

Microsoft Dynamics™ NAV 5.00

# Application Designer's Guide



# APPLICATION DESIGNER'S GUIDE



Information in this document, including URL and other Internet Web site references, is subject to change without notice. Unless otherwise noted, the companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted in examples herein are fictitious. No association with any real company, organization, product, domain name, e-mail address, logo, person, place, or event is intended or should be inferred. Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

© 2007 Microsoft Corporation. All rights reserved.

Microsoft, MS-DOS, Windows, Windows Server, Windows Vista, Application Server for Microsoft Dynamics NAV, AssistButton, C/AL, C/Front, C/Side, FlowField, FlowFilter, C/Side Database Server for Microsoft Dynamics NAV, Microsoft Business Solutions–Navision, Microsoft Dynamics NAV, Microsoft Dynamics NAV Debugger, Navision, NAV ODBC, SIFT, SIFTWARE, SQL Server, SumIndex, SumIndexField are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

All other trademarks are property of their respective owners.

This manual provides information about the C/SIDE<sup>®</sup> development system. It is part of the documentation and Help materials for Microsoft Dynamics<sup>™</sup> NAV.

When you create a C/SIDE application, you combine different types of application objects into a whole that solves a business problem. Each of the seven types of application objects available has its own part in this manual.

The manual is divided into 12 parts. Each part contains one or more chapters. The first chapter in a part always deals with the fundamentals, for example, "Form Fundamentals," and the succeeding chapters present more advanced information.

In addition to this manual, C/SIDE has an online *Reference Guide*. Here you can find reference information about programming issues: functions, triggers, properties, and so on.

You may also find it useful to refer to the following manuals and online Help:

***Installation & System Management: C/SIDE Database Server for Microsoft Dynamics NAV***

This manual explains the more technical aspects of Dynamics NAV. You will find information about user administration, backup procedures and other items that are also relevant for application developers.

***Installation & System Management: SQL Server Option for Microsoft Dynamics NAV***

This manual explains how to install and maintain the SQL Server Option for Dynamics NAV. This program is designed to run on SQL Server 2000.

***Installation & System Management: Application Server for Microsoft Dynamics NAV***

This manual explains how to install and maintain NAV Application Server.

***Development Guide for Communication Components***

This online Help describes the Dynamics NAV Communication Component, Dynamics NAV Named Pipe Bus Adapter and Dynamics NAV MS-Message Queue Bus Adapter. These components allow applications to communicate easily with each other.

## TABLE OF CONTENTS

<b>PART 1</b>	<b>ARCHITECTURE</b>	<b>1</b>
	<b>Chapter 1 Architecture and Installation</b>	<b>3</b>
	The Microsoft Dynamics NAV Client/Server Environment	4
	Installation	6
	<b>Chapter 2 The Server Options</b>	<b>13</b>
	The Different Server Options	14
	<b>Chapter 3 The Dynamics NAV Security Model</b>	<b>29</b>
	Security	30
	Business Areas and Granules	36
<b>PART 2</b>	<b>FUNDAMENTALS</b>	<b>41</b>
	<b>Chapter 4 C/SIDE Fundamentals</b>	<b>43</b>
	The C/SIDE User Interface	44
	What Is a C/SIDE Application?	47
	The Physical and the Logical Database	49
	<b>Chapter 5 Designing a C/SIDE Application</b>	<b>51</b>
	Introduction to C/SIDE Application Design	52
<b>PART 3</b>	<b>TABLES</b>	<b>57</b>
	<b>Chapter 6 Table Fundamentals</b>	<b>59</b>
	What Is a Table?	60
	Viewing and Modifying Properties	66
	Defining Keys	71
	Identifiers, Data Types and Data Formats in the SQL Server Option for Dynamics NAV	78
	Saving tables and Viewing Sorted Data	84
	Special Table Fields	87
	Dividing the Database into Companies	94
	<b>Chapter 7 Customizing and Maintaining Tables</b>	<b>95</b>
	Using Table and Field Triggers	96
	Setting Relationships Between Tables	98
	Changing Tables That Contain Data	104
	Linked Objects	105
	<b>Chapter 8 Special C/SIDE Tables</b>	<b>109</b>
	What Is a Temporary Table?	110
	What Is a System Table?	112
	What Is a Virtual Table?	117

<b>PART 4</b>	<b>FORMS</b>	<b>137</b>
	<b>Chapter 9 Form Fundamentals</b>	<b>139</b>
	What Are Forms?	140
	Creating Forms	142
	Saving, Compiling and Running Forms	150
	<b>Chapter 10 Designing Forms</b>	<b>151</b>
	Form and Control Properties	152
	Types of Controls	155
	Adding Controls	157
	Selecting, Moving and Adjusting Controls	160
	Tools for Customizing Controls	164
	Setting Control Properties	165
	Container Controls	169
	How to Use Controls in Applications	171
	<b>Chapter 11 Extending the Functionality of Your Forms</b>	<b>185</b>
	Main Forms and Subforms	186
	Looking Up Values and Validating Entries	189
	Drilling Down to the Underlying Transactions	193
	Launching Another Form	195
	Designing Menu Buttons	196
	Form and Control Triggers	200
	Form Types and Characteristics	203
<b>PART 5</b>	<b>REPORTS</b>	<b>209</b>
	<b>Chapter 12 Report Fundamentals</b>	<b>211</b>
	What Are Reports?	212
	What Happens When a Report Runs?	215
	The Report Designer	218
	Saving, Compiling and Running Reports	221
	<b>Chapter 13 Designing Reports</b>	<b>223</b>
	Report Properties	224
	Designing a Simple Report	228
	Designing a More Advanced Report	236
	<b>Chapter 14 Extending Report Functionality</b>	<b>241</b>
	Grouping and Totaling	242
	Triggers in Reports	249
	Advanced Sample Reports	250
	Creating a Simple Document	264
	Creating a Nonprinting Report	270
	Types of Report	274



<b>PART 6</b>	<b>CODEUNITS</b> .....	<b>277</b>
	<b>Chapter 15 Codeunit Fundamentals</b> .....	<b>279</b>
	What Is a C/SIDE Codeunit? .....	280
	Creating Codeunits .....	282
	Using Codeunits .....	289
	<b>Chapter 16 Introducing the C/AL Language</b> .....	<b>295</b>
	What Can You Do with C/AL? .....	296
	What Are Statements, Expressions, and Operators? .....	297
	Introducing the Elements of C/AL Expressions .....	305
	The C/AL Control Language .....	314
	<b>Chapter 17 Using C/AL</b> .....	<b>323</b>
	Overview .....	324
	System-Defined Variables .....	326
	Handling Runtime Errors .....	327
	The Essential C/AL Functions .....	328
	<b>Chapter 18 Debugging C/AL Code</b> .....	<b>345</b>
	What Are Bugs? .....	346
	The Microsoft Dynamics NAV Debugger .....	352
	The Code Coverage Tool .....	360
	<b>Chapter 19 Extending C/AL</b> .....	<b>363</b>
	What Is COM? .....	364
	Using COM Technologies in C/SIDE .....	366
	Using C/SIDE as an Automation Controller .....	370
	Receiving Events in C/SIDE .....	386
	Using Custom Controls from C/SIDE .....	394
	Acquiring Controls .....	402
<b>PART 7</b>	<b>DATAPORTS</b> .....	<b>403</b>
	<b>Chapter 20 Dataports</b> .....	<b>405</b>
	What Are Dataports? .....	406
	Designing Dataports .....	411
	Exporting Data .....	417
	Importing Data .....	425
<b>PART 8</b>	<b>XMLPORTS</b> .....	<b>437</b>
	<b>Chapter 21 XMLports</b> .....	<b>439</b>
	XMLport Fundamentals .....	440
	Designing XMLports .....	442
	XMLport Examples .....	445
	Validating Data .....	450
	XMLports and Business Notifications .....	452

<b>PART 9</b>	<b>MENUSUITE OBJECTS</b>	<b>453</b>
	<b>Chapter 22</b>	<b>MenuSuite Objects</b> . . . . . <b>455</b>
		Menu Suite Fundamentals . . . . . 456
		Customizing a Menu Suite . . . . . 457
		Exporting a MenuSuite Object . . . . . 460
		Upgrading Menu Suite Content . . . . . 462
<b>PART 10</b>	<b>MULTILANGUAGE FUNCTIONALITY</b>	<b>463</b>
	<b>Chapter 23</b>	<b>Multilanguage Functionality</b> . . . . . <b>465</b>
		Multilanguage Functionality . . . . . 466
		Developing Multilanguage-Enabled Applications . . . . . 472
		Learning the Code Base Language . . . . . 476
		Number Ranges for Text Constants . . . . . 479
<b>PART 11</b>	<b>BEYOND THE BASICS</b>	<b>481</b>
	<b>Chapter 24</b>	<b>SumIndexFields</b> . . . . . <b>483</b>
		SumIndexFields . . . . . 484
		SIFT and the SQL Server Option for Dynamics NAV . . . . . 486
	<b>Chapter 25</b>	<b>Type Conversion</b> . . . . . <b>505</b>
		Type Conversion in Expressions . . . . . 506
		Type Conversion Mechanisms . . . . . 508
	<b>Chapter 26</b>	<b>Numbering in Dynamics NAV</b> . . . . . <b>515</b>
		How Does Number Sorting Work? . . . . . 516
	<b>Chapter 27</b>	<b>C/SIDE in Multiuser Environments</b> . . . . . <b>519</b>
		Ensuring Data Integrity in a Multiuser Environment . . . . . 520
		Locking in Dynamics NAV – a Comparison of the two Server Options . . . 527
	<b>Chapter 28</b>	<b>Caption Class Functionality</b> . . . . . <b>531</b>
		Syntax . . . . . 532
		Function Code . . . . . 537
	<b>Chapter 29</b>	<b>Supporting Record Level Security</b> . . . . . <b>545</b>
		Record Level Security . . . . . 546
	<b>Chapter 30</b>	<b>Performance</b> . . . . . <b>559</b>
		The DBMS Cache . . . . . 560
		The Commit Cache . . . . . 562
		The Command Buffer . . . . . 563
		Keys, Queries and Performance . . . . . 565
		C/AL Database Functions and Performance on SQL Server . . . . . 567
		Configuration Parameters . . . . . 568
		Login Stored Procedure on the SQL Server Option . . . . . 572

<b>PART 12</b>	<b>APPENDIXES</b>	<b>575</b>
	<b>Appendix A C/SIDE Specifications</b>	<b>577</b>
	Specifications for the DBMS	578
	Specifications for C/SIDE Application Objects	579
	<b>Appendix B Report Flow Charts</b>	<b>581</b>
	Report Flow Charts	582
	Report.Run	583
	Dataltem.Run	584
	Section.Run	585
	Header.Run	586
	Footer.Run	587
	TransHeader.Run	588
	TransFooter.Run	589
	GroupHeader.Run	590
	GroupFooter.Run	591
	Body.Run	592
	NewPage	593
	GetRecord	594
	<b>Appendix C Dataport Flow Charts</b>	<b>595</b>
	Dataport Flow charts	596
	Dataport.Import/Export	597
	Dataltem.Export	598
	VariableRecord.Export	599
	FixedRecord.Export	600
	Dataltem.Import	601
	VariableRecord.Import	602
	FixedRecord.Import	603
	<b>Appendix D NDBCS – The Database Driver</b>	<b>605</b>
	NDBCS – the Database Driver	606
	A Brief History of Performance Improvements	614



**Part 1**  
**Architecture**



## **Chapter 1**

### **Architecture and Installation**

In this chapter, you learn about the architecture of Microsoft Dynamics™ NAV and how to install the Dynamics NAV client, C/SIDE Database Server and SQL Server.

This chapter contains the following sections:

- The Microsoft Dynamics NAV Client/Server Environment
- Installation

## 1.1 The Microsoft Dynamics NAV Client/Server Environment

Microsoft Dynamics NAV is a two-tier application. It consists of a Database Management System (DBMS) that resides on the server, and a Graphical User Interface (GUI) that resides on each client. You can also configure the client as a stand-alone installation, which means that the client functions as a server and a single client in one.

Dynamics NAV has two database options: the standard Dynamics NAV database and SQL Server database. You can choose to use the database engine that is built into each client to run the standard Dynamics NAV database as a stand-alone installation. The database can also be run from a server, which allows many clients to connect simultaneously to the same database. The server runs on a designated computer that all the clients communicate with.

You can also install the SQL Server Option for Dynamics NAV as a stand-alone installation. To do this, you install the Microsoft SQL Server Express with the client. This is a small version of SQL Server. During the installation, the demonstration database is attached to this local instance of SQL Server and is opened the first time you start Dynamics NAV.

If you want to run the SQL Server Option as a client/server installation, you install SQL Server on the server computer. You must then install the Dynamics NAV client on all the client computers.

In the SQL Server Option, the client usually uses the TCP/IP protocol to communicate with SQL Server although it can also use an ODBC connection. In the C/SIDE Database Server option, the client generally uses the TCP/IP protocol to communicate with the server.

Once the server and clients are configured, the user does not need to worry about the way the server and clients work together. It appears seamless to the user.

NAV Application Server is a middle-tier server that supports an n-tier architecture, which executes business logic without user intervention. NAV Application Server allows you to communicate with external services.

NAV Application Server acts as a client towards a database server and can act as a server for other services. When you start NAV Application Server, it opens a predefined database and executes the C/AL code in a predefined codeunit.

NAV Application Server can communicate with both C/SIDE Database Server and the SQL Server Option in the same way as a Dynamics NAV client. NAV Application Server only supports Windows Authentication and automatically reconnects to the database server if there is a problem with the network.

### The Dynamics NAV C/SIDE Client

The Dynamics NAV C/SIDE Client is basically responsible for the user interface, but it actually does much more. The client can connect directly to a standard database file without going through the server. This is the stand-alone setup that was mentioned earlier. The client is also responsible for executing all the business logic. The client reads objects from the database and is also responsible for running the objects and controlling their behavior. Most of the Dynamics NAV application runs on the individual clients.



## The Server

If the clients do most of the work, what is left for the server to do? The standard server component of Dynamics NAV:

- controls the number of users that can connect to the database at one time.
- controls access to the data through locking.
- keeps track of all the read and write transactions performed by each user.
- sends data to each client, as requests are made.
- performs all the key-based filtering and calculates the SumIndexFields.
- caches data that can be requested again.

This is not a complete list and is only designed to give you an idea of what the server does. Microsoft SQL Server also does all of these things. One thing that the standard server does that SQL Server does not, is keep track of the different versions of the same record that different users have accessed. We will discuss this in later chapters.

Together, the client and the database server provide a seamless solution.

## 1.2 Installation

As stated earlier, you can install Dynamics NAV as a client/server installation or as a stand-alone client installation.

There are several ways that you can set up the client/server installation. In this chapter, we will look at the various ways that you can install the client and the server. We will also cover some of the most common mistakes that you must avoid.

To ensure that the customer' installation is installed successfully:

- bring the Dynamics NAV installation DVD.
- bring the customer's database.
- bring the customer's license.
- test the installation, including the server and client connections.

### What to Avoid During Installation

Ensure that you do not do the following on the computer where the database is stored:

- Do not use the disk compression provided by the operating system or any programs such as DoubleSpace, Stacker or DriveSpace.
- Limit the number of services and programs that are run simultaneously because they slow down processing.
- Do not use Lazy Write or any other caching programs.

#### Important

Use Windows Server 2003, Windows XP as the operating system for the server.  
Use Windows Vista, Windows XP or Windows Server 2003 as the operating system for the client.

### The Product DVD

When you insert the DVD, an HTML page automatically opens in your browser. This page contain links to the installation programs as well as the documentation about the Microsoft Dynamics NAV suite of programs.

With the Dynamics NAV product DVD you can install:

- a Dynamics NAV stand-alone client installation.
- a Dynamics NAV stand-alone client installation that runs on the Microsoft SQL Server Express Database Engine.
- C/SIDE Database Server for Microsoft Dynamics™ NAV so that you can set up a Dynamics NAV client/server installation.
- the Application Server for Microsoft Dynamics™ NAV.
- the Commerce Gateway components.
- the Business Notification Server for Microsoft Dynamics™ NAV
- the Automated Data capture System for Microsoft Dynamics™ NAV
- Microsoft Dynamics™ NAV ODBC

You must install Microsoft SQL Server separately if you want to set up a Dynamics NAV client/server installation that runs on SQL Server.

### Installing the Dynamics NAV C/SIDE Client

Click Client, Dynamics NAV Client and a standard Windows installation program opens and guides you through the installation process.

When you are installing the client you can select the features that you want to install. the features that you can select include:

- Help – the online Help for Dynamics NAV.
- Demo Database – a Dynamics NAV database that contains a demonstration company. This database will be opened automatically the first time you start Dynamics NAV.
- Backup of Demo Database – a Dynamics NAV backup of the demonstration database. You can restore this backup into a new database.
- Commerce Integration – the Commerce Gateway and Commerce Portal components. You must install these components if you want to run either Commerce Gateway or Commerce Portal. If you select this feature, the Microsoft .NET Framework is also installed. The .NET Framework is not removed when you uninstall the Dynamics NAV client. It is given an entry of its own in the Add or Remove Programs window and you can uninstall it from there.
- Business Notification Manager – this feature allows you to automatically send e-mails to your employees and business partners informing them of business events.
- Employee Portal – the Employee Portal components. You must install these components if you want to run Employee Portal.
- Outlook Add-in – this feature creates a toolbar in Outlook that allows you to open a Dynamics NAV Contact or a Dynamics NAV To-do from the corresponding Outlook item.
- Dynamics NAV Gantt Server – an ActiveX component that allows production managers to plan shop floor production with the help of Gantt charts and update their schedules in Dynamics NAV.

### Files on the Hard Drive

When the installation program is finished, you can explore the directory where the files were installed. Here is a list of some of the files that are installed:

<code>fin.exe</code>	Executable file for starting the Dynamics NAV C/SIDE client for C/SIDE Database Server or stand-alone.
<code>finsql.exe</code>	Executable file for starting the Dynamics NAV client for the SQL Server Option.
<code>fin.flf</code>	License file that determines the functionality of the installation based on the granules purchased.
<code>cronus.flf</code>	Sample license file renamed to the name of the demonstration company.
<code>database.fdb</code>	Default physical file used for the database, which contains all the objects and data.
<code>master.chm</code>	Main Help file for the application. This file is stored in a sub-directory along with all of the other Help files.

For more information about installing the client, see the manual *Installation & System Management*: for the server option that you want to install. These manuals are available on the product DVD.

### The Stand-Alone Client Installation

The simplest setup that a customer can ask for is a one-user system. We call this a stand-alone client. This type of setup is very straightforward. You install the entire client on the computer that the user will be using. You must then replace the `database.fdb` file with the customized database you created for the customer. Finally, replace the `fin.flf` license file that was installed with the executables with the customer's license file.

To install a stand-alone client for SQL Server, you must install the Microsoft SQL Server Express along with the client. This installs a local instance of SQL Server and attaches a demonstration SQL Server database to the server. When you start the client, it automatically opens the demonstration database.

You can also create your own database on the server, but we'll have more about that later.

### The C/SIDE Database Server Client/Server Installation

This type of setup involves one C/SIDE Database Server and several clients. Here, you must start by installing the stand-alone client as described earlier on the server computer. (This is the first computer system that will run the server program.) By doing this first, you will assure that you have the customer's database and license installed. The only difference here is that you might want to move the database file either to its own directory or to its own drive. In a later chapter, we will discuss the database file and how best to set it up.

Once you have installed the client on the server, you can run the C/SIDE Database Server installation program. This is a completely separate installation program.

C/SIDE Database Server comes with a standard Windows installation program that will guide you through the installation process. If you have difficulty understanding any of the decisions that you must make during the installation, consult the manual *Installation & System Management: Database Server for the Dynamics NAV C/SIDE Client*. This manual contains a detailed step by step description that will guide you through the installation process.

#### Important

.....  
We recommend that you practice installing C/SIDE Database Server before installing it for a customer.  
.....

The installation program will prompt you for the location of the database and the license file. Select the customer's database file and license. The installation program will also create a service and start the service at the end of the installation. It is this service that runs the Dynamics NAV standard server. Once the installation is complete, the server is up and running.

The next step is to install the Dynamics NAV client on all the client computers. This installation is similar to the stand-alone client installation, except for some omissions. You should only install the client. This is the Minimum installation – you do not need a

database file. You do not need the customer's database, as it resides on the server. However, you should still copy the customer's license onto each client.

Once the client is installed, the final step is to set up a shortcut on the desktop for the user and test it. The shortcut should include certain parameters, as in the following example:

```
"C:\Program Files\...\fin.exe"  
servername=MyServer , nettype=TCPS
```

In this example, `servername` specifies the name of your C/SIDE Database Server, and `nettype` specifies the protocol being used. By putting these parameters directly into the shortcut, the settings will not be lost and the user only needs to remember their password.

The final step is to double-click the shortcut and test the connection. If an error occurs, it is usually because the client cannot connect to the server. Depending on the protocol being used, there are certain steps you can take to identify the problem. For more information about working with these protocols, see the section Troubleshooting the Database Connection.

If the connection worked, just repeat the client installation on each client computer and you are finished.

**Running as a service** If you want to run Dynamics NAV Database Server as a service or use the `installasservice` command line parameter when you start the server and use TCPS as the network type, you **must** ensure that the service is running as the NT Authority\Network Service account or the Local System account. The NT Authority\Network Service account only exists on Windows XP and Windows Server 2003. If you are running Windows 2000 Server, you should create an account with least privileges for the service or else the service will be assigned a Local System account. This account should at the most have the same privileges as the normal Users account or be a domain account that is not an administrator either in the domain or on any local computer.

To have the highest level of security, we recommend that you run your Dynamics NAV Database Server on Windows XP or on Windows Server 2003, use TCPS as the network protocol and use the NT Authority\Network Service account as the service account.

You must also remember to give the NT Authority\Network Service account or the user account that the server is running under access to the database file(s) to ensure that the users can connect to the database.

To give the NT Authority\Network Service account read and write access to a database file on Windows XP:

- 1 In Windows Explorer, navigate to the folder that contains the database file.
- 2 Select the database file, right-click it and click Properties.
- 3 In the Properties window, click the Security tab and under the Group and user names field, click Add.
- 4 In the Select Users, Computers, or Groups window, enter Network Service and click OK.

- 5 NETWORK SERVICE has been added to the Group and user names field in the Properties window.
- 6 Select NETWORK SERVICE and in the Permissions field give it Read and Write permission.

### Installing the Microsoft SQL Server Option for the Dynamics NAV C/SIDE Client

The setup changes quite drastically when you are using the SQL Server Option. Follow the Microsoft SQL Server guidelines when installing SQL Server on the server computer.

In this environment, there is only one thing from the Dynamics NAV product DVD that needs to be installed on the server machine – two SQL Server extended stored procedures. From the server computer, access the Dynamics NAV product DVD. In the folder \$:\SQL\_ESP, where the \$ is the DVD drive, click the file `xp_ndo.exe`. This file contains the extended stored procedures. When prompted, enter the path to the BINN subfolder of the SQL Server installation folder. The unzipped `xp_ndo.dll` file must be stored in this folder on SQL Server.

Once SQL Server is installed and working, you can begin installing Dynamics NAV on the client computers.

Once the clients are installed, the final step is to set up a shortcut on the desktop for the user and test it. The shortcut should include certain parameters, as in the following example:

```
"C:\Program Files\...\finsql.exe"  
servername=MyServer,nettype=Named Pipes,ntauthentication=yes
```

Note that the name of the executable file (when creating the shortcut) is `finsql.exe`, and not `fin.exe`.

When a Dynamics NAV client connects to the server for the first time, the program will prompt you to upload the license to the server.

### Adding the Extended Stored Procedure Manually

You can use a Microsoft tool such as Enterprise Manager to add the `xp_ndo.dll` file to the extended stored procedures already installed on SQL Server. The names of the extended stored procedures must be `xp_ndo_enumusergroups` and `xp_ndo_enumusersids`.

#### Note

.....  
If you want to change the security model that is used in the NAV SQL database, you must add both of the extended stored procedures to SQL Server.  
.....

To add the extended stored procedures:

- 1 Open Enterprise Manager.
- 2 Expand the server and expand the databases.
- 3 Expand the master database and the Extended Stored Procedures.

- 4 Click Action, New Extended Stored Procedure and the Extended Stored Procedures Properties window opens.
- 5 Enter the name of the extended stored procedure and browse to the location where it is currently stored.
- 6 Click Ok to add the extended stored procedure.

After the extended stored procedures have been successfully added to the server, you must grant execute permissions to the "public" role.

To grant execute permission to the "public" role:

- 1 Open Enterprise Manager.
- 2 Expand the server and expand the databases.
- 3 Expand the master database and the Extended Stored Procedures.
- 4 In the right side panel, right-click the Extended Stored Procedure called xp\_ndo\_enumusersids and select Properties.
- 5 Select Permissions and grant EXEC rights to both "public" and "guest."

Repeat this procedure for the other extended stored procedure.

### Installing Multiple C/SIDE Database Servers

The next challenge you may face is to install more than one C/SIDE Database Server on the same computer. Remember that the services that run on the server machine generally use the TCP/IP protocol. The trick to installing more than one service on the same computer is to get the two services to communicate on different levels using the same protocol.

When you use TCP/IP, the port must be changed for the services. To change the port, you must change the `services` file on the server. This file is normally located in the following directory on Windows XP:

`C:\WINDOWS\system32\drivers\etc\services.`

If you are not familiar with changing this file, you may want to find someone who is knowledgeable about TCP/IP to help you. Once you have changed this on the server, it will have to be changed on every client as well. Again, you can change the shortcuts on each client to specify the server name.

#### Note

.....

There can be certain restrictions that prevent a customer from having more than one C/SIDE Database Server running at their site. A single license file is meant to be used with one, and only one, server.

.....

### Troubleshooting the Database Connection

A database connection problem can occur when your C/SIDE Database Server is running correctly, but the clients cannot connect to it.

### TCP/IP

The following steps will help you determine why your Dynamics NAV clients cannot connect to your C/SIDE Database Server when you are using the TCP/IP protocol:

- 1 Launch the Dynamics NAV client and attempt to connect to the C/SIDE Database Server using the name of the Service (excluding the "C/SIDE Database Server" prefix).

If this succeeds, the client can connect to the server and the service through TCP/IP using the name of the service. If this fails, Dynamics NAV is having a problem accessing TCP/IP. It could be a problem with the host's file. You must continue to the next step.

- 2 Open a command prompt and ping the IP address of the server (syntax: ping ###.###.###).

If this succeeds, this client computer can connect to the server machine. Continue to step 3. If this fails, the problem is the TCP/IP connection to the server computer and not the Dynamics NAV program. You cannot continue.

- 3 Go to a command prompt and ping the name of the service (excluding the "Dynamics NAV Server" prefix; syntax: ping servicename). Note that you are not pinging the service. You are actually pinging the server in another way.

If this succeeds, the host's file is set up correctly. Your setup is correct.

If this fails, the host's file does not contain an entry that tells TCP/IP where the service is running. Change the host's file to include an entry that has the name of the service (excluding the "C/SIDE Database Server" prefix) and the correct IP address of the server.

### Service Packs and Security Updates

.....

The installation is not complete until you have installed the latest service packs and applied the latest security updates to your system. Keeping your system up-to-date by installing the latest service packs is one of most important things you can do in managing the security of your system.

You should visit Microsoft Update and install all the relevant updates on every computer in your Dynamics NAV installation. We also recommend that you enable Automatic Updates on each computer so that they can receive security & critical updates automatically.

.....



## **Chapter 2**

### **The Server Options**

In this chapter, you learn about some of the important differences between running the Dynamics NAV C/SIDE Client on C/SIDE Database Server and on the SQL Server Option.

This chapter contains the following sections:

- The Different Server Options

## 2.1 The Different Server Options

As stated earlier, Dynamics NAV can run on two different servers – C/SIDE Database Server and Microsoft SQL Server.

This chapter is a brief introduction to the two server options that Dynamics NAV supports. It describes some of the most important differences between the two server options and explains some of the advantages that the SQL Server Option has over C/SIDE Database Server.

To the client these two server options look and perform exactly the same. However, there are some important differences between them including:

- the way you create a database.
- the backup facilities that are available.
- the ability to access the data in the database with third party tools.
- the security features.
- the way that SIFT™ works.
- performance monitoring
- scalability
- multi-processor support

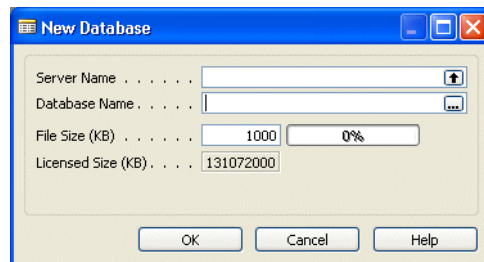
We will elaborate on these topics in the following sections.

### Creating Databases

When you create a database, you must follow different procedures and make different decisions depending on the server option that you are using.

#### C/SIDE Database Server

To create a database on C/SIDE Database Server, open a client and click File, Database New and the **New Database** window appears:

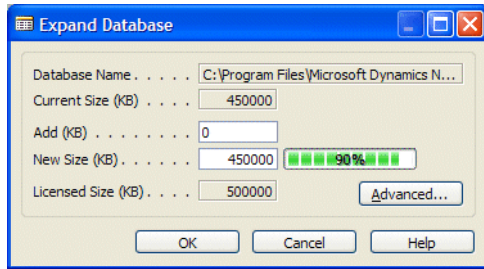


In this window you can:

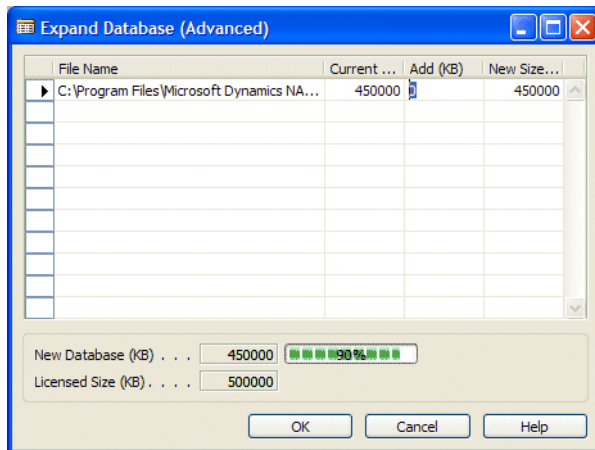
- select the computer on which the server is installed.
- specify the name and location of the database.
- specify the size of the database file.

If you want to divide the database into separate files and store them on different disks, you must expand the database.

To expand the database, click File, Database, Expand and the **Expand Database** window appears:



In this window you can specify the size of the file you want to add. If however you want to add more than one file and specify that these database files should be stored on different locations, click Advanced to open the **Expand Database (Advanced)** window:



Each line that you enter in this window is a new database file. We recommend that these files are all the same size because this ensures an even distribution of the database across the different files and drives.

The more physical disks that you use, the greater the speed and performance of C/SIDE Database Server. However, you can only divide a database into a maximum of 16 parts on C/SIDE Database Server. Any more parts than that and performance will suffer.

**Important**

.....

After you have created the database files you *must* restart the client and reconnect to the server to rebuild the list of free blocks. You can now restore the Dynamics NAV database backup file. The database will now be distributed evenly across the all the database parts and physical disks.

.....

**SQL Server Option**

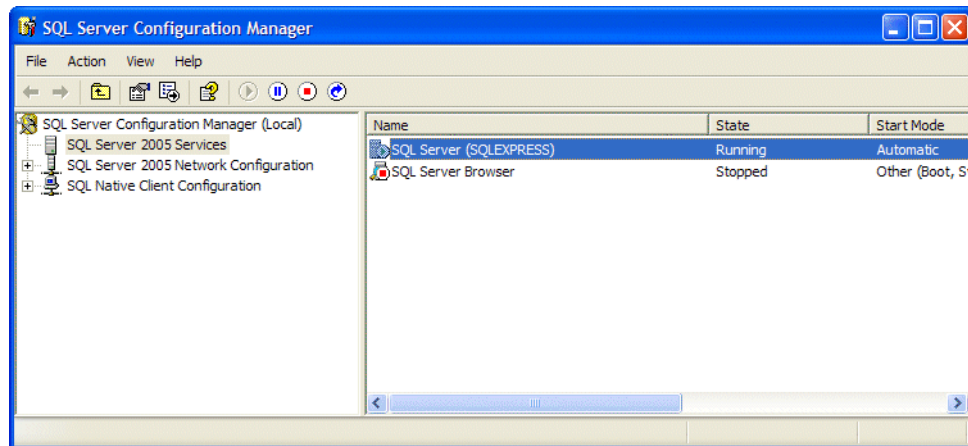
Before you can access SQL Server 2005 from Microsoft Dynamics NAV, you must set trace flag 4616 on SQL Server 2005. If you are running on SQL Server 2000 you do not have to enable the trace flag.

If your SQL Server installation has the default name MSSQLSERVER, then the trace flag is set automatically. You must remember to re-start SQL Server.

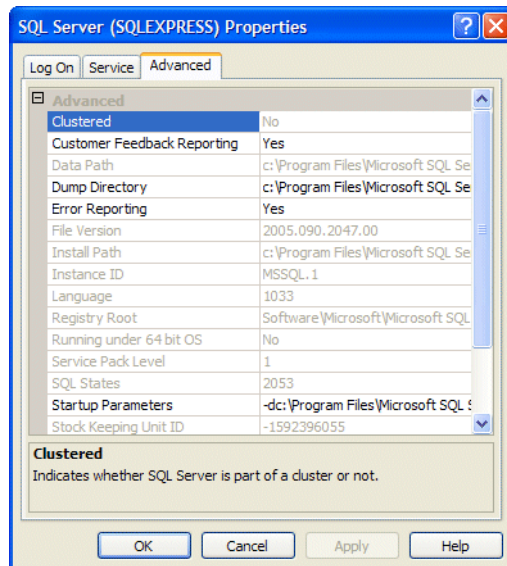
If your SQL Server installation has a different name, then follow the steps outlined below:

Enabling a trace flag To enable the trace flag:

- 1 Open SQL Server Configuration Manager.
- 2 In the left-hand panel, right-click SQL Server 2005 Services and click Open to see all the services:

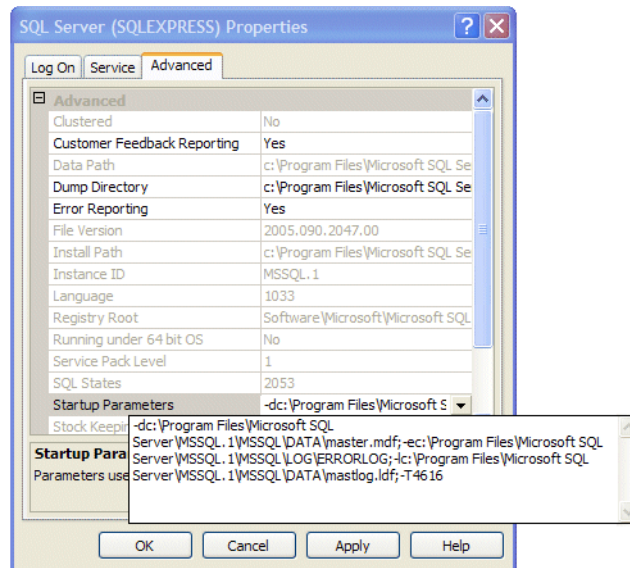


- 3 In the right-hand panel, right-click SQL Server (MSSQLSERVER) or SQL Server (SQLEXPRESS) and select Properties to open the **Properties** window:



- 4 In the **Properties** window, click the **Advanced** tab and expand the Advanced option if necessary.

5 Click the Startup Parameters property and open the drop down list:



6 Enter ;-T4616 at the end of the line in the drop down list.

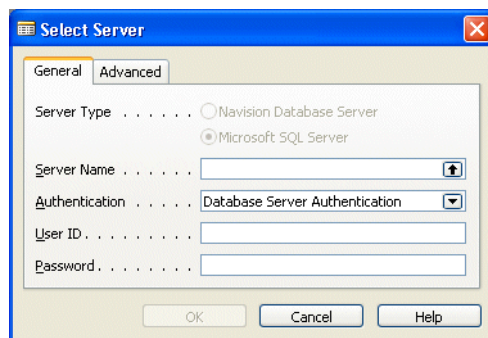
7 Restart SQL Server.

### Creating a Database

Creating a database in the SQL Server Option is a more streamlined process and allows you to specify some general database options.

To create a database in the SQL Server Option:

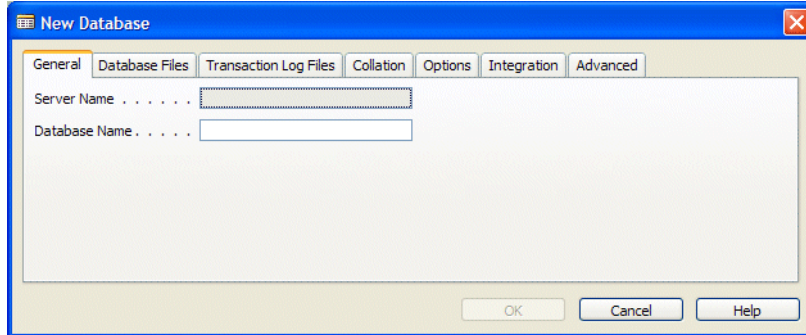
1 Click File Database, New and the **Select Server** window appears:



In this window you:

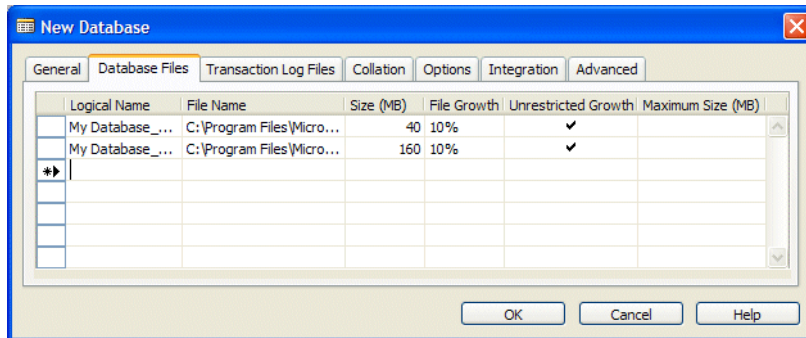
- select the server that you want to create the database on.
- select the type of authentication that you want to use to logon to the server.
- enter your User ID and password if you select Database Server Authentication.
- select the Net Type that you want to use (the **Advanced** tab).

2 When you have done this, click OK and the **New Database** window appears:



As you can see this window contains several tabs.

On the **General** tab you see the name of the server that you selected and you enter the name of the database that you are going to create.



On the **Database Files** tab, you can see that the first two database files have already been created. Each line that you enter in this window is a new database file. You can specify the size of and the location where the file is stored. Furthermore, you can specify how much the file can grow by, whether it can grow without restriction or whether it should have a maximum size.

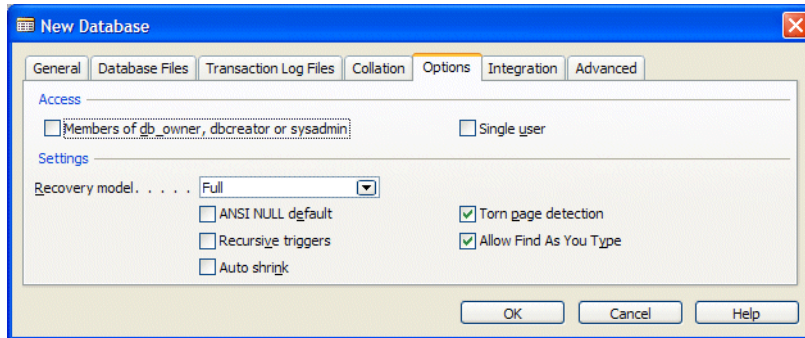
On the **Transaction Log Files** tab, you can specify the same things for the transaction log file(s) that are updated as you use the database. A transaction log is a file containing details of all the transactions that are performed in the database. This information is used after a database crash, for example, to recreate all the transactions that were completed since the last backup was made.

On the **Collation** tab, you specify the type of collation that is used in the database. The collation is probably the most important option that you set when you create a database. The collation determines the way that data is ordered and sorted in the database and this influences the performance of your database. Changing the collation after you have created the database is a time consuming process.

The SQL Server Option for Dynamics NAV allows you to choose between Windows collations and SQL collations:

- A Windows collation corresponds to the collations supported by the Windows operating systems, where they are known as Regional and Language Options.
- SQL collations are the original collations introduced in SQL Server 7.0 and are still supported for backwards compatibility.

We recommend that you use a Windows collation when you create a database. This type of collation closely follows the collation rules of the operating system.



On the **Options** tab, you can specify a number of database options. When you create a new database these options contain default values that are suitable for most databases. These options can be easily changed after you have created the database.

You can:

- limit access to the database by specifying that it is in single user mode or that only users who have been assigned certain SQL Server roles can access it.
- specify the following database settings:

Field	Comment
<b>Recovery model</b>	This setting determines the kind of information that is written to the transaction log and therefore the kind of recovery model that you want to use in this database. The options are: <i>Bulk-Logged</i> <i>Full</i> <i>Simple</i>
<i>Bulk-Logged</i>	If you select <i>Bulk-Logged</i> the transaction log file will only contain limited information about certain large-scale or bulk copy operations. The Bulk-Logged recovery model provides protection against media failure combined with the best performance and the minimal use of log space for certain large-scale or bulk copy operations. The backup strategy for Bulk-Logged recovery consists of: database backups. differential backups (optional).

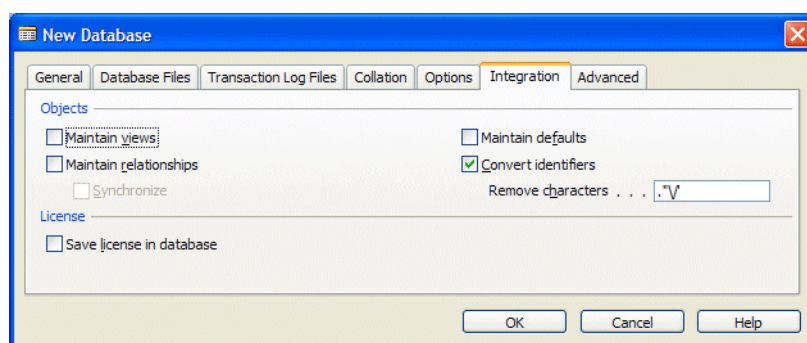
Field	Comment
<i>Full</i>	<p>If you select <i>Full</i>, the details of every transaction are stored in the transaction log and this information can be used when you apply transaction log backups. The Full recovery model uses database backups and transaction log backups to provide complete protection against media failure. If one or more data files are damaged, media recovery can restore all the committed transactions. Incomplete transactions are rolled back.</p> <p>Full recovery allows you to recover the database to the point of failure or to a specific point in time. All the operations, including bulk operations such as <code>SELECT INTO</code>, <code>CREATE INDEX</code> and bulk loading data, are fully logged to guarantee that the database is completely recoverable.</p> <p>The backup strategy for Full recovery consists of:</p> <ul style="list-style-type: none"> <li>database backups.</li> <li>differential backups (optional).</li> <li>transaction log backups.</li> </ul>
<i>Simple</i>	<p>If you select <i>Simple</i>, the database can be recovered to the point at which the last backup was made. However, you cannot restore the database to the point of failure or to a specific point in time. To do that, choose either the Full or Bulk-Logged recovery model.</p> <p>The backup strategy for Simple recovery consists of:</p> <ul style="list-style-type: none"> <li>database backups.</li> <li>differential backups (optional).</li> </ul>
<b>ANSI NULL default</b>	<p>This setting controls the database default <code>NULL</code> settings for column definitions and user-defined data types.</p> <p>When you select this option, all user-defined data types or columns that have not been explicitly defined as <code>NOT NULL</code> default to allow <code>NULL</code> entries. Columns that have been defined with constraints follow the constraint rules regardless of this setting.</p>
<b>Recursive triggers</b>	<p>When you select this option, triggers fire recursively. Triggers can have two different types of recursion:</p> <ul style="list-style-type: none"> <li>Direct recursion, which occurs when a trigger fires and performs an action that causes the same trigger to be fired again.</li> <li>Indirect recursion, which occurs when a trigger fires and performs an action that causes a trigger on another table to fire. This second trigger updates the original table, causing the first trigger to fire again.</li> </ul>
<b>Auto close</b>	<p>This setting is used for determining whether or not the database is closed and shut down properly when all processes in the database are complete and the last user exits the database, thereby freeing resources.</p> <p>The auto close option is useful for databases using the SQL Server Desktop Edition because it allows database files to be managed as normal files. They can be moved, copied to make backups, or even e-mailed. The auto close option should not be used for databases that are accessed from an application that continuously makes and breaks connections to SQL Server. Closing and reopening the database between each connection will impair performance.</p>



Field	Comment
<b>Torn page detection</b>	<p>When you select this option, SQL Server can detect incomplete Input/Output operations that have been caused by power failures or other system outages.</p> <p>Torn pages are usually detected during recovery because any page that was written incorrectly is likely to be read by recovery.</p> <p>If a torn page is detected, an I/O error is raised and the connection is terminated. If a torn page is detected during recovery, the database is marked suspect. The database backup should then be restored, and any transaction log backups should be applied.</p>
<b>Auto shrink</b>	<p>This setting determines whether or not database files are subject to periodic shrinking. Both data files and transaction log files can be shrunk automatically by SQL Server.</p> <p>When you select this option, SQL Server will automatically shrink a data file or transaction log file when more than 25 percent of the file is taken up by unused space. The file is shrunk until only 25 percent of the file consists of unused space, or to the size of the file when it was created, whichever is greater.</p> <p>Dynamics NAV performs slightly better when this setting is not selected.</p>
<b>Allow Find As You Type</b>	<p>This setting determines whether or not you can use the Find As You Type option when using the Find function to find an entry in a table or form. Using the Find As You Type facility can affect performance because requests are sent to the server for each character that is typed.</p>

These options can be changed later.

On the **Integration** tab, you can set some database settings that affect the way Dynamics NAV integrates with SQL Server and external tools. These options can be changed later.



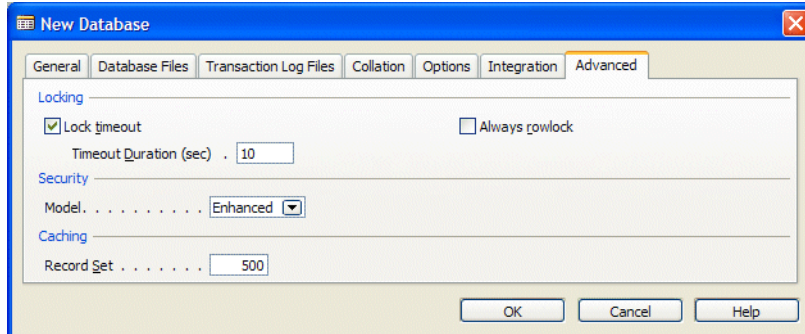
The **Integration** tab is divided into two sections and contains the following settings:

Field	Comment
<b>Maintain Views</b>	<p>This setting determines whether or not SQL Server will create and maintain a view for each language ID that is added to a table or field in Dynamics NAV.</p> <p>If you select this option, external tools can use the SQL views to gain access to the <i>Caption ML</i> property of the object in the required languages rather than the name supplied in the table.</p>
<b>Maintain Relationships</b>	<p>This setting determines whether or not SQL Server will create and maintain foreign key constraints for each <i>TableRelation</i> property of a Dynamics NAV table.</p> <p>If you select this option, external tools will have access to the table relationships (foreign key constraints) that exist between the Dynamics NAV tables. These relationships are disabled and are not used to enforce data integrity but are intended for modelling purposes only.</p> <p>This manual contains more information about table relationships in the SQL Server Option.</p>
<b>Synchronize</b>	<p>This setting is linked to the Maintain Relationships setting and is only active if you have already decided to create and maintain the table relationships on SQL Server.</p> <p>This manual contains more information about table relationships in the SQL Server Option.</p>
<b>Maintain Defaults</b>	<p>This setting determines whether or not SQL Server will create and maintain default constraints for each field of a Dynamics NAV table.</p> <p>If you select this option, external tools can use the defaults when inserting data into or modifying data in Dynamics NAV tables.</p>
<b>Convert Identifiers</b>	<p>This setting allows you to select the invalid characters in the names of all the SQL Server objects (tables, columns, constraints) in the database and map them to the underscore character. The <b>Remove characters</b> field contains a list of the characters that are converted to underscores. You can modify this list.</p> <p>When the conversion is completed, the database must be closed and reopened before you can use the new identifiers.</p> <p>For more information about identifiers and SQL Server, see Identifiers, Data Types and Data Formats in the SQL Server Option for Dynamics NAV on page 78.</p>
<b>Save license in database</b>	<p>This setting allows you to specify that the license file is uploaded and stored in the database instead of on the server. This is useful if you are hosting several databases with separate license files on the same server.</p> <p>If you select this option when you are creating or altering a database, you will be prompted to upload the license file to the database.</p>

The database files on SQL Server do not have to be the same size. After you have created the database files, you can restore the Dynamics NAV database backup file.

**Advanced**

The **Advanced** tab contains some settings that let you control the way locking and security is handled in the database.



The tab contains the following settings:

Field	Comment
<b>Lock timeout</b>	This setting allows you to specify whether a session will wait to place a lock on a resource that has already been locked by another session.
<b>Timeout duration (sec)</b>	This setting allows you to specify the maximum length of time that a session will wait to place a lock on a resource that has already been locked by another session. The default value is 10 seconds. You can change this value. If this option is left unchecked, the session will wait indefinitely.
<b>Always rowlock</b>	This setting allows you to specify that Microsoft Dynamics NAV always places row-level locks instead of page- and table-level locks.
<b>Model</b>	This setting allows you to specify whether this database uses the <i>Normal</i> or the <i>Enhanced</i> security model. The default setting is Enhanced
<b>Record Set</b>	This setting allows you to specify how many records are cached when Microsoft Dynamics NAV performs a <code>FINDSET</code> operation with the <code>ForUpdate</code> parameter set to <code>FALSE</code> . If a <code>FINDSET</code> statement reads more than the number of records specified here, additional SQL statements will be sent to the server and this will degrade performance. Increasing this value will also increase the amount of memory that the client uses to support each <code>FINDSET</code> statement. The default setting is 500.

**Backup Features**

The two server options offer very different backup and restore facilities.

**C/SIDE Database Server**

If you are running on C/SIDE Database Server, you can choose between two kinds of backup: a client based backup and a server based backup.

The Dynamics NAV client based backup is initiated by clicking Tools, Backup.

The advantages of using the Dynamics NAV backup function are:

- The system tests the database for errors, so incorrect information is not copied to a backup.
- The data is compressed, so it takes up as little space as possible.
- The system calculates how much space the backup will use.
- You can keep working in Dynamics NAV while you are making a backup.

### **HotCopy**

C/SIDE Database Server has a server based backup program called HotCopy. HotCopy is much faster than the client based backup facility. This program is installed with C/SIDE Database Server and is stored in the same directory as C/SIDE Database Server.

HotCopy can only be run from the server location and can only create backups on hard disks. You cannot make incremental or differential backups. You can make a backup of a database while clients are using it. The backups are file copies of the database and are not compressed.

### **SQL Server Option**

Microsoft SQL Server supports four different types of backup. You should choose the type of backup you will be using carefully in order to ensure that you get the level of security you require.

The four types of backup are:

- Database backup – this makes a backup of the entire database.
- Transaction log backup – this makes a backup of the entire transaction log.
- Differential backup – this makes a backup of all committed entries since the last database backup.
- File and filegroup backup – this makes a backup of individual files or filegroups within a database.

These can be combined to form many different types of backup and restore procedures. This allows you to design a backup and restore strategy that fits your database needs.

For more information about SQL Server backup and restore strategies, consult Microsoft's SQL Server documentation.

You can also use the Dynamics NAV client based backup/restore tool when you are running on the SQL Server Option. However, the SQL Server backup/restore system is server-based and is therefore considerably faster than the Dynamics NAV backup/restore tool, which is client-based.

It is possible to restore a SQL Server backup of a Dynamics NAV database directly into SQL Server without using Dynamics NAV. You can also create a database directly in SQL Server without first having to create it in Dynamics NAV and then restore a SQL Server backup of a Dynamics NAV database directly into the database on SQL Server.

SQL Server allows you to make backups when the system is in use. With SQL Server, you can also automate many of your administrative tasks, including making backups. SQL Server allows you to establish a database maintenance plan (with the help of a wizard) that includes database optimization, integrity tests and a backup plan.

One of the great advantages that SQL Server has over C/SIDE Database Server is its ability to record a transaction log. Transaction logs give SQL Server a roll forward capability that you can use to recover all the committed transactions that were carried out up to the point of failure. Roll forward is achieved by restoring your last database backup and applying all subsequent transaction log backups to recreate these transactions.

In such cases, only uncommitted work (incomplete transactions) will be lost, provided the active transaction log is also backed up and applied. The active transaction log also contains details of all uncommitted transactions. When you apply the active transaction log backup, SQL Server will roll back the uncommitted transactions.

## SIFT

SumIndexField Technology (SIFT) has been designed to improve performance when carrying out certain activities such as calculating customer balances. In traditional database systems this involves performing a series of database calls and calculations before arriving at a result.

### C/SIDE Database Server

The power and efficiency of SIFT on C/SIDE Database Server makes calculating sums for numeric columns in tables extremely fast, even in tables that contain thousands of records. This powerful feature is used throughout the Dynamics NAV application.

Let us say you want the sum of all the values in the **Amount** field of a table. In a conventional system, the Database Management System (DBMS) is forced to access every record and add each value in the **Amount** field, a very time-consuming operation in a database with thousands of records. In Dynamics NAV, you create a FlowField, define the calculation formula of this FlowField to sum the **Amount** field and then the DBMS only needs to retrieve the value from the SumIndexField.

SIFT has been built into the index structure used on C/SIDE Database Server and the more SumIndexFields that are added the larger the index becomes. However, the time used to maintain the accumulated sum for SumIndexFields is negligible due to a special index structure used in the DBMS.

### SQL Server Option

The way that SIFT is implemented in the SQL Server is much different than the way it is implemented in C/SIDE Database Server.

In the SQL Server Option, the SIFT system has been reproduced by creating extra so-called SIFT tables on SQL Server. A SIFT table is a SQL Server table that is created and maintained automatically by Dynamics NAV and used to store pre-calculated totals based on the values that are stored in SumIndexFields in standard Dynamics NAV tables. A SIFT table is created for every standard Dynamics NAV table key that has at least one SumIndexField associated with it. No matter how many SumIndexFields are associated with a key, only one SIFT table is created for that key.

This means that every time you update a key or a SumIndexField in a Dynamics NAV table all of the SIFT tables that are associated with this Dynamics NAV table must also be updated.

If you have a very dynamic Dynamics NAV table that is constantly having records inserted, modified and deleted, the SIFT tables that are associated with it will also have

to be updated constantly. The SIFT tables can get very large, both because of the new records that are entered and because the records that are deleted from the Dynamics NAV table are not removed from the SIFT tables. This can also badly affect performance, particularly when the SIFT tables are queried to calculate sums.

This means that the number of SIFT tables that you create can affect performance.

For a more detailed explanation of the way that SIFT works, see `SumIndexFields` on page 483.

## Performance Monitoring

Another area in which the two server options differ is the way in which you monitor performance.

The Client Monitor is the most important tool that you can use when you want to monitor the performance of your application. This tool can be used with both server options. When you use the Client Monitor with the SQL Server Option it contains some extra options and fields that give you more insight into how your application is performing.

The Client Monitor is an important tool for troubleshooting performance and locking problems. You can also use it to identify the worst server calls and to identify index and filter problems in the SQL Server Option. The Client Monitor and the Code Coverage tool now work closely together allowing you to easily identify, for example, the code that generated a particular server call.

Dynamics NAV also contains a debugger that you can use to refine functions that you write in C/AL code. The debugger can also be used with both server options.

When you are using the SQL Server Option you can supplement these tools with the SQL Server Error Log. By enabling trace flags 1204 and 3605, you generate extra diagnostic error messages in the error log. These give you information about the type of locks that are involved in a deadlock.

For a detailed description of how to use these tools and of performance troubleshooting in general, see the manual *Performance Troubleshooting Guide* that is available on the Dynamics NAV Tools CD. The Tools CD also contains some extra tools that you can use for troubleshooting.

## Other Differences

Other differences between the two server options are:

- Scalability  
Another one of the main differences between the two server options is scalability. The SQL Server Option can support more simultaneous users than C/SIDE Database Server.
- Security  
The SQL Server Option supports record level security – see "Supporting Record Level Security" on page 545.
- Multi-Processors  
C/SIDE Database Server doesn't make use of multi-processors while SQL Server does.

- Accessing the Database with Third Party Tools

It is much easier to access data in the database with third party tools when you are running on the SQL Server Option for Dynamics NAV.





## **Chapter 3**

### **The Dynamics NAV Security Model**

This chapter introduces you to the Dynamics NAV security system. It explains the different layers of security that exist in Dynamics NAV and how they work.

There is also a brief a description of the business and functional areas that exist in Dynamics NAV. The Dynamics NAV license system is also explained.

This chapter contains the following sections:

- Security
- Business Areas and Granules

### 3.1 Security

The Dynamics NAV security system allows you to control which objects (tables and so on) each individual user can access within each database. You can specify the type of access that each user has to these tables and records – whether they are able to read, modify or enter data.

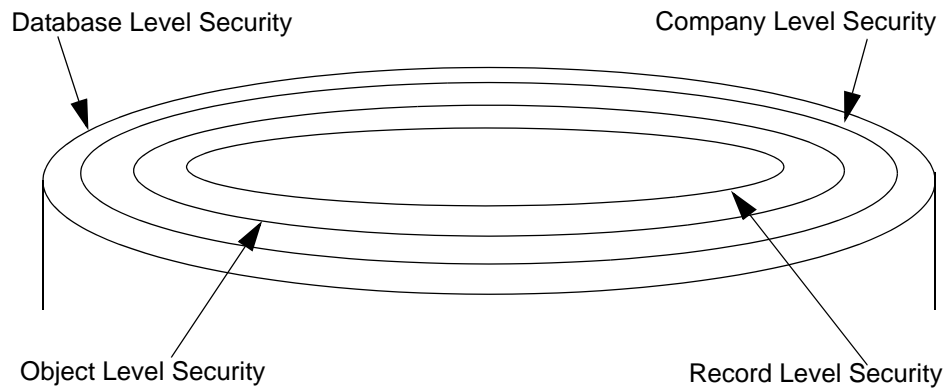
Furthermore, in the SQL Server Option you can specify which particular records that are stored in these tables each individual user is allowed to access. In other words, permissions can be allocated at both table level and at record level in the SQL Server Option for Dynamics NAV.

The Dynamics NAV security system contains information about the permissions that have been granted to each individual user who can access each particular database. This information includes the roles that the users have been assigned as well as any particular permissions that they have been granted as individual users.

The Dynamics NAV security system, even though it is a homogenous integrated system, can be said to consist of four different levels of security:

- Database Level Security
- Company Level Security
- Object Level Security
- Record Level Security

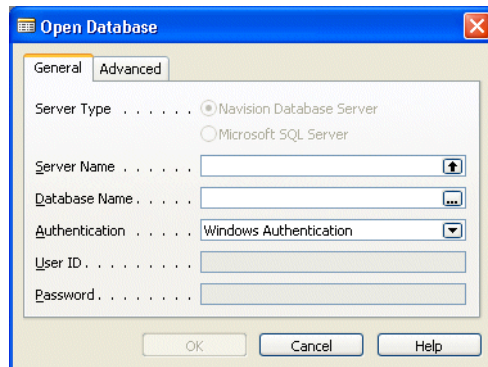
Graphically these can be represented as the layers of an onion where the central layer is the records in the database:



#### Database Level Security

The first layer of security that you encounter when you open Dynamics NAV is database security. When you have started the program and try to open a database, your credentials are checked and if you have not been granted permission to open the database you receive an error message informing you of this fact.

To open a database, click File, Database, Open and the **Open Database** window appears:



In this window you use the AssistButtons to locate the server that you want to access and the database that you want to open. If you are running on C/SIDE Database Server, you can select the server from a list of C/SIDE Database Server. If you are running on SQL Server, you can select the server from a list of SQL Servers.

In the Authentication field, you select the type of authentication that you want to use to verify your credentials and give you access to the database.

Dynamics NAV supports two kinds of authentication:

- Windows Authentication
- Database Authentication

The authentication that you must use depends on which kind of login you have been granted.

## Windows Logins

Users are given a Windows login when you use Windows authentication to control access to Dynamics NAV. Windows logins in Dynamics NAV correspond to the Windows users and groups of the Windows domain. These are administered and listed in a separate table and window.

With Windows authentication, when a user tries to connect to a server and open a database, they do not have to supply a user ID or password. Dynamics NAV automatically asks Windows to confirm whether or not this user, who has already logged on to the network, has a valid Windows account and whether this account gives them the right to access this particular server.

If the user is allowed to access the server, Dynamics NAV checks to see if the user has been assigned a Windows login within Dynamics NAV. If the user has a Windows login, they will be granted access to the database.

The user will be granted access to Dynamics NAV and be given the permissions specified for that Windows user and those specified for any Windows groups of which they are a member.

If the user does not have a valid Windows account or if their account does not include permission to log on to the Dynamics NAV database, authentication fails and the user receives an error.

### Database Logins

Users are given a database login when they have their own user ID and password in Dynamics NAV. They must enter their user ID and password to access the database.

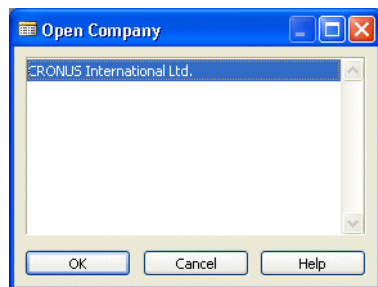
In the SQL Server Option they must also have a login on SQL Server. SQL Server carries out its own authentication of the user's ID and password. SQL Server does this by checking whether a SQL Server login with this user's ID and password has been created. This login must first have been created by a SQL Server administrator, with a SQL Server tool. If a SQL Server login has not been set up, authentication fails and the user receives an error.

The user is granted access to the server after their login has been authenticated. Database security then validates the user's permissions by checking the database user accounts on the server. The permissions that the user has been granted to the various objects within the database, such as tables, are determined by the information contained in the user's database user account. It also contains information about any additional permissions that the user may have been granted to alter the database itself.

### Company Level Security

After you have gained access to the database, you can open the company that you want to work with.

To open a company, click File, Company, Open and the **Open Company** window appears:



This window lists all of the companies that have been created in the current database and that you have been given access to. A Dynamics NAV database can contain several companies. These companies can use their own tables and they can also share some tables with each other.

Select the company that you want to open, click OK and the company opens. If there are companies in the database that you have not been given permissions to access, you will not be able to see them in this window.

### Object Level Security

You have now opened a company and your ability to work in it is still determined by the Dynamics NAV security system.

The Dynamics NAV security system consists of roles and permissions that you can assign to the users who have access to the company. The security roles in Dynamics NAV determine the access you have and the tasks you can perform on the objects that exist in the database.

The Dynamics NAV security system divides the database into the following objects:

Object	Description
Table Data	The actual data that is stored in the tables.
Table	The tables themselves.
Form	The forms that are used to view and enter data.
Report	The reports that are used to present the data.
Dataport	The dataports that are used to import and export data.
Codeunit	The codeunits that are used in the database.
XMLport	The XMLports that are used to import and export data in XML format.
MenuSuite	The object that contains the menus that are displayed in the Navigation Pane.
System	The system tables in the database that allow the user to make backups, change license file and so on.

The various roles that exist in Dynamics NAV determine the tasks that you can perform on these objects. The following picture illustrates how these permissions are allocated:

Object T...	Object ID	Object Name	Read ...	Insert...	Modif...	Delete...	Execu...
Table Data	5619	FA Journal Template	Yes	Yes			
Table Data	5620	FA Journal Batch	Yes	Yes		Yes	
Table Data	5621	FA Journal Line	Yes	Yes	Yes	Yes	
Table Data	5622	FA Reclass. Journal Template	Yes	Yes			
Table Data	5623	FA Reclass. Journal Batch	Yes	Yes		Yes	
Table Data	5624	FA Reclass. Journal Line	Yes	Yes	Yes	Yes	
Table Data	5625	Maintenance Ledger Entry	Indirect	Indirect	Indirect		
Table Data	5626	Maintenance	Yes				
Table Data	5629	Ins. Coverage Ledger Entry	Indirect		Indirect		
Table Data	5648	FA Allocation Dimension	Yes				

This picture shows some of the permissions that have been granted to the FA – Journal, Post role. As you can see this role has been granted permission to perform various tasks on a long list of objects. The permissions that a role can have on an object are:

Permission	Description
Read	You can read this object.
Insert	You can insert data into this object.
Modify	You can modify data in this object.
Delete	You can delete data from this object.
Execute	You can run this object.

If you have been granted permission to read a form, you can open it and view the data that it displays. If, however, you do not have write permission you are not allowed to enter data into this form.

Sometimes, when you open a form it displays information that is drawn from several tables. However, to access this form you must have permission to view all the data displayed by the form. You might not have permission to read directly from all the tables that the form uses. In this case you must have what is known as indirect permission to read from the tables in question. Having indirect permission to a table means that you cannot open the table and read from it but can only view the data it contains indirectly through another object, such as a form or report, that you have direct permission to access.

Dynamics NAV comes with a number of standard predefined security roles. You can use these roles as they are or you can change them to suit your particular needs. You can also create your own security roles and give them the permissions that you want.

### **Record Level Security**

Record level security is a system that allows you to limit the access that a user has to the data in a table by specifying that the user only has permission to access certain records in the table. Record level security is only available in the SQL Server Option for Dynamics NAV.

Record level security is implemented by applying security filters to the tables and the table data that a user has access to. You can specify, for example, that a user can only read the records that contain information about a particular customer and cannot access the records that contain information about any of the other customers.

For a brief description of how to apply security filters, see the manual *Installation & System Management: SQL Server Option for the C/SIDE Client*.

For a more detailed description of how to implement record level security, see "Supporting Record Level Security" on page 545.

### **Things to Remember about the Dynamics NAV Security System**

There are some important things that you must remember about the Dynamics NAV security system:

- The Dynamics NAV security system is initiated when you create the first login. Until you create the first login, anyone can carry out all the transactions they wish in a Dynamics NAV database. Furthermore, the first login you create must be that of a superuser. The superuser then owns and administers all access to this database from within Dynamics NAV.
- You can only grant permissions to other users that you already possess yourself. We therefore recommend that the user who administers security in Dynamics NAV should be a superuser.
- One of the first things that the superuser should do is create logins for the other people who will have access to the database and grant the appropriate permissions to these users.
- In Dynamics NAV a table can contain a FlowField that generates sums based on values that are stored in another table. In this case, the user must have permission to read both tables or they will not be allowed to read the first table.

For a more detailed description of how to manage security in Dynamics NAV including detailed instructions on how to create logins and roles, as well as how to grant and modify permissions, see the *Installation & System Management* manual for the server that you are using.

## 3.2 Business Areas and Granules

Now that you are familiar with the different levels of security that exist in Dynamics NAV, you should learn something about the business areas that Dynamics NAV contains and how your license file controls the access that you have to these areas and the functionality that they contain.

The standard Dynamics NAV application is divided into several business areas. Each business area consists of a number of functional areas. In turn, these functional areas contain a large amount of functionality and different customers will probably only need some of the functionality provided. To facilitate this, a Dynamics NAV license file consists of a number of granules. Each granule represents a small area of functionality.

This system makes it possible for each customer to purchase a license that only gives them and their employees access to exactly the functionality that they need.

Dynamics NAV contains the following business areas and functional areas:

<b>Business Areas</b>	<b>Functional Areas</b>
Finance Management	General Ledger Cash Management Receivables Payables Fixed Assets Inventory
Sales & Marketing	Sales Order Processing Marketing Inventory & Pricing Analysis & Reporting History
Purchase	Planning Order Processing Inventory & Costing Analysis & Reporting History
Warehouse	Orders & Contacts Planning & Education Goods Handling Inventory History
Manufacturing	Product Design Capacities Planning Execution Costing History
Jobs	Jobs Reports History Periodic Activities



<b>Business Areas</b>	<b>Functional Areas</b>
Resource Planning	Resources Reports History Periodic Activities
Service	Contact Management Planning & Dispatching Order Processing History
Human Resources	Employees Absence Registration Reports
IT Administrator	IT Administration Application Setup

Each of these functional areas represents an important field of activity. However, even though you have a General Ledger, you don't necessarily need all the functionality that is available in this area.

The following table lists the granules that are available in the General Ledger:

<b>Granule – Name &amp; Number</b>	<b>Description</b>
Basic General Ledger (3,010)	<p>You use this granule to set up a company and post to the general ledger. The granule provides you with the basic facilities necessary for setting up a company and posting to the general ledger: chart of accounts, general journals, VAT facilities, recurring journals and source codes. It also includes facilities for internal and external reporting.</p> <p>The granule allows you to post and report in the company's base currency. If you also purchase the Multiple Currencies granule, you can post and report in an additional currency as well.</p> <p>The granule allows two languages from the beginning – the English US language and the native language for the particular country/region.</p> <p>The granule allows 1 instance of NAV Application Server.</p> <p>This granule must always be included as part of the initial purchase of a solution because it includes 1 session and the first company.</p> <p>Requirements: The granule Dynamics NAV Version 3.XX</p>
Budgets (3,030)	<p>This granule allows you to work with budgets in G/L accounts. Once you have created a budget, you can print a balance compared to the budget showing variances by percentages. You can work with several budgets. Budgets are normally entered per period for the relevant G/L accounts. You can create, copy and work with any number of budgets at the same time. You can work with, for example, a 100% budget, a 110% budget, and so on.</p> <p>Requirements: Basic General Ledger</p>

<b>Granule – Name &amp; Number</b>	<b>Description</b>
Account Schedules (3,040)	<p>You use this granule for financial reporting. You can arrange reports based on the figures in the chart of accounts and budgets, but with a different arrangement of financial figures, texts or details than in the chart of accounts. The Account Schedules granule is like a filter for the chart of accounts that enables you to choose the accounts that you want to include (or not include). You can also use it to change the order of the accounts or combine the figures in various ways, and you can set up which columns to print. In addition, it is possible to make simple calculations.</p> <p>Requirements: Basic General Ledger</p>
Consolidation (3,050)	<p>This granule enables you to consolidate companies in Dynamics NAV. The companies can come from one or from several different Dynamics NAV databases or from another type of file. There are facilities for imports and exports of financial information in the Consolidation granule. If the data used is retrieved from several Dynamics NAV solutions, the granule is only used in the parent company itself.</p> <p>Requirements: Basic General Ledger</p>
Allocations (3,020)	<p>This granule allows you to allocate general ledger entries to combinations of accounts, departments and projects using allocation keys.</p> <p>The allocation keys can be based on amount, percentage or quantity. Allocations can be used for many purposes, for example, when allocating overhead (such as rent) to company profit centers.</p> <p>Requirements: Basic General Ledger</p>
Responsibility Centers (3,060)	<p>With this granule you can setup profit centers and/or cost centers. A company can sell items with specific prices and related to a responsibility center. The functionality provides the ability to tie a user to a responsibility center so that only sales and purchase documents related to the particular user are displayed. In addition, users get assistance with entering extra data, such as dimensions and location codes.</p> <p>Requirements: Multiple Locations</p>
Basic XBRL (3,070)	<p>With this granule you can export documents from Dynamics NAV in XBRL format and import XBRL taxonomies into Dynamics NAV from the Internet, e-mails or from other systems. XBRL is an XML-based specification that uses accepted financial reporting standards based upon standardized, underlying data tags. It is possible to map your general ledger to XBRL taxonomies, meaning that the same XBRL instance document can be used for various purposes, independent of the format required by the receiver of the document.</p> <p>Requirements: Basic General Ledger</p>

Granule – Name & Number	Description
Change Log (3,080)	<p>This granule enables you to log user changes made to Dynamics NAV master data.</p> <p>It is possible to log all direct modifications a user makes to the data in the database, except changes to 'working documents' such as journals and sales and purchase orders. The change log functionality makes it possible to get a chronological list of all changes to any field in any table (except the ones mentioned earlier) and to see who (what user ID) made the changes.</p> <p>Requirements: Basis General Ledger</p>

The information contained in this table does not necessarily correspond with the granule definition that is currently enforced by Microsoft Business Solutions.

Each of the other functional areas can be similarly broken down into areas of more specific functionality.

This means that your license file and the way that it is configured have an important influence on the functionality that is available to you in Dynamics NAV. Your license file and the granules that it contains determine which functional areas you have access to and which functions you can perform in these areas.

The advantage of this system is that the customer doesn't pay for a general license to run the product but instead only pays for the granules that they need to run their business.



**Part 2**  
**Fundamentals**



## **Chapter 4**

### **C/SIDE Fundamentals**

A C/SIDE® application is composed from seven types of application objects. Each type of application object is created using a specific tool called a designer. The application objects you create using these designers are all based on some general concepts. A fundamental knowledge of these concepts speeds up the C/SIDE application development process.

This chapter introduces you to the C/SIDE user interface and presents the general concepts that underlie C/SIDE application objects.

- The C/SIDE User Interface
- What Is a C/SIDE Application?
- The Physical and the Logical Database

## 4.1 The C/SIDE User Interface

In this section, you are introduced to the C/SIDE user interface and to some of the basic concepts that are relevant to application design such as the different object types.

### Designing Application Objects

All C/SIDE applications are based on seven different types of application object:

**Table** You use tables to store data. For example, a business application normally contains a customer table that stores information about each customer, such as, their name, address, phone number and the name of your contact person. Understanding tables is the key to using all the other objects.

**Form** You use forms to access the information that is stored in the tables. You use forms when you enter new information and when you view information that already exists in the database.

**Report** You use reports to present information. You use filters and sorting to select the data that you want to present in a report.

**Dataport** You use dataports to import data from and export data to external text files.

**XMLport** You use XMLports to import and export data in XML format.

**Codeunit** A codeunit contains user-defined functions written in C/AL code. C/AL is the application language you use to write functions in C/SIDE. The functions that a codeunit contains can be used from the other objects in your application. This helps to minimize application size because the same code can be reused again and again.

**MenuSuite** A MenuSuite object contains the set of menus that are displayed in the Navigation Pane.

#### Note

.....  
Every application object is identified by an ID number. There are, however, restrictions about which numbers you should use when you create your own application objects. Please contact your Microsoft Certified Business Solutions Partner for more information.  
.....

### The Object Designer

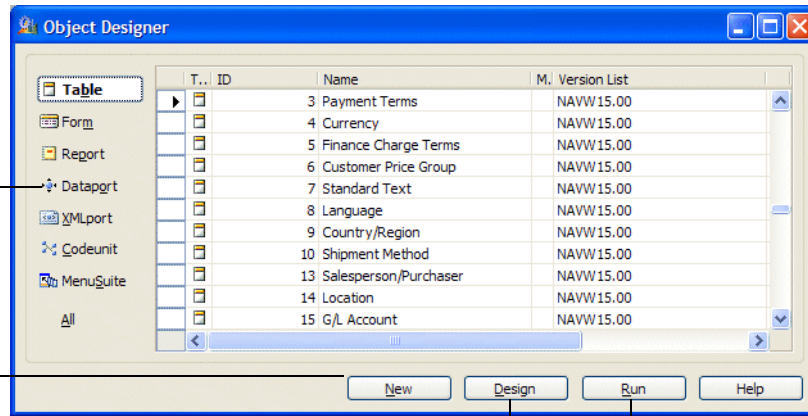
The Object Designer is the main tool for developing C/SIDE applications, and the tool you use to:

- design new tables, forms, reports, dataports, XMLports and codeunits. (You design MenuSuite objects in the Navigation Pane Designer.)
- view existing application objects. However, you view the content of a MenuSuite object in the Navigation Pane Designer (or in the Navigation Pane at runtime).



- modify existing application objects (with the exception of MenuSuite objects, which are modified in the Navigation Pane Designer).
- run an application object (with the exception of XMLports and MenuSuite objects).

This is where you access the designers for different objects. You simply choose the type of object you want to work on here.



Create a new object  
Change the design of the current object

Run the current object

The following table lists the tools that are available in the Object Designer.

Use the...	When working on ...
Table Designer	tables
Form Designer	forms
Report Designer	reports
Dataport Designer	dataports
XMLport Designer	XMLports
C/AL™ editor	codeunits
Navigation Pane Designer	MenuSuite objects

With the exception of MenuSuite objects, you can work on any number of application objects at the same time, each in its own designer. This means that you can run multiple instances of these designers. For example, if you work on three new forms at the same time, then each form is displayed in its own form designer. You can, however, only run one instance of the Object Designer.

### The Object Designer and MenuSuite Objects

When you click the MenuSuite button in the Object Designer, select a MenuSuite object and then click Design, the content of the object is displayed in the Navigation Pane Designer. This is where you modify the content.

When you click the MenuSuite button and then click New, a dialog opens asking you to specify which menu suite level (for example, Developer or Administrator) that you want to create. If you have created a MenuSuite object for each level that you have permissions for, the New button will be disabled. This is because only one MenuSuite object is allowed per level. Once you have made a selection, the Navigation Pane Designer opens and you have the appropriate design rights.

Is Dynamics NAV  
object oriented?

C/SIDE is not object oriented but object-based. This is an important distinction. In an object oriented language or environment, a developer can create new types of objects based on the ones already in the system. In Dynamics NAV, you can only create objects that are either tables, forms, reports, dataports, XMLports, codeunits or menu suite objects.

Because there are a limited number of application objects, C/SIDE works faster and more efficiently. Your design work is also easier, because you know exactly what you have to work with. But the greatest benefit is stability. It is actually difficult to create a severe bug in C/SIDE.

## 4.2 What Is a C/SIDE Application?

You use C/SIDE to create accounting and business management applications. A C/SIDE application consists of the same objects as a C/SIDE database. But, whereas a database is simply a collection of application objects, an application is a set of application objects that are tied together to form a coherent whole.

### General C/SIDE Concepts

With the exception of the MenuSuite object, the different types of C/SIDE application objects are based on some general concepts. Some of these concepts are restricted to one type of application object, but others apply to several types. When you understand these fundamental concepts, you have a good foundation for creating your own applications.

