navision cannot add a decimal value to a date. If you want to include a decimal, integer value or a date with text, you must convert each component into text. You can

easily do this by enclosing each expression component within parenthesis ( ) and writing the code, FORMAT before the parentheses. Next, you must connect the components with a plus sign (+). For example:

...	
SoucreExpr	'Sales are ' + FORMAT("Sales Amount") + ' on ' + FORMAT(TODAY)

### 6.5.6.2

#### Creating Totals

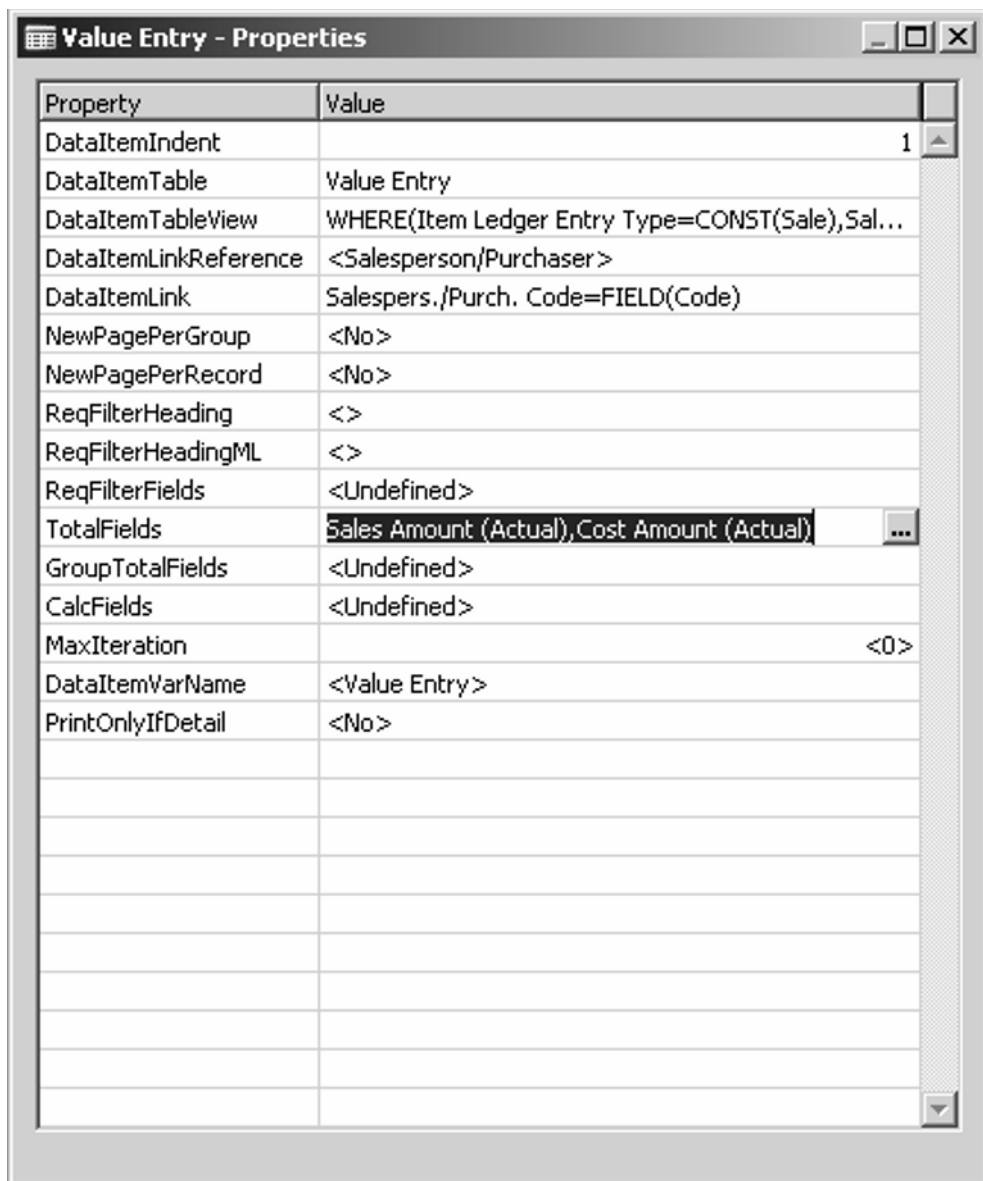
Next we will create and display the section and report totals. To obtain totals for each salesperson, you must instruct Navision to accumulate values each time it reads them from the *Value Entry DataItem*. This means you must define this accumulation within the *Value Entry DataItem* object itself.

Creating totals for the entire report is more difficult because this information does not exist as a mere accumulation of the fields of either *DataItem*. To achieve totals for the entire report, you must use the more advanced and powerful tools of C/AL Code and *Global* variables.

#### 6.5.6.2.1

#### End Totals Per Salesperson

Let us first create and display the totals for each salesperson. Press ESC to leave the *Section Designer* and open the properties of the *Value Entry DataItem*. The following *DataItem* property list will appear:



**Figure 6.32** Value Entry - Properties for DataItem

Select the property, *TotalFields*. Here you can enter the field or field names that you want Navision to accumulate for the *DataItem*. Once you have entered a field(s) then the value of these field(s) will be summed each time Navision reads a record in the *DataItem*. Therefore, enter the fields *Sales Amount (Ac-*

*tual*) and *Cost Amount (Actual)* into the *TotalFields* property of the *Value Entry DataItem*.

Now that you have instructed Navision to accumulate these fields in this *DataItem*, you can go into the *Section Designer* and determine how this new information will be displayed in the report output. Press ESC to leave the *DataItem* properties and then go to:

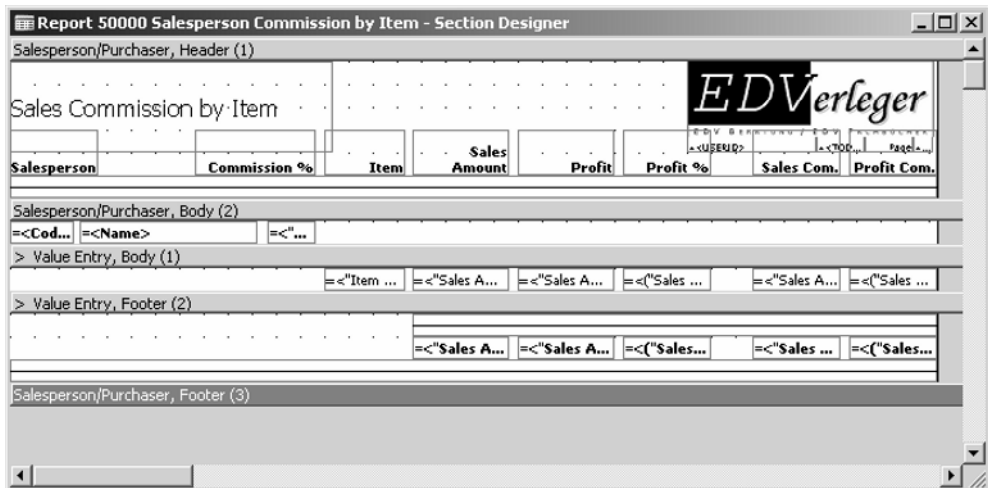
- **View > Sections**

Next, select all the *TextBox* objects in the *Value Entry, Body (1)* section except the *Item No.* object. Select them all at once by holding down the left button on your mouse while circling the desired objects. Once they are selected, click the right mouse button and select Copy from the menu. Now Paste them into the *Value Entry, Footer (2)* section by clicking on the right mouse button and selecting Paste from the menu.

Open the Toolbox by going to:

- **View > Toolbox**

Select the Shape tool. Insert two *Shape* objects into the *Value Entry, Footer (2)* section and then go into their properties lists. In the property, *ShapeStyle*, enter, *HorzLine*. Situate one horizontal line above and one below the *TextBox* objects. Shorten the line above the *TextBox* objects. Now press ESC. Your *Section Designer* window will now look like the following:



**Figure 6.33** Section Designer window



Press ESC twice and then compile and save the report. Next, click the Run button and then the Preview button. The following report view window will appear:

**Sales Commission by Item**

EDV Verlag  
CHRIS\_KUHLWEIN 07/12/01 Page 2

Salesperson	Commission %	Item	Sales Amount	Profit	Profit %	Sales Com.	Profit Com.
		70011	361.50	177.00	49	18.08	8.85
		8908-W	342.60	342.60	100	17.13	17.13
		8916-W	374.20	374.20	100	18.71	18.71
		8924-W	346.30	346.30	100	17.32	17.32
		8916-W	187.10	187.10	100	9.36	9.36
		8924-W	346.30	346.30	100	17.32	17.32
		8908-W	342.60	342.60	100	17.13	17.13
		8924-W	346.30	346.30	100	17.32	17.32
			<b>12,120.52</b>	<b>4,846.32</b>	<b>40</b>	<b>606.03</b>	<b>242.32</b>

Report-generation completed (2 pages)

CHRIS\_KUHLWEIN 01/25/01

**Figure 6.34** Report *Print Preview* (Page 2)

### 6.5.6.2.2

#### Creating the Report End Totals

The report is really taking shape now. One more difficult step remains and then it will be finished. In the last step you must create the sums for the entire report. This is difficult because some of this information does not exist in any table, nor does it exist as an accumulation of values from any *Dataltem*.

The totals for each salesperson were rather easy in comparison because these were merely sums of fields in the *Value Entry Dataltem* multiplied by a single factor. The report totals for *Sales Com.* and *Profit Com.*, on the other hand, require a sum to be multiplied by different factors. For example, the accumulation of the variable *Sales Com.* cannot be calculated for the entire report

like it was for a specific salesperson. This is due to the fact that the commission for the entire sales is neither a sum of all the commission rates nor an average of them. To calculate the *Sales Com.* of the entire report you must add the product of the sales multiplied by commission for every position.

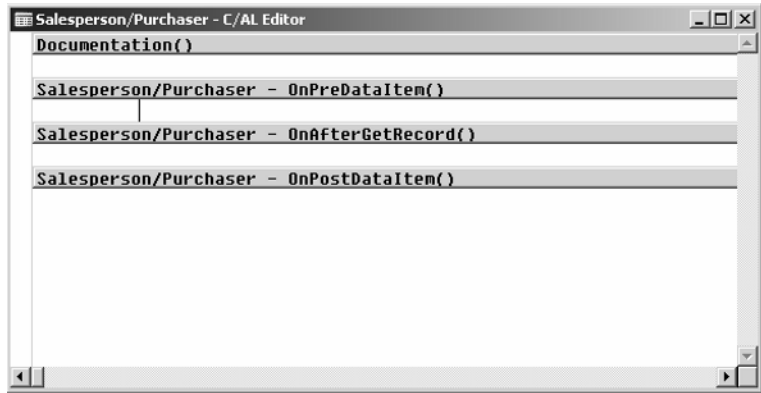
Such calculation complexities are difficult to foresee. In practice, even the best developers must discover such things by trial and error. Unfortunately, few programs are written that have all their bugs worked out, therefore you must thoroughly test and check them.

Try to achieve the report total for the variable *Sales Com.* by typing in the formula *Sales Amount (Actual) \* Commission %* in the *Salesperson/Purchaser, Footer (3)* would only produce the result equal to zero. Navision would not know which salesperson's commission rate to use in the final footer of the report. To solve these problems, you must introduce the tools *C/AL Code* and *Global* variables.

First create the report totals for the variables *Sales Amount (Actual)*, *PROFIT* and *PROFIT %*. These report totals will not require the use of *Global* variables, however, they will require the use of some *C/AL Code*.

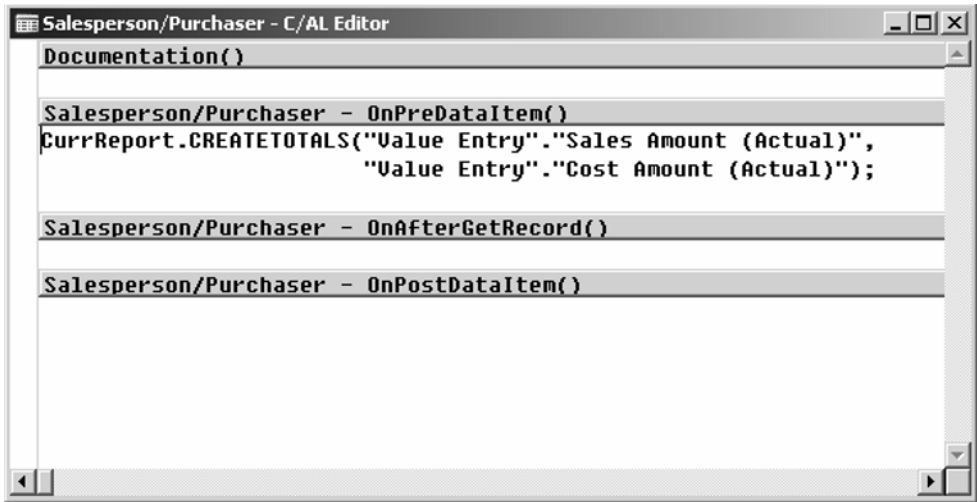
You need more powerful tools to control the *DataItem* objects than the *DataItem* property, *TotalFields*, for these report totals. This is because a report total, such as *Sales Amount (Actual)*, is an accumulation of a field not located in the *Salesperson DataItem*. Therefore, you must define this field value accumulation within the *DataItem* objects within the complex *C/AL Editor* window. This window gives you ultimate control over the fields you want to work with.

Click on the first *DataItem* and then press F9 to open the *C/AL Editor*. The following window will appear:



**Figure 6.35** *C/AL Editor* window

In this window you must instruct Navision to prepare an accumulation process for the two *Value Entry* fields. Within the section, *Salesperson/Purchaser - OnPreDataItem()*, type the code displayed in the following window:

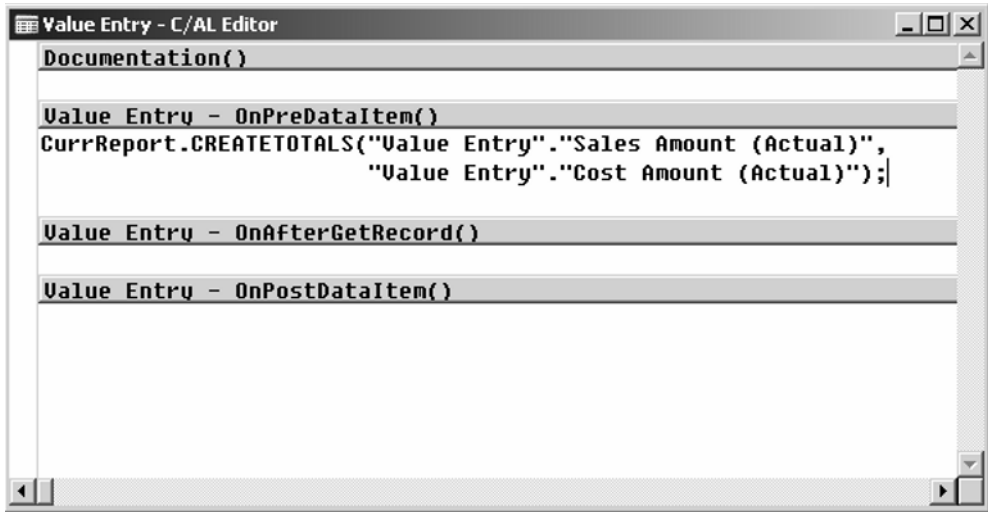


**Figure 6.36** Code in *C/AL Editor* window

This code tells Navision to create totals within the current report from the fields *Sales Amount (Actual)* and *Cost Amount (Actual)*, from the *DataItem, Value Entry*.

Now copy this piece of C/AL code and paste it into the *C/AL Editor* of the second *DataItem, Value Entry*. Press ESC to leave the *Salesperson/Purchaser - C/AL Editor* and open the *Value Entry - C/AL Editor* by clicking on the *Value Entry DataItem* and then

pressing F9. The *Value Entry - C/AL Editor* must look like the following:



**Figure 6.37** *Value Entry - C/AL Editor* window

Now that you have defined the *DataItem* objects so that they contain information about the reports totals, you can return to the *Section Designer* and instruct Navision how to display this information.

Open the *Section Designer* by going to:

- **View > Sections**

You have already inserted the section, *Salesperson/Purchaser, Footer (3)*. All that remains to be done is to create three new *TextBox* objects without labels and insert them into the *Salesperson/Purchaser, Footer (3)* section.

You must enter the name of the *Value Entry DataItem* followed by a period (.) when you are referring to a field from the *Value Entry DataItem* within the *Salesperson/Purchaser, Footer (3)* section. This section is from a different *DataItem*. (See point number 4 concerning the *SourceExpr* property in section 7.5.6.1.)

Select the first *TextBox* object and enter the following values into the following properties:

Xpos	7950
Ypos	0

Width	1800
Height	423
...	
Caption	Sales
...	
HorzAlign	Right
...	
FontName	Tahoma
...	
FontBold	Yes
...	
SourceExpr	"Value Entry"."Sales Amount (Actual)"

In the next *TextBox* enter the following values for the following properties:

Xpos	9900
Ypos	0
Width	1800
Height	423
...	
Caption	PROFIT
...	
HorzAlign	Right
...	
FontName	Tahoma
...	
FontBold	Yes
...	
SourceExpr	"Value Entry"."Sales Amount (Actual)" + "Value Entry"."Cost Amount (Actual)"

In the third new *TextBox* insert following values into the following properties:

Xpos	11850
Ypos	0
Width	1800
Height	423
...	
Caption	PROFIT %
...	
HorzAlign	Center
...	
FontName	Tahoma
...	
FontBold	Yes
...	
SourceExpr	("Value Entry"."Sales Amount (Actual)" + "Value Entry"."Cost Amount (Actual)") / "Value Entry"."Sales Amount (Actual)" * 100

You might consider adding a horizontal line to achieve a pleasant visual effect. Make the *Section Designer* look like the following window:

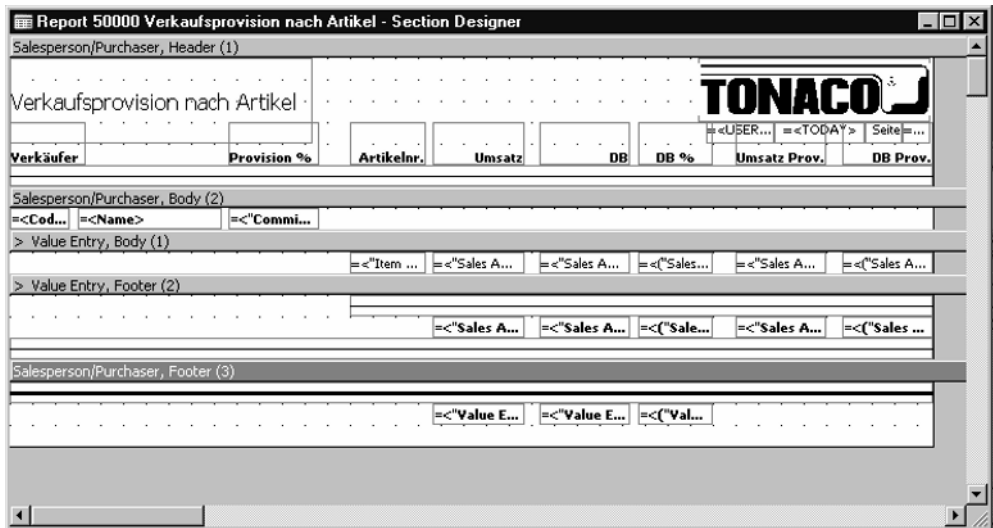


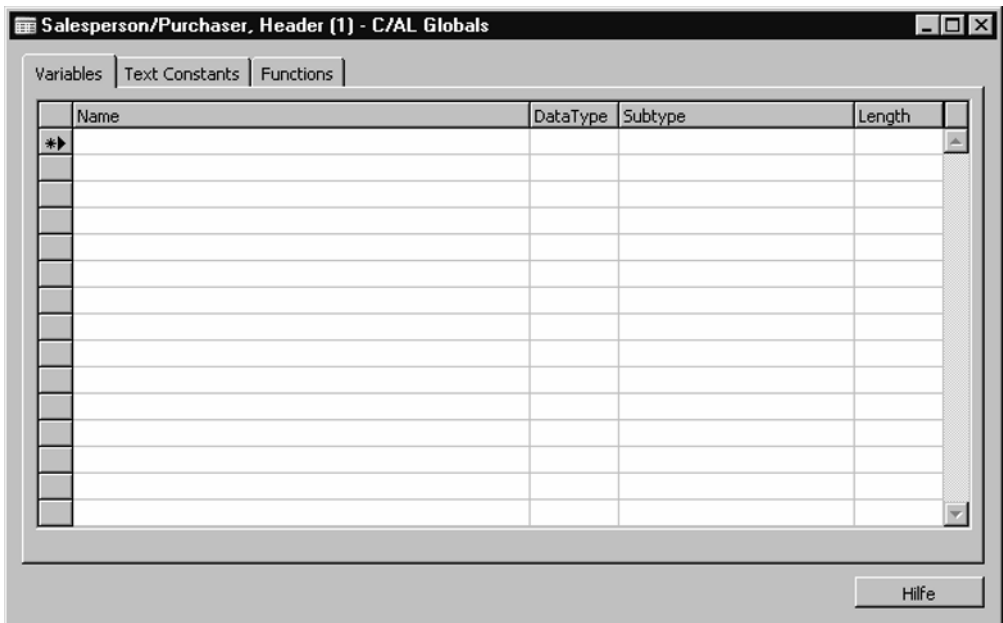
Figure 6.38 Section Designer window

Next we will create the more complex report totals for *Sales Com.* and *Profit Com.* Both are complex in nature and must be accumulated as single values as the report goes on. The only way to capture such information is to create a new variable within the report. This new variable should be available to every part of the report, therefore while it exists as a report variable it is not trapped within a single *DataItem* like the *DataItem* table fields are. Such a variable is called a *Global* variable and you can create as many as you need in the report. There are many types and uses of *Global* variables.

To see where report *Global* variables are created, go to:

- **View > Globals**

The following window will appear:



**Figure 6.39** *C/AL Globals* window

In this window you can create a variable for storing information for use anywhere in the report.

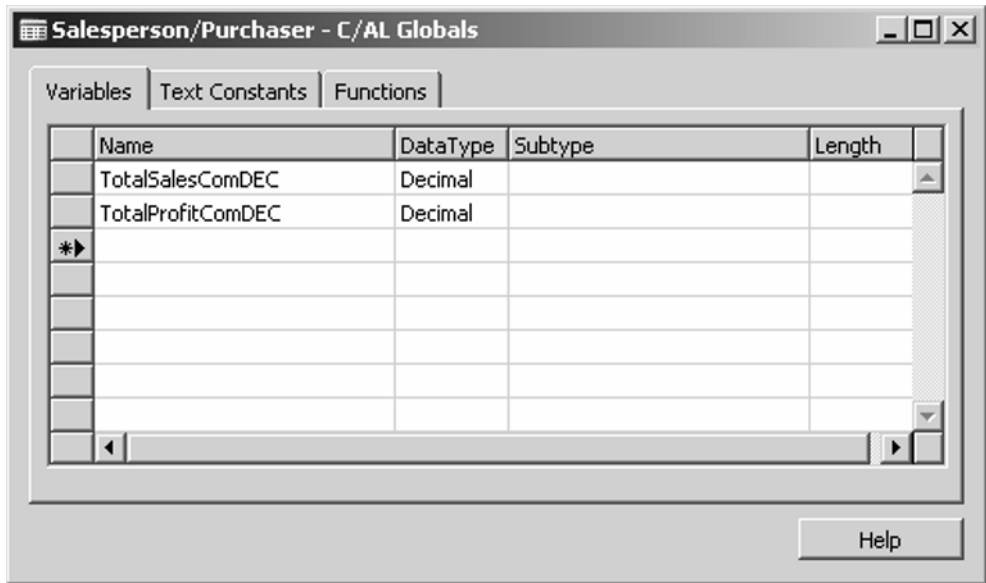
It goes without saying, that the *Global* variables are a very powerful tool in report-building. When you click on the *DataType* column and look through the menu list that appears, you see there are many new *DataType* categories that were not available in

the *DataType* categories in the table fields. Many of these report *DataTypes* are quite special and also complex. Some are used to export or import information from other programs like Microsoft Excel, for example. Other *DataType* categories, such as *Code-Unit*, *Form*, *Record*, *Report*, *Dataport*, are used to open other objects and parts of the Navision system.

The *DataType Record*, for example, allows you to create a *Global* variable that can store information from any table in Navision and make it available throughout the entire report—thus operating like a virtual *DataItem*. As we move through the book you will see more examples of the use of these powerful *Global* variables.

For the purpose of this report, you need only create a variable that holds a value that you want applied for the duration of the report. The value you need it to hold is a monetary one, therefore, choose the *DataType Decimal*. Enter into the first column, *Name*, *TotalSalesComDEC*. In the second column enter, *Decimal*. You need not concern yourself with the third and fourth columns. In this *Decimal Global* variable you can store and accumulate the value of *Sales Com*. Enter a second *Global* variable with the name, *TotalProvComDEC*, and give it also a *Decimal DataType*. In this variable you can store and accumulate the value of *Profit Com*. The *C/AL Globals* window will now look like the following:





**Figure 6.40** *C/AL Globals* window

The authors recommend that you follow a naming convention here when creating the *Global* variables. We suggest that you adhere to the following:

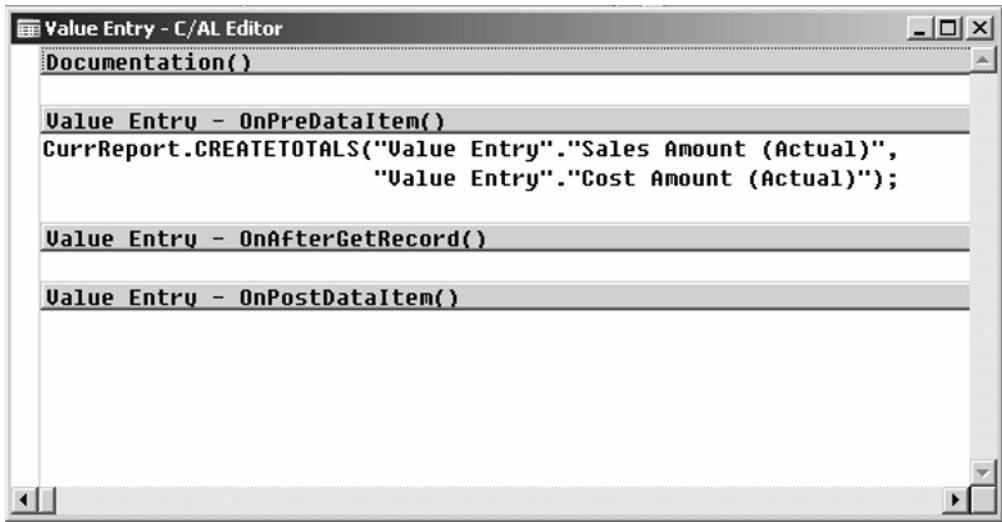
- No spaces or special characters within the name
- Easily recognizable name, analogous to its function
- Use three letters at the end of the variable name signifying the *Data Type*. This is useful for two reasons: you will be reminded not to mix differing *Data Types* in your calculations or *TextBox* objects without converting them first; it will help you to easily distinguish *DataItem* fields from *Global* variables in the report

Now that you have created variables to store decimal values, you must now assign values to them.

Go the *DataItem*, *Value Entry*, via the *DataItem* window. Next, open the *C/AL Editor* by going to:

- **View > C/AL Code**

The following window will appear:



**Figure 6.41** Value Entry - C/AL Editor window

In the *C/AL Editor* window you see horizontal gray bars similar to those in the *Section Designer*. The gray bars here function also in a similar way—marking off sections that are activated by events just as the different sections in the *Section Designer*. These C/AL Code sections are known as *triggers* in the Navision literature.

The first section in the *C/AL Editor* window is *Documentation()*. This section is inactive and exists only as a place for the developer to write notes concerning the history and function of the program code.

The second section, *OnPreDataItem()*, is activated each time Navision opens this *DataItem*, but before it has read or processed any of the records it finds within the *DataItem*. It is in this section that you should enter functions you wish applied to the *DataItem* as a whole—such as creating totals for every instance of a specific field. This section is analogous to the *Header* section of the *Section Designer*. Navision processes it directly before it processes a *Header* of the same *DataItem* in the *Section Designer*.