The following table summarizes and explains what each type of application object is used for.

<b>Application Object Type</b>	<b>What is it used for?</b>	<b>Which concepts is it based on?</b>
Table	A table is used for storing the actual data. Typically a business application will have a Customer table that stores information such as name, address, phone number and contact person for each of your customers.	Properties, Fields, Keys, C/AL
Form	A form is used to access the information in your tables. Forms are used both when you enter new information and when you view existing information.	Properties, C/AL, Controls
Report	A report is used to present data that contains summary information. For example, you will use a report to print a list of customers.	Properties, C/AL, Controls, Dataltems, Sections, Templates, RequestForm
Dataport	A dataport is used to import and export information to and from other programs (a comma-separated file from a spreadsheet, for example).	Properties, C/AL, Dataltems, RequestForm
XMLport	An XMLport is used to import and export data in XML format.	Properties, C/AL
Codeunit	A codeunit contains user-defined functions written in C/AL code. These functions can be used from the other objects in your application. This minimizes the size of the application because the same code can be reused over and over again.	C/AL
MenuSuite	A MenuSuite object contains the menus that are displayed in the Navigation Pane.	

Description of the concepts

Here is a description of the basic C/SIDE concepts:

**Properties** Properties control the appearance and behavior of the application objects and all their sub-objects. Properties are used to control the appearance of data, specify default values, specify colors and define relationships.

**C/AL** C/AL is the language you use to write functions in C/SIDE. In the previous table, "C/AL" refers to functions written in this language.

**Triggers** A trigger is a mechanism that is built into an application object. When certain actions are performed on the application object, the trigger initiates an action. You can add your own C/AL code to the trigger to modify the default behavior of the application object or extend its functionality.

**Keys** Keys define the order in which data is stored in your tables. You can speed up searches in tables by defining several keys which sort information in different ways.

**Fields** A field is the smallest building block in your database. A field typically stores information such as a name or a number.

**Controls** Controls are objects on a form or report that display data, perform actions or enhance the appearance of the form. Typical examples of controls are command buttons and text labels.

**Request Form** A request form is a form that is used in a report. Before a report is run, a request form appears to let the user specify filters and options for the report.

**Template** A template defines the overall layout of a report.

**Data Items** A data item is a building block for defining a model of your data when you create a report. You use a hierarchy of data items to define which data the report will contain. A data item represents a table, and when you run a report, the system cycles through the records in the associated table. A data item can have one or more sections.

**Sections** A section is a substructure of a data item, where you place controls to display information. You will typically use sections that define the body, header, and footer in your report.

## 4.3 The Physical and the Logical Database

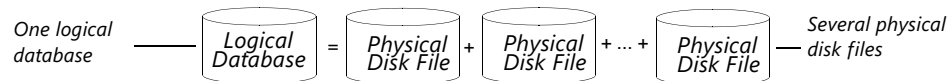
The previous section described the general concepts underlying the different types of application objects in C/SIDE. This section presents another view of C/SIDE applications. In this section we are only concerned with how the information in your application is structured.

When you use a database, you are not usually concerned with where each piece of data is stored, or what size it is. You just want to be sure that when you refer to a name, for example, the correct value is returned. This is why the C/SIDE database system provides a conceptual representation of data that does not include too many details of how the data is stored. An abstract data model is used for this conceptual representation. This data model uses logical concepts (such as objects, their properties and their relationships), which are much easier to understand.

This leads us to distinguish between the logical and the physical database. When we speak about the logical database we are concerned only with the structure of the data and the relationships between different pieces of information. That is, we do not deal with how these structures and relations are implemented. When we speak about the physical database, we only deal with how the structures in the logical database and the search paths between them are implemented.

In this book, the term database normally means the logical database, unless indicated otherwise.

What the user sees as a coherent set of information in the C/SIDE database system can be stored in several physical disk files, but this is transparent to the user. The following figure illustrates how one logical database can be physically stored on three hard disks but still comprise a single (logical) database.



### The Logical Structures in Your Database

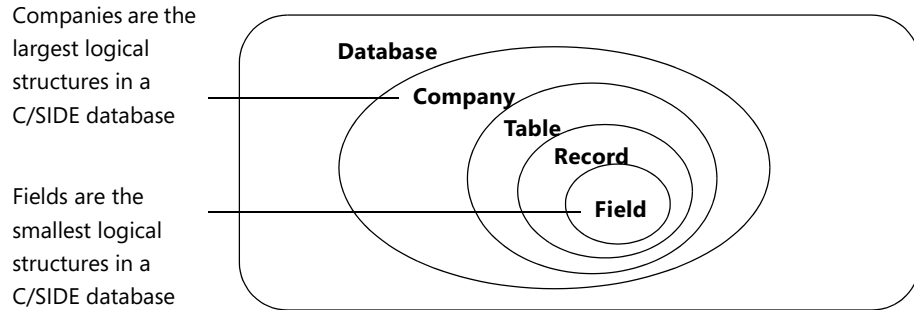
Access to the data is made possible by a well-defined logical organization composed of:

**Fields** Fields are the smallest logical structure in a C/SIDE database. A field holds a single piece of information, such as a name, "Joe," or an amount, "2,352.00." Any particular field can hold one specific type of information. (The C/SIDE database system distinguishes between 17 different types of information.) Fields are assembled into a structure called a record. On its own, a field is not very useful, as it can hold only a limited amount of information. Assembling these small bits of information into records produces a much more flexible "information-holder", which also keeps together fields that belong together.

**Records** A record is a logical structure assembled from an arbitrary number of fields. A record stores a single entry in the database. The fields in a record store information about important properties of the entry. Records are organized in tables.

**Tables** A table can be thought of as an N times M matrix. Each of the N rows describes a record and each of the M columns describes a field in the record. Tables are organized in companies.

**Companies** A company is the largest logical structure in a C/SIDE database. A company can be thought of as a sub-database; its primary use is to separate and group large portions of data together. A company can contain private tables as well as tables that are shared with other companies.



## **Chapter 5**

### **Designing a C/SIDE Application**

Carefully planning the details of your database applications will help you end up with a sound design. A properly designed application is easier to build and maintain.

This chapter provides guidelines for creating quality applications in C/SIDE using the well-known methodology of analysis, design, and implementation.

- Introduction to C/SIDE Application Design

## 5.1 Introduction to C/SIDE Application Design

Carefully planning the details of your database application will help ensure that your database has the best possible design. An application that has been properly designed is easier to build and maintain. This section contains some guidelines for creating quality applications in C/SIDE using the well-known methodology of analysis, design, and implementation.

Designing a C/SIDE database application usually includes the following three steps:

**Understanding the Problem** Make sure you understand the business problem you are trying to solve. Be sure you know who will be using the application and what they will be trying to accomplish.

**Designing the Tables** Begin by designing a data model to determine how the data is stored and how it can be most meaningfully utilized. The data model determines:

- which tables the database must contain.
- what kind of data you want to store in the fields of each table.
- how the data in the tables are related to each other.
- constraints that are necessary to ensure data integrity.

**Designing the Application** When you have designed the database tables, you are ready to begin designing the application itself. Designing the application involves:

- designing forms (to enter and retrieve data) and reports (to view and present data).
- creating C/AL code to connect the application objects.

These steps depend on each other. When you move from one step to another you often have to rethink some of the decisions you made in the previous step.

### Understanding the Problem

To decide which information you should store in the database, you have to understand the purpose of the database and how it will be used. The best way to do this is to talk to the people who will be using it. Involving the end user in this process, as early as possible, helps eliminate problems that can stem from misunderstandings about the purpose of the database. Interviewing the end users will help you understand the tasks they expect the system to perform. Based on this understanding, you can determine the kind of data and thereby the kind of tables, forms and reports that are necessary for completing these tasks. This will often be the most difficult part of the design process as well as the most important. The usefulness of the entire application depends on whether the tables, forms and reports have been designed correctly.

Your discussions with the end users will give you a lot of insight into the tasks they need to perform. You will then know what information the forms and reports should provide. This does not, however, necessarily tell you how the tables should be designed.

### Designing the Tables

The next important task is to divide the information you want to store in the database into basic categories such as customers, products, employees, and so on.



Before you create the tables you should define a data model. The data model must describe:

- the tables in the database.
- the fields in the tables.
- the relations between the fields in your tables.
- constraints for fields and relations.

The Entity-Relationship model (ER model) is a suitable tool for defining a data model. An ER model can map real-world situations to a relational database system such as C/SIDE.

An ER model divides all the elements of a real world situation into two categories: entities and relationships. An entity is a "thing" in the real world with an independent existence. An entity may be an object with a physical existence, such as a particular car or person, or it may be an object with a conceptual existence, such as a company or a job. Relationships describe how the entities are related.

To use the ER model, you must complete the following steps:

- 1 Identify the types of entities associated with your problem. Create tables to represent each of these types of entities.
- 2 Identify the properties of each type of entity and create fields in the tables to represent each of these properties.
- 3 Identify the relationships between the entities and add these relationships to the tables.

The following information is not a description of all the facets and implications of the ER model. However, it does provide an overview of the model and illustrates the benefits of applying a formalized design method.

### How Are ER Model Concepts Related to C/SIDE Concepts?

A real world problem usually contains groups of entity types that are similar. For example, consider a company that has hundreds of customers. All of the customers are entities. These customer entities share the same properties, but each entity will have its own values for the properties. Such similar entities define an entity type, that is, a set of entities with the same properties. When you implement the abstract ER model in C/SIDE, you transform all the abstract elements in your model into concrete representations. Each entity type corresponds to a table in C/SIDE and each of the entity's properties corresponds to a field in the table.

The following table summarizes how basic ER model concepts relate to C/SIDE concepts.

ER Model Concept	Corresponding Concept in C/SIDE
An entity type	A table
An entity	A record
A property	A field

### Determining Field Types

In the ER model, after you have identified the entity types and their properties, you determine the types of values these properties can have. In C/SIDE this corresponds to determining the data types of the fields in your tables.

#### Example

Your analysis using the ER model has revealed that you have an entity type describing your company's customers. This leads you to define a Customer table:

**Customer Table** —

Company Name	Contact Person	Phone	...	Payment Method
⋮	⋮	⋮	⋮	⋮

Your analysis shows that you need fields such as Company Name, Contact Person, Phone and Payment Method. So when you create the Customer table, you select the following data types:

Field Name	Description	Data Type
<b>Company Name</b>	Stores the name of the customer (for example, "Microsoft Business Solutions ApS").	Text
<b>Contact Person</b>	The contact person in the company (for example, "JLJ").	Text
<b>Phone</b>	The customer's phone number (for example, "45662111").	Text
<b>Payment Method</b>	The payment method for the customer (for example, "pay in cash").	text

For more information about the C/SIDE data types, see "Choosing Data Types" on page 63.

### Role of Keys in C/SIDE

The ER model places a very important constraint on the entities of a particular entity type (records in a table). This is the *key* or *uniqueness* constraint on the properties (fields). An entity type usually has at least one property which contains unique values for each individual entity. This property is used to uniquely identify each individual record. The following table shows how the ER model concepts are related to C/SIDE concepts.

ER-Model Concept	Corresponding Concept in C/SIDE
Constraints on the entities of an entity type	Constraints on the records in a table
The uniqueness constraint on entity properties	A key based on fields in a table

The records in a table must be arranged according to some criterion (that is, a key) so that C/SIDE can work efficiently with the data in tables. For example, an Employee table can be ordered according to the employees' social security numbers because this number uniquely identifies each employee.

In order for a field to be a key for a table, the uniqueness constraint must hold for every record in the table. This constraint prevents any two records from having the same value in the key field. It is not a constraint on a specific record, but a constraint on all the records in the table.

Sometimes, a key consists of several fields. In this case, the combination of the values in all the fields in the key must be unique for each record. Sometimes you can define several keys for a table. For more information about creating keys, see the section "How to Define a Primary Key" on page 71.

### Determining the Relationships

At this point in the process, you have carefully designed a number of tables to store individual types of information. In your final application you want to retrieve the information in a meaningful way. Very often an answer from your database will consist of information stored in several tables. To enable you to get these answers, C/SIDE uses relationships to link the tables that contain the related information.

Database terminology normally distinguishes between three types of relationships:

**One-to-Many Relationships** A record in Table 1 can have more than one matching record in Table 2, while a record in Table 2 can have no more than one matching record in Table 1. This is the most common type of relationship in a relational database.

**Many-to-Many Relationships** A record in Table 1 can have more than one matching record in Table 2, and a record in Table 2 can have more than one matching record in Table 1. This represents a problem in database design and may signal an inefficient design. Normally, you break down a many-to-many relationship into two one-to-many relationships.

**One-to-One Relationships** A record in Table 1 can have no more than one matching record in Table 2, and a record in Table 2 can have no more than one matching record in Table 1. This kind of relationship is inefficient and can often be avoided by simply combining the two tables.

### Assuring the Quality of the Design

When you are defining the tables and setting up relationships, you often have to select from among several possible solutions. To make sure that you select the most appropriate solution, you need a way to measure the quality of your design. You use the normalization process to measure the quality of your design. The normalization process takes your design through a series of tests to verify whether it belongs to a certain normal form. There are six normal forms. Most texts on relational database design explain how to obtain these normal forms. For more information, see the books listed at the end of this chapter.

### Designing the Application

After you have designed the tables, you are ready to begin work on the application itself. The analysis phase has already given you an overview of the answers that the application is expected to provide. The table design phase has given you a clear understanding of where and how the information will be stored. Based on this understanding, you are ready to begin assembling the entire application.

This part of the application design involves:

**Creating Forms** Forms are used to present or collect information. You have access to a number of design elements, such as text, data, pictures, lines, and color.

**Creating Reports** Reports are used to present data as printed documents. When you want to present summary information, reports are more flexible than forms.

**Creating C/AL Codeunits** Codeunits are containers for storing C/AL code. When you put the code into a codeunit, you can reuse the same algorithms many places in your application. This reduces the size of the application and makes it easier to maintain.

**Testing and Refining the Application** Before you release your application, you have to analyze your design for errors. This is normally an iterative process. When you have completed all this you will have a useful application. If you took the time to plan all the steps of the application design carefully, you will also have an application that is fully documented. This will be a great help when you need to make adjustments and additions to the application in the future.

### **Recommended Books on Database Design**

Some of the most well-known books about relational database design are:

C. J. Date. An Introduction to Database Systems. Addison-Wesley Publishing Co.

Elmasri, R. A. and Navate, S. B. Fundamentals of Database Systems.  
Benjamin/Cummings.

Dutka, A. F. and Hanson, H. H. Fundamentals of Data Normalization. Addison-Wesley Publishing Co.

Michael J Hernandez, Database Design for Mere Mortals. Addison-Wesley Publishing Co.

**Part 3**  
**Tables**



## Chapter 6

### Table Fundamentals

Tables are the fundamental objects in any database. This is true no matter what kind of data you need to store. When you create a new database, you begin by building the tables. Later on, you create forms and reports in order to access and view the data in the tables.

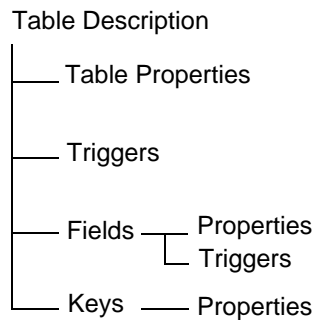
This chapter explains how to design appropriate tables to store your data.

- What Is a Table?
- Viewing and Modifying Properties
- Defining Keys
- Identifiers, Data Types and Data Formats in the SQL Server Option for Dynamics NAV
- Saving tables and Viewing Sorted Data
- Special Table Fields
- Dividing the Database into Companies





The following figure illustrates that a table description contains *properties*, *triggers*, *fields* and *keys* and shows how these are related:



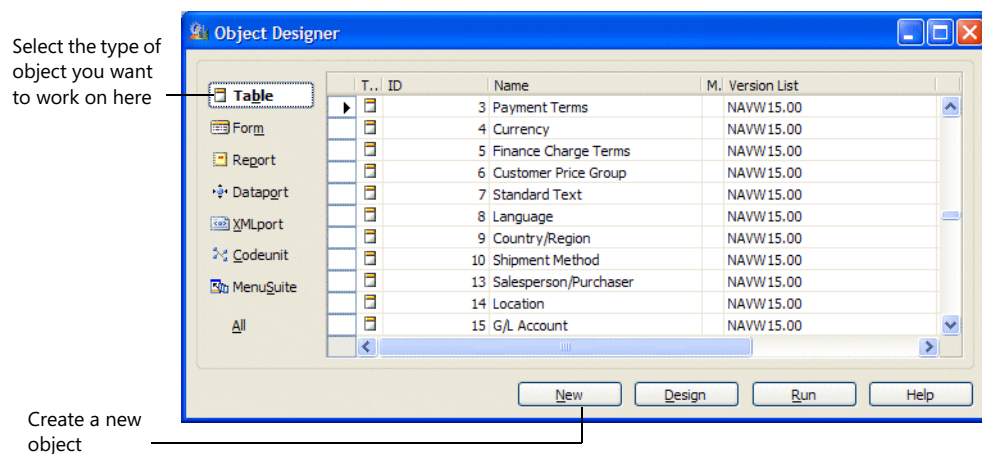
The table description contains some properties that are related to the table, others that are related to the fields in the table, and other properties related to keys. You can also see that triggers are defined both for the table and for the fields in the table.

## Creating a Table

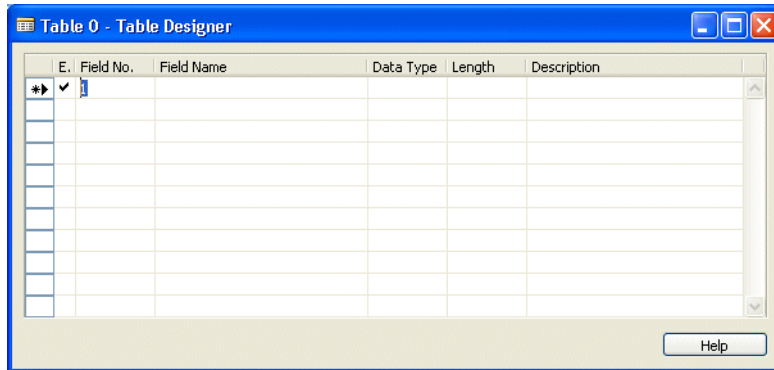
When you first create a table, it does not contain any data. Nevertheless, when you create the table you must also decide what types of information you want to store in it. The information is held in fields, and each field can be declared as one of the data types that are available in Dynamics NAV.

To create a table:

- 1 Click Tools, Object Designer (SHIFT+F12) and C/SIDE opens the Object Designer:



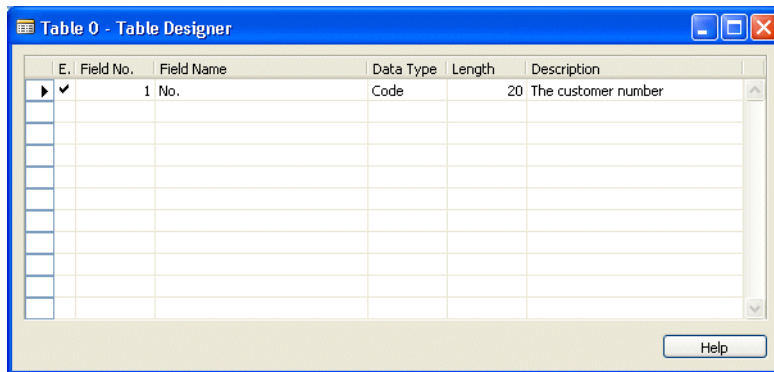
2 Click Table, New and the Table Designer appears:



In the Table Designer, for each field you add to the table, you enter the field number, name, data type and optionally, a length and a description.

### Adding Fields to Your Table

Designing a field means assigning a number of characteristics to it. These characteristics depend on what you intend to use the field for.



After you have added fields to a table in the Table Designer, you must save the table before you can add any records. Once you have saved a table, it appears in the list of tables in the Object Designer.

All the tables and fields you create have two forms of identification:

- A unique identification number (integer). When you access your database using either C/SIDE or C/FRONT, this number uniquely identifies all the tables and fields.
- A name (an alphanumeric string) that serves as a label (such as CUSTOMER or CITY). This name appears on the screen when you run the table and should be meaningful and easily understood. This name is secondary information and can be changed at any time.

## Choosing Data Types

When you have selected an identification number and name for a field, you have to select an appropriate data type. You can use many different types of fields in the C/SIDE database system. Each type is designed to hold a specific kind of information, such as text, numbers, dates and so on.

Fields in a record can be of the following types:

Data Type	Description	Size
Option	Denotes an integer in the range -2,147,483,647 and 2,147,483,647. An option field is defined with an option string, which is a comma-separated list of strings representing each valid value of the field. This string is used when a field of type Option is formatted and its value is converted into a string. An example: The Option field "Color" is defined with the option string "Red,Green,Blue". Valid values of the field are then 0, 1 and 2, with 0 representing "Red" and so on. When the "Color" field is formatted, 0 is converted into the string "Red", 1 into "Green", and 2 into "Blue". The size of the corresponding SQL data type, <code>INTEGER</code> , is 4 bytes. <sup>(A)(B)</sup>	4 bytes
Integer	Denotes an integer between -2,147,483,647 and 2,147,483,647. The size of the corresponding SQL data type, <code>INTEGER</code> , is 4 bytes. <sup>(A)(B)</sup>	4 bytes
Decimal	A decimal number between $-10^{63}$ and $10^{63}$ . The exponent ranges from -63 to +63. Decimal numbers are held in memory with 18 significant digits. The representation of a decimal number is a Binary Coded Decimal (BCD). The size of the corresponding SQL data type, <code>DECIMAL ( 38 , 20 )</code> , is 17 bytes. <sup>(A)(B)</sup>	12 bytes
Text	Any alphanumeric string. The field must be defined to be between 1 and 250 characters. The space used by a text field equals the maximum length of the text plus one byte. This extra byte is used to hold the length of the string. An empty text string has the length zero. The size of the corresponding SQL data type, <code>VARCHAR</code> , is 1 byte per character in the field's value. <sup>(A)(B)</sup>	Maximum string length + 1 byte (see note).

Data Type	Description	Size
Code	<p>An alphanumeric string, which is right-justified if the contents are numbers only. If letters or blanks occur among the numbers, the contents are left-justified. All letters are converted to uppercase upon entry.</p> <p>The field must be defined to be between 1 and 250 characters. The space used by a code field equals the maximum length of the text plus two bytes. The first of the extra bytes holds information about the length of the string, and the second byte stores alignment information.</p> <p>In the SQL Server Option for Dynamics NAV, code fields work in a different way. You can use the <i>SQL Data Type</i> property to indicate whether code fields can contain integers or text strings. Refer to the online <i>C/SIDE Reference Guide</i> for information about the <i>SQL Data Type</i> property. Further, Numbering in Dynamics NAV on page 515 contains information about the sorting of numeric values in code fields.</p> <p>The size of the corresponding SQL data type, <code>VARCHAR</code>, is 1 byte per character in the field's value.<sup>(A)(B)</sup></p>	Maximum string length + 2 bytes (see note).
Date	<p>A date value in the range from January 1, 0 to December 31, 9999. An undefined date is expressed as 0. All dates have a corresponding closing date. The system regards the closing date for a given date as a period that follows the given date but comes before the next normal date; that is, a closing date is sorted immediately after the corresponding normal date but before the next normal date.</p> <p>The size of the corresponding SQL data type, <code>DATETIME</code>, is 8 bytes.<sup>(A)(B)</sup></p>	4 bytes
Time	<p>Any time in the range 00:00:00 to 23:59:59.999. A time field contains 1 plus the number of milliseconds since 00:00:00 o'clock, or 0 (zero), an undefined time. A time value is calculated in the following way:</p> <p>Time = 1 + (number of milliseconds since 00:00:00).</p> <p>The size of the corresponding SQL data type, <code>DATETIME</code>, is 8 bytes.<sup>(A)(B)</sup></p>	A time field is stored as an integer (four bytes).
Boolean	<p>Assumes the values TRUE or FALSE. When formatted, a boolean field is shown as "Yes" or "No".</p> <p>The size of the corresponding SQL data type, <code>TINYINT</code>, is 1 byte.<sup>(A)(B)</sup></p>	4 bytes
Binary	<p>Contains binary data. The binary data is stored in the record. The size of the corresponding SQL data type, <code>VARBINARY</code>, is the number of bytes in the field's value.<sup>(A)(B)</sup></p>	Maximum length is 250 bytes (see note).
BLOB	<p>Binary Large Object. Used to store bitmaps and memos. Notice that the BLOB isn't stored in the record, but in the BLOB area of the table.</p> <p>The size of the corresponding SQL data type, <code>IMAGE</code>, is the number of bytes in the field's value.<sup>(A)(B)</sup></p>	8 bytes in the record + size of BLOB data. (max. 2 GB)

Data Type	Description	Size
DateFormula	Used to verify the date entered by the user. The syntax is for example: 30D (=30 days) CM+1M (=current month plus one month) D15 (=on the 15th of each month)	4 bytes
TableFilter	This data type is used to apply a filter to another table. Currently, this can only be used to apply security filters from the Permission table.	
BigIntInteger	A 64 bit integer.	8 bytes
Duration	Represents the difference between two points in time, in milliseconds. This value can be negative.	8 bytes
DateTime	Represents a point in time as a combined date and time. The datetime is stored in the database as Coordinated Universal Time (UTC) and is always displayed as local time in Dynamics NAV. Local time is determined by the time zone regional settings used by your computer. You must always enter datetimes as local time. When you enter a datetime as local time, it is converted to UTC using the current settings for the time zone and daylight saving time. The DateTime datatype does not support closing dates.	Stored as two 4 byte integers
GUID	Globally unique identifier	16 bytes
RecordID	Unique record identifier	

(A) The calculation of the size of a specific SQL Server record requires more than simply summing the sizes of the field values. Refer to Microsoft's SQL Server documentation for further information.

(B) This is the SQL Server data type that Dynamics NAV uses when it creates the Dynamics NAV data type. for further information, see page 78.

#### Note

.....

In C/SIDE Database Server, data is stored with a four byte alignment because of performance considerations. The sizes of text, code and binary fields (that can have variable lengths) are rounded up to the nearest value that is a multiple of four. This means that, for example, a text string of 10 characters will occupy 12 bytes.

.....

Besides the ordinary fields discussed in this section, the C/SIDE database system also includes two special types of fields:

- FlowField®
- FlowFilter®

How these special fields provide powerful data retrieval mechanisms is described on page 87.

## 6.2 Viewing and Modifying Properties

This section describes how you can use properties in your table design. As you learned earlier, there are three kinds of properties:

- Table Properties
- Field Properties
- Key Properties

### Viewing and Modifying Table Properties

A table in C/SIDE has a number of properties that determine the behavior of the table. When you create a table, C/SIDE automatically defines a number of default values for these properties. Depending on what the table is going to be used for and how it is related to other application objects, you may want to change these default values.

C/SIDE contains the following table properties:

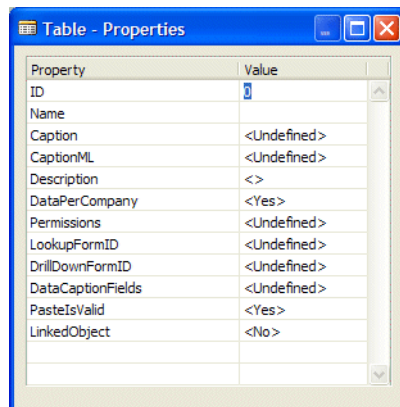
Property Name	Use this property to...
ID	define the ID of the table.
Name	define a name (used as caption) for the table.
Caption	display the caption in the currently selected language. The value is taken from the <i>CaptionML</i> property if this property is set. A caption is the text the system uses to show the identity of a control (for example, in the caption bar of a form or as the basis for a label for another control).
CaptionML	provide the text that will be used to identify a control or other object in the user interface. CaptionML is multilanguage enabled. This means that it can contain a list of texts in different languages. The text that is actually used will be selected according to the current language setting of the user.
Description	include an optional description of the table. This description is for internal purposes only and is not visible to the end user. A short description of the table's purpose makes it easier to maintain the application.
DataPerCompany	determine whether the system will create a version of the data for each company in the database.
Permissions	define extended permissions for the table.
LookupFormID	define the ID of the form you want to use as a lookup.
DrillDownFormID	define the ID of the form you want to use as a drill down.
DataCaptionFields	define a list of fields to be used as captions when a record from this table is displayed in, for example, a form.
PastelsValid	tell the system whether it should be allowed to insert records in this table by pasting.
LinkedObject	determine whether this Dynamics NAV table description is to be linked to an existing SQL Server object.

Property Name	Use this property to...
LinkedInTransaction	determine whether the linked object supports transactions and can be accessed within Dynamics NAV transactions or does not support transactions and is not under transaction control. This property is only available when the value of the <i>LinkedObject</i> property is set to <i>Yes</i> . For more information, see the section "Linked Objects" on page 105.

For more information about these properties, see the *C/SIDE Reference Guide* online Help.

To view or modify table properties:

- 1 Click Tools, Object Designer (SHIFT+F12) and in the **Object Designer** window, click Table to see a list of the tables.
- 2 Select a table and click Design. C/SIDE displays the table in the Table Designer.
- 3 Place the cursor on an empty line in the Table Designer (press F3 to create an empty line) or click Edit, Select Object. (If you place the cursor on a line defining one of the existing fields in the table, you see the properties for this field instead of those for the table.)
- 4 Click View, Properties (SHIFT+F4). C/SIDE displays the **Properties** window:



- 5 If you want to modify the setting of a property, simply enter the new value in the **Properties** window. When you have entered the new value, update the property by either pressing ENTER or simply moving the cursor away from the field.
- 6 To get Help for a property, select it in the **Properties** window and press F1.

#### Example

*LookupFormID* is a typical example of a property you will want to modify. The default value for the *LookupFormID* property is *<Undefined>*. By changing this value, you can determine which form the system will display when F6 (Lookup) is pressed. Look at the *LookupFormID* property of the **Customer** table (18). This property tells C/SIDE which form to use to lookup values in the **Customer** table. The value of the property is the **Customer List** form.

## Viewing and Modifying Field Properties

Just like tables, all the fields in C/SIDE have a number of properties that determine their behavior. When you create a field, C/SIDE automatically suggests a number of default values for these properties. Depending on the purpose of the field, you will sometimes want to change these default values.

C/SIDE contains the following field properties:

Property Name	Use this property to...
Field No.	assign a unique numeric ID to this field.
Name	specify the name of the field.
Caption	specify the text the system uses to identify a control based on the field.
CaptionML	specify the text that is used to identify a control or other object in the user interface. CaptionML is multilanguage enabled and can contain a list of texts in different languages.
CaptionClass	enable a field in a database table or a control to use caption classes.
Description	include an optional description of the field. This description is for internal purposes only and is not visible to the end user.
Data Type	specify the data type of a table field.
Enabled	determine whether the field is enabled.
Data Length	specify the maximum length of the data stored in this field.
InitValue	define an initial value for a field.
FieldClass	define the class for a field (that is, specify whether it is a normal field, a FlowField or a FlowFilter field).
CalcFormula	define a formula used by a FlowField.
AltSearchField	define an alternative search field.
DecimalPlaces	set the number of decimal places shown to the user. This property also performs validation of whether user input conforms to this setting.
Editable	determine whether a field can be edited.
NotBlank	force the user to make a non-blank entry in this field.
BlankNumbers	tell the system to blank a range of numbers as it formats them.
Numeric	force the user to enter numbers in this field.
CharAllowed	set the characters you will allow the user to enter in this field.
DateFormula	validate the syntax of a date expression entered by the user.
Standard day/time unit	specify the unit of measure that is used when you enter data into Duration fields.
MinValue	set the minimum value for the contents of a field.
MaxValue	set the maximum value for the contents of a field.
Title	add a title to a field. The first letter in each word is capitalized.
ValuesAllowed	specify the values you want to allow in the field. Can be specified either as a range or as distinct values, or as a combination of these.



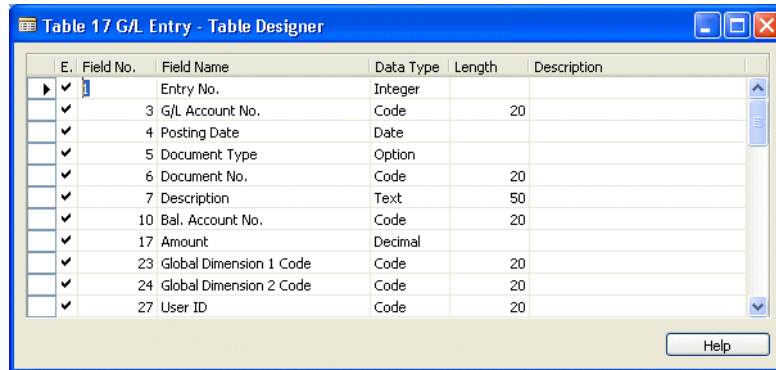
Property Name	Use this property to...
AutoIncrement	specify whether or not each field value is automatically given a new number that is greater than the number given to the previous value.
TableRelation	define relationships to other tables. See the section "Setting Relationships Between Tables" on page 98 for a detailed description of how to create table relations.
ValidateTableRelation	tell the system whether or not it should validate a table relationship.
TestTableRelation	tell the system whether or not you want it to include this field when it tests the table relationships.
TableIDExpr	specify the ID of the table to which you want to apply a table filter.
BlankZero	define that the field will appear blank if the value is 0 (zero) or FALSE.
DataLength	define the length of a data field.
OptionString	define an option string (a comma-separated string of options). The maximum size is 1000 characters.
ClosingDates	determine whether closing dates are allowed.
AutoFormatType	determine how data is formatted.
AutoFormatExpr	determine how data is formatted.
SignDisplacement	shift negative values to the right for display purposes.
SQLDataType	specify the data type you want to allow in a code field. This property applies to code fields in the SQL Server Option for Dynamics NAV.
ClearOnLookup	tell the system to delete the current contents of the field before it adds the value the user selects via the lookup.
SubType	define the subtype of a BLOB field, for example a Bitmap or a Memo.
Compressed	specify whether or not a BLOB is compressed. This property only applies to BLOB fields and only on the SQL Server Option.
OptionCaption	define the text string options that will be displayed to the user.
OptionCaptionML	set the strings that will be displayed to the user for selecting an option. <i>OptionCaptionML</i> is only used if the field has an <i>OptionString</i> property. The <i>OptionString</i> property contains the set of values that are acceptable choices, and it is one of these values that will be saved in the database or used in C/AL code.

For more information about these properties, see the *C/SIDE Reference Guide* online Help.

To view or modify field properties:

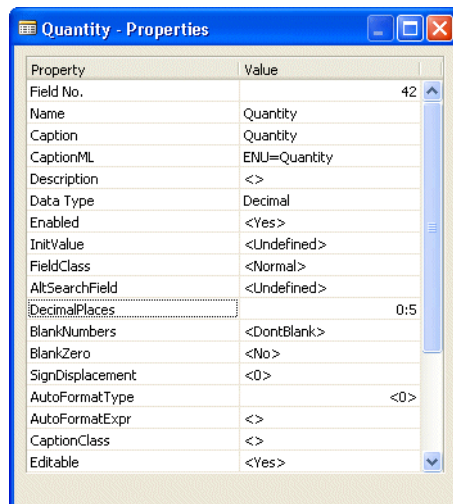
- 1 Click Tools, Object Designer (SHIFT+F12).
- 2 In the **Object Designer** window, click Table to see a list of the tables.

3 Select a table and click Design to display it in the Table Designer:



4 Place the cursor on the line in the Table Designer that defines the field whose properties you want to access.

5 Click View, Properties (SHIFT+F4) to open the **Properties** window:



6 If you want to modify a property, simply enter the new value in the **Properties** window. When you have entered the new value, update the property by either pressing ENTER or simply moving the cursor away from the field.

7 To get Help for a property, select it in the **Properties** window and press F1.

### Example

The *DecimalPlaces* property is a typical example of a field property you may want to change. When you create a new field of type Decimal, C/SIDE will assume that you want the value to be formatted as a currency. If your decimal field will not contain a currency, you can use this property to determine the number of decimal places that will appear on the screen. For example, in the G/L Entry table, the *DecimalPlaces* property of the Quantity field (Field No. 42) has been set to 0:5. This means that the minimum number of decimal places you can enter is 0 and the maximum is 5.

## 6.3 Defining Keys

The Database Management System (DBMS) keeps track of each field by means of the field number, and the record's primary key. The primary key is composed of up to 20 fields in a record. The combination of values in fields in the primary key makes it possible for the DBMS to perform a unique identification of each record. The primary key determines the logical order in which records are stored, regardless of their physical placement on disk.

Logically, records are stored sequentially in ascending order, sorted by the primary key. Before adding a new record to a table, the DBMS checks that the information in the record's primary key fields is unique, and only then inserts the record into the correct logical position. Records are sorted "on the fly," so the database is always structurally correct. This allows fast data manipulation and retrieval.

A table description contains a list of keys. A key is a sequence of one or more field IDs from the table. Up to 40 keys can be associated to a table. The first key in the list is the primary key.

The primary key is always active and the DBMS keeps the table sorted in primary key order and rejects records with duplicate values in primary key fields. Therefore, the values in the primary key must always be unique. Be aware that it is not the value in each field in the primary key that must be unique, but rather the combination of values in all the fields comprising the primary key.

The C/SIDE database system does not support tables that do not have any keys.

### How to Define a Primary Key

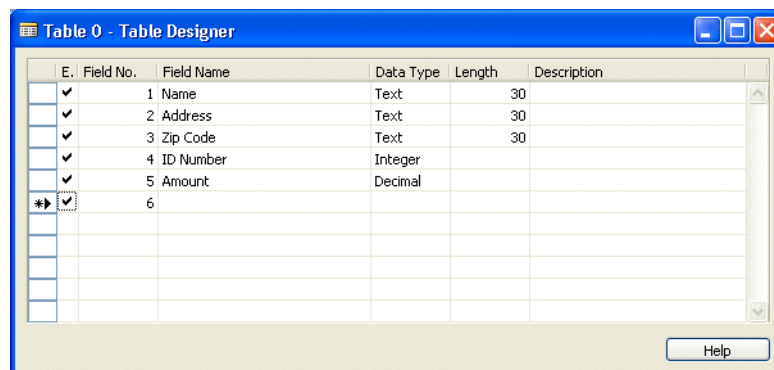
A maximum of 20 distinct fields can be used in a primary key definition. The number of fields in the primary key limits the number of fields in the other (secondary) keys.

When you create a table in the table designer, C/SIDE automatically uses the field with the lowest field number as the primary key.

To define a primary key:

To define a primary key:

- 1 Assume that you have created the following table in the Table Designer:

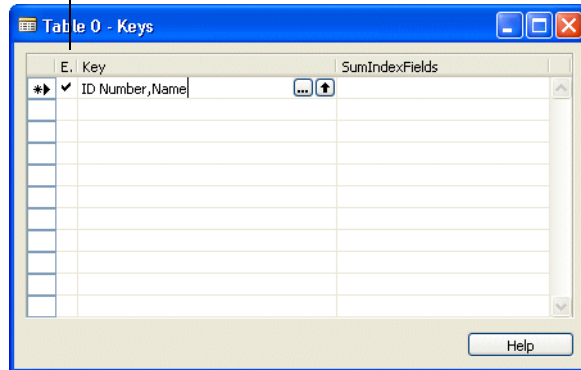


The screenshot shows a window titled "Table 0 - Table Designer". It contains a table with the following columns: E., Field No., Field Name, Data Type, Length, and Description. The table has 6 rows of data. The first five rows have checkmarks in the 'E.' column, and the sixth row has a key symbol (a vertical line with a horizontal bar at the top) in the 'E.' column, indicating it is the primary key.

E.	Field No.	Field Name	Data Type	Length	Description
✓	1	Name	Text	30	
✓	2	Address	Text	30	
✓	3	Zip Code	Text	30	
✓	4	ID Number	Integer		
✓	5	Amount	Decimal		
*	6				

- 2 Click View, Keys to define a primary key. C/SIDE displays the **Keys** window:

Define the primary key here



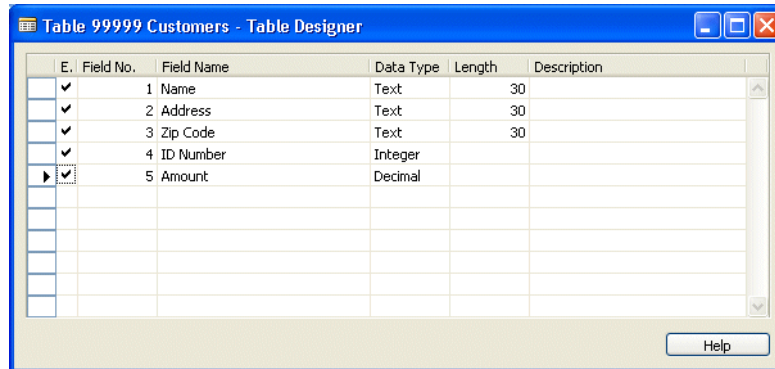
- 3 In the first line in the Keys window, enter the primary key as a comma-separated list (for example: ID Number, Name).

### How to Create Secondary Keys

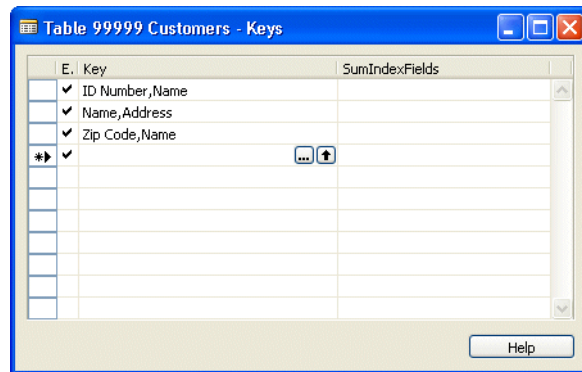
Remember that up to 40 keys can be defined for a table, and that the first is the primary key. All the other keys are secondary keys and are optional. Secondary keys are used to view records in an order that is different from the order defined by the primary key fields.

To create a secondary key:

- 1 Click Tools, Object Designer (SHIFT+F12) and open your table in the Table Designer:



2 Click View, Keys and C/SIDE will display the **Keys** window for the table:



3 The first line shows the primary key. Enter the secondary keys on the following lines as comma-separated lists (for example: Name,Address).

#### Note

The number of fields in the primary key together with all the fields in secondary keys must not exceed 20.

This means that if your primary key includes four distinct fields, your secondary keys can include these four fields, and at most 16 other fields. Correspondingly, if your primary key consists of 20 distinct fields, then your secondary keys must consist only of combinations of these fields.

A secondary key uses an additional structure called an index. This is similar to the idea of an index used in textbooks. A textbook index alphabetically lists important terms at the end of a book. Next to each term are the page numbers where it appears. The index can be quickly searched to find a list of page numbers (addresses), and the term easily located by searching the specified pages. The index is an exact indicator that shows where each term occurs in the textbook.

When you define a secondary key and mark it as active, the system will automatically maintain an index reflecting the sorting order defined by the key. Several secondary keys may be active at the same time.

A secondary key can be changed into an inactive key (which doesn't occupy database space). This means that the DBMS does not use time during updates to maintain its index. Inactive keys can be reactivated, although this may be time-consuming because the DBMS has to scan the entire table to rebuild the index.

The fields comprising the secondary keys are not guaranteed to contain unique data, and the DBMS does not reject records with duplicate data in secondary key fields. If two or more records contain identical information in the secondary key, the DBMS will use the primary key for the table to resolve this conflict.

#### Sort Orders and Secondary Keys

The following example shows how the primary key influences the sorting order when a secondary key has been activated.

Assume that the **Customer** table includes four entries (records). The records in the **Customer** table have two fields: **Customer No.** and **Customer Name**.

The Key List for the **Customer** table is:

Key No.	Key Type	Definition
1	Primary	Customer No.
2	Secondary	Customer Name

**Customer** table sorted by the primary key:

Customer No.	Customer Name
001	Microsoft® Business Solutions
002	IBM
003	Lotus
004	Microsoft Business Solutions

If you select the secondary key for sorting, the ordering is based on the contents of the **Customer Name** field. As the contents of these fields are not unique, the records have to be sub-sorted according to the primary key.

Customer Name	Customer No.
IBM	002
Lotus	003
Microsoft Business Solutions	001
Microsoft Business Solutions	004

In this case the last two records, which have the same Customer Name, have been ordered by Customer No.

## How Keys Affect Performance

Searching for specific data is normally easier if several keys have been defined and maintained for the table holding the desired data. The indexes for each of the keys provide specific views that enable quick flexible searches. There are, however, both advantages and drawbacks to using a large number of keys. Consider the following situations:

If you...	Performance improves...	Performance slows...
increase the number of secondary keys marked as active.	when you retrieve data in several different sorting sequences because the system has already sorted the data.	when you enter data because C/SIDE has to maintain the indexes for each secondary key.

If you...	Performance improves...	Performance slows...
decide to use only a few keys.	when you enter data because C/SIDE has a minimal number of indexes to maintain.	when you retrieve data. You may have to define or reactivate the secondary keys to get the appropriate sortings. Depending on the size of the database, this can take some time, as the system builds the index.

The decision whether to use few or many keys is not easy. The choice of appropriate keys and the number of active keys to use should be the best compromise between maximizing the speed of data retrieval and maximizing the speed of data updates (operations that insert, delete or modify data). In general, it may be worthwhile to deactivate complex keys if they are rarely used.

The overall speed of C/SIDE depends on a number of factors:

- The size of your database
- The number of active keys
- The complexity of the keys
- The number of records in your tables
- The speed of your computer and its disk system

### How Keys Are Stored

As illustrated in the figure on page 60, keys are stored in the Table Description, which contains a list of keys. The next figure illustrates part of the key list for a **Cust. Ledger Entry** table.

Key Description	1 (Entry No.)				Primary Key
	3 (Customer No.)	4 (Posting Date)			Secondary Key
	5 (Document Type)	6 (Document No.)	3 (Customer No.)		Secondary Key
	3 (Customer No.)	36 (Open)	43 (Positive)	37 (Due Date)	Secondary Key

The figure shows the first four keys of this table; the primary key and three secondary keys. The primary key consists of a single field ID. The first secondary key contains two field IDs, while the second and third secondary keys contain three and four fields, respectively.

### Viewing and Modifying Key Properties

The keys associated with a table have properties that describe their behavior, just as tables and fields do. When you create a key, C/SIDE automatically suggests a number of default values for these properties. Depending on the purpose of the key, you will sometimes want to change these default values.

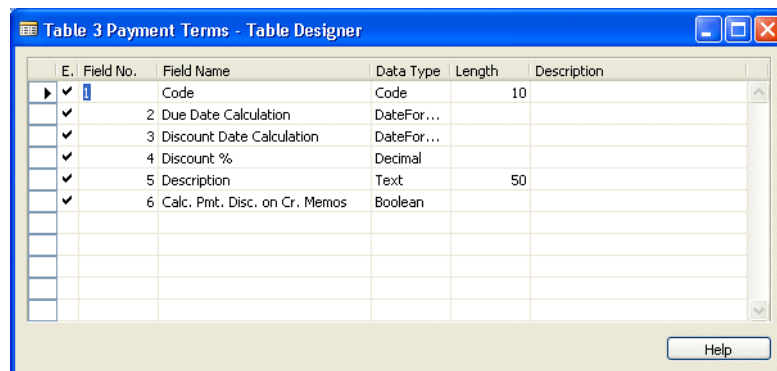
C/SIDE contains the following properties for keys:

Property Name	Use this property to...
Enabled	determine whether the system will maintain an index for the key. You cannot use a key unless it is enabled.
Key	define the key.
SumIndexFields	determine the fields for which the system will maintain a SumIndex®.
KeyGroups	determine which key groups the key is a member of. By making the key a member of a predefined key group you can have the key defined and only enable it when it is going to be used.
BackupKey	see whether any errors occurred the last time you restored a backup.
MaintainSQLIndex	determine whether or not a SQL Server index corresponding to the Dynamics NAV key should be created.
MaintainSIFTIndex	determine whether or not SIFT structures should be created in SQL Server to support the corresponding SumIndexFields for the Dynamics NAV key.
SIFTLevelsToMaintain	specify which SIFT levels are maintained for a key.

Refer to the online *C/SIDE Reference Guide* for additional information about these properties.

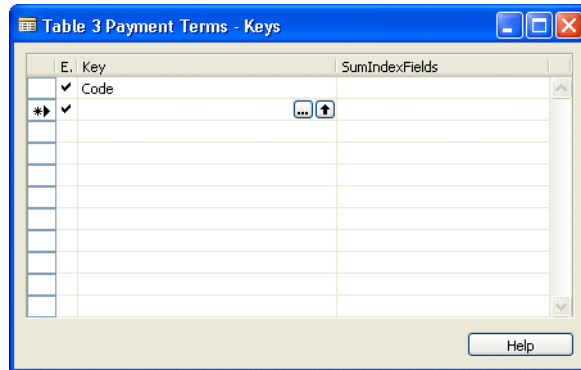
To view or modify properties for the keys of a particular table:

- 1 Click Tools, Object Designer, Table to see a list of the tables.
- 2 Select a table and click the Design button. C/SIDE displays the table in the Table Designer:



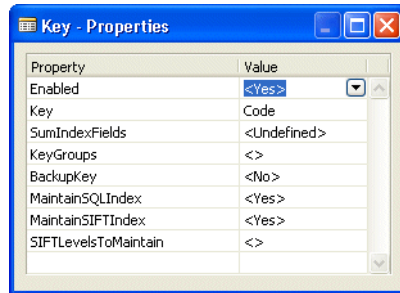


3 Click View, Keys and C/SIDE displays:



4 Place the cursor on the line defining the key for which you want to view or modify the properties.

5 Click View, Properties and C/SIDE displays the **Properties** window:



6 If you want to modify the setting of a property, simply enter the new value in the **Properties** window. When you have entered the new value, update the property by either pressing ENTER or simply moving the cursor away from the field.

7 To get Help for a property, select it in the **Properties** window and press F1.

## 6.4 Identifiers, Data Types and Data Formats in the SQL Server Option for Dynamics NAV

This section describes the identifiers, data types and data formats that are used in the SQL Server Option.

### Naming Identifiers

Identifiers for SQL Server tables and columns are based upon the table names and field names for the corresponding tables and fields of a Dynamics NAV table definition. If you set a table's *DataPerCompany* property to *Yes*, the SQL Server table name is prefixed by the company name. The two names are separated by the (\$) symbol. For example, the SQL Server table name for the **Customer** table of the CRONUS International Ltd. company is *CRONUS International Ltd\_ \$Customer*. If the *DataPerCompany* property of a table is set to *No*, there is no prefix.

The primary key of a Dynamics NAV table is created in a SQL Server table as a primary key constraint. The name of the primary key will be based on the table name with a suffix of \$0, for example, *CRONUS International Lt\_ \$Customer\$0*. Any secondary keys in a Dynamics NAV table that must be created and maintained in SQL Server – the *MaintainSQLIndex* key property is set to *Yes* – will have SQL Server indexes created that are named after an internal key ID with a \$ prefix. Examples of this are \$1 and \$4.

If the database maintains SQL views for language IDs, the system creates a SQL view by prefixing the SQL Server table name with the Windows language ID. For example, if you want to refer to the **Customer** table in the CRONUS International Ltd. company in German (Standard), the SQL view is *DEU\$CRONUS International Ltd\_ \$Customer*. For more information about multilanguage functionality, see Chapter 18.

If the database maintains relationships, the system creates foreign key constraints using the SQL Server table name and *TableRelation* property information. The names of the constraints have the following format: <table name>\$FK\$T<referencing table ID>\_F<referencing field ID>\$T<referenced table ID>. Here is an example using the **Customer** table: *CRONUS International Ltd\_ \$Customer\$FK\$T18\_F107\$T308*.

When you create a Dynamics NAV table with keys that contain *SumIndexFields*<sup>®</sup>, this causes additional tables to be created in SQL Server to support the SIFT™ functionality. These tables are named after the company, the table ID and an internal key ID. For example, the SIFT table name for *SumIndexFields* of the key (G/L AccountNo., Posting Date) in the **G/L Entry** table in CRONUS International Ltd. is *CRONUS International Ltd\_ \$17\$0*.

#### Important

If you create a Dynamics NAV table with keys that contain *SumIndexFields*, you must not give the table the same name as its ID. SIFT tables whose names are the same as their ID cannot be saved. If you try to do so, you will receive an error message.

### Representation of Dynamics NAV Data Types

Every available Dynamics NAV data type is mapped to an appropriate SQL Server data type in the tables of the SQL Server Option for Dynamics NAV. The following table

shows which SQL Server data type is used for the corresponding Dynamics NAV data type:

Dynamics NAV Data Type	SQL Server Data Type
Integer	INTEGER
Option	INTEGER
Code(n)	VARCHAR ( n ) INTEGER SQL_VARIANT
Text(n)	VARCHAR ( n )
Decimal	DECIMAL ( 38 , 20 )
Date	DATETIME
Time	DATETIME
DateTime	DATETIME
Boolean	TINYINT
Binary(n)	VARBINARY ( n )
BLOB	IMAGE
DateFormula	VARCHAR ( 32 )
TableFilter	VARBINARY ( 252 )
BigInteger	BIGINT
Duration	BIGINT
GUID	UNIQUEIDENTIFIER
RecordID	VARBINARY ( n )

Each of the SQL Server data types is created as NOT NULL except the IMAGE type, which allows NULL.

### Compatibility of Data Types

Some of the SQL Server data types listed previously are compatible with other Dynamics NAV data types. The following table shows the extended compatibility of SQL Server data types with Dynamics NAV data types:

SQL Server Data Type	Dynamics NAV Data Type
CHAR ( n )	Code(n) Text(n) DateFormula
NCHAR ( n )	Text(n)
NVARCHAR ( n )	Text(n)

SQL Server Data Type	Dynamics NAV Data Type
INTEGER	Code
TINYINT	Integer Option
SMALLINT	Integer Option
NUMERIC(p,s), MONEY, SMALLMONEY, REAL, FLOAT(n), DECIMAL	Decimal Integer Option Boolean
SMALLDATETIME	Date
BIT	Integer Option Boolean
BINARY(n)	Binary(n)
TEXT	BLOB
NTEXT	BLOB
UNIQUEIDENTIFIER	Binary(16) Text(36)

### Data Format Considerations

When you are using the SQL Server Option for Dynamics NAV, you must be aware of the effect the data formats will have on the way your data is compared and sorted.

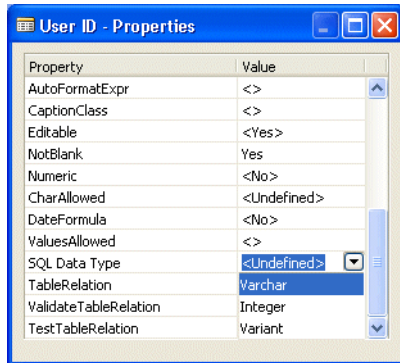
### Code Fields

In the SQL Server Option for Dynamics NAV, code fields can be represented by several SQL Server data types.

Code fields have a property, *SQL Data Type*, that determines whether they contain integers, text strings or a mixture of both. You set this property in the following way:

- 1 Click Tools, Object Designer.
- 2 Click Table and select the appropriate table.
- 3 Click Design.

- 4 Select the field whose data type is defined as code and then click View, Properties. The **Properties** window for that field appears:



You can set the *SQL Data Type* property to *Varchar*, *Integer* or *Variant*. Leaving the value as *Undefined* is the same as selecting *Varchar*, which is the default value.

When you create a table in the SQL Server Option for Dynamics NAV, the code field data is stored in `VARCHAR`, `INTEGER` or `SQL_VARIANT` columns in the SQL Server table that correspond to the *SQL Data Type* property's values *Varchar*, *Integer* or *Variant*.

When you set the value of the *SQL Data Type* property of a code field to *Varchar*, all the values in the field are compared and sorted as character data, including numeric values.

When you set the value of the *SQL Data Type* property of a code field to *Integer*:

- All the values in the field are compared and sorted as integers. No alphanumeric values can be stored in the field.
- If you enter negative values in the column outside Dynamics NAV using external tools, they cannot be read into Dynamics NAV.
- The value "0"(zero) is used to represent an empty string in Dynamics NAV.
- Non-numeric code values or any numeric values beginning with "0"(zero) cannot be entered in the code field.

When you set the value of the *SQL Data Type* property of a code field to *Variant*:

- The values in the field are compared and sorted according to their base data type. Numeric values are sorted after alphanumeric values.
- Data that is entered into the code field in Dynamics NAV is stored as either the `VARCHAR` or `INTEGER` base data type, depending on the value that has been entered.
- Any value beginning with "0"(zero) can be entered in the code field and is stored as an `INTEGER` base data type.

**Note**

Be aware that not all the third-party tools that can be used to access data in SQL Server databases support the *Variant* data type.

## Date and Time Fields

SQL Server stores information about both date and time in columns of the `DATETIME` and `SMALLDATETIME` types. For date fields, Dynamics NAV uses only the date part and places a constant value for the time. For a normal date, this contains 00:00:00:000. For a closing date, it contains 23:59:59:000 for a `DATETIME` and 23:59:00:000 for a `SMALLDATETIME`.

The Dynamics NAV undefined date is represented by the earliest valid date in SQL Server: 01-01-1753 00:00:00:000 for a `DATETIME`, and 01-01-1900 00:00:00:000 for a `SMALLDATETIME`.

For time fields, only a SQL Server `DATETIME` type can be used. Dynamics NAV uses only the time part and places a constant value for the date: 01-01-1754. The Dynamics NAV undefined time is represented by the same value as an undefined date.

In order for Dynamics NAV to interpret date and time values correctly, the formats mentioned earlier must be used when linking Dynamics NAV table definitions to external tables or views. For more information about this, see page 105.

To reformat a `DATETIME` or `SMALLDATETIME` column that is to be used as a date field in Dynamics NAV, an `UPDATE` statement can be applied to the table data. Here is an example of such an update statement:

```
UPDATE [My Table] SET [My Date] = CONVERT(CHAR(10), [My Date], 102)
```

For a closing date, a `CONVERT` style of 120 can be used to set the appropriate time part. To reformat a time field, a similar statement can be used:

```
UPDATE [My Table] SET [My Time] = CAST('1754-01-01 '+CONVERT(CHAR(8), [My Time], 108) AS DATETIME)
```

As an alternative to modifying the table data, you can create a view that applies the necessary conversion to the column and gives the column an alias. However, you cannot update views that are created in this way and it is more efficient to change the data than to apply conversions for every row.

### Note

.....  
 The information in this section only applies to fields of the *Date* and *Time* data type and does not apply to fields of the *DateTime* data type.  
 .....

## Accessing Dynamics NAV Tables with External Tools

You can access data in Dynamics NAV tables with external tools, such as Microsoft Enterprise Manager. When you do this, the values in fields that contain the code, date and time data types and which have a specific format must be manipulated correctly for data modification or comparison. When you use external tools, no special processing of code field data is required to join fields in different tables provided that you use the same SQL data type value for each code field in a join or `CAST` the value to the appropriate data type.

Multilanguage views In the **New Database** and **Alter Database** windows, you can select to maintain SQL views. If you enable this option, SQL Server will create and maintain a view for each language ID that is added to a table in Dynamics NAV. The system creates a SQL view

by prefixing the SQL Server table name with the Windows language ID for each CaptionML value.

This means that external tools can use a view of the object in the user's language, for example Spanish, rather than the object name. The object name could be in an other language, for example English (United States).

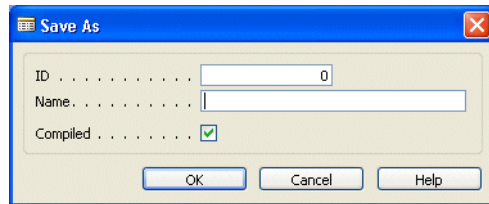
The view is updated by every change in the CaptionML values of a table. For more information, see "Multilanguage Functionality" on page 465.

## 6.5 Saving tables and Viewing Sorted Data

When you have designed the fields and keys for a new table, you have to save the table in the database before you can use it. Once you have saved a table, it appears in the list of tables shown in the Object Designer.

To save a table in the database:

- 1 With the table that you want to save open, make sure that the focus is on the Table Designer and click File, Save. C/SIDE displays:



- 2 In the **ID** field, enter a number that will serve as a unique table identification. There are restrictions about which numbers you can use. Contact your Microsoft Certified Business Solutions Partner for information.

### Viewing data

Normally, you use a form to view the data in a table, but you can also view the data directly by running the table from the Object Designer.

To view the data in a table without using a form:

- 1 Open the Object Designer (SHIFT+F12) and select the table to view.
- 2 Click Run and C/SIDE displays the data in a tabular format:

Entry No.	G/L Acco...	Posting D...	D.	Documen...	Description	Bal. Acco...	Amount
1	1110	C31-12-99		START	Opening Entry		1,324.95
2	1140	C31-12-99		START	Opening Entry		-340.5E
3	1210	C31-12-99		START	Opening Entry		582.87
4	1240	C31-12-99		START	Opening Entry		-362.2E
5	1110	C31-12-99		START	Opening Entry		154.4E
6	1140	C31-12-99		START	Opening Entry		-62.92
7	1310	C31-12-99		START	Opening Entry		49.47
8	1340	C31-12-99		START	Opening Entry		-24.8C
9	2180	C31-12-99		START	Opening Entry		269.94
10	2180	C31-12-99		START	Opening Entry		135.24

### Sorting order

The order in which the information appears in the window is determined by the sorting order defined by the current key. If more than one key is defined for the table, you can switch between the sorting orders that these keys define.



Changing the sort order

To view the table data in different sorting orders:

- 1 Open the Object Designer (SHIFT+F12), select the table that you are working on and click Run to open it.

Entry No.	G/L Account	Posting Date	Document	Description	Balance Account	Amount
1	1110	C31-12-99	START	Opening Entry		1,324.95
2	1140	C31-12-99	START	Opening Entry		-340.5E
3	1210	C31-12-99	START	Opening Entry		582.87
4	1240	C31-12-99	START	Opening Entry		-362.2E
5	1110	C31-12-99	START	Opening Entry		154.4E
6	1140	C31-12-99	START	Opening Entry		-62.92
7	1310	C31-12-99	START	Opening Entry		49.47
8	1340	C31-12-99	START	Opening Entry		-24.8C
9	2180	C31-12-99	START	Opening Entry		269.94
10	2180	C31-12-99	START	Opening Entry		135.24

- 2 Click View, Sort (SHIFT+F8) and C/SIDE displays:

Select the key here

Select ascending or descending order here

- 3 Select the key that defines the sorting order you want, and choose whether you want the records displayed in ascending or descending order. Click OK or Apply to apply the new key. If you choose OK, the Sort dialog closes; if you choose Apply, the Sort dialog stays open. Using Apply is convenient if you frequently change the sorting order.

Adding records without using forms

Data is normally entered in a table by using a form, but you can also enter it directly.

To add records to a table without using a form:

- 1 Open the Object Designer (SHIFT+F12), select the table and click Run. C/SIDE displays the table in a tabular format:

Entry No.	G/L Acco...	Posting D...	D...	Documen...	Description	Bal. Acco...	Amount
2794	5796	01-01-01	I...	103040	Invoice SCI0000019		
2795	5611	01-01-01	I...	103040	Invoice SCI0000019		
2796	2310	01-01-01	I...	103040	Invoice SCI0000019		
2797	5796	01-12-00	I...	103041	Invoice SCI0000020		
2798	5611	01-12-00	I...	103041	Invoice SCI0000020		
2799	2310	01-12-00	I...	103041	Invoice SCI0000020		
2800	5796	01-01-01	I...	103042	Invoice SCI0000021		
2801	5611	01-01-01	I...	103042	Invoice SCI0000021		
2802	2310	01-01-01	I...	103042	Invoice SCI0000021		

- 2 Place the cursor in an empty line or press F3 to create an empty line. Enter data in the fields and press ENTER. You can use TAB, SHIFT-TAB and the arrow keys on the keyboard to navigate between fields.
- 3 When you have finished entering data, close the table. (You do not have to save it, because records are saved and updated whenever you leave a field after entering a value in it.)

## 6.6 Special Table Fields

In addition to the conventional data fields which simply hold values, three kinds of specialized fields are provided for data retrieval:

- SumIndexFields
- FlowFields
- FlowFilter fields

### What Are SumIndexFields?

A SumIndexField is a decimal field that can be attached to a key definition. This is the fundamental feature of the Dynamics NAV database that forms the basis for FlowFields. SumIndexFields permit fast calculation of sums of numeric columns in tables, even in tables with thousands of records. This occurs because SumIndexFields are maintained when the database record is updated.

### What Advantages Do SumIndexFields Offer?

SumIndexFields enable the speedy calculation of sums of columns. The resulting totals are displayed in FlowFields.

For example, assume you want the sum of all the values in the Amount field. In a conventional database system, the DBMS is forced to access every record and add each value in the field Amount. This is a very time-consuming operation in a database with thousands of records. With Dynamics NAV, it can take as little as two accesses (if the best key is used) to sum the Amount for these records.

This special index structure, a SumIndexField, is associated with a key. Each key can have at most 20 SumIndexFields.

During database design, a decimal field can be associated with a key as a SumIndexField. This tells the DBMS to create and maintain a structure that contains the accumulated sum of values in a column. When a new current key is selected, any SumIndexFields associated with it become accessible.

### What Are FlowFields?

FlowFields are a powerful feature of the C/SIDE database system, and strongly influence the way C/SIDE applications are designed. FlowFields and the underlying concept of SumIndexFields increase performance in such activities as calculating the balance of your customers. In traditional database systems, this involves a series of accesses and calculations before a result is available. Why such a result will be immediately available when you use FlowFields will be clear as you read through the rest of this section.

FlowFields are not a permanent part of the table data. A FlowField can be thought of as a virtual field, which is an extension to the table data. Because the information in FlowFields exists only at run time, values in FlowFields are automatically initialized to 0 (zero). To update a FlowField, use the C/AL function `<Record>.CALCFIELDS`. If a FlowField is the direct source expression of a control on a form, the FlowField will automatically be calculated when the form is displayed.

### FlowField Types

There are seven types of FlowFields:

FlowField Type	Field Type	Description
Sum	decimal	The sum of a specified set within a column in a table
Average	decimal	The average value of a specified set within a column in a table
Exist	boolean	Indicates whether any records exist within a specified set in a table
Count	integer	The number of records within a specified set in a table
Min	any	The minimum value in a column within a specified set in a table
Max	any	The maximum value in a column within a specified set in a table
Lookup	any	Looks up a value in a column in another table

### Example

Consider the **Customer** table in the following figure. This table contains two FlowFields. The field named **Any Entries** is a FlowField of the Exist type, and the **Balance** field is a FlowField of the Sum type.

#### Customer (Table data)

Customer	Name	Country/Region Code	Balance (FlowField)	Any Entries (FlowField)
10000	Windy City Solutions	US	60	Yes
10010	Modern Cars Inc.	US	90	Yes
10020	Jean Saint Laurent	FR	210	Yes
10030	Russel Publishing	UK	0	No
10040	La Cuisine Française	FR	300	Yes

#### Customer Entry (Table data)

Customer	Date	Comment	Amount
10000			10
10000			20
10000			30
10010			40
10010			50
10020			60
10020			70
10020			80
10040			90
10040			100
10040			110

Virtual part of the table data

The figure shows that the value in the **Balance** FlowField for customer number 10000 (Windy City Solutions), is retrieved from the **Amount** column in the **Customer Entry** table. The value is the sum of the amount fields for the entries that have the customer number 10000, that is

$$\text{Sum} = 10 + 20 + 30 = 60.$$

The values shown in the **Balance** column in the **Customer** table for customers number 10010, 10020, 10040 are found in the same way. For customer number 10030 the value is 0 (zero), as there are no entries in the **Customer Entry** table that have a Customer No. that equals 10030.

In this example the **Balance** FlowField in the **Customer** table reflects the sum of a specific subset of the **Amount** fields in the **Customer Entry** table. How the calculation of a FlowField is to be made, is defined in a calculation formula. The calculation formula for the **Balance** field is

```
Sum("Customer Entries".Amount WHERE(CustNo=FIELD(CustNo)))
```

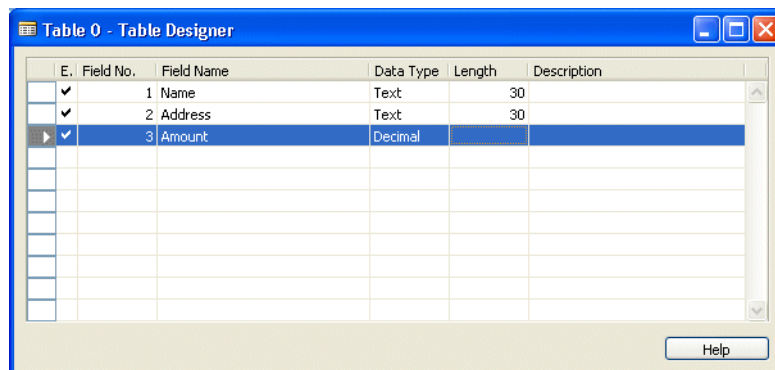
Correspondingly, the **Any Entries** field, which indicates whether any entries exist, has the following definition:

```
Exist("Customer Entries" WHERE(CustNo=FIELD(CustNo)))
```

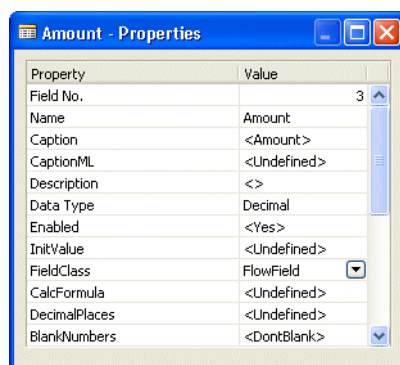
## Creating a FlowField

To create a FlowField:

- 1 Open the Object Designer (SHIFT+F12) and open the table that you want to add a FlowField to in the Table Designer. C/SIDE typically displays:



- 2 Click on the line defining the field that you want to define as a FlowField – in this case the **Amount** field.
- 3 Click View, Properties (SHIFT+F4) and C/SIDE displays the **Properties** window of the **Amount** field:



- 4 Change the value of the *FieldClass* property from *Normal* to *FlowField*.
- 5 Enter a calculation formula for the *FlowField*. This is done with the *CalcFormula* property. The next section tells you how.

### Calculation Formulas and the *CalcFormula* Property

A *FlowField* is always associated with a calculation formula that determines how the *FlowField* is calculated. The valid syntax for the *CalcFormula* property is:

```
<CalculationFormula> ::=
    [-]Exist(<TableNo> [WHERE (<TableFilters>)]) |
    Count(<TableNo> [WHERE (<TableFilters>)]) |
    [-]Sum(<TableNo>.<FieldNo> [WHERE (<TableFilters>)]) |
    [-]Average(<TableNo>.<FieldNo> [WHERE (<TableFilters>)]) |
    Min(<TableNo>.<FieldNo> [WHERE (<TableFilters>)]) |
    Max(<TableNo>.<FieldNo> [WHERE (<TableFilters>)]) |
    Lookup(<TableNo>.<FieldNo> [WHERE (<TableFilters>)])

<TableFilters> ::=
    [<TableFilter> {,<TableFilter>}]

<TableFilter> ::=
    <DstFieldNo>=CONST(<FieldConst>) |
    <DstFieldNo>=FILTER(<Filter>) |
    <DstFieldNo>=FIELD(<SrcFieldNo>) |
    <DstFieldNo>=FIELD(UPPERLIMIT(<SrcFieldNo>)) |
    <DstFieldNo>=FIELD(FILTER(<SrcFieldNo>)) |
    <DstFieldNo>=FIELD(UPPERLIMIT(FILTER(<SrcFieldNo>)))
```

where..

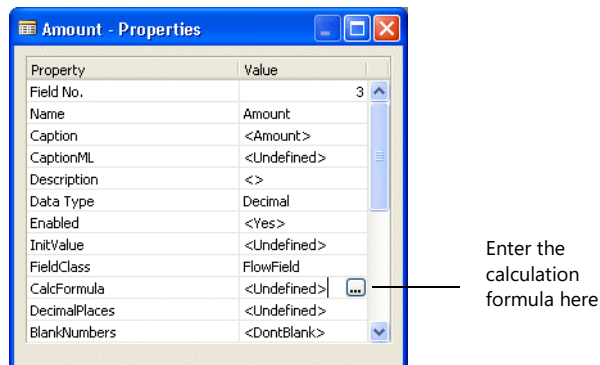
Symbol	Explanation
<TableNo>	Specifies the table holding the information to be used in the <i>FlowField</i> .
<FieldNo>	Specifies the column from which you want to compute values.
<TableFilters>	A list of filters to be used in the computation of the <i>FlowField</i> .
<TableFilter>	A table filter can be one of the following: a constant expression, a filter expression, a value from ordinary fields or a <i>FlowFilter</i> field ( <i>FlowFilter</i> fields are discussed in the next section). Notice that a key for the other table must exist and include the fields used in the filters.
<DstFieldNo>	Specifies the destination field number.
<SrcFieldNo>	Specifies the source field number.
<Filter>	A filter expression such as 10 20..30.

### Creating, Viewing and Editing a Calculation Formula

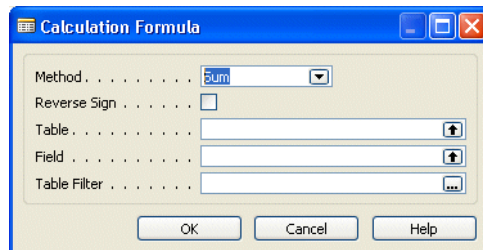
To create, view, or edit a calculation formula:

- 1 Open the Object Designer (SHIFT+F12) and open the table in question.
- 2 Select the field for which you want to create, view, or edit the calculation formula.

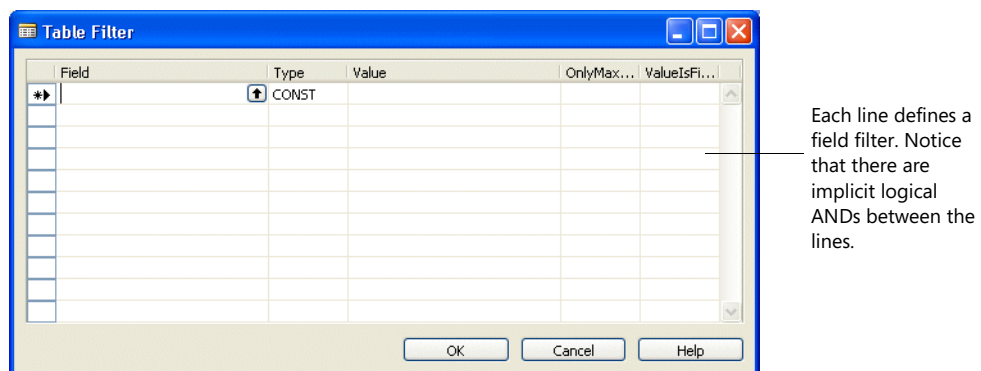
- 3 Click View, Properties (SHIFT+F4) to open the **Properties** window.
- 4 Find the *CalcFormula* property:



- 5 You can either enter the calculation formula directly or click the assist-edit button. When you click the assist-edit button, C/SIDE displays:



- 6 Click the drop-down button to select the appropriate calculation method. Click the Reverse Sign option to reverse the sign of the result (only for Sum and Average). Use the lookup buttons to select the table and column (field) from which to get the information. If necessary, you can add a table filter to specify a limited set of records. Click the assist-edit button to the right of the Table Filter field and C/SIDE opens the **Table Filter** window:



- 7 On each line in this window, you can define a field filter. For each field filter, specify a field, a type, and a value. You can also set the OnlyMaxLimit and the ValueIsFilter options. The following example illustrates where the information in this window comes from.

**Example**

The **Balance at Date** field in the **G/L Account** table is a decimal type FlowField. This field is calculated from values in the **Amount** column in the **G/L Entry** table.

The **Amount** field contains the information to be summed. This field is defined as a SumIndexField in the key for the **G/L Entry** Table.

The **Field** column in the Table Filter window contains references to fields (columns) in the **G/L Entry** table.

The **Value** column in the Table Filter window contains references to fields (columns) in the **G/L Account** table.

Entry No.	G/L Account No.	Posting Date	Document No.	Description	Balance Account No.	Amount
1	1110	C31-12-99	START	Opening Entry		1,324.95
2	1140	C31-12-99	START	Opening Entry		-340.5E
3	1210	C31-12-99	START	Opening Entry		582.87
4	1240	C31-12-99	START	Opening Entry		-362.2E
5	1110	C31-12-99	START	Opening Entry		154.4E
6	1140	C31-12-99	START	Opening Entry		-62.92
7	1310	C31-12-99	START	Opening Entry		49.47
8	1340	C31-12-99	START	Opening Entry		-24.8C
9	2180	C31-12-99	START	Opening Entry		269.94
10	2180	C31-12-99	START	Opening Entry		135.24

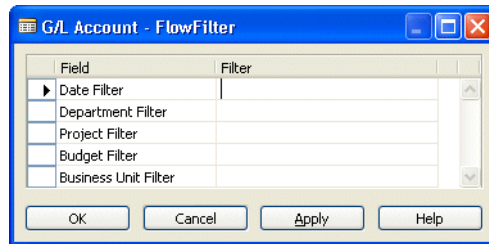
Field	Type	Value	OnlyMax...	ValuesFi...
G/L Account No.	FIELD	No.		
G/L Account No.	FIELD	Totalling		✓
Business Unit Code	FIELD	Business Unit Filter		
Global Dimension 1 Code	FIELD	Global Dimension 1 Filter		
Global Dimension 2 Code	FIELD	Global Dimension 2 Filter		
Posting Date	FIELD	Date Filter		✓

No.	Name	Search No.	A.	Departm...	Project C...	I...	D...	No. 2	Blk
1000	BALANCE SHEET	BALANCE...	H..				B.. B..		
1002	ASSETS	ASSETS	B..				B.. B..		
1003	Fixed Assets	FIXED AS...	B..				B.. B..		
1005	Tangible Fixed Assets	TANGIBL...	B..				B.. B..		
1100	Land and Buildings	LAND AN...	B..				B.. B..		
1110	Land and Buildings	LAND AN...	P..				B.. B..		
1120	Increases during the Year	INCREAS...	P..				B.. B..		
1130	Decreases during the Year	DECREA...	P..				B.. B..		
1140	Accum. Depreciation, Buildings	ACCUM. ...	P..				B.. B..		
1190	Land and Buildings, Total	LAND AN...	E..				B.. B..		

Some of the fields in the **G/L Account** table are FlowFilter fields. By entering filter expressions into these fields, the user can affect the calculation of FlowFields (such as **Balance at Date**) at run time.



The user can enter filter values in a FlowFilter form:



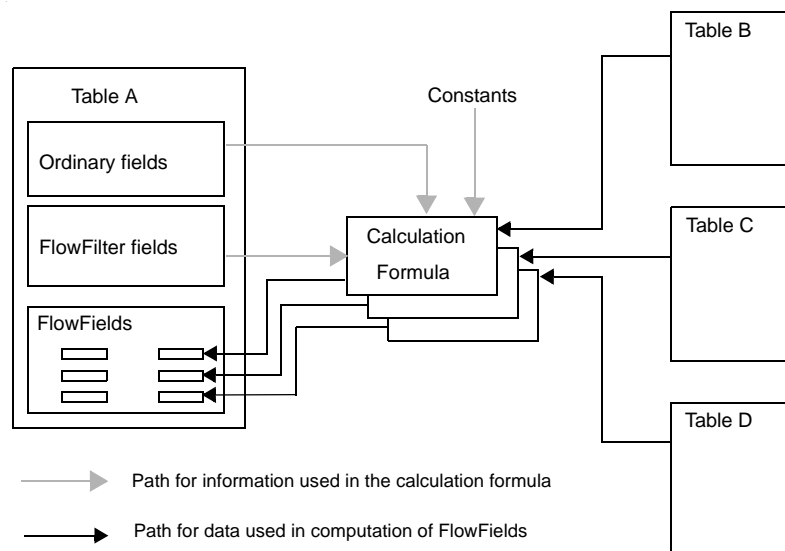
If the user enters a date filter expression in the **Date Filter** field, it is transferred via the table filter and used in the **Date** column in the **G/L Entry** table.

You can use the OnlyMaxLimit option to remove the lower limit from a range defined by a filter expression. For example, if the filter expression is defined as a range x..y, setting the OnlyMaxLimit option transforms the expression into ..y.

The ValuelsFilter option determines how the system interprets the contents of the field referred to in the Value column in the table filter window. For example, if the field contains the value 1000..2000, setting the ValuelsFilter option will cause this value to be interpreted as a filter rather than as a specific value.

### Using FlowFilter Fields in the Calculation Formula

Users may want to limit calculations so that they include only those values in a column that have some specific properties. For example, the user may want to sum only the amounts of customer entries that are entered in April. This is possible if the application has been designed using FlowFilter fields in connection with the FlowFields.



The figure illustrates the relations between various types of database fields and the calculation formula. The filters defined in the calculation formula can consist of constants, values from ordinary fields and of filters given as parameters in FlowFilter fields. In FlowFilter fields, users can enter a filter value (via the user interface in a C/SIDE application) that will affect the calculation of a FlowField.

## 6.7 Dividing the Database into Companies

The DBMS can access only one logical database at a time, but this database can be divided into one or more companies. A company is a "sub-database," and its primary use is to separate and group data in the same database. Fields and tables are identified by a number, but companies are identified by a name. A company "bundles" one or more data tables together into a logical superstructure that is identified by the company name. Other than the shared company name, the different tables within a company have nothing in common.

Opening a company is your first step after opening the database or connecting to a server. How to do this is described in the *Installation and System Management* manual for the server that you are using

Consider a database with four tables as shown in this figure:

Table Description	Company A	Company B	Company C
G/L Account	Data	Data	Data
Customer	Data	Data	Data
Vendor	Data	Data	Data
Report Menu Option	Common Data		

The four table descriptions on the left apply to each of the data tables, which are logically sorted into three companies. The records in the tables **G/L Account**, **Customer** and **Vendor**, all have the same structure and the same field definitions, even though they belong logically in three different companies. Only the data stored in the fields differ. As the information in a Table Description can be used by tables from more than one company, no redundant information will be stored. This minimizes the size of the database.

Even though you have selected a specific company, you can still access data in any table in any other company. Use the C/AL function `<Record>.CHANGECOMPANY` to explicitly define which other company you want to access.

More than one application can access the same company and the same table(s) at the same time. How the DBMS controls these multiple accesses is described in the section *What is Table Locking?* on page 524.

## Chapter 7

### Customizing and Maintaining Tables

As you create tables, you'll want to take advantage of properties and triggers. By setting properties for your tables, you can set up defaults to use throughout your database, and by defining C/AL code in triggers, you can modify the system's default behavior.

This chapter shows you how to use properties and triggers when you design tables. Furthermore, it shows how to create relationships between tables. Finally, the chapter explains how to deal with the problems you may encounter when you change tables that contain data.

- Using Table and Field Triggers
- Setting Relationships Between Tables
- Changing Tables That Contain Data
- Linked Objects

## 7.1 Using Table and Field Triggers

C/SIDE recognizes certain things that happen to a table when you use it, for example, that you insert or modify data. In response, you can get the system to execute C/AL code defined in a trigger. Triggers are predefined functions that are executed when certain things happen. The bodies of these functions are initially empty and must be defined by the developer. Defining C/AL code in triggers allows you to change the default behavior of the system.

The triggers in a C/SIDE table can be divided into two categories:

- Table triggers
- Field triggers

Tables in C/SIDE have the following triggers:

Table Trigger Name	Executed when...
<b>OnInsert</b>	a new record is inserted into the table.
<b>OnModify</b>	a record in the table is modified.
<b>OnDelete</b>	a record in the table is deleted.
<b>OnRename</b>	a record is modified in a field that is part of the primary key.

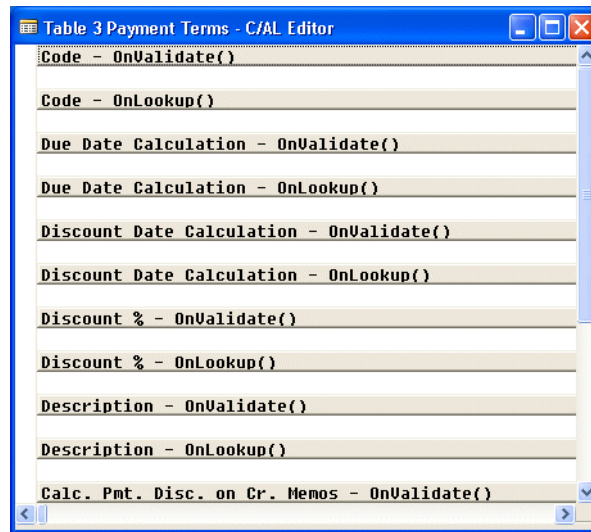
Fields in tables have these triggers:

Field Trigger Name	Executed when...
<b>OnValidate</b>	data is entered in a field or when <Record>.VALIDATE is executed in C/AL code.
<b>OnLookup</b>	Lookup (F6) is activated.

If you are not familiar with C/AL programming, please refer to the chapter "Introducing the C/AL Language" on page 295.

To define or modify a trigger for a table or a field:

- 1 Open the Object Designer (SHIFT+F12) and click Table to see a list of the tables.
- 2 Select the table and click Design. The system will open the Table Designer, containing a list of the fields in the table.
- 3 Click View, C/AL Code (F9). C/SIDE to see the code for the table in the Table Designer.
- 4 The system uses the position of the cursor in the Table Designer to determine what code to display. If you place the cursor on a specific field in the Table Designer, the code in the C/AL Editor is automatically scrolled so that the first trigger related to that field appears at the top of the window. If the cursor is placed on an empty line in the Table Designer, the system shows the first trigger related to the table itself. The position of the cursor in the Table Designer does not restrict your access to other triggers. You can always scroll up and down through the triggers in the C/AL editor.



5 You can now enter or modify the C/AL code in the relevant trigger(s).

## 7.2 Setting Relationships Between Tables

As mentioned in the section "Introduction to C/SIDE Application Design" on page 52, it is common to distinguish among three types of relationships between tables in relational database design:

- One-to-Many Relationships
- Many-to-Many Relationships
- One-to-One Relationships

The one-to-many relationship is the most common. If your database design model indicates that you need to set up a many-to-many relationship, your design is probably inefficient. You normally break down a many-to-many relationship into two one-to-many relationships. A one-to-one relationship is usually undesirable and can often be avoided by simply combining the two tables. To learn more about database design, refer to one of the "Recommended Books on Database Design" on page 56.

### Why Use Relationships?

If your database contains tables with related data you can define a relationship between them. You relate tables by specifying one or more fields that contain the same value in related records. These matching fields often have the same name in each table. You can use relationships to:

- validate data entries.
- perform Lookup in other tables.
- automatically propagate changes from one table to other tables.

### Table Relations and the TableRelation Property

Table relations are defined using the *TableRelation* property. This property is very flexible and allows you to define both simple and advanced table relations. A typical simple table relation consists of just a table ID and an optional field ID. Advanced table relations are typically prefixed with a conditional statement and include filters. The syntax for table relations is:

```
<TableRelation> ::=
    <TableNo>[.<FieldNo>] [WHERE ( <TableFilters> )] |
    IF ( <Conditions> ) <TableNo>[.<FieldNo>]
    [WHERE( <TableFilters> )] ELSE <TableRelation>

<Conditions> ::=
    <TableFilters>

<TableFilters> ::=
    [<TableFilter> {,<TableFilter>}]

<TableFilter> ::=
    <DstFieldNo>=CONST(<FieldConst>) |
    <DstFieldNo>=FILTER(<Filter>)
```

where:

Symbol	Explanation
<TableNo>	Specifies the related table.
<FieldNo>	Specifies a field in the related table.
<Conditions>	Table relations can be conditional.
<TableFilters>	A list of table filters.
<TableFilter>	A table filter can be either a constant expression or a filter expression.
<DstFieldNo>	Specifies the destination field number.
<Filter>	A filter expression such as 10 20..30.

### Creating Basic Table Relations

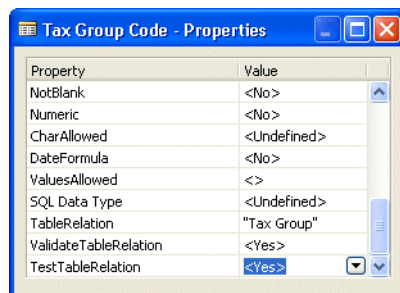
When you create table relations you can either enter them manually or use the assist-edit tool. You will usually enter basic table relations such as:

```
<TableNo>[.<FieldNo>]
```

directly on the **Properties** window, but use assist-edit to enter the more advanced table relations that use conditions and filters. Now you will see how to create (basic) table relations by entering them directly on the **Properties** window. In the next section, you will see how to use the assist-edit tool to do the same.

To create a basic table relation:

- 1 Open the Object Designer (SHIFT+F12) and click Table to see a list of the tables.
- 2 Select a table for which you want to create a relationship, and click Design. C/SIDE opens the table in the Table Designer.
- 3 Make sure that the cursor is placed in the field for which you want to set up a relation. Click View, Properties (SHIFT+F4) and C/SIDE will display the **Properties** window for the field:

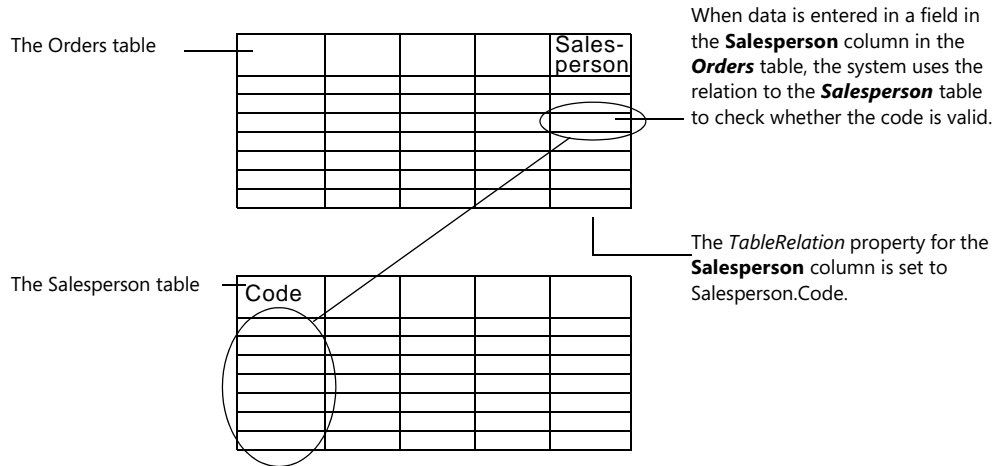


This picture shows the properties of the **Tax Group Code** field of the **Tax Detail** table (322).

- 4 Enter the table relation directly in the **Value** field for the *TableRelation* property. Simple table relations use the syntax: <TableNo>.[<FieldNo>]. Refer to the next section to learn how to use the assist-edit tool to create advanced table relations.

**Example**

Assume that you have an **Orders** table that stores orders and a **Salesperson** table that stores the names of all salespeople in your company. In the **Orders** table, you can include a field called Salesperson that identifies the salesperson. By setting up a relationship between these two tables you can get the system to check whether the **Salesperson** field in the **Orders** table contains a valid code.



**Example**

Assume that you have a **Vendors** table with all your vendors and a **Currency Code** table. You can create a relationship between a **Currency Code** field in the **Vendors** table and the **Currency Code** table. This will allow users to lookup (F6) information about valid currency codes.

Furthermore, if you change one of the currency codes in the **Currency Code** table, the system will automatically propagate this change to all the tables that refer to this code.

**Creating Table Relations with the Assist-Edit Tool**

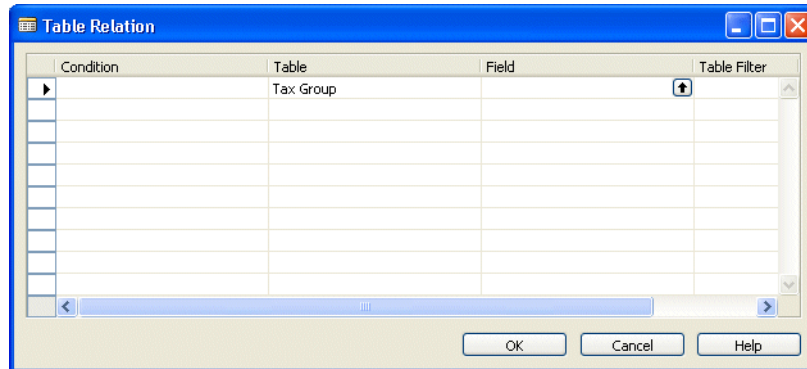
C/SIDE has an assist-edit tool that makes it easier for you to enter advanced table relations. An advanced table relation is prefixed with a conditional statement and uses filters.

To create a table relation using assist-edit:

- 1 Open the Object Designer (SHIFT+F12) and click Table to see a list of the tables.
- 2 Select a table for which you want to create a relationship, and click Design. C/SIDE opens the table in the Table Designer.
- 3 Make sure that the cursor is placed in the field for which you want to set up a relation. Click View, Properties (SHIFT+F4) and C/SIDE display the **Properties** window of that field.



- Click the assist-edit button in the **Value** field of the *TableRelation* property. C/SIDE opens the **Table Relation** window:



This picture shows the **Table Relation** window of the **Tax Group Code** field of the **Tax Detail** table (322).

- Use the assist-edit to fill in the **Condition** fields with the relevant table filters. For example, you can look up into different tables, based on the value in an option field.
- In the **Table** field, enter the name of the table to which you want to make a relation, or use the lookup button to select the table from a list. In the **Field** field, you can enter the name of the field or use the lookup button to select it from a list of the fields in the table you have entered in the **Table** field.

If necessary, define a table filter (for the table in the **Table** field) in the **Table Filter** field.

### Maintaining Table Relationships on SQL Server

The *TableRelation* property in Dynamics NAV can be represented in SQL Server by table relationships known as foreign key constraints. These table relationships are metadata about the tables and are only intended for modelling and diagramming and are not used to enforce data integrity. The foreign key constraints are disabled.

The table relationships in SQL Server can be accessed with external tools that can use this information to generate diagrams illustrating the structure of the database.

You can use the **Maintain Relationships** option on the **Integration** tab of the **New Database** and the **Alter Database** windows to enable and disable the table relationships on SQL Server. For more information about setting this option, see the manual *Installation & System Management: SQL Server Option for Microsoft Dynamics NAV*.

## Requirements

There are certain requirements that must be met before a *TableRelation* property can be represented on SQL Server.

To maintain a table relationship:

- The fields forming the relationship must be of the same data type in both of the related tables. This also applies to any fields that are specified in the **Table Filter** field. Text and code fields are compatible as long as they have the same length.
- The *SQL Data Type* property of code fields must be the same in both tables.
- The table filter that is part of the table relationship must contain only the *FIELD* filter type. Table filters of the *CONST* and *FILTER* filter type cannot be created on SQL Server.
- Conditional relationships have one SQL Server relationship for each condition, as long as all of the criteria listed here are met by each condition.

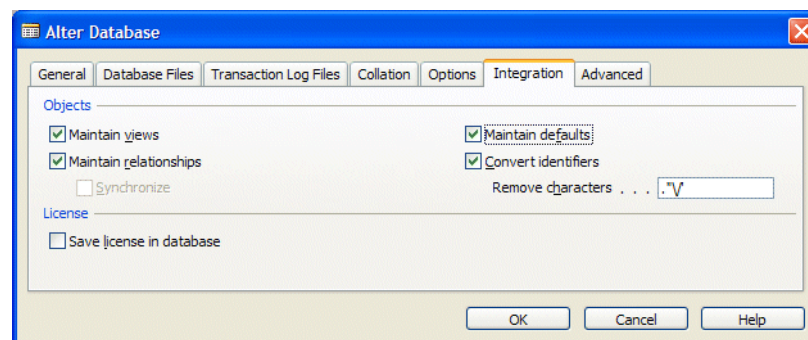
## Synchronization

The *TableRelation* properties and SQL Server relationships are automatically synchronized when you create a table and when you redesign a table. However, there are some situations in which you might need to manually synchronize the relationships. For example, after you have:

- deleted a table in the Object Designer.
- restored a database backup.
- imported a .fob file.

To manually initiate the synchronization process:

- 1 Click File, Database, Alter, and the **Alter Database** window appears.
- 2 Click the **Integration** tab.



- 3 Enter a check mark in the **Synchronize** check box and click OK.

This check box is only enabled when there are table relationships that need to be synchronized because of inconsistencies in the *TableRelation* properties.

If an error occurs during the synchronization process, you will receive an error message informing you that a particular table has an invalid relationship. To correct this error, you must modify the *TableRelation* property of the table in question in the Table Designer and then manually synchronize the relationships again.

**Note**

.....  
Table relationships are not generated or maintained when you import a `.txt` file.  
.....

## 7.3 Changing Tables That Contain Data

When you design the tables in your database, you determine which fields they contain. But you will often need to modify the design of some of the tables after they have been in use for a while. Typically you will want to add or delete fields, or make changes to field names or data types.

**Note**

C/SIDE ensures that you never lose data when you modify the design of a table that contains data.

### Rules for Changing Tables

You can modify tables that don't contain data at any time. However, when the table contains data, a number of restrictions apply. The following table lists some general rules:

Modification	Rules
Changing a field name	You can always change the name of a field.
Changing a field number	You can always change the number of a field. Notice that this causes Dynamics NAV to run through all the application objects in order to update all the references to this field. This can be a time consuming process.
Changing a data type	You can change the data type for a field only if there is no data in this field for any of the records in the table. There is one exception to this rule: you can change the data type of a field from Code to Text even if the field contains data for some records.
Adding a field to a table	You can always add a field to a table.
Deleting a field	In order to delete a field, you must delete all data from the field in all records in the table. You must also remove all references to the field from other tables, forms and reports.
Changing the length of a String field	You can always increase the length of a String field. Whether you can decrease the length of a String field depends on the contents of all the values in the column in the table. The minimum length of a String field is determined by the longest string in the column.

## 7.4 Linked Objects

With Dynamics NAV, you can create a table definition for a SQL Server object (user table, system table or view) that already exists in the current database.

### Defining Linked Object Table Properties

You use the table property *LinkedObject* to link to SQL Server objects by changing the value to *Yes* when creating or modifying a table description in the table designer. When you change this value to *Yes*, the *LinkedInTransaction* property becomes available.

The *LinkedInTransaction* property must be set to *No* when the Dynamics NAV table description refers to a view that depends on objects that are outside the current database or on a linked server.

The *LinkedInTransaction* property allows you to read and modify data from linked server data sources, such as Excel, Access or another SQL Server. This access is not under Dynamics NAV transaction control. This means that if a Dynamics NAV transaction is aborted, any changes that were made during this transaction to a linked object that is outside the current database or on a linked server will remain in effect.

For information about linked sever data sources, see "Accessing Objects in Other Databases or on Linked Servers" on page 108.

#### Note

You cannot run tables with the *LinkedInTransaction* property set to *No* when the database has been set to single user mode.

### Creating a Dynamics NAV Table Description

The following descriptions illustrate the different kinds of Dynamics NAV table descriptions that you can create, depending on the *LinkedObject* and *LinkedInTransaction* table property values. You must be a member of the *db\_owner* fixed database role to create a table description.

To create a non-linked table:

- Set the value of the *LinkedObject* property to *No*.
- When you save this table, a SQL Server table that is owned by the *db\_owner* fixed database role is created with the name you have specified (including the company name, if necessary).
- If an object with this name already exists, an error message is displayed and the table is not saved.

To create a linked object that is under transaction control:

- Set the *LinkedObject* property to *Yes*.
- Set the *LinkedInTransaction* property to *Yes*.
- The table is saved without checking its validity. Dynamics NAV will check that the corresponding SQL object exists and that it is compatible with the Dynamics NAV table description when the table is accessed.

To create a linked object that is not under transaction control:

- Set the *LinkedObject* property to *Yes*.
- Set the *LinkedInTransaction* property to *No*.
- The table is saved without checking its validity. Dynamics NAV will check that the corresponding SQL object exists and that it is compatible with the Dynamics NAV table description when the table is accessed.

### Deleting a Dynamics NAV Table Description:

When the *LinkedObject* property is set to *No*:

- The SQL Server object is deleted if it is a user table.
- The SQL Server object is not deleted if it is a system table or a view. It can only be a system table or a view if it has been changed to one of these object types with the aid of an external tool. The *LinkedObject* property must be set to *Yes* in order to be able to link to a system table or a view.

When the *LinkedObject* property is set to *Yes*:

- The SQL Server object is not deleted.

This means that if you create a Dynamics NAV table with the *LinkedObject* property set to *No* and then change it to *Yes*, its corresponding SQL Server object is not deleted.

When you modify the *LinkedInTransaction* property of a Dynamics NAV table:

- All access to the linked SQL Server object will be made under or outside transaction control, depending on the setting you choose.

When you access data in a linked object:

- If the *LinkedInTransaction* property is set to *Yes*, all access to the linked object will be performed under transaction control – within Dynamics NAV transactions.
- If the *LinkedInTransaction* property is set to *No*, all access to the linked object will be performed outside transaction control – independent of Dynamics NAV transactions.

### Requirements for Linking Objects

When you are using a linked object, you should take the following into account:

- The name of the SQL Server object that includes any company prefix and (\$) separator must match exactly with the name of the Dynamics NAV table.
- As is the case when creating regular v tables, you must be a member of the *db\_owner* fixed database role in the current database.
- As is the case with regular Dynamics NAV tables, the object must exist in the current database and be owned by a user in the database who is a member of the *db\_owner* fixed database role. A SQL Server view can be used to access objects outside the current database (including those residing on separate servers) or owned by other users. For more information about accessing objects outside the current database, see page 108.
- Dynamics NAV will automatically grant the required SQL permissions on the object so that you can access it in the same way that regular Dynamics NAV tables are

accessed. It will then be subject to permissions assigned in the Dynamics NAV security system.

- The object being linked must have a SQL Server table definition that is compatible with the Dynamics NAV table definition.
- A view that cannot be updated in SQL Server (for example one containing computed/converted columns or unions) will also be read-only if it is used as a linked object from Dynamics NAV. With SQL Server 2000, you can write `INSTEAD-OF` triggers to define the logic that allows such a view to be updated. This logic is not part of Dynamics NAV.

### Rules Determining Compatibility

There are a number of rules that you need to keep in mind when you use linked objects:

- All columns in the object must be type compatible with those named in the Dynamics NAV table definition. It is not necessary to name all the columns in the Dynamics NAV table definition. For more information about type compatibility, see page 79.
- `SumIndexFields` cannot be defined for any object type.
- If the object is a user table, it must have a primary key constraint that contains the same number of columns as those listed in the Dynamics NAV primary key, and these columns must have the same names.
- If the object is a view or system table, a primary key must be defined, and any secondary keys may also be defined if required. These keys will only be used in Dynamics NAV. They will have no effect on a view, its underlying objects in SQL Server or on a system table. It is important that the data in the columns named in the primary key is unique. This will not be enforced as a physical constraint by the view or system table in SQL Server. However, Dynamics NAV will order the data as though a primary key was physically defined. Dynamics NAV relies on this uniqueness in order to correctly identify and order records.
- If the object is a view, it can have only one column of the SQL Server timestamp type, but it does not need to have any unless BLOB fields are present in the Dynamics NAV table definition. A timestamp column must exist in a user table.
- An `IDENTITY` column can be used in a user table or a view, and Dynamics NAV will ignore this column when inserting records into the table. This allows the `IDENTITY` column to be used as intended. Similarly, a computed column in a user table is also ignored. For a view, a column defined on a computed table column cannot be used if insert operations are required.
- You cannot link to a SQL Server temporary table.
- Multilanguage views are not created or maintained for linked objects. For more information about multilanguage views, see the section "Creating and Maintaining Databases" in the manual *Installation & System Management: SQL Server Option for Microsoft Dynamics NAV*.

Once an object has been linked, Dynamics NAV treats it like a regular table. However, depending on the object type, SQL Server may prevent certain operations from taking place. For example, a non-updateable view cannot be updated in Dynamics NAV, and a SQL Server error message appears if you attempt to do this. The ability to redesign the object from within Dynamics NAV is limited, and these restrictions are described in the next section.

## Redesigning the Dynamics NAV Linked Object Table Definition

A Dynamics NAV linked object table definition can be redesigned in accordance with the following rules:

- It cannot be renamed by changing the table definition name or the company name.
- No fields in the table definition can be renamed.
- New fields can be added providing they exist in the view, and existing fields can be deleted. In either case, the definition of the view in SQL Server is not changed.
- The primary and secondary key definitions can be changed. Also, new keys can be added, and existing keys can be deleted.
- The Dynamics NAV field data types can be modified provided that the new type remains compatible with the column type in the view.
- A linked user table can undergo any design changes that are applicable to a regular table that is created from within Dynamics NAV.
- If the *DataPerCompany* property of the Dynamics NAV table definition is changed, it will result in an attempt to link to a new object. This new object will be based on the new company name. The previously linked SQL Server object will no longer be linked by the table definition.
- The *LinkedObject* table property can only be changed from *Yes* to *No* for a user table.

## Accessing Objects in Other Databases or on Linked Servers

You can access objects outside the current database or server from Dynamics NAV by linking to an appropriately defined view in the current database. You can create a view definition outside of Dynamics NAV that accesses data on SQL Server linked servers, which can access heterogeneous data sources. This could, for example, involve performing a join of an Oracle table, a Microsoft Access table or a Microsoft Excel spreadsheet.

To access objects in other databases or on linked servers you must comply with the following rules:

- You must set the *LinkedInTransaction* table property to *No* in order to use a view referring to objects outside of the current database. The ability to modify data in these objects is dependent on the data providers that the objects refer to.
- You must be a member of the *db\_owner* fixed database role in the current database to access objects in other databases or on linked servers.
- All security permissions for objects in another database or on linked servers must be granted outside Dynamics NAV to the appropriate SQL Server logins.
- If a linked object refers to a view that accesses objects that are stored in another database on the same server, this view must be treated as though it were accessing a linked server.



## Chapter 8

### Special C/SIDE Tables

In addition to the normal database tables, C/SIDE has three other types of tables that serve special purposes in C/SIDE applications. These are called temporary, system and virtual tables. Temporary tables are used as a repository for temporary information at run time, while the two other types are system generated tables that provide various information about the current state of the system.

This chapter introduces you to the special C/SIDE tables and explains how to use them in your design.

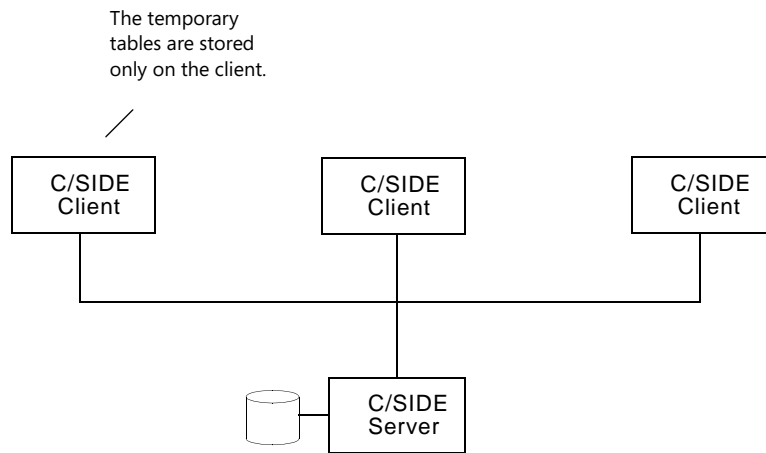
- What Is a Temporary Table?
- What Is a System Table?
- What Is a Virtual Table?

## 8.1 What Is a Temporary Table?

A temporary table can be regarded as a temporary variable that holds a table. A temporary table is used as a buffer for table data in your C/AL programs. If you are not familiar with C/AL, please refer to "Introducing the C/AL Language" on page 301.

You can do almost anything with a temporary table that you can do with a normal database table. The differences between a normal database table and a temporary table are:

- Temporary tables aren't stored in the database, but are only held in memory until the table is closed.
- The write transaction principle that applies to normal database tables does not apply to temporary tables. If you are not familiar with the transaction principle, see the section "Write Transactions and Recovery" on page 528.



The advantage of using a temporary table is that all the interaction with a temporary table takes place on the client. This reduces the load on both the network and the server.

When you need to perform many operations on the data in a specific table in the database, you can load the data into a temporary table while you modify it. Loading the data into a temporary table speeds up the process because all the operations are performed locally on the client.

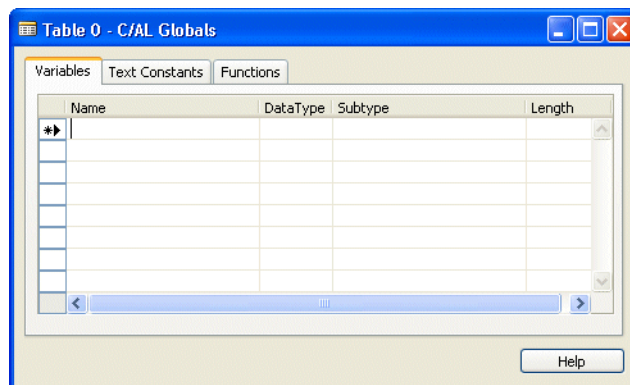
## Defining and Using a Temporary Table

You must define the temporary table before you can use it in your C/AL code. The variable that holds a temporary table is defined just like any other global or local variable.

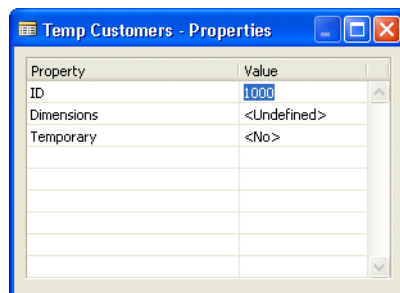
To define a temporary table:

- 1 Open the Object Designer (SHIFT+F12) and create a new table.
- 2 Click View, C/AL Globals or C/AL Locals, depending on whether your variable is going to be global or local.

If you choose C/AL Globals, the **C/AL Globals** window appears:



- 3 Enter a name for the temporary table variable, and enter or select Record as the data type. Use the lookup button in the Subtype field to select the table to copy.
- 4 With the cursor still on the line that defines the temporary table, click View, Properties (SHIFT+F4) to display the **Properties** window:



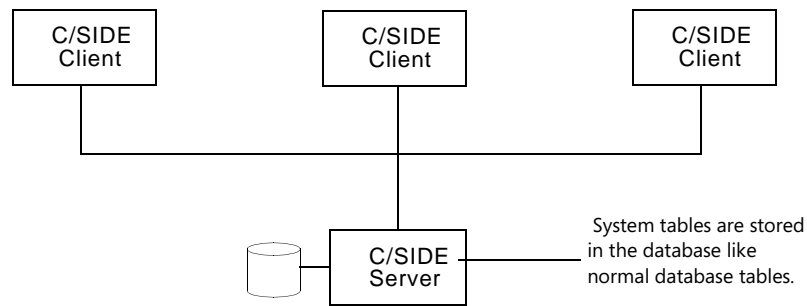
- 5 Change the value of the *Temporary* property to Yes.

After you have created a temporary table, you can use it in your C/AL code. You can apply filters and perform searches just as you can with normal database tables.

## 8.2 What Is a System Table?

System tables are stored in the database just like normal database tables. However, unlike normal database tables, they are created automatically by the system. The information in system tables is closely related to the DBMS, which uses the system tables to manage, for example, system security and permissions in C/SIDE.

You can read, write, modify and delete the information in system tables.



There are eight system tables in C/SIDE:

- **User**
- **Member Of**
- **User Role**
- **Permission**
- **Windows Access Control**
- **Windows Login**
- **Company**
- **Database Key Groups**

The first six tables in this list deal with system security.

### About permissions

.....

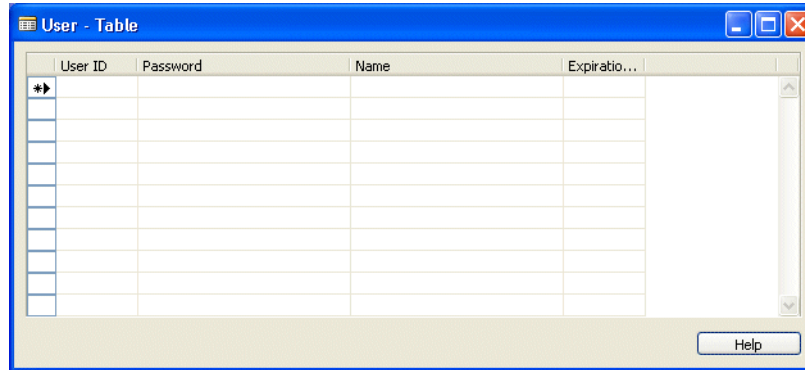
In order to insert, modify or delete information in the **User**, **Member Of**, **User Role**, **Permission**, **Windows Access Control** and **Windows Login** tables, you must have at least the same permissions as the users whose permissions you want to modify. This means that you can only assign or take away permissions that you yourself have.

.....

The following subsections provide an overview of these system tables. For further information about security in Dynamics NAV, see the *Installation & System Management* manual for the server option that you are using.

## The User System Table

The **User** system table gives you an overview of all the user IDs with database logins that you have created for users in your database. Each record in the **User** table defines a single user ID.



User ID	Password	Name	Expiration...
**▶			

The **User** table contains information about the password (displayed in encrypted form on screen), the real name of the user, and how long the user's ID is valid for each user ID defined in your database. You can create new user IDs by entering the appropriate data in this table. You can also remove a user ID by deleting the record from this table. (Of course, this depends on your own permissions.)

### Deleting a record

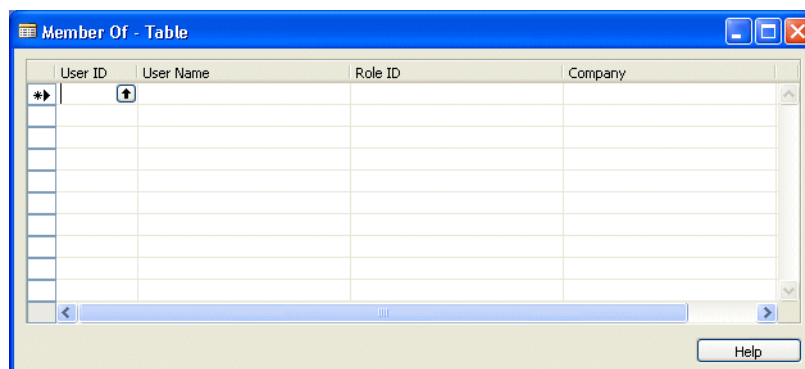
.....

If you delete a record in the **User** system table, the corresponding entries in the **Member Of** system table are automatically deleted.

.....

## The Member Of System Table

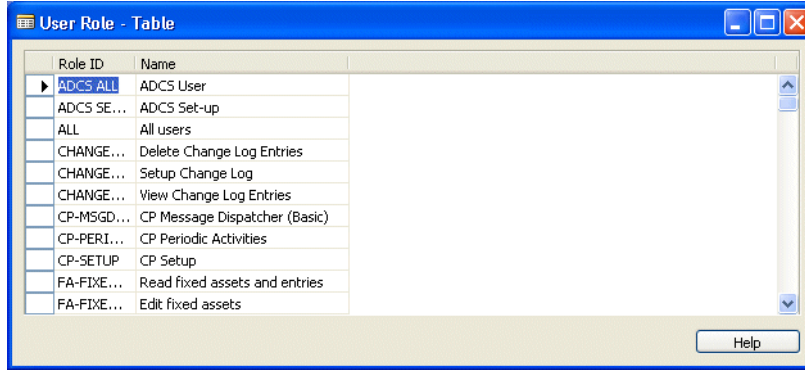
The **Member Of** system table gives you an overview of which user groups (roles) that each user is a member of. Each user (ID) can be a member of any number of user groups.



User ID	User Name	Role ID	Company
**▶	⬆		

### The User Role System Table

The **User Role** system table gives you an overview of the user roles in your database. A user role specifies a set of permissions. The exact permissions for each user role are defined in the **Permissions** system table.



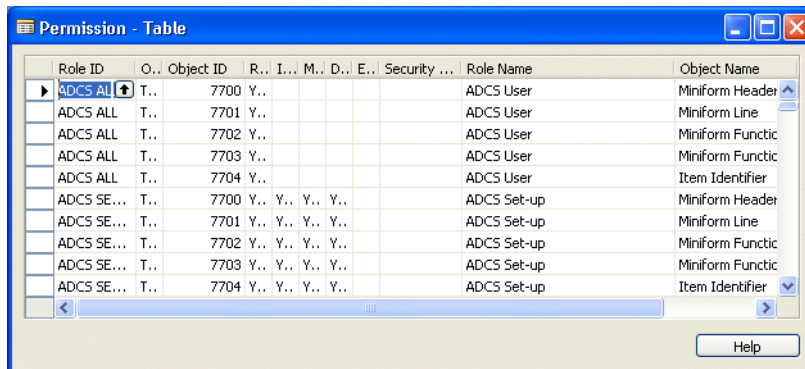
#### Deleting a Record

If you delete a record in the **User Role** system table, the corresponding entries in the **Member Of** and **Permission** system tables are also automatically deleted.

### The Permission System Table

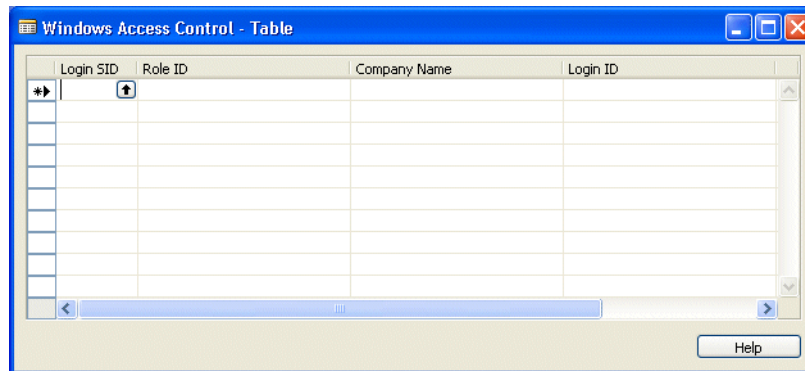
You can use the **Permission** system table to define what the different user roles are allowed to do. Permissions are specified for objects; so you can specify the exact set of permissions per table, form, and so on. You can specify that a user role has no (blank field), Full (Yes), or Indirect permissions to perform the following actions:

- Read
- Create/Insert
- Modify
- Delete
- Execute



### The Windows Access Control System Table

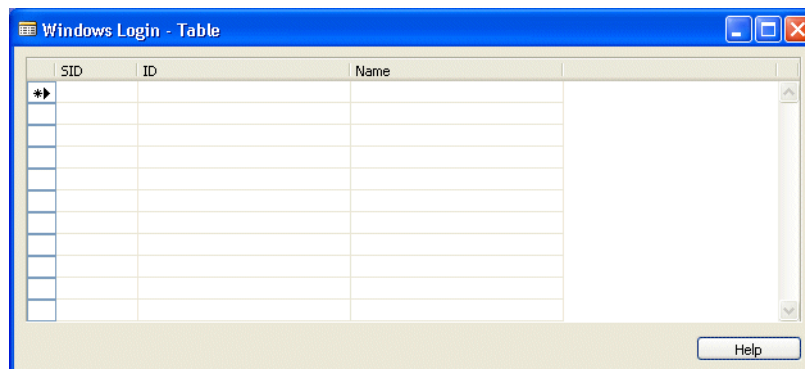
You can use **Windows Access Control** system table to manage the Windows access rights of a user or group of users, and thereby control their access to Dynamics NAV. Each user's or group's Windows login has a unique security identifier (SID). In addition, each user or group has a role ID, which relates to a set of permissions within a certain company in Dynamics NAV. The information displayed in the **Login ID** and **Role Name** fields is based on the login SID and role ID, respectively.



Login SID	Role ID	Company Name	Login ID
**▶	⬇		

### The Windows Login System Table

You use the **Windows Login** system table to define which Windows users and groups can log on to the system. Only those Windows users or those who are members of a Windows group that are listed here can log on. Each Windows user or group has a unique security identifier (SID). The name of the user or group that is displayed in the **ID** field is generated from the name of the user or group that is identified by the SID. The **Name** field is currently unused.

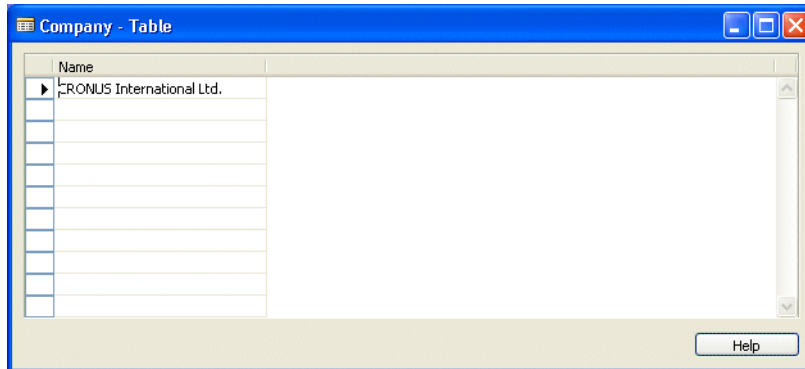


SID	ID	Name
**▶		

### The Company System Table

The **Company** system table gives you an overview of the companies in your database. It contains a record for each company in your database. You can create a new company by entering a new record in this table. You can also delete a company from your database by deleting the corresponding record in the **Company** table.

When you delete a company, you delete all the tables in the company and all the permissions that include this company.



### The Database Key Groups System Table

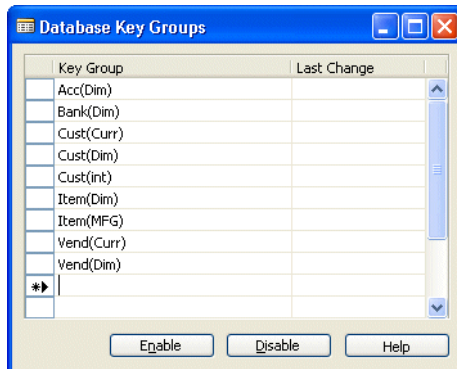
The **Database Key Groups** system table gives you an overview of the key groups defined in your database. Each record in this table shows a key group.

#### Note about Key Groups

By making your keys members of key groups, you can activate or deactivate various combinations of keys in your tables by enabling or disabling the key groups.

To enable and disable key groups:

- 1 Click File, Database, Information and the **Database Information** window appears.
- 2 Click Tables and the **Database Information (Tables)** window appears.
- 3 Click Key Groups and the **Database Key Groups** window appears:



- 4 Select a key group and click Enable or Disable to enable or disable it.



## 8.3 What Is a Virtual Table?

A virtual table contains information provided by the system. In C/SIDE you have access to a number of virtual tables. They work in much the same way as normal database tables, but you cannot change the information in them. That is, you can only read the information. Another difference is that virtual tables are not stored in the database (as normal tables are) but are computed by the system at run time.

### When to Use Virtual Tables

Virtual tables give you a consistent interface to a variety of different information. Because a virtual table can be treated just like an ordinary table, you can use the same methods to access information in virtual tables as you use when you are working with ordinary tables. For example, you can use filters to get subsets or ranges of integers or dates.

The virtual tables provide such information as:

- integers in the range – 1.000.000.000 to 1.000.000.000.
- dates within a given period.
- an overview of the operating system files.
- an overview of the logical disk drives.
- a trace of database requests from your client to the database.
- an overview of the users that are currently connected to the database.
- an overview of the operating system files that store the database.

### Overview of the Virtual Tables

C/SIDE contains numerous virtual tables, including:

#### Virtual Tables

---

*Date, Integer, File, Drive, Monitor, Session, Database File, Table Information, Field, Key, Server, Windows Object, Windows Group Member, SID - Account ID, User SID*

---

### Using the Virtual Tables

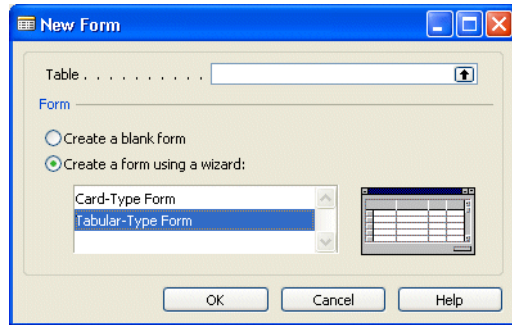
The first of these virtual tables gives you easy access to dates, integers, information about your operating system files, and the logical drives on your computer.

Because the virtual tables are not stored in the database, you cannot view them directly. To view a virtual table, you must create a tabular form based on it.

To view a virtual table:

- 1 Open the Object Designer (SHIFT+F12).
- 2 Click Form, New to open the **New Form** window.

- 3 Select the **Create a form using a wizard** option and make sure you select the Tabular-Type Form:



- 4 In the **Table** field enter the name of the virtual table that you want to base the form on. Alternatively you can use the lookup button to select the table from a list of all the tables in the database. The virtual tables use the highest number range (2000000001 – 2000000203). In this example we use the **Date** table.

- 5 Click OK and the **Tabular-Type Form Wizard** appears:



- 6 Click the >> button to move all the available fields over to the **Field Order** field.
- 7 Click Preview to view the table and its contents. Alternatively, click Finish and save the form.

This is the method used in the rest of this chapter to view virtual tables.

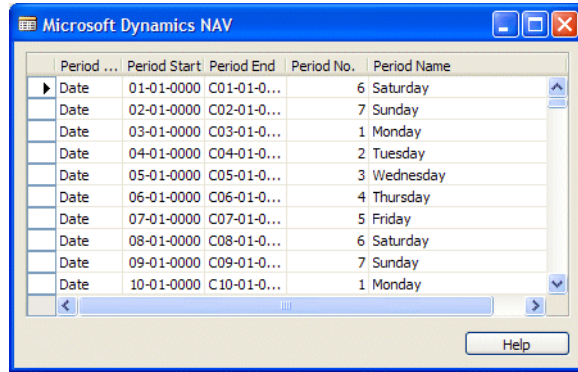
### The Date Virtual Table

This virtual table gives you easy access to days, weeks, months, quarters and years. The **Date** virtual table has the following fields:

Field	Comments
<b>Period Type</b>	Days, weeks, months, quarters or years
<b>Period Start</b>	The date of the first day in the period
<b>Period End</b>	The date of the last day in the period
<b>Period No.</b>	The number of the period

Field	Comments
Period Name	The name of the period

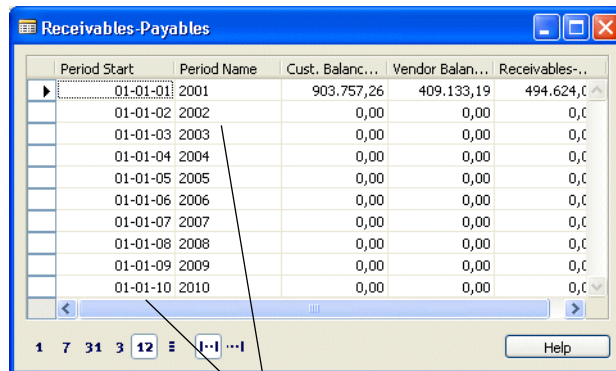
The following picture illustrates how you should think of the **Date** virtual table. For each period type, there are many records in the **Date** table:



You can apply filters to the **Period Type**, **Period Start**, and **Period End** fields to easily get a subset or range of days, weeks, months, quarters or years to use in your forms or reports.

**Example**

The **Date** virtual table is most frequently used to provide a range of dates, the **Receivables-Payables** form is a typical example.



This information is provided by the **Date** virtual table

**The Integer Virtual Table**

This virtual table includes integers in the range – 1,000,000,000 to 1,000,000,000. The **Integer** virtual table contains only one field:

Field	Comments
Integer	An integer in the range -1.000.000.000 to 1.000.000.000

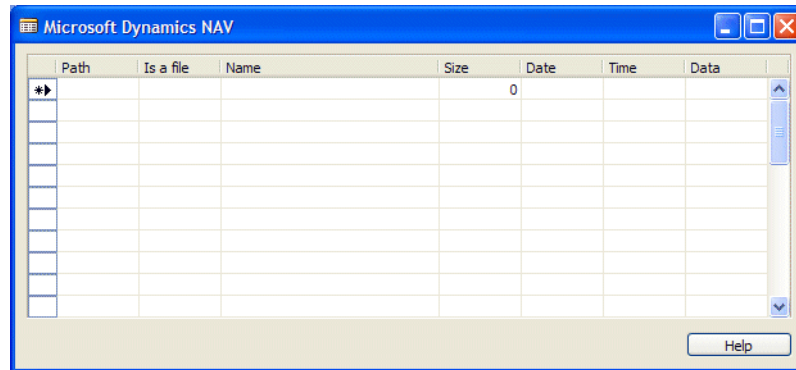
By applying a filter to this virtual table, you can easily get a subset or range of numbers that can be used to control looping in reports.

**The File Virtual Table**

This virtual table gives you an overview of the files in a directory on your disk system. The **File** virtual table contains the following fields:

Field	Comments
<b>Path</b>	The filter on this field determines which directory will be shown.
<b>Is a File</b>	The value Yes indicates that the entry is a file, while the value No indicates that the entry is a directory.
<b>Name</b>	The name of the file or directory.
<b>Size</b>	The size of the file in bytes.
<b>Date</b>	The date the file was last modified.
<b>Time</b>	The time the file was last modified.
<b>Data</b>	A BLOB field with the contents of the file.

You must create a tabular-type form to access the **File** virtual table:

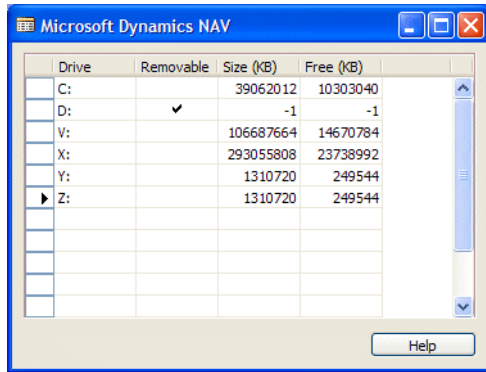


**The Drive Virtual Table**

This virtual table gives you an overview of the logical drives on your computer. The **Drive** virtual table contains the following fields:

Field	Comments
<b>Drive</b>	The name of the drive, such as A: or D:
<b>Removable</b>	Indicates whether the disk is removable (a floppy disk or CD ROM) or a fixed disk
<b>Size (KB)</b>	The total size of the disk
<b>Free (KB)</b>	The amount of free space on the disk

You must create a tabular-type form to access the **Drive** virtual table:



**Note**

If there is no disk in your disk drive, the **Size (KB)** and **Free (KB)** fields will contain **-1**.

The other virtual tables are most commonly used by the system administrator. They give the system administrator information about the users that are currently connected and the current state of the system.

**The Monitor Virtual Table**

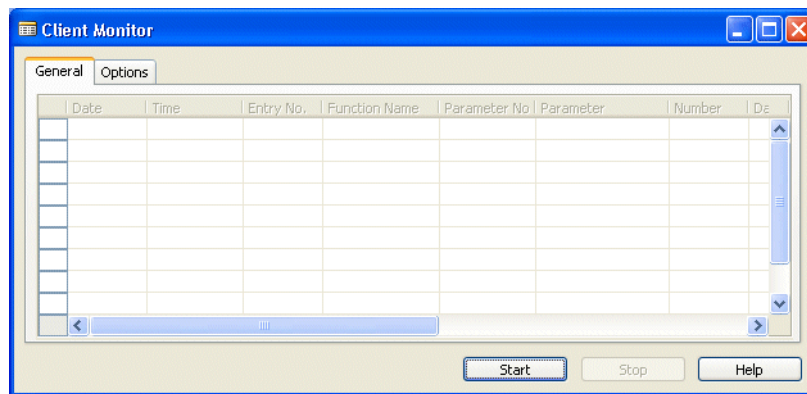
This virtual table traces all the database requests made by the client to the tables in your database. The **Monitor** virtual table is used by C/AL programmers to get an overview of the amount of time that specific operations take. C/AL programmers can use the information in this virtual table to optimize the performance of their code.

The **Monitor** virtual table contains the following fields:

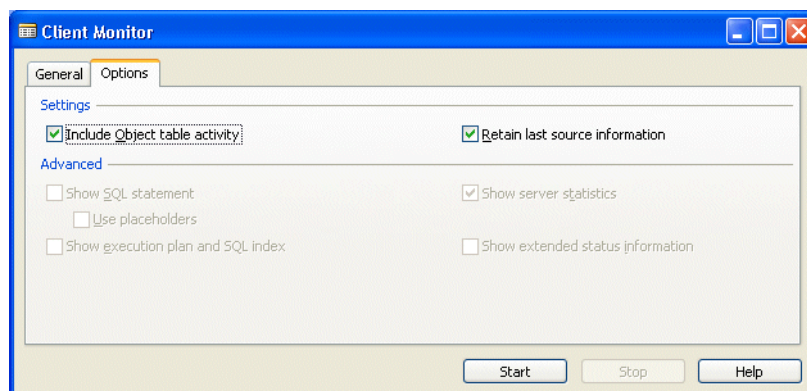
Field	Comments	Possible Values
<b>Entry No.</b>	Successive numbers that are increased for each database request	From 1 to $2^{31}-1$
<b>Function Name</b>	The type of database request	LOCKTABLE, DELETE, MODIFY, INSERT, DELETEALL, Create Key, Delete Key, Redesign Table, FIND/NEXT, CALCSUMS, CALCSUMS (Slow), COMMIT, Delete Table, Create Database, Close Database, Open Database, Delete Database, Expand Database, Get Table Statistics, COUNT, Get Database Statistics, Optimize Key, Login, Read Database Block, Read BLOB, Insert BLOB, Delete BLOB, Clear Old Versions, Get Database Free Percent, Preload Database Block, and so on.
<b>Parameter No.</b>	The number of the parameter	Depending on the number of parameters

Field	Comments	Possible Values
<b>Parameter</b>	The name of the parameter	Table, Key, Order, Filter, Search Method, Search Result, Records Found, Sum, CPU (ms), Records Read, Sum Intervals, Records Deleted, Records Modified, Disk Reads, Disk Writes, Record, Wait, SumIndexFields, BLOB Field, Commit, User ID, File Name, Source Object, Source Trigger/Function, Source Line No., Source Text, Record Count, Timeout Status, Time Out (ms)
<b>Number</b>	If the parameter is a number, the value is shown in this column	Any numeric value
<b>Data</b>	Any non-number parameter is shown in this column	Any string

To access the **Monitor** virtual table, click Tools, Client Monitor and the **Client Monitor** window opens:



The **Client Monitor** window contains two tabs. You use the **Options** tab to specify the kind of information that is collected by the Client Monitor.



The **Options** tab contains the following parameters, and the advanced parameters are only available in the SQL Server Option:

Parameter	Behavior When Selected
Include Object table activity	Every function that acts on the <b>Object</b> table is written to the Client Monitor.
Retain last source information	The parameters Source Object, Source Trigger/Function, Source Line No. and Source Line are written to the Client Monitor even for functions that are not related to the execution of C/AL code.
Show SQL statement	The SQL statement is displayed.
Use placeholders	The SQL statement uses '?' placeholders instead of filter values.
Show server statistics	The following statistics are collected – 'Server Time', 'Logical Reads' and 'Records Read'.
Show execution plan and SQL index	<p>The SQL plan is displayed (for most statements), as a collapsed tree with the format:</p> <p>Plan Step(Object)[n,p];... Here, node n has parent p. Example: Computer Scalar[2,1];Clustered Index Seek(User\$0)[4,2]</p> <p>This defines the tree: 1 -- Root -----2 -- Computer Scalar -----4 -- Clustered Index Seek(User\$0)</p> <p>The SQL index is also displayed as a list of fields in the same way as the Order parameter.</p>
Show extended status information	The SQL status displays additional information: internal unique statement ID, reuse status, prepared status, cursor type, optimizer hints, transaction type.

### Client Monitor – Additional Parameters for the SQL Server Option

Some extra parameters are available on the Client Monitor to improve troubleshooting and performance analysis when you are running the SQL Server Option for Dynamics NAV. These parameters can be configured dynamically and include status information about caching, SQL statements and execution plans.

Collecting server statistics is time-consuming. If you are performing benchmarking to get the most accurate value for the *Elapsed Time (ms)* parameter, you should not collect statistics at the same time.

Similarly, displaying the execution plans is extremely time-consuming and should not be done when you are benchmarking. This parameter is useful when you are

troubleshooting problematic application areas to determine if a particular SQL statement is a bottleneck, and can be valuable to users who are unable to run the SQL Profiler tool.

**Note**

.....

Executing a SQL statement in the Query Analyzer to use its graphical execution plan does not necessarily give the same plan or statistics as when the same statement is executed from within Dynamics NAV. This is due to cursor type differences. The SQL Profiler or the Client Monitor SQL Plan parameter give the most accurate plan.

.....

Showing extended status information is useful when you want to see which properties of a SQL statement are being used in an operation. It is also possible to see how frequently statements are being reused or re-created. The unique ID can be used to cross-reference the statements that are being reused and determine the original SQL statement entry.

To use the Client Monitor to monitor server calls:

- 1 Click Tools, Client Monitor to open the Client Monitor.
- 2 Click the **Options** tab and select the parameters that you want to use.
- 3 Click Start to activate the Client Monitor.
- 4 You can now close the **Client Monitor** window while you perform the tasks that you want to investigate.
- 5 When you have completed these tasks, click Tools, Client Monitor to open the **Client Monitor** window again.
- 6 Click Stop to stop the Client Monitor.



## The Session Virtual Table

This virtual table gives you an overview of the users that are connected to C/SIDE Database Server (NDS) or to SQL Server.

The **Session** virtual table contains the following fields:

Field	Comments	SQL Server	NDS
<b>Connection ID</b>	The ID of the connection.	X	X
<b>User ID</b>	The user ID of the connected user.	X	X
<b>My Session</b>	Whether or not a session belongs to you.	X	X
<b>Login Time</b>	The time when the user logged in and started this session.	X	X
<b>Login Date</b>	The date when the user logged in.	X	X
<b>Database Name</b>	The name of the database that this session has opened.	X	X
<b>Application Name</b>	The name of the application connected to the server.	X	X
<b>Login Type</b>	Whether this session is a Windows login or a database login.	X	X
<b>Cache Reads</b>	The number of cache read operations performed by this session. <sup>(A)</sup>		X
<b>Disk Reads</b>	The number of disk read operations performed by this session. <sup>(A)</sup>		X
<b>Disk Writes</b>	The number of disk write operations performed by this session. <sup>(A)</sup>		X
<b>Records Found</b>	The number of records found since this session logged in.		X
<b>Records Scanned</b>	The number of records scanned by this session since they logged in.		X
<b>Records Inserted</b>	The number of records inserted by this session since they logged in.		X
<b>Records Deleted</b>	The number of records deleted by this session since they logged in.		X
<b>Records Modified</b>	The number of records modified by this session since they logged in.		X
<b>Sum Intervals</b>	The number of jumps between value intervals made by the system when calculating sums since this session logged in. A high value may indicate that an inefficient key is being used.		X
<b>Host Name</b>	The name of the workstation used by this session.	X	
<b>CPU Time (ms)</b>	The cumulative amount of CPU time by this session.	X	

Field	Comments	SQL Server	NDS
<b>Memory Usage (KB)</b>	The number of kilobytes in the procedure cache that are currently allocated to this session.	X	
<b>Physical I/O</b>	The cumulative amount of disk reads and writes for this session.	X	
<b>Idle Time</b>	The amount of time that has passed since the server last received a database request from this session. If the field is empty, the session is currently performing a database request. This could be a long running database request, such as a table redesign.	X	X
<b>Blocked</b>	Whether or not this session is blocked (waiting to acquire a lock) by another session.	X	
<b>Wait Time (ms)</b>	The amount of time that this session has been waiting.	X	
<b>Blocking Connection ID</b>	The ID of the connection that is blocking this session.	X	
<b>Blocking User ID</b>	The user ID of the connection that is blocking this session.	X	
<b>Blocking Host Name</b>	The name of the workstation used by the connection that is blocking this session.	X	
<b>Blocking Object</b>	The name of the SQL object that is blocking this session.	X	

(A) Only if Commitcache = Yes

**Important**

.....

If a solution uses any of the fields that are not available in the SQL Server Option for Dynamics NAV, it must be modified to run on SQL Server. These fields will not be created on SQL Server. If the program tries to access them, an error message appears.

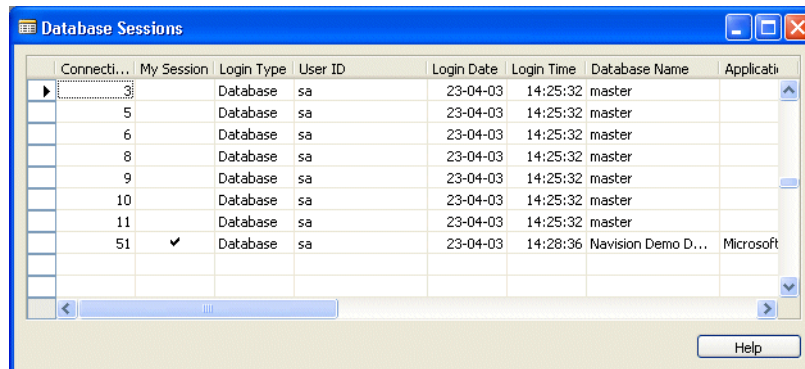
.....

C/SIDE uses the **Session** virtual table to display database information.

To access this virtual table from C/SIDE:

- 1 Click File, Database, Information and the **Database Information** window appears.
- 2 Click the **Sessions** tab.

- 3 Click the AssistButton ↓ in the **Current Sessions** field and the **Database Sessions** window (SQL Server Option) appears:



#### Note

Depending on which server option you are running, the **Database Sessions** window displays different fields from the **Session** virtual table.

An administrator can terminate one of the sessions by selecting the line in question and deleting it. The user will then be disconnected from the server and will have to restart the program if they want to continue working. To terminate a session, the administrator must have permission to delete data from the **Session** table.

Furthermore, on the SQL Server Option, the administrator *must* be a member of either the *sysadmin* or *processadmin* SQL Server server roles.

#### Note

On C/SIDE Database Server, the terminated session will no longer take up any resources on the server.

### The Database File Virtual Table

This virtual table gives you an overview of the operating system files that store the database. The **Database File** virtual table contains the following fields:

Field	Comments
<b>No.</b>	The number of the operating system file.
<b>File Name</b>	The operating system file name.
<b>Size (KB)</b>	The size of the operating system file in KB.
<b>Total Reads</b>	The number of read accesses since the database was opened. <sup>(B)</sup>
<b>Mean Read Time (ms)</b>	The average time for a read operation (in milliseconds). <sup>(B)</sup>
<b>Reads In Queue</b>	Number of read operations waiting in queue. <sup>(A)(B)</sup>
<b>Total Writes</b>	Number of write operations since the database was opened. <sup>(B)</sup>
<b>Mean Write Time (ms)</b>	The average time for a write operation (in milliseconds). <sup>(B)</sup>
<b>Writes In Queue</b>	Number of write operations waiting in queue (in milliseconds). <sup>(A)(B)</sup>
<b>Disk Load (%)</b>	A percentage weight describing the load on the disk. <sup>(B)</sup>

(A) Only if Commitcache = Yes

(B) Not available in the SQL Server Option for Dynamics NAV.

#### Important

.....

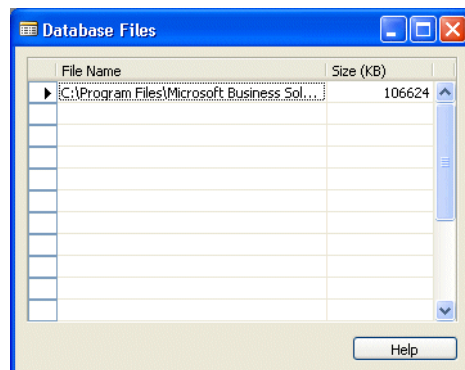
If a solution uses any of the fields that are not available in the SQL Server Option for Dynamics NAV, it will have to be modified to run on SQL Server. These fields will not be created on SQL Server. If the program tries to access them, an error message appears.

.....

C/SIDE uses this virtual table to show database information.

To access the **Database File** virtual table:

- 1 Click File, Database, Information and the **Database Information** window appears.
- 2 Click the **Database** tab.
- 3 In the **Database Name** field, click the AssistButton ↓ and the **Database Files** window appears:



**Note**

The **Database Files** window displays only some of the fields in the **Database File** virtual table. This window does not appear in the SQL Server Option.

**The Table Information Virtual Table**

This virtual table contains information about database tables. The **Table Information** virtual table contains the following fields:

Field	Comments
<b>Company Name</b>	The name of the company the table belongs to.
<b>Table No.</b>	The ID number for the table.
<b>Table Name</b>	The name of the table.
<b>No. of Records</b>	The number of records in the table.
<b>Record Size</b>	A value expressing the average size of a record. Calculated as $1024 * \text{Size(KB)} / \text{Records}$ .
<b>Size (KB)</b>	How much space the table occupies in the database (in KB).
<b>Optimization</b>	A percentage of Size that expresses how much data there is in a table. Some of the remaining size is used for internal administration in the table while other is slack-space. Slack-space can be minimized by optimizing the table. <sup>(A)</sup>

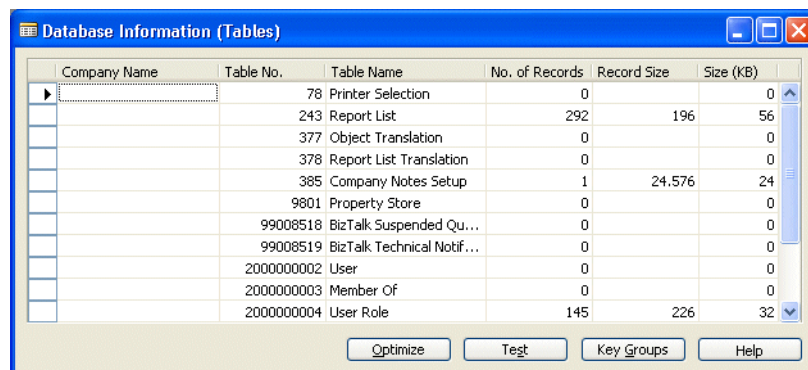
(A) Not available in the SQL Server Option for Dynamics NAV.

**Important**

If a solution uses any of the fields that are not available in the SQL Server Option, it must be modified to run on SQL Server. These columns will not be created on SQL Server. If the program tries to access them, an error message appears.

To access the **Table Information** virtual table:

- 1 Click File, Database, Information and the **Database Information** window appears.
- 2 Click Tables and the **Database Information (Tables)** window appears:



**Note**

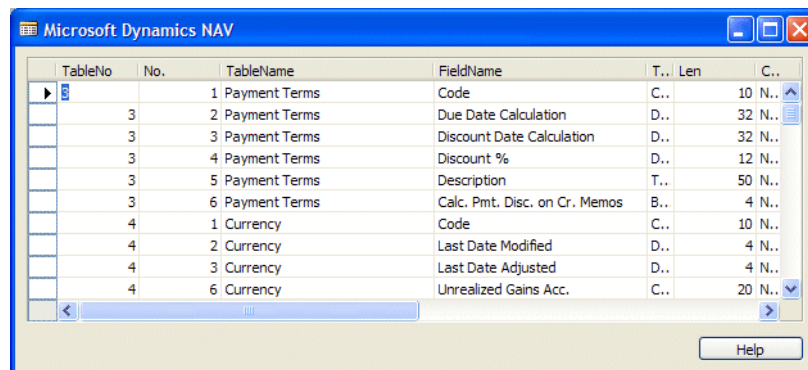
This screen shot displays the **Database Information (Tables)** table for the SQL Server Option. On C/SIDE Database Server this window contains an extra field and an extra button.

**The Field Virtual Table**

This virtual table contains information about fields in database tables. The **Field** virtual table contains the following fields:

Field	Comments
<b>TableNo</b>	The ID number for the table.
<b>No.</b>	The number assigned to the field.
<b>Table Name</b>	The name of the table.
<b>FieldName</b>	The name of the field.
<b>Type</b>	The data type assigned to the field, for example, decimal.
<b>Len</b>	The length of the field entry in bytes.
<b>Class</b>	The class of the field, for example, FlowField.
<b>Enabled</b>	Whether the field is enabled.
<b>Type Name</b>	The data type assigned to the field. The length of the field entry in bytes is included for Code and Text data types.
<b>Field Caption</b>	The caption of the field in the language that has been selected.
<b>RelationTableNo</b>	The ID number for the table that the field is related to.
<b>RelationFieldNo</b>	The number of the field in another table that the field is related to.
<b>SQLDataType</b>	The data type assigned to code fields in the SQL Server Option.

You must create a tabular-type form to access the **Field** table:



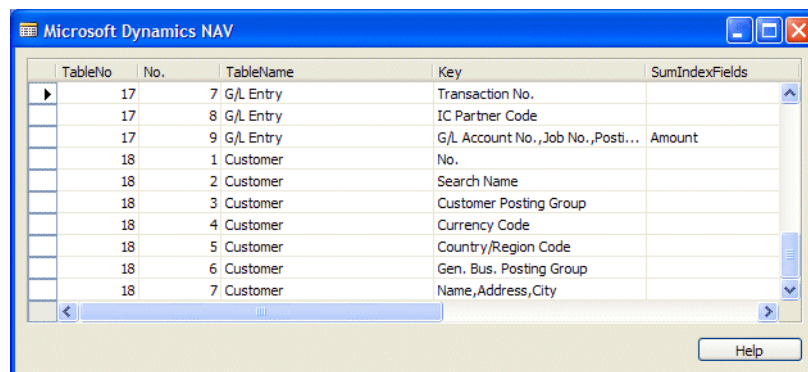
**The Key Virtual Table**

This virtual table contains information about the keys that are defined in each table in the database.

The **Key** virtual table contains the following fields:

Field	Comments
<b>TableNo.</b>	The ID number for the table.
<b>No.</b>	The number of the key.
<b>TableName</b>	The name of the table.
<b>Key</b>	The fields that make up the key.
<b>SumIndexField</b>	The SumIndexFields that are defined for this key.
<b>SQLIndex</b>	The actual fields that have been defined in Dynamics NAV and are used in the corresponding index on SQL Server instead of those defined in the key.
<b>Enabled</b>	Whether or not the index is enabled. This field can be modified.
<b>MaintainSQLIndex</b>	Whether or not the MaintainSQLIndex property has been activated. This field can be modified.
<b>MaintainSIFTIndex</b>	Whether or not the MaintainSIFTIndex property has been activated. This field can be modified.
<b>Clustered</b>	Whether or not the key is clustered. This field can be modified.

You must create a tabular-type form to access the **Key** table:



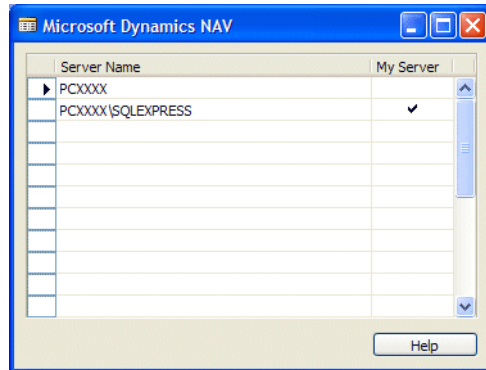
### The Server Virtual Table

If you are running on the SQL Server Option for Dynamics NAV, you can access the **Server** virtual table.

The **Server** table contains the following fields:

Field	Comments
Server Name	The name of the computer on which the server is installed.
My Server	Whether or not this is the server that you are logged on to.

You must create a tabular-type form to access the **Server** table:



### The Windows Object Virtual Table

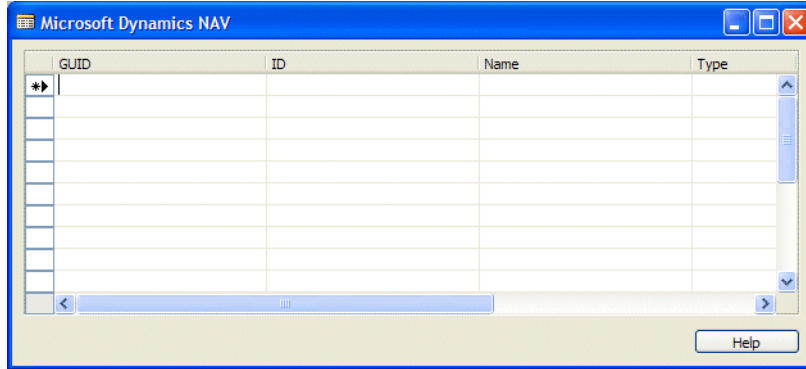
This virtual table gives you an overview of the Windows users and Windows groups that can be integrated into the Dynamics NAV security system. You can see this information when Dynamics NAV is running on a client that is Active Directory enabled. Dynamics NAV retrieves the data from Active Directory.

The **Windows Object** virtual table has the following fields:

Field	Comments
<b>GUID</b>	The globally unique identifier (GUID) for the Windows user or group.
<b>ID</b>	The ID of the Windows user or the Windows group. This information is displayed in the <b>User ID</b> fields of the <b>User</b> and <b>Member Of</b> system tables.
<b>Name</b>	The name of the Windows object. This object can be a Windows user or a Windows group. The object name is displayed in the <b>Name</b> field of the <b>User</b> system table and in the <b>User Name</b> field of the <b>Member Of</b> system table.
<b>Type</b>	Whether the object is a Windows user or a Windows group.
<b>SID</b>	The unique security identifier (SID) of the Windows user or group.
<b>Distinguished Name</b>	Identifies the domain that holds the Windows object as well as the complete path to the object. Every object in the Active Directory has a unique distinguished name.



You must create a tabular-type form to access the **Windows Object** table:



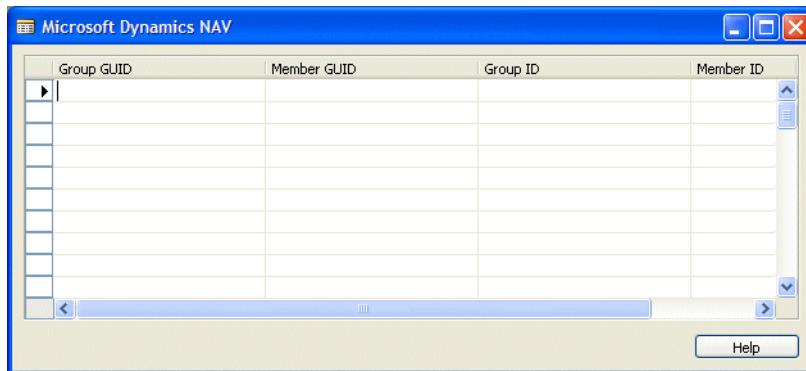
**The Windows Group Member Table**

This virtual table contains information about the members of Windows Groups who can be integrated in the Dynamics NAV security system. A Windows group member who has permissions in the Dynamics NAV security system does not have to enter a password when they log on to Dynamics NAV.

You can see this information when Dynamics NAV is running on a client that is Active Directory enabled. The **Windows Group Member** virtual table contains the following fields:

Field	Comments
<b>Group GUID</b>	The globally unique identifier (GUID) of the Windows group.
<b>Member GUID</b>	The GUID for the Windows group member.
<b>Group ID</b>	The ID of the Windows group to which the Windows group member belongs.
<b>Member ID</b>	The ID for the Windows group member.

You must create a tabular-type form to access the **Windows Group Member** table:



**The SID - Account ID Virtual Table**

This virtual table can convert the security identifier (SID) of a Windows object into an ID. It can also convert an ID of a Windows object into a SID. The Windows object can be a Windows user or group. The ID is calculated on the basis of the SID.

If you request a record with a specific SID, C/SIDE looks up the information in the **SID - Account ID** virtual table and returns the ID.

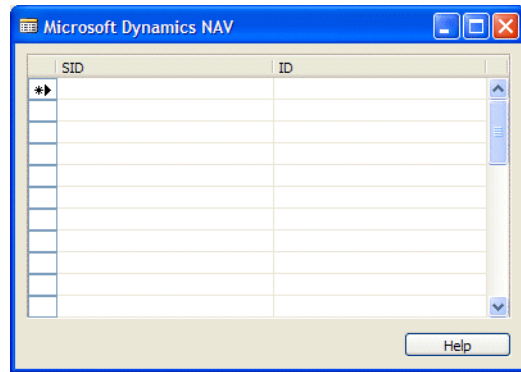
The **SID - Account ID** virtual table has the following fields:

Field	Comments
<b>SID</b>	The security identifier (SID) of the Windows user or group.
<b>ID</b>	The ID of the Windows user or group. The ID is calculated on the basis of the SID.