The third section is *OnAfterGetRecord()*. This section is activated when Navision reads a record within the *DataItem*. This is also where you can work with the individual record values. This section is analogous to the *Body* section of the *Section Designer*.

Navision processes it directly before it processes a *Body* of the same *DataItem* in the *Section Designer*.

The fourth section is *OnPostDataItem()*. This section is activated just before Navision leaves this *DataItem*. This section is analogous to the *Footer* section of the *Section Designer*. Navision processes it directly before it processes a *Footer* of the same *DataItem* in the *Section Designer*.

You need to accumulate the *Sales Com.* and *Profit Com.* variable for each position. Therefore, make the accumulation within the *OnAfterGetRecord* section which is activated for each record within the *Value Entry DataItem*.

Type these lines of C/AL Code into the *Value Entry - OnAfterGetRecord()* section within the *C/AL Editor*.

```
TotalSalesComDEC := TotalSalesComDEC +
    ("Value Entry"."Sales Amount (Actual)" *
    "Salesperson/Purchaser"."Commission %" / 100);

TotalProfitComDEC := TotalProfitComDEC +
    (("Value Entry"."Sales Amount (Actual)" +
    "Value Entry"."Cost Amount (Actual)") *
    "Salesperson/Purchaser"."Commission %" / 100);
```

The C/AL Code language states that the “variables must be valued at their former value plus the result of a new calculation.” The result will be the accumulation of the result of this calculation which is stored within the variables, *TotalSalesComDEC* and *TotalProfitComDEC*. Because these are *Global* variables we can use them equally well anywhere within the report. For our purposes we will wish to paste them where they will contain the result of these calculations for the entire report and be displayed at the reports end. The place for this is in the *Salesperson/Purchaser, Footer (3)* section within the *Section Designer*.

The *C/AL Editor* will now look like the following:

```

Value Entry - C/AL Editor
Documentation()

Value Entry - OnPreDataItem()
CurrReport.CREATETOTALS("Value Entry"."Sales Amount (Actual)",
    "Value Entry"."Cost Amount (Actual)");

Value Entry - OnAfterGetRecord()
TotalSalesComDEC := TotalSalesComDEC +
    ("Value Entry"."Sales Amount (Actual)" *
    "Salesperson/Purchaser"."Commission %" / 100);

TotalProfitComDEC := TotalProfitComDEC +
    (("Value Entry"."Sales Amount (Actual)" +
    "Value Entry"."Cost Amount (Actual)") *
    "Salesperson/Purchaser"."Commission %" / 100);

Value Entry - OnPostDataItem()

```

**Figure 6.42** Value Entry - C/AL Editor window

Note that the name of the *DataItem* is followed by a period (.) before the field name is stated. This is necessary—as it was in the *SourceExpr* property of a *TextBox* object—if the field referred to is a field from another *DataItem*. It is necessary here for the field, *Commission %*, because it is not a part of the *Value Entry* table but rather part of the *Salesperson/Purchaser* table. The same was done with *Value Entry* fields although it is not necessary. It is a good practice to write everything in “longhand” (as above) when you begin using C/AL Code because it is easier for you to understand.

Now you can add two new *TextBox* objects to the *Salesperson/Purchaser, Footer (3)* in the *Section Designer*. Use the two new *Global* variables in *TextBox SourceExpr* properties. Navision will now display the accumulated values for the *Sales Com.* and *Profit Com.* variables within the *Salesperson/Purchaser, Footer (3)* section.

Press ESC and go to:

- **View > Sections**

Open the *Section Designer* window. Now open the Toolbox menu by going to:

- **View > Toolbox**

Select the *TextBox* tool and insert two *TextBox* objects into the *Salesperson/Purchaser, Footer (3)* section and enter the following values into the following properties:

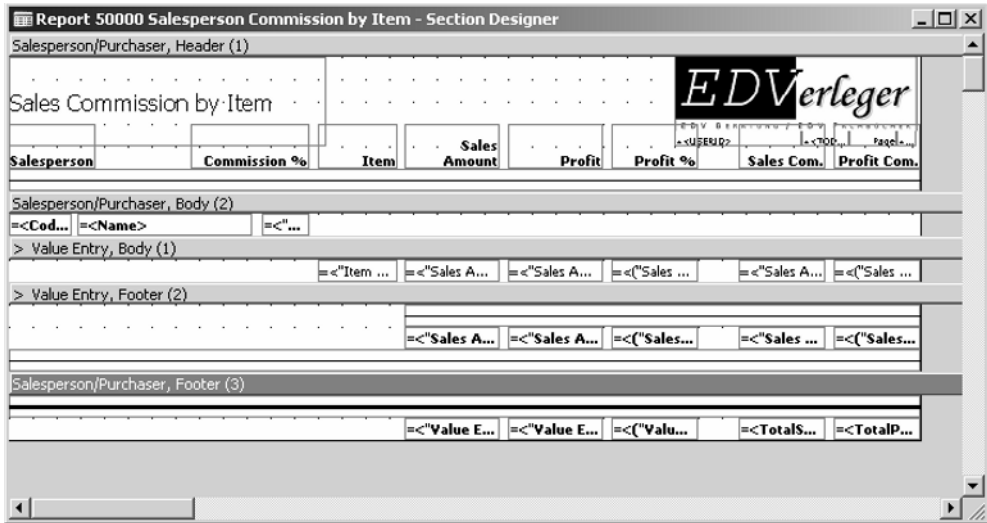
Xpos	13500
Ypos	423
Width	1800
Height	423
...	
Caption	Sales Com.
...	
HorzAlign	Right
...	
FontName	Tahoma
...	
FontBold	Yes
...	
SourceExpr	TotalSalesComDEC

Write the following values into the properties of the second new *TextBox*:

Xpos	15450
Ypos	423
Width	1800
Height	423
...	
Caption	Profit Com.
...	
HorzAlign	Right
...	
FontName	Tahoma
...	

FontBold	Yes
...	
SourceExpr	TotalProfitComDEC

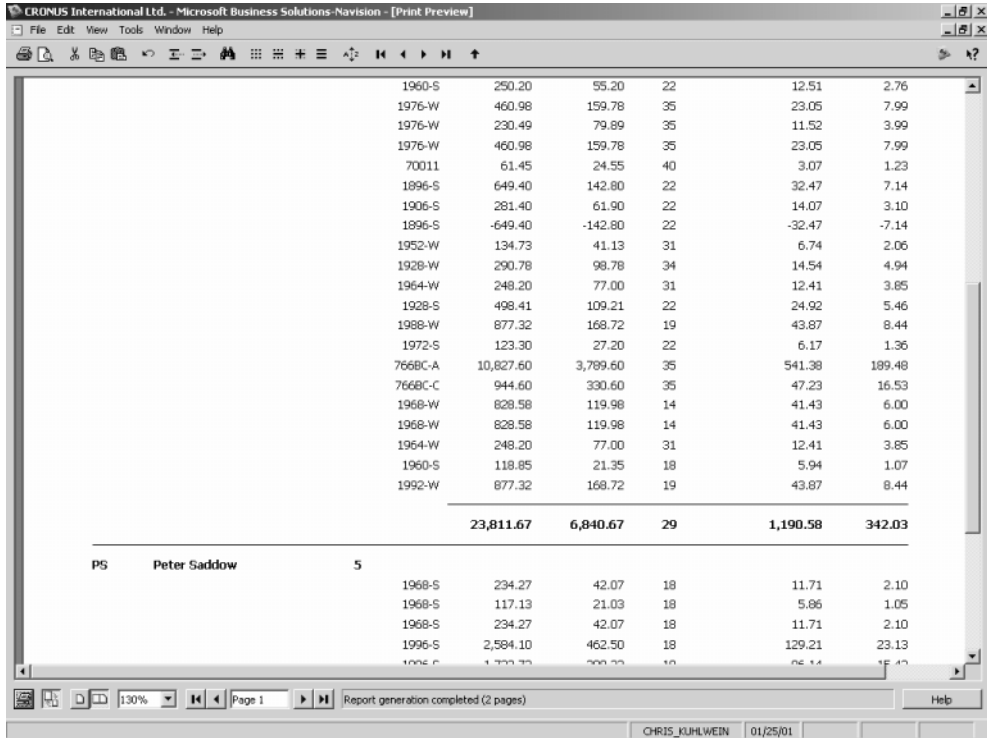
The *Section Designer* window will look like the following.



**Figure 6.43** *Section Designer* window

Press ESC twice and compile and save the report. Next, run and preview the report.

### 6.5.7 Viewing the Report



**Figure 6.44** *Print Preview, Page 1*

The last page showing the final sums will look like this:

**Sales Commission by Item**

**EDV Verlag**  
EDV-BERATUNG / EDV-FACHBUCHER  
CHRIS\_KUHLWEIN 07/12/01 Page 2

Salesperson	Commission %	Item	Sales Amount	Profit	Profit %	Sales Com.	Profit Com.
		70011	361.50	177.00	49	18.08	8.85
		9908-W	342.60	342.60	100	17.13	17.13
		9916-W	374.20	374.20	100	18.71	18.71
		9924-W	346.30	346.30	100	17.32	17.32
		9916-W	187.10	187.10	100	9.36	9.36
		9924-W	346.30	346.30	100	17.32	17.32
		9908-W	342.60	342.60	100	17.13	17.13
		9924-W	346.30	346.30	100	17.32	17.32
			<b>12,120.52</b>	<b>4,846.32</b>	<b>40</b>	<b>606.03</b>	<b>242.32</b>
			<b>35,932.19</b>	<b>11,686.99</b>	<b>33</b>	<b>1,796.61</b>	<b>584.35</b>

Report generation completed (2 pages)

CHRIS\_KUHLWEIN 01/25/01

**Figure 6.45** End sums shown in *Print Preview*

Now enter this report into the user environment which will allow your boss to find and use it herself. Press ESC twice to return to the end user environment and open the *Navigation Pane Designer*.

You should use your pocket calculator to check all the sums because the users rarely check the results of such detailed reports. Now, at the push of a button, your boss has an analysis that she can take to the salespeople which will not only help to motivate them but also help them to make good decisions about what to sell in the future.



# 7

## Introduction to Powerful Code Techniques

---

In this chapter we will discuss Microsoft Navision's deepest and most powerful development tool—C/AL Code. We offer a practical approach to development, covering the most important functions in the C/AL Code language and provide examples that you can immediately put into practice.

### 7.1

#### **A Practical Approach to Development**

Newcomers to program development may be scared by the mere appearance of programming code and the development environment. However, truth-be-told, virtually no professional programmer knows programming language syntax and functions completely or correctly. In practice, most developers have a handful of powerful techniques they know well and use over and over again. Using reference material as well as trial and error practice are the tools of this trade, therefore, do not feel uneasy about using them. Remember to test your techniques in a test Navision database before pasting them into your company's live Navision system.

As a rule, it is better to know a lot about a little than a little about a lot. This is true for business as well as crafts, and programming is a craft. The key is mastering the least amount of tools but that give you maximum power. The goal of this book is to guide you to discovering and mastering these key tools.

The best and quickest way to learn something is to connect it to something you already know and then expand your knowledge by building bridges. To apply this type of learning in Navision, you should first strive to know the end user environment very well and then peek into the code behind it. A sense of curiosity is invaluable. Always be on the look out for special features in Navision; open them up to see what kind of objects are behind them, what tables and fields and finally what code. This will be an enormous help when you need to build something for your own purposes. Over time you will build up a knowledge-base and know where to go in Navision to find something similar to what is you are trying to build. Often you can copy pieces of

code right out of the standard application and apply them to your own development projects. In this way you will also learn to program in the standard style of Navision. Being able to recognize patterns of similarity and copying will make it easier to work with programs of your own creation in the future.

If you understand how to do something by hand in Navision then you are ready to learn how the same thing can be done with C/AL Code. You need an understanding of what pieces of code function like the manual techniques that you already know and use. Making the leap from being solid Navision user to a solid Navision developer is not so difficult. Everything that can be done manually can be done programmatically, however, when it is done programmatically, you automate processes which reduces human error and speeds up the completion of tasks tremendously.

In the end, to successfully command Navision, the developer needs the same basic tools as the end user. These things boil down to a few items or steps:

- 1) Opening tables
- 2) Sorting these tables
- 3) Searching through these tables
- 4) Telling the system what do if it finds a specific condition.

Steps 1 through 3 can be accomplished with the C/AL Code functions: SETCURRENTKEY, SETFILTER and FIND. In Step 4, the system must make a decision based on a value it finds. You can use the following functions: IF, THEN and ELSE. Almost every action or activity can be accomplished by using some repetition of these four steps. These four steps can be mastered using just a few C/AL functions.

Understanding the logic behind these four steps will be the key to finding and building your own effective program solutions. Knowing what tables to choose, connect and search, as well as what to do when you find what you're looking for requires understanding and creativity. This is most easily accomplished when you can visualize the table relationships and understand the layers that make up a finished object.

Development and programming may seem like dry and determined work, however, in reality there are many ways to accomplish the same goal. Some are inferior to others and some quicker than others. In the end, what is most important is that you find a solution quickly so your firm can start profiting from

rather than suffering from its Navision investment. If you can master a few powerful techniques, you can use them over and over again gaining confidence and speed in your development work. The authors do not recommend memorizing all the C/AL Code functions and syntax; extensive help files and the compiler are there to help you. Instead, concentrate on what your firm needs, locating the information your firm needs and determining what decisions must be made when certain conditions in this information occur. To do this, you must have an understanding of your business, know how to use the Navision reference material and understand the four basic steps listed above.

## **7.2 Syntax and Style**

In this section we will discuss the syntax and style used in C/AL Code. More specifically, we will look at the functions that determine the basic architecture of C/AL Code.

### **7.2.1 Dealing With the Rigor of a Programming Language**

To the uninitiated, the rigor of the syntax within a given programming language can be dismaying. Certainly nothing is more frustrating than having a one thousand line program fail because of a single, misplaced period (.). However, take heart; while it is possible for such a thing to happen in programming, you will have much help along the way and such syntax problems should never be realized.

In Navision, you can hit the F11 key when you have finished writing and be immediately advised concerning any syntax errors you may have made. Likewise, it automatically places the cursor at the point where it can no longer interpret your code.

Remember, the best guide to Navision code is the code within standard Navision. If you can find something in the standard code that accomplishes the same function in something that you are building, then imitate the syntax until you know it yourself.

You can avoid syntax problems by practicing good code writing style and clear documentation. Every programming book and instructor will tell you how important documentation is, yet in reality, few practice it. In the business world there simply is not enough time to document everything well. However, you should make an effort because in the future it will help you isolate previous changes you have made. Likewise, organizing the parts of your program make it easier to complete the program. Likewise,

good documentation gives others an understanding of what you have done.

With respect to clear writing style, it is again good to follow the example of the standard Navision code. You can develop your own style, however it must be easy for others to follow the structure of your program. Your style should not contradict the standard.

Now let us look at some of the most important C/AL Code functions and others that deal most closely with the syntactic structure of C/AL Code. As we cover these few functions we will discuss some writing style conventions that go with them.

As already shown, C/AL Code can be used in the *SourceExp* property in *TextBox* objects within reports and forms. In these places you will only be able to write a single sentence of C/AL Code. For this reason, the *SourceExpr* property is not powerful enough for most complex tasks, therefore, you must begin using C/AL Code in the *C/AL Editor* environment. You can practice in the *C/AL Editor* environment of the report created in the last section, "Creating a Simple Report." Go into the *Object Designer* via:

- **Tools > Object Designer**

Click on the Report button and find the report you created earlier, "Salesperson Commission by Item." Click on the Design button and then on the *Value Entry DataItem*. Now press F9. The following window will appear:

Now practice some syntax in the section, *Value Entry - OnAfterGetRecord()*. Create a *Global* variable to use in the following examples by going to:

- **View > Globals**

Enter new *Globals* according to the following *C/AL Globals* window:

```

Documentation()

Value Entry - OnPreDataItem()
CurrReport.CREATETOTALS("Value Entry"."Sales Amount (Actual)",
    "Value Entry"."Cost Amount (Actual)");

Value Entry - OnAfterGetRecord()
TotalSalesComDEC := TotalSalesComDEC +
    ("Value Entry"."Sales Amount (Actual)" *
    "Salesperson/Purchaser"."Commission %" / 100);

TotalProfitComDEC := TotalProfitComDEC +
    (("Value Entry"."Sales Amount (Actual)" +
    "Value Entry"."Cost Amount (Actual)") *
    "Salesperson/Purchaser"."Commission %" / 100);

Value Entry - OnPostDataItem()

```

Figure 7.1 Value Entry - C/AL Editor window

Value Entry - C/AL Globals

Variables | Text Constants | Functions

Name	Data Type	Subtype	Length
TotalSalesComDEC	Decimal		
TotalProfitComDEC	Decimal		
testINT	Integer		
testCODE	Code		10
testDATE	Date		
*▶ testDEC	Decimal		

Help

Figure 7.2 Value Entry - C/AL Globals window

Press ESC to return to the *C/AL Editor*. Note that when writing C/AL Code in the *C/AL Editor*, Navision will always ignore anything written following two forward slashes (*//*). Such slashes are useful for writing documentation and notes to clarify what is happening in the code. The authors will use this style of documentation in upcoming code sets.

### 7.3 The Sentence: The Most Basic Unit of Syntax

Code must be written in sentences. A *sentence* is a unit by which Navision must take at least one action. The action could be anything from assigning a formula value to a variable to setting a filter on a table.

A sentence may contain other sentences within it. For example, you can tell Navision to look in a table for a specific set of records and before this action sentence is finished, tell Navision to set the values it finds for another variable. What is required to end each sentence is a semi-colon (;). The semi-colon tells Navision that a sentence is finished and that an action must be taken. The following is an example of a sentence within a sentence. Notice that the semi-colon ends the inner sentence before the outer is ended.

```
// SENTENCE ONE: Looking into a table.
IF "Value Entry".FIND('-') THEN BEGIN

    // SENTENCE TWO & THREE: Setting found values to global variables.
    testCODE := "Value Entry"."Item No.";
    testTEXT := "Value Entry".Description;

// End SENTENCE ONE with a semi-colon.
END;
```

The semi-colon function is useful because it allows you to write a sentence on many lines. Like the example in your report, you can spread over three lines the C/AL Code sentence to accumulate *Profit Com*. Doing so makes the code easier to read. Good style generally dictates that when you continue a sentence on the next line you begin the code two spaces to the right or directly below the main operator of the sentence. This is the style you followed in your report example, restated below:

```
TotalProfitComDEC := TotalProfitComDEC +
    ("Value Entry"."Sales Amount (Actual)" +
    "Value Entry"."Cost Amount (Actual)") *
    "Salesperson/Purchaser"."Commission %" / 100;
```

## 7.4 Referring to Variables

When you refer to a variable that has spaces or special characters within its name, you must enclose the variable name within quotation marks (“ ”). If you are in doubt as to whether the name contains a special character, it is safer to put it within quotation marks. Remember to press F11 if you have any doubts about Navision’s ability to interpret what you have written. Several examples of the use of variable names within quotation marks follow.

As you have already seen, when referring to variables in the *SourceExpr* of the *TextBox* object you may need to include the name of the *DataItem* or table followed by a period (.) before writing the variable name. This is necessary when a variable from a different *DataItem* than the one you are in, is being referred to.

When you are learning to program, it is helpful to include the *DataItem* or table name before the variable because it makes the code easier to read. It is possible that two different tables have a variable with the same name. This can make code difficult to read, especially if you are inexperienced or the program is very long. An example of confusing code follows:

```
"Salesperson/Purchaser".SETFILTER
  ("Salesperson/Purchaser".Code, "Salespers./Purch. Code");

IF "Salesperson/Purchaser".FIND('-') THEN
  TestTEXT :=Code;
```

You might ask here, if the variable, *Code*, is a description of the salesperson or of the sale? Perhaps the programmer simply forgot to include the *DataItem* name before the variable. If so, this will cause Navision to take the value of this variable from the current *DataItem*, the *Value Entry DataItem*. It is better if the programmer is more explicit in their code and includes documentation. For example:

```
// Look in "Salesperson/Purchaser" table for the salesperson of
// the current "Value Entry" record.

// Filter "Salesperson/Purchaser" table on the current
// salesperson of the "Value Entry" record.
"Salesperson/Purchaser".SETFILTER
  ("Salesperson/Purchaser".Code,
  "Value Entry"."Salespers./Purch. Code");
```

```
// If we find this salesperson then enter the "Value Entry"  
// Code into the TestTEXT variable.  
IF "Salesperson/Purchaser".FIND('-') THEN  
    TestTEXT := "Value Entry".Code;
```

## 7.5 Inserting a Value Into a Variable

In Navision it is necessary to use a colon (:) followed by an equals sign (=) after a variable when you want to enter a new value into it. For example:

```
TestTEXT := "Value Entry".Description;
```

Keep in mind, if you only use an equals sign, Navision will only make a comparison between the variables rather than transfer a value. It is possible to insert a variable value within its existing value. This is useful for creating accumulations. This was used in the report code above. For example:

```
TotalSalesComDEC := TotalSalesComDEC +  
    ("Value Entry"."Sales Amount (Actual)" *  
    "Salesperson/Purchaser"."Commission %" /  
    100);
```

## 7.6 Implications

One of the most important structures in programming is the IF, THEN logical implication. Using this function you can force Navision to make a choice based on a logical comparison between a set of variables. This function can contain other functions or even another implication within it. Sometimes an entire program, made up of hundreds of lines, can be contained within one IF, THEN decision. This is difficult to handle sometimes while the semi-colon which ends the IF, THEN sentence is separated many lines after the beginning of the function. This can make it difficult to determine which decision is being based on which condition. Maintaining a good writing style is imperative and can almost always ensure that complex code is easy to understand.

The most basic IF, THEN implication structure is as follows:



```
IF x = y THEN z := x;
```

Between the IF and THEN you can place any logical relation or function that evaluates to either true or false. Below is a list of logical operators that you can use:

<b>C/AL Code Logical operators</b>		
<b>Simple binary operators</b>		
=	$(x + y) = (y + z)$	True if compared expressions have the same value.
<>	$(x + y) <> (x - y)$	True if compared expressions have the different values.
>	$(x + y) > (x - y)$	True if left expression has a greater value than the right expression.
<	$(x - y) < (x + y)$	True if left expression has a smaller value than the right expression.
<b>Complex binary operators</b>		
AND	$(x = y) \text{ AND } (x < Z)$	True if both expression are true.
OR	$(x = y) \text{ OR } (x < Z)$	False if both expressions are false.
<b>Unary Complex operator</b>		
NOT	NOT $(x = y)$	True if expression is false.

**Figure 7.3** C/AL Code Logical Operators

With logical relations you can build complex conditions upon which Navision will decide to execute what follows the THEN keyword.

You can also use other C/AL Code functions as the condition upon which Navision will execute the THEN function. For example, suppose you want Navision to change the text in a variable only if it finds a salesperson in the *Value Entry* table. You could write:

```
// If SalespersonX is found then enter "Description"
// into the variable TestTEXT
IF "Value Entry".FIND('-') THEN
    TestTEXT := "Value Entry"."Description";
```

Here the C/AL Code function, FIND, is the condition upon which Navision will decide to enter the Description information into the variable, *TestTEXT*.

It is possible to give Navision a counter option to execute if it finds that the condition after the IF to be false. You can do this

by adding an ELSE after the THEN actions. The ELSE component continues the IF, THEN sentence, therefore extending the use of the semi-colon (;) which will end the IF, THEN sentence. For example:

```
// If SalespersonX is found then enter "Description"
// into the variable TestTEXT
IF "Value Entry".FIND('-') THEN
  TestTEXT := "Value Entry"."Description"
  ELSE
  TestTEXT := 'Note: Salesperson has no positions ' +
    'in the Value Entry table.';
```

Directly after the THEN statement you must either enter one sentence followed by a semi-colon or an ELSE. If you want to enter more than one sentence you must type BEGIN. The BEGIN opens the IF, THEN function to the possibility of executing many things based on one IF condition. After you type BEGIN, you can enter as many sentences as needed. Each of these must be self-contained and therefore have a semi-colon at the end. Navigation will consider everything after a BEGIN keyword to depend on the IF condition until it finds the keyword END. For example:

```
// SENTENCE ONE: Looking into a table.
IF "Value Entry".FIND('-') THEN
  BEGIN
    // SENTENCE TWO & THREE: Setting found values to
    // global variables.
    testCODE := "Value Entry"."Item No.";
    testTEXT := "Value Entry".Description;
    // End SENTENCE ONE with a semi-colon.
  END;
```

You must be careful with your writing style so that it is clear which actions depend on which conditions. As a general rule, if something depends on something else it should be indented. Now it is visually apparent that a piece of code is nested within another.

In some cases it is necessary to have one question depend on another. In such an instance, you will need to write IF, THEN statements within IF, THEN statements. For example:

```
IF "Salesperson/Purchaser".FIND('-') THEN
  IF "Value Entry".FIND('-') THEN
    BEGIN
```

```

testCODE := "Value Entry"."Item No.";
testTEXT := "Value Entry".Description;
END;

```

As you can see, the semi-colon (;) after the END keyword ends both IF, THEN statements. If you were to put all the available possibilities together, you could write very complex IF, THEN constructions. Always use the F11 compiling tool to check the placement of your semi-colons and keywords and the intelligibility of your programming.

Mastering the construction of IF, THEN functions will give you tremendous power to direct Navision's decision-making in your program.

## 7.7

### Looping

Often it is necessary to construct a section of code that repeats. The REPEAT, UNTIL function allows you to accomplish this. You can design the REPEAT, UNTIL function so that it repeats a section of code until a specific condition is achieved. In the following example we will use the REPEAT, UNTIL function to accumulate a daily charge.

```

// Resetting the global variable's values.
AccChargeDEC := 0;
DailyChargeDEC := 5.25;
MovingDATE := StartDATE;

// Beginning to add daily charge for each day
// from StartDATE to EndDATE.
REPEAT

    AccChargeDEC := AccChargeDEC + DailyChargeDEC;

    // MovingDATE is one day closer to EndDATE each
    // time this line repeats.
    MovingDATE := CALCDATE('+1D',StartDATE);

// Exit this loop when it has repeated once for
// each day in period.
UNTIL NewDATE = EndDATE;

```

The REPEAT, UNTIL loop can be also be used to step through the records of a table. This is useful when you want to perform a calculation for each record that you find within a table. To get Navision to move through the records of your table, you must

use the NEXT keyword at the end of your REPEAT, UNTIL loop. In the following example, calculate the average commission of the salespeople by stepping through the *Salesperson/Purchaser* table in a REPEAT, UNTIL loop.

```
// Find first salesperson in
// the "Salesperson/Purchaser" table.
IF "Salesperson/Purchaser".FIND('-') THEN
  // If found then go through each salesperson's record.
  REPEAT
    CommisSumDEC := CommisSumDEC +
      "Salesperson/Purchaser"."Commission %";
    RecordCountINT := RecordCountINT + 1;
  // Exit loop when there are 0 more records.
  UNTIL "Salesperson/Purchaser".Next = 0;
  // Calculate average salesperson's commission.
  AveCommisDEC := CommisSumDEC /
    RecordCountINT;
```

If you are moving through table records that differ from the *DataItem* where you are writing the code, then you must include the name of the *DataItem* followed by a period (.), followed by the keyword, NEXT. This tells Navision within which *DataItem* it is supposed to find the next record. What comes after the NEXT keyword is the description of how you want Navision to step through the table. If you write:

```
UNTIL "Salesperson/Purchaser".Next = 2;
```

Navision will step through the records in the *Salesperson/Purchaser* table two at a time and will end the loop when only two records remain in the table. In most cases, you will want to go through every available record in the table, therefore you should use NEXT = 0;—just like the code in the example above. Now Navision will step through the table until there are no more records in the table.

## 7.8 Globals and C/AL Functions: Establishing Table Relations

In this section you will learn how you to create a variable and link it to a table. This technique allows you to pull any information out of any table and bring it into a report. Next, you can use this new knowledge to further improve your report.

### 7.8.1 **Calling Foreign Table Information**

Remembering the guiding principle that a report should answer more questions than it raises, take another look at the report we created in the previous section. Although the report we created is called "Sales Commission by Item," we failed to include the name of the products or items in the report. Look once more at the report output in the *Print Preview* window below: (See figure 7.4)

You should not assume that the salespeople have the item numbers of the products memorized. While we have included the number of the salesperson and their name, we have not been consistent by including the name of the product.

Therefore, let us find a way to include the item name in this report. Doing so is a good exercise for learning the table-connecting C/AL Code functions and it will improve the report.

It would be helpful if the name of the item printed next to the item number on every line in the report where the sale of an item also appears. Open the internal structure of the report again and try to accomplish this. Enter the *Object Designer* window by going to:

- **Tools > Object Designer**

Click on the Report button. Next, select the report you created earlier, "Sales Commission by Item." Because you want the item name displayed in the report printout, you must go into the *Section Designer* where the print layout of the report is developed. Therefore, go to:

- **View > Sections**

Sales Commission by Item

EDV Verleger  
EDV BERATUNG / EDV FACHBUCHER  
CHRIS\_KUHLWEIN 07/13/01 Page 2

Salesperson	Commission %	Item	Sales Amount	Profit	Profit %	Sales Com.	Profit Com.
		70011	361.50	177.00	49	18.08	8.85
		8908-W	342.60	342.60	100	17.13	17.13
		8916-W	374.20	374.20	100	18.71	18.71
		8924-W	346.30	346.30	100	17.32	17.32
		8916-W	187.10	187.10	100	9.36	9.36
		8924-W	346.30	346.30	100	17.32	17.32
		8908-W	342.60	342.60	100	17.13	17.13
		8924-W	346.30	346.30	100	17.32	17.32
			12,120.52	4,046.32	40	606.03	242.32
			35,932.19	11,686.99	33	1,796.61	584.35

Report generation completed (2 pages)

CHRIS\_KUHLWEIN 01/25/01

**Figure 7.4** Print Preview of “Sales Commission by Item” report

The name of the item should appear each time the item is sold; this information is output with each *Value Entry DataItem* record. Therefore, click on the *Value Entry - Body (1)* section and go to:

- **View > Field Menu**

Select the possible fields in this *DataItem*. As you look through this list, you will notice there is no field that contains the item name, only a field with the item number. This is due to the fact that the *Value Entry* table (which is the table upon which this *DataItem* is built) does not contain the item name. When the tables, upon which your report is built, do not contain all the information necessary for your report, you must create a virtual *DataItem*. Using the virtual *DataItem* you can bring in information from other Navision tables that do not exist in the main *DataItem* objects of your report.

You have seen how to establish a relation between two or more tables in reports by connecting *DataItem* objects. There is an-

other way within programs to establish table relations—that is by creating special *Global* variables that refer to a table. Such special *Global* variables behave like virtual *DataItems*. Then we can call this "Global" variable and work on it just like it was one of the *DataItem* objects. Thus a *Global* variable can be used to bring information from any table into your program. The techniques discussed here can be used throughout Navision in any type of object, in forms, reports, dataports, etc.

Now we will go through the standard method for building a table relation between the report and the item name information. First, you must find the tables that hold the required information. Second, check to see if there is a way to link these two tables by mapping the primary key of one table to a field in the second table. Lastly, you must work out the technique of linking and processing.

You want to have the name for each item that has a number that appears within the *Value Entry* lines of the report. You already have the item number within the report in the *Value Entry DataItem*.

Now you need information about the name of the item; the most obvious place to find this information is in the *Item Card*. Go to:

- **Warehouse > Planning & Execution > Item**

Here is the *Item Card*. The table that exists under this form contains both the item number and name. Open the interior of the *Item Card* to see what table it is built on. Press CTRL+F2 and then go to:

- **View > Properties**

In the *SourceTable* property you see the table name, *Item*. This is the name of the table that holds the information you need within your report.

Now the *Item* table and the *Value Entry* table must be linked. In this case, the item number within the *Value Entry* table is the primary key within the *Item* table. Therefore, you can set up a link between these two tables simply by telling Navision that “for every item number it finds within the *Value Entry* table it must find one and only one item in the *Item* table.” Because the item number is the primary key in the *Item* table, you can be sure that Navision will find an item in the *Item* table for each item listed in the *Value Entry* table.

The last step is to determine how we can actually link these two information sources. So far in this book you have seen several

ways to establish connections between tables. For example, in the forms and tables you can simply use the property, *SourceTable*, to link to a table. In report-building you can use *DataItem* objects to link tables to your reports. Another powerful and flexible way to link tables involves using C/AL Code and *Global* variables.

For the purposes of your report let us create a special *Global* variable that will itself be a link to a table. You can call this special variable a *Record Global*.

Go to:

- **View > C/AL Globals**

Enter *ItemREC* as the name of the new variable. In the *Data Type* column type, *Record*, or select it from the list that appears in this column. This *Data Type* tells Navision that the *Global* variable is to be a link to a table. When you click on the *Subtype* column and then on the assist that appears, you will see a list of the tables that exist within Navision. Type, *Item*, or select it out of the table list that appears in this column. Now, every time you refer to this special *Global* variable in your program—whether in the *C/AL Code Editor* or in a *TextBox SourceExpr* property—Navision will access a record in the *Item* table.

Now you must determine which record in the *Item* table it will access. Doing so requires the use of C/AL Code functions to establish a table relation between one of the *DataItem* objects and the new *Global* variable, *ItemREC*. For every line of the *Value Entry DataItem* you must call the correct record within the *Item* table where the item's name is located. This information must be available for use within the *Sections* area of the report so that it can be displayed with the output.

This table connection must take place within the *Value Entry DataItem* itself. The only place within the *Value Entry DataItem* that is flexible enough to accomplish a connection with a *Record Global* variable is within the *C/AL Editor* window of the *Value Entry DataItem*. Open the *Value Entry C/AL Code Editor* window by clicking on the *Value Entry DataItem* and

- **View > C/AL Code**

First look to see if the key of one table exists as the primary key of the other table. You already know that the primary key of the *Item* table is the *Item* number. The field name of the *Item* number is, *No*. The *Item* number also exists in the *Value Entry* table with the field name, *Item No*. You must link these two fields.



Now you must learn the C/AL Code functions that allow to bring the *Item* table information into the correct relation with *Item No.* field of the *Value Entry DataItem*.

When a *Global* variable, defined as *Data Type Record*, is used in a program, Navision leaves the variable on the record that either appears first in the table according to the table sorting and any filters set on it or leaves it on the last record called. When you use a *Global* record variable, you must be sure that there are no other filters set on it or that no previous use of it will interfere with what you need it to do.

To remove previous filters and/or sorting on the variable, use the C/AL Code function, RESET, before beginning work with the *Record Global* variable. Although you may be positive that your new variable, *ItemREC*, has not been used elsewhere in the report, you can use this function to eliminate the possibility that later someone may write code that precedes your code. Therefore, write the following code into the *OnAfterGetRecord()* section within the *C/AL Code Editor* window of the *Value Entry DataItem*:

```
// This clears the variable from any former uses.  
ItemREC.RESET;
```

As usual, if you are referring to a record outside the present *DataItem*, you must use the name of the foreign *DataItem*. In this case, you are writing code into the *Value Entry DataItem* but referring to the *Record Global*, *ItemREC*. Therefore, you should type: *ItemREC* followed by a period (.) before using the RESET function on it.

Now you must determine how Navision should sort the records within the *ItemREC Global* variable. How the records in this variable are sorted will determine the type of information you will find inside the table. For example, it is possible to sort the table behind this *Record Global* variable by the *Item* number in a descending order. Now if you ask Navision to tell you the first *Item* in table, naturally, Navision returns the first *Item* when the table is sorted backwards. In C/AL Code you can control the sorting of a *DataItem* or *Record Global* variable with the function SETCURRENTKEY. You should state the *DataItem* sorted first, followed by a period (.) and then the function name followed by the fields—separated by commas (,) and contained within parentheses ( )—in the order you want them to be sorted. For example:

```
// Setting the sorting of the "ItemREC" record variable.  
ItemREC.SETCURRENTKEY(ItemREC."No.");
```

In the present example it is not necessary to apply this function because you want the *ItemREC* to be sorted by *No.*—the primary key of the table. The standard sorting of any table will always be a sorting by the primary key unless you direct Navision to use a different sorting via the SETCURRENTKEY function. When you are learning Navision, you should go through all the steps because it is good practice and provides clarity.

Now you must determine which record in the *Item* table Navision will find when it opens the *ItemREC* variable. To do this, set a filter on this *Record Global* so that the only record Navision can access is the record containing the *Item* number of the current *Value Entry* record. This is just like setting a "" on the *Item* table within the end user environment, except in this case it was accomplished purely through the use of code.

Set "" on the *Record Global*, *ItemREC*, by using the C/AL Code function, SETFILTER. After the line with the SETCURRENTKEY function, type the following:

```
// Filter the Item table on the item  
// referred to this "Value Entry" record.  
ItemREC.SETFILTER(ItemREC."No.",  
                  "Value Entry"."Item No.");
```

You must give the name of the *DatalItem* or *Global* record variable followed by a period (.) before you can use the SETFILTER function. Now the filter itself is determined by two conditions. These must be contained within parentheses ( ) and separated by a comma (,). The first condition must be the name of the field against which you wish to set a filter. In the present example you want to set the filter against the field, *No.*, within the *Global* variable *ItemREC*. Therefore you should write: *ItemRec. No.*

The second condition is the filter to be set. This can be a variable, a record or simply a static text such is used when setting *s* in the end user environment. Here the filter should correspond with the *Item* number in the *Value Entry* record. Therefore, type the *DatalItem* and field name of the *Value Entry* *Item* number as the filter: *Value Entry Item No.* Now, the field, *No.*, in the *ItemREC* will be filtered. Navision will only access the *Item* number of the *Item No.* of the *Value Entry DatalItem*.

If you want to use static text as your filter, you must write the text within apostrophes ( ' '). The text you write must obey the same rules as those governing all filters in Navision. This was covered earlier in the section on filtering. Suppose you want to use C/AL Code to set a static text filter and you want Navision to define a filter on the *ItemREC* Global record variable where only records of *Items* with a *Item* number beginning with the number 1 are to be accessed. You would write the following code to establish this:

```
// Filter ItemREC so Navision only
// accesses items beginning with the number 1.
ItemREC.SETFILTER(ItemREC."No.",'1*');
```

Here the *Table Filter 1\** is set on the *Item* table. This produces the same effect as when you go into the *Item Card* or *Item List* view and click on the Table Filter tool and write *1\** in the *Filter* column across from the field, *No.* It is as if you are simulating what an end user would do, but within a program using C/AL Code. When you are learning to program, always try to think about the parallel activity of the end user as it relates to what you are doing with code. Doing so helps you understand what is necessary in both environments so that you can properly control Navision. This also gives you two methods for checking your results as opposed to one.

Next, instruct Navision to open the *Global* record variable. You can do this with the C/AL Code function, FIND—which you have seen in previous examples. FIND is an extremely flexible function and it avoids errors that other C/AL Code functions sometimes return when the goal record does not exist.

## 7.8.2

### **FIND: Searching For a Specific Record in Tables**

You can use the FIND function to open a table and access information within it. Using sorting and *Table Filter* setting functions, you can determine which records Navision will find in your table. The FIND function will find only one record in a table at a time. You must first create a *Record Global* variable that connects to a table. Next, you can use the FIND function to open the table connected to the *Record Global* variable.

In the present example, the *ItemREC* is a *Record Global* variable connected to the *Item* table. Find the *Item* name in the *Item* table. You have already determined that the *ItemREC* is sorted by

the primary key, *No.* It is already filtered on the item number in the *Value Entry DataItem*. Now when you instruct Navision to open the *ItemREC* with the FIND function, it will only access *Items* with numbers equal to the *Item* number within the *Value Entry DataItem*. Now write the code needed to establish the table connection. Navision executes the instructions within the *OnAfterGetRecord( )* section of the *Value Entry - C/AL Editor* every time it finds an *Item* sale in the *Value Entry* table. You must also instruct Navision to find an *Item* name for each *Item* sold. Therefore, you must put code in this same section so Navision finds a new *Item* name for each sale found. Below is required C/AL Code:

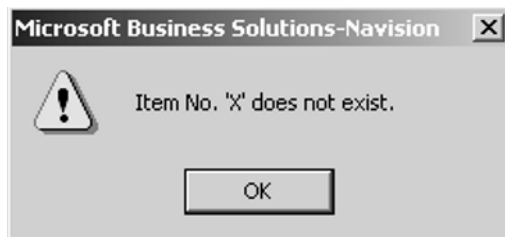
```
// This clears the variable from any former uses.
ItemREC.RESET;

// Setting the sorting of the "ItemREC" record variable.
ItemREC.SETCURRENTKEY(ItemREC."No.");

// Filter the Item table on the item
// referred to in this "Value Entry" record.
ItemREC.SETFILTER(ItemREC."No.", "Value Entry"."Item No.");

// Open the Item table and find the first
// record allowed by filters and sorting
ItemREC.FIND('-');
```

Now when you ask Navision to find a record in a table it is possible that under the present filtering conditions, it may find no record. This can happen, for example, if you inquire about the sale of an *Item* that has been deleted from the *Item* table. The code above is constructed in such a manner that if no record is found, the program will fail and return the following error message:



**Figure 7.5** Error Message

To avoid receiving such an error message, use the FIND function in the following manner:

```
// Open the Item table and find the first
// record allowed by filters and sorting
IF ItemREC.FIND('-') THEN;
```

By using the FIND function within an implication, you eliminate the possibility of crashing the program. Now Navision will not return an error message if it fails to find something in the table. You can include an automatic message that informs the user when no match is found. The program will not crash and the user is informed that there are *Items* in the *Value Entry* table which cannot be found in the *Item* table. Simply add a MESSAGE function as the alternative condition in the implication. For example:

```
// Open the Item table and find the first record
// allowed by filters and sorting if Navision
// finds no match for an item it will tell the user.
IF ItemREC.FIND('-') THEN
  BEGIN END
  ELSE
    MESSAGE('Item ' + "Value Entry"."Item No." +
            ' nicht gefunden.');
```

Unfortunately, it is necessary to type BEGIN and END between the THEN and ELSE keywords because Navision expects a command to exist between the THEN and ELSE. You could write the code alternatively:

```
// Open the Item table and if the "Value Entry"
// item is not found then tell the user
// what item could not be found.
IF NOT ItemREC.FIND('-') THEN
  MESSAGE('Item ' + "Value Entry"."Item No." +
          ' nicht gefunden.');
```

The second method uses less code, runs faster and uses less storage space. The disadvantage with this method is that it necessitates placing the most probable outcome first in the IF, THEN implication. Later, if you want to add a sentence that will be completed when the *Item* is found, you must add this sen-

tence **after** an ELSE keyword. Doing so results in a slower execution of the code because Navision must consider the less likely case first.

The FIND function should follow the name of the *DataItem* that it must open. A period (.) must also be placed between the name of the *DataItem* and the FIND keyword.

Also note that the FIND function is followed by a minus sign (-) contained within apostrophes ( ' ') which are contained within parentheses ( ). With the minus sign, as with other signs, you have the option to choose whether the FIND function opens the last record in a table or the first. This option is controlled by using either a plus (+) or minus sign (-) contained within apostrophes ( ' ') and parentheses ( ). You must use a minus sign if you want Navision to choose the first record it finds in the table it opens. If you want Navision to find the last record, then you must type a plus sign.

It is useful for the last record found in a table to be opened if you want to refer to the last time you bought a particular *Item*. Suppose you create a *Record Global* variable, *ItemLedgerREC*, that is connected to the *Item Ledger Entry* table. You can write the following code to find the date on which the *Item* was last purchased:

```
// Filter on item purchases only.
ItemLedgerREC.setfilter("Entry Type",'Purchase');

// Filter on current item in "Value Entry".
ItemLedgerREC.setfilter("Item No.",
                        "Value Entry"."Item No.");

// Enter date of item's last purchase
// into testDATE global variable.
IF ItemLedgerREC.FIND('+') THEN
    testDATE := ItemLedgerREC."Posting Date";
```

In our report we are filtering the *Item* table by an entry in its primary key. Every entry in a primary key must be unique so that Navision will find only one record in the *Item* table for each record in the *Value Entry* table. Therefore, it makes no difference whether you instruct Navision to find the first or the last record.

Once Navision finds it, you will need a new *Global* text variable in which to store the name of the *Item*. Go to:

- **View > C/AL Globals**

Enter the name, *ItemNameTEXT*, into the leftmost column. Select *Text* as the *Data Type* and type 30 in the fourth column which is *Length*. You can store the new *Item* name in this new *Global* variable and then paste this variable into the report layout so it appears in the printout.

The name of the field in the *Item* table where the *Item* name is stored is, *Description*. You must refer to this field as *ItemREC.Description* in the code so Navision knows what *Global* to open to find *Description*.

Enter the following code into the *Value Entry - C/AL Editor* of the report, section, *OnAfterGetRecord()*:

```
// This clears the variable from any former uses.
ItemREC.RESET;

// Setting the sorting of the "ItemREC" record variable.
ItemREC.SETCURRENTKEY(ItemREC."No.");

// Filter the Item table on the item
// referred to in this "Value Entry" record.
ItemREC.SETFILTER(ItemREC."No.", "Value Entry"."Item No.");

// Open the Item table and find the first record
// allowed by filters and sorting. If an item is
// found, enter its name into the variable called ItemNameTEXT
// If Navision finds no match for an item it will tell the user.
IF ItemREC.FIND('-') THEN
    ItemNameTEXT := ItemREC.Description
ELSE
    MESSAGE('Item ' + "Value Entry"."Item No." +
        ' not found!');
```

After typing this code into the *C/AL Editor*, your window should now look like the following:

You could have solved this issue for the report with much less code by using the GET function. This was an option in this specific case because you are filtering on the primary key of the *Record Global* variable. However, we had you avoid using the GET function method so that you first learn the most flexible and powerful method. You can pursue the goal of designing perfectly efficient code after you have attained a foundation of understanding.

```

Documentation()

Value Entry - OnPreDataItem()
CurrReport.CREATETOTALS("Value Entry"."Sales Amount (Actual)",
                        "Value Entry"."Cost Amount (Actual)");

Value Entry - OnAfterGetRecord()
TotalSalesComDEC := TotalSalesComDEC +
                    ("Value Entry"."Sales Amount (Actual)" *
                     "Salesperson/Purchaser"."Commission %" / 100);

TotalProfitComDEC := TotalProfitComDEC +
                    (("Value Entry"."Sales Amount (Actual)" +
                     "Value Entry"."Cost Amount (Actual)") *
                     "Salesperson/Purchaser"."Commission %" / 100);

// This clears the variable from any former uses.
ItemREC.RESET;

// Setting the sorting of the "ItemREC" record variable.
ItemREC.SETCURRENTKEY(ItemREC."No.");

// Filter the Item table on the item
// referred to in this "Value Entry" record.
ItemREC.SETFILTER(ItemREC."No.", "Value Entry"."Item No.");

// Open the Item table and find the first record
// allowed by filters and sorting. If an item is
// found, enter its name into the variable called ItemNameTEXT
// If Navision finds no match for an item it will tell the user.
IF ItemREC.FIND('-') THEN
    ItemNameTEXT := ItemREC. Description
ELSE
    MESSAGE('Item ' + "Value Entry"."Item No." +
            ' not found!');

Value Entry - OnPostDataItem()

```

**Figure 7.6** Value Entry - C/AL Editor window

Press F11 to make sure you have made no syntax errors or forgotten any of the *Global* variables referred to in the code. Press CTRL+S to save the report.



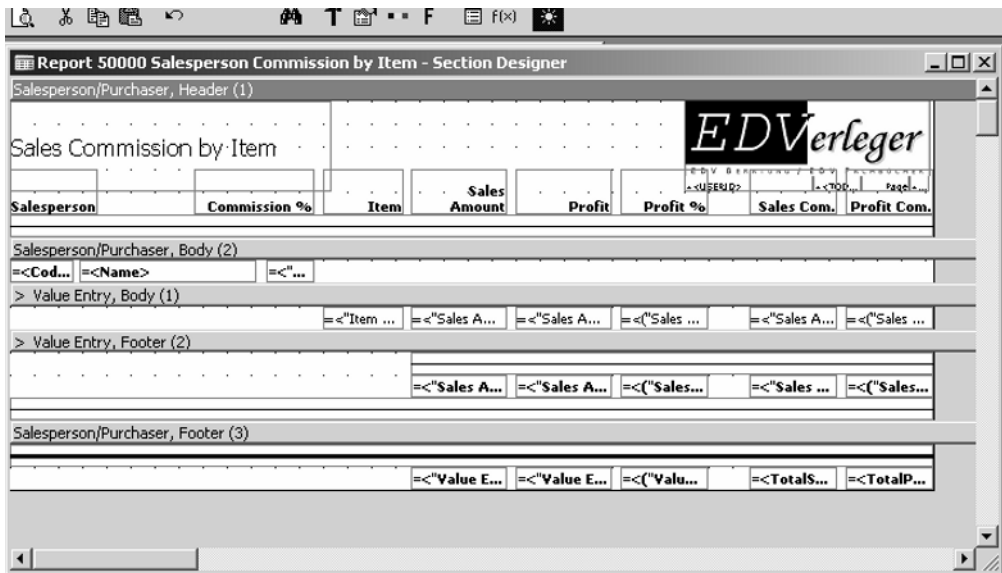
### 7.8.3 Representation of the Item Relationships

Now you must enter the new information so it appears in the report printout. Press ESC to return to the main *DataItem* window of the report. Next, go to:

- **View > Sections**

The following window will appear:

Now create a space in the layout for the *Item* name to be printed. Move all the *TextBox* boxes so they fit within the existing margins. Adjust the heading and length of the horizontal lines.



**Figure 7.7** *Section Designer* window

Open the Toolbox by going to:

- **View > Toolbox**

Enter one new *TextBox* object. Open the object properties list of the *TextBox* and enter the following values into the following properties: (You may have to move around and resize the various objects within the *Section Designer* until they fit well and according to your personal style.)

Xpos	6150
Ypos	0
Width	2550
Height	423
...	
Caption	Item
...	
HorzAlign	Left
...	
FontName	Tahoma
FontSize	6
FontBold	No
...	
SourceExpr	ItemNameTEXT

Now you will have to move around the other fields so they fit into the same horizontal space as before. Adjust the object positions until the *Section Designer* window looks like the following:

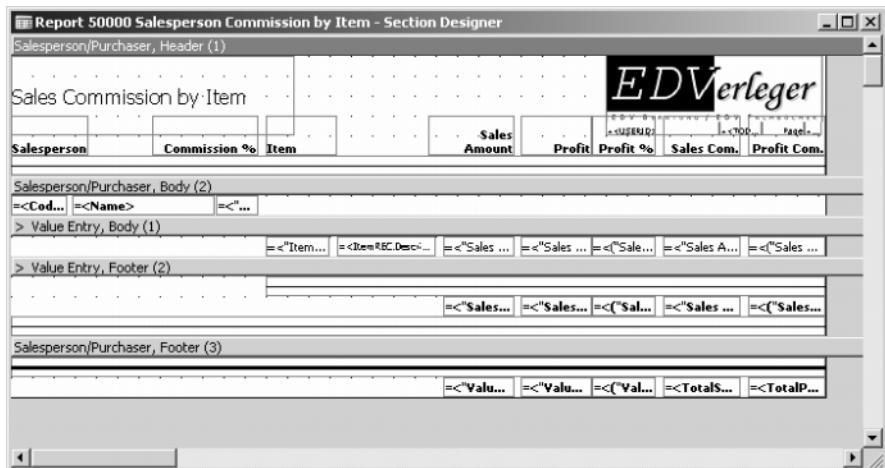


Figure 7.8 Section Designer window

Press ESC until Navision prompts you to save changes to the report and answer “Yes.” Select the report from the *Object Designer* list and click the Run button. Next click on Preview. The following *Print Preview* will appear:

**Sales Commission by Item**

EDV Verlag  
EDV BERATUNG | EDV FACHSCHAFT  
CHRIS\_KUHLWEIN 07/13/01 Page 2

Salesperson	Commission %	Item	Sales Amount	Profit	Profit %	Sales Com.	Profit Com.
70011		Glass Door	361.50	177.00	49	18.08	8.85
8908-W		Computer - Highline	342.60	342.60	100	17.13	17.13
8916-W		Computer - TURBO P	374.20	374.20	100	18.71	18.71
8924-W		Server - Enterprise P	346.30	346.30	100	17.32	17.32
8916-W		Computer - TURBO P	187.10	187.10	100	9.36	9.36
8924-W		Server - Enterprise P	346.30	346.30	100	17.32	17.32
8908-W		Computer - Highline	342.60	342.60	100	17.13	17.13
8924-W		Server - Enterprise P	346.30	346.30	100	17.32	17.32
			<b>12,120.52</b>	<b>4,846.32</b>	<b>40</b>	<b>606.03</b>	<b>242.32</b>
			<b>35,932.19</b>	<b>11,686.99</b>	<b>33</b>	<b>1,796.61</b>	<b>584.35</b>

Report generation completed (2 pages)

CHRIS\_KUHLWEIN 01/25/01

**Figure 7.9** *Print Preview* of *Item* name in report

Now you have a report that is convenient to read and gives the user complete information about *Sales Commission by Item*.

## 7.9 C/AL Calculation Functions

Now we will take a look at the functions used to calculate mathematical, date and text calculations.

### 7.9.1 Elementary Operators

The following are the basic mathematical operators used in C/AL Code:

Operator Function

- + **Addition**
- **Subtraction**
- \* **Multiplication**
- / **Division**

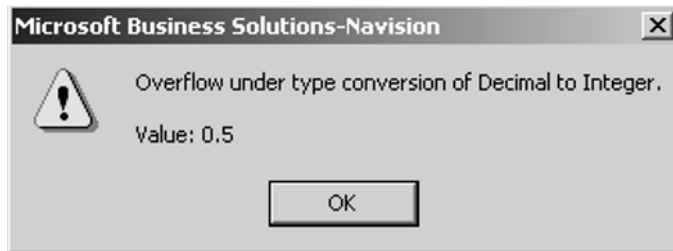
These operators can be used to calculate or compare values. They can be used between any two quantitative expressions or values.

## 7.9.2 Incompatibility Problems with Data Type

You must be careful not mix *Data Types* in way that causes a run-time error. For example, if the *Global* variable, *testINT*, is defined as *Data Type* integer and you write the following in the C/AL Code editor:

```
testINT := testINT + 0.5;
```

Navision will return no error when you press F11 to compile and test the syntax. However, when you run this program, it will crash and Navision will return the following error message:



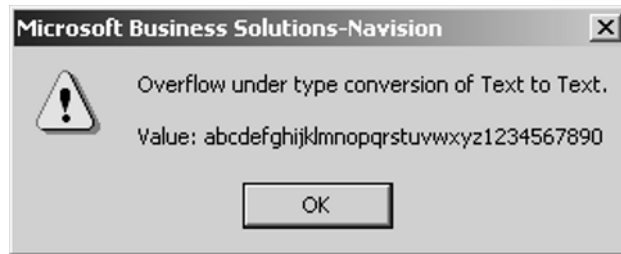
**Figure 7.10** Error message indicating *Data Type* conversion problem

Anytime you see “type conversion” in an error message, you can be sure there is an incompatibility between the *Data Type* of a variable and the information the program is trying to enter into the variable. This can also happen with text variables when you try to enter text into a text variable that has a greater length than has been defined in the *Length* column of the *Global* editor variable. If the *Global* variable, *testTEXT*, is defined in the *C/AL*

*Global* window as having a *Length* equal to 30, and you type the following code:

```
testTEXT := 'abcdefghijklmnopqrstuvwxy1234567890';
```

you will receive this error message:



**Figure 7.11** Error message indicating type conversion problem

### 7.9.3

#### POWER: Calculating Exponents

Some special mathematical operations can be accomplished with the functions POWER and ABS. Using the POWER function you can perform exponential calculations. For example, if you want to calculate the result of the number 2 multiplied against itself 3 times, you must write:

```
// Two to the third power = 8
testDEC := 2;
testDEC := POWER(testDEC,3);
```

To use the POWER function, place the number, variable or expression you wish used as a base, followed by a comma (,) and then the exponent. Enclose everything within parentheses (). The exponent number must be greater than 0. If you want to calculate the result of a negative exponent (to find the root of a number), you must calculate the reciprocal of the absolute value of your exponent as a decimal value. Then you can enter this decimal value into the C/AL Code function.

```
// The second root of four = 2
testDEC := 4;
testDEC := POWER(testDEC,0.5);
```

### 7.9.4 ABS: Using Absolute Numbers

Using the *ABS* function you can convert any value into a positive value. Simply write the keyword, *ABS*, followed by the number, variable or expression enclosed within parentheses ( ). For example:

```
testDEC := ABS(100 - testDEC);
```

Navision always enters a positive result into the variable, *testDEC*.