**Note**

.....  
 This table will always appear to be empty.  
 .....

You must create a tabular-type form to access the **SID - Account ID** table:

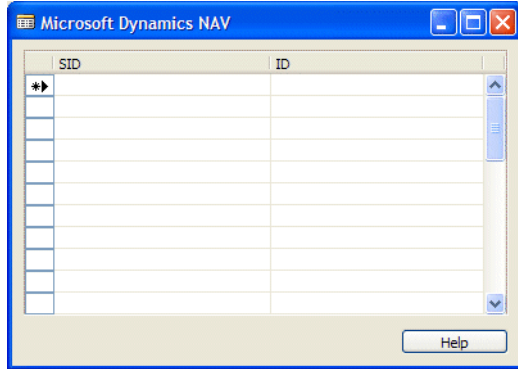


**The User SID Virtual Table**

This virtual table shows the security identifiers (SIDs) and IDs of the groups that the user who is logged on to the system is a member of. The **User SID** virtual table has the following fields:

Field	Comments
<b>SID</b>	The security identifier (SID) of the groups that the user who is logged on to the system is a member of.
<b>ID</b>	The ID of the groups that the user who is logged on to the system is a member of. The ID is calculated on the basis of the SID.

You must create a tabular-type form to access the **User SID** table:





**Part 4**  
**Forms**



## **Chapter 9**

### **Form Fundamentals**

Forms are used to enter and display data. For example, you can use a form to enter information about new customers or to update and review information about existing customers.

This chapter introduces the fundamental concepts and basic tasks involved in designing and using forms.

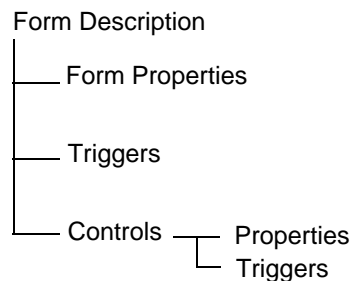
- What Are Forms?
- Creating Forms
- Saving, Compiling and Running Forms

## 9.1 What Are Forms?

After you have created tables, the next step in developing a C/SIDE application should be to design forms. In contrast to programs written in traditional programming languages, C/SIDE applications do not execute sequentially: they are *event-driven*. A major part of the logic of an application could be said to rest with the forms. You use forms to enter information into database tables and to retrieve and display information from them. It is through forms that you generate the events that determine the flow of the application.

Forms can be used to access one table at a time, or they can combine information from a number of different tables. A form can display information that is calculated on the fly as the form is displayed, and it can contain information (such as a label) that is not related to any table, or purely decorative elements (such as bitmap pictures).

The following figure shows how the components of a form are related:



Forms are created and edited in the Form Designer.

### What are Controls?

All the information displayed in a form is presented in *controls*. Controls are objects that can display data from a database table field, the value of a C/AL expression, bitmap pictures or static information such as a descriptive text.

Some controls are called container controls. A frame is a container control and does not display data or information but acts as a container for a number of other controls. The tab control is another container control. Tabs are really a number of frames or pages stacked on top of each other. You click the tabs to switch between the pages. Tab controls are another way to group information on a form. Tabs are useful controls because they help avoid clutter and make it easy to move from one page to another.

Control branches consist of a parent control, and subordinate or child controls. An example is a text box and a label. The child control inherits some properties from the parent, and the entire branch can be moved together.

### What Are Bound and Unbound Forms and Controls?

Typically, a form is bound to a database table, and used to enter information into the table and to display information from that table. An unbound form is not related to a table. An example of an unbound form is a form that is used as a menu from which you can choose to run other forms or reports.



Controls on a form that is bound to a table are usually bound to fields in the same table. There doesn't have to be a control for every field, nor do all controls need to be bound to fields. Controls that are not bound to fields are unbound controls. For example, a command button that prints the information on the form and a control that contains a descriptive text are unbound controls. Unbound controls include controls that display information that is calculated as the form is displayed. This information is based on the underlying table, or on values that are entered by the user. Note that a blank form is unbound when it is created.

### What Are Form and Control Properties?

Properties are used to define how a control is placed on a form, what field it is related to in the underlying table and what happens when information is entered into the field. Different types of controls have different sets of properties. For example, a text box normally displays the contents of a database field and so has more properties than a picture box.

The form itself also has properties. For example, you can specify whether the form should only be used for displaying information or for inserting new records or updating existing ones. Properties are defined on the **Properties** window which can be edited when the form is opened in the Form Designer.

### What Are Triggers?

When C/AL functions are executed as a result of you performing certain predefined events on either a form or a control, it is said that the event triggers the function. The event and function together make a trigger.

Form triggers include OnOpenForm (statements that are executed when the form opens), and OnModifyRecord (statements that are executed before the system accepts changes that the user has made to a record. Triggers are edited in the C/AL editor that can be opened from the Form Designer.

## 9.2 Creating Forms

Forms can be created and designed manually. Although this method gives you the highest degree of control, it may take some time to master. Alternatively you can use a form wizard, which is fast and easy and normally preferable when you are just beginning to create forms. A form wizard prompts you for the minimum amount of information needed to create a form and then does the rest of the work. After the form has been created by the wizard, you can use the Form Designer to alter it to meet your requirements.

You can create two kinds of form in Dynamics NAV:

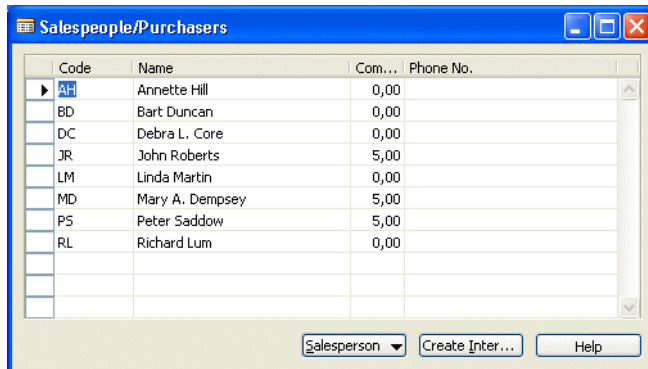
- Card Forms that display one record at a time.
- Tabular forms that display several records at a time.

You can use the form wizard to create both kinds of form.

A card form



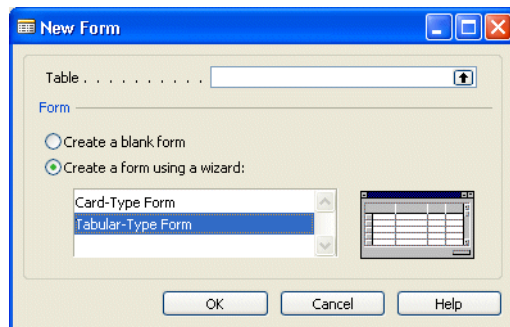
A tabular form



## Creating Forms with a Form Wizard

To create a form with the form wizard:

- 1 Click Tools, Object Designer (SHIFT+F12) to open the Object Designer.
- 2 In the Object Designer, click Form to open the Form Designer.
- 3 Click the New button to open the **New Form** window:

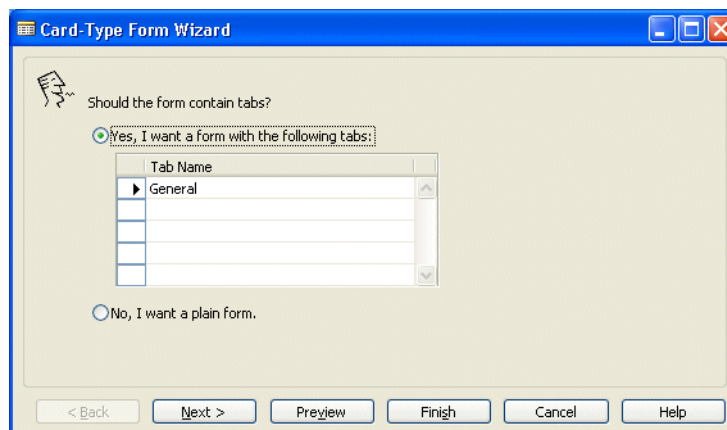


- 4 If you are creating a form that is related to a table, type the name of the table in the **Table** field. Alternatively, you can click the lookup button and select the table from a list.
- 5 Select the "Create a form using a wizard" option.
- 6 Select the type of form you want the wizard to create: a card form or a tabular form and click OK.

## Creating a Card Form

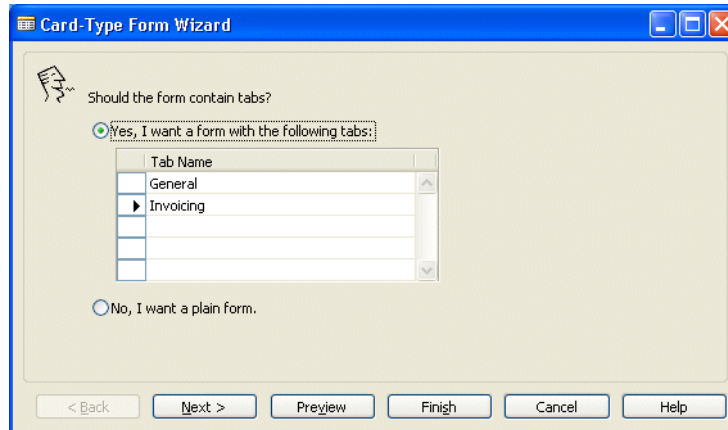
To create a card form, follow the steps that have just been described and proceed as follows:

- 1 After you have chosen to use the wizard to create a card-type form and clicked OK, the Card Form Wizard opens and will guide you through the process of creating the form:



- 2 The first thing you must decide is whether the form should have tabs or if it should be a plain form. A form with tabs is a multi-page form that allows you to switch between the pages by clicking the tabs.

In this example, we will create a form with tabs that is based on the **Customer** table.



3 Enter a caption or name for each tab that you want and click Next.



4 You must now decide which fields from the **Customer** table you want to include on your form.

When you are creating a form with tab controls, you begin by choosing the tab on which you want the fields to appear. You can switch between the tabs by clicking them. The tabs have the captions that you defined earlier.

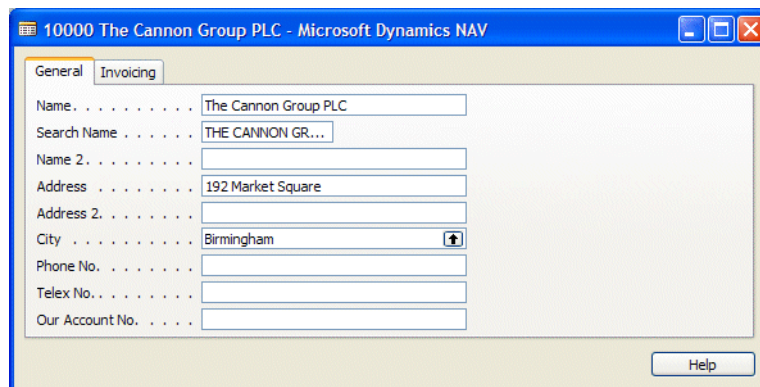
5 The form that the wizard displays contains two lists: the **Available Fields** list, which contains all the fields in the table and the **Field Order** list, which contains the fields that you have selected.

To insert a field in the **Field Order** list, select it in the **Available Fields** list and click >. You can insert all the fields at once by clicking >>. You can remove fields from the **Field Order** list by selecting them and clicking <. You can remove them all at once by clicking <<.

**Note**

.....  
 The contents of the **Available Fields** list are the *Caption* properties of the fields in the table - not the *Name* properties.  
 .....

- 6 The order of the fields in the **Field Order** list is the order in which the fields will appear on the form. If you want the fields to appear in a different order, you can move a field by removing it from the **Field Order** list and inserting it again in the position you want.
- 7 To insert a small amount of extra vertical space between the controls, click Separator. This allows you to group information together in a logical and visually pleasing way. (Tabs provide a more powerful way of grouping information). The separator is inserted after the field that is currently selected on the **Field Order** list. You can remove a separator by selecting it and clicking <.
- 8 To insert a column break, click Column Break. The rules for insertion and deletion are the same as for a separator. If you need to create three or more columns, consider using tabs instead.
- 9 To see what your form looks like, click Preview.



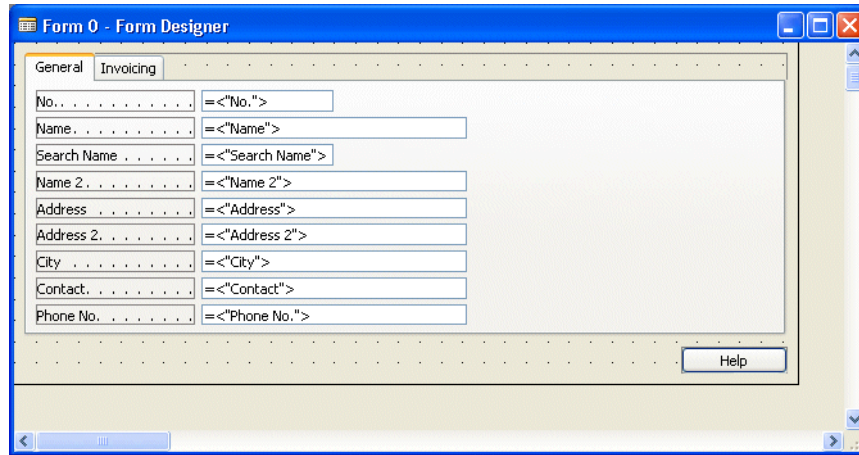
The screenshot shows a window titled "10000 The Cannon Group PLC - Microsoft Dynamics NAV". The window has two tabs: "General" and "Invoicing", with "Invoicing" selected. The form contains the following fields:

Name . . . . .	The Cannon Group PLC
Search Name . . . . .	THE CANNON GR...
Name 2 . . . . .	
Address . . . . .	192 Market Square
Address 2 . . . . .	
City . . . . .	Birmingham
Phone No. . . . .	
Telex No. . . . .	
Our Account No. . . . .	

A "Help" button is located at the bottom right of the form.

- 10 If you are not yet satisfied with the form, just close it and you are back in the wizard. You can continue to adjust the form until you are satisfied with its design.

11 When you are satisfied with the form, click Finish and the form is opened in the Form Designer.

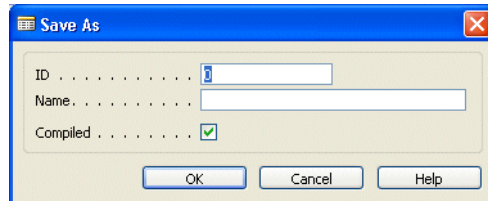


You can now manually adjust the form. More on this later.

For the moment, you can just close and save the form.

12 Close the form. You are now asked whether or not you want to save it.

13 Click Yes and the **Save As** window appears:



14 Enter an ID number and name for the new form, and you can choose whether or not the form will be compiled now.

You can also compile it later, by selecting the form in the Object Designer and clicking Tools, Compile.

15 When the form has been saved and compiled, you can run it. Select the form in the Object Designer and click Run.

## Creating a Tabular Form

To create a tabular form, follow the initial steps described in the section called "Creating Forms with a Form Wizard" on page 143. Then proceed as follows:



- 1 You must now decide which fields from the **Customer** table you want to include on your form. The form that the wizard displays contains two lists: the **Available Fields** list, which contains all the fields in the table, and the **Field Order** list, which contains the fields that you have selected.

To insert a field in the **Field Order** list, select it in the **Available Fields** list and click **>**. You can insert all the fields at once by clicking the **>>**. You can remove fields from the **Field Order** list by selecting them there and clicking **<**. You can remove them all at once by clicking **<<**.

### Note

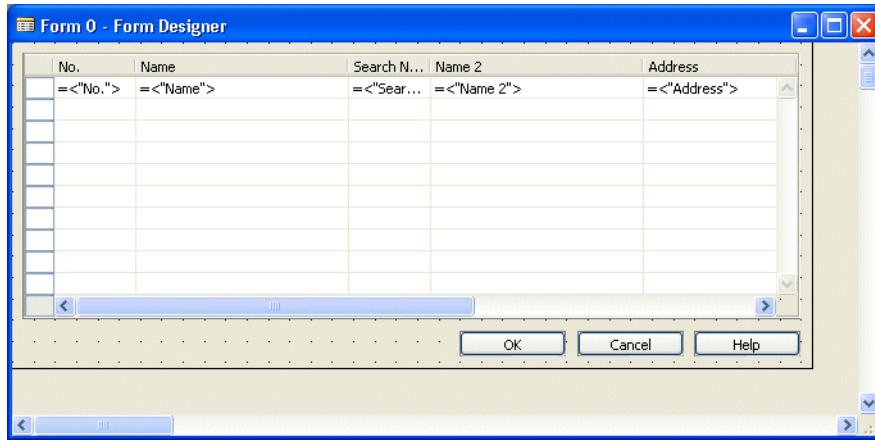
.....

The contents of the **Available Fields** list are the *Caption* properties of the fields available to you – not the *Name* properties.

.....

- 2 The order of the fields in the **Field Order** list is the order in which the fields will appear on the form. If you want the fields to appear in a different order, you can move a field by removing it from the **Field Order** list and re-inserting it in the position you want.
- 3 Click **Preview** to see what your form looks like.

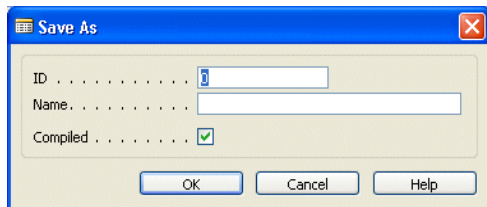
- When you are satisfied with the form, click Finish and the form is opened in the Form Designer:



You can now manually adjust the form. More on this later.

For the moment you can just close and save the form.

- Close the form. You are now asked whether or not you want to save it.
- Click Yes to save the form. You are prompted to enter an ID number and name for the new form, and you can choose whether or not the form will be compiled now. You can also compile it later, by selecting the form in the Object Designer and clicking Tools, Compile.



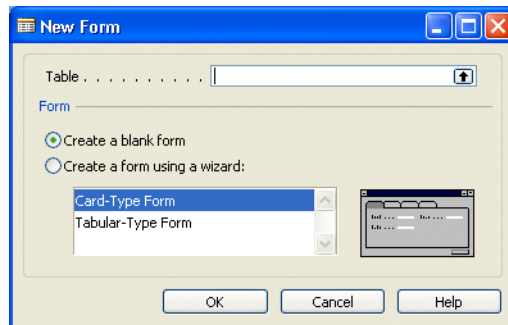
- When the form has been saved and compiled, it can be run. Select the form in the Object Designer and click Run.




## Creating Forms Without a Wizard

To create a form without using a wizard:

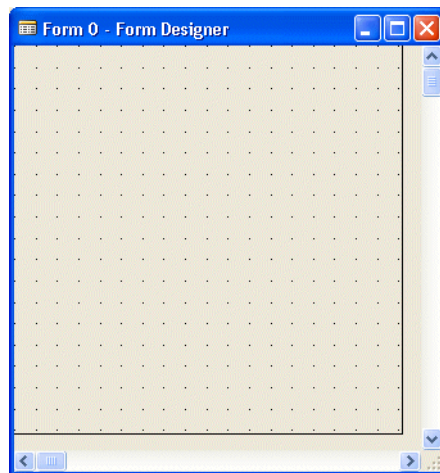
- 1 Click Tools, Object Designer (SHIFT+F12) and the Object Designer opens.
- 2 In the Object Designer, click Form, New and C/SIDE opens the **New Form** window:



If you are creating a form that is related to a table, type the name of the table in the **Table** field. Alternatively, you can click the AssistButton  and select the table from a list.

- 3 Select Create a blank form, and click OK.

The Form Designer opens displaying an empty form:



Chapter 10, "Designing Forms" on page 153, describes in detail how to design the form by adding controls, changing properties and so on.

- 4 Close the Form Designer, and answer Yes to save the form. You are prompted to enter an ID number and name for the new form, and you can choose whether or not the form will be compiled now. You can also compile it later, by selecting the form in the Object Designer and clicking Tools, Compile.

## 9.3 Saving, Compiling and Running Forms

After you have designed a form, you must save and compile it before it can be run. Normally, you do this when you have finished designing the form. However, you may want to save a form that is not yet finished and cannot be compiled, particularly if the form contains C/AL code. You can also test-compile and test-run a form without closing or saving it.

### Saving and Closing a Form

A designed form is closed when the Form Designer is closed.

To save a form:

- 1 When you are closing a form, C/SIDE asks whether you want to save the form or not. If it is a new form (a form that has not been saved before), you will have to give it an ID and a name. The ID must be unique and follow the rules for numbering objects – your local Microsoft Certified Business Solutions Partner will provide you with this information.

If you enter ID and Name as form properties, these values will be used, and you will not be prompted for an ID and a name when you close the form.

- 2 The **Compiled** option field is selected by default (a check mark). If your form is not ready to be compiled, click in the field to remove the check mark.
- 3 Click OK to save the form.

You can save a form without closing it by clicking File, Save or Save As... You can also use Save As... to give a form a new name.

### Compiling a Form

Forms, like the other objects in C/SIDE, must be compiled before they can be run. As described earlier, you can compile a form whenever you save it.

While you are designing a form, you may want to test-compile it to find possible errors. This is particularly useful when the form contains C/AL code in triggers. You can test-compile a form when you are designing it by clicking Tools, Compile.

### Running a Form

In a finished application, your forms are incorporated into menus or else they are called from other forms. However, when you are designing forms, you will often want to run them before they are integrated into an application.

You can run a form from the list of forms in the Object Designer by selecting it and clicking Run. Forms can also be run from inside the Form Designer by clicking File, Run.

## **Chapter 10**

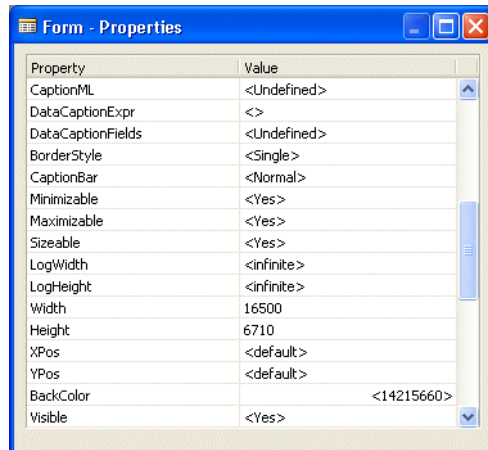
### **Designing Forms**

This chapter describes how to design forms by adding controls and by changing the properties of forms and controls.

- Form and Control Properties
- Types of Controls
- Adding Controls
- Selecting, Moving and Adjusting Controls
- Tools for Customizing Controls
- Setting Control Properties
- Container Controls
- How to Use Controls in Applications

## 10.1 Form and Control Properties

Remember that properties are a system-wide feature and every application object has some properties. You edit the properties of forms and controls by opening the **Properties** window in the Form Designer (click View, Properties or SHIFT+F4). As you select a form or control, the **Properties** window displays the properties of the selected object. The title bar of the **Properties** window shows which kind of object (form, text box, label, and so on) is currently selected. The first line in the window shows the ID of the object.



Each line in the **Properties** window contains a property which you can set in the **Value** field. When you leave the field, press ENTER or use the arrow keys, the property is updated. If your entry contains an error (for example, if you accidentally change the ID of one control to be the same as that of another), the update is not accepted.

The default values are displayed in angle brackets <>. If a property has a default value, you can reset it by deleting the current value and then moving out of the field. Some properties do not have default values – mainly those that describe the position of the control within the form. These properties are updated by C/SIDE when the control is moved.

### How Properties Are Inherited

Controls that have a direct relationship to table fields inherit the settings of those properties that are common to both the field and the control. You can change the settings of these properties for the control, but you cannot overrule the settings of the field properties that concern data validation. For example, if the field property that determines which characters the user can enter is set to *lowercase only*, you cannot use the properties of the control to reset it to also accept uppercase characters. You can narrow the accepted range of characters but not broaden it. On the other hand, you can change properties like the caption because this property has nothing to do with data validation.

When you design an application, you must consider whether these common properties should be specified at field or control level. The advantage of using the lowest level (the field level) is that it ensures consistency because whenever the field is used as the data source of a control, these settings are used as defaults.

## Form Properties

The following table briefly describes some of the more important form properties. There is a description of all the properties in the *C/SIDE Reference Guide* online Help. You can get context-sensitive Help by opening the form's **Properties** window, placing the cursor on a property, and pressing F1.

Property Name	Use this property to...
ID	set the numeric ID of the object. This property can also be set when you save a form. The ID must be unique among forms. Your Microsoft Certified Business Solutions Partner can tell you the number range that you can use.
Name	give the form a descriptive name. No two forms can have the same name.
Minimizable	specify whether you can minimize the form window.
Maximizable	specify whether you can maximize the form window.
Sizeable	specify whether you can resize the form window.
SavePosAndSize	specify whether information about the user-made changes to the size and placement of a form window are saved. If it is set to Yes, this information is saved, and the next time the window is opened, these values are used. Otherwise, the designed values are used.
Editable	specify whether you are allowed to edit controls in the form. If it is set to No, controls may not be edited, even when their individual <i>Editable</i> properties are set to Yes.
InsertAllowed	specify whether the form can be used to insert records in the database.
ModifyAllowed	specify whether the form can be used to modify records in the database.
DeleteAllowed	specify whether the form can be used to delete records from the database.
CalcFields	specify a list of FlowFields that you want the system to calculate when the form is updated. If the FlowField is a direct source expression, it is automatically calculated. However, if it is indirect (part of an expression) it is not.
UpdateOnActivate	specify whether you want the system to update the form when it is activated.
SourceTable	specify the source table of the form. Normally, you specify the table when you first create the form. If you have created a form without an underlying table, you can enter a table name here to bind the form to a table.
SourceTableView	create a view (what you can see) of the source table for this form. You can specify the key, sorting order and filter that the system will use.
SaveTableView	specify whether the system will save information about which record the user is viewing when the form is closed, the sorting order and the current filter, and then reapply this information when the form is opened again.

## General Properties for Controls

The following table briefly describes those properties that are common to several types of controls. There is a description of all the properties in the *C/SIDE Reference Guide*

online Help. You can get context-sensitive Help for a property by opening the **Properties** window for a form, selecting a control, placing the cursor on a property and pressing F1.

<b>Property Name</b>	<b>Use This Property to...</b>
ID	set the numeric ID of the control. The system assigns a sequential number by default. If you delete a control, and then add another in its place, you may want to give the newly created control the number of the one you deleted. The ID must be unique among controls and menu items on the form.
Name	give the control a descriptive name.
Caption	specify the text that the system displays for this control.
HorzGlue	to anchor a control horizontally on the form. You can choose Left, Right or Both. If you choose Both, the control is resized when the form is resized.
VertGlue	to anchor a control vertically on the form. You can choose Top, Bottom or Both. If you choose Both, the control is resized when the form is resized.
Visible	specify whether the control is visible when the form is opened. This property can be changed from C/AL at runtime. Notice that if the control is a child control and the parent has Visible = No, the child will not be visible, even if it has Visible = Yes.
ParentControl	specify the ID of a parent control, thereby turning the control into a child.

## 10.2 Types of Controls

This section contains a brief overview of all the controls that can be added to a form. The following list is structured according to the broad categories into which controls can be grouped.

### Static controls

Static controls are used to contain and display descriptive or graphical information. You cannot change the contents of static controls at run time.

The static controls are:

**Label** A label is used to display text, most commonly the caption of another control. In this situation the label is normally (and conveniently) a child of the other control, but labels can also be used as stand-alone controls.

**Image** An image control is used for displaying a picture.

**Shape** A shape is a graphical element (line, circle, rectangle and so on).

### Data controls

Data controls display the value of a C/AL expression, for example the value of a table field or a variable or a "real" expression. (The simplest expression is just the name of a table field or a variable.) The valid combinations of data control and data type are:

Control	Valid data types
Check Box	Booleans and BLOBs
Option Button	All, except BLOBs
Text Box	All
Picture Box	Boolean, option, integer and BLOBs
Indicator	Integer, decimal, date, time

Data controls *must* have a relation to data, defined as their *SourceExpr* property.

### Containers

Container controls are used to group other controls together. Some properties of the container overrule the same property in the contained controls. If the container is not editable, then none of the contained controls can be edited (even if they individually have the *Editable* property set to *True*).

**Frame** A frame is simply a rectangle into which other controls can be "dropped". In the Form Designer, the frame and the controls that it contains can be moved together. A frame can have different border styles and colors than the form that it is part of.

**Tab Control** A form can be thought of as a book that contains several pages, or frames. Only one of these pages is visible at a time. You can switch between the pages by clicking on the tabs.

## Data Containers

**Table Box** A special kind of container that holds repeated data controls and is used to create columnar tables. Each data control contained by the table box constitutes one column for which a static control is used as a heading. The rows arise from vertically repeating each data control. If the table box displays records from a table, each row displays one record.

## Other Controls

**Command Button** A command button is not related to data. It performs an action when it is clicked, or when ENTER or the spacebar is pressed while the button has the focus.

**Menu Button** A menu button can be clicked just like a command button, but it does not perform an action. When you click it, a menu opens containing a number of menu items that you can choose.

**Menu Item** The lines in a menu that can be chosen are called menu items. Each menu item resembles a command button because it can perform an action when you click it.

**Subform** A subform control is used to display a second form in a control on a form (a main form). This allows the main form to show data from two different tables. For example, the main form could be a card form and show records from a customer table, while the subform could be a tabular form and show details about any purchases the customer has made.

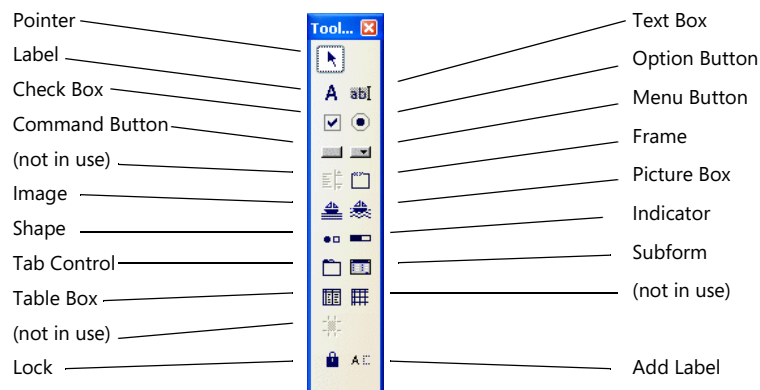


## 10.3 Adding Controls

This section consists of a few examples that show how to add controls without using the form wizards.

### The Toolbox

You use the Toolbox to insert controls. To open the toolbox, click View, Toolbox or click the Toolbox button on the toolbar. You select a specific tool by clicking the corresponding icon in the toolbox.



Note that some of these tools are not implemented in the current version of C/SIDE, but that the icons are already present – they will, however, always appear disabled.

You use the Pointer tool to select controls on a form in the designer. When you select any of the other tools, the cursor changes from a selection tool to an insertion tool.

Normally, after you have inserted a control, you must select the control again in the toolbox before inserting the next control. However, the Lock tool maintains the current control selection and you can continue inserting controls of the same type without having to select the control again in the toolbox.

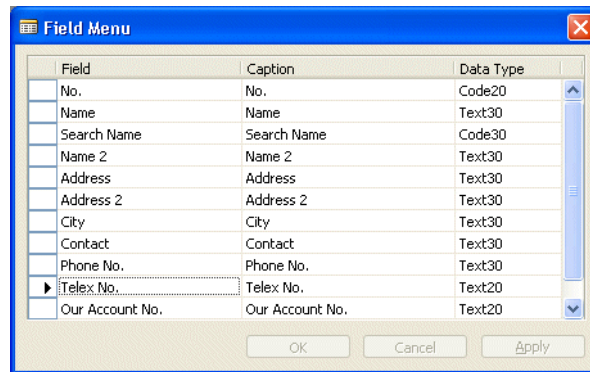
If the Add Label tool is also selected, all the controls that you add to the form are given a label when you insert them.

### Adding a Text Box

If the text boxes that you want to add to the form are related to database table fields, the easiest way to add them is to use the Field Menu. The Field Menu is a window that lists all the fields in the table that has been defined as the Source Table for the form. You can open the Field Menu in the Form Designer by clicking View, Field Menu.

To add a text box that has a specific field in the table as its SourceExpr (that is, has this field as its underlying table field) to the form:

- 1 Open the Form Designer.
- 2 Create a blank form based on the **Customer** table.
- 3 Click View, Field Menu to open the **Field Menu** window:



- In the **Field Menu** window, select the field or fields. The **Field** field contains the *Name* property, and the **Caption** field contains the *Caption* property. For more information about these properties, see Chapter 22, "MenuSuite Objects".
- In the **Field Menu** window, select the field (or fields) that you want to place on the form. The **Field** field contains the *Name* property, and the **Caption** field contains the *Caption* property. The **Data Type** field tells you the data type of the field.



Move the cursor into the design area and it changes into the Control Insertion cursor. You do not have to drag and drop; just select the field(s) and move the cursor over the form designer.

- In the Form Designer, click once to activate the designer and click again to insert the text box(es). The text box is inserted below the point where you clicked. You can move it later if you want.

The text box that is created has the default size. You can click and drag to create a text box with a different size.

If you selected more than one field, the text boxes are inserted and aligned in a column below the mouse position. They are not linked together and can be moved apart later.

Each text box has these characteristics:

- It has the table field as its SourceExpr.
- The default settings for the *Name* and *Caption* properties are the same as the setting for the *Name* property of the underlying table field.
- In general, every property that is both a field property and a text box property has the value of the field property in the underlying table as its default value.
- The text box has a label with a caption that defaults to the caption of the text box.

The advantage of using the Field Menu to add text boxes with labels is that you are effortlessly assured that both the naming and the properties are consistent.

If the data type of the field is anything but Boolean, a text box is created, but if the data type is Boolean, a check box is created.

### Adding a Text Box without Using the Field Menu

Although the easiest way to add a text box is by using the Field Menu, you can add text boxes without using the menu. This is the method you use when you want to add a

calculated text box, that is, a text box that is used to display a calculated value. It is also possible to add an unbound text box and then, later on, bind it to a table field.

To add an unbound text box:

- 1 Open the form in the Form Designer.
- 2 Open the Toolbox and select the Text Box tool.
- 3 Move the cursor into the design area.
- 4 Click to add a text box of the default size, or click and drag to create a text box with a different size.

Now you have an unbound text box control on the form. Notice that no characteristics were inherited, and that the text box has no label.

Later you will learn how to change the properties of a control including how to bind the text box to a table field and add a label. You will also learn how to use a text box to display a value that is calculated on the fly.

### Creating Labels That Display Descriptive Text

You can add a label that is not the child of another control to a form. You can do this, for example, if you want to have descriptive text on the form. This could be instructions about using the form, or other static information not related to any database table field.

To add a label:

- 1 Open the form in the Form Designer.
- 2 Open the Toolbox and select the Label tool.
- 3 Move the cursor into the design area.
- 4 Click to create a label of the default size, or click and drag to create a label of a different size.

As the label is not part of a control branch, it is given a default name and caption (like *Control4*). You can change the name and the caption on the **Properties** window for the label (see "Changing the Properties of a Control" on page 165).

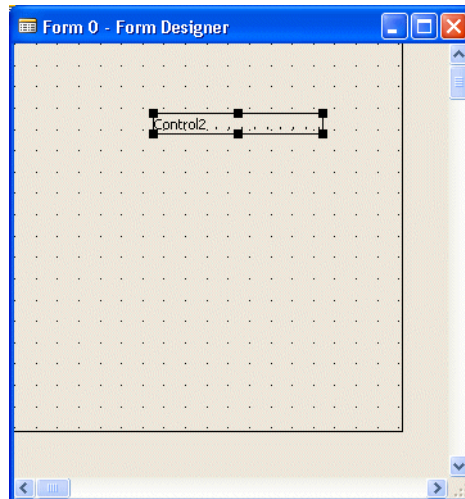
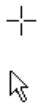
## 10.4 Selecting, Moving and Adjusting Controls

This section describes how to select and move controls, and how to adjust controls by aligning and sizing them.

### Selecting Controls

Before you can move or adjust a control, you must select it. As some operations can be applied to only one control at a time and others to a group of controls, controls can be selected both individually and as groups.

You select a control by clicking on it. To make it easier to see what the mouse cursor is pointing at, the appearance of the cursor changes as it is moved around the design area. The default appearance is a cross, which means that the cursor is not currently pointing at any control. As soon as the cursor points at a control, it changes into a selection cursor. Clicking the mouse selects the control, which is then surrounded by a box with sizing handles.

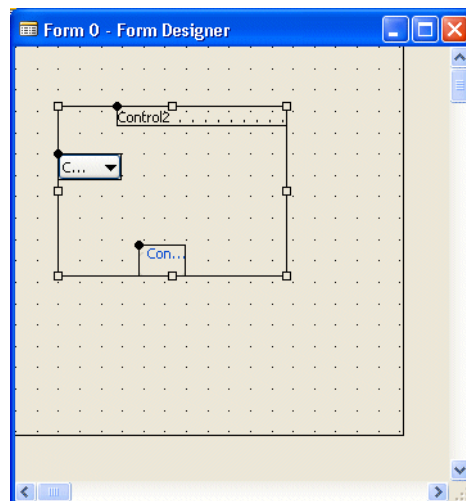


### Multiple Selections

A multiple selection is a group of controls that are all selected. You make a multiple selection, for example, in order to align all the controls in the selection (how to actually align the controls is described later).

### Adding to a Selection

When one control has been selected, you can add other controls to the selection by holding down the CTRL key and then selecting the other controls. As you select more controls, a box appear around the complete selection and each control in the selection is marked by a circle in the upper left corner of its own bounding box.



### Marquee Selection

Another way to make a multiple selection is to use *marquee selection*. When the mouse cursor is in the design area but not pointing at anything (appears as a cross), press the left mouse button and hold it down. Then drag the mouse and a rectangle appears. This is called a marquee.

As the rectangle expands, any control that it overlaps, completely or partially, is selected; and a circle appears in the upper left corner of its bounding box. Release the mouse button when you have selected all the controls that want.

Controls can be added to the selection individually (as described earlier), and a marquee selection can be added to an existing selection by holding down the CTRL key while you carry out the marquee selection.

#### Note

There is an option that determines how marquee selection works. Click Tools, Options. In the **Options** window, the Marquee Full Selection option can be set to *Yes* or *No*. If the option is set to *No*, the marquee selection works as described earlier. If it is set to *Yes*, a control is only selected if the marquee overlaps the control completely – not just partially.

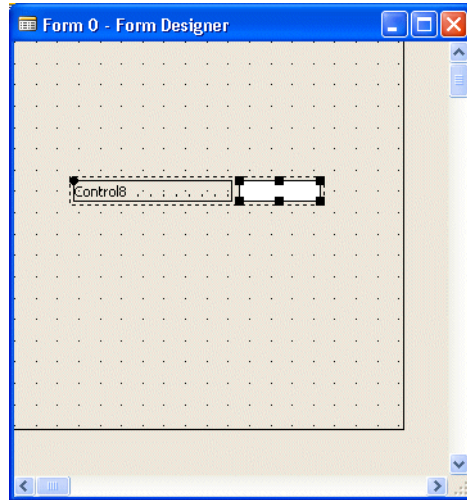
### Selection and Container Controls

When you select a container control, all the controls that it contains are selected, even if they are not marked as such by individual bounding boxes. Beware that a contained control can be moved so that it is only partly inside the container. However, it is considered part of the control as long as any part of it overlaps the container. To select such a control, you must click in the part that is still inside the container.

### Selection and Control Branches

A control branch consists of the control itself and one or more child (subordinate) controls. An example is a text box with a label. Both are controls – the text box holding information that can change during program execution and the label holding static information (usually a caption for the text box), changeable only during form design. The label is said to be a child of the text box.

When a control branch, such as a text box with a label, is selected, the control itself is displayed in a bounding box with sizing handles, as usual. The child controls that are part of the control branch are marked by a box with a circle in the upper left corner, and the whole branch is surrounded by a dotted emphasis frame. If you click on the emphasis frame (the cursor changes into a selection cursor as it touches the frame), the child controls are added to the selection; this turns the selection into a multiple selection that can be moved as a whole.



### Moving Controls

When the selection cursor appears, you can move the control below it by pressing the left mouse button and holding it down while you drag the control to the desired position. The control is dropped when you release the mouse button.

### Moving Selected Controls

Controls also can be moved after they have been selected. To move a selected control, move the mouse cursor towards it. When the cursor touches the border of the control, it looks like a hand. Press and hold down the left mouse button, drag the control to the desired position, and then release the mouse button.

Multiple selections are moved as a whole and their relative positions within the selection are not changed.

### Aligning Controls

If you created a form without using a wizard – or if you did use a wizard but rearranged some controls afterwards – you may want to align the controls more precisely.

There are two methods for aligning controls easily and accurately in C#/SIDE:

- 1 You can turn on the Snap to Grid option by clicking Format, Snap to Grid. Now when you move a control you will notice that it doesn't move smoothly, but rather in small, fixed increments. The dots in the design area represent some of the actual grid points that the controls snap to when they are moved.

Hint: the distance between the grid points are properties (*HorzGrid* and *VertGrid*) of the form. The unit is 1/100 millimeters.





- 2 To align several controls, make a multiple selection of the controls and click Format, Align. In the submenu that appears, select one of the four ways to align the controls.

If, for example, the controls are in a column, you will want to align them vertically, either to the left or to the right. Select Left or Right to do this. Correspondingly, a row of controls can be top or bottom aligned. Beware however that if, for example, you inadvertently choose to top align a column of controls the system will indeed do just this and will place all the controls on top of each other.

### Sizing and Resizing Controls

When the wizard adds controls to a form, these controls are sized evenly according to a default scheme. If you move the controls around, the sizes that the wizard assigned may no longer be appropriate. Other situations where you will want to change the size of a control are if you change the font size, or if you don't want to display all the information from a very large table field, but only the first part. You can only resize one control at a time.

To resize a control:

- 1 Select the control. It is now surrounded by a bounding box with sizing handles.
-   2 Select a sizing handle and the cursor turns into a sizing cursor.
-   3 Click the left mouse button and drag the control until it has the size you want. If Snap to Grid is turned on, the sizing takes place in fixed increments.

### Sizing Container Controls

If you have created a container control, you can size the contained controls individually in the usual manner. The containing control – for example, the frame – can be sized like any other control. When a containing control is sized, the contained controls are not affected, that is, neither their size nor their position changes.

When you enlarge a container, any control that becomes completely overlapped by it will automatically be 'adopted' as a child that is contained by the container control.

You should also be aware that you can reduce the size of a container control so that a control that it contains seems to be outside the container. However, this control is still considered to be part of the container. If no part of the control is inside the container control, it cannot be selected. The remedy for this is to enlarge the container so that all the contained controls are inside it.

## 10.5 Tools for Customizing Controls

In addition to the **Properties** window, you can use two special tools to set some of the properties of controls:

- Color tool, for selecting color properties and border styles.
- Font tool, for setting font properties.

### Using the Color Tool

To open the Color tool, click View, Color:



When a control with color properties is selected, you can pick the colors for foreground (text), background and border by clicking in the palette. The corresponding properties are *ForeColor*, *BackColor* and *BorderColor*.

The Background and Border check boxes are used to determine whether or not the background color and the border color are displayed. The corresponding properties are *BackTransparent* and *Border* (if these options are *off*, a background or border color has no effect).

If the control has a border, the nine buttons at the bottom can be used to select the style and width of the border.

### Using the Font Tool

To open the Font tool, click View, Font. The tool looks like this:



When a control that can display text is selected, you can use this tool to set the font properties. You can enter the font name, the font size, attributes (bold, italic and underline) and the horizontal alignment of the text (left, center, right and general – *general* means that the text is left aligned and the numbers are right aligned).



## 10.6 Setting Control Properties

This section gives some examples of how to change the properties of a control, and indicates some of the typical situations where this will be necessary.

### Changing the Properties of a Control

If you have added controls without using the form wizards, you often need to change some of the properties of these controls. Even if you did use a wizard, you may want to change some of the properties to meet specific requirements that the wizards cannot consider.

### Changing the Name and Caption of a Control

Every control has an ID and a Name. Controls that display data also have a *SourceExpr* (source expression) property which is a C/AL expression. It can just be the name of a table field name or it can be a complex expression, perhaps with a field as an operand.

When you use a form wizard or the Field Menu to create a text box that has a direct relationship to a table field, the Name and Caption are set by default to the name of the table field (unless the table field has a Caption, in which case this Caption is used). The label has a Caption derived from the Caption of the parent control (the text box). You can supply a Caption in either place if you want to have a different, perhaps more descriptive, text than the field name as a caption.

Notice the following dependencies:

- If you change the *Caption* property of the text box, the *Caption* property of the label is set to this value as a default (displayed in angle brackets). If you change the text box *Caption* property again, the *Caption* property of the label is also changed again.
- If you change the *Caption* property of the label directly, the value you enter is displayed without angle brackets, signifying that it is no longer a default value. Now if you change the property of the text box again, the value here is not be updated.

### Changing an Unbound Control into a Bound Control

An unbound text box, or other data control, can be easily changed to a bound control. You must simply change the *SourceExpr* to the one that you want.

If it is the name of a field in the database table, the values for Name and Caption automatically default to the standard values of the bound control, that is, they default to the name of the table field. This will not automatically add a label to the text box.

If you want to change the source expression so that it refers to a field from another table, you must enter a more complex C/AL expression.

### Adding a Label to a Text Box

If you have created a bound text box by changing the *SourceExpr* of an unbound text box, the bound text box does not automatically get an attached label. You add a label by adding a label control to the form and then changing the *ParentControl* property of the label from the default (undefined) to the ID of the text box. You can see the ID of the text box in the first line of the **Properties** window for the text box.

The control branch resulting from this operation can be selected and moved just like any other control.

## Display Properties

Controls that you add to a form – either by using a wizard or manually – have a default set of properties that define how the control itself and the data it displays are formatted. While this ensures a consistent visual design throughout your applications, it cannot cover all your needs. You may therefore have to change some properties that affect the way your forms and their controls are displayed.

### Controlling the Display of Numbers

This is a short description of the properties that determine the way numbers are displayed. For a full description of these properties, see the *C/SIDE Reference Guide* online Help.

**DecimalPlaces** This property (that specifies both the minimum and maximum allowed values) determines how many decimals are displayed, as well as how many decimal places can or must be entered. A typical situation where you would use this property is when amounts are stored in the database with 5 decimal places for high precision but you only want to see the customary number of decimal places for the currency, for example 2. The table field would then have the *DecimalPlaces* property set to 2:5, while the + property of the text box is set to 2:2.

**BlankNumbers** You can choose from an option list whether a range of numbers will be displayed or blanked.

**BlankZero** The default is No. If you change it to Yes, zero values and Booleans that would have been displayed as a No are blanked out.

**Divisor** The default is Undefined. If a number is entered, numeric values are divided by this number when they are displayed. Any remainder is discarded. You could, for example, use the *Divisor* property to display only the thousands part of a number by entering 1000 (then 16400 and 16800 would each be displayed as 16).

### Formatting Data Display

This is a short description of the properties that determine the way data is formatted. For a full description of these properties, see the *C/SIDE Reference Guide* online Help.

**Format** This property defines how the system formats the *SourceExpr* of a text box. For each data type, there is a default. There is also a set of standard formats that you can select. You can also build your own formats if you need to.

**HorzAlign and VertAlign** These properties define how data in a text box or a caption on a label is aligned horizontally and vertically.

**MultiLine** This property determines whether or not labels and text boxes can contain multiple lines of text. The default is No with one exception: the label of a column in a table box will have this property set to Yes. For more details, see the subsection "Displaying More Than One Line of Text" on page 171.

**PadChar** This property specifies the character that should be used to pad a string. The character will be added to the left or right, or both, depending upon the text alignment defined by the *HorzAlign* property.

**LeaderDots** This property specifies whether or not there are leading dots before the data. The dots are placed according to the data's horizontal alignment. If the data is left aligned, the dots are placed on the right and if it is right aligned, dots are placed on

the left. If centered, there are dots both before and after the data. If this property is set to Yes, the PadChar setting is overruled.

### Properties That Control Input

This group of properties are used to control user input, that is, restrict user input to certain values or a certain length.

**Numeric** Restricts input to numeric values only if it is set to Yes.

**MinValue, MaxValue** Sets a minimum or maximum value that you can enter.

**ValuesAllowed** Specifies the values that you are allowed to enter. Enter the values separated by semicolons, like *1;7;4711* or *a;b;c*.

**CharAllowed** Specifies the characters that you can enter. You can use a range, for example *AZ*, to limit entry to uppercase characters only, or several ranges, for example *amot*, specifies two ranges: a to m and o to t.

**NotBlank** Specifies whether or not an entry can be blank. If this property is set to Yes, an entry that consists of nothing, one blank space or several blank spaces is not accepted. A blank can be part of string that contains other characters.

**MaxLength** Specifies the maximum number of characters that can be entered in a text box.

**AutoEnter** If set to Yes, the system accepts your entry when the maximum number of characters allowed has been entered into a table box, and it will then move the focus to the next control – that is, you do not have to press ENTER.

**PasswordText** Specifies whether or not your input is displayed as text. If this property is set to Yes, your input is not displayed but shown as asterisks *\*\*\*\*\**.

### Assisting the User

This group of properties is used to give some help to the user.

**ToolTip** This property allows you to assist the user by displaying text that describes a control.

If you enter a text for this property, it is displayed in a small pop-up window when the user holds the cursor over the control for a short while. The text is supposed to be a short, perhaps just one word, description of what the control is used for.

**DataCaptionField, DataCaptionExpr** This property is used to control the label that is displayed on the title bar of the form window. This is a static caption, usually the name of the underlying table.

By using DataCaptionField (either at table or form level), you can select fields from the record whose contents are displayed (and updated) in the caption bar as you page through the table. With DataCaptionExpr (form only) you can create a C/AL expression to be displayed in the caption bar. The expression is reevaluated when you select a new record or the present record is changed. For more information, see the *C/SIDE Reference Guide* online Help.

**Note**

.....

If the Status Bar option (click Tools, Options) is set to Yes, the caption of text boxes and check boxes is displayed in the status bar together with the current data contents of the control (if any) when the control gets the focus.

.....

## 10.7 Container Controls

As explained earlier containers are used to gather controls together in logical groups. This can help make the application more intuitive for the user and thereby easier to use. They can also make it easier for the designer to maintain.

### Using a Frame to Contain Controls

A frame is used to contain other controls. When controls are contained in a frame, you can perform some operations on all these controls during design. For example, the controls are all moved when the frame is moved, with their relative positions intact and if the frame is invisible all the contained controls are also invisible.

A frame can have a border that can be raised or sunken. This feature can be used to distinguish a group of controls (such as a group of option buttons) visually.

To create a frame with contained controls:

- 1 Open the form in the Form Designer.
- 2 Select the Frame tool; then click and drag in the design area to create a frame.
- 3 Create the controls to be contained by the frame in the usual way, placing them inside the frame as you add them to the form.
- 4 Set the properties of the frame to suit your purpose.

A common change is to stop caption display by setting the *ShowCaption* property to *No*. By default, the border style is *Raised*. If the frame's purpose is not to distinguish the contained controls, set the *Border* property to *No*, so that no border is displayed.

When an existing control is dragged inside a frame and dropped, it will be contained in the frame. When a control is dragged outside a frame, it is no longer considered to be contained by the frame. If a container is resized and then overlaps existing controls completely, these controls will be contained by the frame. If a frame is deleted, all the contained controls are also deleted.

### Creating a Tab Control

A tab control is useful when you are designing a form that is based on a table with many fields. Instead of creating a large form cluttered with controls, you can group controls together on pages that the user can bring to the front by clicking the tabs.

You can use a form wizard to create a form with a tab control.

To create a tab control manually:

- 1 Open the form in the Form Designer.
- 2 Select the Tab Control tool and click and drag in the design area to create a tab control.
- 3 Open the **Properties** window for the tab control, and create the pages you need by entering a name for each page as a comma-separated list in the *PageNames* property. The names are used as captions on the tabs.

The tabs are created while you are in the Form Designer. You can select pages by clicking the tabs.

Add controls on the pages. You can think of each page in a tab control as a frame and add controls as you would in a frame.

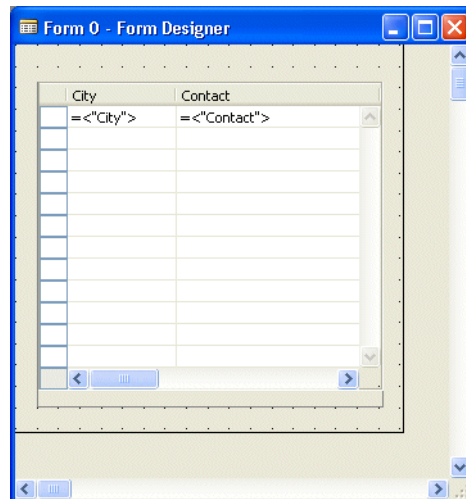
### Creating a Table Box

A table box is useful when you need to display many records from a database table at the same time. A table box contains columns and rows, and you can move through the records, either by using the vertical scroll bars or by using the arrow keys or PAGEUP and PAGEDOWN. If the form is too narrow to display all the columns, a horizontal scroll bar is automatically added to the control.

You can use a form wizard to create a form with a table box.

To create a table box manually:

- 1 Open the form in the Form Designer.
- 2 Open the Toolbox and select the table box tool.
- 3 Click and drag in the design area to create a table box.
- 4 If the form is not related to a table, you can now establish a relation by setting the *SourceTable* property of the form to the name of the table.
- 5 Click View, Field Menu to open the **Field Menu** window.
- 6 Select the fields you want in the table box from the Field Menu and click twice inside the table box. Once to activate the Form Designer and once to insert the fields. A column is added for each field, and each row will display a record from the table. A label, derived in the same way as a label for any text box, is added as a column heading.



## 10.8 How to Use Controls in Applications

This section describes some more of the methods that you can use to control the way data is displayed in your applications.

### Displaying More Than One Line of Text

If a database table contains very large fields, lengthy descriptive texts for example, using the standard one-line text box is not a very good way to present this information. Instead, you can customize a text box to wrap text into multiple lines.

To create a multiline text box:

- 1 Open the form in the Form Designer.
- 2 Select the text box in question and enlarge it vertically by resizing.
- 3 Open the **Properties** window for the text box (SHIFT+F4) and set the value of the *MultiLine* property to Yes.
- 4 Run the form. Entering or editing text will still take place on one line that scrolls horizontally. When the focus is not on the text box, the contents of the field will be formatted in multiple lines. Automatic line breaks occur only after a space character, and the user can insert line breaks ("hard newlines") by embedding a backslash character ("\") in a text string. (To display a backslash, enter "\\".)
- 5 You may have to experiment with the vertical resizing of the text box to find the size that suits your purpose best.

### Displaying a Calculated Value

A control can be used to display a value that is not stored in the database but calculated as the form is displayed. One situation where this could be useful is when all the information needed for the calculation is actually stored in the database, and – conforming to the rules for a relational database system – the calculated value is not stored separately. However, the users of the application do sometimes need this value. Adding a calculated control can give this information, without violating the rules for good database design.

To display a calculated value:

- 1 Open the form in the Form Designer.
- 2 Select a tool that inserts an appropriate data control (check box, text box, indicator) in the Tool Box.
- 3 Click twice in the design area to add the control.
- 4 Open the **Properties** window for the control. Type the expression you want as the *SourceExpr* property.

#### Example

You have designed a table with a field that contains the Unit Price of an item, and another field that contains the Employee Discount Rate. On the form, you want to see the price that an employee actually has to pay. Add an unbound text box and enter as the *SourceExpr*:

"Unit Price" - ("Unit Price" \* "Employee Discount Rate" / 100)

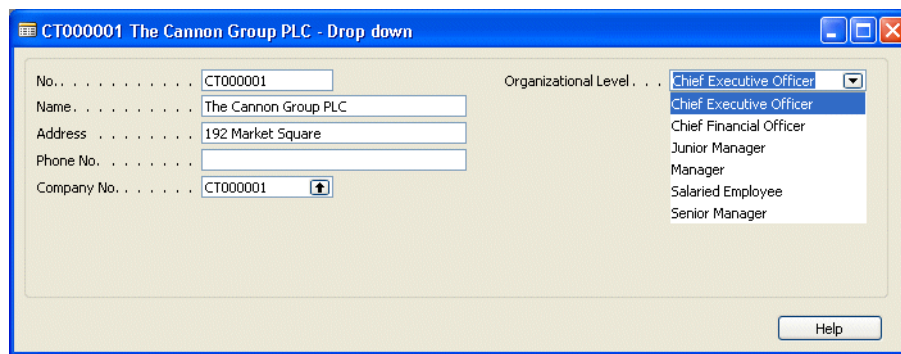
### Presenting a Set of Options

A recurring task in application programming is to present the user with a fixed set of options to choose from. For example, in a program where the user frequently has to enter the title of a contact, it could be a list of titles.

In C/SIDE, you can present these options in several ways. The following sections describe two different approaches.

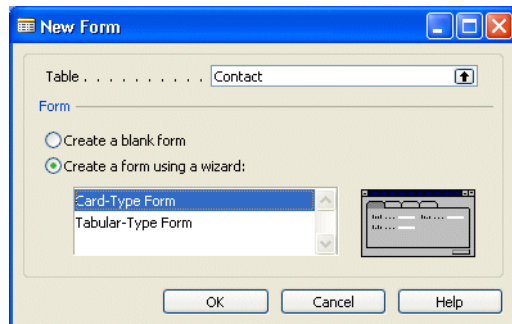
### Creating a Drop-Down List of Options

The list of options can be presented as a drop-down text box:



Before you create the drop-down list, you must create the form.

1. Open the Object Designer and click Form, New to open the **New Form** window:



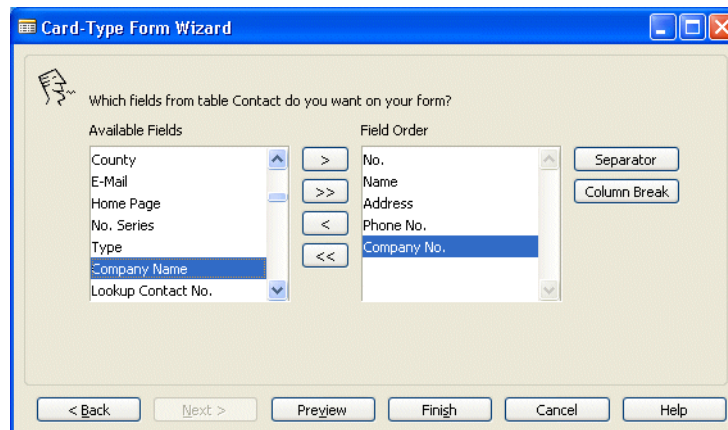
2. Create a Card-Type Form based on the **Contact** table and click OK.



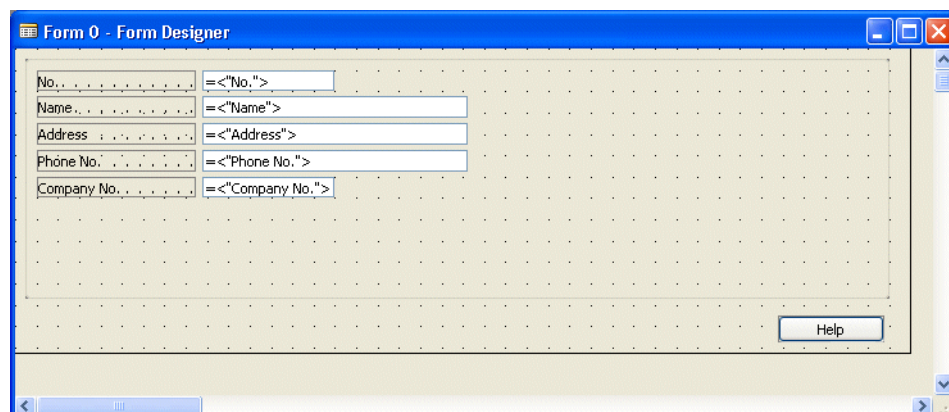


3 In the **Card-TypeForm Wizard** window click the **No, I want a plain form** option and click Next.

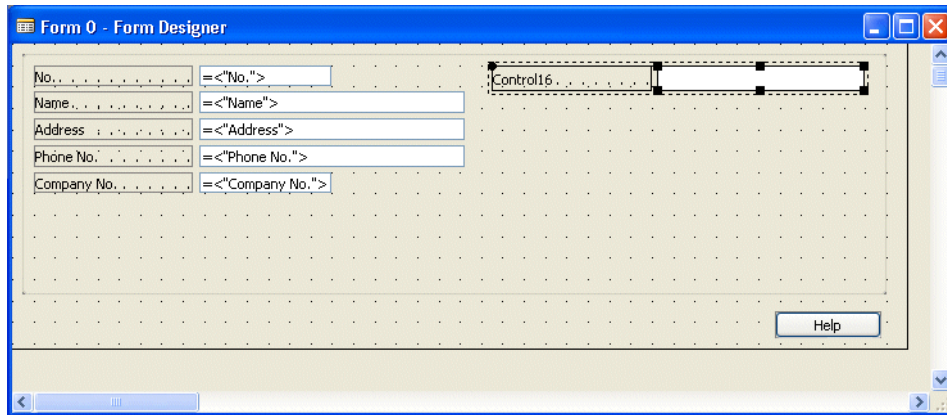
4 Add the following fields to the form:



5 Click finish and in the **Form Designer** window, expand the form and the container control.

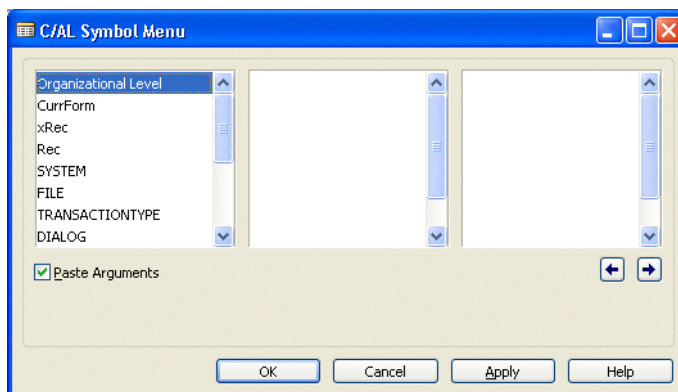


6 Add a text box and a label:



Now you have created the basic form. The next step is to create the drop-down option list.

- 1 Select the text box that you just created and open the **Properties** window (SHIFT+F4).
- 2 In the **Caption** field enter *Organizational Level*.  
 The field or variable that is the SourceExpr of the control must be of the Option data type, and the options must have been entered as the *OptionString* property of the field or variable. In this example you will use a variable.
- 3 Click View, C/AL Globals to Open the **C/AL Globals** window.
- 4 Create a variable called *Organizational Level* of data type Option and open the **Properties** window (SHIFT+F4) for this variable.
- 5 In the **Value** field *OptionString* property, enter the following list: Chief Executive Officer, Chief Financial Officer, Junior Manager, Manager, Salaried Employee, Senior Manager. These are the values that are currently available in the the **Organizational Levels** table.
- 6 Open the **Properties** window (SHIFT+F4) of the text box and bind it to the variable by clicking the AssistButton ... in the **Value** field of the *SourceExpr* property to open the **C/AL Symbol Menu** window:



- 7 Make sure that the **Paste Arguments** option is active, select Organizational Level and click OK.
- 8 Save and compile the form.

When you run the form, you can open the list by clicking the AssistButton ▾ in the text box.

If the option text box is built on a field in the table that the form is based on, you can use the **Field Menu** window to insert the text box.

#### Note

You can only enter options that have been defined in the **Properties** window of the field or variable. The first option is displayed in the text box. If the *OptionString* property has a blank as the first option, the text box is blank. This does not mean that you can enter options that are not in the *OptionString*.

In the *OptionString* property of the control, you can select a subset of the options already defined for the field – you cannot add options.

### Creating an Option Button Group

Another way of presenting a set of options is as an option button group. The functionality is the same as that provided by a drop-down list, but the visual presentation is, of course, quite different. An option button group looks like this:

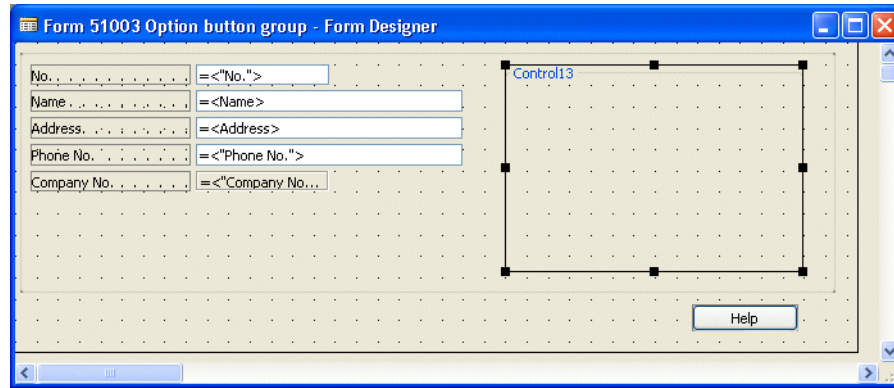


The advantage of using an option button group is that the user of the application can see all the available options and the currently chosen option at a glance. The disadvantage is that an option button group takes up more space on the form than a drop-down text box does.

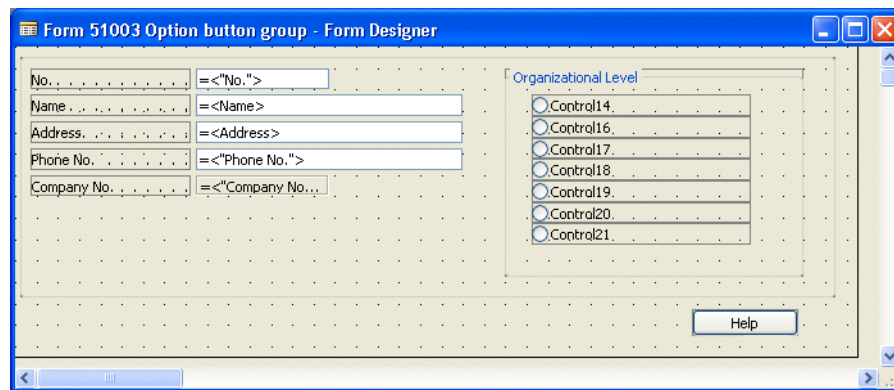
In this example, you add an option button group to the form that you created in the previous example.

- 1 Open the form you created earlier in the Form Designer.
- 2 Delete the the text box with the drop down list.

- Open the toolbox and add a frame to the form:



- Open the **Properties** window and enter *Organizational Level* as the Caption.
- Set the *BorderStyle* property to BumpUp.
- In the frame that you have just inserted add six control buttons, one for each of the options available in the the **Organizational Levels** table:



- In the **Properties** window of each option button, enter "*Organizational Level Code*" as the value in the **SourceExpr** field. Remember to enter the quotation marks.
- In the **Caption** field, enter the six values that are currently available in the **Organizational Level Code** table:

Code	Description
CEO	Chief Executive Officer
CFO	Chief Financial Officer
J-MANA	Junior Manager
MANA	Manager
SALEMP	Salaried Employee
SENMAN	Senior Manager

The screenshot shows a table window titled 'Organizational Level - Table'. It contains a list of organizational levels with their corresponding codes and descriptions. A 'Help' button is visible at the bottom.

- In the *OptionValue* property for each button, enter the corresponding value from the **Organizational Level Code** table.

10 Save and compile the form.

The final form should now look something like the picture shown at the beginning of this section.

The *OptionValue* property of each button has the same value as its caption.

Because the option buttons have the same source expression, only one of them can be chosen at a time. When you choose an option by clicking on the button, any previously-chosen button will be marked as *not chosen*.

### Using a Check Box to Display Booleans

A check box control is a handy way of displaying data of type Boolean. In a text box, boolean values will be shown as Yes and No. In a check box, Yes is displayed as a check mark, while No is displayed as a blank.

The screenshot shows a window titled "0 - Microsoft Dynamics NAV". Inside the window, there is a form with the following fields and values:

Name . . . . .	John Doe	Company . . . . .	ACME Sales Ltd.
Address . . . . .	17 Riverside Drive	E-mail . . . . .	jd@acme.com
City . . . . .	London	Birthday . . . . .	12-11-73
		Title . . . . .	Manager
		Active . . . . .	<input checked="" type="checkbox"/>

A "Help" button is located at the bottom right of the form.

To add a check box:

- 1 Open the form in the Form Designer.
- 2 If the check box will have a direct relationship to a table field, select the field in the Field Menu. Otherwise proceed to create an unbound check box.
- 3 If you want a label attached to the check box, click the Add Label tool (check boxes do not by default have labels).
- 4 Choose the Check Box tool; then click in the design area to create the check box (if you have selected a field of type Boolean from the Field Menu, you only have to click in the design area).
- 5 If the check box is unbound, bind it to the variable now by entering the name of the variable as the SourceExpr of the text box.

## Creating and Using Command Buttons

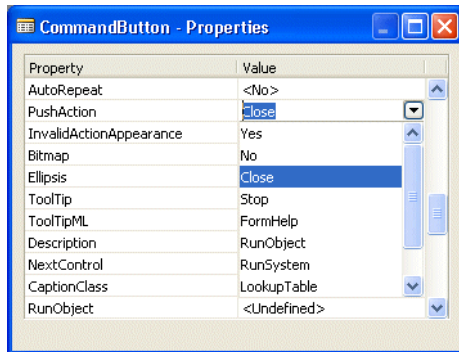
Command buttons are useful for a number of purposes. If you have used a wizard to create forms, you will have noticed that a Help button has been added to all forms. Other common uses are Yes and No buttons in contexts where the user must decide whether a certain task will be performed or not. Still another use is to launch another form, or even another program.

To add a command button:

- 1 Open the form in the Form Designer.
- 2 Select the Command Button tool and click in the design area to add the command button.

This will create the command button. The next step is to define the action associated with the button.

- 3 Open the **Properties** window for the command button. The *PushAction* property specifies what happens when the command button is pushed.
- 4 Open the drop-down list in the *PushAction* property value field. You will see this list of possible actions:



- 5 A common action would be to run another form. To make this command button run another form select RunObject in the drop-down list.
- 6 In the *RunObject* property, open the look-up table of system objects, and choose the object you want to run when the command button is pushed.

**Note**

Not all settings of the *PushAction* property require additional information. Some do, such as RunSystem, while others, such as Yes or No, do not.

While this method of adding an action to a command button is easy to use, it does have some limitations. For example, you cannot pass parameters. A more powerful method is to use the *OnPush* trigger for the button. Triggers are explored in Chapter 11, "Extending the Functionality of Your Forms".

## Containing Controls Within a Frame

A frame is used for containing other controls. When controls are contained in a frame, you can perform some operations on these controls as a whole: during design, they are moved when the frame is moved, with their relative positions intact; if the frame is invisible, all the contained controls are invisible too.

A frame can have a border that can be raised or sunken. This feature can be used to distinguish a group of controls (such as a group of option buttons) visually.



To create a frame with contained controls:

- 1 Open the form in the Form Designer.
- 2 Open the Toolbox and select the Frame tool.
- 3 Click and drag in the design area to create a frame.
- 4 Create the controls you want to be contained by the frame in the usual way, placing them inside the frame just as you would add them to the form.
- 5 Set the properties of the frame to suit your purpose.

A common change is to prevent the caption form being displayed by setting the *ShowCaption* property to No. By default, the border style is Raised. If the purpose of the frame is not to distinguish the contained controls, you can set the *Border* property to No, meaning that no border will be displayed.

When you drag an existing control inside a frame and drop it, it is contained by the frame. When a control is dragged outside a frame, it is no longer contained by the frame.

If a container is resized and overlaps some existing controls completely, these controls become contained.

If a frame is deleted, all the controls that it contained are deleted. For more information about resizing controls, see "Sizing and Resizing Controls" on page 163.

## Adding Shapes and Pictures

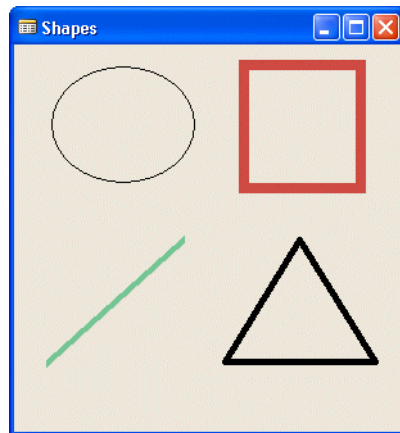
You can add graphical elements such as shapes and bitmap pictures to forms. These can be used to emphasize information, by adding a shape that makes some controls stand out – or for purely decorative purposes.

## Using Shapes

The *ShapeStyle* property lets you choose from a number of shapes: rectangle, rounded rectangle, oval, among others. You can adjust the width and color of the lines by changing the *BorderWidth* and *BorderColor* properties.

To create a shape:

- 1 Open the form in the Form Designer.
- 2 Open the Toolbox and select the Shape tool.
- 3 Click and drag in the design area to specify the size of the shape that you want to add. You can adjust it again later.
- 4 In the **Properties** window for the shape, click *ShapeStyle* and select a shape from the drop down list.
- 5 Use the *BorderWidth* and *BorderColor* properties to further refine the design of the shape that you have added to the form.



## Adding a Static Picture as an Image

The simple way to add a bitmap picture to a form is to add it as a control of type image.

To add an image:

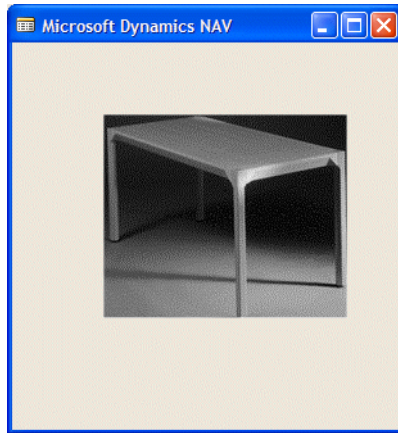
- 1 Open the form in the Form Designer.
- 2 Open the Toolbox and select the Image tool.
- 3 Click and drag in the design area to create an image control.
- 4 Open the **Properties** window of the image control, and enter the path and name of the bitmap file in the value field of the *Bitmap* property.

The bitmap is actually imported, so you don't need external files for your application. But if you change the bitmap during development, you must update the imported copy.

- 5 To update the bitmap, open the **Properties** window, select the *Bitmap* property Value field and press F2.



This causes the field to be reevaluated, and forces the bitmap to be imported again. The size of the bitmap can not exceed 32 Kb.



### Adding a Data Dependent Picture as a Picture Box

Adding a bitmap in a picture box control instead of in an image control gives you some more advanced possibilities. While the image control is static, the picture box control is dynamic. If you create a list of bitmaps, a bitmap from this list can be chosen at run time (the total size of all the bitmaps in the list can be 32 Kb).

One of the other advantages of a picture box is that it can display pictures that are stored in BLOB fields. A BLOB field can have a size of up to 2 GB.

To add a picture box:

- 1 Open the form in the Form Designer.
- 2 Open the Toolbox and select the Picture Box tool.
- 3 Click and drag in the design area to create a picture box control.
- 4 Open the **Properties** window for the picture box control.

To create a list of bitmaps from which one can be selected at run time for display, enter a comma-separated list of the path and names of the bitmaps that you want to use.

Remember that the size of the entire list of bitmaps must not exceed 32 Kb.

The system provides a series of standard bitmaps that can be chosen by entering a number between 1 and 53 – see the Bitmap entry in the *C/SIDE Reference Guide* online Help for details.

- 5 The value of SourceExpr determines which bitmap is chosen by the system: the first in the list has number 1, and so forth. If the SourceExpr calculates a value outside the range of bitmaps, no bitmap is displayed. We will explain how to enter C/AL expressions into the SourceExpr field later.

To display a picture stored in a BLOB field, enter the field name of the BLOB field as the SourceExpr. Do not enter a *BitmapList* property.

### Pictures on Command, Menu and Option Buttons and in Check Boxes

Command buttons, menu buttons, option buttons and check boxes can all display a bitmap picture instead of – or in combination with – a caption.

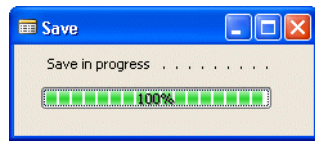
They all have a property called *Bitmap*. Here you can enter the path and name of a bitmap. The maximum size of the bitmap is 32 Kb, and it is actually imported, not referenced. This means that if you change the original bitmap, you will have to re-import it.

They also have a property called *BitmapPos*, where you can select the alignment of the bitmap within the control. This is especially useful when you combine a caption and a bitmap.

Using a bitmap on a button or a check box can sometimes make the interface more intuitive. A well-chosen picture may be easier to remember and identify than a label.

### Using an Indicator to Display Values

The indicator control allows you to display values graphically, as an analog gauge. A minimum and a maximum value must be defined, so that the system can calculate the scale of the indicator. If you do not provide these values, the system uses default values (see the online *C/SIDE Reference Guide* for details).



To create an indicator:

- 1 Open the form in the Form Designer.
- 2 Select the indicator tool, and click and drag in the design area to create the Indicator.
- 3 As the *SourceExpr* of the Indicator, enter the value you want to control the indicator.
- 4 Set the *MinValue* and *MaxValue* properties of the Indicator.

- 5 Set the *Percentage* property to choose whether or not the indicator will display percentages. If this property is Yes, the “%” symbol is displayed – otherwise, it is not. The gauge itself is the same when Percentage is No and when it is Yes. (The percentage is calculated as  $((\text{value of SourceExpr}) - \text{MinValue}) / (\text{MaxValue} - \text{MinValue}) * 100$ ).

### Creating a Tab Control

A tab control is useful when you are designing a form that is based on a table with many fields. Instead of creating a large form, cluttered with controls, you can group controls together on pages that the user can bring to the front by clicking the tabs.

You can use a form wizard to create a form with a tab control.

To create a tab control manually:

- 1 Open the form in the Form Designer.
- 2 Select the Tab Control tool and click and drag in the design area to create a tab control.
- 3 Open the **Properties** window for the tab control, and create the pages you need by entering a name for each page as a comma-separated list in the *PageNames* property. The names will be used as captions on the tabs.
- 4 The tabs are created while you are in the Form Designer. You can select pages by clicking the tabs.
- 5 Add controls on the pages. You can think of each page in a tab control as a frame and add controls as you would in a frame.

### Creating a Table Box

A table box is useful when you need to display many records from a database table at the same time. A table box contains columns and rows, and the user can move through the records, either by clicking navigation buttons (⬆️) or by using arrow keys and PAGEUP and PAGEDOWN. If the form is too narrow to display all columns, a horizontal scroll bar will automatically be added to the control.

You can use a form wizard to create a form with a table box.

To create a table box manually:

- 1 Open the form in the Form Designer. Select the table box tool, and click and drag in the design area to create a table box.
- 2 If the form is not related to a table, you can establish a relation now, by setting the *SourceTable* property of the form to the name of the table.

- 3 Open the Field Menu.
- 4 Select the fields you want in the table box from the Field Menu and click inside the table box. A column will be added for each field, and each row will display a record from the table. A label, derived in the same way as a label for any text box, is added as a column heading.

## **Chapter 11**

### **Extending the Functionality of Your Forms**

This chapter explores form design further, including sections on forms related to multiple tables, on creating menus and on writing C/AL code in triggers.

- Main Forms and Subforms
- Looking Up Values and Validating Entries
- Drilling Down to the Underlying Transactions
- Launching Another Form
- Designing Menu Buttons
- Form and Control Triggers
- Form Types and Characteristics

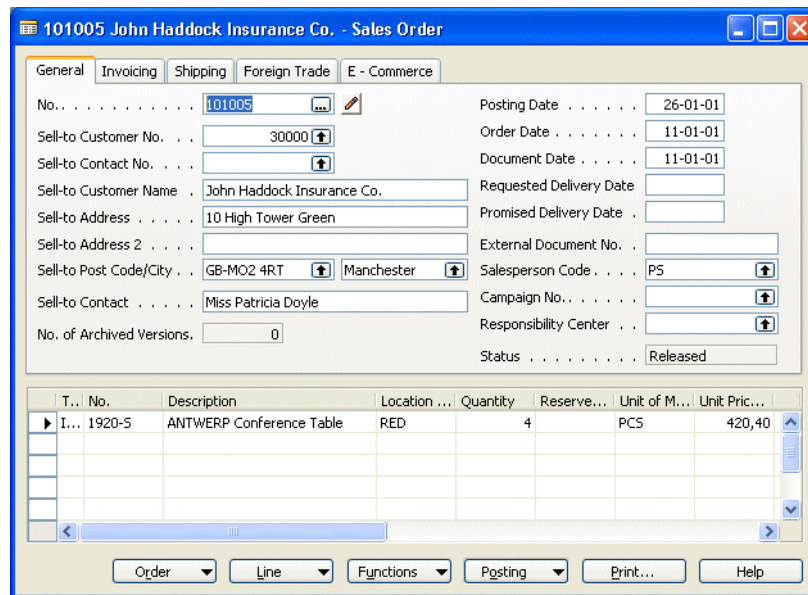
### 11.1 Main Forms and Subforms

A well-designed database does not store redundant information, but has a number of relationships between tables. The typical relationship is a one-to-many relationship.

For example, if you are designing an application that handles sales orders, there can be many items on one single sales order, but one specific item can only be part of one sales order. Some of the information on a sales order, such as the address of the customer, is per order, while other information, for example the item number, is per item. In a well-designed database, with no redundant information, this means that the information on a sales order is stored in two tables: one, a header table, with the general order information, the other, a lines table, with the information about each item. There is a one-to-many relationship between the tables.

However, users of the application need to view information from both tables at the same time.

The header information together with the lines could look like this:



Although this looks like a normal form, it is actually two forms.

The main form is the one side of the one-to-many relationship. In this example, it is based on the **Sales Order Header** table. The subform is the many side of the relationship and is based on the **Sales Order Line** table. When you select a sales order header in the main form, the subform is updated to display only sales order lines that are related to this sales order header. There is a link between the main form and the subform to keep the information synchronized.

## Designing the Main Form

There are no special procedures involved in designing the main form in a main form/subform relationship.

To create a subform, you add a subform control. The subform control establishes the link between the main form and the subform, but it is not itself a form. You can, however, display any form in the subform control.

If you are going to use an existing form as the subform, follow the procedure described here. If you are going to create a new form to use as the subform, it may be more convenient to create the subform first. This is described in the next subsection.

To create the main form in a main form/subform relationship:

- 1 Open the existing form in the Form Designer.
- 2 Open the Toolbox and select the Subform tool.
- 3 Click and drag in the design area to create the subform control.
- 4 Open the **Properties** window (SHIFT+F4) of the subform control.
- 5 Enter the name of the form that the subform is based on as the *SubFormID* property (or use the lookup button to select the form from a list).
- 6 Enter the expression that links the two tables (for example the field that is common for the tables) as the *SubFormLink* property. There is an assist-edit function available to help you (click the AssistButton ... to open the assist-edit window). Select the field name from the *many* side of the relationship (the subform table) as the Field. Then select FIELD as the Type of the relationship. Finally, select the field from the one side of the relation (the main form table) as the Value.
- 7 In the *SubFormView* property, you can specify the key, sort order and table filter to apply to the table when it is displayed in the subform. (You do not have to enter anything.)

### Example

In the *SubFormLink* property, apart from FIELD you can select other types of link. If you choose CONST, Value must be a constant expression that selects records where the Field matches this expression. If you choose FILTER, Value must be a filter expression, for example, 10|30..40.

## Designing the Subform

There are no special requirements for a subform. However, you must remember that the form is going to be used to display the many side of a one-to-many relationship, and that all forms are not suitable for this task.

Typically a subform is a tabular form, that is, a form with a table box. The table box should fill out the form completely, and the *HorzGlue* and *VertGlue* properties of both the table box on the subform and the subform control on the main form should be set to Both. This ensures that the subform and the table box are resized when the main form is resized.

## Hints and Advice

Although creating a form with a subform is no different than creating controls on forms in general, you may have to experiment before you find the best way to do it.

Here are some hints and advice to help you along:

- It can be difficult to get the sizing of the subform control on the main form and the size of the subform right. You should finish the design of the subform first. Get the values for the width and the height of the form from the **Properties** window of the subform. Then, in the main form, click and drag a subform control of any size. In the **Properties** window, insert the width and height of the subform as the width and height of the subform control.
- Generally, if the subform is a tabular form, it looks better if you let the table box completely fill out the form vertically. This ensures that there is no extra space around the table inside the subform control. Set the *HorzGlue* and *VertGlue* properties to Both.
- If the subform is a tabular form, you should set the size of the form so that it only displays a few records at a time. Then, in the main form, set the *VertGlue* and *HorzGlue* properties of the subform control to Both. You can resize the main form vertically and horizontally, and the subform is resized along with it so that either more or less records and fields are displayed.

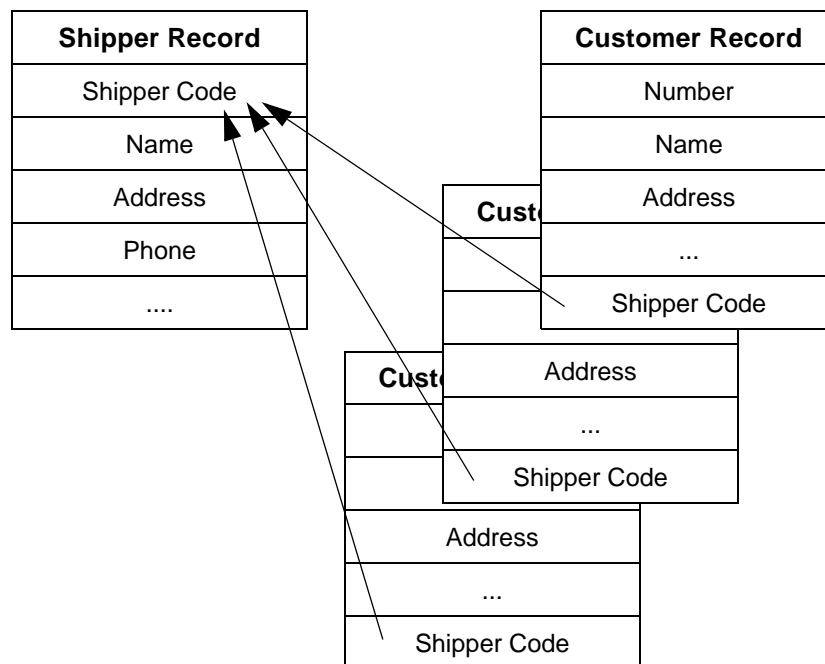


## 11.2 Looking Up Values and Validating Entries

The previous section described how to create a main form and a subform in order to display data from a one-to-many relationship. The main form was bound to the table on the *one* side of the relationship, and the subform was bound to the *many* side of the relationship.

Suppose instead, that you are designing a form that is bound to a table that contains information about customers. Some of this information is unique for each customer, but some of the information is not. The names and addresses of the customers are unique, but perhaps you want to store information about the shipper that is normally used for deliveries to each customer. There are only a few shippers, and it would be redundant and in violation of relational database design rules to store information such as the addresses of these shippers in the customer records.

Instead, you should create a **Shipper** table, and use a **Shipper Code** field to create a link between this table and the **Customer** table. You should only store the shipper code in each customer record in the **Customer** table and store all the other information about the shippers in the **Shipper** table.



This is the many side of a one-to-many relationship. Each customer can be associated with only one shipper, but a shipper can be associated with many different customers. A main form/subform structure is not applicable here. (Although it would be applicable if you were to design a form to display information about the shippers. The subform could then display a list of the customers that use each shipper.)

There are two things to consider when creating the **Customer** form (and table):

- Do you want an easy way to enter the shipper code?
- Do you want to validate the **Shipper Code** field in the **Customer** table against the **Shipper** table? That is, do you want the system to verify that the contents of the **Customer** table field are present in the **Shipper** table?

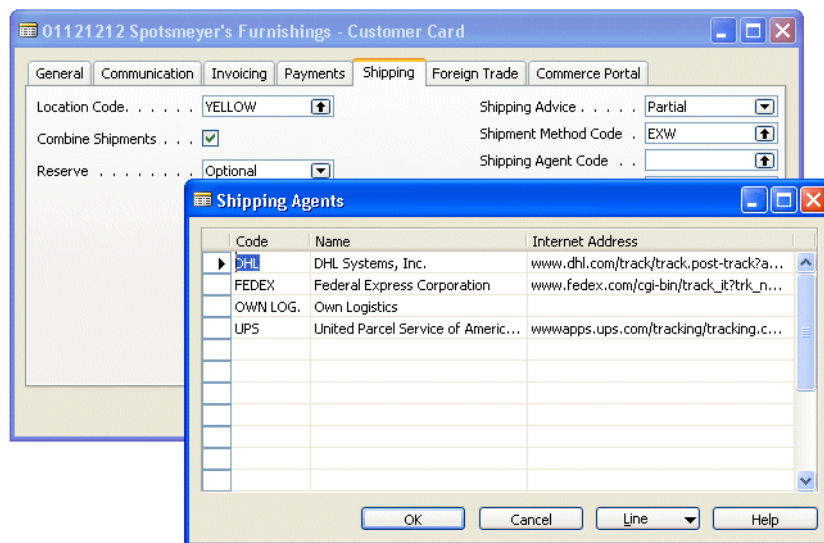
If you do not establish a relationship to the **Shipper** table, you must memorize the shipper codes. This means that you can easily enter a code that does not exist in the **Shipper** table. If the tables are related, the system provides a lookup function into the **Shipper** table, so that you can press F6 or click a lookup button  $\uparrow$  and select the shipper code from a list that displays the codes as well as other information such as name and address.

A control can be related to a field in another table in two different ways. The first is by defining the relationship at table level, as a property of the **Shipper Code** field in the **Customer** table. The second is by defining the relationship at form level, as a property of the text box that displays the shipper code on the **Customer** (main) form.

If you want to make certain that you do not enter non-existent shipper codes into the **Customer** table, you can set the system up so that it validates the entries against the **Shipper** table. The *ValidateTableRelation* property, either of the field (at table level) or of the text box (at form level), determines whether or not the values you enter must exist in the **Shipper** table.

Apart from simply ensuring that the entered codes exist in the **Shipper** table, you can create more advanced validation rules that check the entered codes against combinations of values of fields in both tables. For example, you can have the system check whether the shipper allotted to a customer operates in the customer's country/region. To do this, you have to create the validation rule by writing C/AL code in the *OnValidate* trigger of the control on the main form.

A form with a lookup on the **Shipping Agent Code** field looks like this when the lookup function has been activated:



## Defining the Table Relation

The relationship to a table can be defined in two different places - as part of the table description or as part of the form description. In both places the relationship is defined in the *TableRelation* property of the field or control. There is no functional difference between a table relationship defined at the table level and one defined at the form level. But there is a difference when you are designing an application. If the relationship is defined at the table level, all the text boxes in the forms that have a direct relationship to the field will have the lookup functionality – with no effort required from the person designing the forms. You can suppress this function by setting the *Lookup* property of the text box explicitly to No.

To define a table relationship:

- 1 Open the form in the Form Designer.
- 2 Open the **Properties** window (SHIFT+F4) of the field or control.
- 3 In the **Value** field of the *TableRelation* property, click the AssistButton ....
- 4 In the **Table Relation** window, enter the name of the table that you want to lookup in the **Table** field or choose it from the list that appears when you click the lookup button.
- 5 In the **Field** field, enter the name of the field in the table (or choose it from the list).

You can use the **Condition** and the **Table Filter** fields to create a more advanced relationship than this basic one.

By using the **Condition** field, for example, you can lookup different tables, depending upon the value of a field in the current table. Each condition line corresponds to a statement in an *if then...else if* sequence.

In the **Table Filter** field, you can set a filter on the lookup table.

## Validating Entries

Entries can easily be validated against the contents of a field in a related table. If you set the *ValidateTableRelation* property to Yes – either at field level or at control level – only entries that exist in the related table will be accepted.

If you need a more advanced validation, you can write C/AL code in the *OnValidate* trigger of either the control or the field.

## Using the Default Lookup or Writing Your Own?

If you want more control over the way a lookup functions than you can achieve by using conditions and filters, you can write C/AL code in the *OnLookup* trigger. This allows you to bypass the default lookup function completely and write your own.

The rules for determining which lookup function is performed are:

- A trigger at the form level takes precedence over one at the table level.
- Both of these take precedence over the system default action.

## Defining a Lookup Form

When you are using the system lookup function, you must define which form to use to display the results of the lookup. You can define the form in two ways:

- Each table can have a form based on it that is used for looking up into the table. This is done by setting the *LookupFormID* property of the table.
- The form can be defined by setting the *LookupFormID* property of the control for which the lookup is provided.

If both properties are set, the form that is defined as a control property is used.

If no lookup form is defined (either at table or form level), and although the text box contains a lookup button, a lookup is not performed when the button is clicked. If you are writing your own lookup function in the *OnLookup* trigger, you have to explicitly run a form by using the `RUNMODAL C/AL` function.

### Hint

.....

If you always design a basic tabular form for a table and enter this form as the *LookupForm* (and *DrillDownForm*) of the table, you will never forget to provide a lookup form. If you later decide that this form is not adequate for some lookups, you can add customized forms as control properties.

.....

## Permanent Assist

This is a control property. If set to Yes, the lookup button is permanently displayed; otherwise, it is displayed only when the control has the focus.

## Looking Up in the Current Table

By setting the *Lookup* property of a text box to Yes, you can lookup into the source table of the form. This makes it easy for you to select records. In effect, the lookup provides a list of all the records in the table and you can select a record from the list, and this record then becomes the current record.

A lookup form must be defined either at table or form level in the same way as it must be defined when the lookup is into another table. However, you cannot set conditions and filters as you can when the lookup is into another table. The default behavior is to display all the records in the table. If you need to change this, write your own lookup function in the *OnLookup* trigger.

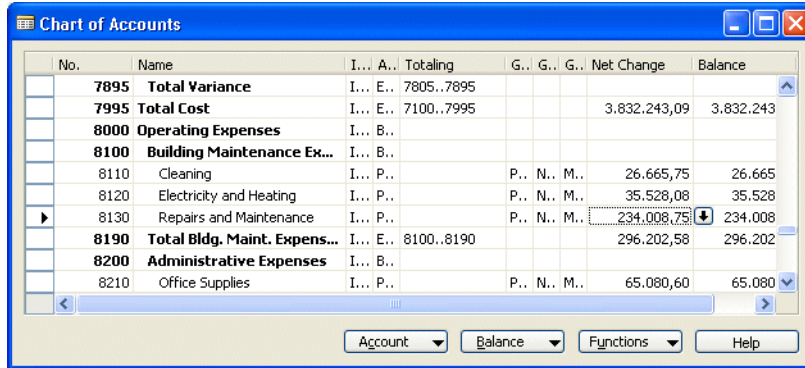
If a lookup into a related table is defined – regardless of how the relation is defined – setting the *Lookup* property to Yes is overruled. But if *Lookup* is explicitly set to No (as opposed to its default value <No>), no lookups, including to related tables, are performed.

You can provide the same functionality by using the *LookupTable* action (applicable to command buttons and menu items). In this way you can provide both types of lookup on the same form: lookups to related tables from text boxes, and lookups to the source table from command button actions.

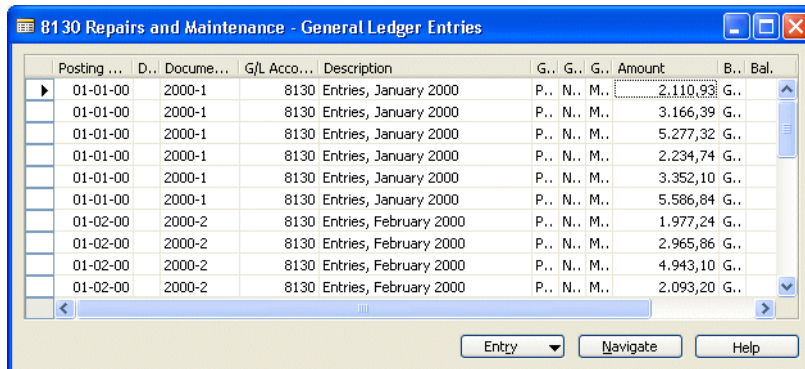
### 11.3 Drilling Down to the Underlying Transactions

FlowFields were introduced in the chapter "Table Fundamentals" on page 59. When a text box is based on a FlowField, a *drill-down* button (↕) is automatically attached to the text box. When you click this button (or press SHIFT F6), the transactions that the system used to calculate the value of the FlowField are displayed.

In the Chart of Accounts, you can execute a drill-down function in the **Net Change** field - a FlowField that summarizes transactions in this account:



The **General Ledger Entries** window opens and displays the result of this drill-down – a detailed list of the transactions:



The drill-down facility is provided whenever a text box is directly related to a FlowField – you do not have to do anything special when designing the form except to make certain that a DrillDownFormID is defined, either at table level or at form level. For more information about defining the DrillDownFormID, see "Defining a Lookup Form" on page 192.

Drill-downs resemble lookups in many ways, and allow you to do most of the things that you can do with lookups. One exception is that drill-downs only apply to FlowFields, which have to be defined when the table is designed.

#### Customizing Drill-downs

You can customize a drill-down in these ways:

- You can disable the drill-down altogether by setting the *DrillDown* property of the text box explicitly to No.

- Text boxes based on the same FlowField will have different drill-down forms if you define separate DrillDownFormIDs at form level.
- You can decide whether the drill-down button should be displayed permanently or only when the text box has the focus.

This is done by setting the *PermanentAssist* property of the text box. Yes means that the button is always displayed, and No means that the button is only displayed when the text box has the focus.

- You can change the drill-down behavior by writing C/AL code in the *OnDrillDown* trigger of the table field or in the control.

In this case you have to run a form explicitly from your trigger code.

## 11.4 Launching Another Form

When you use a lookup function to select values in a related table, you typically only display a subset of the fields in the lookup table.

In some situations, however, it would be convenient if you were able to update more fields in the related lookup table than are displayed on the lookup form without having to close the current form.

If you are taking orders by phone, it is convenient when you can use a lookup function on the sales order form to find the customer numbers as the customers call in. However, a customer could call in to order something and inform you that they have changed their address. It is both time-consuming and annoying to have to close the sales order form, select the customer form, find the customer, change the address, and then return to the sales order form to start entering the order again.

A better solution would be to launch the customer form directly from the sales order form, automatically select the appropriate customer record, update and close the customer form, and continue filling out the sales order form.

In order to launch another form, you can add a control that has a *PushAction* property and run the customer form with parameters that select the correct record from the **Customer** table whenever you "press" the control. You can use command buttons, menu items, check boxes or option buttons.

### Adding a Command Button

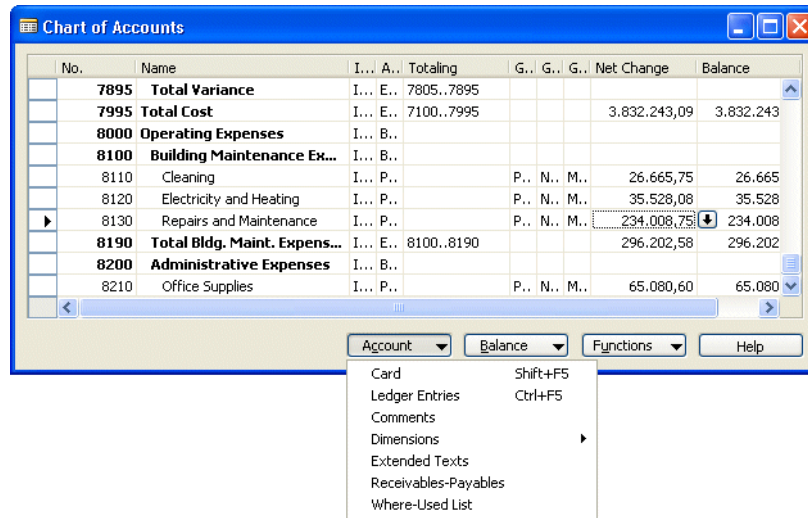
To add a command button that launches a form:

- 1 Add a command button (the procedure is described in "Designing Forms" on page 151).
- 2 Set the *PushAction* property of the command button to RunObject.
- 3 Set the *RunObject* property of the command button to the name of the form you want to launch. Because you can use RunObject to run any object, you must specify the type of object (Form, Codeunit, and so on). You can choose the object from the lookup list provided (in this case, the type of the object is inserted automatically).
- 4 Set the *RunFormLink* property to establish the link to the form that you want to launch. Use the assist-edit button to create the expression. First, in the **Field** field select a field from the table that the form you want to launch is based on. In the **Type** field, select FIELD as the type of the relationship. Finally, in the **Value** field, select the field in the table that underlies the current form and remember that this field must match the field that you selected from the other table.

## 11.5 Designing Menu Buttons

Although command buttons are a convenient way of adding functionality to forms, having too many buttons on a form makes it look cluttered and detracts from both its usability and visual design. If you have to add many command buttons to a form, you should consider creating a few menu buttons instead.

When you click a menu button, a menu opens:



Each line in a menu is called a *menu item*. A menu item can:

- Perform an action when it is clicked. This can be an action from the same set of actions as command buttons or it can be an action written in C/AL. Menu items have *OnPush* triggers like command buttons.
- Contain a submenu that is opened when the line is clicked.
- Be a separator – a line used for grouping items in a menu together.

### Adding a Menu Button to a Form

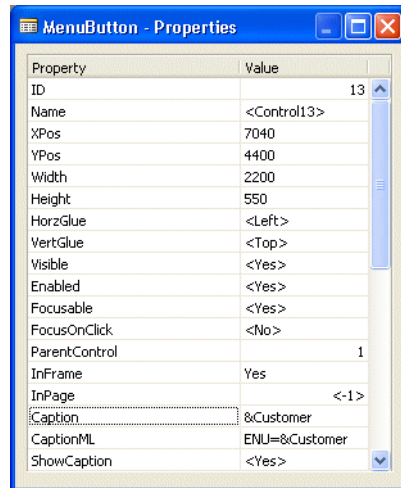
A menu is created in two steps. First you add a menu button to your form. This is exactly the same procedure as adding a command button. Then you open the Menu Designer for the menu button and create the menu items.

To add a menu button:

- 1 Open the form in the Form Designer.
- 2 Open the Toolbox and select the Menu Button tool.



- 3 Select the menu button and open the **Properties** window (SHIFT+F4) for the menu button. As a menu button does not have a relation to data – field or variable – the **Name** and **Caption** properties are set to default values (such as Control7). Change the caption to an appropriate text. If the text contains an ampersand (&), the system interprets the following letter as an access key.

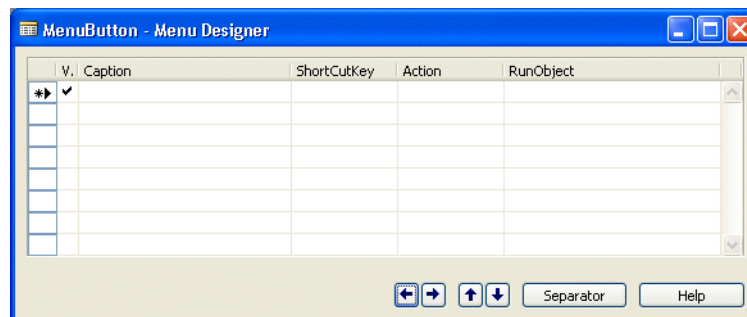


### Adding a Menu Item to a Menu

After you have created and modified a menu button as described in the previous section, you can add the menu items to the menu.

To add a menu item:

- 1 Select the button and click View, Menu Items to open the **Menu Designer**.



- 2 The first field, Visible, is by default set to Yes. Leave it like this.
- 3 Add the menu items by filling out the **Caption** field. If you do not create a shortcut key (by embedding an ampersand in the Caption text), the system automatically uses the first letter of each caption as a shortcut key. If you add menu items that start with the same letter, set the shortcuts yourself to avoid overloading.
- 4 You can also define a shortcut key by entering the name of the key in the **ShortCutKey** field. The different ways to enter Keys are:

Key	Entered as
Function keys	F1, F2, F3, ...

Key	Entered as
CONTROL, ALT, SHIFT	CTRL, ALT, SHIFT
Other keys	A, B, C, ... (these keys must be part of a key combination with CTRL or ALT).
Key combinations	For example: CTRL+A, SHIFT+F2

A shortcut key is active as long as the focus is on the form that the menu button is a child of. Avoid accidentally overloading the key combinations so that they perform different actions when different forms have the focus – this causes confusion. You should also try to avoid using shortcut keys that the system already uses.

- In the **Action** field, enter the action that you want the menu item to initiate. You can use the drop-down list to select the action from a list. Alternatively, you can write some C/AL code in the *OnPush* trigger of the menu item.

If you select RunObject you can choose to run either a form, a report or a codeunit.

- In the **RunObject** field, click the lookup button to select the object that you want the menu item to run. For other parameterized actions (such as, RunSystem) you must set the parameters in the **Properties** window of the menu item. This is explained in the next section.

### Adding Other Menu Items

In addition to items that perform actions, menus can contain separators and items that open submenus.

#### Separators

A separator is a horizontal line in a menu that cannot be selected or perform any action. Separators help you group items on a menu. To add a separator to a menu, select the line after which you want to insert it and click the Separator button in the Menu Designer.

#### Submenus and menu levels

Menu items can be nested, that is, when you click a menu item, another menu can open.

You define submenus in the Menu Designer by indenting one or more menu items. To indent a submenu, select it and click the right-arrow button. The indented item becomes a menu item on a submenu.

In the **Properties** window of a menu item and you see that when the menu item is created the *MenuLevel* property is set to the default value of zero. As the item are indented, the MenuLevel is set to 1, 2, 3 and so on – one level for each click on the indentation button (you can cancel indentation by clicking the left-arrow button – each click cancels one level of indentation).

There are a few logical rules that you must follow when creating submenus:

- At least one item in each menu must have MenuLevel 0 (zero).
- Each MenuLevel can only be one level higher than the preceding level in the list.
- If a higher MenuLevel follows a lower one (for example, 1 follows 0), the menu item with the MenuLevel 0 opens the submenu, and the item with MenuLevel 1 becomes an item on this submenu. A menu item that opens a submenu cannot have any action associated with it.
- There can be up to 10 menu levels (numbered from 0 to 9).
- If the MenuLevel reverts to lower numbers (with a lesser indentation), the menu items become items in a previous menu at the level indicated by the MenuLevel.
- Separators cannot open submenus. Separators can only separate items that are on the same level. This means that you cannot put a separator as the first, last or only item on a menu or submenu.

### Displaying Check Marks on Menu Items

When menu items that act as toggles, the on/off state is indicated by the presence or absence of a check mark next to the menu item.

The *SourceExpr* property of a menu item controls whether a check mark is displayed. Initially, the *SourceExpr* property is undefined. You can define it with a valid C/AL expression that evaluates to a Boolean. The check mark appears when the value is TRUE.

This feature is used in C/SIDE itself. For example, on the Format menu, the Snap to Grid menu item can either be on or off. When it is on, the check mark is displayed.

## 11.6 Form and Control Triggers

Although the system interprets and acts upon many events in a predefined way, certain actions – such as opening a form or clicking a command button – make the system execute a user-defined C/AL function. The event *triggers* the function.

You typically use triggers for advanced validation, to initialize variables in a non-trivial way, or perhaps to format text boxes according to the value of a field or control. In short, you use triggers whenever the system's default behavior does not suit your purpose.

### Overview of Form Triggers

The following triggers apply to forms in C/SIDE:

Form trigger name	Executed when...
OnInit	the form is loaded, but before the controls are available.
OnOpenForm	the form is initialized (the controls are available).
OnQueryCloseForm	the form is about to close, but before <i>OnCloseForm</i> . If this trigger returns FALSE, the form is not closed. The intended use is to ask the user if they really want to close the form.
OnCloseForm	the form is about to close, and after <i>OnQueryCloseForm</i> .
OnActivateForm	the form is activated, that is, when the form becomes the active window.
OnDeactivateForm	the form ceases being the active window.
OnFindRecord	the form is opened and a record is retrieved – and also when the user chooses to go to the first or the last record.
OnNextRecord	the system determines how to select the next record, for example after a user pressed PAGEDOWN (in a card form).
OnAfterGetRecord	a record has been retrieved but not yet displayed.
OnAfterGetCurrRecord	the current record is retrieved. In a table box, <i>OnAfterGetRecord</i> is called for all the records displayed, while this trigger is called for the current record.
OnBeforePutRecord	a record is about to be saved.
OnNewRecord	a new record has been initialized but not yet displayed.
OnInsertRecord	a new record is about to be inserted in the table.
OnModifyRecord	a record is about to be modified in the table.
OnDeleteRecord	a record is about to be deleted from the table.
OnTimer	after the <i>OnOpenForm</i> trigger and after the time specified in the <i>TimeInterval</i> property of the form has elapsed.
OnCreatehyperlink	you create a URL to a form so that you can send it by e-mail or paste it to your desktop.
OnHyperlink	you click a hyperlink and after the <i>OnInit</i> trigger is executed.

The table only sketches out the main purpose of each trigger. For more extensive descriptions and details about these and other triggers, see the *C/SIDE Reference Guide* online Help.

#### Note

.....

The three triggers *OnInsertRecord*, *OnModifyRecord*, and *OnDeleteRecord* correspond to triggers at table level. If you use triggers at both form and table level, the triggers at form level are executed first.

.....

## Overview of Control Triggers

Controls have a varying number of triggers depending on the type of control. Static controls and container controls have no triggers at all, while text boxes have a full range of triggers.

Other data controls and data container controls have a subset of the possible triggers. Controls that can be clicked - such as command buttons, menu items and check boxes - have a special trigger to handle this. The following table outlines the full range of triggers. The column on the right indicates the controls that the trigger applies to.

Control trigger name	Executed when...	Controls
OnActivate	the control is activated.	1,2,3,4,5,6,7,8
OnDeactivate	the control is deactivated.	1,2,3,4,5,6,7,8
OnFormat	the control is about to be updated.	5
OnBeforeInput	the control is selected for input and before any input is actually entered.	5
OnInputChange	the user is entering data. This trigger is repeatedly executed, after each keystroke.	5
OnAfterInput	the user finishes input.	5
OnPush	the control is pushed.	1,3,4,6,7,9
OnValidate	the control loses focus.	3,4,5,6,7
OnAfterValidate	the value entered has been validated.	3,4,5,6,7
OnLookup	the user requests a lookup (by clicking a lookup button or pressing F6).	5
OnDrillDown	the user requests a drill-down (by clicking a drill-down button or pressing SHIFT F6).	5
OnAssistEdit	the user requests assist-edit (by clicking an assist-edit button or pressing SHIFT F2).	5

Controls are

1 - command button, 2 - menu button, 3 - check box, 4 - option button, 5 - text box, 6 - picture box, 7 - indicator, 8 - subform, 9 - menu item

For more extensive descriptions of these triggers, see the *C/SIDE Reference Guide* online Help.

**Note**

*OnValidate* is also a field trigger at the table level. If both triggers (field and control) are defined, the field trigger is executed before the control trigger (and the system default validation before anything else).

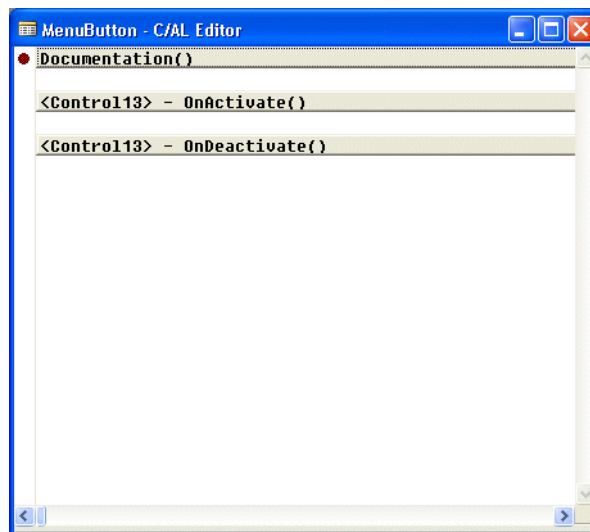
*OnLookup* is also a field trigger at the table level. The flow is different here: when a lookup is requested, the system executes the control lookup trigger, if it is defined, in place of the field lookup or system default. If no control lookup trigger is defined, a field lookup trigger (if defined) replaces the system default lookup function.

### How to Define and Modify Form and Control Triggers

You can define a function that is triggered by a form or control event or modify an existing function.

To define or modify a form or control trigger:

- 1 Open the form in the **Form Designer**.
- 2 Select the form or the control (or menu item) and click View, C/AL Code (F9) to open the **C/AL Editor**:



- 3 In the editor, you have access to the triggers that apply to the object that you selected. Enter C/AL code in the triggers you want to use, or modify the existing triggers.
- 4 Click Tools, Compile (F11) to test-compile the form including the code that you have added or modified.

If you are not familiar with the C/AL programming language, you should read Part 6, "Codeunits" on page 277.

## 11.7 Form Types and Characteristics

This section describes the types of windows that Dynamics NAV contains. It then describes the different types of form that you can create in your applications, gives some examples and explains the characteristics of each.

Dynamics NAV contains five basic kinds of window:

**Dialogs** These are simple windows that display information and prompt you for a response. You can respond by pressing a button, such as OK, Cancel, Yes, No and so on. These are generated by C/AL functions such as `ERROR`, `MESSAGE`, `TESTFIELD`, `CONFIRM` and so on, or are created by the programmer using a variable of the Dialog data type. Dialogs are not associated with forms.

**Request Panels** These windows are found only within reports and are not associated with forms.

**Unbound Forms** These are forms that are not associated with a table. Like Request Panels they are used when you must answer a few questions before the system can continue processing.

**One-Record Forms** These are forms that are associated with a table. They let you see and possibly edit only one record from one table at a time. A card form is a typical example of a one-record form.

**Multi-Record Forms** These are forms that are associated with a table. They let you see multiple records from one table at once and possibly edit them. Examples of these include Tabular Forms, TrendScape Forms and Matrix Forms.

The forms that are associated with tables (the last two listed) are the building blocks of Dynamics NAV.

Although C/SIDE lets you create many kinds of form that look and operate in many different ways, Dynamics NAV only uses a few of these possibilities. This gives the application a consistent look and feel.

It is strongly recommended that you follow this policy for all the modifications that you make to Dynamics NAV and for any application that interfaces with it. All the following descriptions assume that you follow the Dynamics NAV standards.

### Types of Forms and Examples

There are several types of standard form in Dynamics NAV. The following table lists the most common types of form and contains some examples of each type. A more detailed description of each type appears in the following sections..

Type	Single Record/Multi-Record	Examples
Card Form	Single Record	Customer Card, Vendor Card, Item Card
Statistics Form	Single Record	Customer Statistics, Vendor Statistics, Employee Statistics

Type	Single Record/Multi-Record	Examples
Tabular Form	Multi-Record	Currencies, Payment Terms
List Form	Multi-Record	Customer List, Item List, Item Ledger Entries
Worksheet Form	Multi-Record	General Journal, Cash Receipts Journal, Item Transfer Journal
Header Form, Line Form	Single Record and Multi-Record	Sales Invoice, Posted Purchase Credit Memo, Finance Charge Memo
Setup Form	Single Record	General Ledger Setup, Company Information, Sales & Receivables Setup

When you understand these standards you will be able to understand the different areas of the application after just a superficial look at the objects.

### Card Forms

**Card Form Characteristics** A card form lets you view and edit one record in a table at a time. A card form is used when there are too many fields, to view them all conveniently on only one line. Card forms always have tabs (like index tabs) across the top, which you can select to view different groups of fields.

Even if there are only a few fields, there is at least one **General** tab. The **General** tab is always first.

The table's primary key field is always the first field in the **General** tab. Tables that use card forms only have one field in the Primary Key.

**Naming Card Forms** Card forms are named after the table they are associated with, followed by the word "Card". For example, the card form associated with the **Customer** table is called the **Customer Card**. Card forms also have at least one menu button at the bottom of the frame and this button has the same name as the table that the card is based on and gives you access to related information.

### Statistics Forms

**Statistics Form Characteristics** A statistics form is a one-record form that lets you view but not edit information. It usually contains FlowFields, which let you drill down to get to more information. Usually, a statistics form also contains calculated or derived information contained in variables, which cannot be drilled down.

Statistics forms can also contain tabs that help organize the information.

The table's primary key is only displayed in the form's title bar.

**Naming Statistics Forms** Statistics forms are named after the table with which they are associated, followed by the word "Statistics". For example, the statistics form associated with the **Customer** table is called **Customer Statistics**.



Entry statistics forms are a special version of the statistics form. They are named after the table they are associated with, followed by the words "Entry Statistics". For example, the entry statistics form associated with the **Customer** table is called **Customer Entry Statistics**.

### Tabular Forms

**Tabular Form Characteristics** A tabular form is a multi-record form that lets you view multiple records from a table and edit them. Each record is displayed as a single row in the tabular form and each field is displayed as a column forming a spreadsheet-like table within the form itself.

The primary key of the associated table is displayed in the left-most column. If there are multiple fields in the primary key, they are displayed in order of importance in the columns, starting from the left.

**Naming Tabular Forms** Tabular forms are named after the table they are associated with – only in the plural. For example, the tabular form associated with the **Country/Region** table is called **Countries/Region**.

In the case of associated tables that have multiple fields in the primary key, the name can be different. For example, the tabular form associated with the **General Posting Setup** table is called **General Posting Setup**.

### List Forms

**List Form Characteristics** This is a multi-record form that lets you view multiple records from a table at once, but does not allow you to edit them. It has the same rows-and-columns look as the tabular form.

The primary key fields of the associated table are displayed in the left column.

**Naming List Forms** These are named after the table they are associated with, followed by the word "List". For example, the list form associated with the **Customer** table is called **Customer List**.

**The "Specialized" Ledger Form** A more specialized version of the list form is the Ledger Form. These are used only for Ledger Entry tables. They differ from ordinary list forms, in that although you cannot insert or delete records, you can sometimes edit a small number of the fields. Also, the primary key is always an integer named "Entry No." and is displayed in the right hand column rather than the left hand column.

The ledger form is given the plural of the name of the associated table. For example, the ledger form associated with the **Customer Ledger Entry** table is called **Customer Ledger Entries**.

### Worksheet Forms

**Worksheet Form Characteristics** A worksheet form is a specialized version of the tabular form. It is a multi-record form that lets you view multiple records from a table and edit them. The difference is that when you insert a new record, the record does not jump to another position within the form, but instead stays in the same order as you inserted it.

This is done using the *AutoSplitKey* property of the form, combined with an integer as the last field in the table's primary key.

The primary key fields of the associated table are not displayed on the worksheet form.

**Naming Worksheet Forms** Worksheet forms are named to reflect the purpose of the associated table. One example is a Journal table. In this case, the name of the worksheet form will end with the word "Journal".

## Header/Line Forms

**Header/Line Form Characteristics** Many forms within Dynamics NAV have the characteristics of both a card form and a tabular form, for example the **Sales Invoice** form.

The fields that are common to the entire invoice are located on a card-like form with tabs, showing one invoice at a time. However, the invoice lines display in a table-like section of the form, where multiple invoice lines (from the same invoice) can be viewed at the same time and edited. These are called "Header/Line" forms.

Header/Line forms are, in fact, two separate forms that are associated with two different tables. The main form is a card form that is associated with one table. The main form also contains a sub-form control that displays a worksheet form that is associated with a different table, a table that is "subsidiary" to the first table. The sub-form control manages the link between the two forms.

**Naming Header/Line Forms** In many cases, a Header/Line form represents a document.

In the previous example, the **Sales Invoice** form, the name of the form will be the name of the document that it represents. In other cases, the name of the form will be whatever the name of the main form would have been without the sub-form.

These situations are described in more detail when we discuss the table building blocks.

## Setup Forms

**Setup Form Characteristics** A setup form is a one-record form that lets you view and edit the one and only one record in a setup table. You are not allowed to insert or delete this record from this form. Since there are many fields, these forms use tabs to organize the information.

Since there is only one record, the primary key is not displayed on this form.

**Naming Setup Forms** Setup forms are named after the table they are associated with. For example, the setup form associated with the **General Ledger Setup** table is called **General Ledger Setup**.

## Menu Forms

**Menu Form Characteristics** A menu form is a non-bound form (not related to any table) that gives you access to many of the other forms that are related to a functional area.

The form usually consists of command buttons or menu buttons. However, the appearance of the buttons is usually not what you would expect. The buttons properties change so that they look basically like labels with small squares or triangles in front of the caption. The buttons still behave like normal buttons. The only difference is their appearance.

**Naming Menu Forms** Menu forms are named after the functional area they are associated with plus the word "Menu". For example, the menu form associated with the General Ledger functional area is called **General Ledger Menu**.

## Other Multi-Record Forms

There are other forms that let you view and/or edit multiple records at the same time. These include TrendScape forms and Matrix forms. These forms give you greater functionality within Dynamics NAV, but do not impact on the main system architecture.

An example of a TrendScape form is the **Contract Trendscape** window.

An example of a Matrix form is the **Budget** window.

## Standard Navigation

This section describes how you move to one form from another. We simply mention the standard forms of navigation that must be supported to ensure consistency throughout the application.

### Card and List Forms

Most master tables have both a card and a list form associated with them. Because both of these forms access the same table, there are many standards that determine what each form must do and these standards must be adhered to.

- On the card form, you must be able to view and edit one record at a time.
- On the list form, you must be able to view but not edit all the records. Making this form non-editable allows you to begin a search by typing in a particular column.
- On the card form, you must be able to open the list form in the following ways:
  - By pressing F5.
  - By clicking the List menu item on the menu button with the same name as the table.
  - By clicking the List or lookup button on the tool bar.
- In the list form, you must be able to select a new record and click OK to exit the list form and revert to the original card form. The card form must now display the record that you selected in the list form.
- In the list form, you must be able to select a record and open a new card form by pressing SHIFT+F5 or clicking the Card menu item on the menu button with the same name as the table. The new card form must display the record that you selected on the list form.

### Master Statistics Forms

As mentioned earlier, most master tables have a card and a list form associated with them. Many master tables also have a statistics form. You should be able to open this statistics form from either the card form or the list form in exactly the same way.

The standard way of getting to the statistics form is to click F9 or the Statistics menu item (on the menu button with the same name as the table).

### Master and Ledger Forms

Every master table has at least and usually only one ledger table linked to it. From the master forms (card, list and statistics), you must be able to open the ledger form. You must be able to open it in three different ways:

- by clicking the drill down button on a FlowField

- by clicking CTRL+F5
- by clicking the Ledger Entries menu item (on the menu button with the same name as the table)

If a drill down button is used, only the records that are used to create the calculated value are displayed (this is a function of the built-in drill down functionality). If the shortcut key or the menu item is used, all the ledger entries for this particular master record are shown. This requires a *RunFormLink* on the menu item that links the two tables. You must also keep the two forms synchronized whenever the master form is updated. To achieve this, you must change the *RunFormLinkType* property to *OnUpdate*.

**Note**

For performance reasons, you should always set the *RunFormView* property if the *RunFormLink* property is also set. In fact, the sort order chosen in the *RunFormView* property must contain the fields listed in the *RunFormLink* property or else performance is decreased.

**Journal Forms**

Every journal form has similar buttons at the bottom of the form. These buttons allow you to navigate to another form or perform a task.

The normal buttons on a journal form include: Line, Functions, Posting, Account and a menu button named after the master table for the functional area. This menu button must contain menu items that let you go directly to the card form for the master table or to the ledger form for the ledger entries. In both cases, the form that is opened should be linked to the master table record that the journal line is associated with.

The "Posting" menu button should contain the following menu items related to posting the journal lines – Post, Post and Print, Test Report.

## **Part 5 Reports**



## **Chapter 12**

### **Report Fundamentals**

Reports are used to print information from a database. A report can be used to structure and summarize information, and reports can be used to print documents such as invoices. Reports can also be used to process data without printing anything.

This chapter introduces the fundamental concepts and basic tasks involved in designing reports.

- What Are Reports?
- What Happens When a Report Runs?
- The Report Designer
- Saving, Compiling and Running Reports

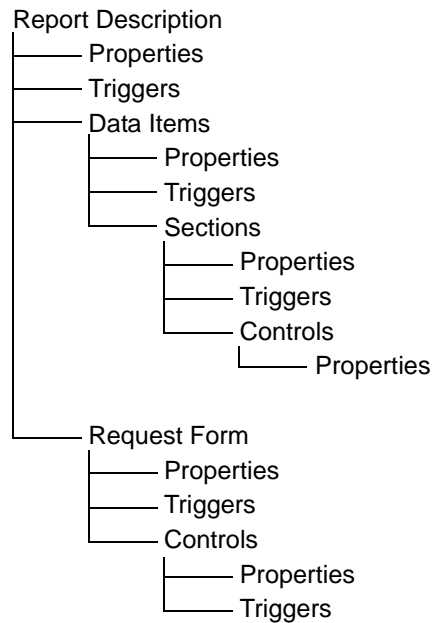
## 12.1 What Are Reports?

Reports have several purposes in Dynamics NAV:

- Reports are used to print information from a database in a structured way. For example, you can create a report that lists all the customers and all the orders placed by each customer.
- Every application document must be created as a report. For example, to print an invoice, you must create a report that is automatically filled out with the relevant information.
- Reports can also be used for other tasks and not just for printing. A report can be used to automate many recurring tasks such as updating all the prices in an item list. This could be achieved by writing C/AL code in a codeunit, but using a report is much easier because you can take advantage of the powerful data modeling facilities available in the report designer.

### The Report Components

Reports consist of several components. The following diagram shows how these components are related:



**Report Description** This is the complete description of the report including how data is collected, and how it is presented on paper when the report is run. The report description is stored in the database.

**Data Item** This corresponds to a table. To retrieve information from tables, you define data items. When a report uses more than one table, you must set relations between the data items so that you can retrieve and organize the data the way that you want.

**Section** In a report that is going to be printed, each data item has one or more sections. A section can be thought of as a block of information that should be printed. The complete report is composed of a number of sections. Some sections are printed



only once, for example, a header and some are printed for each record retrieved from the database.

**Control** The information printed in the sections is composed of controls. The available controls are:

- text boxes – for printing the result of the evaluation of any valid C/AL expression, such as the contents of a table field (but also for complex calculations).
- labels – for printing static text such as a caption for a column of data.
- shapes, images and picture boxes – for printing graphical elements (lines, circles) and bitmap pictures.

**Request Form** A form that is run before the report starts to execute. You use request forms to enter requests and select options for the report, for example, the sort order or the level of detail that you want in the report.

**Property** A property is an attribute of an object – report, data item, section and so on, for example, color, size, and whether or not it is displayed. Properties are set in the **Properties** window of the object.

**Trigger** Certain predefined events that cause the system to execute a user-definable C/AL function. The event triggers the function. The report itself, the data items, the sections, the request form, and the controls all have triggers.

## Logical and Visual Design

There are two parts to designing a report:

- Defining the logical structure or data model.
- Designing the visual layout.

Defining the data model means defining how the data is collected. This includes:

- creating data items that define the tables that the report uses.
- defining the relationships between the data items if the report uses more than one table.
- defining the key, sort order and filters to use with the data items.
- defining how data is grouped.
- defining how totals and subtotals are calculated.
- writing C/AL code in data item triggers (if necessary) to improve the functionality of the report.

### Data items

The report data model is built from data items, which each correspond to a table. When the report is run, each data item is iterated for all the records in the underlying table. When a report is based on more than one table, you indent the data items to establish a hierarchy of data items and control how the information is gathered.

### Example

To make a report that prints out a list of customers and lists the sales orders placed by each customer, you must define two data items. One data item corresponds to the **Customer** table and the other corresponds to the **Sales Order** table. The second data item should be indented. This

means that as the report works through the records in the **Customer** table, each customer's sales orders are found by going through the records in the **Sales Order** table.

Sections

The visual layout of a report includes the sections. In a printing report (remember that reports do not actually have to print anything), one or more sections are attached to each data item.

There are several types of sections, each with a specific function. Normally, the bulk of the data is printed in the body section of a data item, while the header section is used to print information before any data item record is printed (for example, column captions). But some reports do not use the body section, and all the information is printed in other sections.

The following picture shows a finished report:

Sales Statistics ORIONUS International Ltd.						
No.	Name	Balance	25-01-01 24-02-01	25-02-01 24-03-01	25-03-01 24-04-01	Var...
<b>0142594 Progressive Home Furnic</b>						
	Sale (LCY)	1,400.00	0.00	0.00	0.00	0.00
	Profit (LCY)	305.10	0.00	0.00	0.00	0.00
	Profit %	20.4	0.0	0.0	0.0	0.0
	Inv. Discounts (LCY)	0.00	0.00	0.00	0.00	0.00
	Finl. Discounts (LCY)	0.00	0.00	0.00	0.00	0.00
	Finl. Disc Tot. (LCY)	0.00	0.00	0.00	0.00	0.00
	Finl. Tolerance (LCY)	0.00	0.00	0.00	0.00	0.00
<b>0145495 New Concept Furniture</b>						
	Sale (LCY)	222,041.02	0.00	0.00	0.00	0.00
	Profit (LCY)	0.00	0.00	0.00	0.00	0.00
	Profit %	0.0	0.0	0.0	0.0	0.0
	Inv. Discounts (LCY)	0.00	0.00	0.00	0.00	0.00
	Finl. Discounts (LCY)	0.00	0.00	0.00	0.00	0.00
	Finl. Disc Tot. (LCY)	0.00	0.00	0.00	0.00	0.00
	Finl. Tolerance (LCY)	0.00	0.00	0.00	0.00	0.00
<b>10000 The Cannon Group P/L</b>						
	Sale (LCY)	300,122.34	3,150.00	0.00	0.00	0.00
	Profit (LCY)	5,385.46	2,410.89	0.00	0.00	0.00
	Profit %	1.7	76.8	0.0	0.0	0.0
	Inv. Discounts (LCY)	727.34	0.00	0.00	0.00	0.00
	Finl. Discounts (LCY)	0.00	0.00	0.00	0.00	0.00
	Finl. Disc Tot. (LCY)	0.00	0.00	0.00	0.00	0.00
	Finl. Tolerance (LCY)	0.00	0.00	0.00	0.00	0.00
<b>20000 Selanganian Ltd.</b>						
	Sale (LCY)	210,087.80	0.00	0.00	0.00	0.00
	Profit (LCY)	2,511.90	0.00	0.00	0.00	0.00
	Profit %	1.2	0.0	0.0	0.0	0.0
	Inv. Discounts (LCY)	201.36	0.00	0.00	0.00	0.00

This report prints sales statistics information and retrieves all the data that it uses from one table. It demonstrates a range of features that are available for designing reports.

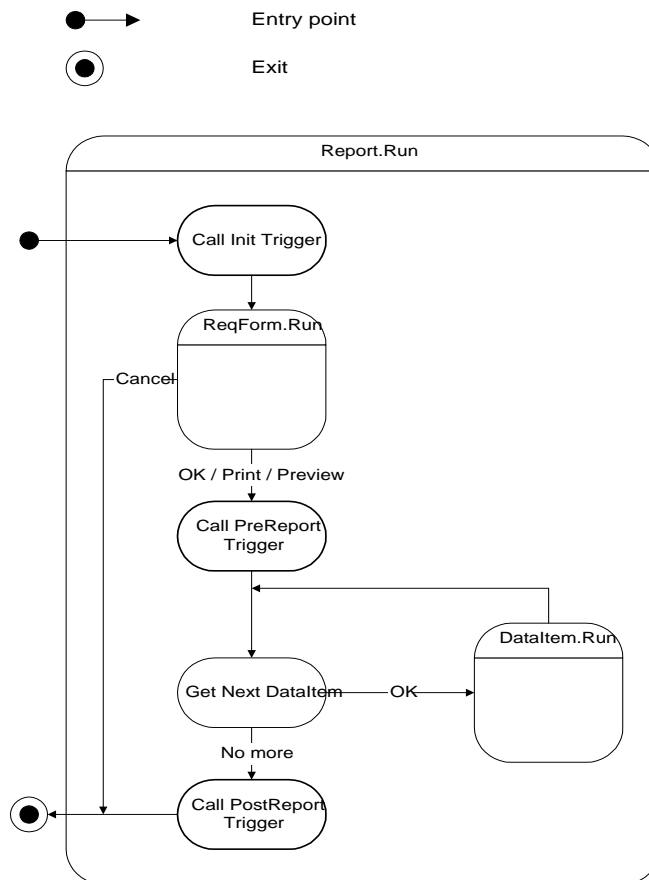
- Before any record from the table is printed, there is a header. The header contains a title and information about the filter used on the customer numbers.
- Each body section prints several lines of information about a customer. The % lines are calculated as the report is run.
- After all the records (selected by the filter) have been printed, a footer section is printed that contains totals for the selected customers.
- In the body section and in the footer section, a filter is applied to create columns where data is collected and totalled for different periods.

## 12.2 What Happens When a Report Runs?

The following flow charts are simplified versions of those shown in Appendix A, Report Flow Charts, on page 636. If you want to acquaint yourself with all the details – including why and when triggers are executed – consult that appendix. This section only describes the way a report is run in general terms.

### The Report Run

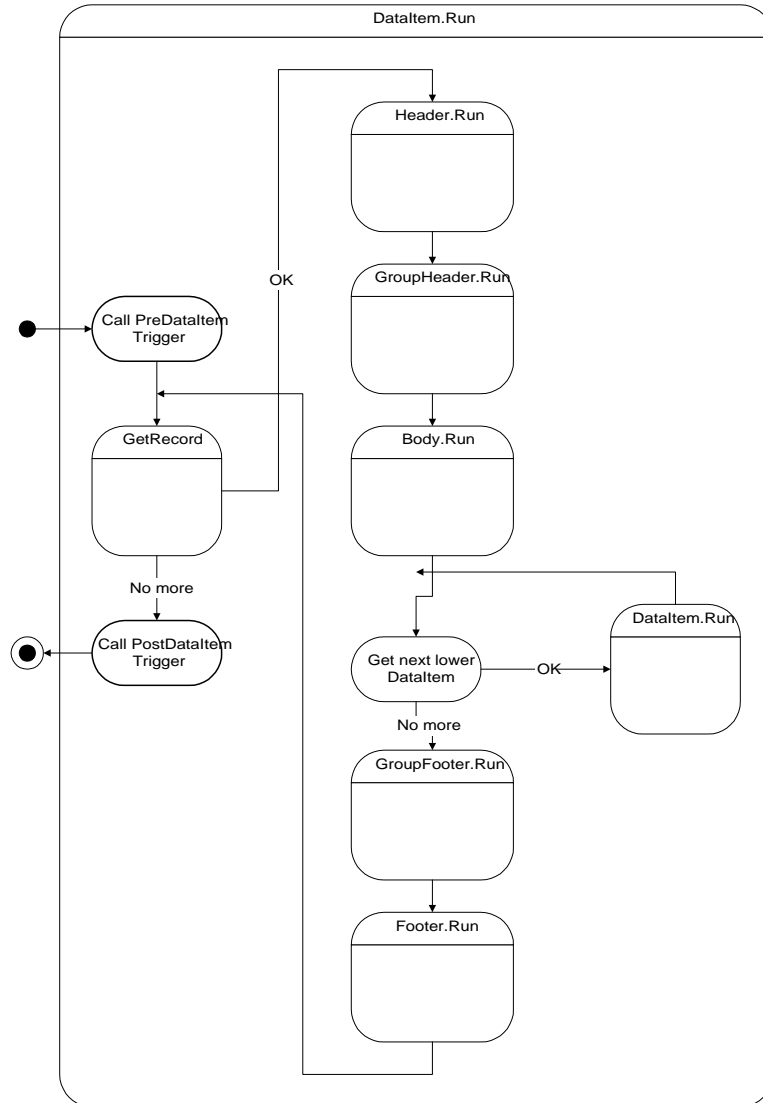
The first flow chart illustrates the events that take place when you run a report:



- 1 When you initiate the report run, the *OnInitReport* trigger is called. This trigger performs any processing that is necessary before the report is run. It can also stop the report.
- 2 If the *OnInitReport* does not end the processing of the report, the request form for the report is run, if it is defined. Here, you select the options that you want for this report. You can also decide to cancel the report run.
- 3 If you decide to continue, the *OnPreReport* trigger is called. At this point, no data has yet been processed.
- 4 When the *OnPreReport* trigger has been executed, the first data item is processed (provided that the processing of the report was not ended in the *OnPreReport* trigger).

- 5 When the first data item has been processed, the next data item, if there is any, is processed in the same way.
- 6 When there are no more data items, the *OnPostReport* trigger is called to do any necessary post processing. For example, removing temporary files.

The next flow chart elaborates on step 4 – how a data item is processed:



- 1 Before the first record is retrieved, the *OnPreDataltem* trigger is called, and after the last record has been processed, the *OnPostDataltem* trigger is called.
- 2 Between these two triggers, the data item records are processed. Processing a record means executing the record triggers and outputting sections. C/SIDE also determines whether the current record should cause the outputting of a special section, such as a header, footer, group header or a group footer.
- 3 If there is an indented data item, a data item run is initiated for this data item (data items can be nested 10 levels deep).

- 4 When there are no more records to be processed in a data item, control returns to the point from which processing was initiated. For an indented data item, this means the next record of the data item on the next highest level. If the data item is already on the highest level (indentation is zero) control returns to the report – as shown in the first flow chart (Report.Run).

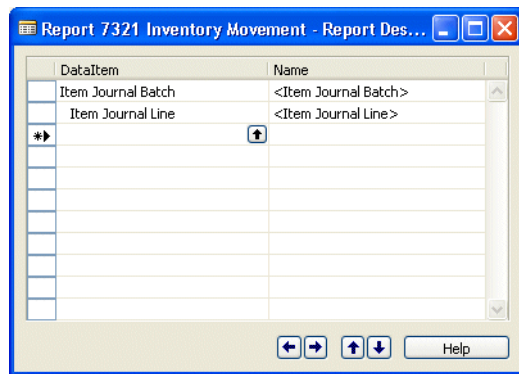
## 12.3 The Report Designer

You use the Report Designer to create the reports that you want to use in your application. The Report Designer contains two additional designers:

- Section Designer, used for designing the layout of reports.
- Request Options Form Designer, used for designing request options forms.

### The Report Designer

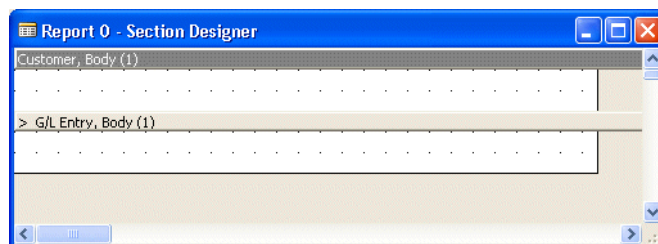
You use the **Report Designer** window to define the data model by adding data items and indenting them appropriately:



You can edit the properties and triggers for each of the data items by selecting the data item and opening the **Properties** window or the C/AL Editor, respectively. To edit the properties of the report itself, select an empty line in the **Report Designer** window and open the **Properties** window. To edit the triggers of the report itself, select an empty line in the **Report Designer** window and click View, C/AL Code. Alternatively, you can click Edit, Select Object and then open the **Properties** window or the C/AL Editor.

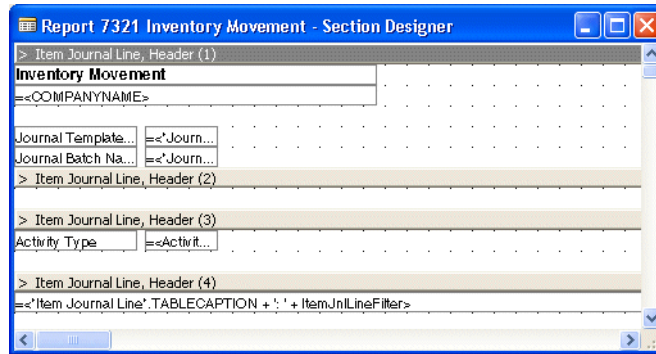
### The Section Designer

After you have defined one or more data items, you can design the visual layout of the report in the Section Designer:



You can use the Field Menu to select fields and place them in the sections as controls. This is described previously for forms.

In the following picture, a number of text boxes and labels have been placed in four sections.



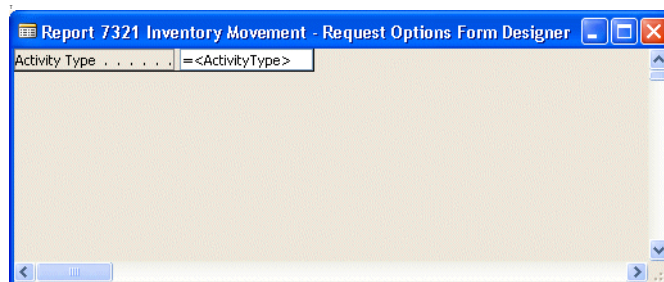
You can think of each section as one or more lines on the paper that the report will eventually be printed on. A header section is printed only once, while a body section is typically printed several times as each loop of the report is iterated. You can control whether the header should be printed every time a page break occurs while the body sections of the same data item are being printed.

You can edit properties and triggers for each section by opening the **Properties** window or the C/AL editor, respectively, while the section is selected.

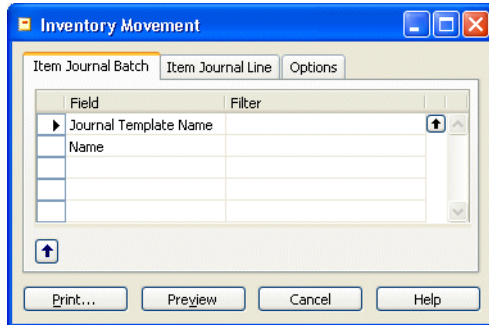
The controls you place in the sections have a subset of the properties that controls have on forms (not all the properties are relevant for a report), and you can use the same tools to modify the properties (the Font Tool, the Color Tool). You can see a list of the properties in the **Properties** window. For more information about these properties, see the *C/SIDE Reference Guide* online Help.

### The Request Options Form Designer

You use the Request Options Form Designer to create a form with fields that prompt the user for options before the report is run. This designer works like the Form Designer:



You only use the Request Options Form Designer if you want to prompt the user to select some options before running the report. When a report is run, the request form can look something like this:



As you can see, a form with a tab control has been created. The first two tabs correspond to data items. They are created automatically (though you can control the contents by setting properties of the data items), and used for setting filters and defining the sort order.

The third tab, **Options**, only appears when the Request Options Form Designer has been used to create a request options form.

The form has the same properties and triggers as any other form, and the same controls can be placed on it.



## 12.4 Saving, Compiling and Running Reports

After you have designed a report, you must save and compile it before it can be run. Normally, you do this after you finish designing the report. However, you may want to save a report that is not yet finished and therefore cannot be compiled, for example, if the report contains C/AL code that is incomplete. You can also test-compile a report without closing or saving it.

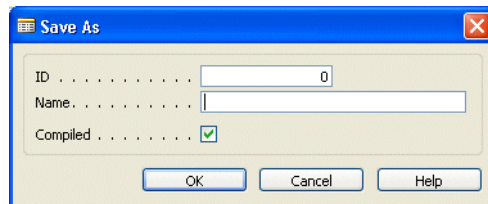
### Saving and Compiling a Report

A report is closed when the *Report Designer* window is closed.

To save a report:

- 1 When you close a report, you are prompted to decide whether or not the report should be saved. If it is a new report (a report that has not been saved before) you must give it an ID and a name. The ID must be unique and follow the rules for numbering objects - your local Microsoft Certified Business Solutions Partner will provide you with this information.

Hint: if you enter ID and Name as report properties, these values are used, and you are not prompted for an ID and a name when you close the report.



- 2 The **Compiled** option field is set to TRUE by default (displayed as a check mark). If your report is not yet ready to be compiled, remove the check mark by clicking in the field.
- 3 Click OK to save the report.

You can save a report without closing it by clicking File, Save or Save As. By using Save As, you can rename an existing report (in effect copying it).

Like all the other objects in C/SIDE, reports must be compiled before they can be run. When you are designing a report, you might want to test-compile it to find any errors that it may contain (this is more important when the report contains C/AL code in triggers). To test-compile a report when you are designing it, click Tools, Compile.

### Running a Report

In an application, your reports can be incorporated into menus, or they can be called from, for example, a command button on a form. However, when you are designing reports, you will often want to run them before they are integrated into the application.

**Test-running reports** When you are designing a report, you can test-run the report by clicking File, Run. The report is compiled and run in its current stage of development. It is not saved, which

means that you can use this function to verify that the changes you are making work as intended before you save them.

Running reports  
from the Object  
Designer

You can run a report from the list of reports in the **Object Designer** window by selecting it and clicking the Run button.

## **Chapter 13**

### **Designing Reports**

This chapter describes the properties of reports, and then, by creating two examples, shows the basic steps involved in designing reports.

- Report Properties
- Designing a Simple Report
- Designing a More Advanced Report

## 13.1 Report Properties

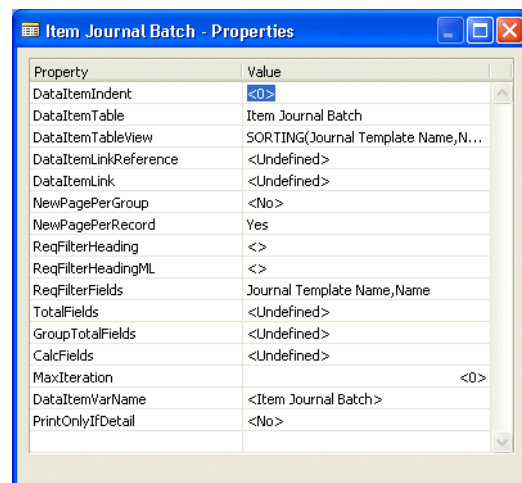
Properties are a system-wide feature and every application object has some properties. Every object in a report has properties, including:

- The report itself
- Data items
- Sections
- Controls in the section
- Request forms
- Controls on a request form

You can select these objects in the following ways:

- Select a data item in the **Report Designer** window by clicking it.
- Select the report itself by clicking an empty line in the Report Designer or by clicking Edit, Select Object.
- Select a section in the Section Designer by clicking either the section bar or somewhere in the section (but not on a control).
- Select a control by clicking it.

When an object is selected, click View, Properties to open the **Properties** window:



You set the value of each property in the **Value** field. As soon as you leave this field (by pressing ENTER or by moving with the arrow keys), the property is updated. If you entered an error, the update is not accepted and you must enter a correct value.

Default values are displayed in angle brackets (<>). You can reset any property to its default value by deleting the current value and then moving out of the field. This obviously only applies to properties that have a default value.

### How Properties Are Inherited

Controls that have a direct relationship to table fields inherit the settings of those properties that are common to the field and the control. For example, if you have an accounting application that stores some calculated amounts with five decimal places (for precision). But on a printed report, you only want to display currency amounts with

the usual number of decimal places. You can then change the *DecimalPlaces* property of the text box control to display fewer decimals than the default (but not more).

## Report Properties

The following table briefly describes the report properties. The *C/SIDE Reference Guide* online Help contains more detailed information about these properties and you can get context-sensitive Help for a property by opening the **Properties** window for the report, selecting the property in question and pressing F1.

To open the **Properties** window of a report, select an empty line in the Report Designer or click Edit, Select Object and then click View, Properties (SHIFT+F4).

Property	Meaning
ID	ID of the report – must be unique among reports.
Name	Name of the report.
Caption	Caption (shown on request form window, for example – default is the same as Name).
CaptionML	The translations of the caption.
ShowPrintStatus	Whether or not the printing status window should be displayed during printing (with the opportunity to cancel printing).
UseReqForm	Whether or not the request form should be run before the report.
UseSystemPrinter	If Yes, then the system default printer is suggested as printer for the report. If No, then the printer defined for the combination User/Report in the setup of the system is suggested.
ProcessingOnly	If No, printing–only processing. If Yes, the report cannot have sections.
TransactionType	The behavior of a transaction in Dynamics NAV and takes effect from the beginning of a transaction. There are four basic transaction type options: Browse, Snapshot, UpdateNoLocks and Update. There is a Report option that maps to one of the basic options and enables a report to use the most concurrent read-only form of data access. When you use C/SIDE Database Server, it maps to Snapshot and when you use SQL Server, it maps to Browse.
Description	Description – for internal purposes, as it is not visible to the user.
TopMargin	Topmargin in 1/100 mm.
BottomMargin	Bottom margin in 1/100 mm.
LeftMargin	Left margin in 1/100 mm.
RightMargin	Right margin in 1/100 mm.
HorzGrid	Distance between horizontal gridlines (1/100 mm).
VertGrid	Distance between vertical gridlines (1/100 mm)
Permissions	The permissions of the report to access database objects. (The report can have wider permissions than the individual user, thereby enabling the user to print reports that retrieve information from tables that he or she cannot normally access.)

Property	Meaning
Orientation	Sets the page orientation for this report – Portrait or Landscape.
PaperSize	Sets the paper size for this report.
PaperSourceFirstPage	Specifies the paper source to use for printing the first page of this report. This is useful if the report has a cover.
PaperSourceOtherPage	Specifies the paper source to use for the rest of the pages in this report.
DeviceFontName	Use this property for reports that are designed specifically for dot matrix printers to prevent the printer from switching into graphics mode when printing text. Specify the name of a device font (a font that is built into a printer).

### Data Item Properties

The next table briefly describes the data item properties.

The *C/SIDE Reference Guide* online Help contains more detailed information about these properties and you can get context-sensitive Help for a property by opening the **Properties** window for the report, selecting the property in question and pressing F1.

Property	Meaning
DataltemIndent	The indentation level (can be set in the designer when creating data items).
DataltemTable	The table that the data item is based on (can be set in the designer when creating data items).
DataltemTableView	The key, sort order and filters to apply.
DataltemLinkReference	The DataltemVarName of a less-indented data item that this data item will be linked to.
DataltemLink	Link between the current data item and the data item specified by DataltemLinkReference.
NewPagePerGroup	Whether or not each group should be printed on a separate page.
NewPagePerRecord	Whether or not each record should be printed on a separate page.
ReqFilterHeading	The caption for the request form tab that relates to this data item (default is the name of table that the data item is based on).
ReqFilterHeadingML	Translations of the caption for the request form tab.
ReqFilterFields	Names of the fields that will be included in the request filter form.
TotalFields	Names of the fields for which totals will be calculated.
GroupTotalFields	Names of the fields that will be used for grouping data.
CalcFields	Names of the fields that will be calculated after a record has been retrieved.
MaxIteration	Maximum number of data item loop iterations.
DataltemVarName	Name of record as variable (default is the name of table that the data item is based on).

Property	Meaning
PrintOnlyIfDetail	Print item only if sublevels generate output.

### Section Properties

The next table briefly describes the section properties. All the properties are described in detail in the online *C/SIDE Reference Guide* online Help. You can get context-sensitive Help for a property by opening the **Properties** window for a section, selecting the property in question and pressing F1.

Property	Meaning
PrintOnEveryPage	Whether or not the header and footers should be printed on every pages.
PlaceInBottom	Whether the footer should be placed below the last line or at the bottom of the page.
SectionWidth	Width in 1/100 mm.
SectionHeight	Height in 1/100 mm.

### Control Properties

Controls in reports have the same properties as controls on forms. The **Properties** window of a control shows the properties, and they are described in the *C/SIDE Reference Guide* online Help.

## 13.2 Designing a Simple Report

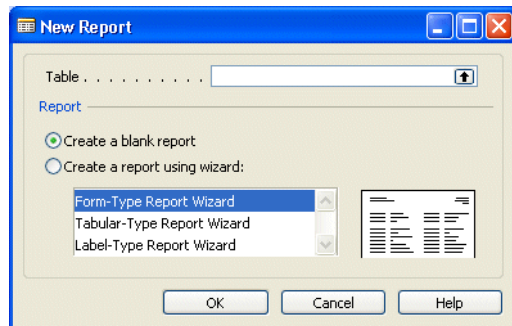
This example shows you how to create a very simple report, in which a list of customers is created, based on one table that contains customer information.

### Defining the Data Model

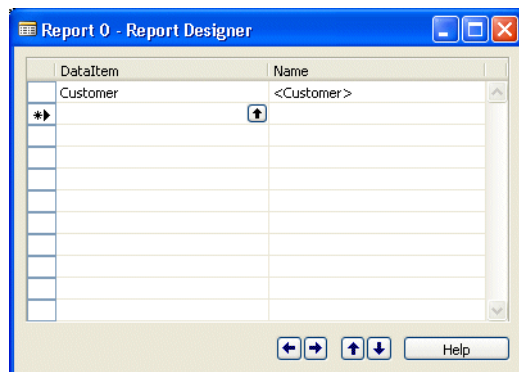
The first step in creating this simple report is to define the data model on which the report is based. You define the data model by creating the data items.

To create a data item:

- 1 Click Tools, Object Designer (SHIFT+F12).
- 2 In the Object Designer, click Report, New and C/SIDE opens the **New Report** window:



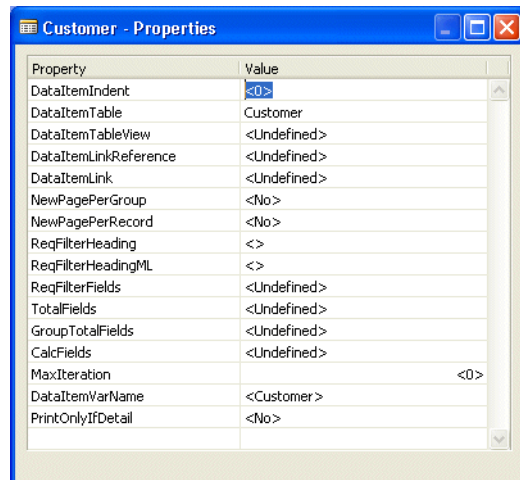
- 3 In the **Table** field, click the AssistButton  $\uparrow$  to select a table from the **Table List** window.
- 4 In the **Report** section, click **Create a blank report** and then click OK. The **Report Designer** window opens:



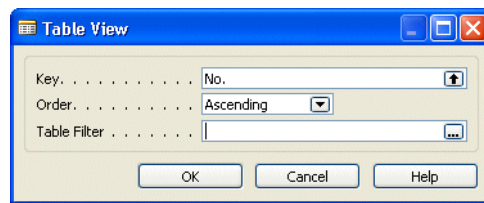
- 5 In the first **Data Item** field, click the AssistButton  $\uparrow$  and select a table from the **Table List** window. For this example, select the **Customer** table.



6 Click View, Properties (SHIFT+F4) to open the **Properties** window for the data item.

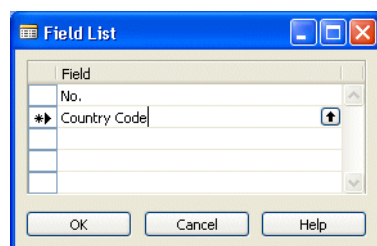


7 Select the *DataItemTableView* property and in the **Value** field, click the AssistButton ... to open the **Table View** window:



8 Select the key, sort order and filters to use and then click OK. In this example, the **No.** field is chosen as the key, and the sort order is set to Ascending. The **Table Filter** field is left empty, meaning that a permanent filter is not defined on the table.

9 Select the *ReqFilterFields* property, and in the **Value** field, click the AssistButton ... to open the **Field List** window:

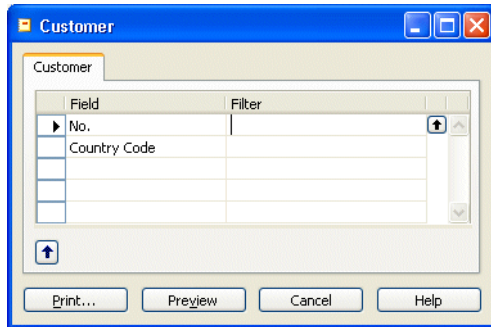


10 Select the fields on which users will frequently need to set filters. You can use the lookup function to select them. In this example, the fields **No.** and **Country/Region Code** are selected. When you have selected the fields, click OK.

11 Click File, Save to save the report and then run it from the Object Designer to see the request form for this report. Alternatively you can just click File, Run to see the request form.

To learn more about saving and compiling reports see, "Saving, Compiling and Running Reports" on page 221.