### 7.9.5 ROUND: Rounding Decimals

Another important, basic mathematical function is the *ROUND* function. Using *ROUND* you can control how many places around the decimal point are entered into a variable. Perhaps you want Navision to divide ten dollars by three and save the result in a money amount. The result must be rounded or Navision stores a repeating decimal. To round your calculation on the cent, write the following formula:

```
// $10 / $ 3 = $3.333333333 ???  
TestDEC := ROUND( 10/3 , 0.01 , '=' ) ;
```

The *ROUND* must be followed by parentheses ( ). Within these parentheses there are three parts separated by commas (,): the value, variable or expression to be rounded; the precision with which Navision should round the value; something that tells Navision to round upward, downward or to the nearest unit.

You can control the precision by typing in the number 1 and as many zeros (0) between the one and the decimal point as you wish to have in the resulting value. In the above example we have written 0.01—the value of one cent. Using this formula, Navision will always round to the nearest cent.

In the third part you can enter the equals symbol enclosed in apostrophes ('=') if you want Navision to round to the nearest unit of precision. This is the symbol used in the above example, meaning Navision would round to the nearest cent. A greater-than symbol enclosed within apostrophes ('>') in the third position would cause Navision to round upward while a less-than

symbol within apostrophes ( ' < ' ) would cause Navision to round downward.

## 7.9.6 CALCDATE: Calculating Dates

Now let us look at a function that can be used to calculate dates. In a previous example, we wanted Navision to step through the days between two dates and perform an action on each day. The CALCDATE function was used to accomplish this:

```
// Resetting the global variable's values.
AccChargeDEC := 0;
DailyChargeDEC := 5.25;
MovingDATE := StartDATE;

// Beginning to add daily charge for each
// day from StartDATE to EndDATE.
REPEAT
    AccChargeDEC := AccChargeDEC + DailyChargeDEC;
    // MovingDATE is one day closer to EndDATE
    // each time this line repeats.
    MovingDATE := CALCDATE('+1T',StartDATE);
// Exit this loop when it has repeated once for each day in period.
UNTIL NewDATE = EndDATE;
```

Here the *Global* date variable, *MovingDATE*, is being filled with a new date each time that Navision performs the CALCDATE function. The code between REPEAT and UNTIL will repeat until the result of the CALDATE function is equal to the date within EndDATE.

The CALCDATE function must be followed by two components surrounded by parentheses ( ). These components are separated by a comma (,). The first component is a formula that tells Navision what change it should perform on the date written as the second component. You can call the formula used in the CALCDATE function the “date expression” and the date used as the second component of the CALCDATE function the “base date.”

The date expression must be contained within apostrophes ( ' ' ). Within the date expression you must refer to time periods by writing the first letter of the name of the period. The components of the date expression are also determined by the standard language of the Navision database you are working within. For example, if you want to add one day to the base date and you are using an English version of Navision then you must write the letter 'D' to signify the time unit of a day. If you are using a Navi-

sion database whose standard language is in German then you would use the letter 'T' for the time unit of a day (*Tag*), and so forth for other languages. In an English language database you can use the following letters for the following time units:

D = Day  
M = Month  
Y = Year  
W = Week  
Q = Quarter

These date formula units can be used with values and the basic mathematical operators. Here are some examples of their use in code:

```
// One day plus one week after StartDATE  
MovingDATE := CALCDATE('+1T+1W',StartDATE);  
  
// Two months minus one day after StartDATE  
MovingDATE := CALCDATE('+2M-1T',StartDATE);  
  
// One year before StartDATE plus one day  
MovingDATE := CALCDATE('-1J+1T',StartDATE);
```

## 7.9.7

### CALCFIELDS: Controlling Flow Fields with C/AL Code

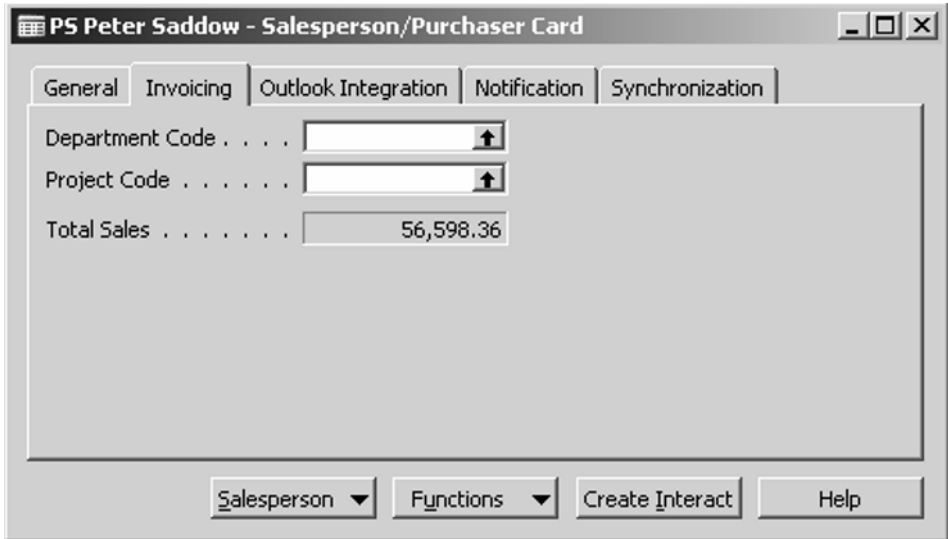
Suppose you want to use a *Flow Field* such as *Inventory* or *Sales* in a report, form or other object and you want the *Flow Field* to be controlled by some variable within the program. Until now, you have controlled *Flow Fields* exclusively by entering static text filters. To control the *Flow Field* with a variable in your program you need to use the C/AL Code function, *CALCFIELDS*. Simply tell Navision which record of which *DataItem* or *Record Global* variable to calculate. Recall that *Flow Fields* are controlled by special filters, called *Flow Filters*.

Now we will look at how to manually use a *Flow Field*. Suppose you want to use the *Flow Field*, *Sales*, created earlier in the book. Go to:

- **Sales & Marketing > Sales > Salespeople**

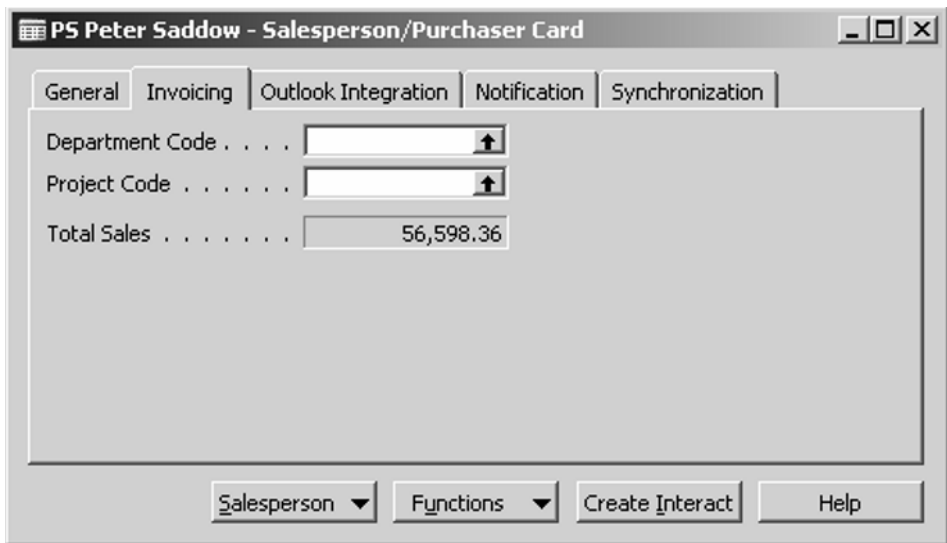


Go to the salesperson with the code *PS* and then click on the second tab, *Invoicing*. The following window will appear:



**Figure 7.12** *Salesperson/Purchaser Card* window

Suppose you want Navision to calculate the *Sales Flow Field*, but only for sales within the month of January, 2001 (the CRONUS test database only contains information for the time period 2000 to 2001). To control the dates of sales that Navision sums in the *Sales Flow Field* you must set a filter in the *Flow Filter, Date Filter*. Click on the *Flow Filter* tool and then enter the following text into the *Date Filter* Flow Field and then click OK: 01/01/01..01/31/01. Now Navision will display only the January sales of the salesperson on the *Salesperson/Purchaser Card*. The following window and corresponding *Sales* will now appear:



**Figure 7.13** Salesperson/Purchaser Card with Total Sales

Now we will take a look at how to control *Flow Filters* from within a report using C/AL Code.

Suppose that your boss is very happy with the report you made, but she wants one other extra thing. Recall that the purpose of the report is to encourage the salespeople to sell more of the items that have high returns. At present, your report allows the boss to see which items each salesperson has sold, but not the items that are still available to be sold. Your boss does not want to encourage the salespeople to sell items that are not in the inventory. Therefore, she wants the inventory of each *Item* written in the report next to the *Item* name. The variable in the *Item* table that shows inventory has the field name, *Inventory*. This field is a *Flow Field* located in the *Item* table. Calculating and showing the number of items in the inventory in the report will help your boss to better inform and encourage the salespeople to sell the appropriate items.

The inventory should be shown in the report on every line where an *Item* is shown, to the right of the *Item* name. The information concerning *Items* sold is found in a record in the *Value Entry DataItem*. Navision must calculate the inventory of an *Item* for each record in the *Value Entry DataItem*. Therefore, open the *C/AL Code Editor* for the *DataItem*, *Value Entry*. Clear all other code previously written here to test the syntax. Create the same following code in your *C/AL Code Editor* window:

```

Value Entry - C/AL Editor
Documentation()

Value Entry - OnPreDataItem()
CurrReport.CREATETOTALS("Value Entry"."Sales Amount (Actual)",
                        "Value Entry"."Cost Amount (Actual)");

Value Entry - OnAfterGetRecord()
TotalSalesComDEC := TotalSalesComDEC +
                    ("Value Entry"."Sales Amount (Actual)" *
                     "Salesperson/Purchaser"."Commission %" / 100);

TotalProfitComDEC := TotalProfitComDEC +
                    (("Value Entry"."Sales Amount (Actual)" +
                     "Value Entry"."Cost Amount (Actual)") *
                     "Salesperson/Purchaser"."Commission %" / 100);

// This clears the variable from any former uses.
ItemREC.RESET;

// Setting the sorting of the "ItemREC" record variable.
ItemREC.SETCURRENTKEY(ItemREC."No.");

// Filter the Item table on the item
// referred to in this "Value Entry" record.
ItemREC.SETFILTER(ItemREC."No.", "Value Entry"."Item No.");

// Open the Item table and find the first record
// allowed by filters and sorting. If an item is
// found, enter its name into the variable called ItemNameTEXT
// If Navision finds no match for an item it will tell the user.
IF ItemREC.FIND('-') THEN
    ItemNameTEXT := ItemREC. Description
ELSE
    MESSAGE('Item ' + "Value Entry"."Item No." +
            ' not found!');

Value Entry - OnPostDataItem()

```

**Figure 7.14** Value Entry - C/AL Editor window

Because the *Flow Field, Inventory*, exists in the *Item* table, you must link the *Item* table to the *Value Entry DataItem*. You can do this by setting up a match between the *Item No.* field, which exists in the *Value Entry DataItem*, and the field, *No.*, in the *Item* table.

If you look at the above code you will notice that you have already correctly linked the *Item* table to the *Value Entry DataItem*. This was accomplished when you retrieved the information about the *Item* name from the *Item* table. Simply tell Navision to calculate the *Inventory Flow Field* which exists in the *Item* table. Remember that the *Item* table is represented here as *ItemREC* which is a *Record Global*. This *Record Global* has already been defined in the *C/AL Globals* as linked to the *Item* table.

Now modify the code containing the *ItemREC FIND* function so that it appears as follows:

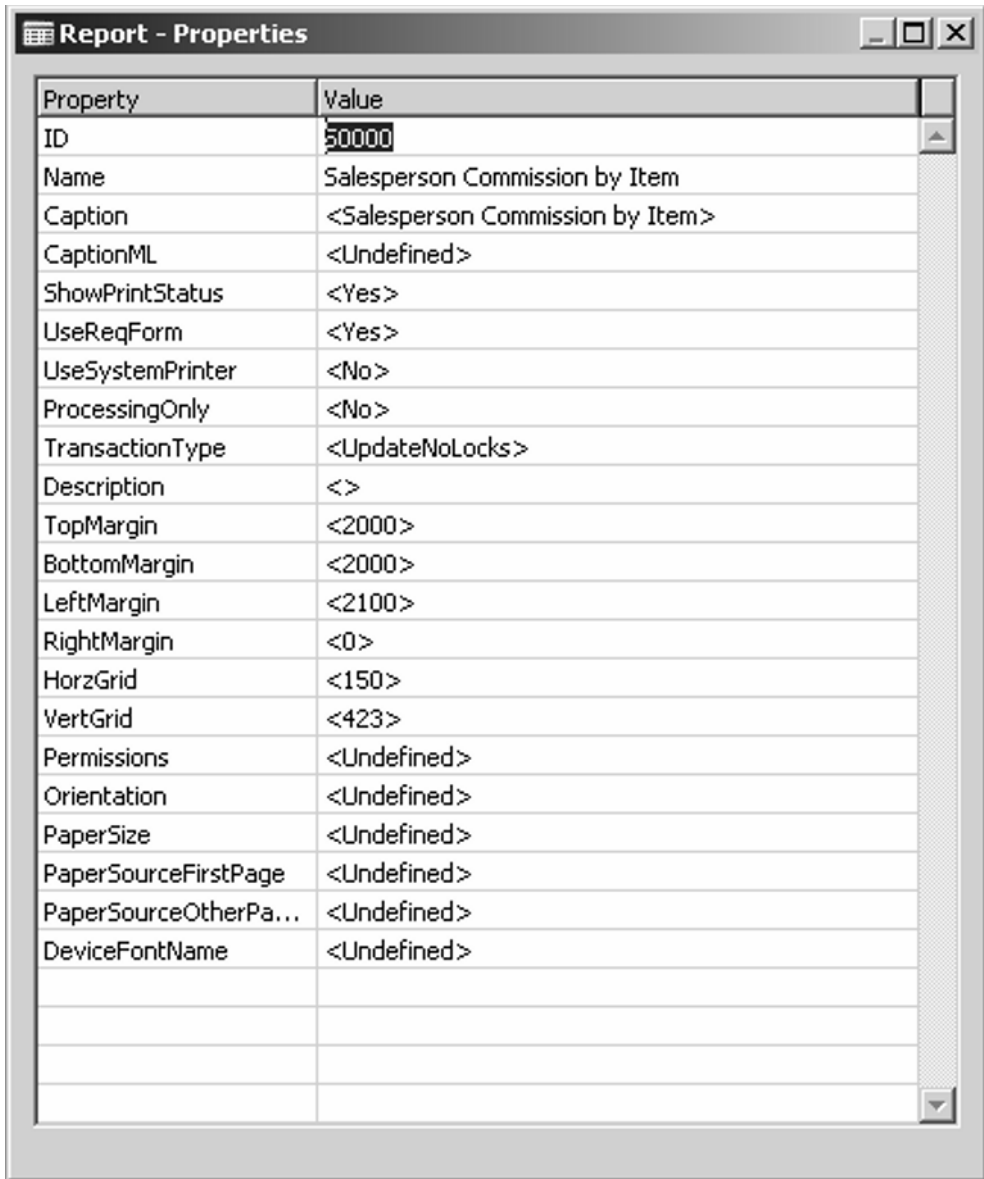
```
// Open the Item table and find the first record
// allowed by filters and sorting. If an item is
// found, enter its name into the variable called ItemNameTEXT
// If Navision finds no match for an item it will tell the user.
IF ItemREC.FIND('-') THEN
  BEGIN
    ItemNameTEXT := ItemREC. Description;
    // Inventory's calculation for this item.
    ItemREC.CALCFIELDS(Inventory);
  END ELSE
  MESSAGE('Item ' + "Value Entry"."Item No." +
    ' not found!');
```

Above is written the *DataItem* or *Record Global* name followed by a period (.), the function name, *CALCFIELDS*, and then the name of the *Flow Field* to be calculated. This field name must be enclosed within parentheses (). You can write the name of more than one *Flow Field* within the parentheses, however, you must then separate each of them by commas (,). Navision will now count the inventory of each *Item*. The inventory *Flow Field* will be influenced by any *Flow Filters* that have been set on *ItemREC*. If you want to control the inventory *Flow Field*, simply add another filter on the *ItemREC* with the *SETFILTER* function.

Now press F11 to compile the report and check for errors. Next, press ESC and then click on the empty line after the *Value Entry DataItem* in the *DataItem* list of the report. Now go to:

- **View > Properties**

The following list of global *Properties* for the report will appear:



**Figure 7.15** Report - Properties window

Because the inventory information will be added to the report, you must increase the horizontal dimension of the report print-out. Change the *Value* in the property, *Orientation*, to *Landscape*, which changes the direction of the report printout and gives you more space for information.

Now press ESC and then go to:

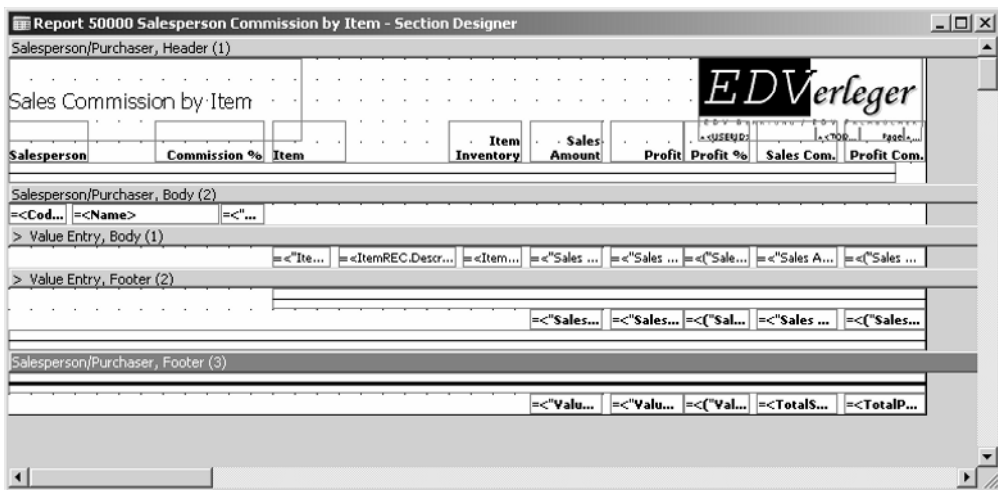
- **View > Sections**

Now add a *TextBox* object and insert it into the *Value Entry - Body (1)* section. Open the property list of this new *TextBox* object and insert the following values into the following properties:

DecimalPlaces	0:3
SourceExpr	ItemREC.Inventory

Add a *Label* object and enter the text, *Inventory* into the property, *Caption*. Now format both new objects like the others in the report sections.

Next, reorganize the layout of all the objects within the report sections to your liking and so that they fill about one and a half times the horizontal space they formerly did. Make you layout to look like the following window:



**Figure 7.16** Section Designer window

Now press ESC until Navision prompts you to compile and save the report changes. Answer “Yes” and then run the report by clicking on the Run button at the bottom of the *Object Designer*

window. Next, click Preview. The following report *Print Preview* window will appear:

**Sales Commission by Item**

EDV Verlag  
EDV BERATUNG / EDV FACHBUCHER  
CHRIS\_KUHLWEIN 07/13/01 Page 1

Salesperson	Commission %	Item	Item Inventory	Sales Amount	Profit	Profit %	Sales Com.	Profit Com.
JR	John Roberts	5						
		1972-S	MUNICH Swivel Chair,	122	739.80	163.20	22	36.99
		1968-S	MEXICO Swivel Chair,	265	493.20	108.80	22	24.66
		1980-S	MOSCOW Swivel Chair,	100	369.90	81.60	22	18.50
		1920-S	ANTWERP Conferenc	96	840.80	184.80	22	42.04
		1900-S	PARIS Guest Chair, bl	299	750.60	165.60	22	37.53
		1996-S	ATLANTA Whiteboar	181	906.70	199.50	22	45.34
		1968-S	MEXICO Swivel Chair,	265	246.60	54.40	22	12.33
		1968-S	MEXICO Swivel Chair,	265	123.30	27.20	22	6.17
		1968-S	MEXICO Swivel Chair,	265	123.30	27.20	22	6.17
		1960-S	ROME Guest Chair, gr	177	375.30	82.80	22	18.77
		1960-S	ROME Guest Chair, gr	177	250.20	55.20	22	12.51
		1960-S	ROME Guest Chair, gr	177	250.20	55.20	22	12.51
		1976-W	INNSBRUCK Storage	1	460.98	159.78	35	23.05
		1976-W	INNSBRUCK Storage	1	230.49	79.89	35	11.52
		1976-W	INNSBRUCK Storage	1	460.98	159.78	35	23.05
		70011	Glass Door	2,211	61.45	24.55	40	3.07
		1896-S	ATHENS Desk	254	649.40	142.80	22	32.47
		1906-S	ATHENS Mobile Perle	254	281.40	61.90	22	14.07

Report generation completed (2 pages)

CHRIS\_KUHLWEIN 01/25/01

**Figure 7.17** *Print Preview* of “Sales Commission by Item” report

Now the report is becoming quite powerful. At the click of a button, your report gives useful information to your boss that would otherwise take hours to prepare manually.

## 7.10

### Option: Special Data Type

Variables of *Data Type Option* have a special syntax. They are stored in the database as a number and then converted to text when called in the context of their field definition. What this means is that when you refer to an option field such as *Document Type* in *the Customer Ledger Entries*, for example, you must refer to the field name and option name explicitly. The field name of the *Document Type Option Data Type* variable is *Document Type*. Take a look at following code:

```
// Go to next record if this record is not an invoice.
IF "Document Type" <> "Document Type"::Invoice THEN
    CurrReport.SKIP;
```

Here we have referred to the *Document Type Option* by writing the field name followed by two colons (::) followed by the name of the *Option* to be tested.

## 7.11 MESSAGE and ERROR: Sending Information to the User

We have already used the MESSAGE function many times, which as you know, is used to send information to the user. A window containing an OK button and the defined text is displayed with this function. The user clicks OK and the program continues.

The ERROR function is like the MESSAGE function except the former will end the program in use. You could use an ERROR function to end a report when the program discovers that a necessary record is missing from a table. For example, in your report you could have decided that if Navision does not find an *Item* in the *Item* table for every *Item* in the *Value Entry* table, then the program should return an error message and then stop. To do this you must write into the report *Value Entry OnAfterGetRecord()* section, within the *C/AL Code Editor*, the following code:

```
// Open the Item table and find the first record allowed
// by filters and sorting If an item is found, enter its
// name into the variable called ItemNameTEXT
// If Navision finds no match for an item it will tell the user.
IF ItemREC.FIND('-') THEN
    ItemNameTEXT := ItemREC. Description
ELSE
    ERROR('Item ' + "Value Entry"."Item No." +
        ' not found.');
```

Now, if no *Item* is found in the *ItemREC*, Navision returns an error message to the user and the report quits running.

The syntax of the MESSAGE and ERROR functions includes parentheses ( ) after the function name and within the parentheses all text must be contained within apostrophes ( ' '). Variables and text fragments must have a plus sign (+) between them.

## 7.12 Data Editing Functions

You can use data editing C/AL Code functions to automate changes in information in tables. This is an incredibly powerful yet dangerous set of C/AL Code functions. If you are not careful, you could accidentally lose all the data in a table or make changes that lead to disastrous results.

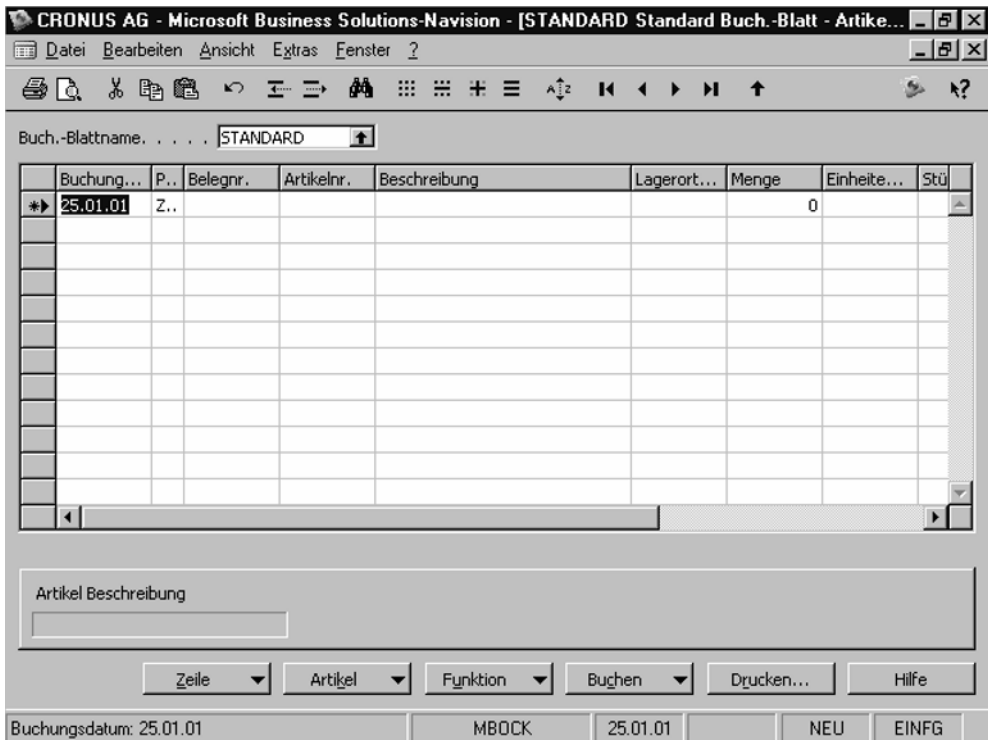


We will begin with the more friendly functions, used for creating new records, before discussing those that delete or modify data. Suppose your company has accidentally posted an entire week of false purchase orders and now the inventory is a mess. Suppose the dates of this week are from the 01/01/01 to 01/07/01.

When you want to enter something directly into the *Ledger Entries* of an *Item*, you must go to:

- **Warehouse > Inventory > Item Journals**

The following window will appear before you:



**Figure 7.18** Item Journal

To make a report that creates new lines in the *Item Journal* you must use some new C/AL Code functions. Keep in mind that creating a correct report on the first attempt is difficult because each *Journal* is complex and has its own special keys and options which are required for the posting of the lines.

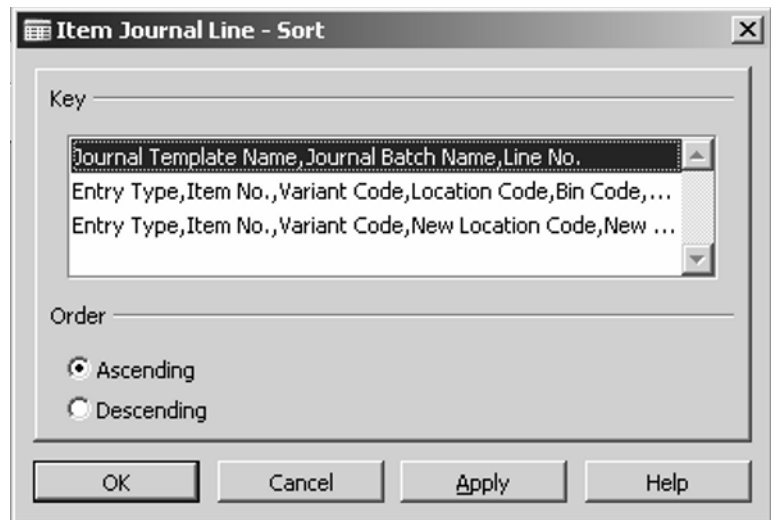
You can solve the example inventory problem by creating a report that puts a posting entry into the *Item Journal* for each line of the purchasing orders you wish cancelled. You can ask Navi-

sion to look into the movements of each *Item*, filter on only lines that are *Document Type* equal to *Purchase* and within the dates of the problem week. Now, for each line found that must be corrected, it can insert a correction posting into the *Item Journal*. When the report is finished, look into the *Item Journal* and then post everything that Navision has entered there from your report. You can correct the entire week of false purchase orders automatically.