The following picture shows the request form that you see when this report is run:



You have already specified the key and sort order when you were designing the report, so the only choice left is to set the filters.

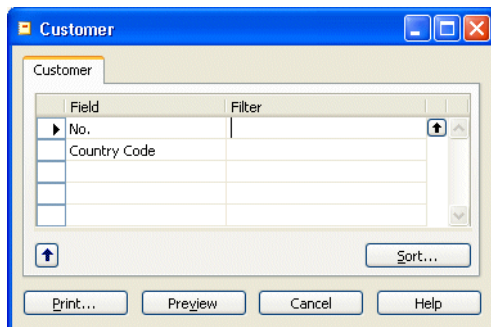
The fields that you defined as ReqFilterFields are shown, but you can decide to place a filter on other fields by adding lines below those already used. Remember that users can almost always choose to set filters on fields which you did not specify.

Nevertheless, it is usually a good idea to add the fields that the users of the report will frequently set filters on. You should always strive for balance in your design. If the table has a lot of fields, inexperienced users may find it difficult to find the relevant fields that they want to set filters on.

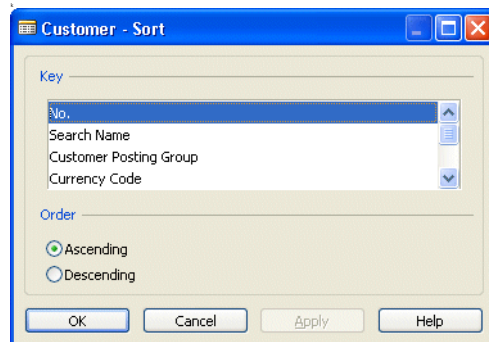
You can remove the tab where you set the filters altogether by not defining any ReqFilterFields for the data item and by setting the DataItemTableView to define a sort order. If you create a request options form, it will still be shown.

If there is no request options form, an empty form is displayed. On this, users can choose Print, Cancel, and so forth. If you set UseReqForm to No, the report will start printing as soon as it is run. In this case, users cannot change their minds and cancel the report run. (It will still be possible to cancel the print run, but some pages will probably be printed).

If a DataItemTableView is not defined, users can select the key and sort order at runtime. Then, the request form looks like this:



When you click Sort, you can choose the key and sort order from this form:



### Note

Be careful what you allow users to change. In a more complex report that involves data from several tables, the functionality may depend on a specific key and sort order. On the other hand, letting the user choose filters freely does not interfere with the logic of the report. In a very simple report like this, you can select a key and define a sort order if you want, or leave it up to the user.

### Using the Wizards

In the **New Report** window, in the **Report** field, you could have chosen to use one of the wizards, for example Form-Type Report Wizard, rather than creating the report from scratch.

The wizards guide you through the process of selecting the fields that the report is based on and the sorting order.

Note that on the first page of the wizard, the contents of the **Available Fields** field are the *Caption* properties of the fields – not the *Name* property.

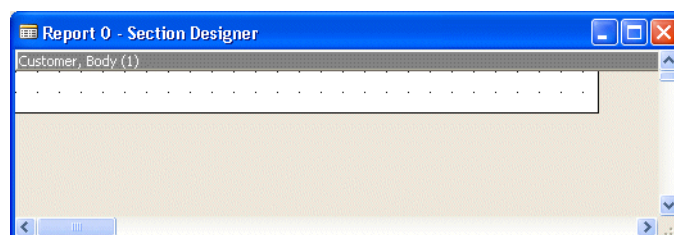
### Designing the Sections

So far, only the data model of our simple report has been defined and nothing will be printed. The next step is to design the sections.

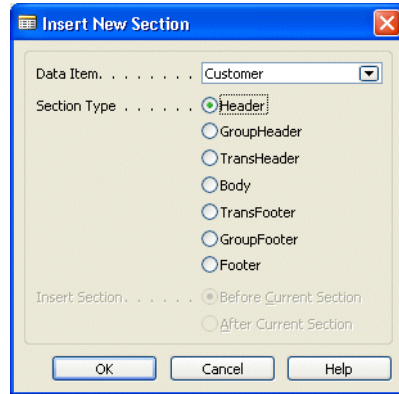
To design the report sections:

- 1 While the **Report Designer** window has the focus, click View, Sections to open the Section Designer.

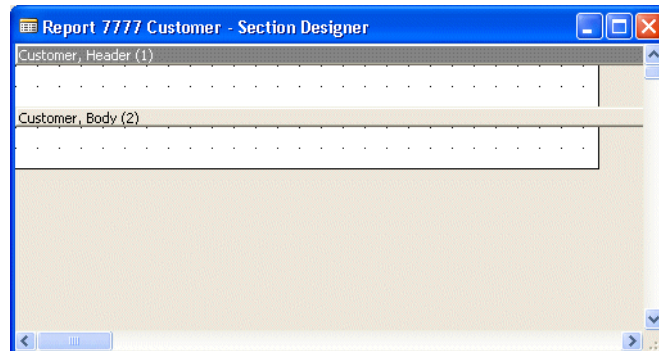
After you have created a data item as described earlier, the Section Designer looks like this:



- 2 As you can see, a section named "**Customer,Body (1)**" has been inserted. ("1" means that this is currently the first section of this data item.) By default, a Body section is inserted for each data item that has been created. These sections appear in the same order as the data items in the Report Designer.
- 3 Click Edit, Insert New to insert a header section for the Customer data item. The **Insert New Section** window appears:

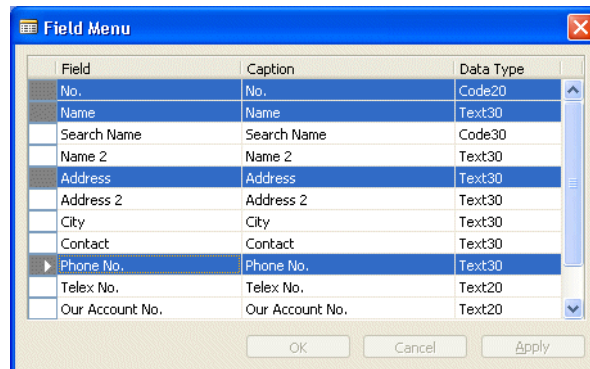


- 4 Select Header as the Section Type and click OK. The Section Designer now looks like this:

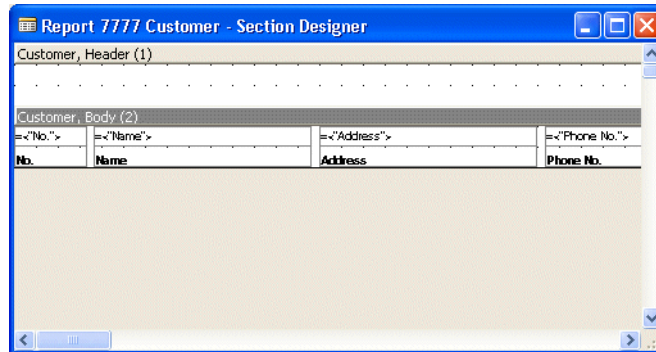


- 5 Click View, Field Menu to open the field menu. Select the fields that you want in the report. (You can select multiple fields by holding down CTRL while you select the fields you want.)

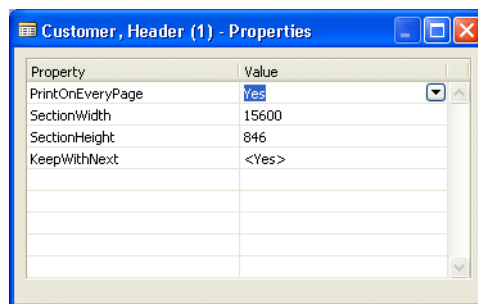
In the following example, four fields have been selected:



- 6 Move the mouse cursor into the Body section of the data item. Click once to activate the window – the cursor changes into the Control Insertion cursor. Place the cursor at the left side of the section and click again. A text box with an attached label is inserted for each field that you selected.

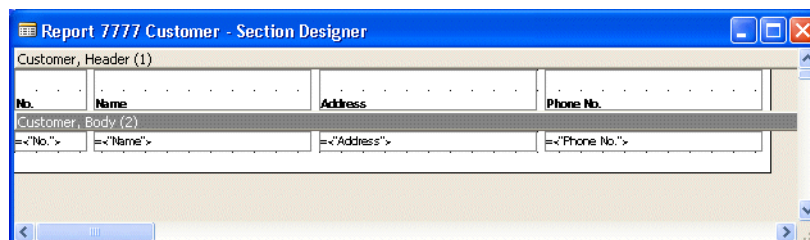


- 7 Click View, Properties (SHIFT+F4) to open the **Properties** window. Click in the Header section to ensure that the **Properties** window displays the properties of the header. Look at the setting of SectionWidth (the unit of measure is 1/100 mm).



The width of all the sections has been modified to make room for the controls you inserted (the default width, when no controls have been inserted, is 12000). In this case, the resulting width is 15450, or 15.45 cm. If the report is going to be printed on A4 paper, this is perfectly acceptable – A4 paper is 21 cm wide.

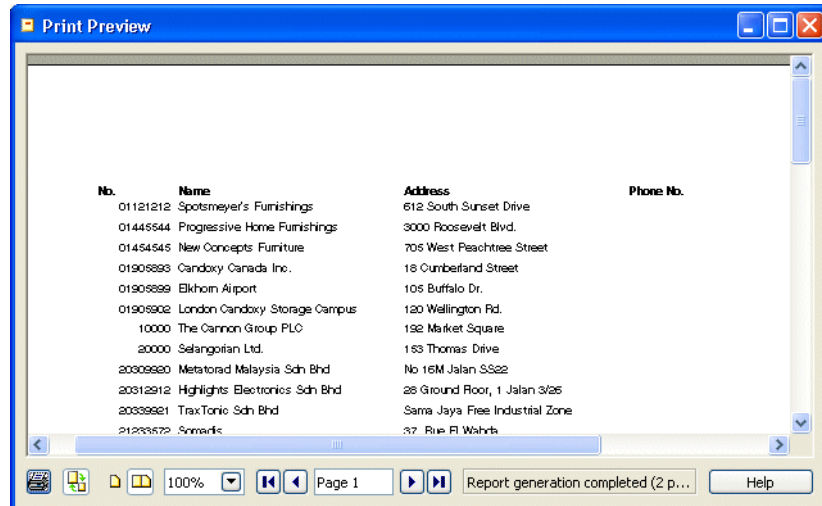
- 8 Select all the labels in the Body section (hold down CTRL while clicking), and move them all into the Header section in one go (this preserves the alignment of the labels and text boxes).



Now, the report is ready to be printed, but it still needs some work before it will look good on paper.

- 9 If the report consists of more than one page when it is run, you will want the Header section, that contains the labels, to appear on every page. Open the **Properties** window for the Header section and set the *PrintOnEveryPage* property to Yes.

- 10 Moving the labels out of the Body section has left this section too high – there will be an empty line for each customer record that is printed. To resize the Body section, move the cursor into the Section Designer until it touches the lower bound of the Body section and turns into the vertical resizing cursor. Then click and drag the section upwards until it has the same height as the text boxes.
- 11 Close and save the report and run it from the Object Designer. Click Preview and you can see that the example described so far gives the following result with the sample data:



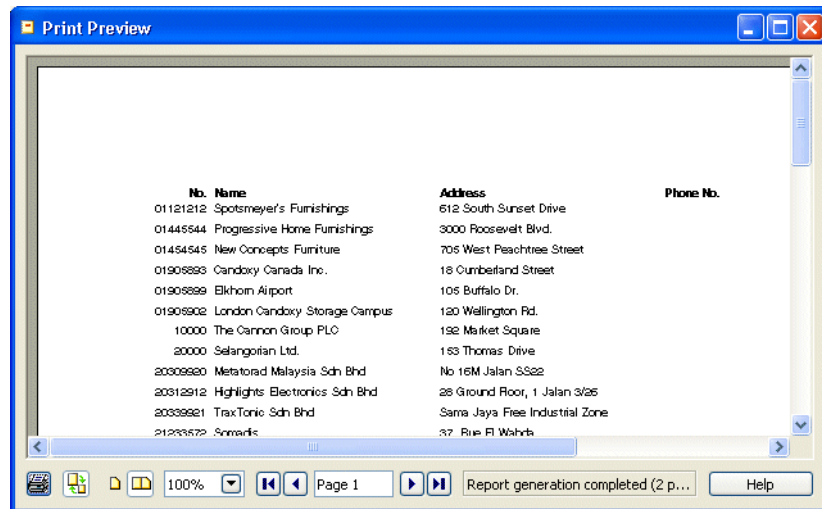
As you can see, the label and the data in the **No.** column do not line up very nicely. This is because both controls have their alignment set to General (the default). The label is left aligned because it contains text, while the text boxes are right aligned because they contain numbers.

A simple solution is to right align the label.

To realign the labels:

- 1 Reopen the report in the Report Designer and open the Section Designer.
- 2 Select the No. label and open the **Properties** window. Set the *HorzAlign* property to Right.
- 3 Close and save the report.
- 4 Run the report in the Object Designer and click Preview.

Now the report looks like this:



The screenshot shows a 'Print Preview' window with a report table. The table has four columns: 'No.', 'Name', 'Address', and 'Phone No.'. The data is as follows:

No.	Name	Address	Phone No.
01121212	Spotsmeyer's Furnishings	612 South Sunset Drive	
01445544	Progressive Home Furnishings	3000 Roosevelt Blvd.	
01454545	New Concepts Furniture	705 West Peachtree Street	
01905593	Candoxy Canada Inc.	18 Cumberland Street	
01905599	Elkhorn Airport	105 Buffalo Dr.	
01905902	London Candoxy Storage Campus	120 Wellington Rd.	
10000	The Cannon Group PLC	192 Market Square	
20000	Selangorisan Ltd.	153 Thomas Drive	
20309980	Metatorad Malaysia Sdn Bhd	No 16M Jalan SSe2	
20312912	Highlights Electronics Sdn Bhd	28 Ground Floor, 1 Jalan 3/25	
20339921	TraxTonic Sdn Bhd	Sama Jaya Free Industrial Zone	
21233572	Somarks	37 Rue El Wahyb	

The window includes a toolbar at the bottom with icons for print, save, zoom (100%), navigation (back, forward), page number (Page 1), and a status bar indicating 'Report generation completed (2 p...)' and a 'Help' button.

## 13.3 Designing a More Advanced Report

The report designed in the previous exercise was very simple: it was based on one table which it ran through and printed the records. In this section, you will learn how to design reports that are based on more than one table.

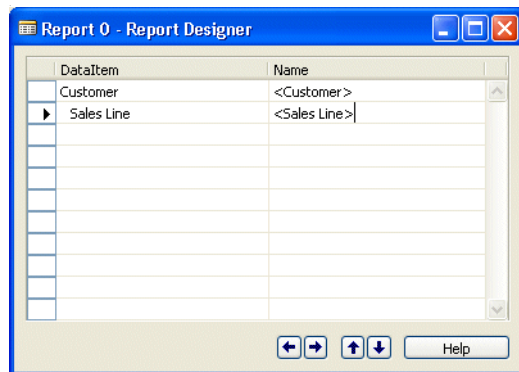
The sample report that you will create uses two tables: one is the **Customer** table that was used in the preceding example. The other table is the **Sales Line** table that contains not-yet-posted sales orders that contain information about the actual items that have been ordered by the customers. There is a one-to-many relationship between the two tables: while one customer can have many items on order, each sales line can only refer to one customer.

### Defining the Data Model

The description of the steps involved in creating this report presumes that you are familiar with the techniques explained earlier.

To define the data model:

- 1 Open the Object Designer (SHIFT+F12), click Report, New and create a blank report that is not based on any table.
- 2 In the **Report Designer** window, select the **Customer** table as the first data item, and the **Sales Line** table as the second data item.
- 3 Select the **Sales Line** data item and click the right-arrow button once to indent it:



This data model works like this:

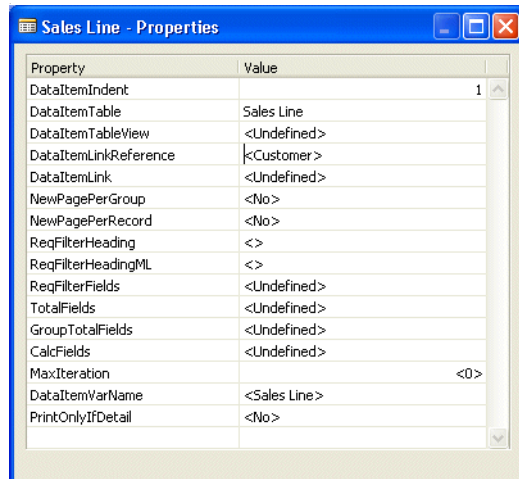
- The report runs through the Customer data item.
- For *each* record in the Customer data item, the report runs through the *entire* Sales Line data item.

This is clearly not what the report should do. The report should be able to select only those Sales Line records that are related to the current customer. This is done by using the *DataItemLink* and *DataItemLinkReference* properties.

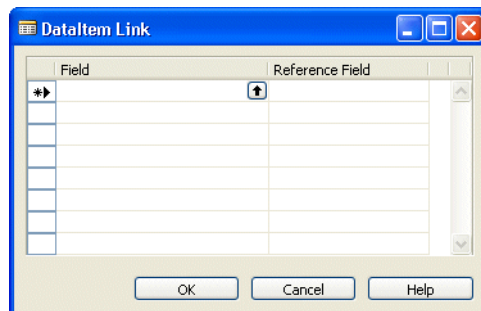
The *DataItemLinkReference* property points to a data item on a higher level (with less indentation) and the *DataItemLink* property specifies a field in each data item: here, records are selected from the **Sales Line** table only when the Sell-to Customer No. is the same as the No. in the **Customer** table.



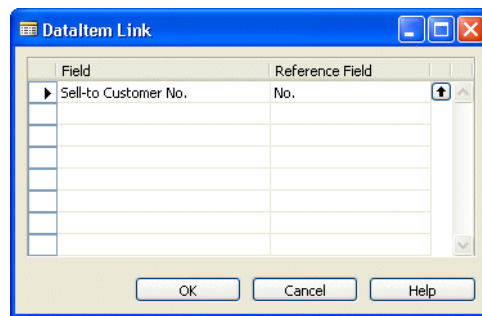
- 4 Open the **Properties** window for the Sales Line data item.
- 5 Set the *DataItemLinkReference* property to the name of the less-indented data item (Customer) that the more-indented data item (Sales Line) must be related to. In most cases, including this one, this is the default.



- 6 In the **Value** field of the *DataItemLink* property, click the AssistButton... to open the **DataItem Link** window:



- 7 In the **Field** field, enter the name of the field from **Sales Line** table (the more-indented data item) that must correspond to a field from the **Customer** table (the less-indented data item). Use the lookup function to select the field. For this example, select the **Sell-to Customer No.** field.
- 8 In the **Reference Field** field, enter the name of the field from the **Customer** table that must correspond to the field from the **Sales Line** table. Again, use the lookup function to select the field. In this example, select the **No.** field from the **Customer** table.



- 9 Finally, open the **Properties** window for the Customer data item, and set the *PrintOnlyIfDetail* property to Yes. This means that the Customer body sections are only printed if there is data to print from the **Sales Line** table.

The data model now works like this:

- The report runs through the Customer data item.
- For each record in the Customer data item, records in the Sales Line data item are selected if the **Sell-to Customer No.** field has the same value as the **No.** field in the Customer data item.
- If there are no Sales Line records for a Customer, nothing is printed – not even the information from the Customer data item.

## Designing The Sections

As you already know how to design the sections for a report with just one data item, the following description concentrates on showing you how to design sections that involve two data items.

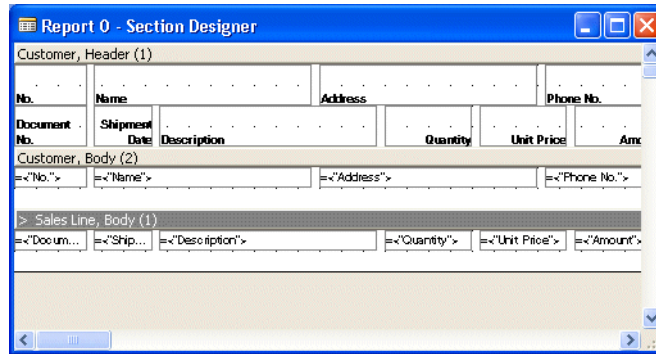
To design the sections:

- 1 When you first open the Section Designer, there is already a Body section for each data item. Click Edit, Insert New and add a Header section for the Customer data item.
- 2 Click View, Field Menu to open the **Field Menu** window and add the **No.**, **Name**, **Address** and **Phone No.** fields to the **Customer, Body (2)** section. Move the labels up into the Header section.

So far, the procedure has been exactly the same as that used for creating the first, simple report.

- 3 Open the **Field Menu** window and add the **Document No.**, **Shipment Date**, **Description**, **Quantity**, **Unit Price** and **Amount** fields to the **Sales Line, Body (1)** section.
- 4 At this point, you need to decide what to do with the labels for the controls in the **Sales Line, Body (1)** section. If they stay where they are, they will be printed for each record of the data item. If a header section is added for Sales Line, this header section will be printed each time the data item loop is entered. The data item loop starts for each record in the Customer data item.

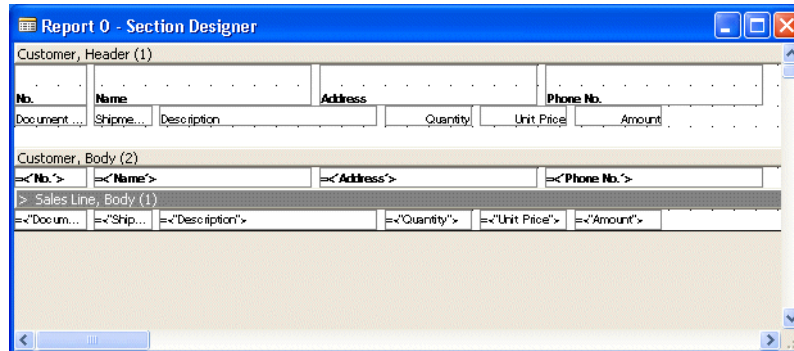
As neither of these solutions seems very good, you can take a third approach and move the labels into the header section of the Customer data item, like this:



- 5 Set the *PrintOnEveryPage* property of the **Customer, Header (1)** section to Yes.

Now, the labels for both the Customer records and the Sales Line records will be printed as column captions in the **Customer, Header (1)** section.

To make the connection between labels and data clear, the labels for the Sales Line columns can be changed to the normal font weight instead of the default bold. Change the text boxes of the Customer data item to bold, to make these records stand out among the lines that are printed. (There will be a lot more records from Sales Line than from Customer.) Furthermore, the Sales Line labels can be resized to occupy only one line, and an empty line can be added to the header section.



- Save and close the report, and run it from the Object Designer. This report should look something like this when it is run with the sample data:

The screenshot shows a 'Print Preview' window with a report. The report has the following columns: No., Name, Address, Quantity, Unit Price Excl., and Amount. The data is grouped by supplier.

No.	Name	Address	Quantity	Unit Price Excl.	Amount
<b>014545 New Concepts Furniture</b>					
101018	25-01-01 MOSCOW Swivel Chair, red	705 West Peachtree Street	6	190,036	0,00
<b>10000 The Cannon Group PLC</b>					
1001	25-01-01 Bicycle	192 Market Square	1	1.000,00	0,00
1002	25-01-01 Bicycle		1	1.000,00	0,00
1003	25-01-01 Bicycle		1	1.000,00	0,00
101016	25-01-01 ANTWERP Conference Table		1	420,40	420,40
2001	17-01-01 Manual for Loudspeakers		4	0,00	0,00
2006	17-01-01 Manual for Loudspeakers		10	0,00	0,00
2011	17-01-01 Loudspeaker, Cherry, 180W		10	129,00	1.290,00
<b>20000 Selangorlan Ltd.</b>					
101017	25-01-01 ST.MORITZ Storage Unit/Drawers	163 Thomas Drive	2	342,10	0,00

The window also shows a status bar at the bottom with 'Page 1', 'Report generation completed (3 pages)', and a 'Help' button.

## **Chapter 14**

### **Extending Report Functionality**

This chapter describes how to group and total data when creating reports in C/SIDE. It also gives an overview of the report triggers and, finally, uses some of the advanced facilities of the Report Designer.

- Grouping and Totaling
- Triggers in Reports
- Advanced Sample Reports
- Creating a Simple Document
- Creating a Nonprinting Report
- Types of Report

## 14.1 Grouping and Totaling

How you group and total data is a crucial element of creating useful reports. By grouping and totaling data, your reports can contain information that is not otherwise readily available.

The second report that you created in the previous chapter was based on two tables and listed customers and the entries from the **Sales Line** table that related to each customer. You can use grouping and totaling to enhance this information in several ways.

First, if the report is meant to provide statistics, it will be more useful if the sales lines are grouped according to the items (grouping on the item number, which identifies each item) instead of printing all the lines from all the sales documents. Each line would then contain figures for the total quantity and the total amount for each item per customer.

Second, it would be useful to have a total amount per customer, showing how much this customer has on order.

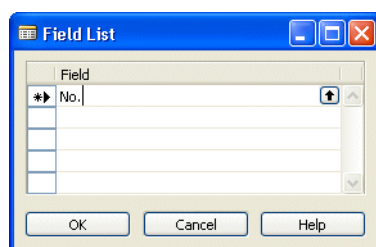
This is the report that you create in this chapter.

### Defining the Data Model

The first steps in creating this report involve designing a report similar to the advanced report that you created in the previous chapter. You can therefore begin by reopening that report in the report designer.

To add the grouping and totaling, follow these steps:

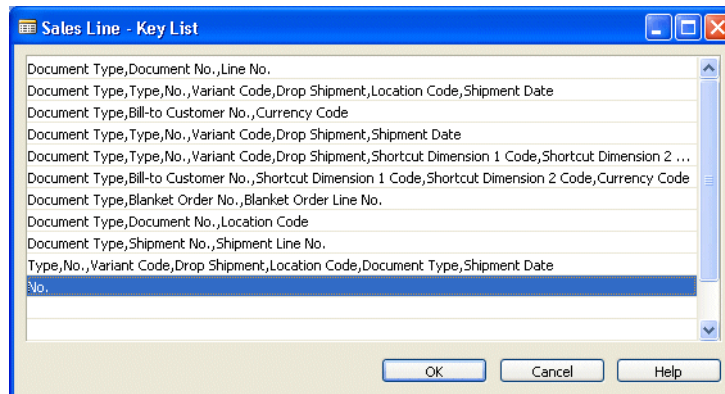
- 1 Open the **Properties** window (SHIFT+F4) of the Sales Line data item.
- 2 In the **Value** field of the *GroupTotalFields* property, enter the name of the field that you want to use to group the records. You can use the AssistButton... to help select the field. In this example, the **No.** field is used:



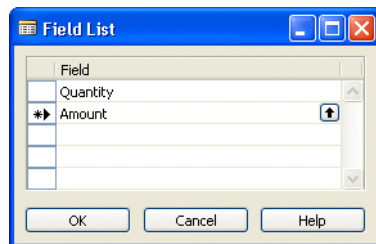
- 3 In the **Value** field of the *DataItemTableView* property, use the AssistButton... to select a key. You must select a key that contains the field that you want to group by.

If the key you select is a composite key, the grouping can fail if there are other fields in the key before the grouping field, and the contents of one of these fields changes. You may have to create a distinct key for reports that access data in ways other than those used by your application in general.

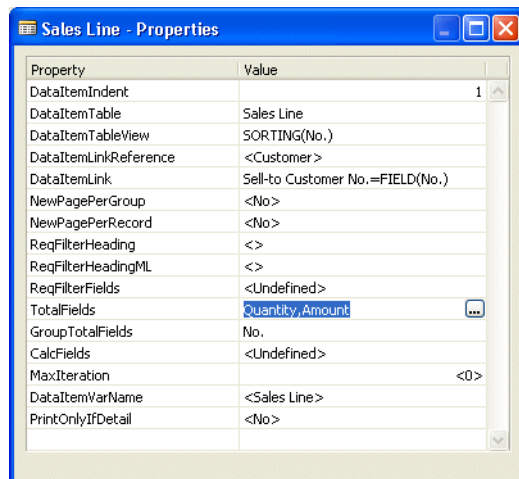
For this report, a secondary key consisting only of the **No.** field has been created for the **Sales Line** table:



- 4 In the **Value** field of the *TotalFields* property, enter the names of the fields for which totals should be calculated. Use the AssistButton... to select the fields. In this example, the **Quantity** and **Amount** fields are selected:



The **Properties** window of the Sales Line data item should now look like this:



This data model is now defined. This is what has been accomplished:

- For each record in the Customer data item, the records in the Sales Line data item that are related to this customer are selected.
- Records from the Sales Line data item are grouped according to the item number.
- Totals are maintained for the **Quantity** and **Amount** fields of the Sales Line data item.

## The Relationship between Totals and Sections

In this report, a hierarchy of data items has been established, where Customer is the data item at the highest level and Sales Line is an indented data item. Furthermore, the Sales Line records are grouped on the **No.** field, and totals are calculated for the **Amount** and **Quantity** fields.

What, then, is the relationship between these totals and the sections, and how can these totals be printed?

Until now, only the Header and Body sections have been used. To print totals, you will need to use some new sections. The following table lists all the different types of section:

Section Name	Output
Header	Before a data item loop begins and (if the <i>PrintOnEveryPage</i> property of the section is Yes) also on each new page.
Body	For each iteration of the data item loop. When there is an indented data item, the complete loop for this data item begins after the Body section of the higher level data item has been printed.
Footer	After the loop has finished, and (if the <i>PrintOnEveryPage</i> property of the section is Yes), also on each new page. Moreover, if the <i>PlaceInBottom</i> property of the section is Yes, the Footer section is printed at the bottom of the page, even if the data item loop ends in the middle of a page.
GroupHeader	A new group starts.
GroupFooter	A group ends.
TransportHeader	If a page break occurs during a data item loop, the header is printed at top of the new page. This section is printed after a possible Header section of the data item.
TransportFooter	If a page break occurs during a data item loop, this header is printed before the page break. This section is printed before a possible Footer section of the data item.

To print out the totals, use both a GroupFooter and a Footer section for the Sales Line (indented) data item.

The totals for Quantity and Amount for the defined group will be in the GroupFooter section – remember that the **No.** field was used for grouping.

When the entire data item has been iterated, the grand total can be printed in the Footer section of the Sales Line data item.

The flow in this example can be summarized as follows:

- 1 For each record of the Customer data item, a loop for the Sales Line data item is begun.
- 2 Whenever the contents of the **No.** field change, the GroupFooter is outputted.
- 3 When the Sales Line loop ends, the Footer is outputted. As the Body section of the Customer data item was printed before any section of the indented data item, the



Footer is also the last section that is printed. Therefore, this section can be used to print summary information about the customers.

The Quantity and Amount totals for each item that a specific customer has on order will be placed in a GroupFooter section of the Sales Line data item. The grand total for the Amount that the customer has on order will be placed in a Footer section of the Sales Line data item. (A Quantity total is also maintained, but this information is not too useful, since it will be a total of quantities for all kinds of different items.)

#### Note

The properties of sections, such as PrintInBottom and PrintOnEveryPage, apply to an entire data item. This means that you cannot, for example, have two Footers for a data item, one for the "normal" pages and one for the last page.

## Designing the Sections

The next step is to design the sections in the report.

To design the sections:

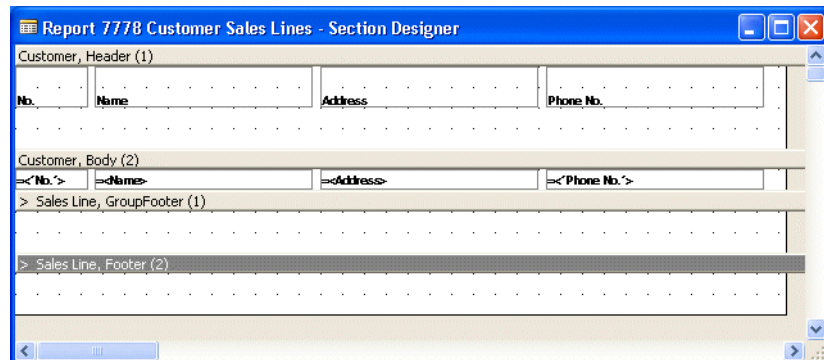
- 1 In the Report Designer, open the report that you were working on earlier.
- 2 Click View, Sections and as usual, when you open the Section Designer, a body section for each data item has been inserted.
- 3 Select the Customer data item and click Edit, Insert New to open the **Insert New Section** window.
- 4 Add a Header section. This section will be used to print headings for the columns in the report.
- 5 Select the Sales Line data item.
- 6 Click Edit, Insert New and add a GroupFooter section. This section will be used to print the summary information about each item.
- 7 Click Edit, Insert New and add a Footer section for the Sales Line data item. This section will be used to print the summary information about each customer.

In this report, nothing will be printed in the Body section of the Sales Line data item. You must therefore, delete this section.

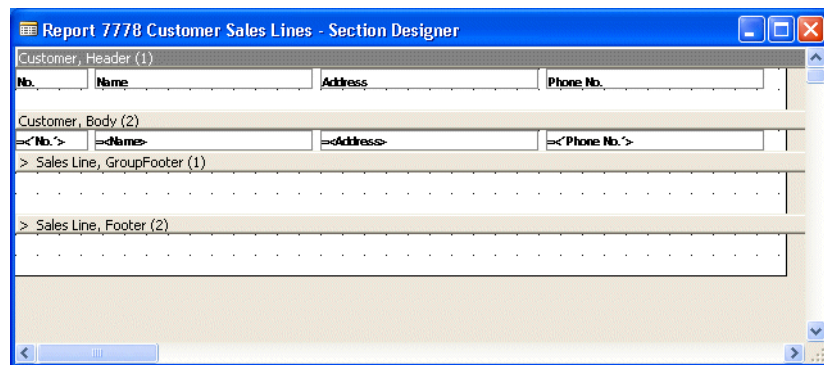
To delete this section:

- 1 Select the section bar for the Body section of the Sales Line data item and click Edit, Delete (F4) or press DELETE on the keyboard.
- 2 You are prompted to confirm that you want to delete the section.

- 3 Select the labels for the Sales Line that you added to the Customer Header earlier and delete them. The Section Designer now looks like this:



- 4 Adjust the vertical size of the labels in the Header section and of the Header section itself. The Section Designer now looks like this:

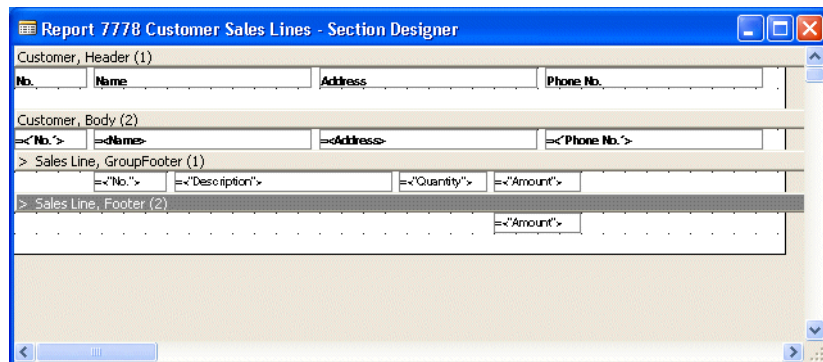


Now it is time to design the Footer sections and ensure that they display the information you want.

To design the Footer sections:

- 1 Select the GroupFooter section of the Sales Line data item (by clicking the section bar) and click View, Field Menu to open the **Field Menu** window.
- 2 In the **Field Menu** window, select the **No.**, **Description**, **Quantity** and **Amount** fields.
- 3 Click twice in the GroupFooter section of the Sales Line data item to insert these fields. Delete the labels, and resize the section vertically.
- 4 Select the Footer section of the Sales Line data item.
- 5 In the **Field Menu** window, select the **Amount** field and insert it in the footer section directly below the **Amount** field in the GroupFooter section. Select the label

and delete it. Let the section keep its default size so there is some empty space before each new customer. The Section Designer should now look like this:



- 6 Select the No. label in the Customer Header and open the **Properties** window (SHIFT+F4) and set the value of the *HorzAlign* property to Right.
- 7 As you may have noticed in the preview of the simple report that you designed in the previous chapter, the demonstration data does not contain any phone numbers for the customers. You can therefore delete the phone number fields from the report.
- 8 Save, close and run the report.
- 9 The report will look like this in the **Print Preview** window:

No.	Name	Address		
<b>014545 New Concepts Furniture</b>				
<b>705 West Peachtree Street</b>				
1980-S	MOSCOW Swivel Chair, red		6	0.00
				0.00
<b>1000 The Cannon Group PLC</b>				
<b>192 Market Square</b>				
1000	Bicycle		3	0.00
1980-S	ANTWERP Conference Table		1	420.40
LS-190	Loudspeaker, Cherry, 150W		10	1,290.00
LS-MAN-10	Manual for Loudspeakers		14	0.00
				1,710.40
<b>2000 Selangorian Ltd.</b>				
<b>163 Thomas Drive</b>				
1988-W	ST.MORITZ Storage Unit/Drawers		2	0.00
1984-W	INNSBRUCK Storage Unit/G.Door		1	0.00
1978-W	INNSBRUCK Storage Unit/W.Door		1	0.00
LS-10PC	Loudspeakers, White for PC		20	0.00
LS-120	Loudspeaker, Black, 120W		6	0.00
LS-190	Loudspeaker, Cherry, 150W		8	1,032.00
LS-2	Cables for Loudspeakers		30	210.00
LS-75	Loudspeaker, Cherry, 75W		25	0.00

Obviously, this report needs some more work before it will look very good and is truly functional. For example, you need to devise a way to place captions on the columns from the indented data item.

However, the logic works: for each customer, there is a list of items where quantities and amounts have been summarized, and the total amount for each customer is also calculated.

One desirable improvement would be to add a line at the end of the report where the grand total for all the customers is printed. To do this, however, you need to write C/AL code in a report trigger. The section "Advanced Sample Reports" on page 250 gives examples of how to do this.

## 14.2 Triggers in Reports

Although the system interprets and acts upon many events in a predefined way, certain actions cause the system to execute a user-definable C/AL function (the event *triggers* the function). In reports, triggers are typically used to perform calculations and to control whether or not to output sections. This depends, for example, on the value in a field, or a choice the user made in the request form. But the most important point about triggers is that they allow you to control how data is selected and retrieved in a more complex and effective way than you can achieve by using properties.

Reports can contain the following triggers:

### Report Triggers

These triggers apply to the report itself:

Trigger	Executed
OnInitReport	When the report is loaded.
OnPreReport	Before the report is run – but after the RequestForm has been run.
OnPostReport	After the report has run – but not if the report was stopped manually or by the Break function.
OnCreateHyperlink	After the user creates a URL to a form or a report.
OnHyperlink	After the <i>OnInitReport</i> trigger is executed for a report. The trigger executes a URL string.

### Data Item Triggers

The following triggers apply to each data item of the report:

Trigger	Executed
OnPreDataItem	Before the data item is processed, but after the associated variable has been initialized.
OnAfterGetRecord	When a record has been retrieved from the table.
OnPostDataItem	When the data item has been iterated for the last time.

### Section Triggers

These triggers apply to each of the sections of a data item:

Trigger	Executed
OnPreSection	Before processing a section.
OnPostSection	After processing a section but before printing it.

For more information about these and other triggers, see to the *C/SIDE Reference Guide* online Help.

## 14.3 Advanced Sample Reports

This section includes examples of reports that are slightly more complicated than those described earlier. These examples are not intended to be complete or ready to run, but are meant to give you some inspiration that might be useful when designing your own reports.

### Using Virtual Tables

C/SIDE includes a number of *virtual tables*, such as the **Integer** table and the **Date** table. The virtual tables are described in the chapter, "Special C/SIDE Tables" on page 109. This section shows you how to use the **Date** table, in a report.

### Using the Date Table

The **Date** table consists of five fields, **Period Type**, **Period Start** and **Period End**, **Period No.** and **Period Name**.

**Period Type** can be Date, Week, Month, and so on, while **Period Start** is the starting date of each period and **Period End** is the last date in the period. (**Period End** dates are closing dates.)

In this example, the **Date** table is used to create a report that prints information from the **Cust. Ledger Entry** table. For each day in a range of dates (that can be chosen by the user when they want to run the report), the report summarizes the entries made on that date.

The report lists the different types of document that were entered, the number of documents of each type and the total number of documents entered on each date. The report also displays the total amount entered on each date. Finally, the total number of entries and the total amount entered for the selected date range is printed at the bottom of the report.

You could create the report by grouping the entries according to the **Cust. Ledger Entry** table alone, but the field that contains the posting date in that table is not part of any key. Creating a special key just for this report is not advisable, because it would slow down all the other transactions that involve this table. Every sales entry would be affected.

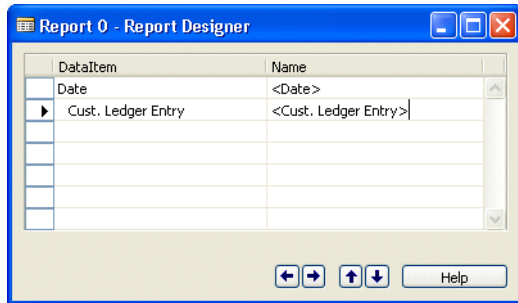
### Defining the Data Model

This data model contains two data items – one based on the **Date** table and one based on the **Cust. Ledger Entry** table.

To define the data model:

1. Open the Report Designer and create a new blank report with two data items with the **Date** and the **Cust. Ledger Entry** tables as the underlying tables.

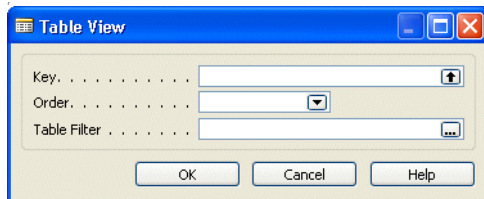
2 Indent the Customer Ledger Entry data item:



Indenting the Customer Ledger Entry data item ensures that the system only searches through the **Customer Ledger Entry** table when a date is found in the **Date** table, that lies within the range specified by the user.

3 Select the Date data item and open the **Properties** window (SHIFT+F4).

4 In the **Value** field of the *DataItemTableView* property, click the AssistButton ... to open the **Table View** window:



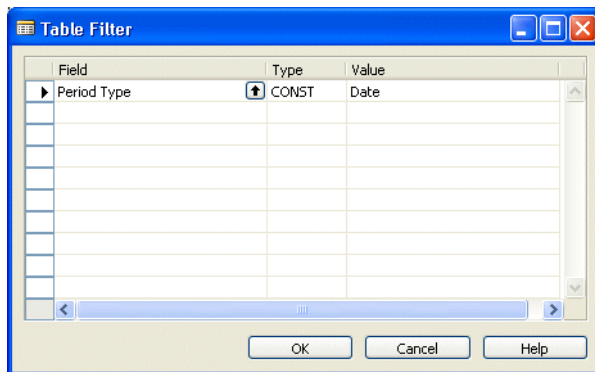
5 In the **Table Filter** field, click the AssistButton ... to open the **Table Filter** window.

6 In the **Table Filter** window, set a filter that selects records whose Period Type is Date.

**Important**

.....  
 This is an important step, as the iteration of the Date data item would otherwise run through all the records, including those for Weeks, Quarters, Months and Years.  
 .....

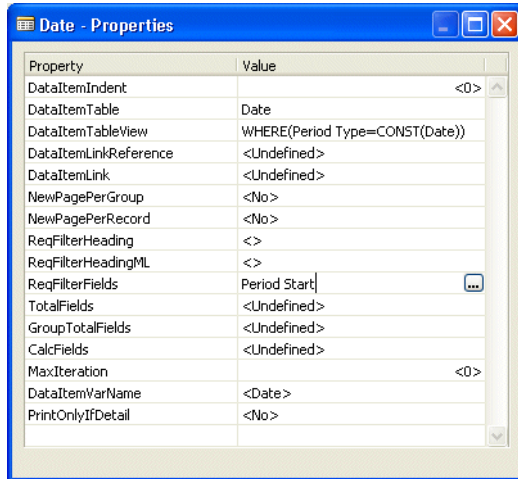
The **Table Filter** window should look like this:



7 Click OK in the **Table Filter** window and in the **Table View** window.

- In the **Properties** window of the Date data item, in the **Value** field of the *ReqFilterFields* property, enter *Period Start*. This lets the user enter a range of dates when they fill out the request form before running the report. This date range is then used to filter the information displayed in the report.

The **Properties** window should look like this:

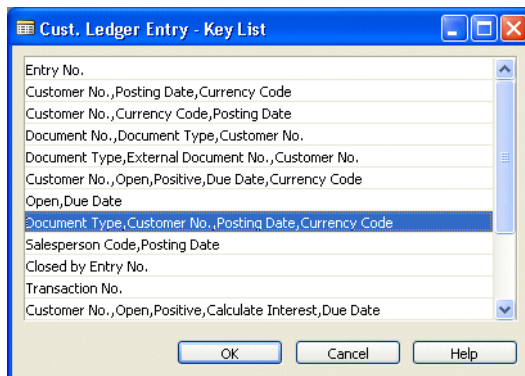


- Open the **Properties** window of the Cust. Ledger Entry data item. In the Value field of the *DataItemTableView* property, click the AssistButton ... to open the **Table View** window.

In the **Key** field, use the AssistButton ↑ to open the **Key List** window.

In this example, an appropriate key is a key that contains the **Document Type** field, as the definition of a group in this report is based on these fields. Here, a key is selected that has this field as its first component, the other fields that are included in the key are not important.

No individual entries will be printed, so they do not need to be sorted in any specific way.

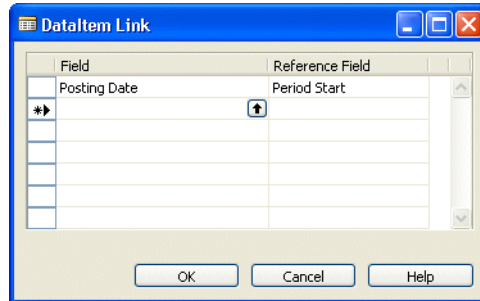


- Make sure the *DataItemLinkReference* property points to the Date data item – this is the default value.

- In the *DataItemLink* property, you must specify the field that establishes the link between the two data items. In the **Value** field of the *DataItemLink* property, click the AssistButton ... to open the **DataItem Link** window.



In the **Field** field, use the AssistButton  $\uparrow$  to select the **Posting Date** field from the Cust. Ledger Entry data item, and in the **Reference Field** field, select the **Period Start** field from the Date data item.



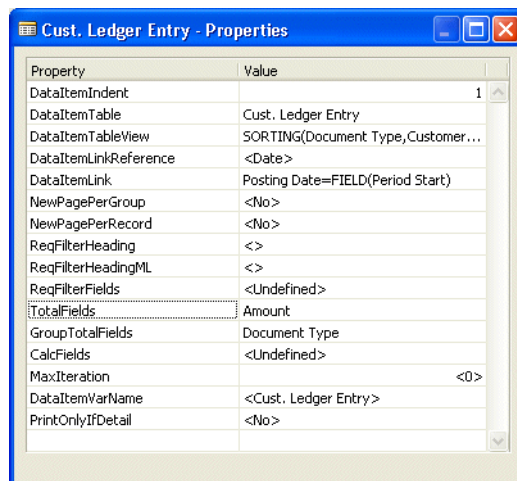
12 In the **Value** field of the *GroupTotalFields* property, use the AssistButton ... to select the **Document Type** field.

This tells the report to group the entries from the **Cust. Ledger Entry** table according to their document type – Payment, Invoice, Credit Memo, Finance Charge Memo, Reminder or Refund.

13 In the **Value** field of the *TotalFields* property, use the AssistButton ... to select the **Amount** field.

This tells the report to add up the entries in the **Amount** field for each document type for the date in question.

The **Properties** window of the Cust. Ledger Entry data item should now look like this:



So far, the report works like this:

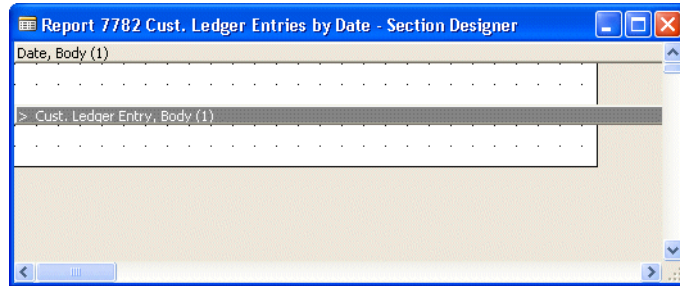
- You can select a range of dates from the request form of the report.
- The report runs through the **Date** table, with a constant filter on the **Period Type** field that selects only records whose type is *Date*. If you select a date range, only dates within this range are selected; otherwise all the dates are used.
- For each date you selected, the records in the Cust. Ledger Entry data item that were posted on that date are selected.

- The records of the Cust. Ledger Entry data item are grouped according to the value of the **Document Type** field, and the totals are maintained for the **Amount** field.

### Designing the Sections

The design of the sections is fairly straightforward.

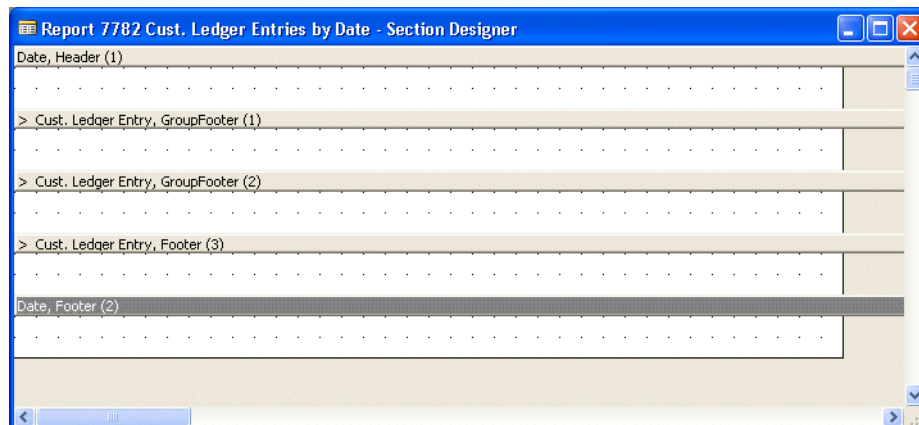
- 1 Click View, Sections to open the Section Designer:



As you can see, a body section has been created for each data item.

- 2 Click Edit, Insert New (F3) and insert a Header section for the Date data item. This header will be used to contain the labels for the columns of data in the report.
- 3 Insert two GroupFooter sections for the Cust. Ledger Entry data item. You create two GroupFooter sections for the Cust. Ledger Entry data item so that you can control their output separately. Both sections contain summarized information about the groups created by this data item.  
  
The reason for this construction, as well as how to use it, is explained later.
- 4 Insert a Footer section for the Cust. Ledger Entry data item. This section is used to display the subtotals.
- 5 Insert a Footer section for the Date data item. This section is used to display the grand totals at the end of the report.
- 6 Neither data item requires a Body section – so you can delete them.

The Section Designer should now look like this:

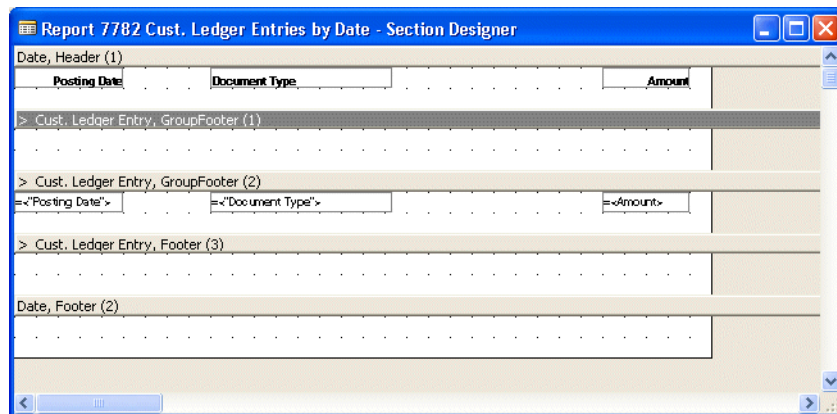


### Adding the Fields to the Report

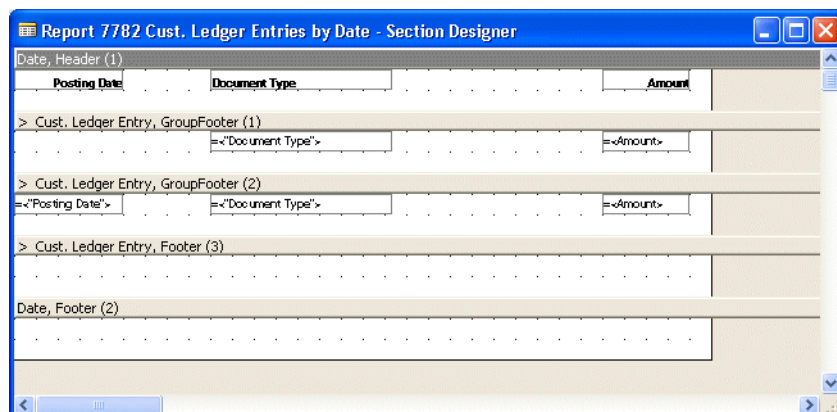
The next step is to add the fields that you want to include in the various sections of the report.

To add the fields to the report:

- 1 In the Section Designer, select the **Cust. Ledger Entry, GroupFooter (2)** section and click View, Field Menu to open the **Field Menu** window.
- 2 In the **Field Menu** window, select the, **Posting Date, Document Type** and **Amount** fields.
- 3 In the Section Designer, click twice in the **Cust. Ledger Entry, GroupFooter (2)** section to add these fields.
- 4 Move the labels into the Header section and align the fields so that they look something like this:



- 5 In the Section Designer, select the **Cust. Ledger Entry, GroupFooter (1)** section and add the **Document Type** and **Amount** fields.
- 6 Delete the labels for the fields that you have just added to the **Cust. Ledger Entry, GroupFooter (1)** section – these extra labels serve no purpose.
- 7 Align the fields so that they look something like this (you might need to enlarge the design area):



As you can see, this report is far from finished. You still need to add the fields that:

- Contain the number of documents of each type that were posted on each date.
- Calculate and display the total number of documents posted on each date.
- Calculate and display the grand total number of documents posted in the period covered by the report.
- Calculate and display the total amount posted on each date.
- Calculate and display the grand total of the amount posted in the period that the report covers.
- Display the date range that the user used to filter the information in the report.

To accomplish this you need to write a small amount of C/AL code in the triggers and to define some variables as well as add the appropriate fields.

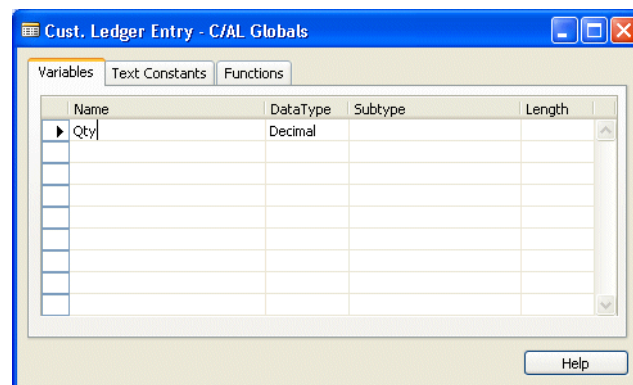
### Calculating the Number of Entries

The records in the Cust. Ledger Entry data item contain an amount and the total amount is calculated by using the properties of the data item in the report. You accomplished this in step 12 on page 253, when you set the *TotalFields* property of the Cust. Ledger Entry data item to calculate a total based on the **Amount** field in the **Cust. Ledger Entry** table. This total is calculated separately for each date and is the basis for the grand total that you will calculate later.

Unfortunately, the number of entries cannot be calculated in the same way because there is no field in the data item that contains this information. However, each record in the **Cust. Ledger Entry** table corresponds to exactly one entry and this means that the number of entries can be calculated by simply counting the records.

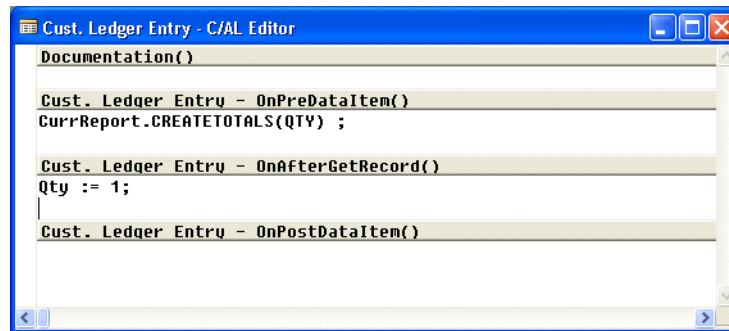
To calculate the total number of entries made on each date:

- 1 Create a global variable (here it is called Qty) and define its data type as *Decimal*:



Select the Cust. Ledger Entry data item in the Report Designer and click View, C/AL Code (F9) to open the C/AL Editor.

2 Add the following C/AL code to these triggers in the Cust. Ledger Entry data item:



The statement in the *OnPreDataItem* trigger maintains totals for the Qty variable in the same way that the *TotalFields* property specifies that totals are maintained for a field in a record. The statement in the *OnAfterGetRecord* trigger simply assigns a value of one to the Qty variable each time a record is retrieved.

You are going to use the `CREATETOTALS` function to maintain totals for each group and a grand total for the iteration of the data item loop. As data items are grouped according to the **Document Type** field, Qty contains the sum of all the entries of the same document type each time the **Cust. Ledger Entry GroupFooter** section is printed. When the Footer section is printed, Qty contains the sum of all the entries (that were selected, that is, all the records that were entered on the same date).

The variable in the argument of the `CREATETOTALS` function must be of the Decimal data type because the function is usually used to add up amounts. This is why the Qty variable was declared as a Decimal rather than an Integer (which would have been the more intuitive choice).

You must now add some fields to the report that will display the totals calculated by this code:

- 1 Open the Section Designer and open the Toolbox.
- 2 Add a text box to the **Cust. Ledger Entry, GroupFooter (1)** section and open the **Properties** window of the text box.
- 3 In the **Value** field of the *SourceExpr* property, click the AssistButton ... to open the **C/AL Symbol Menu** window.
- 4 Select the Qty variable that you just defined.
- 5 Add a label to the Header section, open the **Properties** window of the label and in the *Caption* property, enter Qty and make sure that the *HorzAlign* property is set to *Right*.

The Qty variable that you have just defined is a decimal. You must therefore change the formatting of the Qty text boxes in the sections so that they do not show any decimal places. The format of the Qty text boxes are defined as <2:2> by default when the SourceExpr is of the Decimal data type.

To change the formatting of the Qty text box:

- 1 Select the text box in the Section Designer and open the **Properties** window.

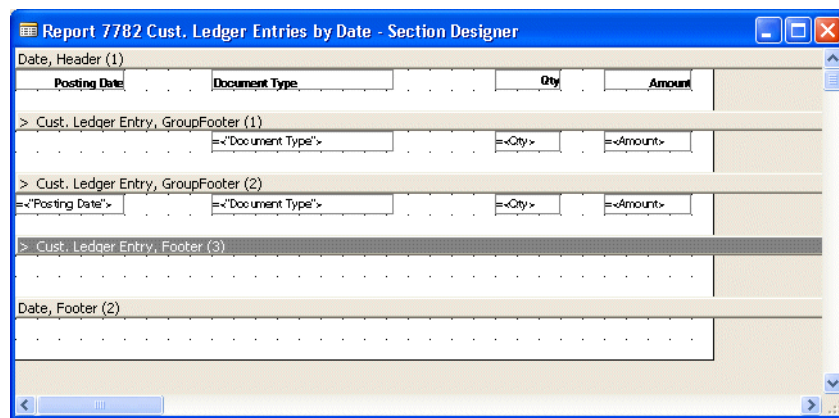
- 2 In the **Value** field of the *DecimalPlaces* property change the value to 0:0. The field will no longer display any decimals.

The next step is to add the field that will contain the total number of documents of each type that are entered on each date that is printed in the report.

- 1 Copy the Qty text box that you just added to the **Cust. Ledger Entry, GroupFooter (1)** section.
- 2 Paste it into the **Cust. Ledger Entry, GroupFooter (2)** section.

This text box will now display the total number of documents of each type entered on each date.

The Section Designer should now look something like this:



The report now contains fields that list the entries in the **Cust. Ledger Entry** table according to the date when they were entered and tells you how many documents of each type were entered on that date.

### Calculating the Total Number of Entries and the Total Amount Per Date

The next step in designing this report is to ensure that it displays some subtotals that tell you how many entries were made on each date, as well as the total amount entered on each date.

This is why you created the **Cust. Ledger Entry, Footer (3)** section.

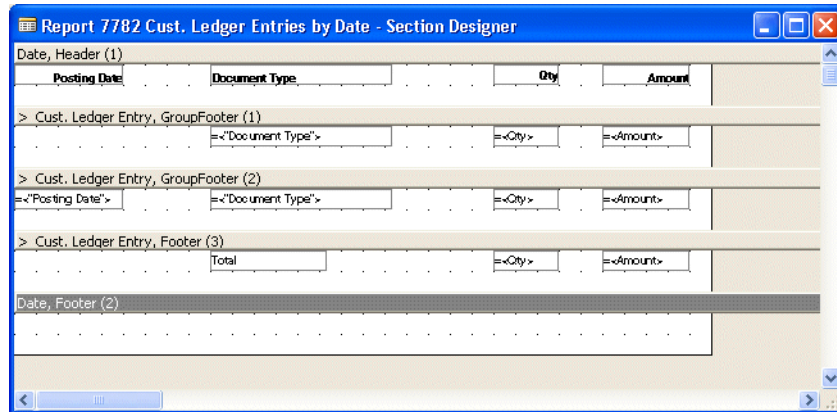
To calculate these subtotals:

- 1 Copy the **Qty** and **Amount** fields from the **Cust. Ledger Entry, GroupFooter (2)** section into the **Cust. Ledger Entry, Footer (3)** section.

These fields will now display the total amount entered per date and the total number of documents entered on the date in question.

- 2 Add a label to the **Cust. Ledger Entry, Footer (3)** section, open the **Properties** window of the label and in the **Value** field of the *Caption* property enter *Total*.

The Section Designer should now look something like this:



### Printing the Date in the First Iteration Only

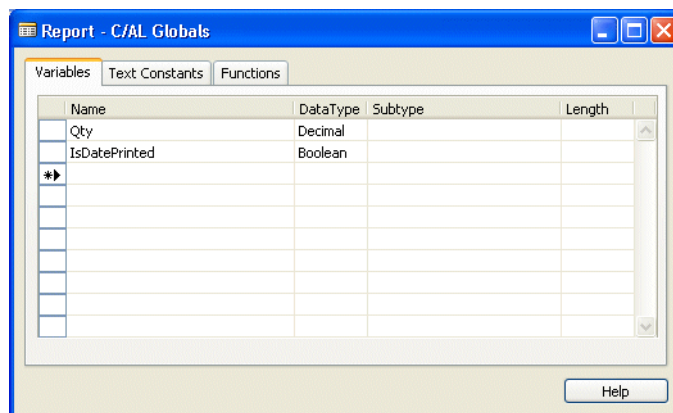
In each iteration of the Date data item loop, you want to print the Date when the information selected from the Cust. Ledger Entry data item was entered. We could, of course, just have used the body section of the Date data item and printed the date there (the **Period Start** field of the data item), but this would mean that the date would be printed on a line by itself. Instead, we would like the date to be printed along with the other information on the first line that comes from the Cust. Ledger Entry data item.

One solution is to use the **Date** field from the Cust. Ledger Entry data item. Unfortunately, this creates another problem. If it is added to the GroupFooter section, the date will be printed on every line. While this would be an easy way to solve the problem, the finished report will not be very attractive and will be cluttered with redundant information.

A better solution is to define *two* GroupFooter sections for the Cust. Ledger Entry data item. One that includes the **Date** field and one that does not, and then control when they are output. This is why we created two GroupFooter sections earlier.

To add a date and control when the different GroupFooter sections are used:

- 1 Create a global variable called `IsDatePrinted` and define its data type as Boolean.



- 2 Add the following C/AL code to the *OnPreDataItem* trigger of the Cust. Ledger Entry data item:

```
IsDatePrinted := FALSE;
```

This code initializes the *IsDatePrinted* variable, with the value *FALSE*, before each iteration of the data item loop.

- 3 Add the following C/AL code to the *OnPreSection* trigger of the **Cust. Ledger Entry, GroupFooter (1)** section of the Cust. Ledger Entry data item:

```
IF IsDatePrinted THEN
    CurrReport.SHOWOUTPUT(TRUE)
ELSE
    CurrReport.SHOWOUTPUT(FALSE);
```

- 4 Add the following C/AL code to the *OnPreSection* trigger of the **Cust. Ledger Entry, GroupFooter (2)** section:

```
IF IsDatePrinted THEN
    CurrReport.SHOWOUTPUT(FALSE)
ELSE BEGIN
    CurrReport.SHOWOUTPUT(TRUE);
    IsDatePrinted := TRUE;
END
```

These two pieces of code ensure that:

- When a new iteration of the Cust. Ledger Entry data item begins, a date is not printed.
- If the loop generates any output at all, only the second GroupFooter section (containing the **Date** text box) is included as output in the first iteration.
- If additional output is generated, only the first GroupFooter section (without the **Date** text box) is printed.

### Calculating the Grand Total Amount and the Grand Total Quantity

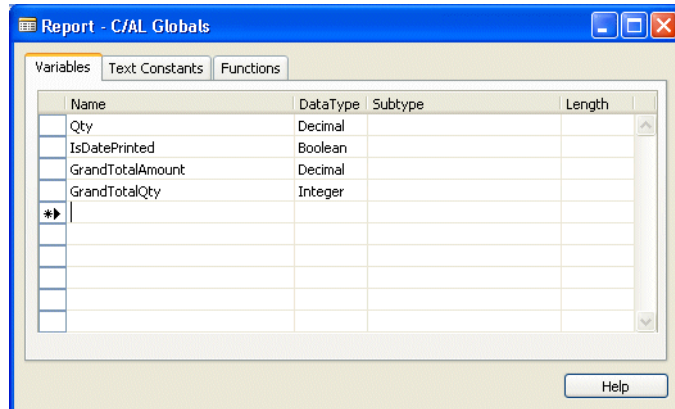
To complete this report you must ensure that it displays the total number of documents posted as well as the grand total of the amount posted in the period covered by the report. It should display this information at the end of the report – in the **Date Footer (2)** section.

The next task is to calculate the grand total of all the amounts that are printed in the report. However, these amounts come from the Cust. Ledger Entry data item, not the Date data item. This means that you cannot use the *TotalFields* property of the Date data item to do the totaling.



To calculate these grand totals:

- 1 Declare two global variables: GrandTotalAmount and GrandTotalQty of data type Decimal and Integer respectively:



- 2 Add the following lines of code to the *OnAfterGetRecord* trigger of the Cust. Ledger Entry data item:

```
GrandTotalQty := GrandTotalQty + 1;
GrandTotalAmount := GrandTotalAmount + Amount;
```

The first line simply adds one to the GrandTotalQty variable whenever a record is retrieved, while the second line adds the retrieved Amount to the GrandTotalAmount.

- 3 Open the Section Designer and add two text boxes to the **Date, Footer (2)** section.
- 4 The first text box will be used to calculate the total number of documents in the report.  
Place it under the other **Qty** fields and open the **Properties** window of the text box.
- 5 In the **Value** field of the *SourceExpr* property, click the AssistButton ... to open the **C/AL Symbol Menu** window.
- 6 Select the GrandTotalQty variable that you just defined.
- 7 The second text box will be used to calculate the grand total of the amounts displayed in the report.  
Place it under the other **Amount** fields and open the **Properties** window of the text box.
- 8 In the **Value** field of the *SourceExpr* property, click the AssistButton ... to open the **C/AL Symbol Menu** window.
- 9 Select the GrandTotalAmount variable that you just defined.  
This text box will now display the grand total of all the amounts in the report.
- 10 Add a label to the **Date, Footer (2)** section and place it under the **Total** field in the **Cust. Ledger Entry, Footer (3)** section.
- 11 Change the caption of this label to Total.

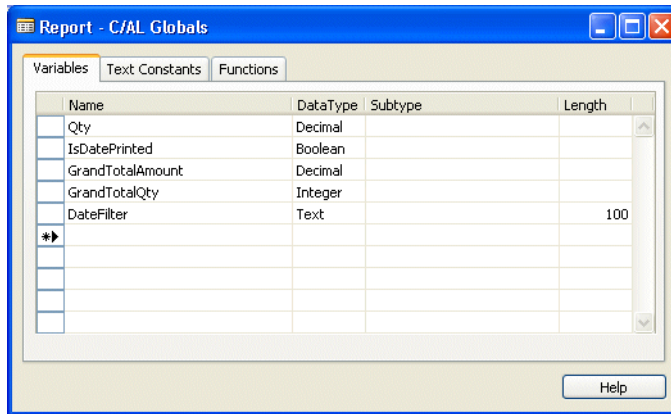
The **Date, Footer (2)** section will now contain the grand totals for the report.

**Printing the Selected Range of Dates**

The posting dates are used as a kind of header in the left margin of the report, so the report would look good if the final line could display the range of dates that the user selected before they ran the report.

This is easy to implement:

- 1 Create a variable called DateFilter and define its data type as Text, with a length of 100.



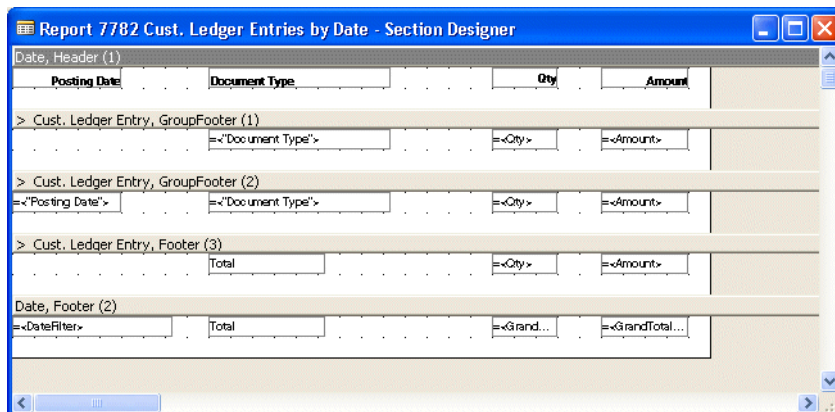
- 2 Add the following C/AL code to the *OnPreReport* trigger of the report:

```
DateFilter := Date.GETFILTER("Period Start");
```

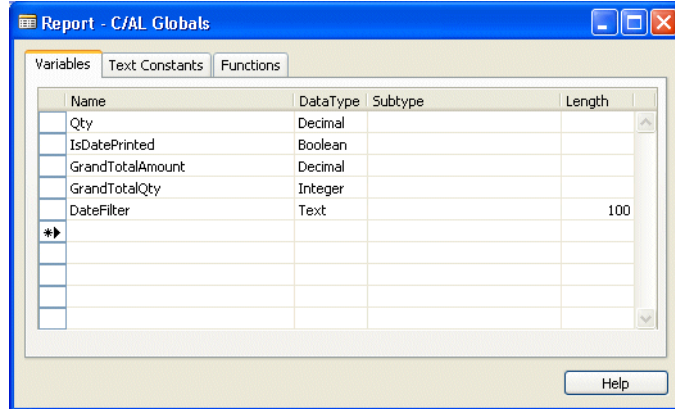
- 3 Add a text box to the footer section of the Date data item, open the **Properties** window of the text box and enter *DateFilter* as its source expression.

When the *OnPreReport* trigger is executed, the RequestForm has already been run. The GETFILTER function returns any filters on the field, which are passed as an argument, as a text string.

The report should be finished at this stage and the **Section Designer** window should look something like this:

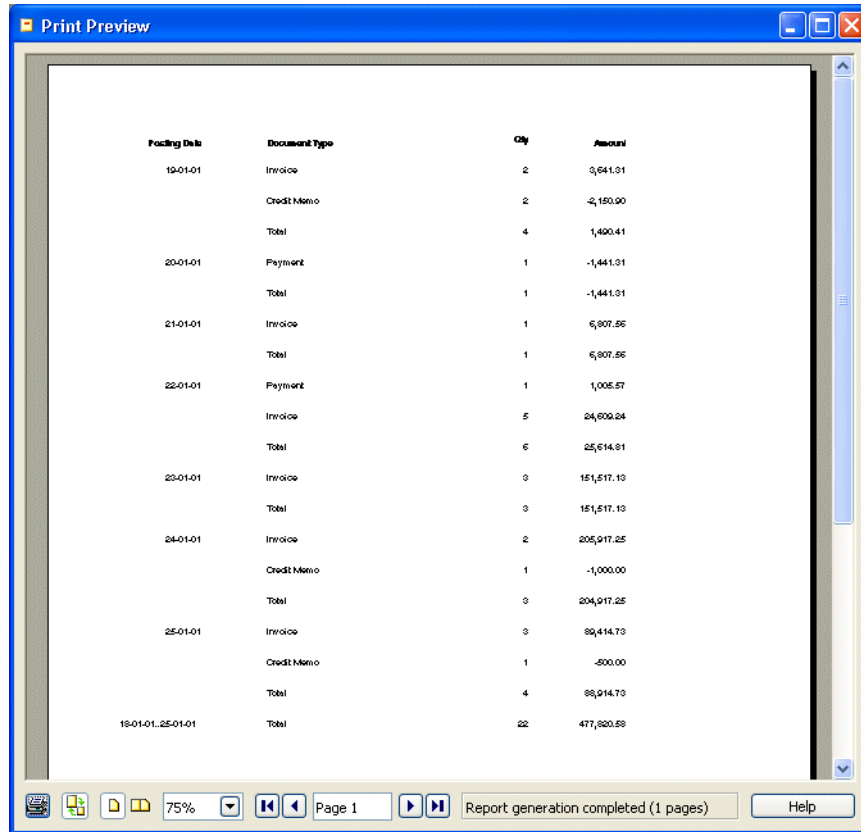


The **C/AL Globals** window should look something like this:



The final report

When you run the final report with the sample data, it should look something like this:



## 14.4 Creating a Simple Document

This section describes how to use the Report Designer to create a document. The example used is a simple sales invoice that does not take the complexities of VAT calculations into account and does not test a number of conditions that would have to be tested in a real-life situation. Furthermore, it does not print out all the information you would expect to find on an invoice.

### Defining the Data Model

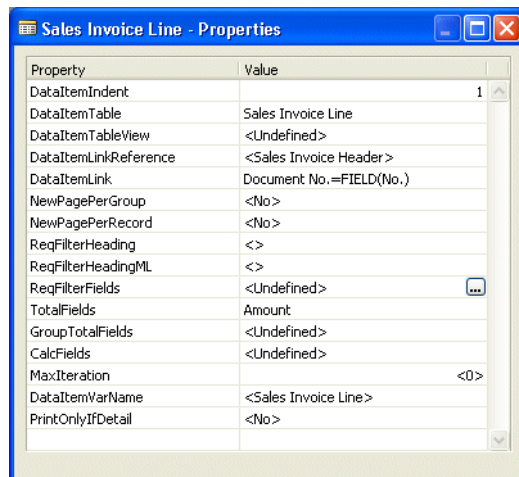
The two primary tables involved in creating a sales invoice are the **Sales Invoice Header** and the **Sales Invoice Line** tables. Some supporting tables are used to expand the codes used in the invoice tables to more descriptive texts (Payment Terms, Shipment Method), and the **Company Information** table is used to retrieve information about the company that is preparing the invoice.

The **Sales Invoice Header** table contains general information about each sales invoice that has been posted, while the **Sales Invoice Line** table contains the individual lines that make up each invoice. The tables are related through a field that is called **No.** in the header table (it is the primary key of this table) and **Document No.** in the lines table.

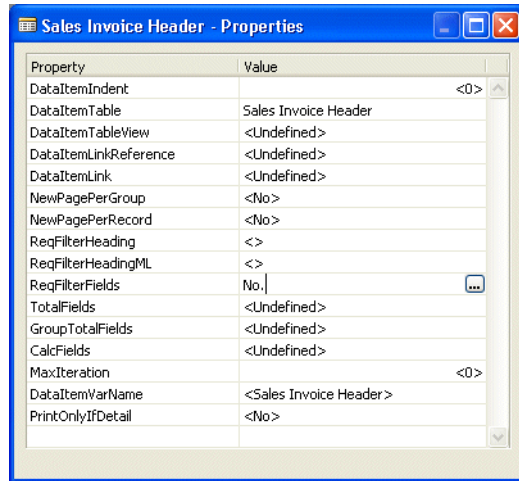
To define the data model:

- 1 Open the Object Designer and create a new blank report.
- 2 In the Report Designer, create a data item, based on the **Sales Invoice Header** table.
- 3 Create another data item, based on the **Sales Invoice Line** table, and indent this data item one level.
- 4 Open the **Properties** window of the Sales Invoice Line data item (SHIFT+F4). By default, the *DataItemLinkReference* property of the Sales Invoice Line data item points to the Sales Invoice Header data item. Leave it like this, and set the value of *DataItemLink* property to `Document No.=FIELD(No.)`.
- 5 Enter the **Amount** field as the value of the *TotalFields* property of the Sales Invoice Line data item. This calculates the total amount for all the lines in the invoice.

The **Properties** window of the Sales Invoice Line data item should now look like this:



- 6 Finally, change to the **Properties** window of the Sales Invoice Header data item. Enter the **No.** field as the value of the *ReqFilterFields* property. This lets the users of the report select a posted invoice to print.

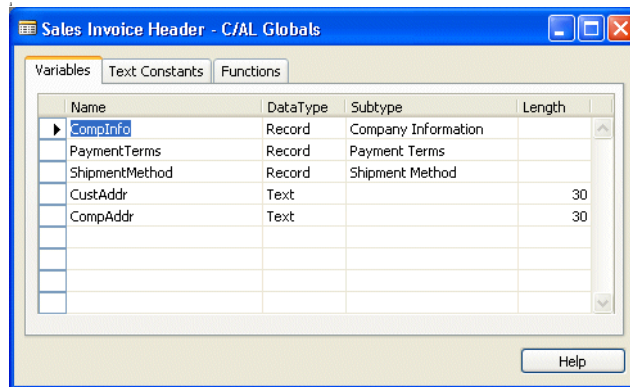


The data model has now been defined.

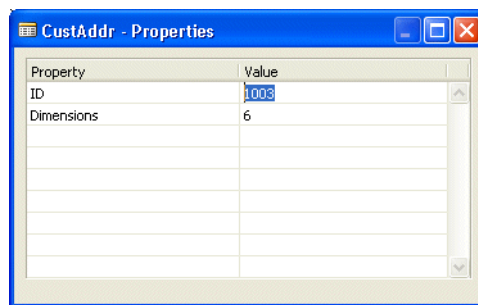
In this report, some supporting variables are needed to access information from tables that cannot be fitted into the data model.

To create the variables:

- 1 Click View, C/AL Globals.
- 2 Declare the variables as shown in the following picture:



- 3 The two last variables must be declared as arrays. Open the **Properties** window of each variable and set Dimensions to 6 for the variable called CustAddr, and to 4 for the variable called CompAddr.



The data model has now been defined. Next, a small amount of C/AL code must be added to the report triggers.

### Using the Triggers

This basic version of the report needs a very limited amount of C/AL code in order to function correctly.

The code must be entered in the triggers of the Sales Invoice Header data item.

To enter the code:

- 1 Select the Sales Invoice Header data item and click View, C/AL Code (F9).

The following picture contains all the code that is needed:

```

Documentation()

Sales Invoice Header - OnPreDataItem()
CompInfo.GET;
CompAddr[1] := CompInfo.Name;
CompAddr[2] := CompInfo.Address;
CompAddr[3] := CompInfo."Address 2";
CompAddr[4] := CompInfo.City;
COMPRESSARRAY(CompAddr);

Sales Invoice Header - OnAfterGetRecord()
CustAddr[1] := "Bill-to Name";
CustAddr[2] := "Bill-to Name 2";
CustAddr[3] := "Bill-to Contact";
CustAddr[4] := "Bill-to Address";
CustAddr[5] := "Bill-to Address 2";
CustAddr[6] := "Bill-to City";
COMPRESSARRAY(CustAddr);
PaymentTerms.GET("Payment Terms Code");
ShipmentMethod.GET("Shipment Method Code");

Sales Invoice Header - OnPostDataItem()

```

The code in the *OnPreDataItem* trigger works as follows:

- The first line, `CompInfo.GET`, retrieves a record – in fact, the only record – from the **Company Information** table.
- The next four lines assign the contents of a field in the record in the **Company Information** table to an element of the `CompAddr` array.
- The final line of the *OnPreDataItem* trigger uses the `COMPRESSARRAY` function with the `CompAddr` array as an argument to eliminate empty elements from the array. You do this because you cannot be certain that all the fields in the retrieved record contain values. If you just printed each field on a separate line, an empty field would cause an empty line to be printed.

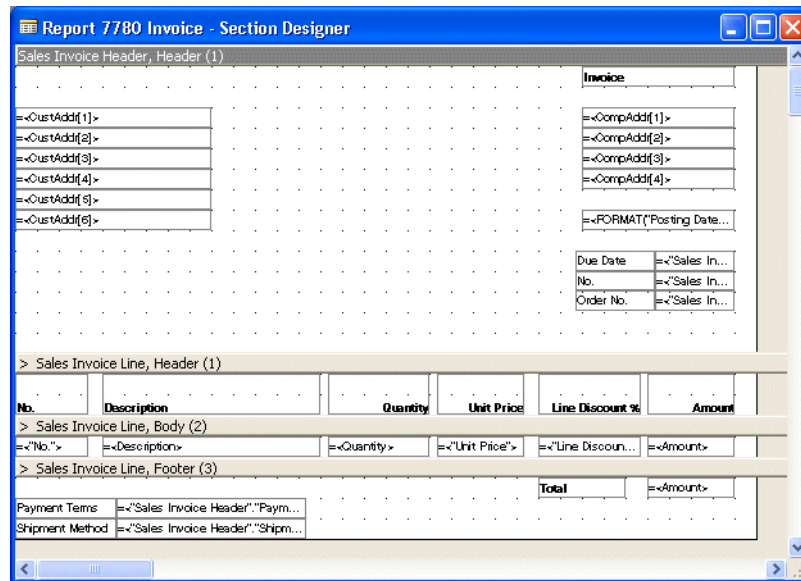
The code in the *OnAfterGetRecord* trigger works like this:

- The first six lines assign values from the record in the Sales Invoice Line data item to elements of the `CustAddr` array.
- After this, `COMPRESSARRAY` is used to eliminate empty elements from the array as described earlier.
- The last two lines use the `GET` function (with the codes for **Payment Terms** and **Shipment Method** from the Sales Invoice Header record as arguments) to retrieve the related records from the **Payment Terms** and **Shipment Method** tables. When you design the sections, the full text descriptions can then be extracted from these records.

## Designing the Sections

Now that you have defined the data model and written C/AL code to retrieve supporting information, you can design the sections.

The following picture shows the Section Designer after the necessary sections have been inserted and the relevant controls added to the sections:



In the Header section of the Sales Invoice Header data item, you should notice the following points:

Six text boxes have been inserted with `CustAddr[1] . . CustAddr[6]` as source expressions. If you compare it with the document reproduced next, you will see that in this particular invoice only four of these array elements contain data. Using `COMPRESSARRAY` has removed the empty fields and tightened up the data, so to speak.

- Similarly, in the invoice shown next, only three of the four elements of the `CompAddr` array contain data.
- The text box that prints the posting date does not have the **Posting Date** as its direct source expression. Instead, the source expression is the C/AL expression `FORMAT( "Posting Date" ,0,4)`, which, in the example here, formats the date as 22. January 2001.
- In the Footer section of the Sales Invoice Line data item, the **Amount** field is a totaled field, containing the total of all the amounts printed in the Body sections.
- In the same section, the full text descriptions of Payment Terms and Shipment Method are printed using:

`"Sales Invoice Header"."Payment Terms Code"`

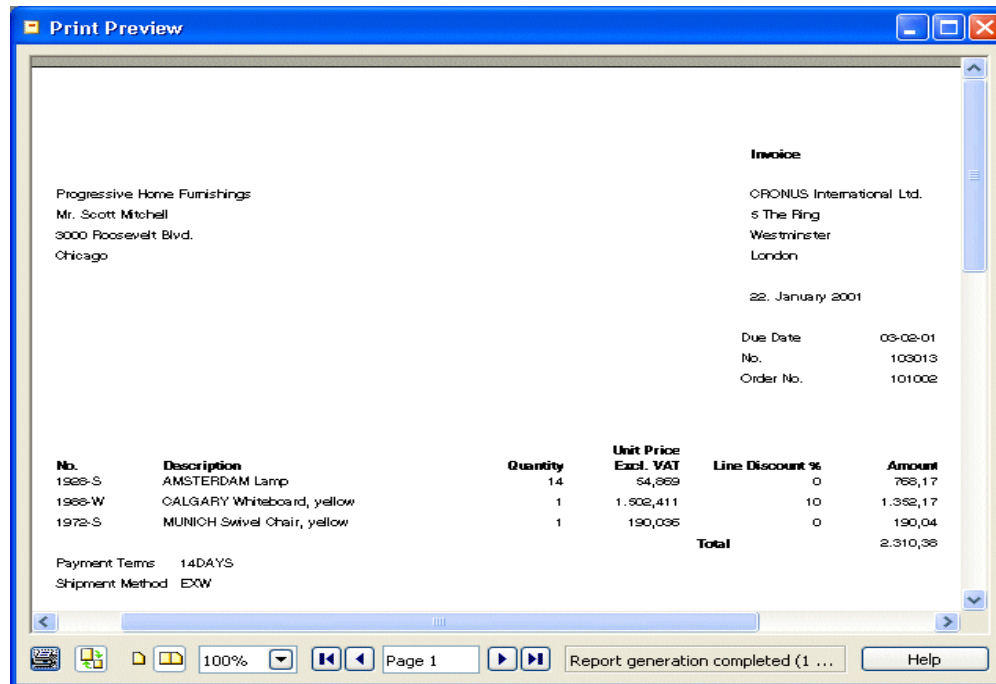
and

`"Sales Invoice Header"."Shipment Method.Code"`

as the source expressions, respectively.



This is how the invoice document looks when sample data is used:



## 14.5 Creating a Nonprinting Report

The last report that you are going to create is a non-printing report. Although you can achieve the same functionality by writing a codeunit, there are several good reasons for using non-printing reports whenever you can:

- The functionality that is available through a request form (that prompts for options and filters) is achieved with little effort, while recreating this functionality in a codeunit is a considerable task involving a deal of C/AL code.
- Using the features of the Report Designer to prompt for options and to set filters ensures consistency in the application that you are creating.
- Instead of writing C/AL code to open tables and retrieve records, you just define a data item.

This report is a simple one: it adjusts prices in the *Item* table. Users can set filters on some of the fields in the table to select a range of items by number, by posting group or by vendor, and can choose the factor by which to adjust the prices.

### Defining the Data Model

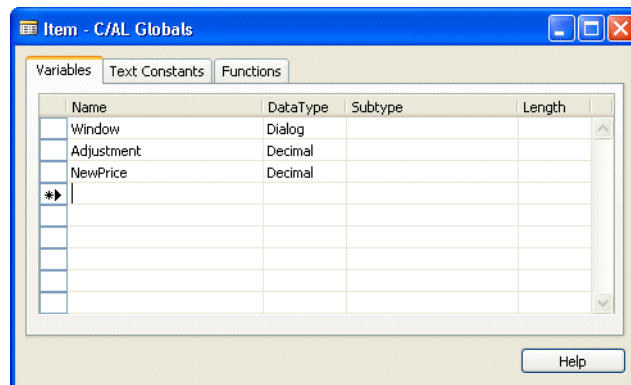
This report has one data item, based on the *Item* table.

To define the data model:

- 1 Open the Report Designer and create a new blank report.
- 2 Create a data item based on the *Item* table.
- 3 Open the **Properties** window of the report and set the value of the *ProcessingOnly* property of the report to Yes.
- 4 Open the **Properties** window of the data item and use the AssistButton ... to set the value of the *DataltemTableView* property to No.. Though this is not strictly necessary for the functionality of the report, it removes the Sort ... button from the request form that is presented to the user when they want to run the report. As the report will not print anything, the order in which the system runs through the data items is irrelevant.
- 5 Use the AssistButton ... in *ReqFilterFields* property to select the fields that the users can filter by:



6 Declare the following three variables:



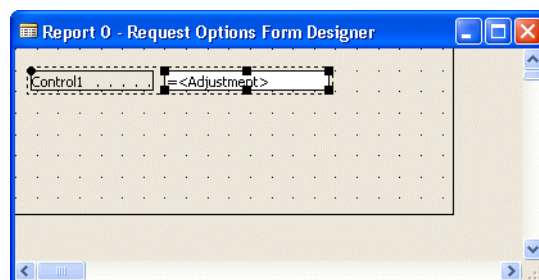
- The Window variable, of data type Dialog, prints a screen message.
- The Adjustment variable is used for the value that the users enter in the request form.
- NewPrice is used to store an intermediate result.

### Creating the Request Form

In the previous procedure you created a request form that contains a tab where you can set filters on some of the fields in the data item. You must now add an **Options** tab, where you can define the adjustment factor.

To create an **Options** tab:

- 1 Click View, Request Form to open the Request Options Form Designer.
- 2 Add a text box with a label to the form (to have the label added automatically, click the Add Label button in the Toolbox before selecting the Text Box tool).
- 3 In the **Properties** window of the text box, set the value of the *SourceExpr* property to Adjustment, the variable you have just created:



### Using the Triggers

Now that you have defined the data model and designed the request form, add a small amount of C/AL code to the triggers of the Item data item to perform the actual price adjustment.

The following picture shows all the necessary C/AL code:

```

Documentation()
Item - OnPreDataItem()
Window.OPEN('Processing item #1#####');
IF Adjustment = 0 THEN
    CurrReport.QUIT;

Item - OnAfterGetRecord()
Window.UPDATE(1,"No.");
NewPrice := Adjustment * "Unit Price";
VALIDATE("Unit Price",NewPrice);
MODIFY;

Item - OnPostDataItem()

```

The code works like this:

- The first statement in the *OnPreDataItem* trigger opens a progress window that shows the progress of the report as it is run. Because the report is non-printing, the usual printing progress window is not shown. If the table is large, the report may run for a while. Therefore, it is a good idea to tell the user that something is happening and give them an idea of how it is progressing.
- The first statement in the *OnAfterGetRecord* trigger enters the item number in the window each time a new record has been retrieved.
- The second statement in the *OnPreDataItem* simply ends the report without doing any processing if the adjustment factor is 0 (zero). If the adjustment factor were allowed to be zero, all the prices in the table would be set to zero and this would never be the intention. The statement used here is a crude way of handling this. In a more polished version, you would, for example, have a chance to reenter the adjustment factor (or be notified of the reason for quitting the report).
- The last three lines in the *OnAfterGetRecord* trigger actually update the prices. First, the adjusted value is assigned to the `NewPrice` variable. Then, the `VALIDATE` function of the **Unit Price** field is used to update the price. In this way, any special processing (for example, updating of related fields) in the *OnValidate* trigger of the table field is performed. Finally, the `MODIFY` function is used to commit the change.

When you run this report the request form looks like this:

Field	Filter
No.	
Inventory Posting Group	
Vendor No.	

In the **Filter** field for No., use the AssistButton  to specify the item whose price you want to change.

In the **Filter** field for Inventory Posting Group, use the AssistButton↑ to specify the posting group of the item whose price you want to change.

In the **Filter** field for Vendor No. use the AssistButton↑ to specify the Vendor of the item whose price you want to change.

On the **Options** tab, specify the amount by which you want to change the price.

## 14.6 Types of Report

In this section, you learn about the types of reports that are found in a normal functional area. The types of reports are not nearly as formal as the types of forms or tables. This is just a brief outline that will help you understand the system architecture.

### List Reports

A list report contains a single data item that corresponds to the table that is listed. The table can either be a Master table or a Supplemental table.

Each column contains a field from the table, and the data is printed from that table, not brought in from other tables or calculated from other tables.

The name of the report is usually the name of the table followed by the word "List".

Examples:

- Customer - List
- Insurance - List
- Vendor - List

### Test Reports

A test report is a report that is printed from a Journal table. Its purpose is to test each of the lines in the journal according to the same criteria that will be used for Posting. This ensures that any errors that exist can be found and fixed before the lines are posted.

This is useful because if an error is found during posting, processing stops and the error must be fixed before posting can be resumed. Running a test report is a good way to identify these errors.

The name of the report is usually the name of the corresponding Journal form followed by the word "Test".

Examples:

- General Journal - Test
- Resource Journal - Test

### Posting Reports

A posting report can be printed as part of the "post and print" option on a Journal. It is actually a report that is printed from the Register, and has the same name as that Register. It lists all the transactions (that is, Ledger Entries) that have been posted into that Register.

Examples:

- G/L Register
- Vendor Register

### Transaction Reports

A Transaction report contains two data items. The first is a master table, and the second is the corresponding ledger table. Normally, a transaction report lists all of the ledger entries for each record in the ledger table. Normally, there is a subtotal for each master table record, and a grand total for all the tables that are printed.

This type of report is used to view all the transactions for a particular master record. There is no standard name for this kind of report.

Examples:

- Trial Balance
- Vendor - Trial Balance

### **Other "Normal" Reports**

Reports are more loosely defined than other application objects because they are so often customized for a particular client. However, most reports do consist of a tabular list with records listed horizontally and each field displaying in its own column. There is often some sort of group heading or total to split the lines among various categories and subtotal the lines according to the categories.

Examples:

- Vendor Information
- Item Sales by Customer

### **Document Reports**

Document reports are different from most other reports, in that, many of the fields are not displayed in columns.

An example of this kind of report is an Invoice, where the header information is printed as though filling out an invoice document and this header information is repeated at the top of each page and no page contains information from more than one header.

The lines for the invoice are printed out like a normal report in rows and columns. The lines correspond to the header on the same page, and lines from different invoices are not displayed on the same page.

Examples:

- Sales Invoice No.s
- Purchase Invoice No.s





**Part 6**  
**Codeunits**



## **Chapter 15**

### **Codeunit Fundamentals**

This chapter explains what a C/SIDE codeunit is and how to create one. It also shows you how to use the functions in a codeunit from other application objects.

- What Is a C/SIDE Codeunit?
- Creating Codeunits
- Using Codeunits

## 15.1 What Is a C/SIDE Codeunit?

In earlier parts of this book you have seen examples of C/AL code used in forms. This code was always stored in the form. In simple applications the normal approach is to place the code in the object that calls the functions. However, as your application grows, in both size and complexity, you will often find that you use the same functions again and again in many different objects. Instead of declaring the same functions over and over again, it would be useful if you only had to define them once. This is where the codeunit comes in. A codeunit is a container for C/AL code that you can use in many application objects.

In codeunits you can define:

**Functions** A function is a sequence of C/AL statements that you define to create new functionality.

**Local Variables** Within each function you can define variables whose scope is limited to the function in which they are defined. These are known as local variables.

**Global Variables** A global variable is a variable whose scope covers all the functions in the codeunit.

**Temporary Tables** A temporary table is a table that is not stored in the database. Temporary tables are used mainly as structured variables that hold data temporarily while you work on it.

Each function you add to a codeunit is shown in a separate section when you view the file in the C/AL Editor.

```

Codeunit 2 Company-Initialize - C/AL Editor
Documentation()

OnRun()
Window.OPEN(Text000);

WITH GLSetup DO
  IF NOT FIND('--') THEN BEGIN
    INIT;
    INSERT;
  END;

WITH CompanyNotesSetup DO
  IF NOT FIND('--') THEN BEGIN
    INIT;
    "Company Note Heading" := Text060;
    "Writing Link Text" := Text061;
    "Editing Link Text" := Text062;
    INSERT;
  END;

WITH SalesSetup DO

```

When you add your own functions they are shown here

Every codeunit contains two default sections called **Documentation** and **OnRun**. In the **Documentation** section, you can add descriptive information that covers such things as the purpose of the codeunit, a version number and so on. In the **OnRun** section, you can include code that you want the system to execute when the codeunit is run.

**Codeunits Contain Functions But Can Also Be Run**

.....

Besides being a container for functions that can be run individually, a codeunit can itself be run by writing `<Codeunitname>.Run`. When you run a codeunit, it is the code in the **OnRun** section of the codeunit that is executed.

.....

## 15.2 Creating Codeunits

You use the Object Designer to create a new codeunit or to modify an existing codeunit.

To create a codeunit:

- 1 Open the Object Designer (SHIFT+F12) and click Codeunit.
- 2 Click New to create a new codeunit. The C/AL Editor opens and this is where you can create functions.

To modify an existing codeunit:

- 1 Open the Object Designer (SHIFT+F12) and click Codeunit
- 2 Select the codeunit you want to modify and click Design. The C/AL Editor opens and you can modify the codeunit by changing the existing functions or adding new functions.

### Using the C/AL Editor

The C/AL Editor is designed to make it easy to create and modify C/AL code. When you are working in the C/AL Editor, you have access to the C/AL Symbol Menu that helps you define C/AL functions. From the C/AL Symbol Menu, you can get help about all the C/AL commands. Select the C/AL function name in the column to the right and press F1. Read more about the C/AL Symbols Menu in the section "Using the C/AL Symbol Menu" on page 287.

When you create a codeunit, the C/AL Editor contains the two default sections described earlier (the **Documentation** and the **OnRun** sections).

You can open as many codeunits as you like. Each time you create a new codeunit or open an existing one, it is displayed in a separate window. This makes it easy to cut and paste lines of code between the codeunits.

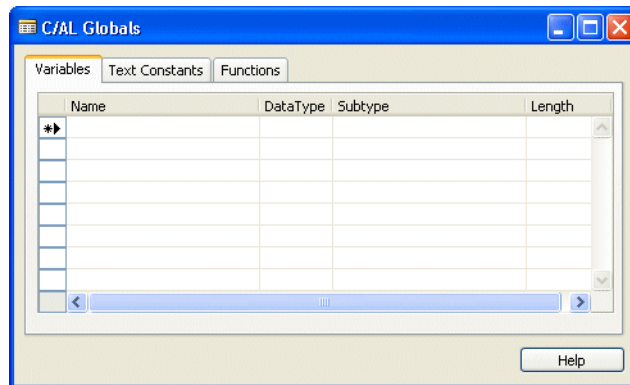
If you have used other Windows editors, you'll find the C/AL Editor easy to use. You can access the editing functions - Cut, Copy and Paste from either the Edit menu or by using the buttons on the toolbar. You can also use the standard shortcut keys:

To...	Press...
cut the selected text to the clipboard.	CTRL+X
copy the selected text to clipboard.	CTRL+C
paste the text at the clipboard into the codeunit at the cursor position.	CTRL+V
open the <b>Find</b> window to search for trigger names.	CTRL+F

### Defining Variables, Text Constants and Functions in Codeunits

After you create a new codeunit, the next step is to define the global variables, text constants and functions that you need in the codeunit. You use the C/AL Globals tool

for this. To access the C/AL Globals tool, make sure that the focus is on the C/AL Editor and click View, C/AL Globals and the **C/AL Globals** window opens:



In the **C/AL Globals** window, you must decide whether you want to add a global variable, a text constant or a function.

#### Global variables

To add a global variable:

- 1 Click the **Variables** tab in the **C/AL Globals** window.
- 2 Enter a name and select a data type. If the data type you select corresponds to an application object, you must also select a subtype, that is, the name of a specific object in the database.

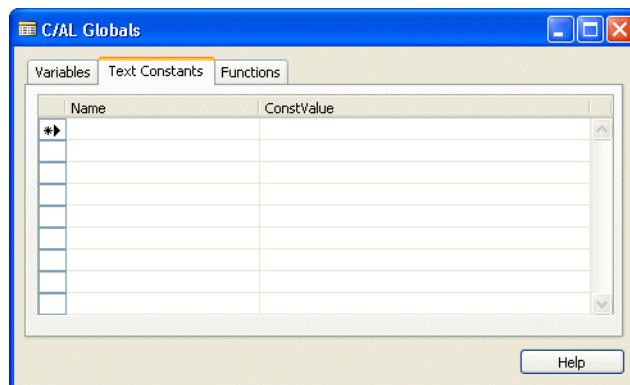
If you select text or code, you must define a length for the variable (the default length is 10 characters for code and 30 for text). If you select OCX or Automation, you must add a subtype. This is described in Chapter 19 "Extending C/AL".

#### Text constants

To add a text constant:

When you create a message for the user in the C/AL Editor you must:

- 1 Click the **Text Constants** tab in the **C/AL Globals** window:



- 2 In the first available **Name** field, enter the name of the new text constant.

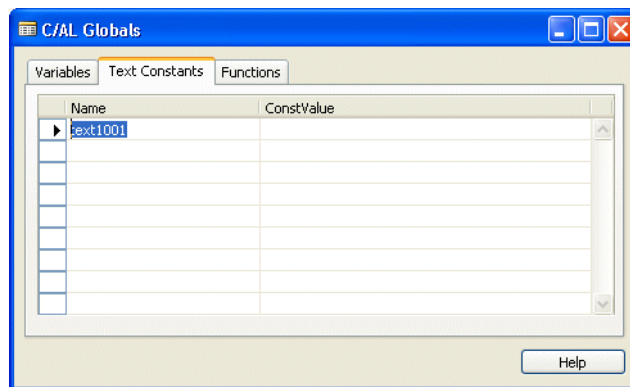
**Note**

There is no naming convention for the text constants. Using the unique ID for the name is a suggestion but not a requirement.

- 3 Open the **Properties** window of the text constant.

A unique ID number has been automatically assigned to the text constant in the **ID** field.

- 4 Copy the ID number to the **Name** field in the **C/AL Globals** window, for example *Text1001*, if the ID number in the **ID** field is 1001.



- 5 In the **ConstValue** field, click the AssistButton ... to open the **Multilanguage Editor** window.
- 6 In the **Language** field, enter ENU for English (United States).
- 7 In the **Value** field, enter the message string that this text constant represents.
- 8 Click OK to exit. If you do not click OK, the information is not saved.
- 9 In the C/AL Editor, copy the ID number to the place where you want the message or error message to appear.

**Example**

```
IF FileName = ' ' THEN
    ERROR(Text1001);
```

Text1001 is a number that is available in the text constants number series for that object.

In the C/AL Editor, when you move the cursor into the new text constant, you see its contents in the status bar.

**Note**

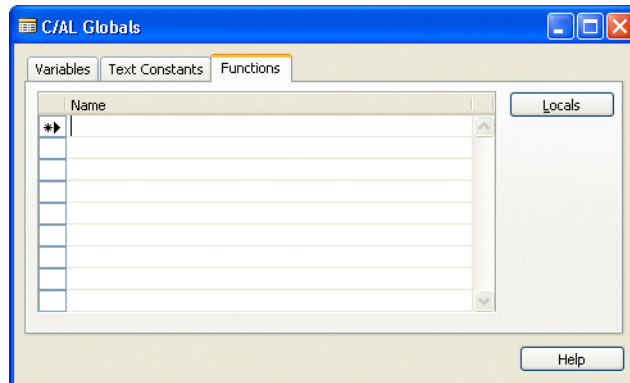
If you remembered to set the application language to English (United States) before entering the Object Designer, you can enter the message string directly into the **ConstValue** field in the **C/AL Globals** window. Then you should open the Multilanguage Editor to make sure that the text is saved as English (United States).



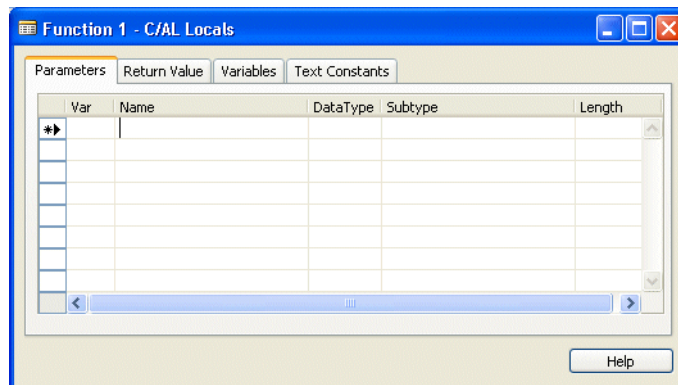
## Functions

To add a function:

- 1 Click the **Functions** tab in the **C/AL Globals** window:



- 2 Enter a name for each function that you want to add.
- 3 Click Locals to define the parameters, return value, local variables and text constants for each function. The **C/AL Locals** window opens:

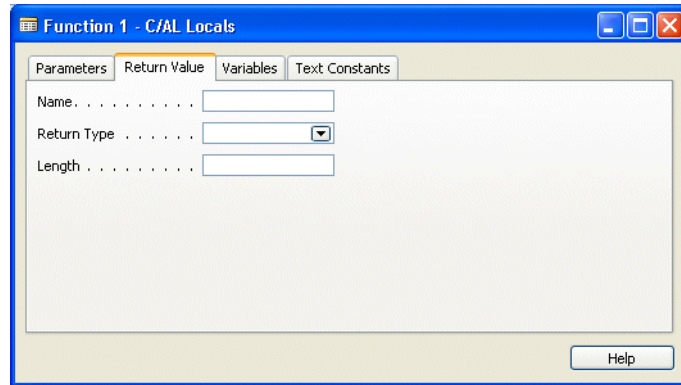


- 4 You must specify the calling method, name, and data type of each parameter. You can also specify a subtype and a length, but this is optional.

The calling method can be specified as **Var**, which means that the parameter is passed by reference rather than by value. The value of a variable can only be changed by a function when it is passed to the function by reference. When the parameter is not specified as **Var**, only a copy of the variable is passed to the function. If the function changes that value, the change only affects the copy and not the variable itself.

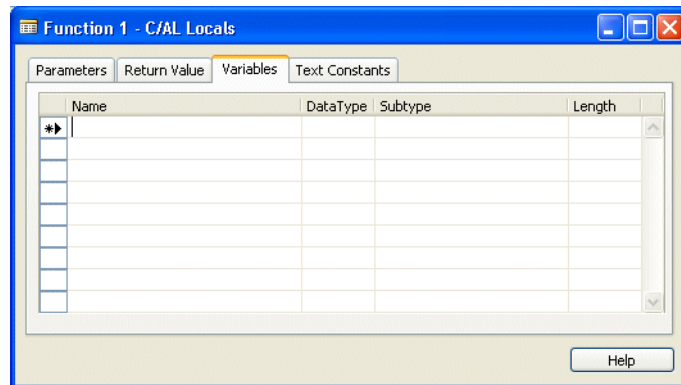
If the type you select corresponds to an application object, you must also add a subtype, that is, the name of a specific object in the database. If you select text or code you have to define a length for it (the default length is 10 characters for code, and 30 for text).

- 5 Click the **Return Value** tab to define the return value for your new function.



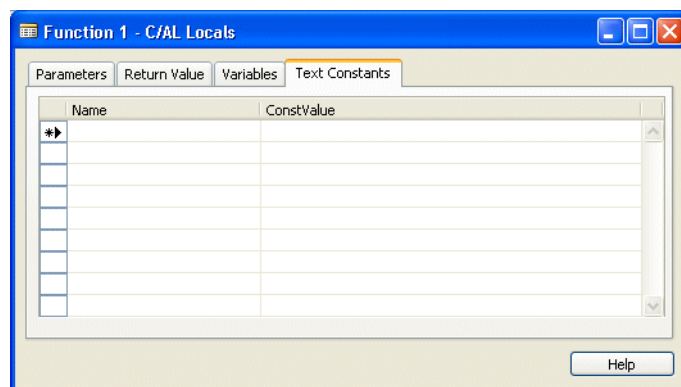
- 6 Enter a name for the return value and select a data type from the drop-down list. You can also select a length, but only if the type is text or code.

- 7 Click the **Variables** tab in order to define local variables:



- 8 For each local variable, you must enter a name and select a data type. If the data type you select corresponds to an application object, you must also add a subtype, that is, the name of a specific object in the database. If you select text or code, you must define a length for the variable (the default length is 10 characters for code and 30 for text).

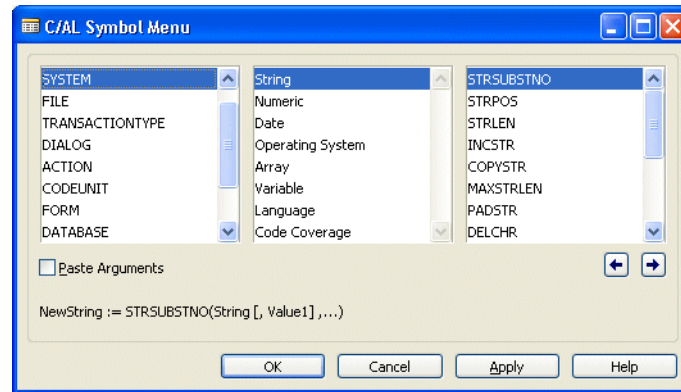
- 9 Click the **Text Constants** tab to define text constants for the function.



## Using the C/AL Symbol Menu

When you write C/AL code in the C/AL Editor, you can use the **C/AL Symbol Menu** window to get an overview of:

- All the variables defined in the codeunit
- All the C/AL functions



The **C/AL Symbol Menu** window is divided into three columns:

- The column to the left lists the names of the variables (if you have defined any) and the function categories.
- The second column lists the subcategories.
- The third column lists the functions in the category you selected.

You can see the syntax and other information, such as the *Caption* property corresponding to the field name you have selected, in the bottom left-hand corner of the window. For more information about the *FieldCaption* subcategory, see page 473.

In some cases, for example when a control on a form is a subform or when a field is a BLOB field, there are more than three columns.

In the C/AL Symbol Menu, click OK or Apply to paste the syntax description into the C/AL Editor. When you click OK, the **C/AL Symbol Menu** window is closed automatically; when you click Apply, the window stays open.

If you need help with any of the C/AL functions shown in the column to the right, select the function name and press F1 to activate the context-sensitive *C/SIDE Reference Guide* online Help.

## Compiling and Saving Codeunits

Before you can run the functions in a codeunit, you must save and compile the code. When you compile the code, the system checks the syntax of the statements. If the compiler finds any errors in the code it displays an error message.

To compile the code in a codeunit:

- 1 Click Tools, Compile.
- 2 If the system finds any errors in your code, you receive an error message. Correct the errors and compile the code again.

To save the codeunit:

- 1 Click File, Save and C/SIDE displays the **Save As** window.
- 2 Enter an ID number and a name. The number is used as a unique identification, while the name serves as a label.
- 3 Select whether or not you want the system to compile the code before it is saved.

If you save the codeunit without compiling it, you won't be able to run it or call any of the functions it contains.

**Why Save without Compiling?**

.....

If you are working on a large and complicated codeunit, you may want to save your work at regular intervals, even though it is not yet finished and cannot yet be compiled. In this case, you have to remove the check mark from the Compiled box before you save the codeunit.

.....

## 15.3 Using Codeunits

When you use codeunits, you eliminate the need to duplicate code and at the same time make the code easier to maintain. If you use the same code repeatedly in your forms or reports, you should create a function in a codeunit. When you have created a function in a codeunit you can access it by writing:

```
<CodeunitName>.<FunctionName>
```

### Example

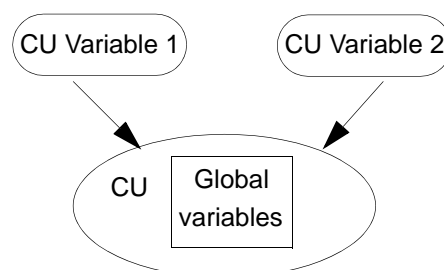
Assume that you have created a codeunit that contains two statistical functions named F and G. The following example shows you how to access these functions from a form.

Codeunit named StatFun
F(x:integer)
Begin
...
End
G(x:integer)
Begin
...
End

Any form
...
Result :=
StatFun.F(3425)+StatFun.G(346);
...

This method is generally applicable. That is, from any application object you can access functions in other application objects by writing the name of the application object that contains the function followed by the name of the function.

You can access codeunits through codeunit variables – either by explicitly declaring a variable with the data type codeunit or by setting the *RunObject* property on forms to a codeunit. A codeunit variable does not contain a codeunit, but only a reference to a codeunit. More than one codeunit variable can refer to the same codeunit as shown in the following figure:



Codeunits contain internal variables that are defined as global variables. These variables cannot be accessed directly from code outside the codeunit, but they can be accessed through user-defined functions on the codeunit. Whenever a codeunit variable is used for the first time, a new instance of the codeunit is created, that is, a

new set of internal variables is initialized so that different codeunit variables use different sets of internal variables.

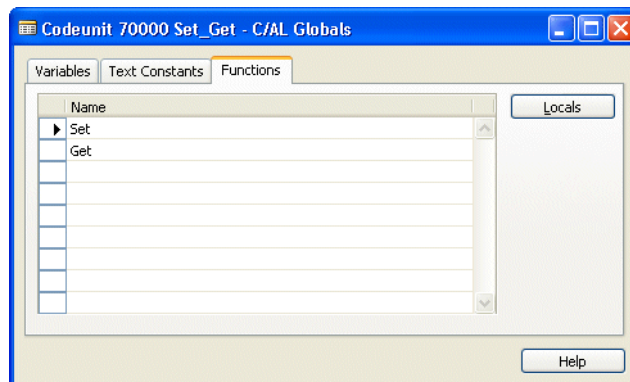
**Codeunit assignment** Codeunits can be treated as objects – one codeunit variable can be assigned to another codeunit variable, which creates a new reference to the same codeunit instance. In other words, the codeunit variables then use the same set of internal variables.

In this example, you create two codeunits:

- one codeunit that has two functions `Set` and `Get`. `Set` sets an internal variable to the value of the parameter given. `Get` returns the value of the internal variable. and
- one codeunit that has two variables that call the first codeunit and are then assigned to each other so that they use the same instance of the first codeunit.

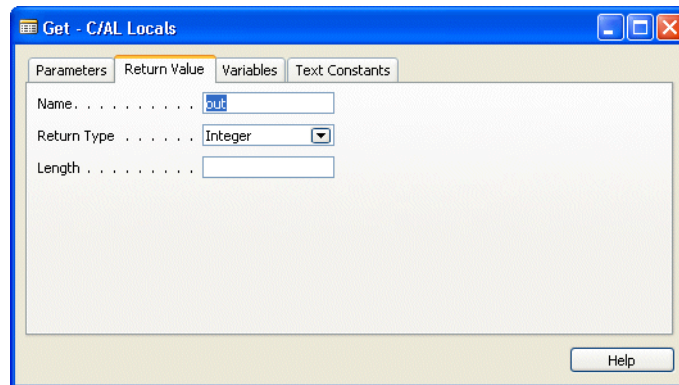
To create the first codeunit:

- 1 Create a new codeunit and open the **C/AL Globals** window.
- 2 Declare a variable called *InternalInt* of data type Integer.
- 3 In the **C/AL Globals** window, click the **Functions** tab and create two functions called `Set` and `Get`.

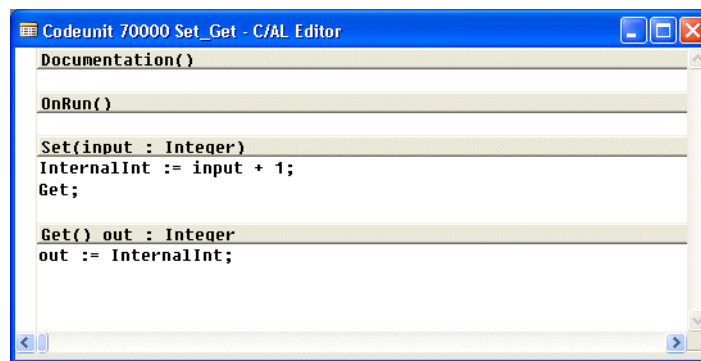


- 4 Select the `Set` function and click **Locals** to open the **C/AL Locals** window.
- 5 In the **C/AL Locals** window, click the **Parameters** tab and create a parameter called *input* of data type Integer and click OK.
- 6 In the **C/AL Globals** window, select the `Get` function and click **Locals** to open the **C/AL Locals** window.

- 7 In the **C/AL Locals** window, click the **Return Value** tab and create a return value called *out* of return type Integer and click OK.



- 8 In the C/AL Editor for the codeunit, two new sections have been added – one for each of the functions that you just created. Enter the following code in the C/AL Editor:



The code in the `Set` function increases the value of the input by one and sets this as the value of the internal variable. It then calls the `Get` function which specifies that this new value is the output value generated by this codeunit

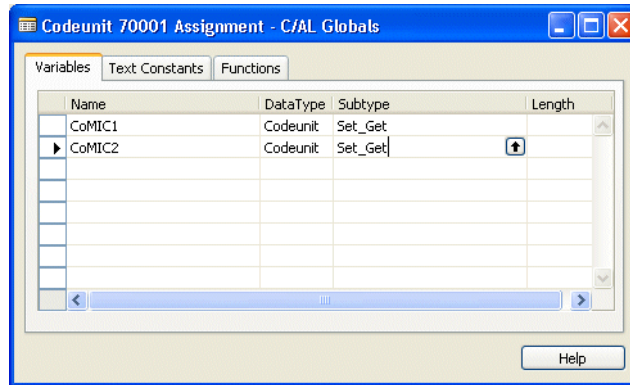
- 9 Save and compile the codeunit. In this example it has been saved as codeunit 70000 and called 'Set\_Get'.

The next step is to create the codeunit that calls two instances of the `Set_Get` codeunit and supplies the input values that these instances should use. This codeunit contains two variables and these are then assigned to each other so that they both use the same instance of codeunit 70000.

To create the second codeunit:

- 1 Create a new codeunit and open the **C/AL Globals** window.

2 In the **C/AL Locals** window, create the following two variables:



These two variables both call the Set\_Get codeunit.

3 Add the following code to the OnRun trigger

```
CoMIC1.Set(1);
CoMIC2.Set(2);

//Calls two instances of codeunit 70000 with the input parameters 1
and 2 respectively.

MESSAGE('When running two instances, the value of CoMIC1 is %1 and
the value of CoMIC2 is %2.',CoMIC1.Get,CoMIC2.Get);

//The message retrieves the return values from codeunit 70000 and
shows them in the message.

//CoMIC2 returns 2; CoMIC2 returns 3, because you have added 1 to
each parameter.

CoMIC2 := CoMIC1;

//CoMIC2 is assigned to CoMIC1 and they both use the same instance of
codeunit 70000.

MESSAGE('After assinging CoMIC2 to CoMIC1, the value of CoMIC1 is %1
and the value of CoMIC2 is also %2.',CoMIC1.Get,CoMIC2.Get);

//CoMIC1 and CoMIC2 now show the same value.
```

The first statement in this codeunit calls two instances of the Set\_Get codeunit and supplies the input values that these instances should use. The next statement assigns the two variables to each other so that they both use the same instance of codeunit 70000 and therefore generate the same return value.

4 Save and compile the codeunit. In this example it has been saved as codeunit 70001 and called 'Assignment'.

**CLEAR on codeunits** When you use the CLEAR function on a codeunit variable that has a reference to a codeunit instance with two or more references, CLEAR only deletes the reference to the codeunit and not the actual instance of the codeunit. In other words, the codeunit stays



intact and can still be used by other codeunit variables that may have been assigned a reference to this codeunit.

To delete an instance of a codeunit, you must clear all the references to the codeunit with the `CLEAR` function. To clear the internal variables in a codeunit, you must call the `CLEARALL` function from a user-defined function within the codeunit. A local codeunit variable is automatically cleared when it goes out of scope and is no longer used by the codeunit.

Single instance  
codeunit

In some cases, only one instance of a codeunit needs to exist. This means that all the codeunit variables of a particular codeunit use the same set of variables. When you set the *SingleInstance* property of the codeunit to *Yes*, all the codeunit variables of that codeunit use the same instance, thereby allowing you to create global variables.

#### Note

.....

We recommended that you avoid using global variables for most types of code. However, in certain situations, it may be necessary to use them, for example, to make sure that you are only using one instance of an external variable.

.....

A single instance codeunit is instantiated when you use it for the first time. Normal codeunit instances (codeunits that do not have the *SingleInstance* property set) are deleted whenever the last codeunit variable that uses that codeunit instance goes out of scope. However, single instance codeunits remain instantiated until you close the company.

#### Example

Open Codeunit 70000 that you created earlier in this chapter and set the *SingleInstance* property to *Yes*. Save it as codeunit 70002 and call it *SingleInst*.

Open codeunit 70000 again and create two variables:

Variable	Data Type	Subtype
CoMIC1	Codeunit	SingleInst
CoMIC2	Codeunit	SingleInst

In codeunit 70000 add the following code to the OnRun trigger:

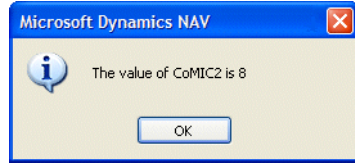
```

    CoMIC1.Set(7);
    //a codeunit instance is created if one did not exist
    CoMIC2.Get();
    //returns 8, that is 7 + the 1 added by the set function in codeunit
    //70002. CoMIC2 uses the same instance as CoMIC1, they use the same
    //internal variables.

MESSAGE('the value of CoMIC2 is %1',CoMIC2.Get);

```

Now, when you run codeunit 70000, you see the following message:



The input value in codeunit 70000 is 7 and codeunit 70002 adds 1 to this value.

Now open codeunit 70002 and change the value in the *set* trigger to, for example, `input+4`. Close, save and compile codeunit 70002.

Now run codeunit 70000 and you still get the same message as before. This occurs because once you have loaded a single instance codeunit it remains loaded on the client until you restart the client.

**Note**

.....  
It is possible to use a single instance codeunit across objects and not only within the same object.  
.....

**Limitations on Codeunits**

Global variables and temporary tables in a codeunit cannot be accessed directly from other application objects. The only way to access these values is through the functions you have created in the codeunit.

Every C/AL function can be used in a codeunit. However, you cannot create a function with the same name as a pre-defined function. Neither can two or more user-defined functions have the same name (unless they are part of different application objects).

## Chapter 16

### Introducing the C/AL Language

This chapter introduces the C/AL language. It describes how to use the language to create functions, as well as describing the syntax of the language.

- What Can You Do with C/AL?
- What Are Statements, Expressions, and Operators?
- Introducing the Elements of C/AL Expressions
- The C/AL Control Language

## 16.1 What Can You Do with C/AL?

In the previous sections of this book you learned how to design some basic database objects such as tables and forms. But simply getting these objects up and running is not enough. To create a coherent application, you must make these database objects work together. C/AL code is the glue that binds all the database objects together to form a unified whole.

When you are designing professional applications you often need specialized functions. C/AL lets you create functions that extend the functionality of C/SIDE. For example, you can create special functions for use anywhere in the database.

The most important things that you can do with C/AL are:

**Design Your Own Functions** Although C/SIDE has a large number of in-built functions, it will sometimes be necessary or perhaps just more convenient for you to create your own functions. For example, you will need to develop your own functions when the application you are developing repeatedly uses the same non-trivial processing.

**Connect Database Objects** C/AL code glues your database objects together. C/AL includes a number of commands that control how the individual database objects in your application interact.

**Read, Write and Modify Data** C/AL includes standard functions for reading, writing and modifying table data.

## 16.2 What Are Statements, Expressions, and Operators?

In this section, the following terms are introduced and explained:

- Statements
- Expressions
- Data types
- Operators

Consider the following C/AL code sample:

```
Amount := 34 + Total;
```

This individual code line is also called a statement. The following table illustrates how the statement can be broken into smaller elements.

Element	Description
34 + Total	An expression In this case the expression consists of an arithmetic operator (+) and two arguments (34 and Total), which also could be called sub-expressions. Every valid C/AL expression can be evaluated to a specific value.
:=	The assignment operator When the expression on the right-hand side has been evaluated, this operator is used to assign or store the value in the Amount variable.
Amount	This is called a variable. It is used to reference a memory location where data is stored.

### What Is a C/AL Expression?

An expression is a fundamental C/AL concept. This section describes expressions and how they are used.

An expression can be used as an argument for a C/AL function. Consider the following C/AL statement:

```
Date := DMY2DATE(31, 12, 2001);
```

This function takes three simple expressions as arguments, 31, 12 and 2001.

A C/AL expression is a group of characters (data values, variables, arrays, operators and functions) that can be evaluated, with the result having an associated data type.

All the expressions in C/AL are built from:

- Constants
- Variables
- Operators
- Functions

Depending on the elements in the expression, the evaluation results in a value with a C/AL data type. The following table shows some typical expressions:

Expression	Evaluates to:
'Welcome to Hawaii'	The string 'Welcome to Hawaii'
'Welcome' + ' to Hawaii'	The string 'Welcome to Hawaii'
43.234	The number 43.234
ABS(-7234)	The number 7234
len1 < 618	TRUE or FALSE depending on the value of len1

The first row shows a text string which is evaluated to itself. The second row evaluates into a concatenation of the two strings. The third row shows a decimal number, which is evaluated to itself. The expression in the fourth row contains a function, with which the given argument is evaluated to the number 7234. The last row shows a comparison between a variable and a numerical constant.

These examples show that when C/AL expressions are evaluated, the results have a specific data type. The next section explains the C/AL data types in more detail.

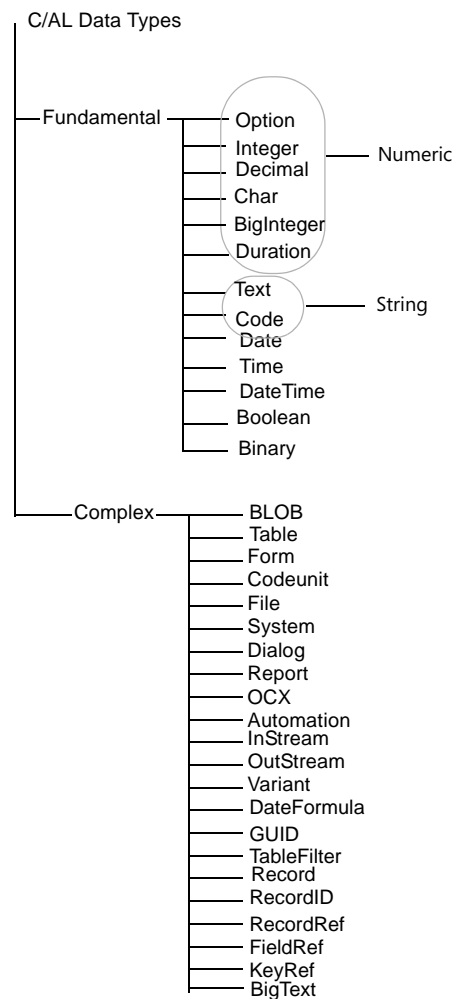
### Introducing the C/AL Data Types

As you have already seen, variables can be used to store data of various types. By declaring variables of the correct type, you:

- create faster code.
- save space.
- avoid runtime errors due to overflow.
- avoid runtime errors caused by impossible type conversions.

For example, if you know that a variable will always contain a number between 0 and 700, you should use an integer variable instead of a decimal variable. Any calculations that are performed will be faster because the system uses 4 bytes per integer operation instead of the 12 bytes that decimal variables require. On the other hand, you must use a data type that can hold every possible value that is needed in your calculations. For example, if you try to store the value 1233.345 in an integer variable you will get a runtime error.

C/AL contains a wide range of data types. These data types can be divided into the following categories:



### Fundamental Data Types

C/AL contains a number of fundamental data types, which are designed to store boolean values, numbers, text, time and dates.

**Boolean** The possible values are TRUE or FALSE.

**Integer** Used to store integers between -2,147,483,647 and 2,147,483,647.

**BigInteger** Used to store very large whole numbers.

**Duration** Used to represent the difference between two datetimes, in milliseconds.

**Option** This denotes an option value. Option values can freely be converted to numeric ones. The values range from -2,147,483,647 to 2,147,483,647.

#### Example

Assume that Number is a numeric variable and that Type denotes a field of type Option in the **Purchase Header** table. In the following statement, the option value is converted to a number:

```
Number := "Purchase Header".Type;
```

**Example**

This example illustrates how the possible values of an option field can be used as constants in your C/AL code:

```
"Purchase Header".Type := "Purchase Header".Type::Invoice;
```

**Decimal** Denotes decimal numbers ranging from  $-10+E63$  to  $+10+E63$ . The exponent ranges from  $-63$  to  $+63$ . Decimal numbers are held in memory with 18 significant digits.

**Date** Denotes dates ranging from January 1, 0 (the year zero) to December 31, 9999. An undefined date is expressed as 0D. All dates have a corresponding closing date. The closing date for a given date is regarded by the system as a period following the given date but before the next normal date. A closing date is therefore sorted immediately after the corresponding normal date but before the next normal date.

**Time** Denotes a time. An undefined time is expressed as 0T. Any time in the range 00:00:00 to 23:59:59.999 is valid.

**DateTime** Denotes a date and the time of day.

The datetime is stored in the database as Coordinated Universal Time (UTC). UTC is the international time standard (formerly Greenwich Mean Time, or GMT). Zero hours UTC is midnight at 0 degrees longitude. The datetime is always displayed as local time in Dynamics NAV. Local time is determined by the time zone regional settings used by your computer.

You must always enter datetimes as local time. When you enter a datetime as local time, it is converted to UTC using the current settings for the time zone and daylight saving time.

There is only one constant available when you use this data type: undefined datetime. `DateTime := 0DT`

- C/SIDE Database Server

The earliest permitted datetime is January 1, 0000, 00:00:00.000.

The latest permitted datetime is December 31, 9999, 23:59:59.999.

- SQL Server

The earliest permitted datetime is January 1, 1754, 00:00:00.000.

The latest permitted datetime is December 31, 9999, 23:59:59.999.

Any datetimes that are not within this range and that you try to enter or construct by, for example, adding a datetime to a duration, are regarded as undefined datetimes and give an error message.

Undefined dates are stored as January 1, 1753, 00:00:00.000.

**Char** Stores a single character as a value in the range 0 to 255. This data type can be freely converted between a number and a character. This means that you can use the same mathematical operators as you can with a variable of a numerical data type.



**Example**

You can assign a constant string of the length 1 to a char variable:

```
C := "A";
```

**Example**

You can also assign a single char in a text, code or binary data type variable to a char variable:

```
C := S[2];
```

**Note**

When you use the text and code data types, it is important to distinguish between the maximum length of the string and the actual length of the string. The maximum length can be seen as the upper limit for the number of characters in the string, while the actual length describes the number of characters used in the string.

**Text** Denotes a text string. The length of the string ranges from 1 to 1024 characters. You can index any character position in a string – for example A[65] refers to the 65th character in the variable called A. The resulting values will be of data type char. The length of a variable of data type text corresponds to the number of characters in the text. For example, an empty text string has length 0.

The following table illustrates some typical examples of text strings. In the following examples it is assumed that the variable t is of data type text and has a maximum length of 6:

Assignment	Results in...
t := 'AbC';	The variable t now contains "AbC".
t := '123456abx';	Results in a runtime error because the length (9) exceeds the maximum length (6).

**Code** Denotes a special type of text string. When a given text is assigned to a variable of data type code, the text is changed to uppercase, and any leading and trailing spaces are removed. You can index any character position in a string – for example, A[65]. The resulting values will be of the char data type. The maximum length of a variable of data type code ranges from 1 to 250 characters. The length of a variable of data type code always corresponds to the number of characters in the text without leading and trailing spaces.

**Example**

The following table shows some typical examples of code string assignments. In these examples, it is assumed that the variable c is of data type Code, and has a maximum length of 4:

Assignment	The variable c now contains...	The length is...
c := 'AbC';	'ABC'	3
c := '1';	'1'	1
c := '';	" (empty string)	0 (zero)

Assignment	The variable c now contains...	The length is...
c := ' 2 ';	'2'	1
c := '1 2';	'1 2'	3

### Descriptive Data types

The following table summarizes the correspondence between the descriptive data types and the simple C/AL data types:

Descriptive data type	Includes these system data types...
Numeric	char, integer, biginteger, duration, option, and decimal
String	text and code

### Complex Data Types

C/AL also contains a number of complex data types. Complex data types are used when you need to work with, for example, records in tables, pictures (bitmaps) or disk files. As C/AL is object oriented, each complex data type can include both member variables and member functions.

**BLOB** This is a Binary Large Object. Variables of this data type differ from normal numeric and string data type variables in that they have a variable length. BLOBs are used to store memos (text), bitmaps (pictures) or user-defined types. The maximum size of a BLOB is normally determined by your system's disk storage capacity, as the upper limit is 2GB.

**Record** This is a complex data type, consisting of a number of simpler elements called fields. A record corresponds to a row in a table. Each field in the record is used to store values of a certain data type. The fields are accessed using the variable name of the record (often the same as the name of the corresponding table), a dot (a period) and the field name. A record is typically used to hold information about a fixed number of properties.

**Form** Variables of this data type are used to store forms. This is a complex data type and can contain a number of simpler elements called controls. Controls are used to display information to the user or to receive user input.

**Codeunit** Variables of this data type are used to store codeunits. This is a complex data type which can contain a number of user-defined functions.

**File** Variables of this data type give you access to operating system files.

**Dialog** Variables of this type are used to store dialog windows. A number of functions are available for manipulating dialogs.

**Report** Variables of this data type are used to store reports. This is a complex data type that can contain a number of simpler elements called controls. Controls are used to display information to the user.

**DateFormula** Use this data type to contain a date formula that has the same capabilities as an ordinary input string for the `CALCDATE` function. The DateFormula data type is used to provide multilanguage capabilities to the `CALCDATE` function.

**GUID** Use this data type to give a unique identifying number to any database object.

The Globally Unique Identifier (GUID) data type is a 16 byte binary data type. This data type is used for the global identification of objects, programs, records and so on. The most important property of a GUID is that each value is globally unique. The value is generated by an algorithm, developed by Microsoft, which assures this uniqueness.

The GUID is a 16 byte binary data type and can be logically grouped into the following subgroups: 4byte-2byte-2byte-2byte-6byte. The standard textual representation is {12345678-1234-1234-1234-1234567890AB}.

**TableFilter** Use this data type to apply a filter to another table. At the moment, this data type can only be used when you are setting security filters from the Permission table.

**RecordRef** This complex data type identifies a row in a table. Each record consist of fields (which form the columns of the table). A record is typically used to hold information about a fixed number of properties.

The RecordRef object can refer to any table in the database. Use the `RecordRef.OPEN` function to select the table you want to access. When you use the `RecordRef.OPEN` function a new object is created. This object contains references to the open table, filters and the record itself and all the fields it contains.

If one RecordRef variable is assigned to another RecordRef variable, they both refer to the same table instance.

**RecordID** This complex data type contains the table number and the primary key of a table. You can store a RecordID in the database but you cannot set filters on a RecordID.

**FieldRef** This complex data type identifies a field in a table and gives you access to this field. The fieldref object can refer to any field in any table in the database.

**KeyRef** This complex data type identifies a key in a table and the fields in this key. This gives you access to the key and the fields it contains. The keyref object can refer to any key in any table in the database

**InStream and OutStream** Variables of these data types enable you to read from or write to files and BLOBs. In addition, you can use InStream and OutStream to read from and write to objects of the Automation and OCX data types.

**VARIANT** This data type can contain the following C/AL data types: record, file, action, codeunit, Automation, boolean, option, integer, decimal, char, text, code, date, time, binary, DateFormula, TransactionType, InStream and OutStream.

**BigText** This complex data type is used to contain large text documents. Data of the BigText data type cannot be displayed, in for example, the debugger or in a message window. However, you can use the BigText functions to extract part of a big text and place it in a normal text string that can be displayed.

The maximum length of a BigText variable is 2147483647 characters. This is the equivalent of 2 Gb.

**OCX and Automation** See "Extending C/AL" on page 363.

For more information about these data types, see the *C/SIDE Reference Guide* online Help.

## Creating Arrays of Variables

You can create 10-dimensional variables, using the simple and complex data types. There are no limitations on how many elements a dimension can contain but an array variable can never have more than 1,000,000 elements in all. The physical size of an array is limited to 2 GB (or available memory). Arrays are always indexed with a number for each dimension that ranges from 1 to (and including) the size of the dimension. If you accidentally index outside the range of the dimensions of an array, a runtime error occurs.

### Example

Assume that Foo is a one-dimensional array variable of data type Integer, with the dimension 10.

To index the first element, use Foo[1]. To index the last element, use Foo[10].

### Example

Assume that Bar is an array variable of data type Date with the dimensions 2x3x4. Then Bar has 24 elements.

To index the first element, use Bar[1,1,1]. To index the last element, use Bar[2,3,4].

## 16.3 Introducing the Elements of C/AL Expressions

In the previous sections you were introduced to C/AL expressions and data types. The aim of this section is to present the basic elements of C/AL expressions to you. The following subsections will briefly discuss:

- Constants
- Variables
- Operators
- Functions

### Constants

A constant is the simplest type of operand used in C/AL. The value of a constant cannot be changed during the execution of the code. Constants can be defined for each of the simple data types in C/AL.

#### Entering Values in C/SIDE

Beware that in the following examples, numbers such as 2,147,483,647 and 999,999,999,999,999.99 cannot be entered in the C/AL system in this form. The commas are only used to increase the legibility of this document. If you use commas when you enter numbers in the C/AL editor, a compilation error occurs.

**boolean constant** A boolean constant can have either the value TRUE or FALSE.

**integer constant** An integer constant has a value in the range -2,147,483,647 to 2,147,483,647.

**decimal constant** A decimal constant must contain a decimal point "." (depending on your regional settings) and have at least one digit to the right of the decimal point (for example the digit "0"). A constant of type decimal can be used to represent decimal numbers between -999,999,999,999,999.99 and 999,999,999,999,999.99 with 18 significant digits.

**date constant** A date constant is written as six or eight digits followed by the letter "D" (the date constant expressing "undefined date" is, however, entered as "0D"). The digits specify the date in the format MMDDYY or MMDDYYYY.

**time constant** A time constant is written as six or nine digits followed by the letter 'T' (the "undefined time" constant is, however, entered as "0T"). The nine digits specify the time in the format HHMMSS[.XXX], that is, a 24 hour format with an optional part specifying thousandths of a second.

**text constant** A text constant is a character string. C/SIDE assigns unique IDs to text constants, so that an ID number represents a specific text constant. Examples of text constants are error messages, messages and warnings.

The following table illustrates different types of C/AL constants:

Constant	Description
TRUE	boolean constant

Constant	Description
50000	integer constant
-23.7	decimal constant
122101D	date constant (December 21, 2001)
141230T	time constant (the time 14:12:30)
ABC	text constant

## Using Variables in C/AL

There are two types of variables in the C/AL system: user-defined variables and system-defined variables.

### User-defined variables

User-defined variables are ones that you define when you create new C/AL code. You can define variables that are global and apply to all the functions within a codeunit and you can define variables that are local and apply to a single function in a codeunit. Both types of user-defined variables are local to the codeunit in which they are defined. These variables can be used to store information at runtime, and the values can be changed as desired.

### System-defined variables

System-defined variables are provided by the system. These variables are automatically maintained by the system. The system-defined variables are, for example, Rec, xRec, CurrForm and CurrReport.

When the system is running, it executes code in functions and triggers, for example entry-processing code for a table. Before the code is executed, the system automatically assigns values to the associated system-defined variables, and the values of these variables can be used in the triggers and the local functions.

When triggers and functions are executed, the system-defined variables can be used just like normal variables (new values can be assigned to them). That is, the values of the system-defined variables are not updated by the system while the C/AL code is being executed, but only before the function or trigger is executed.

### Note

The value in a system-defined variable does not propagate backwards. In other words the user cannot use a system-defined variable to modify the state of the system.

## Variable Names

There are numerous rules and restrictions that you must follow when naming variables:

- Variable names must be unique – a codeunit cannot contain two user-defined variables with the same name.
- A user-defined and a system-defined variable cannot have the same name.
- Uppercase and lowercase letters are not distinct, that is, Smith and SMITH refer to the same variable.
- In C/AL, you can use special characters (for example, spaces) in the name of a variable (an identifier).
- The maximum length of a variable name is 30 characters.

- A variable cannot have the same name as a C/AL function or a reserved word. Please note that this rule applies to both uppercase and lowercase spellings. For example, neither BEGIN nor begin is valid.

All ASCII characters are valid in variable names, except:

- Control characters (ASCII 0-31, 255)
- The character " (ASCII 34)

When you name a variable, be careful to note that characters cannot be combined freely unless you encapsulate the variable name in double quotes, as in "Customer No.". If you don't, you should name variables like this:

The first character must be:

- a letter in the range: a..z, A..Z (ASCII 97-122, 65-90), or
- an underscore (ASCII 95),

...followed by a maximum of 29 characters, which can be either

- a letter a..z, A..Z (ASCII 97-122, 65-90)
- an underscore (ASCII 95), or
- digits in the range 0..9 (ASCII 48-57).

As mentioned earlier, you can include one or more special characters (spaces, and so on) in a variable name in C/AL. However the entire variable name must be enclosed in double quotes. In this case, the name can contain any mix of letters, digits and special characters.

#### Note

.....

The double quotes are not part of the variable name, but are necessary in order to avoid receiving an error when you compile the codeunit.

.....

Here are a number of examples showing valid variable names:

- Customer
- StockGroup1
- "@Vendor"
- "1st AddressLine"
- "Purchase/Sales"
- "Sales In GBP"
- " YesCrazy Name1Ñ3"

...and the following are examples of invalid variable names

- 34467
- 23" Tubes
- Stock Group4
- "Sale"s in GBP"
- )-Names

- END

### Initialization

Variables are automatically initialized before C/AL code is executed. A boolean variable is set to FALSE and numeric variables are set to the default value zero, while strings (text, code and so on) are initialized to the value '' (the empty string) and date and time variables are set to the undefined time 0T and the undefined date 0D, respectively.

As mentioned earlier, the system automatically handles the system-defined variables. This also includes the necessary initialization. This means that no actions are required by the user before the system-defined variables can be used.

### Assignment and Type Conversion

There are two ways to assign values:

- As parameter assignment, for example FUNCTION(Expression). The data type that results from the evaluation of the expression must correspond to a specific data type or have a data type that can be converted automatically to the correct data type. For a detailed discussion about evaluation and type conversion in expressions, refer to the chapter "Type Conversion in Expressions" on page 506.
- By using the assignment operator " := " (for example Variable := Expression). Generally, the data type that results from the evaluation of the right-hand side expression must be the same data type as the variable (left operand) or have a data type that can be converted automatically to the data type of the left operand.

Automatic type conversion in assignments takes place when:

- a parameter in a function call does not have the correct data type. This happens, for instance, if a function that is supposed to be called with an integer argument is called with, for example, a decimal argument.
- the evaluation of the expression on the right-hand side of an assignment operator ( := ) results in a data type that differs from the data type of the variable on the left-hand side.

Automatic type conversion in assignments can freely take place between the following numeric data types, provided overflow does not occur:

**char ↔ integer ↔ decimal**

Automatic type conversion in assignments can also freely take place between the String data types:

**code ↔ text**

All of these examples are based on simple variables. Nevertheless, the same assignment rules apply for arrays in C/AL. Furthermore, if the left operand in an assignment (the variable) is an array, the dimension(s) of the right-hand expression must correspond to the dimension(s) of the variable.

**text ↔ bigtext**

Text can be easily be converted to bigtext but a bigtext must be broken up into smaller parts before it can be converted to text.



**Note**

.....

The type conversion that takes place in assignments can cause runtime errors even though the data types are convertible. A runtime error can occur in an assignment if the converted value is outside the valid range for the left-hand side variable. Correspondingly a runtime error can occur if the converted value is outside the valid range for a parameter in a function call.

.....

**Example**

Variable B is defined as a one-dimensional array with four text type elements with the maximum length 15. A value could be assigned to the second element in the array as shown here:

```
B[2] := 'Enter your name';
```

**Example**

Result is an option variable, while Amount and Total are both decimal variables. Consider the following assignment statements:

```
Amount := 10;
Total := 4;
...
Result := Amount + Total;
```

This code can always be compiled, but a runtime error will occur if the result of the right-hand side expression "Amount + Total" exceeds the range permitted by the data type of the left-hand side variable Result, that is, outside the range -2,147,483,647 to 2,147,483,647.

**Valid Assignments**

The following tables shows whether it is possible to assign the value of an expression of a given type to a variable of the same type or to a variable of a different type. These tables only cover the numeric and string data types.

Numeric Data Types:

Variable Type	Expression Type					
	char	option	integer	biginteger	duration	decimal
char	●	⦿	⦿	⦿	⦿	⦿
option	●	●	●	⦿	⦿	⦿
integer	●	●	●	⦿	⦿	⦿
biginteger	●	●	●	●	●	⦿
duration	●	●	●	●	●	⦿
decimal	●	●	●	⦿	⦿	●

String Data Types:

Variable Type	Expression Type		
	text	code	bigtext
text	⦿	⦿	⦿
code	⦿	⦿	⦿
bigtext	⦿	⦿	⦿

- The assignment is valid
- ⦿ The assignment is valid, but overflow may occur

**Note**

Bigtext can be assigned by using the bigtext functions, for example, GETSUBTEXT.

**Using Operators in C/AL**

Operators can be used in expressions to combine, investigate and manipulate values and data elements. This section describes the function of the operators in C/AL. The following table shows the valid operators in C/AL:

C/AL operator	Meaning
.	Fields in records, controls in forms and reports
()	Parentheses
[]	Indexing
::	Scope
+	Addition
-	Subtraction or negation
*	Multiplication

C/AL operator	Meaning
/	Division
DIV	Integer division
MOD	Modulus
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
=	Equal to
<>	Not equal to
IN	In range
AND	Logical conjunction
OR	Logical disjunction
NOT	Logical negation
XOR	Exclusive logical disjunction
..	Range

The "+" and the "-" operators can be used both as unary and binary operators. The "NOT" operator can only be used as an unary operator. All the other operators are binary.

Most of the operators can be used on different data types. The action of these operators may depend upon the data type of the expression that they are used on. Here are some typical examples:

#### Example

The "+" operator used as a binary operator:

```
number + number
```

This returns the sum of the numbers, that is, a result of the type number.

#### Example

The "+" operator used as a binary operator:

```
string + string
```

This returns the concatenation of the strings, that is, a result of the type string.

#### Example

The "+" operator can be used as an unary operator to indicate sign, for instance:

```
+ 34545
```

For more information about the way each operator functions, see the chapter Type Conversion Mechanisms on page 508, which explains the type conversion mechanisms in C/AL.

## Operator Hierarchy

The operators that we have just discussed are organized in a hierarchy that determines the order in which the operands in a given expression are evaluated. The following list shows the order of precedence of the C/AL operators:

- 1 .(fields in records), [] (indexing), () (parentheses), :: (scope)
- 2 NOT, - (unary), + (unary)
- 3 \*, /, DIV, MOD, AND, XOR
- 4 +, -, OR
- 5 >, <, >=, <=, =, <>, IN
- 6 .. (range)

The following examples illustrate how this hierarchy works in practice.

Although the expressions appear to be the same, they produce different results.

### Example

The expression

```
2 + 3 * 4
```

is evaluated to 14, whereas the expression

```
(2 + 3) * 4
```

is evaluated to 20.

## Function Calls

C/AL contains a number of functions that can be used for different purposes, such as string handling, text formatting, database handling and so on. Some of these functions can use a varying number of parameters.

In a function call, the parameters are separated by commas, and the optional parameters may be omitted from the right. This means that if a function has, for example, 3 optional parameters, then you cannot omit the second without omitting the third.

### Example

The fictitious function

```
FUNCTION([Optional1] [, Optional2] [, Optional3])
```

can be called as

```
FUNCTION(Optional1, Optional2)
```

but not as

```
FUNCTION(, Optional2, Optional3)
```

**Example**

ABS is a typical example of a C/AL function with a fixed number of parameters (1).

```
Value := -1033; {A negative integer value}
PositiveValue := ABS(Value); {Calculate the positive value 1033}
```

**Example**

The function DMY2DATE is a typical example of a function that can be called with a variable number of parameters.

```
NewDate := DMY2DATE(5, 11, 1992); {Returns the date November 5, 1992}
```

Depending on the use of the DMY2DATE function, 1, 2 or 3 parameters can be passed to the function, as the second and third parameter are optional. When the second and third parameters are not used, the system uses values from the system date as default.

## 16.4 The C/AL Control Language

This section describes the basic structures in the C/AL control language and how to use them. All the C/AL programs that you create consist of one or more statements, which are executed sequentially in top-down order. However, you will often need to control the direct top-down flow of the execution. One or more statements may have to be repeated a number of times, and in another situation you may have to make the execution of a certain statement conditional.

The control structures in C/AL are divided into the following main groups:

- Compound Statements
- Conditional Statements
- Repetitive Statements
- WITH Statements

### Compound Statements

In some cases, the C/AL syntax only allows you to use of a single statement. If however you have to execute more than one simple statement, the statements can be turned into a compound statement, by enclosing the statements between the keywords `BEGIN` and `END`. The syntax is:

```
BEGIN
    <Statement 1>;
    <Statement 2>;
    .
    .
    <Statement n>;
END
```

The individual statements are separated by a semicolon. In C/AL and Pascal a semicolon is used to separate statements, and not, as in other programming languages, as a terminator symbol for a statement. Nevertheless, an extra semicolon before an `END` does not cause an error because it is interpreted by the compiler as an empty statement.

The `BEGIN END` structure is also called a block. Blocks can be very useful in connection with the other control structures to be discussed in the following.

### Conditional Statements

By using a conditional statement, you can specify a condition and one or more commands that should be executed, according to the evaluation of the condition: `TRUE` or `FALSE`. There are two types of conditional statements in C/AL:

- 1 `IF THEN [ELSE]`, when there are 2 choices.
- 2 `CASE`, when there are more than 2 choices.

## The IF THEN ELSE Control Structure

IF THEN ELSE statements have the following syntax:

```
IF <Condition> THEN <Statement1> [ ELSE <Statement2> ]
```

which means

If <Condition> is true, <Statement1> is executed. If <Condition> is false, <Statement2> is executed.

As defined earlier, the square brackets around ELSE <Statement2> mean that this part of the statement is optional.

This statement is used when different actions are executed, depending on the evaluation of the <Condition>.

You can build even more complex control structures by nesting IF THEN ELSE statements. A typical example is:

```
IF <Condition1> THEN IF <Condition2> THEN <Statement1> ELSE
<Statement2>
```

If <Condition1> is false, nothing is executed. If <Condition1> and <Condition2> are both true, <Statement1> is executed. If <Condition1> is true, and <Condition2> is false, <Statement2> is executed. Please note that a semicolon preceding an ELSE is not allowed.

Reading several nested IF THEN ELSE statements can be quite confusing but a general rule is that an ELSE belongs to the last IF that lacks an ELSE.

Here are some examples of IF THEN ELSE statements:

### Example

An IF statement without the optional ELSE part:

```
IF Amount < 1000 THEN Total := Total + Amount;
```

### Example

```
(1) ...
(2) IF Amount < 1000
(3) THEN BEGIN
(4)     IF I > J THEN Max := I
(5)           ELSE Max := J;
(6)     Amount := Amount * Max;
(6)     END
(7) ELSE
(8) ...
```

A common error that is often made by inexperienced C/AL programmers is to put an extraneous semicolon at the end of a line before an ELSE (line 4). As mentioned earlier, this is not valid according to the syntax of C/AL, as the semicolon is used as a statement separator. (The end of line 4 is inside the inner IF statement).

## The CASE Control Structure

The syntax of the `CASE` statement is:

```
CASE <Expression> OF
  <Value set 1> : <Statement 1>;
  <Value set 2> : <Statement 2>;
  ...
  ...
  <Value set n> : <Statement n>;
  [ELSE <Statement n+1>]
END;
```

In this definition, `<Expression>` cannot be a record and `<Value set>` must be an expression or a range.

`CASE` statements are also called multiple option statements and are typically used when you must choose between more than two different actions. The function of the `CASE` statement is as follows:

- The `<Expression>` is evaluated, and the first matching value set executes the associated statement, if there is one.
- If none of the value sets matches the value of the expression, and the `ELSE` part has been omitted, no action is taken; but if the optional `ELSE` part is used, then the associated statement is executed.

The data type of the value sets must be the same as the data type of `<Expression>` or at least be convertible to the same data type.

### Note

.....

The data type of the value sets is converted to the data type of the evaluated `<Expression>`, if necessary. This type conversion can cause an overflow at run time if the resulting data type cannot hold the values of the value sets.

.....

### Example

The following C/AL code prints various messages depending on the value of `Number`. If the value of `Number` does not match any of the entries in the `CASE` structure, the `ELSE` entry is used as the default.

```
CASE Number OF
  1,2,9: MESSAGE('1, 2 or 9.');
```

```
  10..100: MESSAGE('In the range from 10 to 100.');
```

```
ELSE MESSAGE('Neither 1, 2, 9, nor in the range from 10 to 100.');
```

```
END
```

## Using Repetitive Statements

A repetitive statement is also known as a loop. The looping mechanisms in C/AL are:

- `FOR`, which repeats the inner statement until a counter variable equals the maximum or minimum value specified.



- WHILE, which repeats the inner statement as long as the specified condition is TRUE. The statement in a loop of this type is repeated 0 or more times.
- REPEAT, which repeats the inner statements until the specified conditions evaluate to TRUE. The statements in a loop of this type are always executed at least once.

**The FOR TO/DOWNTO Control Structure**

The syntax of the FOR TO (and FOR DOWNTO) statement is:

```
FOR <Control Variable> := <Start Number> TO <End Number> DO
<Statement>
```

The data type of <Control Variable>, <Start Number> and <End Number> must be boolean, number, time or date.

Use FOR statements when you want to execute some code a specific number of times. Use a control variable to control the number of times the code is executed. The <Control Variable> can be increased or decreased by one, depending on whether TO or DOWNTO is used.

**When declaring the type of the <Control Variable>...**

.....

When the system executes the FOR statement, the <Start Number > and <End Number> are converted to the same data type as <Control Variable>, if necessary. This type conversion can cause a runtime error.

.....

When using a FOR TO loop, the <Statement> will not be executed if the <START NUMBER> is greater than the end value. Correspondingly, the <Statement> will not be executed in the FOR DOWNTO loop if the start value is less than the end value.

**Note**

.....

If the value of the control variable is changed inside the FOR loop, the behavior of the system is not predictable. Furthermore, the value of the control variable is undefined outside the scope of the FOR loop.

.....

**Example**

Create the following variable.:

Variable	Data Type
Count	Integer

The following initiates a FOR loop that uses the integer control variable named Count.

```
FOR Count := 1000 TO 1000000000000000 DO
```

When this statement is executed, a runtime error occurs because the system tries to convert the start and end values to the same data type as the control variable; but because Count has been declared as an Integer variable, an error occurs when the system attempts to convert 1000000000000000 because this end value is outside the valid range for Integers.

**Example**

This example illustrates how to nest FOR statements.

Create the following variables;

Variable	Data Type
I	Integer
J	Integer
A	Integer

Set the *Dimensions* property of variable A to 5;7.

The following two FOR statements could be used to initialize every element in a 5 x 7 array with the value 23.

```
FOR I := 1 TO 5 DO
    FOR J := 1 TO 7 DO
        A[I,J] := 23;
```

**The WHILE DO Control Structure**

The WHILE DO statement has the following syntax:

```
WHILE <Condition> DO <Statement>
```

If <Condition> is TRUE, <Statement> is executed repeatedly, until <Condition> becomes FALSE. If <Condition> is FALSE from the start, <Statement> is never executed.

The WHILE DO statement can be used when some code should be repeated as long as an expression is TRUE.

**Example**

Create the following variable:

Variable	Data Type
i	Integer

This C/AL code increases the variable *i* until it equals 1000 and displays a message when it is finished:

```
WHILE i < 1000 DO i := i + 1;
Message(format(i));
```

**The REPEAT UNTIL Control Structure**

The syntax for the REPEAT UNTIL statement is:

```
REPEAT <Statements> UNTIL <Condition>
```

<Statements> is executed repeatedly until <Condition> is TRUE.

At first glance, this might seem to function just like a `WHILE` control structure. However, because the `REPEAT UNTIL` statement is executed from left to right, the `<Statements>` is always executed at least once, no matter what the `<Condition>` is evaluated to. This contrasts with the `WHILE` control structure, which performs the evaluation before the `<Statement>` is executed – and means that if the first evaluation of `<Condition>` returns `FALSE`, then no statements are executed.