First you must get some information about the structure of the *Item Journal*. Click on the Sorting tool or go to:

- **View > Sorting**

The following window will appear:



**Figure 7.19** *Item Journal Line - Sort Keys*

The primary key is listed first in the sorting option list. As you can see, the primary key for the *Item Journal* is made of three fields. Next, open the internal structure of this form to find the table on which it is built as well as the field names of the fields you will need to work with later. Press CTRL+F2. Next, go to:

- **View > Properties**

Find the property, *SourceTable*. Here you see that the table name is *Item Journal Line*. Press ESC and then open the *Field Menu*. Make a note of the field names of the variables that are in the table primary key as well as other variables needed to correct the postings, such as *Item No.* for example. Below is the *Field Menu*:

Field	Caption	Data Type
Journal Template Name	Journal Template Name	Code10
Line No.	Line No.	Integer
Item No.	Item No.	Code20
Posting Date	Posting Date	Date
Entry Type	Entry Type	Option
Source No.	Source No.	Code20
Document No.	Document No.	Code20
Description	Description	Text50
Location Code	Location Code	Code10
Inventory Posting Group	Inventory Posting Group	Code10
Source Posting Group	Source Posting Group	Code10
Quantity	Quantity	Decimal
Invoiced Quantity	Invoiced Quantity	Decimal
Unit Amount	Unit Amount	Decimal
Unit Cost	Unit Cost	Decimal
Amount	Amount	Decimal
Discount Amount	Discount Amount	Decimal
Salespers./Purch. Code	Salespers./Purch. Code	Code10
Source Code	Source Code	Code10
Applies-to Entry	Applies-to Entry	Integer
Item Shpt. Entry No.	Item Shpt. Entry No.	Integer
Shortcut Dimension 1 Code	Department Code	Code20
Shortcut Dimension 2 Code	Project Code	Code20
Indirect Cost %	Indirect Cost %	Decimal
Source Type	Source Type	Option
Journal Batch Name	Journal Batch Name	Code10
Reason Code	Reason Code	Code10

**Figure 7.20** *Item Journal Field Menu* window

Next you need the table name that contains the movements of each *Item*. The best place to look first is in the *Item Card*. Therefore, go to:

- **Warehouse > Planning & Execution > Items**

Next, click on the Item button and then Entries and again Entries. Next, press F2 and go to:

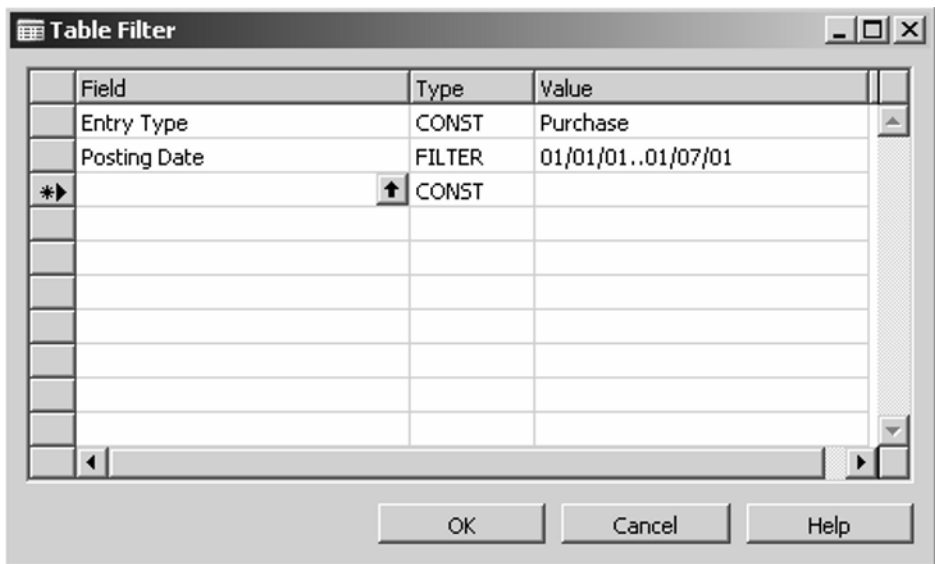
- **View > Properties**

Notice in the property, *SourceTable*, that the name of the table that contains the *Item* movement is, *Item Ledger Entry*.

Next, press ESC twice and do not save the object. Go to the *Object Designer* and click on the Report button. Next, click New at the bottom of the page to create a new report. Click “Create a blank report” and then OK. Enter the table name, *Item Ledger Entry*, into the first column of the first *DataItem* of the report the Now click on the *Item Ledger Entry DataItem* and go to:

- **View > Properties**

Click on the *DataItemTableView* property and then click on the arrow that appears to the right. Here you can limit the records that Navision accesses in each *Item* movement. Next, click on the assist, *Table Filter*, to the right of the field. The window, *Table Filter*, will appear. Enter the following fields and filters into the *Table Filter* window:



**Figure 7.21** *DataItemTableView* in *Table Filter* window

The field, *Entry Type*, is the type of *Item* movement. Here we have specified that only *Purchases* or buys are allowed. Also we have controlled the posting date which is the date when movement occurred, thus isolating the problem week.

Now click OK and then OK again. Press CTRL+S to name, number and save the report. Name the report, “Temp. Inventory Cor-

rector” plus the number 50001 (if this number is available within the parameters of your Navision license).

Next, press ESC and then go to:

- **View > C/AL Globals**

You must create a link to the table where you want Navision to put the correction postings. Type the name, *ItemJournalREC* into the first column of the *C/AL Globals* window. In the second column, type *Record* and in the third column, write the table name of the *Journal*, *Item Journal Line*. Now you need a counting variable so Navision can sort the records in the *Item Ledger Entry* table. Type *INT* in the first column and then *Integer* in the second column.

Now press ESC and then click on the *DataItem Item Ledger Entry* in the report *DataItem* list window. Now go to:

- **View > C/AL Code**

Now type the following C/AL Code in the *C/AL Editor* section, *Item Ledger Entry - OnAfterGetRecord()*:

```
// Close record access by other programs.
ItemJournalREC.LOCKTABLE;
```

The function, LOCKTABLE, protects the record in a table from being changed by another user or program while you are editing it. To do so, write the name of the *DataItem* or *Record Global* which has the table you want shielded, then a period (.) and then type LOCKTABLE.

```
// Counting number used to sort the new
// records in the posting journal.
INT := INT + 1;
```

This line allows the integer *Global* variable to increase with each new record in the *DataItem*. Such a counter is useful for filling the primary key of *Journals* which is an increasing integer.

```
// Open a new record in the posting table.
ItemJournalREC.INIT;
```

This above C/AL Code uses the function, INIT. When you want to create a new record in the *Journal* table you must first tell

Navision to open a new record, fill the relevant fields with information and then finally, insert the new record into the table. The opening of a new record is accomplished by naming the *DataItem* or *Record Global* followed by a period (.) and then typing *INIT*.

```
// Filling primary key fields of our new record.
ItemJournalREC.VALIDATE("Journal Template Name",
                        'ITEM');
ItemJournalREC.VALIDATE("Journal Batch Name",
                        'STANDARD');
ItemJournalREC.VALIDATE("Line No.",INT);
```

We have filled the values of the fields of the primary key in of the new record. The *VALIDATE* function, used here, is a special way to enter information into a table; it runs any C/AL Code instructions that exist in the C/AL Code section, *OnValidate()*, within the field.

Table fields have two C/AL Code sections: *OnValidate()* and *OnLookup()*. If you want to see the C/AL Code of a particular field, open the table, click on the field you want to see, then go to:

- **View > C/AL Code**

When using the *VALIDATE* function, Navision takes the information you have just entered into the field to run the *OnValidate* C/AL Code. The usefulness of this function is clear in the following example: Suppose you enter an *Item* number into a table. Navision can automatically fill a field with the *Item* name. This automatic filling in of the *Item* name can be written into the *OnValidate()* C/AL Code section within the *Item* number field. Such an automatic function—which occurs in many places in Navision—is accomplished by writing C/AL Code into the *OnValidate* C/AL Code section of fields themselves. You should always look at the C/AL Code behind fields if you want to understand how they are reacting to information entered into them.

In a *Journal* there are many keys as well as linked information between the various fields, however, they may be hidden in the end user form. When you manually enter information into a field, Navision automatically runs any “C/AL Code” that may exist in the *OnValidate()* section of the field. Therefore, you must also run the *OnValidate()* section each time information is entered into a field. The *VALIDATE* function simply does what the

user would do by pressing ENTER after entering information into a field.

The syntax of this function is the *DataItem* or *Record Global* name of the table followed by a period (.) and the keyword, VALIDATE. After the keyword follows parentheses ( ) containing the field name, then a comma (,) followed by the information to be entered into the field. If you enter text into a field you must enclose it within apostrophes ( ' '). If you are entering an option into a field of the *Option Data Type*, then you must state the field name followed by two colons (::) and then the option. Below is an example:

```
// Define item movement type to be a removal from inventory.
ItemJournalREC.VALIDATE("Entry Type",
    ItemJournalREC."Entry Type>::"Negative Adjmt.");
```

In the code above, the type of movement is defined by typing the option field name followed by two colons (::) and then the option. In this example, *Negative Adjmt.* has been specified, which means that something is being taken away from the inventory.

```
// Enter new journal line item number and date.
ItemJournalREC.VALIDATE("Item No.",
    "Item Ledger Entry"."Item No.");
ItemJournalREC.VALIDATE("Posting Date",
    "Item Ledger Entry"."Posting Date");
```

Here we have simply copied the *Item* number and posting date from the *Item Ledger Entry* table and entered it into the new record in the *Item Journal*.

```
// Enter a document number as LBKORR + the journal
// line number and description.
ItemJournalREC.VALIDATE("Document No.",
    'LBKORR' + FORMAT(INT));
ItemJournalREC.VALIDATE(Description,
    'Inventory Correction');
```

Above we created a document number that stands for the description *Inventory Correction* followed by the line number contained in the *INT Global* integer.

```
// Remove false quantity from inventory.  
ItemJournalREC.VALIDATE(ItemJournalREC.Quantity,  
    "Item Ledger Entry".Quantity);
```

Below is the line where we enter the quantity of the false posting to be removed from the inventory.

```
// Connect correction journal line to the false item journal entry.  
ItemJournalREC.VALIDATE("Applies-to Entry".  
    "Item Ledger Entry"."Entry No.");
```

Every posting within the *Item Ledger Entries* has a line number called, *Entry No.* When possible, tell Navision what line in the *Item Ledger Entries* you want corrected. This is a tricky function but it helps keep the journal cleaner by closing previously corrected or used entries.

```
ItemJournalREC.INSERT;
```

Finally, type the name of the *DataItem* or *Record Global* where Navision must insert the new record followed by a period (.) then the keyword, *INSERT*. The *INSERT* function now completes the process began by the *INIT* function and enters the newly constructed record in the table connected to *ItemJournalREC*.

The *C/AL Code Editor* of the *Item Ledger Entry DataItem* now looks like the following:



```

Documentation()

Item Ledger Entry - OnPreDataItem()
// Close record access by other programs.
ItemJournalREC.LOCKTABLE;
// Counting number used to sort the new
// records in the posting journal.
INT := INT + 1;
// Open a new record in the posting table.
ItemJournalREC.INIT;
// Filling primary key fields of our new record.
ItemJournalREC.VALIDATE("Journal Template Name",
    'ITEM');
ItemJournalREC.VALIDATE("Journal Batch Name",
    'STANDARD');
ItemJournalREC.VALIDATE("Line No.",INT);
// Define item movement type to be a removal from inventory.
ItemJournalREC.VALIDATE("Entry Type",
    ItemJournalREC."Entry Type":="Negative Adjnt.");
// Enter new journal line item number and date.
ItemJournalREC.VALIDATE("Item No.",
    "Item Ledger Entry"."Item No.");
ItemJournalREC.VALIDATE("Posting Date",
    "Item Ledger Entry"."Posting Date");
// Enter a document number as LBKORR + the journal
// line number and description.
ItemJournalREC.VALIDATE("Document No.",
    'LBKORR' + FORMAT(INT));
ItemJournalREC.VALIDATE(Description,
    'Inventory Correction');
// Remove false quantity from inventory.
ItemJournalREC.VALIDATE(ItemJournalREC.Quantity,
    "Item Ledger Entry".Quantity);
// Connect correction journal line to the false item journal entry.
ItemJournalREC.VALIDATE("Applies-to Entry",
    "Item Ledger Entry"."Entry No.");
ItemJournalREC.INSERT;

```

**Figure 7.22** Item Entry Ledger - C/AL Editor window

Press CTRL+S to compile and save your work. Next, change the general properties of the report. Press ESC and then click on the empty line below the last *DataItem* in the *DataItem* list of the report and then go to:

- **View > Properties**

In the *Report - Properties* window change the property, *ProcessingOnly*, to *Yes* since you do not want to have any printed output from this data processing report.

Compile, save and exit the report. Now run the report and then go to:

- **Warehouse > Inventory > Item Journals**

Here you see the new lines that have been placed there by our report. The following window will be before you:

Buchung...	P	Belegnr	Artfuehr	Beschreibung	Lagerort...	Menge	Einheit...	Stückpreis	Betrag	Einstand...	Ausgleic...
01.01.01	A..	LBKORR1	1964-S	Lagerbestand Korrektur	GRÜN	14	STÜCK	139,494	1.952,92	139,494	129
03.01.01	A..	LBKORR2	1964-W	Lagerbestand Korrektur	GRÜN	15	STÜCK	265,20	3.978,00	265,20	130
03.01.01	A..	LBKORR3	1964-W	Lagerbestand Korrektur	BLAU	25	STÜCK	265,20	6.630,00	265,20	131
05.01.01	A..	LBKORR4	70060	Lagerbestand Korrektur	ROT	250	STÜCK	9,836	2.459,00	9,836	136
05.01.01	A..	LBKORR5	70060	Lagerbestand Korrektur	BLAU	500	STÜCK	9,836	4.918,00	9,836	137
05.01.01	A..	LBKORR6	70011	Lagerbestand Korrektur	BLAU	52	STÜCK	57,133	2.970,92	57,133	138

Artikel Beschreibung  
TOKYO Gästestuhl, blau

Zelle | Artikel | Funktion | Buchen | Drucken... | Hilfe

Buchungsdatum: 01.01.01 | MBOCK | 25.01.01 | EINFG

**Figure 7.23** *Item Journal* output

All that remains to be done is to look over the results and click F11 and OK to post the corrections. You have now created a report that automatically creates new records and places them within a table. The techniques used here can be used to create new records in all Navision tables. Most tables are not as complex as the *Item Journal*, therefore, if you understand this example you should be able to adapt it to more general uses throughout Navision.

The MODIFY and DELETE functions can be used either to change information in a record or to erase that record completely. These functions are easy to use and very powerful. However, they should be used with caution as they can have irreversible results.

Suppose that you wish to change the price of every *Item*. You could create a new report containing the *Item* table as a *DataItem* and write the following code within the *OnAfterGetRecord()* of the *C/AL Code Editor*:

```
// Increase prices by 10%
Item."Unit Price" := Item."Unit Price" * 1.1;
Item.MODIFY;
```

When you run this report Navision automatically increases the price of every *Item* by 10%.

Suppose you want to erase all *Items* that are blocked. You can type the following code in the report:

```
// Erase items no longer in use.
IF Item.Blocked = TRUE Then Item.DELETE;
```

To use these functions, simply type the name of the *DataItem* or the *Record Global* followed by a period (.) then the keyword.

# 8

## Using Dataports to export & import Navision Data

We encounter some of our most challenging problems when trying to integrate two different softwares; for example, when we want Navision to work with shipment processing software. Up until now, we have concentrated on handling data within Navision. Integration implies getting data in and out of Navision and for this work we will need to use a new type of object, a *Dataport*.

### 8.1 Simple Export from Navision

Let us create a basic export from Navision. The *Dataport* objects allow you to easily export data from Navision and save it as a text file.

Imagine a scenario in which you export a list of *Items* from the *Sales Shipment Line* table. This list should include things to be read by shipment software. You can create a *Dataport* to accomplish this task. Open the *Dataport* list via:

**T Extras > Object Designer > Dataport**

Select New and enter *Sales Shipment Line* as the first *DataItem*.

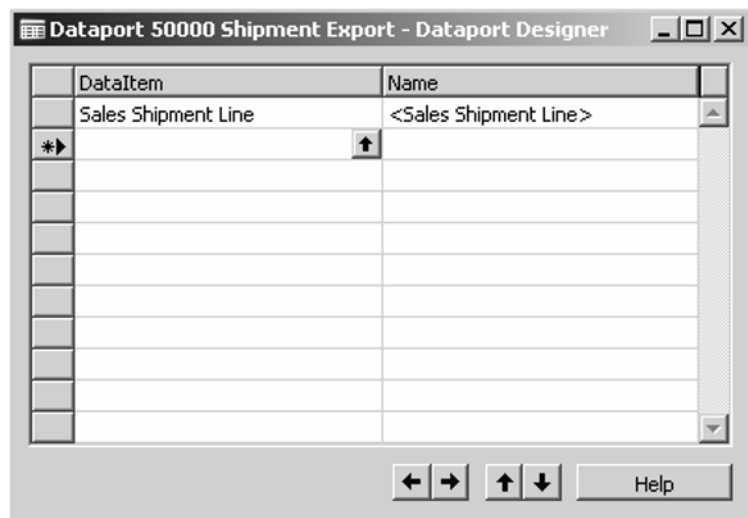


Figure 8.1 *Dataport Designer* window

Press CTRL+S and save your *Dataport* as *Shipment Export*.

Now click on the *Sales Shipment Line* and then select from the uppermost menu:

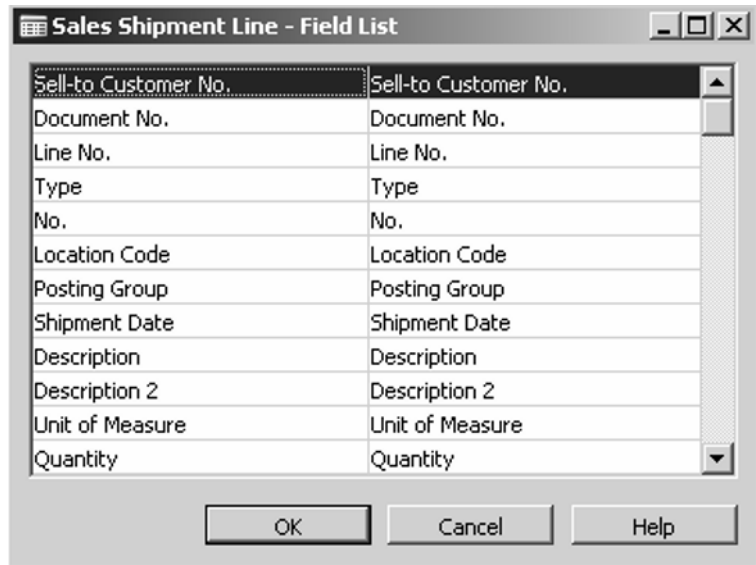
**T View > Dataport Fields**

The following window will appear:



**Figure 8.2** *Sales Shipment Line – Field Designer* window

Click on the *SourceExpr* column and then click on the assist that appears (or press F6 after clicking on the column). The following window will appear:

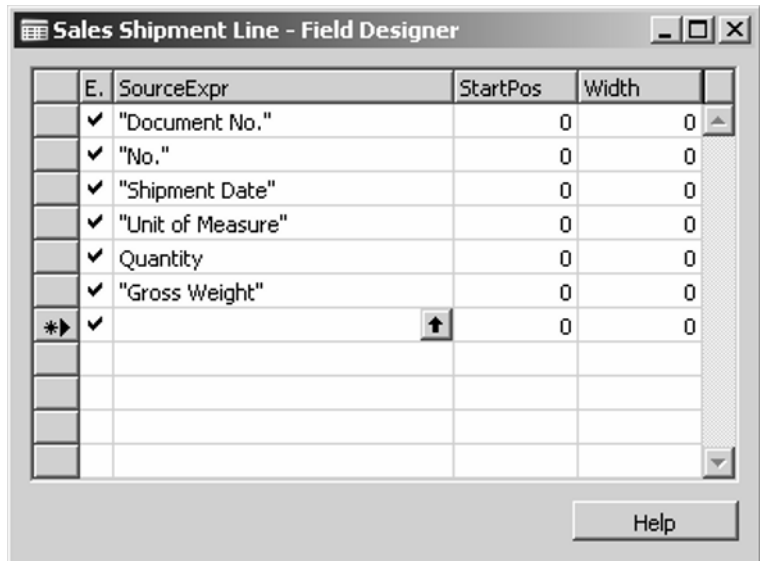


**Figure 8.3** *Sales Shipment Line – Field List* window

Above is a list of all the fields available within the *Sales Shipment Line* table. These fields are available because this table has been entered into the *DataItem*. Now you can double-click on any field you want exported from the *Sales Shipment Line* table. When you double-click on a field it is entered into the *Dataport Field List*. Simply repeat this for as many fields as you wish.

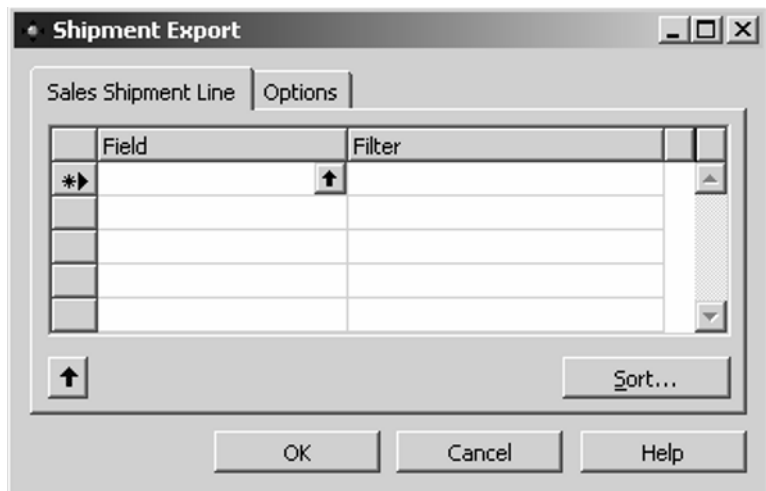
For this demonstration select the following: *Document No.*, *No.*, *Shipment Date*, *Unit of Measure*, *Quantity* and *Gross Weight*.

Your window should now look like the following:



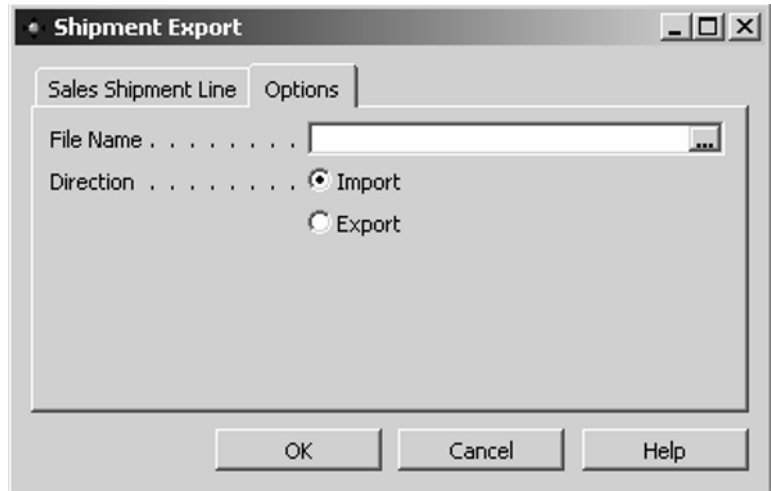
**Figure 8.4** Sales Shipment Line – Field Designer window

Press ESC to return to your *DataItem* window. Press ESC again to save and close the *Dataport*. Now select the *Dataport 50000* and click Run. The following window will appear:



**Figure 8.5** Shipment Export window

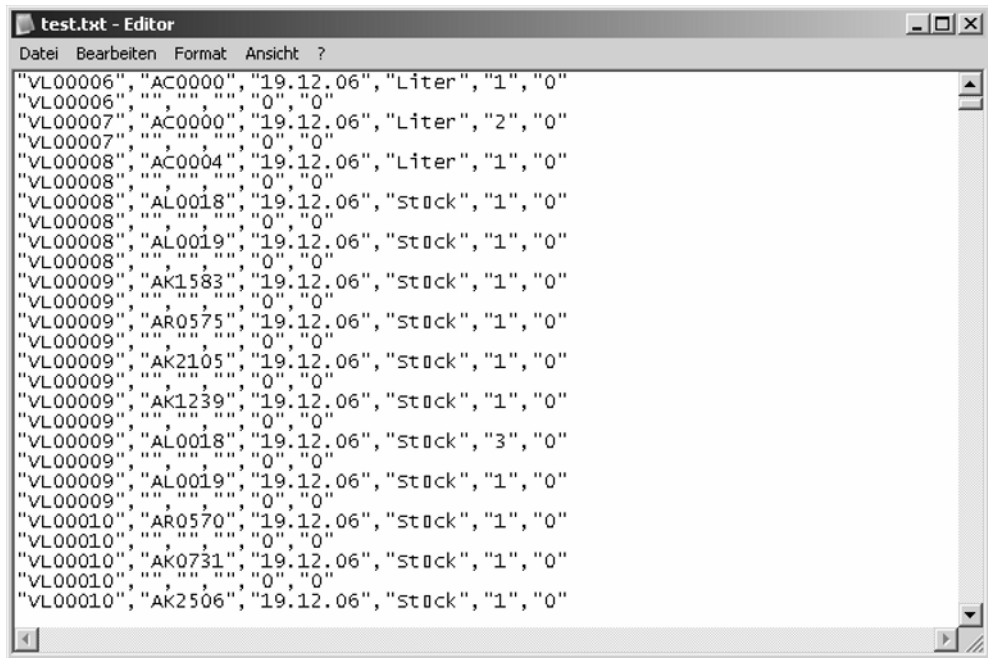
This window is similar to the Microsoft Navision report-run window. The first tab represents the *DataItem* created within the *Dataport*. Here you can manually define filters to limit the *Sales Shipment Lines* that are exported. The *Options* tab looks as follows:



**Figure 8.6** *Shipment Export* with *Options* tab open

Here you must select the option, Export. Enter a file name where you want the output to be saved and click OK. Next, find and open the file where you saved it. It should look something like the following (this is sample content which you will not find in your database):





**Figure 8.7** Text file in *Editor* window

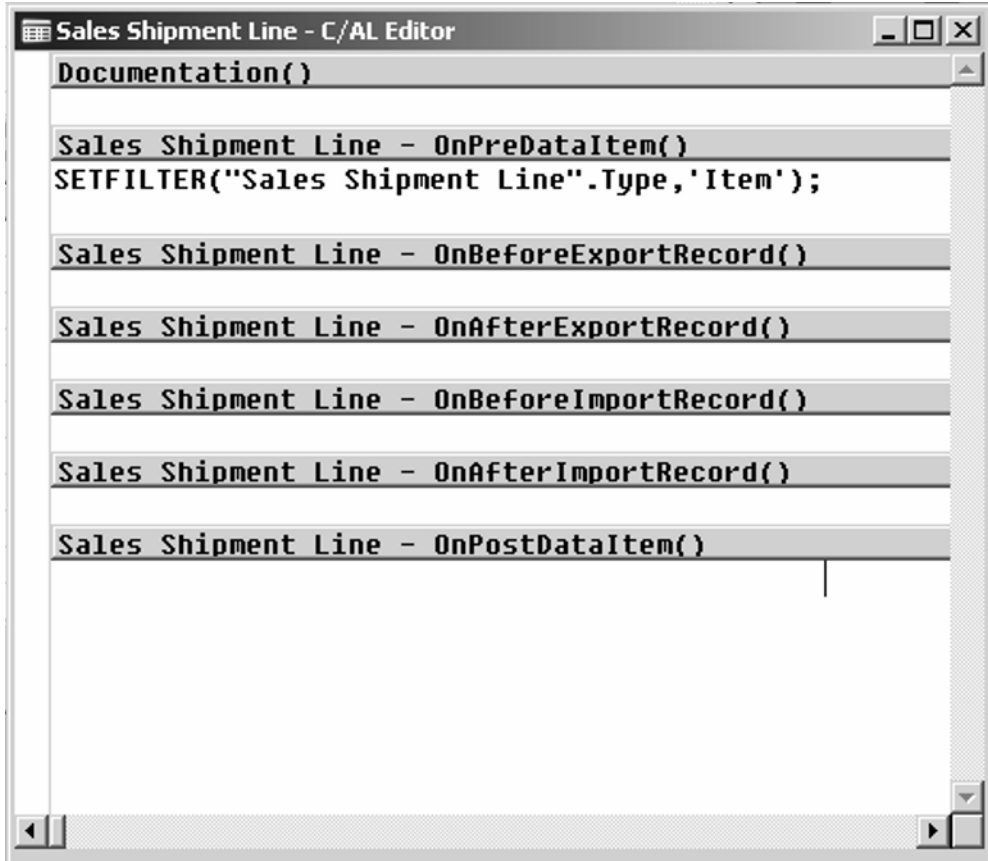
In this example, the *Document No.* always begins with 'VL' and the *Item* always begins with 'A.' Notice that not all lines have an *Item* in them. This is due to the fact that not all lines in sales orders are *Items*—some are merely text lines, others may be financial accounts, etc. Also note the format of this output: each *Sales Shipment Line* is a new line and the contents of each field are contained within quotation marks (" "). Each field is separated by a comma (.). These are all things within your control. Now return to the *Dataport* and change some of these format conditions.

First, filter out *Sales Shipment Lines* that do not contain *Items*. Once again, select the *Dataport 50000* and then click the Design button in the *Object Designer*. Click on the *DataItem* and then press F9, which opens the C/AL Code window for the *Sales Shipment Line DataItem*. Using the *SETFILTER* function (discussed in Chapter 8), you can filter lines.

In the *OnPreDataItem* trigger write:

```
SETFILTER("Sales Shipment Line".Type,'Item');
```

This should appear in your window as follows:



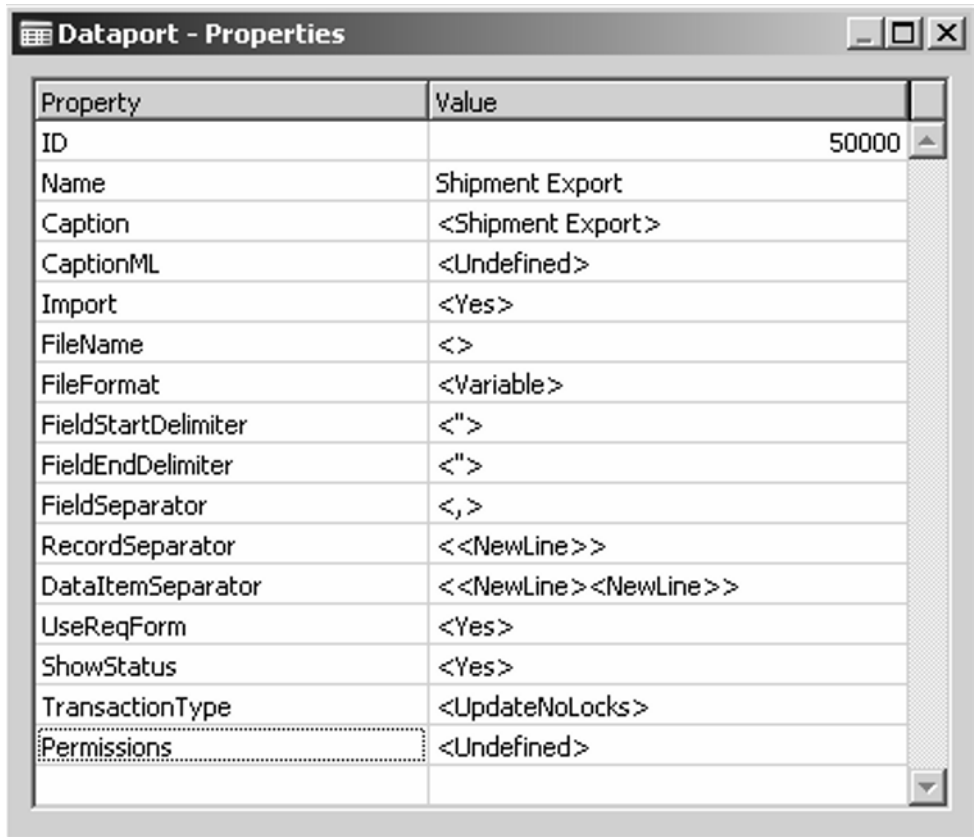
**Figure 8.8** Sales Shipment Line – C/AL Editor window

Press ESC to close the *C/AL Editor*.

Next, change the format of the text file you created. To do this you must edit the general *Properties* of the *Dataport*. Click the first empty line below the *DataItem*. Now press SHIFT+F4 or via menu, go to:

**T View > Properties**

The following window will appear:



**Figure 8.9** *Dataport – Properties window*

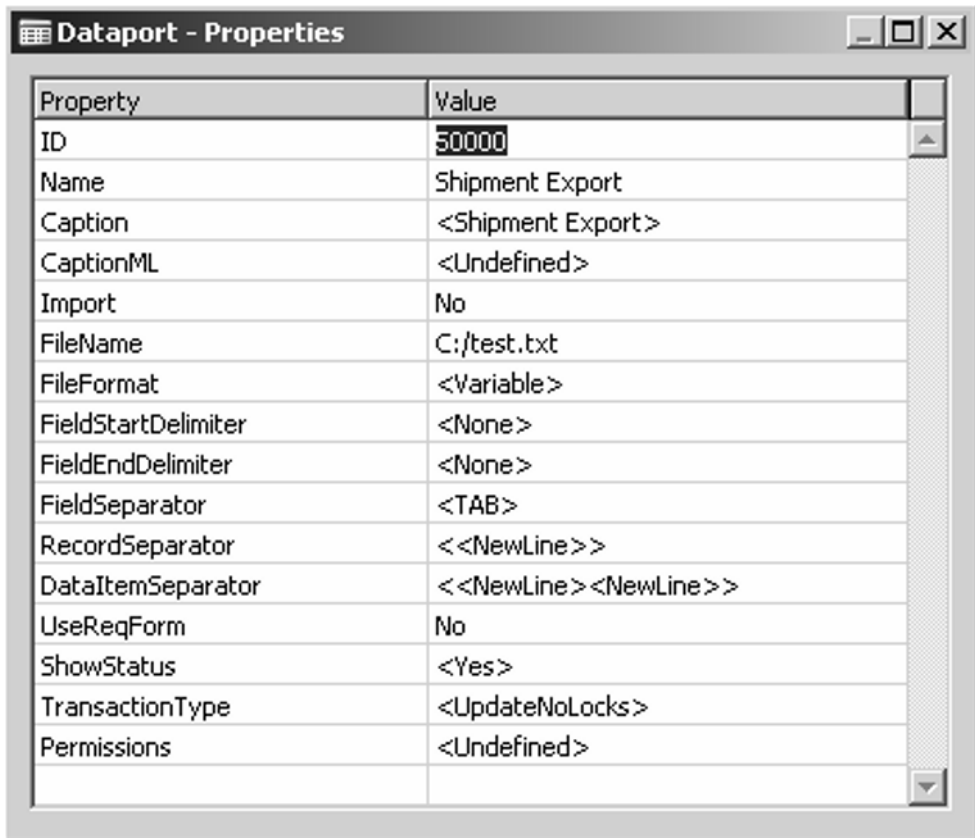
Above you see the general *Properties* for *Dataport*. You must make a few changes here. First, you want this object for export so you can change the line *Import* to *No*. Now define the place and name of the file that the *Dataport* will create by entering C:/test.txt in the *FileName* line. Now the user will not be able to choose a file name themselves. Next, change the format of the output text file and enter *No* in the line *UserRequestForm*. Doing so eliminates the option window when the user runs the *Dataport*.

Strive to use minimal formatting; commas (,) and quotation marks (“ ”) can cause great confusion if they also exist within the

fields you output. Imagine what would happen if you tried to export a customer with the name, *Henry "Buddy" Smith*, in a comma delimited file. The comma and quotation marks within the customer name would be interpreted as the separation between two different fields and likely to creating two customers when in fact there is only one!

Enter <None> in the *FieldStartDelimiter* and into the *FieldEndDelimiter*. Enter <TAB> in the *FieldSeperator* line. This will create what is called a *Tab Delimited* text file. This format is easy to work with in other software programs such as Microsoft Excel.

Your *Dataport Properties* should now look like the following:



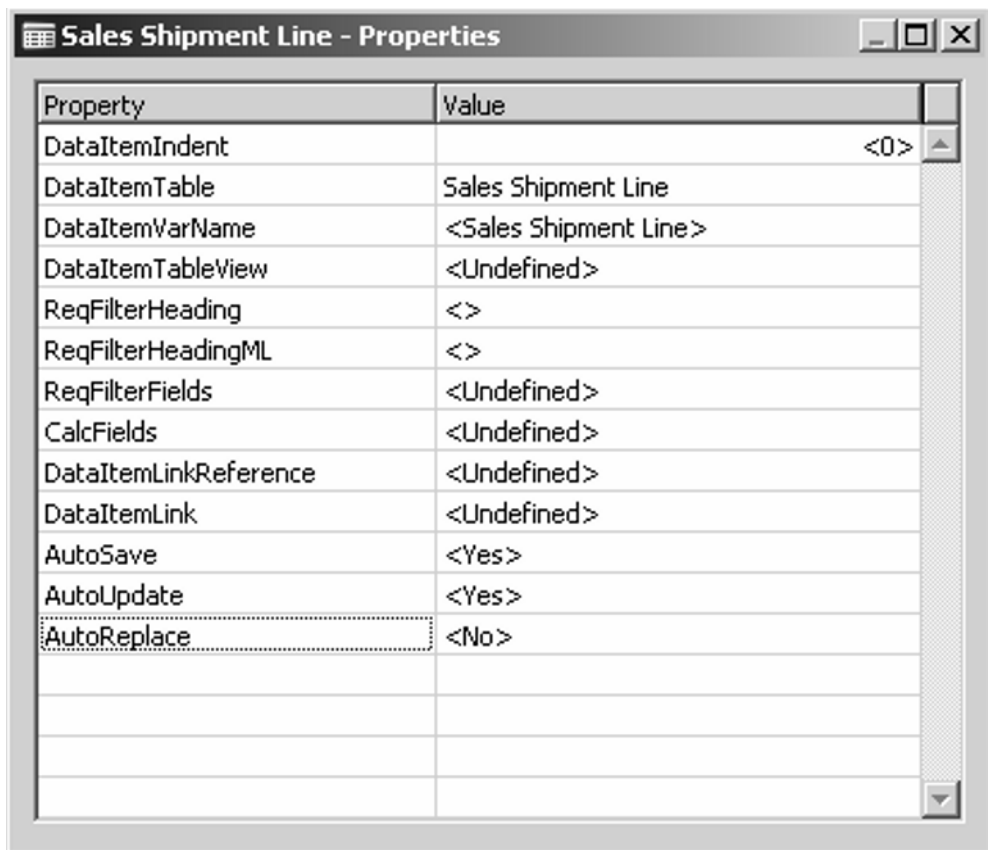
**Figure 8.10** *Dataport - Properties* window

Suppose you do **not** want to give the end user options. For example, when you enter something into the *FileName* property, the end user will not be given the option to define a file name at run-time. To eliminate the *DataItem* tab, you must specify a default sorting and no default filters in the *Sales Shipment Line DataItem*.

Press ESC to return to the *Dataport DataItem* window. Click on the *Sales Shipment Line* and then press SHIFT+F4 or go to:

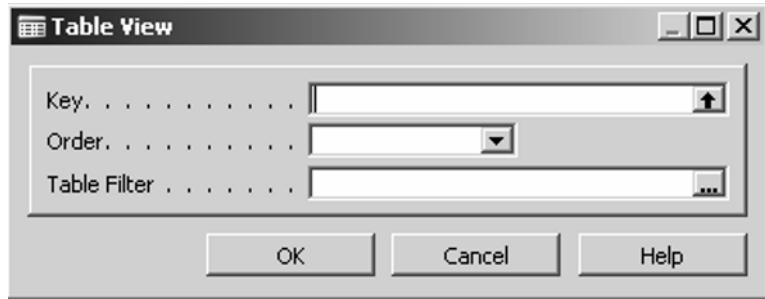
### T Menu View > Properties

The following window will appear:



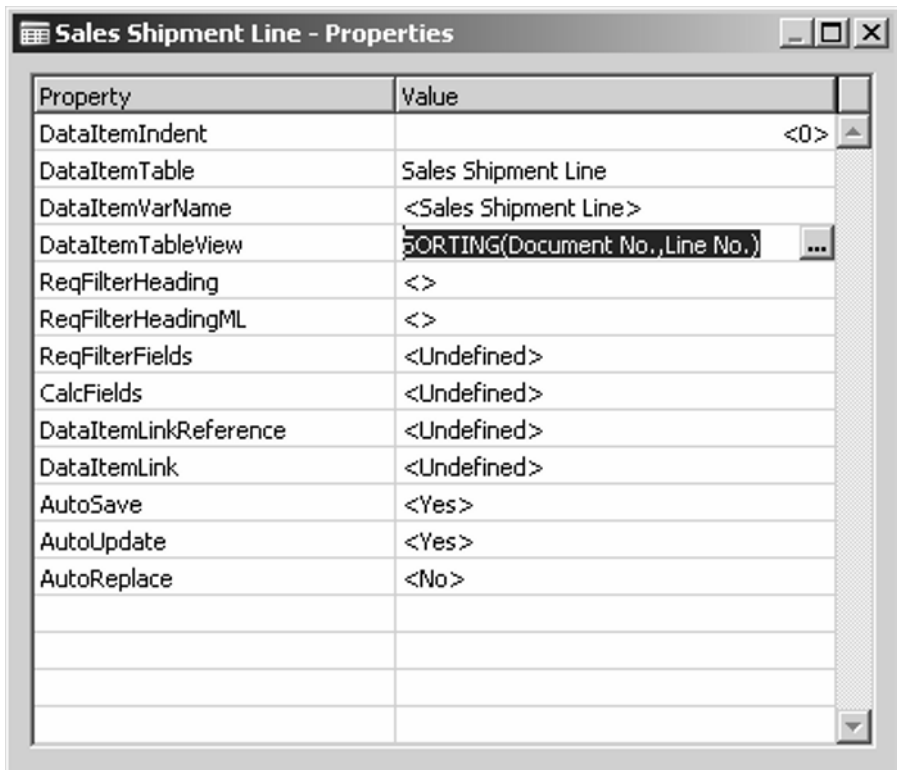
**Figure 8.11** *Sales Shipment Line – Properties* window

In the *DataItemTableView* line press F6 or click on the assist in the line. The following window will open:



**Figure 8.12** *Table View* window

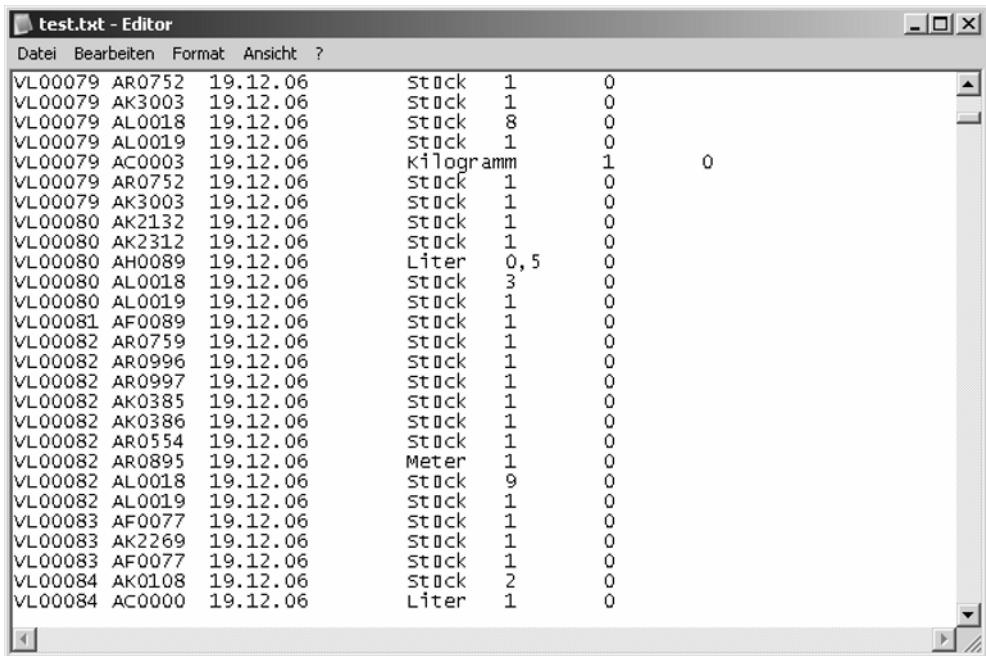
Click into the *Key* field. Next, double-click on the first key in the list that appears. Press OK. The *Properties* should look like the following:



**Figure 8.13** *Sales Shipment Line – Properties* window

Press ESC twice then save and run the *Dataport*. When you run it you will see that no options are given to you; the *Dataport* runs according to conditions predefined within the development environment.

Now view the text file you created. Look on the local C:/ drive of your PC for a file called, test.txt, and open it. The following window should appear:



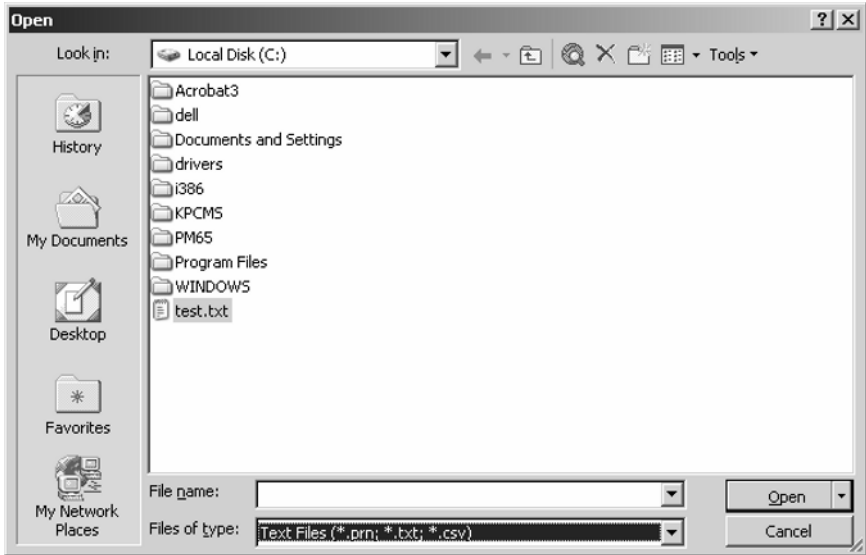
Item	Date	Unit	Quantity
VL00079 AR0752	19.12.06	Stöck	1 0
VL00079 AK3003	19.12.06	Stöck	1 0
VL00079 AL0018	19.12.06	Stöck	8 0
VL00079 AL0019	19.12.06	Stöck	1 0
VL00079 AC0003	19.12.06	Kilogramm	1 0
VL00079 AR0752	19.12.06	Stöck	1 0
VL00079 AK3003	19.12.06	Stöck	1 0
VL00080 AK2132	19.12.06	Stöck	1 0
VL00080 AK2312	19.12.06	Stöck	1 0
VL00080 AH0089	19.12.06	Liter	0,5 0
VL00080 AL0018	19.12.06	Stöck	3 0
VL00080 AL0019	19.12.06	Stöck	1 0
VL00081 AF0089	19.12.06	Stöck	1 0
VL00082 AR0759	19.12.06	Stöck	1 0
VL00082 AR0996	19.12.06	Stöck	1 0
VL00082 AR0997	19.12.06	Stöck	1 0
VL00082 AK0385	19.12.06	Stöck	1 0
VL00082 AK0386	19.12.06	Stöck	1 0
VL00082 AR0554	19.12.06	Stöck	1 0
VL00082 AR0895	19.12.06	Meter	1 0
VL00082 AL0018	19.12.06	Stöck	9 0
VL00082 AL0019	19.12.06	Stöck	1 0
VL00083 AF0077	19.12.06	Stöck	1 0
VL00083 AK2269	19.12.06	Stöck	1 0
VL00083 AF0077	19.12.06	Stöck	1 0
VL00084 AK0108	19.12.06	Stöck	2 0
VL00084 AC0000	19.12.06	Liter	1 0

**Figure 8.14** Text file in *Editor* window

In this output, you see that all lines contain an *Item*—found in the second column and always beginning with ‘A’. The quotation marks (“ ”) and commas (,) are gone and tabs have been used to separate the columns. Now close this file.

## Opening Files in Microsoft Excel

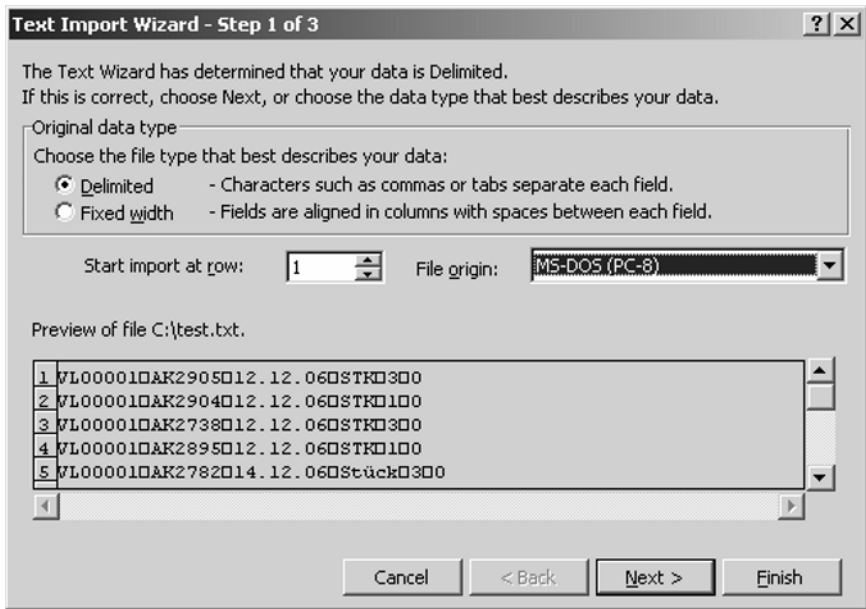
Open Microsoft Excel and click on the menu point, File, and then Open. In the Excel Open file window you must change the file type to Text File. You will see the following window:



**Figure 8.15** Open file window in Microsoft Excel

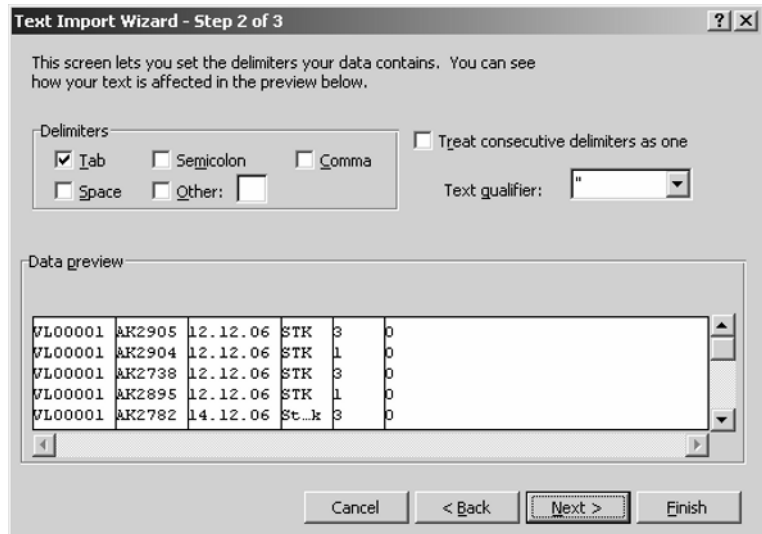
Double-click on the *test.txt* file. The following will open:





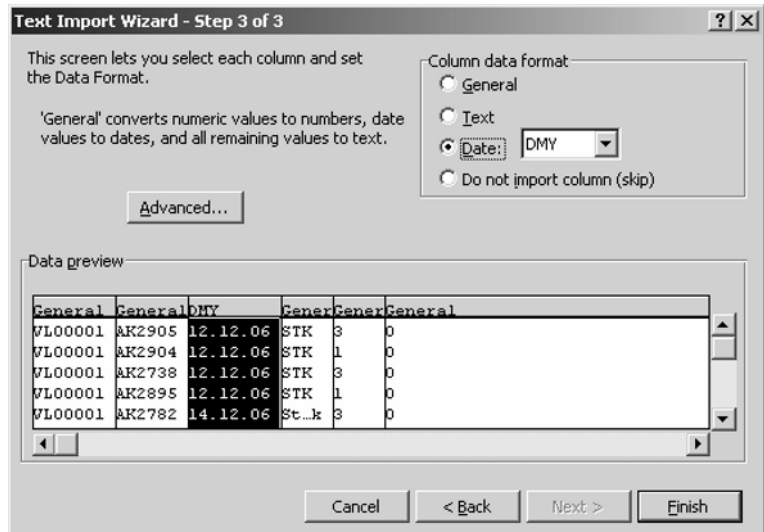
**Figure 8.16** *Text Import Wizard*, Step 1 window

Choose the *Delimited* and the *File* origin in accordance with the language of the file. Click *Next* and the following window will appear:



**Figure 8.17** *Text Import Wizard*, Step 2 window

Choose *Tab* as the delimiter and then click Next. The following window will appear:



**Figure 8.18** *Text Import Wizard*, Step 3 window

Click on the column containing dates. Next, select *Date* in the *Column data format* option. Here you must choose the date format. In this example, day, month and year (DMY) has been selected. Click Finish. The wizard will close and now your Navision data appears in Excel (See below).

	A	B	C	D	E	F	G
28	VL00006	AC0000	19.12.2006	Liter		1 0	
29	VL00007	AC0000	19.12.2006	Liter		2 0	
30	VL00008	AC0004	19.12.2006	Liter		1 0	
31	VL00008	AL0018	19.12.2006	Stück		1 0	
32	VL00008	AL0019	19.12.2006	Stück		1 0	
33	VL00009	AK1583	19.12.2006	Stück		1 0	
34	VL00009	AR0575	19.12.2006	Stück		1 0	
35	VL00009	AK2105	19.12.2006	Stück		1 0	
36	VL00009	AK1239	19.12.2006	Stück		1 0	
37	VL00009	AL0018	19.12.2006	Stück		3 0	
38	VL00009	AL0019	19.12.2006	Stück		1 0	
39	VL00010	AR0570	19.12.2006	Stück		1 0	

**Figure 8.19** Text file in Microsoft Excel

This concludes the example of a simple export.

## 8.2 Complex Export from Navision

Suppose you want to export information from more than one table in your *Dataport*. Or, suppose you need the export to contain the results of calculations. Accomplishing this will not be as easy as merely entering fields directly from the *DataItem* into the *Dataport Fields*.

To solve this problem—and others not yet mentioned—you must introduce the use of *Global Variables* into the *Dataport*.

Now imagine there are additional requirements for the Sales shipment export, such as the customer shipping address and the weight of the packaging material.

Open *Dataport 50000* and make these necessary changes. Click on the *DataItem Sales Shipment Line* and then open the *Field Menu* by going to:

**View > Field Menu**

As you look through the *Field Menu* you will see no address information within the table. You must, rather, find the shipment address in the *Sales Shipment Header* table. You can connect these two tables via the *Document No.*

Now open the *Sales Shipment Header* table—number 110. Click on the Table button in the *Object Designer*. Find object 110 and then click on Design. The following window will appear:

E.	Field No.	Field Name	Data Type	Length	Description
▶	2	Sell-to Customer No.	Code	20	
▼	3	No.	Code	20	
▼	4	Bill-to Customer No.	Code	20	
▼	5	Bill-to Name	Text	30	
▼	6	Bill-to Name 2	Text	30	
▼	7	Bill-to Address	Text	30	
▼	8	Bill-to Address 2	Text	30	
▼	9	Bill-to City	Text	30	
▼	10	Bill-to Contact	Text	30	
▼	11	Your Reference	Text	30	
▼	12	Ship-to Code	Code	10	
▼	13	Ship-to Name	Text	30	
▼	14	Ship-to Name 2	Text	30	
▼	15	Ship-to Address	Text	30	
▼	16	Ship-to Address 2	Text	30	
▼	17	Ship-to City	Text	30	
▼	18	Ship-to Contact	Text	30	
▼	19	Order Date	Date		
▼	20	Posting Date	Date		
▼	21	Shipment Date	Date		
▼	22	Posting Description	Text	50	
▼	23	Payment Terms Code	Code	10	

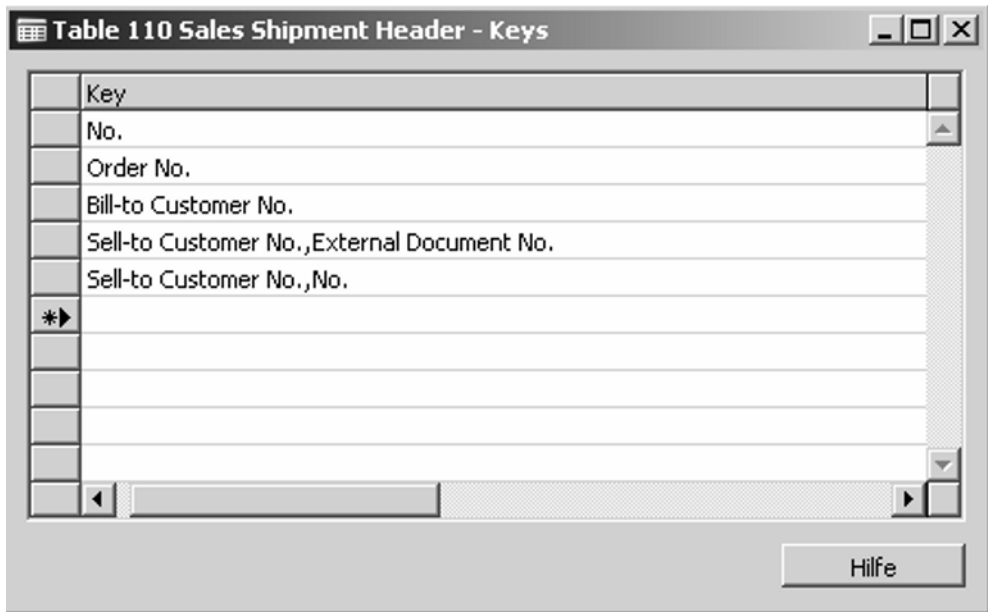
**Figure 8.20** *Table 110 Sales Shipment Header – Table Designer* window

You will find shipping address information in the above table. Look at the keys setup for this table. You will need this information when you connect to this table in the future.

Go to:

**T View > Keys**

The following window will appear:



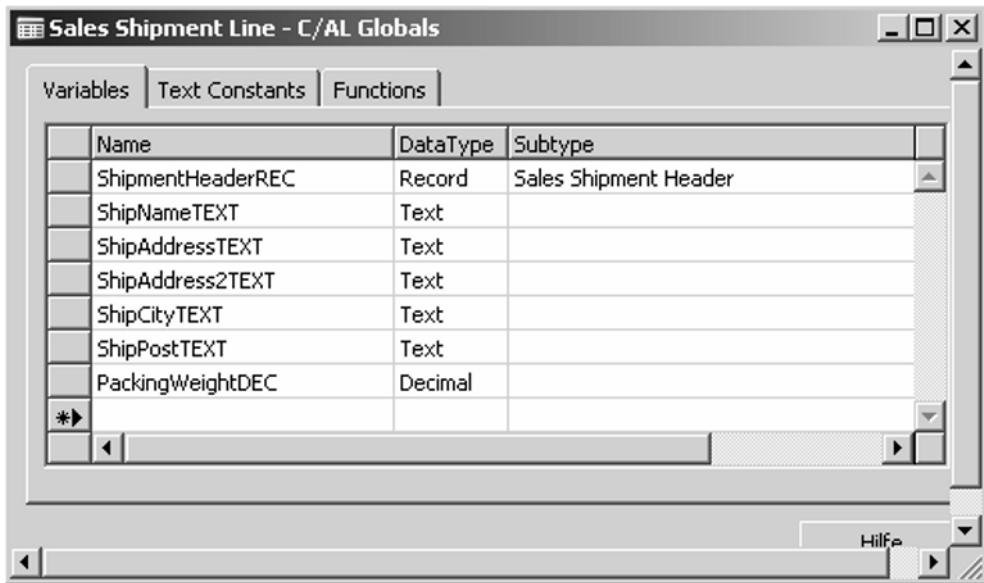
**Figure 8.21** Table 110 Sales Shipment Header – Keys window

Return to the *Dataport* and open the *Global* variable window by going to:

**T View > C/AL Globals**

Type *ShipmentHeaderREC* in the *Name* column, choose *Record* in the *Data Type* column and choose *Sales Shipment Header* in the *Subtype* column. You will refer to the *Sales Shipment Header* with this *Global* variable.

Now you must create some *Global* variables for storing information not directly found in the *DataItem* table. Create the variables shown in the following window:



**Figure 8.22** Sales Shipment Line – C/AL Globals window

Press ESC and then click on the *DataItem* in your *DataPort*. Next, press F9 to enter the *C/AL Editing* environment. First call the *Sales Shipment Header* table via the global variable *ShipmentHeaderREC*.

After initializing the record variable with the RESET function, you must choose a key in the target table. The field *No.* must match the *Document No.* in the *Sales Shipment Line* table. Now set this key in the *ShipmentHeaderREC* variable. Next, set a filter on the *No.* field to the *Document No.* of the *Sales Shipment Line* table. Lastly, find the record in the header associated with the line. The above steps will open, sort and link the *DataItem* to the *Global* variable allowing you to retrieve information out of the *Sales Shipment Header* table concerning a *Sales Shipment Line*.

After you have established a link you need to capture and store this information. Store the information from the *Sales Shipment Header* record in the other *Global* variables.

As a precaution, you should cancel any values in the *Global* variables after each use. This is necessary in the event that Navision does not find a match in the *Global* record. If no match is found, the values from the previous *Sales Shipment Line* are carried into the next record. Such an event is quite unlikely since rarely do lines exist without a headers, however, it is best to program in a defensive manner.

Below is the C/AL Code required to accomplish all these tasks:

```

Sales Shipment Line - C/AL Editor
SETFILTER("Sales Shipment Line".Type,'Item');

Sales Shipment Line - OnBeforeExportRecord()
ShipmentHeaderREC.RESET;
ShipmentHeaderREC.setcurrentkey(ShipmentHeaderREC."No.");
ShipmentHeaderREC.setfilter(ShipmentHeaderREC."No.", "Sales Shipment Line"."Document No.");
if ShipmentHeaderREC.find('-') then
begin
    ShipNameTEXT := ShipmentHeaderREC."Ship-to Name";
    ShipAddressTEXT := ShipmentHeaderREC."Ship-to Address";
    ShipAddress2TEXT := ShipmentHeaderREC."Ship-to Address 2";
    ShipCityTEXT := ShipmentHeaderREC."Ship-to City";
    ShipPostTEXT := ShipmentHeaderREC."Ship-to Post Code";
end else begin
    ShipNameTEXT := '';
    ShipAddressTEXT := '';
    ShipAddress2TEXT := '';
    ShipCityTEXT := '';
    ShipPostTEXT := '';
end;

Sales Shipment Line - OnAfterExportRecord()

Sales Shipment Line - OnBeforeImportRecord()

Sales Shipment Line - OnAfterImportRecord()

```

**Figure 8.23** *Sales Shipment Line – C/AL Editor window*

As you notice, this code must be placed in the *OnBeforeExportRecord()* trigger. Doing so makes this information accessible to Navision before it writes each line in the export text file.

Next, code the calculation for the weight of the packing material. This is calculated simply by subtracting the *Gross Weight* from the *Net Weight* for each line. You can do this with the following code:

```

Sales Shipment Line - C/AL Editor
ShipmentHeaderREC.setcurrentkey(ShipmentHeaderREC."No.");
ShipmentHeaderREC.setfilter(ShipmentHeaderREC."No.", "Sales Shipment Line"."Document No.");
if ShipmentHeaderREC.Find('-') then
begin
    ShipNameTEXT := ShipmentHeaderREC."Ship-to Name";
    ShipAddressTEXT := ShipmentHeaderREC."Ship-to Address";
    ShipAddress2TEXT := ShipmentHeaderREC."Ship-to Address 2";
    ShipCityTEXT := ShipmentHeaderREC."Ship-to City";
    ShipPostTEXT := ShipmentHeaderREC."Ship-to Post Code";
end else begin
    ShipNameTEXT := '';
    ShipAddressTEXT := '';
    ShipAddress2TEXT := '';
    ShipCityTEXT := '';
    ShipPostTEXT := '';
end;

PackingWeightDEC := "Sales Shipment Line"."Gross Weight" -
    "Sales Shipment Line"."Net Weight";

Sales Shipment Line - OnAfterExportRecord()
Sales Shipment Line - OnBeforeImportRecord()
Sales Shipment Line - OnAfterImportRecord()
Sales Shipment Line - OnPostDataItem()

```

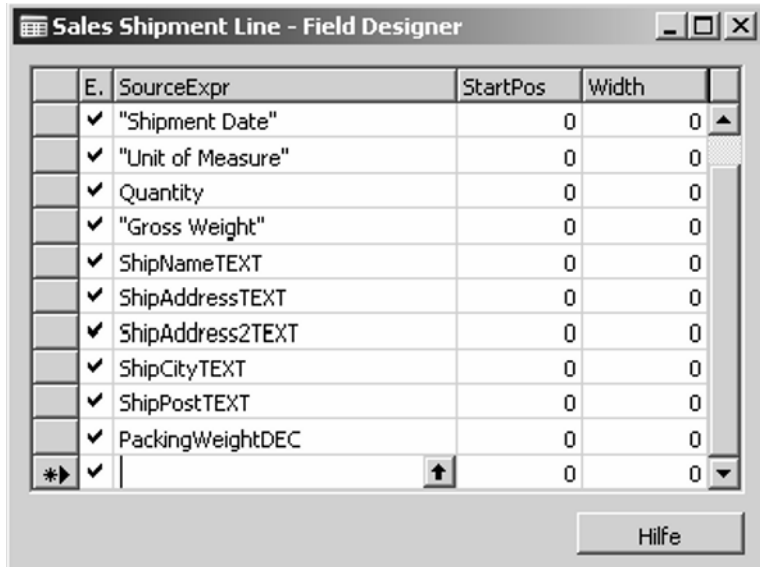
**Figure 8.24** *Sales Shipment Line – C/AL Editor* window

Now all that remains is to enter these variables into the *Dataport Fields*. Press ESC to return to the *DataItem*. Now go to:

**T View > Dataport Fields**

Enter the new variables into the window as seen below:





**Figure 8.25** Sales Shipment Line – Field Designer window

Now close, save and run the *Dataport*. Open the text file in Microsoft Excel according to the method described above. You should see something like the following in Excel (content will differ in your local database from the content in this example):

	A	B	C	D	E	F	G	H	I
1	VL00001	AK2905	12.12.2006	STK	3	0	ANDR. AASLAND AS	Drangedalsveien 2	
2	VL00001	AK2904	12.12.2006	STK	1	0	ANDR. AASLAND AS	Drangedalsveien 2	
3	VL00001	AK2738	12.12.2006	STK	3	0	ANDR. AASLAND AS	Drangedalsveien 2	
4	VL00001	AK2895	12.12.2006	STK	1	0	ANDR. AASLAND AS	Drangedalsveien 2	
5	VL00001	AK2782	14.12.2006	Stück	3	0	ANDR. AASLAND AS	Drangedalsveien 2	
6	VL00001	AK2739	14.12.2006	Stück	3	0	ANDR. AASLAND AS	Drangedalsveien 2	
7	VL00001	AR1053	14.12.2006	Stück	6	0	ANDR. AASLAND AS	Drangedalsveien 2	
8	VL00001	AK2740	14.12.2006	Stück	3	0	ANDR. AASLAND AS	Drangedalsveien 2	
9	VL00001	AK2741	14.12.2006	Stück	2	0	ANDR. AASLAND AS	Drangedalsveien 2	
10	VL00001	AK2783	14.12.2006	Stück	1	0	ANDR. AASLAND AS	Drangedalsveien 2	
11	VL00001	AK2741	14.12.2006	Stück	2	0	ANDR. AASLAND AS	Drangedalsveien 2	
12	VL00001	AK2905	14.12.2006	Stück	6	0	ANDR. AASLAND AS	Drangedalsveien 2	
13	VL00002	AR0736	19.12.2006	Stück	1	0	Heinz Schmitt und Wilh. Kasseler Straße 4		
14	VL00002	AR0752	19.12.2006	Stück	1	0	Heinz Schmitt und Wilh. Kasseler Straße 4		
15	VL00002	AK2672	19.12.2006	Stück	1	0	Heinz Schmitt und Wilh. Kasseler Straße 4		
16	VL00002	AR0568	19.12.2006	Stück	1	0	Heinz Schmitt und Wilh. Kasseler Straße 4		
17	VL00002	AL0018	19.12.2006	Stück	4	0	Heinz Schmitt und Wilh. Kasseler Straße 4		
18	VL00002	AL0019	19.12.2006	Stück	1	0	Heinz Schmitt und Wilh. Kasseler Straße 4		
19	VL00003	AK0720	19.12.2006	Stück	1	0	Hildebrandt GmbH	Ludwig-Erhardt-Str. 15	
20	VL00003	AR0752	19.12.2006	Stück	1	0	Hildebrandt GmbH	Ludwig-Erhardt-Str. 15	
21	VL00003	AK3003	19.12.2006	Stück	1	0	Hildebrandt GmbH	Ludwig-Erhardt-Str. 15	
22	VL00003	AR1010	19.12.2006	Stück	1	0	Hildebrandt GmbH	Ludwig-Erhardt-Str. 15	

**Figure 8.26** Text file in Microsoft Excel

## 8.3 Creating an Import Dataport

Importing data into Microsoft Navision tends to be more difficult than exporting data. One reason for this is that information in Navision is typically interconnected; that is, the content in some fields may be dependent on information in other fields or even in other tables. When bringing information into such interdependent fields, you must *trigger* or recreate these interconnections during the import process. For example, if you import a new *Item* defined in liters, it does not automatically follow that *liter* exists in the *Item Unit of Measure* table. As soon as a user tries to process this new *Item* a failure will occur because Navision does not allow you to use an *Item* unless its basis unit of measure is defined in the *Item Unit of Measure* table.

To avoid this problem, you must validate the basis unit of measure during import.

There is another problem encountered when trying to import: Tables must be searched with each import line to avoid conflicts between import information and information in the destination table.

### 8.3.1 Importing new records into the Item table

In the following example we show you how to solve all these problems by creating an import tool that inserts new records into the *Item* table.

Go to the *Object Designer Dataport* section and click on New. Here you are not particularly reading from the *Item* table, but rather searching and inserting into it after you have read information from an external file. For this reason, we must not use the *Item* table as our *DataItem*. Instead, you must use a special *DataItem*, *Integer*.

*Integer* is a temporary table with a primary key that is a whole number. When using *Integer*, Navision can loop a set of commands as long as you need it to. This loop can be terminated

when a specified number of iterations has been completed or as soon as some condition—defined in the development environment—is met.

Enter *Integer* as the *DataItem* of the import *Dataport*. Next, press SHIFT+F4. Go to menu and then go to:

### T View > Properties

In the *DataItemTableView* press F6. In the *Key* field press F6. The only key available is *Number*. Select *Number* and return to the *DataItem* window. Select the blank line directly below *Integer* and then press SHIFT+F4 to open the general *Dataport Properties*. Assign the *Values* shown in the window below:

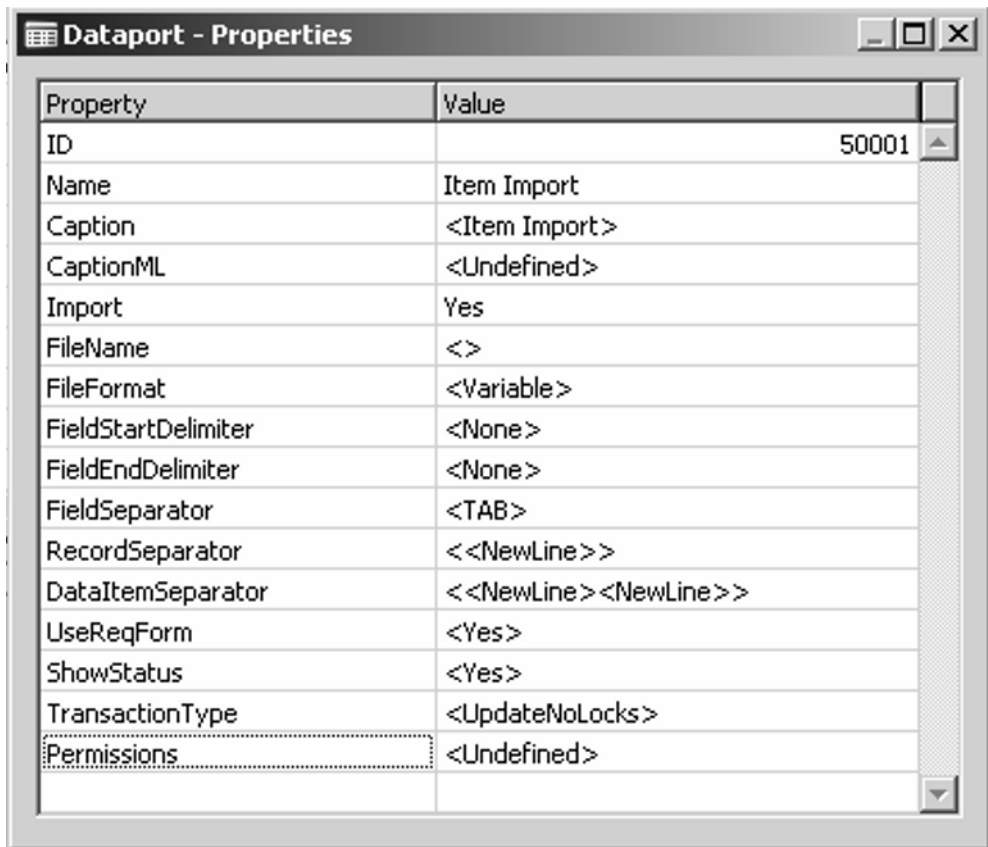
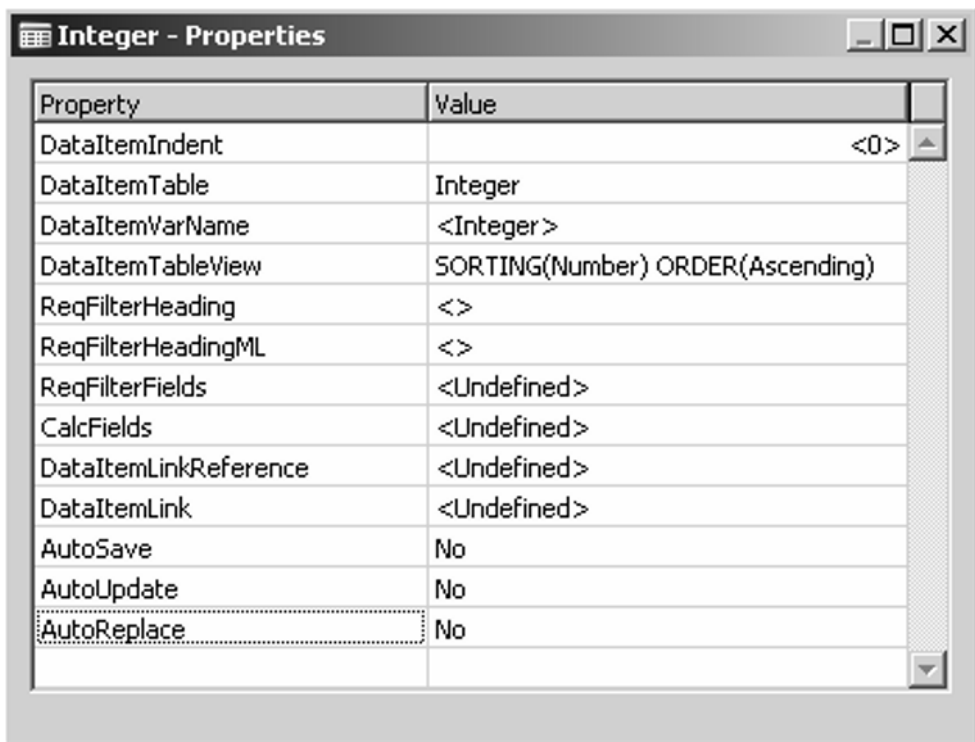


Figure 8.27 *Dataport – Properties* window

Notice that the *Value* after the *Property, Import*, is *Yes* rather than the default *<Yes>*. This difference, although slight, is significant. When *Yes* appears without brackets, the *Dataport* will not allow the user to choose *Export* when running the *Dataport*. When the brackets remain, the import/export option remains available to the end user during run-time.

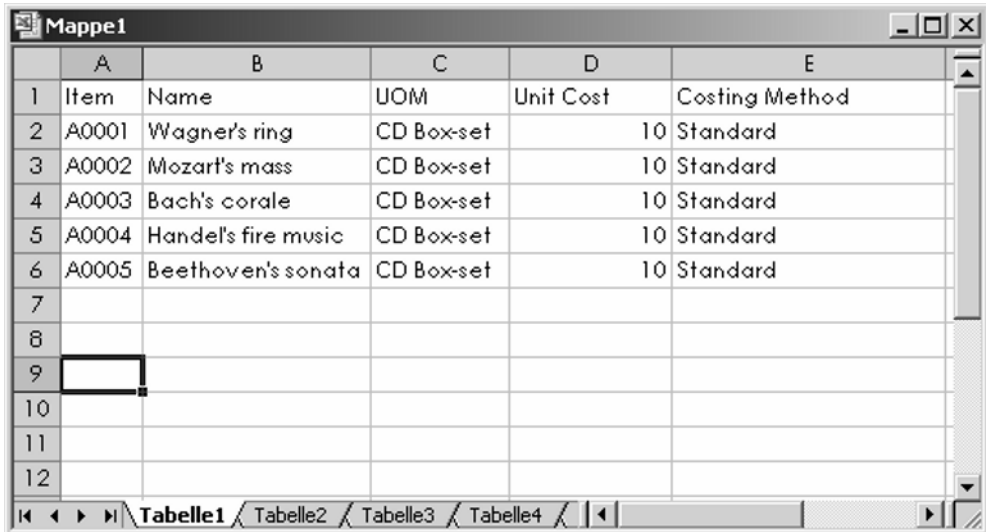
Open the *Properties* for the *Integer DataItem*. Click once more on the *DataItem* and then press *SHIFT+F4*. Set the *Properties* to the following settings:



**Figure 8.28** *Integer – Properties* window

Note that *AutoSave*, *AutoUpdate* and *AutoReplace* have all been set to *No*. These settings are necessary when using *Integer*—a temporary table—as your *DataItem*.

Now take a look at the Import file. For this example, make your preparations from a Microsoft Excel sheet. Create a sheet as shown below:



	A	B	C	D	E
1	Item	Name	UOM	Unit Cost	Costing Method
2	A0001	Wagner's ring	CD Box-set	10	Standard
3	A0002	Mozart's mass	CD Box-set	10	Standard
4	A0003	Bach's corale	CD Box-set	10	Standard
5	A0004	Handel's fire music	CD Box-set	10	Standard
6	A0005	Beethoven's sonata	CD Box-set	10	Standard
7					
8					
9					
10					
11					
12					

**Figure 8.29** Import preparations in Microsoft Excel

Save this Microsoft Excel sheet as text. Go to:

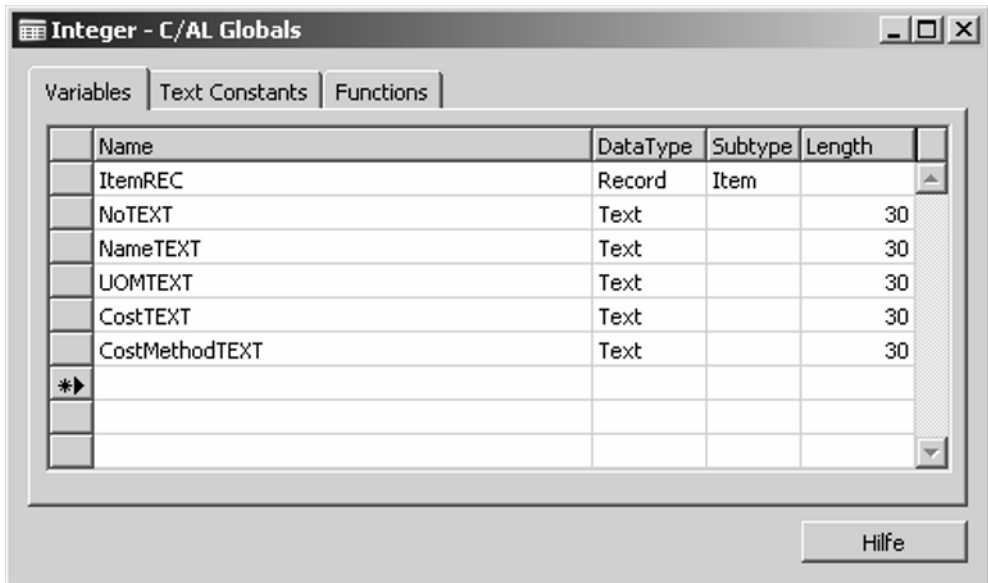
**T File > Save**

Change the *Data Type* to Text (Tab delimited). Save this to your C:/ local drive with the file name, test.txt.

Now return to the *Dataport*. Go to the *C/AL Globals* window. Click on the *Integer DataItem* and then via menu, select:

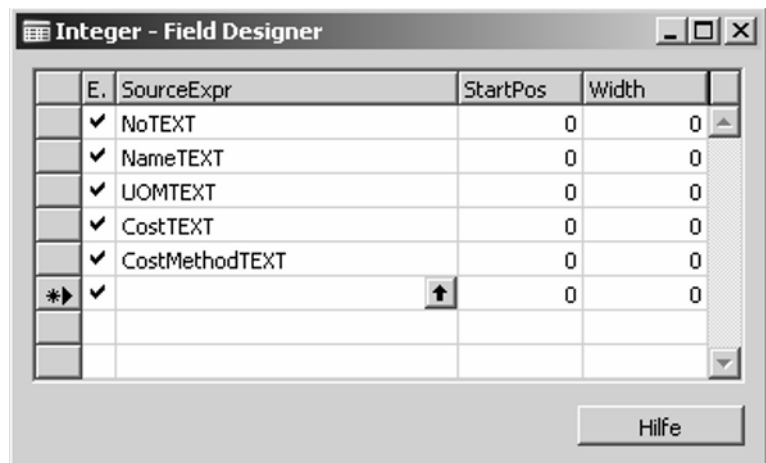
**T View > C/AL Globals**

Create the following variables:



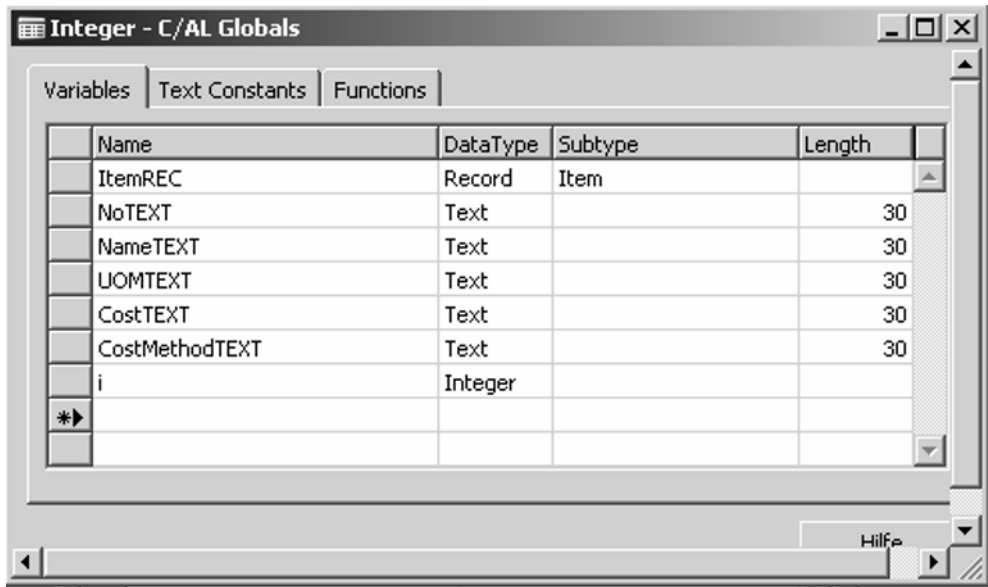
**Figure 8.30** Integer – C/AL Globals window

Note that all variables, with the exclusion of *ItemREC*, are *Data Type*, Text. This is because all imported information is first processed as text. You must, as a second step, re-evaluate the import information and assign the proper Microsoft *Data Type* with C/AL Code. Now enter these text variables into the *Dataport Fields* of the *Integer DataItem*:



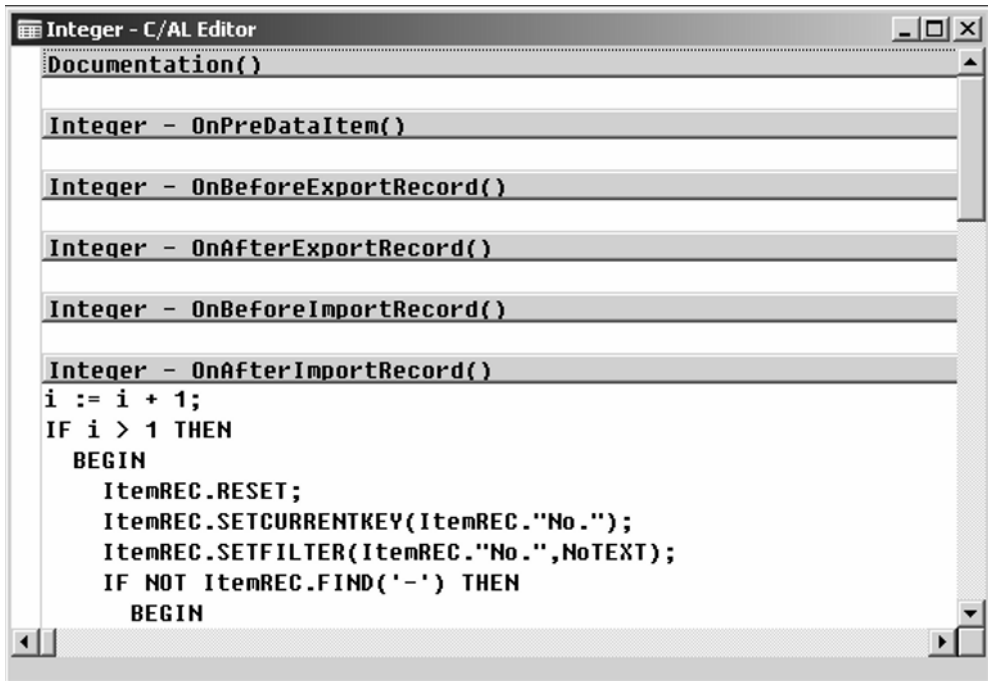
**Figure 8.31** Integer – Field Designer window

Remember there was a header line in the Microsoft Excel text file you created. This line must NOT be imported into the *Item* table; if this happens, the *Dataport* will crash. To avoid this, create an *Integer* variable that counts the number of records the *Dataport* has read. You can call this new *Global* variable, *i*, as shown below:



**Figure 8.32** *Integer – C/AL Globals* window

Now open the *Item* table and make sure there are no *Items* with duplicate *Item Nos*. If none exist, you can initiate the creation of a new *Item*. Now study the following code:



**Figure 8.33** Integer – C/AL Editor window

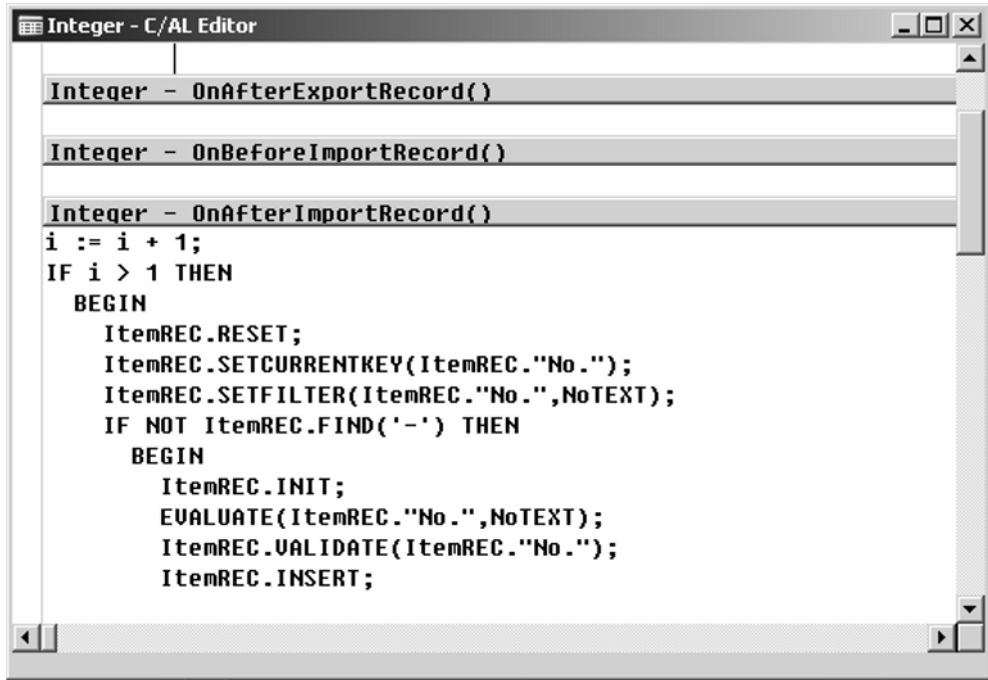
Here you see that the value of *i* increases by 1 each time this code is executed. You can also see that the *ItemREC* will NOT be called if *i* is equal to 1 or less. The header of the text file is the import and *i* equals 1 during import. Using this method you avoid executing *Item* table interpolation with the import of the text header.

To convert the import text into the appropriate Navision *DataType*, you will need a new C/AL Code function, EVALUATE.

This function transfers the value and converts the *DataType* of a text variable into a second variable. This is sometimes impossible. For example, if you “evaluate” a character like the letter ‘A’ into a decimal field, the evaluation will fail.



Now initiate and insert a new *Item* in the *Item* table. You should also “evaluate” the *NoTEXT* import into the *No.* field of the *Item* table. See the following code:



```

Integer - OnAfterExportRecord()

Integer - OnBeforeImportRecord()

Integer - OnAfterImportRecord()
i := i + 1;
IF i > 1 THEN
  BEGIN
    ItemREC.RESET;
    ItemREC.SETCURRENTKEY(ItemREC."No.");
    ItemREC.SETFILTER(ItemREC."No.",NoTEXT);
    IF NOT ItemREC.FIND('-') THEN
      BEGIN
        ItemREC.INIT;
        EVALUATE(ItemREC."No.",NoTEXT);
        ItemREC.VALIDATE(ItemREC."No.");
        ItemREC.INSERT;
      
```

**Figure 8.34** *Integer – C/AL Editor* window

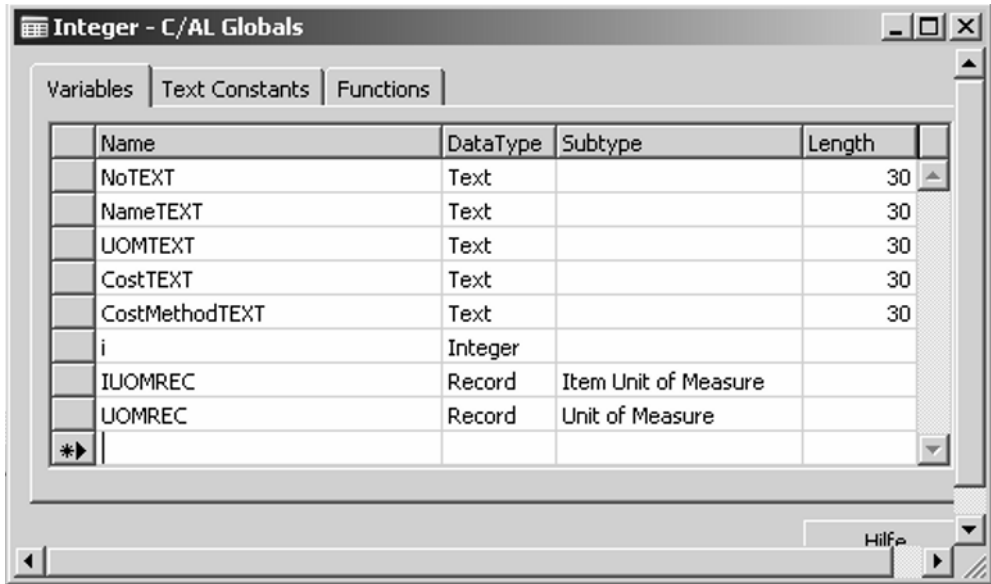
Here you see evaluated the *NoTEXT* into *ItemREC."No."*. Review the previous chapter concerning other C/AL Code functions such as *INSERT* and *VALIDATE*. Although it is not always necessary to validate new *Values* it is a good habit to do so. The only exception to this rule is when you do not want Navision to execute any C/AL Code in the *OnValidate* trigger of the field.

### 8.3.2 Preparing neighboring tables for Item import

You could run into a fatal problem if you try to validate the unit of measure. This is due to the fact that the unit of measure, *CD Box-set*, may not exist in the unit of measure tables. By setting keys and filters as well as through finding and insertion, you can

make the *Dataport* intelligent enough to handle such eventualities.

You must add new *Global* variables that call the *Item Unit of Measure* table and the *Unit of Measure* table. See table below:



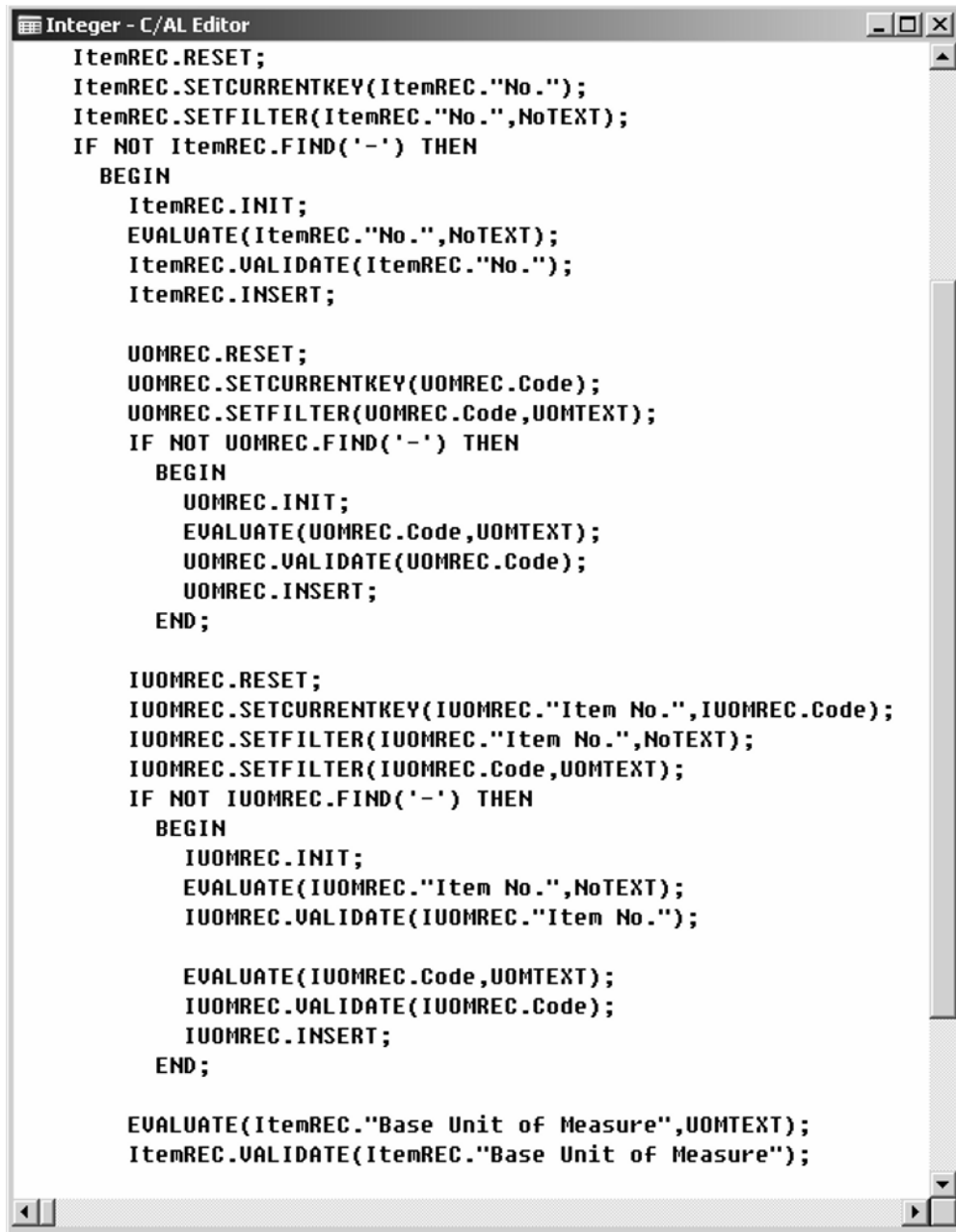
The screenshot shows a window titled "Integer - C/AL Globals" with three tabs: "Variables", "Text Constants", and "Functions". The "Variables" tab is active, displaying a table with the following data:

Name	DataType	Subtype	Length
NoTEXT	Text		30
NameTEXT	Text		30
UOMTEXT	Text		30
CostTEXT	Text		30
CostMethodTEXT	Text		30
i	Integer		
IUOMREC	Record	Item Unit of Measure	
UOMREC	Record	Unit of Measure	
*▶			

The window also features a "Hilfe" button in the bottom right corner.

**Figure 8.35** *Integer – C/AL Globals* window

Now view the following code, beginning with the *UOMREC*, to see an example of how to handle these problems:



```

Integer - C/AL Editor
ItemREC.RESET;
ItemREC.SETCURRENTKEY(ItemREC."No.");
ItemREC.SETFILTER(ItemREC."No.",NoTEXT);
IF NOT ItemREC.FIND('-') THEN
    BEGIN
        ItemREC.INIT;
        EVALUATE(ItemREC."No.",NoTEXT);
        ItemREC.VALIDATE(ItemREC."No.");
        ItemREC.INSERT;

        UOMREC.RESET;
        UOMREC.SETCURRENTKEY(UOMREC.Code);
        UOMREC.SETFILTER(UOMREC.Code,UOMTEXT);
        IF NOT UOMREC.FIND('-') THEN
            BEGIN
                UOMREC.INIT;
                EVALUATE(UOMREC.Code,UOMTEXT);
                UOMREC.VALIDATE(UOMREC.Code);
                UOMREC.INSERT;
            END;

        IUOMREC.RESET;
        IUOMREC.SETCURRENTKEY(IUOMREC."Item No.",IUOMREC.Code);
        IUOMREC.SETFILTER(IUOMREC."Item No.",NoTEXT);
        IUOMREC.SETFILTER(IUOMREC.Code,UOMTEXT);
        IF NOT IUOMREC.FIND('-') THEN
            BEGIN
                IUOMREC.INIT;
                EVALUATE(IUOMREC."Item No.",NoTEXT);
                IUOMREC.VALIDATE(IUOMREC."Item No.");

                EVALUATE(IUOMREC.Code,UOMTEXT);
                IUOMREC.VALIDATE(IUOMREC.Code);
                IUOMREC.INSERT;
            END;

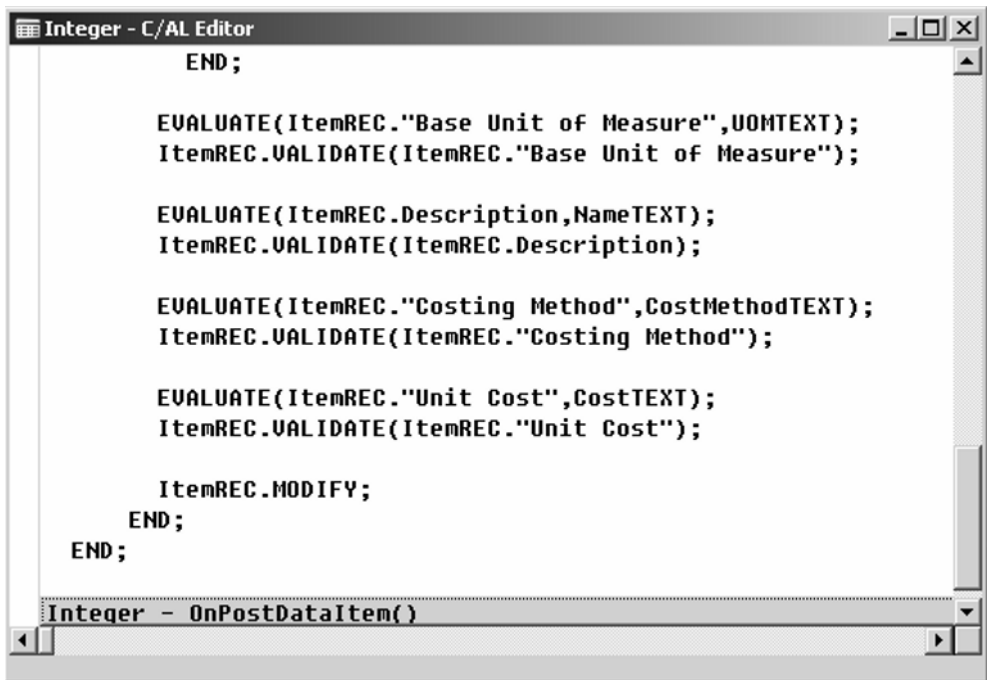
        EVALUATE(ItemREC."Base Unit of Measure",UOMTEXT);
        ItemREC.VALIDATE(ItemREC."Base Unit of Measure");
    END;

```

Figure 8.36 Integer – C/AL Editor window

By repeating the same techniques for the *Unit of Measure* table and the *Item Unit of Measure* table, you have prevented the importation a unit of measure into the *Item* table that is not defined in the unit of measure tables. More importantly, you will have avoided potential crashes.

End your coding by adding the *Name*, *Cost*, *Costing Method* and the final MODIFY statement that effectively writes all changes made to the database. See the following window:



```

Integer - C/AL Editor
END;

EVALUATE(ItemREC."Base Unit of Measure",UOMTEXT);
ItemREC.VALIDATE(ItemREC."Base Unit of Measure");

EVALUATE(ItemREC.Description,NameTEXT);
ItemREC.VALIDATE(ItemREC.Description);

EVALUATE(ItemREC."Costing Method",CostMethodTEXT);
ItemREC.VALIDATE(ItemREC."Costing Method");

EVALUATE(ItemREC."Unit Cost",CostTEXT);
ItemREC.VALIDATE(ItemREC."Unit Cost");

ItemREC.MODIFY;
END;
END;
Integer - OnPostDataItem()

```

**Figure 8.37** Integer – C/AL Editor window

Now save and run the *Dataport*. Target the text file, test.txt, saved earlier on your C:/ drive.

When the *Dataport* has run without any crashes view the *Item Card*. Search for *Wanger's ring*, if you find it then you have discovered the magic of C/AL Code and *Dataports*!

Now we have covered the goals of your ERP system and the concepts upon which it is built. Now you can begin to harness Navision's powerful development tools to optimize your firm.

# index

---

---

## A

ABS 245  
Absolute Numbers 245

---

## B

backup 16, 17, 18, 19, 21, 22, 23  
BEGIN 220, 224, 225, 235, 251  
bitmap 96, 168, 169  
Body 161  
bugs 199

---

## C

C/AL Code 193, 195, 199, 206, 207, 208,  
215, 217, 218, 220, 223, 224, 227, 230,  
231, 232, 233, 234, 242, 243, 244, 247,  
249, 255, 256, 260, 261, 262, 263, 265  
C/AL Editor 199, 200, 201, 206, 207, 208,  
209, 218, 219, 220, 230, 234, 237, 239,  
250, 260, 264  
C/AL Globals 204, 205, 206, 218, 219,  
230, 237, 251, 260  
C/SIDE functions 80  
CALCDATE 225, 246, 247  
CALCFIELDS 247, 251  
CalcFormula 126, 131  
Calculated data 29, 30  
Calculation Formula 126, 130, 131  
Caption 100, 128, 129, 131, 134, 156, 166,  
168, 173, 174, 178, 179, 180, 186, 187,  
189, 191, 192, 202, 203, 210, 241, 253  
Command Box 96  
Command Button 97, 98, 100  
Compiled 151

compiling tool 225  
complexity 7  
CRONUS Database 109  
curiosity 25  
Current Sessions 17  
Cust. Ledger Entry 122, 126, 127, 129, 132  
Customer Ledger Entries 120, 121, 122,  
123, 124, 128, 129, 132, 135

---

## D

data organization 66, 72, 115, 137, 151  
Data Type 90, 91, 103, 106, 123, 126, 194,  
206, 230, 231, 237, 243, 254, 262  
Database 11, 12, 16, 19, 64  
Database Used (KB) 16, 17, 20  
DataCaptionFields 94  
DataItem 111, 149, 150, 151, 152, 153,  
154, 155, 156, 157, 160, 161, 165, 172,  
173, 177, 181, 183, 184, 186, 190, 194,  
195, 196, 198, 199, 200, 201, 204, 205,  
206, 207, 208, 209, 218, 221, 226, 228,  
229, 230, 231, 232, 234, 236, 240, 247,  
249, 250, 251, 259, 260, 261, 262, 263,  
264, 265, 266  
DataItemLink 153, 156  
DataItemTableView 153, 161, 183, 259  
DataPerCompany 93  
Dataport 88, 205, 267, 268, 269, 270, 271,  
272, 273, 274, 275, 276, 278, 281, 282,  
283, 286, 287, 288, 289, 290, 291, 292,  
293, 296, 298  
Date Filter 37, 129, 248  
date formula 247  
DecimalPlaces 174, 175, 179, 180, 186,  
189, 253  
DELETE 266  
Development Concepts 61  
division by zero 185, 194

Document Type 73, 74, 122, 123, 128,  
133, 134, 254, 257  
DocumentationO 207  
DrillDownFormID 93

---

## **E**

Editable 81, 83, 86, 131  
END 220, 224, 225, 235, 251  
ERP 1  
ERROR 255  
Error Message 235, 243  
Export 267, 268, 270, 271, 281, 290

---

## **F**

Field Filter 28  
Field Menu 83, 84, 85, 97, 107, 118, 119,  
122, 123, 126, 129, 134, 143, 144, 173,  
178, 190, 228, 257, 258  
Filtering 35  
Filtering techniques 40, 41  
FIND 216, 220, 221, 222, 223, 224, 225,  
226, 233, 234, 235, 236, 237, 251, 255  
Flow Filter 28, 31, 36, 37, 55, 56, 58, 128,  
129  
FlowFields 115, 247  
Footer 161  
Form Designer 79, 80, 82, 83, 95, 100,  
107  
FORMAT 194, 195, 262

---

## **G**

G/L Account 33, 34, 76, 77, 78, 81, 82, 83,  
86  
GET 238  
Global variable 204, 205, 229, 230, 231,  
232, 237, 243, 260  
GroupFooter 153, 161  
GroupHeader 153, 160, 161  
GroupTotalFields 153, 160, 161

---

## **H**

Header 160

---

## **I**

ID 93  
IF, THEN 222, 223, 224, 225, 236  
Image 96, 168, 169  
import 17, 20, 21, 22, 25, 45, 93, 96, 205,  
267, 288, 289, 290, 292, 294, 295  
IncludeDataInDesc 93  
INIT 261  
INSERT 263  
installation 4, 9, 13  
interconnectedness 3, 8  
Inventory 30, 32, 53, 54, 55, 56, 89, 110,  
114, 115, 128, 247, 249, 250, 251, 253,  
260, 262, 263  
Item Card 56, 57, 58, 59, 89, 94, 102, 103,  
107, 108, 109, 113, 229, 233, 258  
Item Ledger Entry 57, 139, 140

---

## **L**

Label 96, 165, 173, 175, 178, 179, 187,  
188, 189, 190, 191, 192, 193, 253  
license data 20, 21, 90  
License information 21  
Licensed Size (KB) 19, 20, 21  
list view 84  
LOCKTABLE 260  
Login 11, 13, 19  
LookupFormID 93  
LookupOK 98

---

## **M**

MESSAGE 235, 236, 237, 251, 255  
Messages 6, 19, 23, 41, 53, 54, 58, 158,  
177, 234, 235, 243, 255

Microsoft Business Solutions Center VI,  
20, 21, 90  
Microsoft Excel 26, 27, 205  
MODIFY 266

---

## N

Navigation Pane 213  
Net Change 32, 33, 34, 35, 36, 38, 39, 83,  
86, 115

---

## O

Object Designer 87, 88, 90, 93, 94, 100,  
101, 103, 125, 132, 148, 171, 181, 184,  
218, 227, 242, 253, 259  
object ID 88  
OnAfterGetRecord() 207, 208, 231, 255,  
260  
one-to-many relationship 71, 72  
OnPostDataItem() 208  
OnPreDataItem() 200, 207  
Option 91, 97, 123, 124, 128, 254, 262  
OptionCaptionML 123  
OptionString 91, 123

---

## P

PasteIsValid 94  
Permissions 93  
philosophy of continual improvement 7,  
8, 9, 33, 63  
POWER 244  
primary key 71, 72, 73, 74, 75, 89, 90, 92,  
101, 103, 104, 115, 117, 118, 119, 120,  
128, 132, 141, 142, 145, 149, 150, 229,  
230, 232, 234, 237, 238, 257, 258, 260,  
261  
ProcessingOnly 157, 264  
Programming Language 217  
PushAction 98, 100

---

## R

Record Global 230, 231, 232, 233, 234,  
236, 238, 247, 251, 260, 261, 262, 263,  
266  
RemoteServerNo 93  
REPEAT 225, 226, 246  
REPEAT, UNTIL 225, 226  
Report Designer 110, 148, 149  
ReqFilterFields 111  
Request Form 156  
RESET 231, 234, 237  
ROUND 245  
Rounding 245

---

## S

Sales (LCY) 121, 122, 126, 127, 134  
Sales Header 73, 74, 97  
Sales Line 64, 69, 75, 76, 77, 78, 82, 97  
Sales Order 51, 56, 58, 59, 64, 65, 67, 68,  
69, 70, 71, 73, 74, 77, 82, 83, 97  
secondary key 71, 75, 89  
Section Designer 159, 162, 163, 165, 166,  
170, 171, 172, 176, 178, 180, 181, 188,  
193, 195, 197, 201, 203, 207, 208, 209,  
210, 211, 227, 240, 241, 254  
Server 12  
SETCURRENTKEY 216, 231, 232, 234, 237  
SETFILTER 216, 221, 222, 232, 233, 234,  
237, 251  
Sorting 28, 41, 72, 73, 74, 117, 118, 216,  
257  
SourceExpr 166, 167, 186, 188, 189, 191,  
192, 193, 194, 201, 202, 203, 209, 210,  
211, 218, 221, 230, 241, 253  
SourceExpression 86, 97  
SourceTable 80, 81, 83, 86, 95, 103, 118,  
122, 143, 229, 230, 257, 259  
SQL 12, 13, 41, 42, 44, 131, 156  
standardization 2  
stock data 4, 29  
stock information 117, 119, 120, 132  
Subform 97

syntax 131, 156, 158, 177, 193, 215, 217,  
218, 220, 239, 243, 249, 254, 255, 262

---

## **T**

Table 90  
Table Designer 104  
Table Filter 28, 31, 34, 43, 47, 54, 55, 127,  
129, 130, 141, 183, 232, 233, 259  
Table relations 61, 104, 106  
Test Database 14  
Text Box 97, 166, 167, 173, 174, 175, 178,  
179, 180, 186, 188, 190, 191, 192, 193,  
197, 201, 202, 206, 209, 210, 218, 221,  
230, 240, 253  
Toolbox 95, 96, 97, 165, 166, 168, 169,  
188, 190, 192, 197, 210, 240  
TotalFields 153, 196, 199  
training 2, 8  
Transaction data 29  
TransFooter 161  
TransHeader 160

---

## **U**

uncertainty 66  
UNTIL 226, 246

---

## **V**

VALIDATE 261, 262, 263

---

## **W**

workflow 1, 2, 7, 8

---

## **Z**

Zoom 49, 50, 51, 60, 73, 74, 78, 120, 121,  
139, 141, 142, 143