### Example

Create the following variables:

Variable	Data Type	Subtype
Count	Integer	
Customer	Record	Customer

This is a typical example of a `REPEAT UNTIL` control structure:

```
Count := 0;
IF Customer.FIND('-') THEN
REPEAT
    Count := Count + 1;
UNTIL Customer.NEXT <= 0;
    Message('The Customer table contains %1 records.',Count);
```

This code uses a `REPEAT UNTIL` loop to count the number of entries in the **Customer** table. The `FIND` function finds the first entry in the table. Each time `NEXT` is called, it steps one record forward. When `NEXT = 0` there are no more entries in the table. The system exits the loop and displays a message telling you how many entries were found.

## The EXIT Statement

The `EXIT` statement is used to control the flow of the execution. The syntax of an `EXIT` statement is:

```
EXIT([<Value>])
```

An `EXIT` statement is used to interrupt the execution of a C/AL trigger. The interruption takes place even when the code is executed inside a loop or a similar structure. The `EXIT` statement is also used when a local function should return a value: `EXIT(Value)`.

Using `EXIT` without a parameter in a local function corresponds to using the parameter value 0. That is, the C/AL function will return the value 0 or "" (empty string).

A compile-time error occurs if `EXIT` is called with a return parameter from:

- system-defined triggers.
- local functions that are not supposed to return a value.

### Example

The following example illustrates the use of the `EXIT` statement in an arbitrary local function. Assume that the `IF` statement is used to detect an error. If the error condition is met, the execution is stopped and the local function returns the error-code 1.

```

FOR I := 1 TO 1000 DO
BEGIN
    IF Amount[I] < Total[I] THEN EXIT(1);
    A[I] := Amount[I] + Total[I];
END;

```

## The WITH Statement

The syntax of a WITH statement is:

```
WITH <Record> DO <Statement>
```

When you work with records, addressing is carried out as record name, dot (period) and field name: <Record>.<Field>

If you work continuously with the same record, you can use WITH statements. When you use a WITH statement, you only have to specify the record name once.

Within the scope of <Statement>, fields in <Record> can be addressed without having to specify the record name.

Several nested WITH statements can be used. In case of identical names, the inner WITH overrules the outer WITH-statements.

### Example

This example shows two ways of writing the same code that creates a record variable that you can commit later.

Create the following variable:

Variable	Data Type	Subtype
CustomerRec	Record	Customer

```

CustomerRec.No. := '1234';
CustomerRec.Name := 'Windy City Solutions';
CustomerRec.Phone No. := '555-444-333';
CustomerRec.Address := '1241 Druid Avenue';
CustomerRec.City := 'Windy City';
MESSAGE('A variable has been created for this customer.');
```

Another way of expressing the same is:

```

WITH CustomerRec DO
BEGIN
    No. := '1234';
    Name := 'Windy City Solutions';
    Phone No. := '555-444-333';
    Address := '1241 Druid Avenue';
    City := 'Windy City';
    MESSAGE('A variable has been created for this customer.');
```

END;

## How to Annotate Your Programs

You can insert comments about the code or "outcomment" parts of your code to prevent execution.

There are two ways to insert comments:

- Use "//" to insert a single line comment. When the compiler encounters the "//" symbol in your code, it interprets the rest of the line as a comment.
- Use "{" and "}" to mark the beginning and end, respectively, of a block of comments.

You can nest any number of comments. In such cases, the comment runs from the first comment start to the last comment end.

### Example

```
{
This is a sample comment which is ignored by the C/AL compiler
}
```

### Example

```
// This is also a sample comment which is ignored by the C/AL
compiler
```

### Example

```
{ This comment { is partly inside } another comment }
```

### Example

The final example illustrates what you shouldn't do:

```
A := 34;
B := 56;      {*****
C := 345;     * Don't do this! *
D := 781;     *****}
```

Because the comment is to the right of the C/AL statements, the system assumes that the third and fourth lines are part of the comment. That is, only A and B are assigned values, while C and D are not. Instead you should use single line comments:

```
A := 34;
B := 56;      //*****
C := 345;     /* Do it this way! *
D := 781;     //*****
```



## Chapter 17

### Using C/AL

This chapter describes some aspects of using C/AL. The first section gives advice on using the system-defined variables. The second describes how to handle functions that can generate runtime errors, depending on how they are used. The last, and largest, section provides an overview of a subset of C/AL functions and examples of how to use them. The functions in this subset are the most commonly used, and if you understand how to use them, you will be able to create quite sophisticated C/SIDE applications.

- Overview
- System-Defined Variables
- Handling Runtime Errors
- The Essential C/AL Functions

## 17.1 Overview

This chapter describes how to use C/AL. The first sections concentrate on giving some advice on things that you should consider when you use C/AL – the most important subject being where to place the code.

The concepts of system-defined variables and runtime errors are explained. The final section is devoted to the most commonly used C/AL functions. This includes a detailed description of each function as well as examples of how to use them.

### Where to Write C/AL Code

As described earlier, almost every object in C/SIDE contains triggers where you can place your C/AL code. There are triggers for:

- Tables
- Table fields
- Forms, including request options forms
- Form controls
- Reports
- Data items
- Sections

You can initiate the execution of your C/AL code from:

- Command buttons
- Menu items

You can also place C/AL code in codeunits and call it from code in any of the locations mentioned earlier.

As you can see, you can put C/AL code in a variety of places and initiate or trigger its execution in many ways. You should not, however, choose the location for your C/AL code at random.

Here are a few simple guidelines that you should follow:

- In general, place the code as close as possible to the object that it will operate on. This implies that code that modifies records in the database should normally be placed in the triggers of the table fields that are involved.
- In reports, there should always be a clear distinction between logical and visual processing, and you should position your C/AL code accordingly. This implies that it is acceptable to have C/AL code in a section trigger – if that code controls either the visual appearance of the controls or whether or not the section should be printed. On the other hand, you should never place code that manipulates data in section triggers.
- The principle of placing code near to the object it operates on can be overruled in some situations. One very good reason is security. Normal users do not have direct access to tables that contain sensitive data – such as the **General Ledger Entry** table. If you place the code that operates on the general ledger in a codeunit and give the codeunit access to the table and the user permission to execute the



codeunit, you will not compromise the security of the table, and the user will still be able to access the table.

- There are reasons other than security for putting a posting function like the one described earlier in a codeunit. A function that is placed in a codeunit can be called from many places in the application – including, perhaps, some that you did not have in mind when you first designed the application.

## Reusing Code

Perhaps the most important reason for placing C/AL code consistently, and as close to the objects it manipulates as possible, is that it lets you reuse code. Reusing code makes developing applications both faster and easier. However, this is not the most important reason for reusing code whenever you can. If you place your C/AL code as suggested, your applications will be less prone to errors.

By centralizing the code, you will not inadvertently create inconsistencies by performing essentially the same calculation in many places, for example, in a number of control triggers that have the same table field as their source expression. If the code has to be changed, you could easily either forget about some of these controls or make a mistake when editing one of them.

## 17.2 System-Defined Variables

C/SIDE automatically declares and initializes a number of variables that you can use when you are developing applications. These are the system-defined variables:

Variable	Comments
Rec xRec	When a record is modified, the Rec variable contains the current record (including the changes that are made), while the xRec variable contains the original values (before the changes).
CurrForm	Refers to the current form. You can access the controls of the form through this variable and set the dynamic properties of the form and its controls.
CurrReport	Refers to the current report in the same way as CurrForm refers to the current form.
RequestOptionsForm	Refers to the request options form of the current report.
CurrFieldNo	The field number of the current field in the current form – retained for compatibility reasons.

In addition, some triggers (for example, the OnFormat trigger of a control) have a parameter that is defined as a local variable by the system.

### Example

Here is an example of how to use the Rec/xRec pair of records. In an application, data is stored in two tables, a header table and a line table. The header table contains general information about, for example, sales orders, while the line table contains the specific order lines. The form that you use to enter information into the header table has fields that contain the customer's address. These fields are related to the **Customer** table, and can be filled out by using a lookup function in the field that establishes the relationship. In the header table, only the customer number is stored, and the other fields with customer information (name, address, and so forth) are retrieved from the **Customer** table when the **Customer No.** field is validated.

Now, should the user be able to change the customer number? In some situations the answer would be yes, in others no. If the order has already been shipped, the answer should definitely be *no*, but there could be situations where it would be *yes* – it should, for example, be possible to correct an erroneous number on an order that has not been processed completely.

You could do something like this:

- When validating the customer number field, check whether or not the order has been shipped.
- If it has been shipped, compare the customer number fields in the xRec and Rec records. If they differ, reject the change.

In real life, you would certainly add some more checks and some user dialog, but this is the basic idea.

## 17.3 Handling Runtime Errors

In chapter 24, "Debugging C/AL Code", the section "Other Run-time Errors" describes how to handle functions that return a Boolean value that can be either processed or ignored.

When you use these functions, four different scenarios are possible, as shown in the following table:

	Return value is ignored	Return value is processed
Function succeeds	Execution continues	Execution continues
Functions fails	A runtime error occurs	Execution continues, and you must handle the situation yourself

A typical example of a function that can produce a runtime error, depending on how you handle the return value is *GET*. The syntax is:

```
[Ok :=] Record.GET([Value1], [Value2 ],...)
```

*Ok* is a Boolean value, which is TRUE if the record is found and FALSE if the record is not found.

If *GET* is used as follows:

```
Customer.GET("Customer Number");
```

and no record is found, a run-time error occurs.

If, on the other hand, *GET* is used as follows:

```
IF Customer.GET("Customer Number") THEN
    ....
ELSE
    ...
```

and no record is found, execution continues. In this case, you need to handle the situation yourself in the *ELSE* part of the statement.

## 17.4 The Essential C/AL Functions

Although there are more than 100 functions in C/AL, you will find that you are constantly using a limited set of these functions, and only use the rest of the functions occasionally. When you are developing a basic application, you use perhaps no more than 20 different functions. This does not mean that the rest of the functions are obsolete or that you will never use them. However, it does mean that if you are very familiar with this small set of essential functions, you will be able to go a long way when you are programming in C/AL. As you need to add more specialized functionality to your applications, you can familiarize yourself with more of the functions.

The following sections contain some examples of how to use the essential functions. You should, however, always refer to the *C/SIDE Reference Guide* online Help for more information on any C/AL function.

### Searching For Records

The three functions described in this section are used to search for records. When you search for records, it is important to remember the difference between `GET` and `FIND` – and how to use `FIND` and `NEXT` in conjunction.

#### GET

`GET` retrieves one record, based on values of the primary key fields.

`GET` has the following syntax:

```
Ok := Record.GET([Value] , ...)
```

For example, if the **No.** field is the primary key of the **Customer** table, `GET` could be used like this:

```
GET(Customer, '4711');
```

The result is that the record of customer 4711 is retrieved. However, `GET` is one of those functions that produce a runtime error if it fails and the return value is not inspected by the code. This means that the actual code you write should look more like this:

```
IF GET(Customer, '4711') THEN
    .... // do some processing
ELSE
    .... // do some error processing
```

`GET` searches for the records, regardless of the current filters, and it does not change any filters. In other words: `GET` always searches through all the records in a table.

#### FIND

`Find` locates a record in a C/SIDE table based on the values stored in the keys.

`Find` has the following syntax:

```
Ok := Record.FIND([Which])
```

The important difference between `GET` and `FIND` are:

- `FIND` respects (and is limited by) the current filters.
- `FIND` can be instructed to look for records where the key value is equal to, larger than or smaller than the search string.

- `FIND` can find the first or the last record (depending on the sort order defined by the current key).

You can use these features in various ways. When you are developing applications in a relational database, there are often one-to-many relationships defined between tables. An example could be the relationship between an **Item** table, which registers items, and a **Sales Line** table, which registers the detailed lines from sales orders. Obviously, one record in the **Sales Line** table can only be related to one item, but each item can be related to any number of sales line records.

You would not want an item record to be deleted as long as there are still open sales orders that include the item. You can use `FIND` to check whether or not this is the case.

Look at the OnDelete trigger of the **Item** table. It includes the following code that illustrates:

```
SalesOrderLine.SETCURRENTKEY(Type, "No. ");
SalesOrderLine.SETRANGE(Type, SalesOrderLine.Type::Item);
SalesOrderLine.SETRANGE("No.", "No. ");
IF SalesOrderLine.FIND('-') THEN
    ERROR(Text001, TABLECAPTION, "No.", SalesOrderLine."Document Type");
```

#### NEXT

`NEXT` is often used with `FIND` to step through the records of a table.

`NEXT` has the following syntax:

```
Steps := Record.NEXT([Steps])
```

as in this fragment:

```
FIND('-');
REPEAT
    // process record
UNTIL NEXT = 0;
```

Here, `FIND` is used to go to the first record of the table. Afterwards, `NEXT` is used to step through every record, until there are no more (then, `NEXT` returns 0 (zero)).

## Sorting and Filtering Records

The following functions are used to filter records in a table, that is: to set limits on the value of one or more specified fields, so that only a subset of the records are displayed, modified, deleted, and so forth. This section also shows you how to change the sort order of the records in a table.

#### SETCURRENTKEY

This function is used to select a key for a record, and thereby set the sort order that is used for the table in question.

`SETCURRENTKEY` has the following syntax:

```
[Ok :=] Record.SETCURRENTKEY(Field1, [Field2], ...)
```

Remember the following points when you use `SETCURRENTKEY`:

- Inactive fields are ignored.

- When searching for a key, C/SIDE selects the *first* occurrence of the specified field(s).

This means that:

- if you specify only one field as a parameter when you call `SETCURRENTKEY`, the key that is actually selected may consist of more than one field.
- if the field that you specify is the first component of several keys, the key that is selected may not be the key that you expect.
- if no keys can be found that include the field(s) that you specify, a runtime error occurs unless you test the Boolean return value of `SETCURRENTKEY` in your code.

If you do test the return value, you must decide what the program should do if the function returns `FALSE`, because without a runtime error, the program will continue to run even though no key was found.

#### SETRANGE

This function is used to set a simple filter on a field.

`SETRANGE` has the following syntax:

```
Record.SETRANGE(Field [,From-Value] [,To-Value]);
```

In this example:

```
Customer.SETRANGE("No.", '10000', '90000');
```

`SETRANGE` filters the **Customer** table by selecting only those records where the **No.** field has a value between 10000 and 90000.

When you use `SETRANGE` you must remember that:

- `SETRANGE` removes any filters that were set earlier and replaces them with the From-Value/To-Value parameters that you specify.
- If you use `SETRANGE` without setting the From-Value/To-Value parameters, the function removes any filters that are already set.
- If you only set the From-Value, the To-Value is set to the same value as the From-Value.

#### SETFILTER

`SETFILTER` sets a filter in a more general way than `SETRANGE`.

`SETFILTER` has the following syntax:

```
Record.SETFILTER(Field, String [, Value], ...);
```

Field is the name of the field that you want to set a filter on. String is a filter expression that may contain %1, %2 and so on to indicate the locations where the system will insert the given values (but not operators) as the Value parameter(s) in a filter expression.

Here are two examples:

```
Customer.SETFILTER("No.", '>10000 & <> 20000');
```

This statement selects records where the No. is larger than 10000 and not equal to 20000.

```
Customer.SETFILTER("No.", '>%1&<>%2', Value1, Value2);
```

If the variables Value1 and Value2 have been assigned "10000" and "20000", respectively, this statement will have the same effect as the first one.

**GETRANGEMIN** This function retrieves the minimum value of the filter range that is currently applied to a field.

GETRANGEMIN has the following syntax:

```
Record.GETRANGEMIN(Field);
```

A run-time error occurs if the filter that is currently applied is not a range. If, for example, a filter has been set as follows:

```
Customer.SETFILTER("No.", '10000|20000|30000');
```

then

```
BottomValue := Customer.GETRANGEMIN("No.");
```

fails, because the filter is not a range.

**GETRANGEMAX** GETRANGEMAX works like GETRANGEMIN, except that it retrieves the maximum value of the filter range that is currently applied to a field.

GETRANGEMAX has the following syntax:

```
Value := Record.GETRANGEMAX(Field)
```

## Inserting, Modifying and Deleting Records

These functions are used to *maintain* the database by adding, modifying and removing records.

Generally, these functions return a Boolean value that indicates whether or not the function succeeded. If you do not handle the return value in your code, a run-time error occurs when a function returns FALSE. If you handle the return value – by testing its value in an IF statement – no error will occur, and you must take corrective action yourself (knowing that the function did not succeed, of course).

INSERT This function inserts a record in a table.

INSERT has the following syntax:

```
[Ok :=] Record.INSERT([RunTrigger])
```

### Example

Create the following variable:

Variable	Data Type	Subtype
Customer	Record	Customer

```
Customer.INIT;
Customer."No." := '4711';
Customer.Name := 'John Doe';
Customer.INSERT;
```

This statement inserts a new record, with the No. and Name specified in the assigned values, while other fields will have their default values. If No. is the primary key of the **Customer** table, the record will be inserted in the **Customer** table unless the table already contains a record with the same primary key. In this case you receive an error message because the return value is not tested.

MODIFY This function is used to modify a record that already exists.

MODIFY has the following syntax:

```
[Ok :=] Record.MODIFY([RunTrigger])
```

Like INSERT, it returns a Boolean – TRUE, if the record to be modified exists and FALSE if it doesn't exist.

### Example

Create the following variable:

Variable	Data Type	Subtype
Customer	Record	Customer



```
Customer.GET('4711');
Customer.Name := 'Richard Roe';
Customer.MODIFY;
```

These statements change the name of customer 4711 to Richard Roe.

#### MODIFYALL

This function is used to do a bulk update of records.

MODIFYALL has the following syntax:

```
Record.MODIFYALL(Field, NewValue [, RunTrigger])
```

MODIFYALL respects the current filters, meaning that you can perform the update on a specified set of records within a table. MODIFYALL does not return any value, nor does it cause an error if the set of records to be changed is empty.

#### Example

Create the following variable:

Variable	Data Type	Subtype
Customer	Record	Customer

```
Customer.SETRANGE("Salesperson Code", 'PS', 'PS');
Customer.MODIFYALL("Salesperson Code", 'JR');
```

The SETRANGE statement selects the records where **Salesperson Code** is PS, and MODIFYALL changes the **Salesperson Code** of these records to JR.

#### DELETE

This function is used to delete a record from the database.

DELETE has the following syntax:

```
[Ok :=] Record.DELETE([RunTrigger])
```

The record that you want to delete must be specified (using the value(s) in the primary key fields) before calling this function. This means that DELETE does take filters into consideration.

#### Example

Create the following variable:

Variable	Data Type	Subtype
Customer	Record	Customer

Here is an example in which DELETE is used to delete the record for customer number 4711:

```
Customer."No." := '4711';
Customer.DELETE;
```

DELETE returns a Boolean value: TRUE if the record could be found, FALSE if it could not be found. Unless you test this value yourself, a runtime error occurs when DELETE fails (returns FALSE).

When you are developing your own applications, you should consider the following scenario:

- 1 First you retrieve a record from the database.
- 2 Then you perform various checks to determine whether the record should be deleted.
- 3 If step 2 indicated that you should delete the record, you delete it.

Now, this can cause problems in a multi-user environment. Another user can modify or delete the same record between your performing steps 2 and 3. If the record is modified, then perhaps the new contents of the record would have changed your decision to delete it. If it has been deleted by the other user, you can get a seemingly inexplicable run-time error if you have just verified that the record existed (in step 1).

If the design of your application indicates that you can encounter this problem, you should consider using the LOCKTABLE function (described in the next section) – but LOCKTABLE should be used as sparingly as possible, because this function effectively short-circuits the concept of optimistic concurrency, thereby degrading performance.

DELETEALL

This function is used to delete all the records that are specified by the filter settings. If no filters are applied, all the records in the table are deleted.

DELETEALL has the following syntax:

```
Record.DELETEALL([RunTrigger])
```

**Example**

Create the following variable:

Variable	Data Type	Subtype
Customer	Record	Customer

The following statements delete all the records from the **Customer** table where the **Salesperson Code** is PS:

```
Customer.SETRANGE("Salesperson Code", 'PS', 'PS');
Customer.DELETEALL;
```

**Note**

.....

When you use DELETEALL(TRUE), C/SIDE creates a copy of the C/AL variable with its initial values. This means that when you use DELETEALL(TRUE) to run the OnDelete trigger, all the changes that were made to the variables in the function or codeunit that is making the call cannot be seen in the OnDelete trigger. If you want to see the changes made to the variables, you must use Delete(TRUE) in a loop. There is no difference in performance between using DELETEALL(TRUE) and using Delete(TRUE) in a loop.

.....

## Transactions

Normally, you do not need to take transactions and table locking into consideration when you are developing applications in C/SIDE. The chapter C/SIDE in Multiuser Environments on page 519 explains the details.

There are, however, some situations where you have to lock a table explicitly. For example, in the beginning of a function, you inspect data in a table and then use this data to perform various checks and calculations. Finally, you want to insert a record based on the result of these calculations.

To ensure that your calculations make sense, you must be certain that the values you retrieved at the beginning of the transaction are consistent with the data that is in the table now. In other words, you cannot allow other users to update the table while your function is performing its calculations.

### LOCKTABLE

The solution to this problem is to use the `LOCKTABLE` function to lock the table at the start of your function.

`LOCKTABLE` has the following syntax:

```
Record.LOCKTABLE([Wait] [, VersionCheck])
```

For more information about concurrency in Dynamics NAV, see "Read Consistency and Concurrency" on page 522.

## Working with Fields

The following functions perform various actions on fields.

### CALCFIELDS

The `CALCFIELDS` function is used to update FlowFields. As described in "Form and Control Properties" on page 152, FlowFields are automatically updated when they are the direct source expressions of controls, but they must be explicitly calculated when they are part of a more complicated expression.

`CALCFIELDS` has the following syntax:

```
[Ok :=] Record.CALCFIELDS(Field1, [Field2],...)
```

When you use FlowFields in C/AL functions, you must use the `CALCFIELDS` function to update them. In the following statements, the `SETRANGE` function sets a filter and then `CALCFIELDS` is called. The `CALCFIELDS` function calculates the **Balance** and **Balance Due** fields by taking account of the current filter and performing the calculations that are defined as the *CalcFormula* properties of the FlowFields.

#### Example

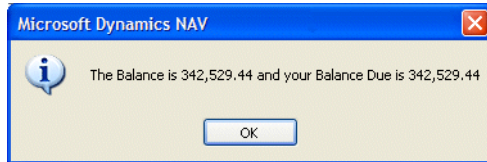
Create the following variable:

Variable	Data Type	Subtype
Customer	Record	Customer

The following code calculates the balance and balance due for customer number 01454545:

```
Customer.GET('01454545');
Customer.SETRANGE("Date Filter",0D,TODAY);
Customer.CALCFIELDS(Balance,"Balance Due");
MESSAGE('The Balance is %1 and your Balance Due is
%2',Customer.Balance,Customer."Balance Due");
```

and displays the following message:



**CALCSUMS**

The CALCSUMS function is used to calculate the sum of one or more fields that are SumIndexFields in the record.

CALCSUMS has the following syntax:

```
[Ok :=] Record.CALCSUMS (Field1, [Field2],...)
```

For CALCSUMS to work, a key that contains the SumIndexFields must be selected as the current key. Like CALCFIELDS, CALCSUMS takes the current filter settings into account when performing the calculation.

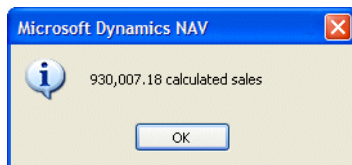
**Example**

Create the following variable:

Variable	Data Type	Subtype
custledgerentry	Record	Cust. Ledger Entry

In this code, an appropriate key is selected, some filters are set, the calculation is performed and finally a message is displayed.

```
custledgerentry.SETCURRENTKEY("Customer No.");
custledgerentry.SETRANGE("Customer No.",'10000','50000');
custledgerentry.SETRANGE("Posting Date",0D,TODAY);
custledgerentry.CALCSUMS("Sales (LCY)");
MESSAGE ('%1 calculated sales',custledgerentry."Sales (LCY)")
```



**FIELDERROR**

The FIELDERROR function triggers a runtime error after having displayed a field-related error message.

FIELDERROR has the following syntax:

```
Record.FIELDERROR(Field, [Text])
```

The function is very similar to the `ERROR` function, described on page 342. However, it does have some uses of its own. For one thing, it is easier to use. But the most important reason for using it is that if the name of a field is changed (for example, translated to another language) in the Table Designer, the message from the `FIELDERROR` function will reflect the current name of the field.

`FIELDERROR` can be called simply:

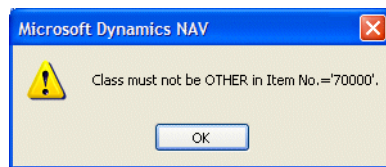
#### Example

Create the following variable:

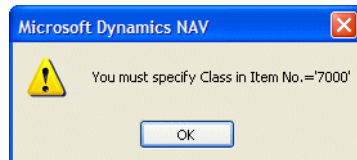
Variable	Data Type	Subtype
Item	Record	Item

```
Item.GET('70000');
IF Item.Class <> 'HARDWARE' THEN
    Item.FIELDERROR(Class);
```

If item 70000 has a Class other than hardware, an error message is displayed:



You see a message like this when a text or code field contains the empty string:



(When a numeric field is empty, it is treated as though it contains the value 0 (zero) – and will produce a message like the first one shown, with "0" instead of "FOOD".)

Finally, if the default texts don't suit your application, you can add your own text. In this case, you call `FIELDERROR` as follows:

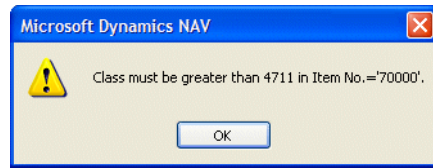
#### Example

Create the following variable:

Variable	Data Type
Class	Code

```
IF Item.Class < '4711' THEN
    Item.FIELDERROR(Class, 'must be greater than 4711');
```

and the message will look like this:



#### FIELDNAME

The `FIELDNAME` function returns the name of a field.

`FIELDNAME` has the following syntax:

```
Name := Record.FIELDNAME(Field)
```

You could simply use the name of the field, as you probably know it when you are writing the code. However, using `FIELDNAME` allows you create messages that always contain the name of the field, even if the name of the field is changed. `FIELDNAME` could be used together with `FIELDERROR`, in a construction like this:

```
FIELDERROR(
    Quantity, 'must not be less than ' +
    FIELDNAME("Quantity Shipped"));
```

#### INIT

The `INIT` function initializes a record.

`INIT` has the following syntax:

```
Name := Record.FIELDNAME(Field)
```

If a default value for a field has been defined (with the *InitValue* property), this value is used for the initialization – otherwise, the default value of each data type is used (see the *C/SIDE Reference Guide* online Help entry for `INIT`).

Note that `INIT` *does not* initialize the fields of the primary key.

#### TESTFIELD

This function is used to test whether a field contains a specific value.

`TESTFIELD` has the following syntax:

```
Record.TESTFIELD(Field, [Value])
```

If the test fails, that is if the field does not contain the specified value, an error message is displayed, and a run-time error is triggered. This means that any changes that were made to the record are discarded. If the value that you test against is an empty string, the field must have a value other than blank or 0 (zero).

#### Example

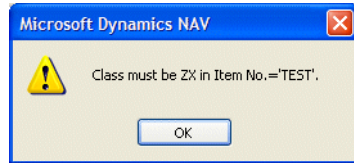
Create the following variable:

Variable	Data Type	Subtype
customer	Record	Customer

The following code:

```
customer.GET('10000')
customer.TESTFIELD("Language Code", 'ZX');
```

tests the **Language Code** field for customer number 10000 in the **Customer** table and tests whether or not the Language Code is ZX. It isn't and the code generates this error message:



## VALIDATE

The `VALIDATE` function is used to call the `OnValidate` trigger of a field.

`VALIDATE` has the following syntax:

```
Record.VALIDATE(Field [, NewValue])
```

### Example

When you enter an account number in a ledger, the system executes some table trigger code to transfer the name of the account from the chart of accounts.

If you enter an account number in a batch job, the system does not automatically execute the code which transfers the name of the account. The following code tells the system to execute the appropriate field-level trigger code.

Create the following variable:

Variable	Data Type	Subtype
GeneralLedgerEntry	Record	G/L Entry

```
GeneralLedgerEntry.VALIDATE("G/L AccountNo", '100');
```

This corresponds to:

```
GeneralLedgerEntry."G/L AccountNo" := '100';
GeneralLedgerEntry.VALIDATE("G/L AccountNo");
```

The `VALIDATE` function is useful for centralizing processing and thereby making your application easier to maintain.

For example, if that the `OnValidate` trigger of the **Total Amount** field performs a calculation that uses values from three other fields as operands, the calculation must be performed again if the contents of any of these fields changes.

You should avoid entering the calculation formula in the `OnValidate` triggers of each field because this can create all sorts of errors if the calculation formula has to be changed later and you have to update the code in all the triggers. Instead, you should enter the calculation formula in the `OnValidate` trigger of only one of the fields and call this trigger code from the `OnValidate` triggers of the other fields.

## User Messages And Dialogs

There are several specialized functions that you can use to display messages and gather input. However, you should use forms whenever possible to ensure that your application has a consistent user interface.

Needless to say, there are situations where it makes sense to use the dialog functions. The three most important uses of the dialog functions are:

- To display a window that indicates the progress of some processing that may take a long time.
- To stop the program executing, in order to display an error message
- To get the user to confirm a choice before the program continues executing.

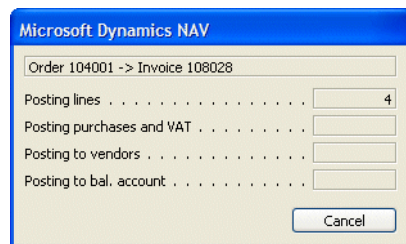
You will also find the `STRMENU` function useful for creating forms that present options to the user. It is much faster to use this function than to design a form solely to present a limited set of options to the user.

### Creating a Window to Indicate Progress

When you have written an application that performs some processing that can – for perfectly good reasons – take a long time, you should consider displaying a window that informs the user of the progress that is being made. The information contained in a progress window or indicator can be superfluous, but it is a good idea to inform the user that something is actually going on and that the program is still running.

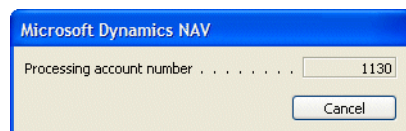
A Cancel button is automatically part of every dialog window and gives the user the opportunity to stop the processing.

In some applications, you can create an indicator control to do this. How to do this is described in the section, "Using an Indicator to Display Values" on page 182. In other applications, you can create a window like this instead:



In this window, each field is updated while the program is running. In this example, the fields are used to count the number of postings being made.

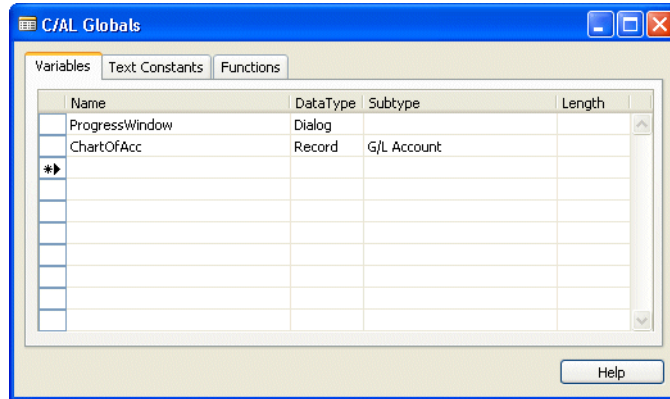
In another situation you could display, for example, the number of the account that is currently being processed, like this:





To create a window like this:

- 1 Open the Object Designer and create a new codeunit.
- 2 Create a variable called ProgressWindow of data type Dialog.
- 3 Create a variable called ChartOfAcc of data type Record and Subtype G/L Account.



- 4 Open the **C/AL Editor** window for the dialog and add the following code:

```

ProgressWindow.OPEN('Processing account number #1#####');
REPEAT
    SLEEP(1000);
    ProgressWindow.UPDATE(1,ChartOfAcc."No.");
    // process the account...
UNTIL ChartOfAcc.NEXT = 0;
ProgressWindow.CLOSE;

```

The first line defines the string that will be displayed in the progress window. The part of the string that contains the pound signs (#) and a number defines the field that will be displayed in the window. The number (1) refers to the field.

In this example, each entry in the **G/L Account** table is updated and the number each account is displayed as it is updated.

The `SLEEP(1000);` function is completely unnecessary and just slows down the processing so that you can see the progress window.

### Other User Messages

There are a number of other dialog functions that you can use to display short messages to the user. A common trait of these dialogs, except `MESSAGE`, is that the program stops executing until the user makes a response.

#### MESSAGE

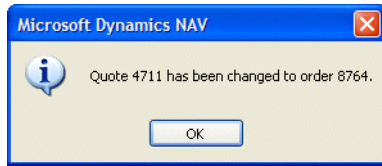
The `MESSAGE` function displays a message in a window that remains open until the user clicks the OK button on the window.

`MESSAGE` has the following syntax:

```
MESSAGE(String [, Value1, ...])
```

`MESSAGE` executes asynchronously, that is: `MESSAGE` is not executed until the function from which it was called ends or another function requests user input. The function is

useful for notifying the user that some processing has been successfully completed, as in this example:



For an example of this see Codeunit 83. The code in the OnRun trigger (among other things) converts a Quote into a Sales Order and ends with a message. The message is generated by the following code:

```
MESSAGE(Text001, "No. ", SalesHeader2. "No. ");
```

Text001 is a text constant that contains the following text:

*Quote %1 has been changed to order %2.*

**Note**

.....  
 Unlike the progress window example, the MESSAGE function was used without first declaring a variable of type Dialog, because the MESSAGE function creates a window of its own.  
 .....

**ERROR** The ERROR function is very similar to the MESSAGE function, except for one detail: when the user has acknowledged the message, execution ends. See also the description of FIELDERROR on page 336.

ERROR has the following syntax:

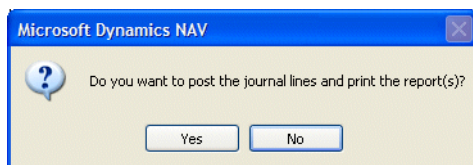
```
ERROR(String [, Value1, ...])
```

**CONFIRM** The CONFIRM function is used just like the MESSAGE function to display a message.

CONFIRM has the following syntax:

```
Ok := Dialog.CONFIRM(String [, Default] [, Value1] ,...)
```

However, unlike the MESSAGE function, the CONFIRM function returns a value that can (and must) be used, depending on whether the user chooses Yes or No. Its obvious use is to ask a question like this:



The window is created by a statement like this:

```
IF CONFIRM('Do you want to post the journal lines and print the report(s)?', FALSE) THEN
    Message('Posting')
```

```

ELSE
    Message('No Posting');
EXIT;

```

The `FALSE` parameter means that No is the default.

## A Quick Options Form

The `STRMENU` function is used to create and display a form with an option group, and to return the selection that the user makes to the program.

`STRMENU`

`STRMENU` has the following syntax:

```
OptionNumber := Dialog.STRMENU(OptionString [, DefaultNumber]);
```

`OptionNumber` is the number of the option that the user chooses. The first option in the `OptionString` is number 1. If the user closes the form with `ESC`, `STRMENU` returns 0 (zero). If it is defined, `DefaultNumber` is used to select the default option. If `DefaultNumber` is not defined, the system will use option number 1 as the default.

### Example

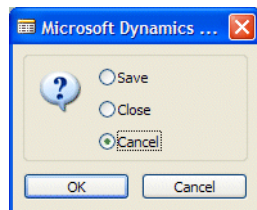
Create the following variable:

Variable	Data Type
Selection	Integer

The following code:

```
Selection := Dialog.STRMENU('Save,Close,Cancel',3);
```

creates the following window:



Notice that `Cancel` is the default option because the `DefaultNumber` parameter was set to 3. It is a good idea to let the default option be a "harmless" action, like `Cancel`, because this option can be selected by pressing `ENTER`. If the user inadvertently presses `ENTER`, nothing disastrous happens, as might be the case if, for example, one of the options was "Delete all".



## **Chapter 18**

### **Debugging C/AL Code**

This chapter describes the nature of program errors, bugs, and how to use the Dynamics NAV Debugger to track down errors.

- What Are Bugs?
- The Microsoft Dynamics NAV Debugger
- The Code Coverage Tool

## 18.1 What Are Bugs?

There are three categories of errors you can meet when you develop applications that use C/AL code

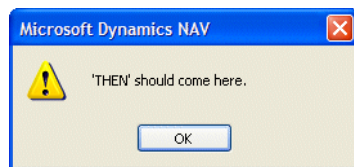
- Syntax errors
- Run-time errors
- Program logic errors

Traditionally, errors in computer programs are called *bugs*, and the process of finding and correcting errors is, correspondingly, called *debugging*.

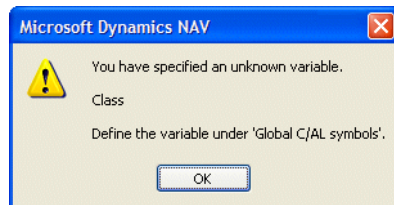
This chapter describes how you can find and eliminate bugs and errors, and it shows how you use the Dynamics NAV Debugger to find run-time and program logic errors.

### Syntax Errors

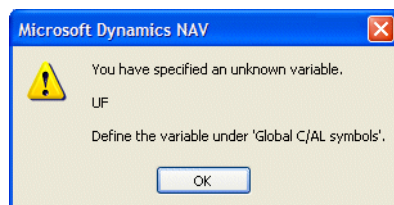
These errors are detected by the C/AL compiler when you try to compile C/AL code, be it in a codeunit or as code in another object (table, form, report, dataport or codeunit). The compiler will notify you of the error with a message like this:



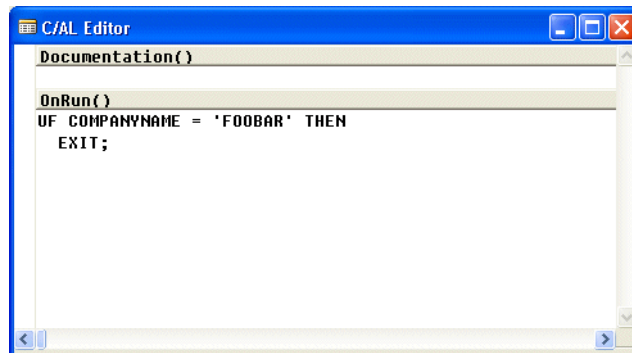
or this:



When you have pressed ENTER and acknowledged the error message, the C/AL editor will appear with the cursor in front of the offending expression. Note that the error message may not always reflect the nature of the error. Consider this message:



When you look at the offending code in the editor, it becomes clear that the error has nothing to do with an unknown variable:



The real error is a misspelling of `IF`, which has been entered as `UF`. From the point of view of the compiler, `UF` is an unknown identifier, hence the error message. When you look at the code, however, it is easy to see what was really the matter.

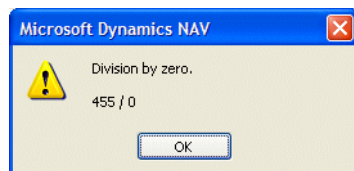
The compiler will not compile code that contains any syntax errors, like a missing `THEN` in an `IF` statement, or code that uses undeclared variables.

## Run-time Errors

Run-time errors occur when the program is executed. These errors are not detected by the compiler, because the code is syntactically correct in these cases. A good example is division by zero. Consider this statement:

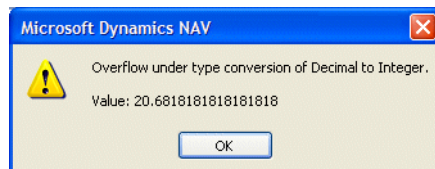
```
Ratio := First_number / Second_number;
```

There is nothing wrong with the syntax, but the statement may cause the following error:



This error occurs because the `Second_number` variable has been assigned a value of 0 (zero), causing a division by zero.

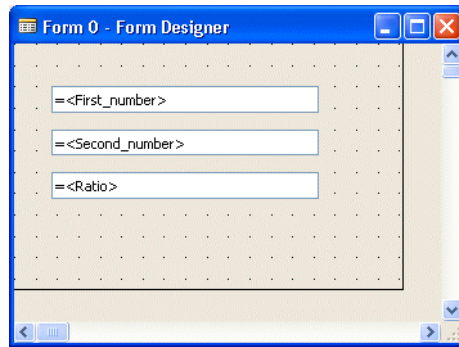
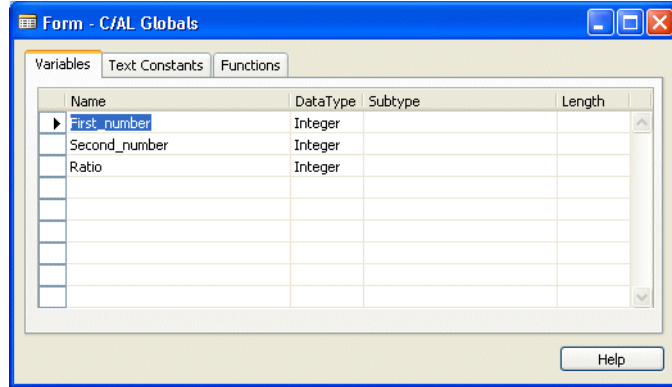
If all three variables are of type integer, the following error could occur:



This error occurs because the result of the division cannot be contained in an integer. Therefore, the result is converted to decimal, but then the conversion back to integer (to fit the result into the `Ratio` variable) fails.

The common trait of these errors is that the code works perfectly in many situations, but fails in others. The real danger is that since there is nothing syntactically wrong with the code, the error could occur when the program is already in use. Unless you handle the run-time error in your code, the default messages shown earlier will appear.

If, as in the example, the division by zero was attempted using three variables that were assigned values in a simple form, the form could have been designed like this:



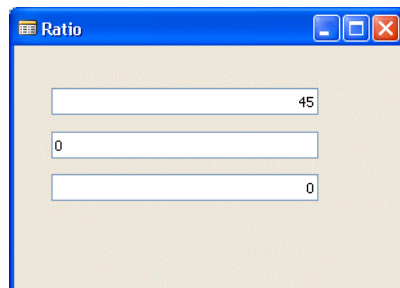
and, finally, the expression

```
Ratio := First_number / Second_number;
```

was entered in the OnValidate trigger of the Second\_number text box.

Then, when the user enters a zero in the second text box the run-time error shown earlier appears.

After the user clicks OK and acknowledges the run-time error, the form looks like this:





Now, the user cannot move out of the `Second_number` text box, or close the form, without entering a value other than 0 (zero).

#### About Run-time Errors and Data Consistency

You may now be wondering whether or not run-time errors can compromise the integrity of the database. For example, if some fields are updated in a trigger and a run-time error occurs while some other fields have not been updated. The chapter "C/SIDE in Multiuser Environments" on page 519 explains how data integrity is always maintained, under all circumstances. When a trigger is entered, a write transaction begins. If a run-time error occurs inside the trigger, the write transaction is rolled back and the execution of the trigger is terminated.

#### How to Avoid Run-time Errors

Basically, run-time errors should never occur, and they won't, if you exercise care when programming. The following description contains some guidelines on how to avoid run-time errors, but they are only guidelines, as the conditions under which run-time errors occur are highly dependent on the context of your application. If, for example, you use the `GET` function to locate a record, you must handle the possibility that a run-time error can occur if there are situations where no record is found. On the other hand, if you are absolutely certain that the specific context precludes this situation, you can omit handling a possible run-time error. (The context could be that the existence of a record is verified before the `GET` function is used.)

Generally speaking, there are two categories of run-time error: those that are related to the use of data types, and those that occur if a function does not succeed in doing what it is supposed to do. Division by zero does not fit readily into either of these categories, but it has been placed in the first one.

The heading of this section is, perhaps, overly optimistic: you can only prevent some errors (mainly the data type-related ones) from occurring. Other errors cannot always be avoided, but you can write code that shields the user from the error. That is, instead of the default error handling (which amounts to displaying a message, closing the form that was active when the error occurred and rolling back any changes to the database), you can write a better error handler that, for example, gives the user a chance to correct the input that caused the error, or, at least, displays a message that explains in further detail why the error occurred.

#### Data Type-Related Errors

The easiest way to avoid this category of run-time error is to use the correct data types. Errors like the type conversion error shown earlier and overflow errors, can be avoided by using the correct data types. In the context of this example, integer was obviously not a good choice for the `Ratio` variable. See "Introducing the C/AL Language" on page 295 for a description of the data types, and "Type Conversion" on page 505 for a description of how type conversion works in C/SIDE.

The division by zero error could have been avoided in several ways, depending upon the context where the code fragment is used. If the user enters the denominator (the `Second_number` variable) in a text box immediately before the evaluation of the

statement, you could test the value of `Second_number` before performing the division, and reject a value of 0 (zero):

```
IF Second_number <> 0 THEN
    Ratio := First_number / Second_number
ELSE
    MESSAGE('Second_number must not be 0');
```

If `Second_number` is a field in a database table, and it should never be allowed to have a value of 0 (zero), the best place to perform this check is in the `OnValidate` trigger of the field. In this way, you ensure that a value of 0 (zero) can never be entered in the field, no matter how many different forms and text boxes are used to enter data in the field.

### Other Run-time Errors

Any function that can fail to accomplish what it is intended to do can cause a run-time error. A good example is the `GET` function, which is used to locate a record in a table according to criteria that you specify. For more information, see the *C/SIDE Reference Guide* online Help for the `GET` function.

The syntax of the `GET` function is:

```
[Ok :=] Record.GET([Value1], [Value2 ],...)
```

The return value of the function is `Ok`, a Boolean. If a record is found, the return value is `TRUE`, otherwise it is `FALSE`. This return value can be ignored, as indicated by the square parentheses. If it is ignored, and the requested record cannot be found, a run-time error occurs and a system-generated error message is displayed. If, on the other hand, you test the return value, a run-time error does not occur, as it is then assumed that you handle the condition yourself.

The *C/SIDE Reference Guide* online Help always tells you whether or not the other functions handle errors in the way as the `GET` function does. You can also look at the syntax description in the Symbol Menu, to see if the function you intend to use returns a value called `Ok`. If it does, you should consult the *C/SIDE Reference Guide* online Help as there are some functions that return a Boolean for other reasons than those described here. For example, the `ASCENDING` function can be used to check the sort order of a table, and in this case it returns `TRUE` if the sort order is ascending, and `FALSE` if it is descending.

#### Example

If you use the return value, in a construction like this:

```
IF NOT Customer.GET("No.") THEN
    Customer.INIT;
```

or like this

```
IF NOT Customer.GET("No.") THEN
BEGIN
    MESSAGE('Customer %1 not found', "No.");
    EXIT;
END;
```

you can shield the user from a run-time error. In the first example, if a Customer record with the given No. cannot be retrieved, an (empty) record is initialized. In the second example, the user is notified that the record cannot be found and the trigger from where the GET function was called is exited. These examples are only general guidelines. You must consider how to handle situations like these in the context of your own application.

### Finding and Correcting Run-time Errors

As you can see from the run-time error messages reproduced on page 347, this type of message is supposed to be read by the end user. Therefore, the messages do not include references to variables or functions, but rather an explanation and the "real" values that caused the error. This means that these errors can be a little harder to locate than, for example, syntax errors.

To track down a run-time error, you need an exact description of the sequence of events that led to the error; that is, what the user was doing at the time of the error, and what values the user had entered or what record caused the error.

If the error was caused by something as simple as a calculation formula that failed to check whether a division by zero was about to be carried out, you should be able to find the statement that led to the error quite easily. If, on the other hand, the circumstances that led to the error are more complicated, and you cannot pinpoint the exact place directly, you can use the debugger as described in "The Code Coverage Tool" on page 360.

### Program Logic Errors

The third major category of errors is program logic errors (strictly speaking, the term bug should perhaps be reserved for errors of this type). A program logic error is an error in an application that could compile perfectly and can run without causing run-time errors, but still fails to function as intended.

It can be argued that many, if not most, run-time errors are also program logic errors. However, the "real" program logic error will not make itself noticed in a similarly spectacular way but will quietly generate incorrect data that may not always be detected straight away.

## 18.2 The Microsoft Dynamics NAV Debugger

Dynamics NAV provides an integrated debugger to help you check, correct or modify code so that your application can build successfully, run smoothly and act as you expected.

### Overall Description

The basic concept in debugging is the *breakpoint*, which is a mark that you can set on a statement. When the program flow reaches the statement, the debugger intervenes and suspends execution (breaks) until you instruct it to continue. Without any breakpoints, the code would just run normally when the debugger is active. The state *disabled breakpoint* means that the breakpoint is still present on the statement but is momentarily disabled (execution will not stop at this breakpoint).

If you wish to track down a run-time error, you simply disable the Break on Triggers setting from within the debugger and click Go. The debugger will automatically stop execution of the code when it encounters an error.

You can also use the debugger to find a logical error. However, finding the error will not be as easy, and you must have a good understanding of how the code is supposed to work. The debugger enables you to execute your C/AL code one statement at a time while you inspect the contents of global variables, local variables and text constants at each step. In this way, you can see whether the values that are actually used differ from those you expected when you designed the application.

The Breakpoint on Triggers setting (SHIFT+CTRL+F12) is enabled by default when you activate the debugger for the first time. Otherwise the code would be executed normally because there are no breakpoints. The debugger will therefore suspend execution of the code when it reaches the first trigger. At this point you can set other breakpoints and then disable the Breakpoint on Triggers option if you want to. If you do not disable the Breakpoint on Triggers setting, the debugger will suspend execution of the code at every trigger it reaches.

The code coverage functionality, which is described on page 360, enables you to log and view code that was executed in one or more transactions. You can use this functionality as an alternative to, or in combination with, the debugger.

### Activating the Debugger

You can activate the debugger from Dynamics NAV and from NAV Application Server:

From Dynamics NAV To activate the debugger from Dynamics NAV, click Tools, Debugger, Active (SHIFT+CTRL+F11). You can also start Dynamics NAV with the debugger active from the command line by using the `debug` parameter:

#### Example

```
fin.exe debug
```

From NAV  
Application Server

To activate the debugger from NAV Application Server, you include the `debug` parameter at start-up:

#### Example

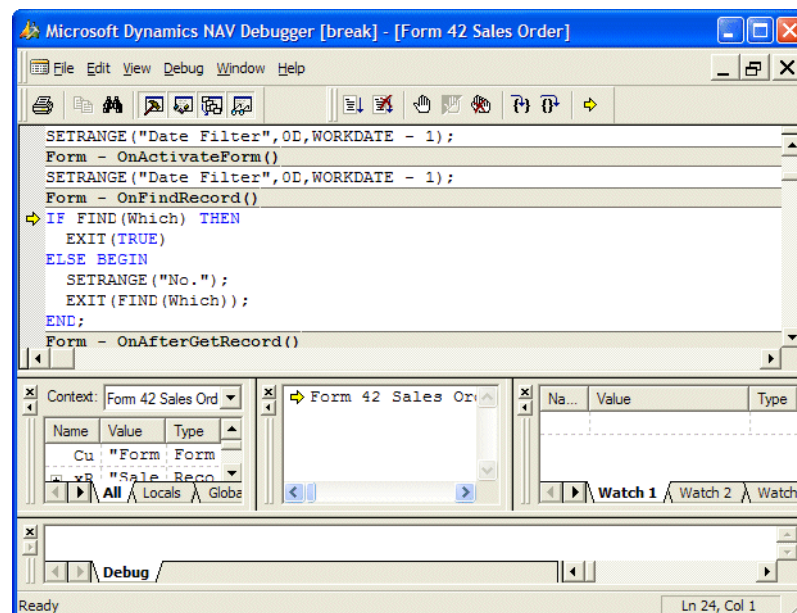
```
nas debug, startupparameter="test", servername=PC0123
```

If you deactivate the debugger, you cannot activate it again unless you terminate NAV Application Server and then start it up with the `debug` parameter.

Note that to be able to activate the debugger, there must be a developer license file in the NAV Application Server installation folder.

## The Debugger Interface

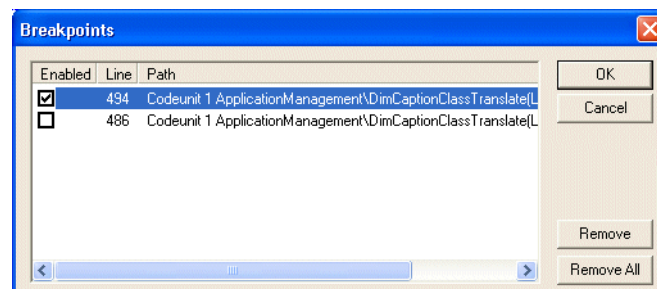
The debugger interface provides special menus, windows and a dialog box. These are described in the following.



## Debugger Menus

You can find debugging commands in the Edit, View and Debug menus:

**The Edit Menu** From this menu, you can access the **Breakpoints** dialog box (SHIFT+F9). It displays a list of the breakpoints that you have set for the object you are debugging. You can enable, disable and remove breakpoints in the list.



**The View Menu** This menu contains commands that display the various debugger windows, such as the **Variables** window and the **Call Stack** window. It also contains a command for adjusting the size of the text shown in the interface, and a command for showing/hiding the standard and debug toolbars.

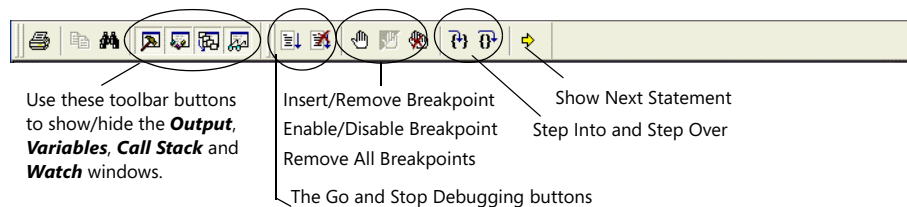
**The Debug Menu** This menu contains commands that start and control the debugging process, for example, Go, Step Into, Step Over and Show Next Statement.

- *Go* executes code from the current statement until a breakpoint or the end of the code is reached, or until the application pauses for user input.
- *Step Into* executes statements one at a time, and you can decide how to continue after each statement. The execution will step into any function that is called, which means that the debugger will single-step through the statements in the function.
- *Step Over* executes statements one at a time, like *Step Into*, but if you use this command when you reach a function call, the function is executed without the debugger stepping through the function instructions. Note, however, that if you use this command when the Breakpoint on Triggers setting is enabled, the debugger will still suspend code execution at every trigger it reaches. Furthermore, if there is a breakpoint in one of the functions you step over, the debugger will break at that breakpoint.
- *Show Next Statement* shows the next statement in your code.

The Debug menu also contains commands for setting, enabling/disabling and removing breakpoints. Note that the Breakpoint on Triggers option is set independently of other breakpoints, so the Remove All Breakpoints command does not affect it.

**The Debugger Toolbar**

The toolbar buttons represent commands that are also available from the menus.

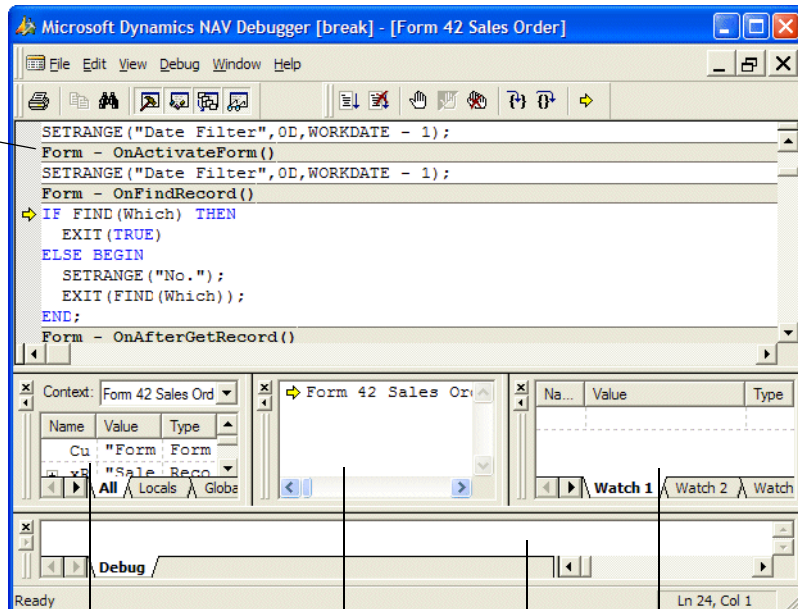


## Debugger Windows

There are four specialized windows for displaying debugging information: **Output**, **Variables**, **Call Stack** and **Watch**. You can access these windows from the View menu and from the standard toolbar.

This window simply contains the code that is being debugged for a specific object - here it is a codeunit.

All built-in functions and AL statements are shown in blue. Comments are shown in green and text strings are shown in red.



The **Variables** window    The **Call Stack** window    The **Output** window    The **Watch** window

**The Output Window** Displays information related to the debugging process.

**The Variables Window** Displays name, value and type information for variables used in the current and previous statements, including the values of an array structure. The window has four tabs: **All**, **Locals**, **Globals** and **Text Constants**. You cannot add variables to the **Variables** window (you must use the **Watch** window for that). You can expand or collapse the variables shown using the tree controls. You can expand a variable if it has a plus sign (+) box in the **Name** field. If there is a minus sign (-) box in the **Name** field, the variable is already fully expanded.

**The Call Stack Window** Displays the stack of function calls that are currently active. When a function is called, it is pushed onto the stack. The debugger displays the currently executing function at the top of the stack and older function calls below that.





When you double-click a call stack line, a green arrow appears to the left of the line. In the window that contains the code being debugged, a corresponding green arrow appears to indicate how far the debugger has reached in the specific trigger for the call stack line that you selected.

**The Watch Window** Use the **Watch** window to monitor variables of special interest while debugging your program. You can drag and drop the name of the variable that you want to watch from the **Variables** window or from the window that contains the code being debugged. You can also type the names of variables in this window.

The **Watch** window contains three tabs: **Watch1**, **Watch2** and **Watch3**. You can group variables that you want to watch together onto the same tab. For example, you could put variables related to a specific window on one tab and variables related to a dialog box on another tab. You could watch the first tab when debugging the window and the second tab when debugging the dialog box.

### Symbols used in the Debugger Interface

The symbols used in the debugger interface are as follows:

Symbol	Meaning
	There is an enabled breakpoint at this statement.
	There is a disabled breakpoint at this statement.
	This is a statement that will be executed.
	Indicates that you have double-clicked a call stack line. This arrow also appears in the window containing the code that is being debugged. Here it indicates how far the debugger has reached in the trigger for the call stack line that you selected.

### Working with Breakpoints in the C/AL Editor

To toggle between setting, enabling/disabling and removing breakpoints in the C/AL Editor, use the F9 key (or select the Tools, Debugger, Toggle Breakpoints menu command). Information about the breakpoints is stored in the **Breakpoints** virtual table when you close the C/AL Editor.

### The Breakpoints Virtual Table

The **Breakpoints** virtual table, which has ID 2000000059, can store the following information about the breakpoints that you set:

Field	Description
Object ID	The ID of the object for which breakpoint information has been stored.
Object Type	Table, Form, Report, Dataport or Codeunit.
Trigger Line	The number of the trigger line where there is a breakpoint.
Code No.	A code number for the trigger that contains a breakpoint. C/SIDE uses this number to identify the trigger at run-time.
Trigger Name	The name of the trigger where there is a breakpoint.
Object Name	The name of the object.
Enabled	A check mark indicates whether or not the breakpoint is enabled.

You must create a tabular form based on the **Breakpoints** virtual table to manage breakpoints.





When a breakpoint element is expanded, you can see four types of information for the breakpoint:

XML Tag	Description
TriggerName	The name of the trigger that contains the breakpoint.
CodeNo	The Code Number for a specific trigger in an object. C/SIDE uses this number to identify the trigger at run-time.
Trigger Line	The number of the line in the trigger where the breakpoint has been defined.
Enabled	A Boolean expression of whether or not the breakpoint is enabled.

### Starting Dynamics NAV or NAV Application Server Using Another Breakpoint File

You can start both Dynamics NAV and NAV Application Server with a `breakpoints` parameter. This enables you to specify a particular file for saving and loading breakpoints.

#### Example

```
FIN.EXE breakpoints=C:\MyBreakpoints.xml
```

### Storage of Debugging Information in the FIN.ZUP File

The selections that you make in the Tools, Debugger, Active and Tools, Debugger, Breakpoint on Triggers menu commands are stored in the `fin.zup` file. This means, for example, that if the debugger was active and set to break on triggers when you logged off, then these selections will apply when you log on again.

## Overview of Shortcut Keys

Here is a list of the shortcut keys for the most common debugging commands:

Shortcut Key	Command
SHIFT+CTRL+F11	Debugger Active
F5	Go
F9	Toggle Breakpoint
SHIFT+CTRL+F12	Breakpoint on Triggers
SHIFT+F9	Open Breakpoints Dialog Box
CTRL+SHIFT+F9	Remove All Breakpoints
F8	Step Into
CTRL+F8	Step Over
ALT+NUM*	Show Next Statement
SHIFT+F5	Stop Debugging

**The Debugger and the Command Buffer**

.....

C/SIDE uses a command buffer to improve performance. However, when you run the debugger, C/SIDE deactivates the command buffer. For more information, see "Performance" on page 559.

.....

## 18.3 The Code Coverage Tool

When you add the function (trigger) with ID 6 to Codeunit 1, you can access the code coverage functionality from the Debugger submenu of the Tools menu. You can now start and stop code logging. You can also view the code that is logged. Further, you can use the `CODECOVERAGELOG` function to start and stop the logging of code. This function can also retrieve the current logging status. See the online *C/SIDE Reference Guide* for information about the `CODECOVERAGELOG` function.

The code coverage functionality is useful when you are customizing Dynamics NAV and want to test your work. It provides a quick overview of the objects for which code has been executed, and it displays the code that has been logged.

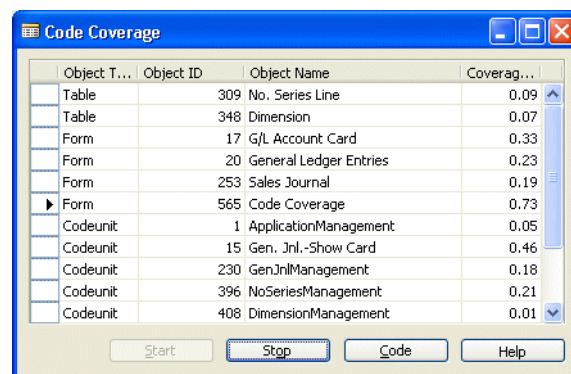
The **Code Coverage** window displays the objects (tables, forms, reports, dataports and/or codeunits) for which code has been executed and logged during one or more transactions. The **Code Overview** window displays the code that has been logged for a selected object. You can read about the **Code Coverage** and **Code Overview** windows in the following section.

### Using the Code Coverage Tool

As stated earlier the Code Coverage tool is useful for giving you an overview of the objects that are called when you perform any tasks and the code that is used during these transactions.

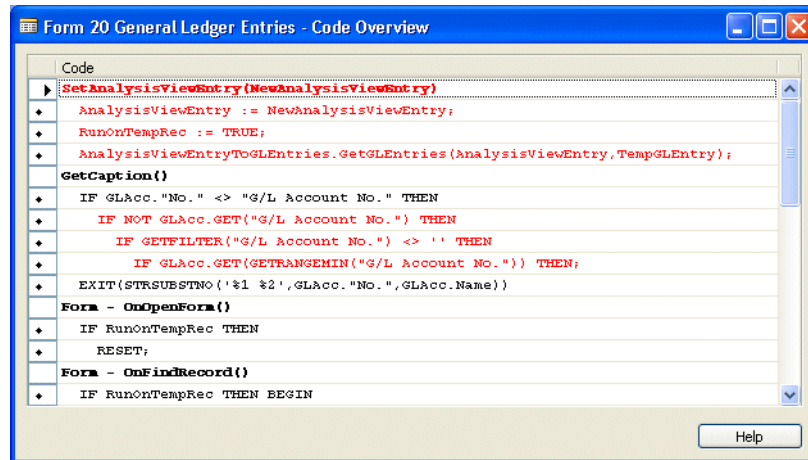
To log code:

- 1 Click Tools, Debugger, Code Coverage. The **Code Coverage** window opens.
- 2 Click Start. The Code Coverage tool is now ready to log code.
- 3 When you have completed the transactions that you want to monitor, return to the **Code Coverage** window. It now contains a list of any tables, forms, reports, dataports and codeunits that were used.



- 4 Click Stop.

- 5 Select an object whose code you wish to view. Click Code to open the **Code Overview** window:



The **Code Overview** window displays code for the object that you selected in the **Code Coverage** window. Lines of code that were executed during the transaction(s) are shown in black. Lines of code that were not executed are shown in red.

The **Code Overview** window displays code in a similar way to the debugger. However, while you see code being executed in the debugger, the **Code Overview** window shows you the end result: the code that has been executed. When a line of code is executable, a bullet symbol is shown on the left of the line. Only the information for lines that are marked with a bullet is correct. The lines of code that are not marked with a bullet are simply displayed in the color of the neighboring code lines.

**Important**

.....

You must not modify objects while using the Code Coverage tool because this will produce inconsistent results.

.....



## Chapter 19

### Extending C/AL

This chapter describes how you can extend C/AL by using COM technologies. C/SIDE supports automation servers by acting as an automation controller and using OCXs (custom controls).

- What Is COM?
- Using COM Technologies in C/SIDE
- Using C/SIDE as an Automation Controller
- Receiving Events in C/SIDE
- Using Custom Controls from C/SIDE
- Acquiring Controls

## 19.1 What Is COM?

This is not the place for anything but a very brief explanation of what the terms COM, OCX, Automation, OLE, ActiveX and so forth mean. The subject is a huge and complicated one that has been described in a number of good books.

### COM and C/SIDE

In C/SIDE, you can use COM technologies in two ways: you can use custom controls (OCXs), and you can use Automation (with C/SIDE in the role of an automation controller or client). There is a vast array of commercially available OCXs that perform all kinds of tasks, and you can develop your own. When you use C/SIDE as an automation controller, you will probably work with programs such as the Microsoft Office suite of products.

If you are going to develop custom controls yourself, you will probably use tools like Microsoft Visual C++ or Microsoft Visual Basic. Both products use wizards to make it very easy to develop COM objects. It is, in fact, entirely possible to develop functional controls without understanding any of the complex details of COM itself. If you are going to use existing COM objects (controls or automation servers) from C/SIDE, you certainly do not need to know anything about COM. Using the functionality provided by a COM object is no different than using any C/AL function.

If, however, you do want to know more, here is a list of recommended books:

- David Chappell. *Understanding ActiveX and OLE*. Microsoft Press (1996).

This book gives a broad overview of the subject without going into too much detail.

- Dale Rogerson. *Inside COM*. Microsoft Press (1997).

This book provides a more technical description.

- Kraig Brockschmidt. *Inside OLE*, 2nd edition. Microsoft Press (1995).

For those who really want to know the details, this book is very extensive (but it is also older than the other two books mentioned here).

The very rapid evolution in this area has turned the concepts and the terminology that is used to describe them into what David Chappell calls "moving targets," which means that it is no easy task to keep printed documentation updated. The Microsoft Web site (<http://www.microsoft.com>) offers a wealth of regularly updated online information, including the latest specifications of all aspects of COM.

### Terminology and History

Parallel with the rapid development of the technology, the terminology used to describe the technology has changed fast. The following table shows how terms have been added and meanings have changed as the technology has evolved:

Term	Description
OLE version 1.0	OLE is introduced as Object Linking and Embedding, allowing users to create compound documents (for example, a Microsoft Excel spreadsheet could be embedded in a Microsoft Word document.)



Term	Description
COM	OLE is generalized into COM: the Component Object Model. COM is seen as an architecture for interaction between software components.
OLE version 2.0	Building on the COM paradigm, OLE version 2.0 refines the linking and embedding concepts of OLE version 1.0, and adds new concepts such as OLE Automation. OLE version 2.0 is a suite of (more or less) related technologies that use COM rather than "just" linking and embedding.
OLE	In accordance with the broadening of the concept, OLE is no longer considered an acronym but a name in its own right (pronounced <i>o-lay</i> ).
OLE Automation	OLE Automation is the name for the ability of one program to expose any or all of its capability for another program to use. In other words: programmability. The preferred term is now <i>Automation</i> , with the program providing functionality being called the <i>Automation server</i> and the program that uses this functionality the <i>Automation controller (or client)</i> .
OLE Controls	Influenced by VBX, Visual Basic Extensions, OLE Controls are defined as COM objects that meet a certain, well-defined set of specifications. An OLE Control (also called a Custom Control) is a COM object that can be "plugged in" and used by a control container. In this way, applications can be built from reusable (binary) software components. OLE Controls usually have .ocx as their file name extension.
ActiveX	The first specifications for OLE Controls were rather strict and demanded, among other things, that a control should implement a vast number of interfaces. With the advent of the Internet and the emergence of the Internet Explorer as a favored control container, the specifications were relaxed in order to make it possible to create controls that have a smaller footprint and therefore will load faster. At the same time, OLE Controls were renamed ActiveX controls.
DCOM	The specifications for DCOM (Distributed COM) were released in 1996. DCOM expands COM to make communication over a network transparent to the clients and servers that are involved.
COM+	COM+ is the backward-compatible successor to COM. It enhances COM with a rich set of new features.

## 19.2 Using COM Technologies in C/SIDE

C/SIDE supports COM technologies in two ways: using custom controls (OCXs) and as an automation controller. This support has a few limitations:

**Only non-visual controls are supported.** This means that a control cannot be used to add graphical elements to a C/SIDE object (you cannot, for example, add a third-party control to a form). The control can, however, display information and interact with the user in a window of its own.

**Exception handling.** C/SIDE does not allow the retrieval of information about exceptions from a control or automation server through the Invoke method of the IDispatch interface and the EXCEPINFO structure (as described, for example, in *Inside OLE*). The samples in the C/OCX Samples – the control and the C/SIDE application that uses it – show a way to work around this limitation. You can find a description on page 400.

### Parameters, Return Values and Data Types

As you can see in the literature about COM, the mechanisms for calling methods in a control, passing parameters and receiving return values are somewhat complicated. Using tools like the wizards in Microsoft Visual C++ shields you from most of the complexities.

You should know, however, that there is not a one-to-one relationship between the data types that you can use when implementing methods in, for example, Visual C++ and the data types in C/AL. Some of the COM data types are not supported in C/AL and some have a limitation imposed on their usage.

When you use the C/AL Symbol Menu, you can see the syntax for a method or property with the return value and the parameters shown with the COM data types.

The following table shows how you map C/AL data types to COM data types:

<b>C/AL Data Type</b>	<b>COM Data Type</b>	<b>Comment</b>
Boolean	VARIANT_BOOL (VT_BOOL)	
Option	long (VT_I4)	
Integer	long (VT_I4)	
BigInteger	long (VT_I4)	
Decimal	CURRENCY (VT_CY)	The CURRENCY type in COM is a special data type with a fixed point that has 15 digits to the left of the point and 4 to the right. You should be aware that the Decimal type in C/AL does not have a fixed point and can have a total of 18 digits. This could possibly lead to some rounding being performed when a type Decimal number is passed to a method that expects a CURRENCY. The server manipulates that number and returns it as a CURRENCY. No matter how many digits the original Decimal had to the right of the decimal point, the returned CURRENCY will have no more than 4 digits.
Char	BSTR (VT_BSTR)	
Text	BSTR (VT_BSTR)	
Code	BSTR (VT_BSTR)	
Date	DATE (VT_DATE)	
Time	void (VT_VOID)	
DateTime	DATE (VT_DATE)	
Automation	TypedObject, UntypedObject (VT_DISPATCH)	
InStream	VT_STREAM	
OutStream	VT_STREAM	
Variant	VARIANT (VT_VARIANT)	

The following table shows how you map COM data types to C/AL data types:

<b>COM Data Type</b>	<b>C/AL Data Type</b>	<b>Comment</b>
VT_UNKNOWN	InStream or OutStream	Only the IID_IStream and IID_SequentialStream interfaces are supported. If you pass any other IUnknown interface, an error will occur at runtime.
short (VT_I2)	Integer	
long (VT_I4)	Integer BigInteger	

COM Data Type	C/AL Data Type	Comment
float (VT_R4)	Decimal	
double (VT_R8)	Decimal	
CURRENCY (VT_CY)	Decimal	The CURRENCY type in COM is a special data type with a fixed point, which has 15 digits to the left of the point and 4 to the right. You must note that the Decimal type does not have a fixed point and can have a total of 18 digits.
DATE (VT_DATE)	Date DateTime	The COM DATE type contains both a date and a time value. C/AL has Date and Time as separate data types. Therefore, the time part of a COM DATE type will be lost when the COM DATE type is mapped to the Date C/AL data type. To keep the date and the time, map the COM Date type to the C/AL DateTime data type.
BSTR (VT_BSTR)	Text	
VARIANT_BOOL (VT_BOOL)	Boolean	
TypedObject/ UntypedObject (VT_DISPATCH)	Automation/OCX	
VT_EMPTY	Text	
VARIANT (VT_VARIANT)	Variant	

### Unsigned char (VT\_UI1), SCODE (VT\_ERROR) and SAFEARRAY (VT\_ARRAY)

You can use the C/AL variant data type to pass unsigned char, SCODE or SAFEARRAY to another variant that supports these types. You cannot assign them to C/AL data types.

#### Further remarks

When you call a method with a ByRef parameter, the normal C/AL type conversions do not take place. This means, for example, that if the parameter is of type float, you have to use a C/AL variable of type Decimal. You cannot use Integer and have C/AL convert it for you. (Hint: if the value you want to pass has a "wrong" type, when, for example, it is a value from a database record field, you can assign it to a C/AL variable of the correct type before calling the COM object method.)

You will sometimes see a COM object method or a property in the C/AL Symbol Menu that has type IDispatch. This means that the method or property returns or expects a COM object. In this case, you must use a C/AL Automation variable that has been declared (through the Subtype) to be the correct COM object. You will have to study the documentation for the automation server to gain the necessary information.

You will also see properties and methods that do not have one of the "normal" types. For example, a method in Microsoft Excel can have a return value of type WORKBOOK.

This means that the automation server has implemented a USERDEF type. C/SIDE supports USERDEF types in two contexts: IDispatch and Enumeration.

If the USERDEF type is an IDispatch, it means that it is an interface (sometimes also called class or object) with a specific GUID. You will have to use the same object for a return value or parameter. You do this by creating an Automation variable with the correct Subtype.

For example, Microsoft Excel has a number of methods that return a WORKBOOK variable. This means that you must declare a variable of type Automation and subtype 'Microsoft Excel 8.0 Object Library'.Workbook.

If the USERDEF type is an Enumeration, you should know that you cannot use the symbolic name (for example, xl3DPie) but instead must use the enumerator (for example, -4102). For Microsoft Office products, you can find this value by using the VBA Object Browser (see page 382).

## 19.3 Using C/SIDE as an Automation Controller

The following description outlines the procedures for using an automation server from C/SIDE. As you will see, there are very few steps required that are specific to C/SIDE (C/AL). Using an automation server consists of five steps:

- 1 Declare the creatable (top-level) interface (class) of the automation server as a variable of data type Automation.
- 2 Declare all the other interfaces (classes) as variables of data type Automation.
- 3 Use the C/AL function CREATE on the variable declared in step 1. *Do not* use CREATE on any other variables.
- 4 Use the methods and properties of the automation server in your C/AL code.
- 5 You can CLEAR (destroy) the top-level object if you want. Otherwise, it will be destroyed automatically when the variable goes out of scope.

You write most of your code during step 4 using the methods and the properties of the automation server. The syntax and the semantics of these methods and properties are documented in the documentation for each automation server. Using these methods and properties in C/AL does not involve any new or changed syntax.

The best way to learn how to use automation is to look at actual solutions. The following two sections show you how to use Microsoft Word and Microsoft Excel, respectively.

### Writing a Letter In Microsoft Word

In this example, you will:

*Implement functionality that writes a letter in Microsoft Word when you click a menu item on the Customer card. The letter will only be created if the customer has bought goods for more than LCY 2,500 during the past year. If the customer fulfills this requirement, the letter offers a 3% discount.*

Most of the information you need to transfer to Word is in the **Customer** table. This includes the information about the customer that you will use in the letterhead, such as the name and the address of the customer and the name of the contact to whom you will address the letter.

The **Customer** table contains a FlowField called **Sales (LCY)** that contains the financial information that you need, namely the aggregated sales for the customer. For the sake of this example (where the emphasis is on using automation), you will simply use this value as it is. Please note that this is not what you would do in "real life." You would have to set up an appropriate date filter to get the sales for the past year only.

You also need to retrieve the information about your own company that you will use in the letterhead and in the greeting of the letter. This information is contained in the **Company Information** and the **User** tables.

### Where to Place Automation Code

You will put all the code in a separate codeunit that is called from a menu item on the Customer card.

There are two major concerns that you must consider when you are deciding where to place the code that uses automation:

- The first is the fact that the automation server must be installed on the computer that compiles an object that uses automation. If you must recompile and modify an object on a computer that does not have the automation server installed on it, you will have to modify the code drastically in order to compile it again. It is therefore recommended that you isolate code that uses automation in separate codeunits.
- The second concern is performance. There is some overhead involved in creating an automation server (using the `CREATE` system call). If the automation server is to be used repeatedly, you will gain better performance by designing your code so that the server is created only once (as opposed to making a series of `CREATE/CLEAR` calls).

That said, it is obvious that these two concerns will sometimes clash and you will have to make some trade-offs, based on the actual context in which your code will be used.

In this example, you have chosen not to put the automation code on the customer card, but to isolate it in a separate codeunit. Performance could be improved by putting the code on the Customer card as this would mean that you don't have to open and close Word for each letter that is created in the session.

However, there is a simple workaround for this problem: if Word is already open when it is called from the code, the running instance is reused. This means that the user can simply open Word "manually" or just not close it after creating the first letter.

### Background Information about Using Word in this Example

You want to extract and transfer data about one customer at a time to Word. You also want to initiate this and the subsequent processing in Word from the customer card.

This approach to mail merge is different from the mail merge you can achieve with C/ODBC, which is better suited for bulk processing (creating a large number of letters).

However, this approach forces you to use Word in a slightly unorthodox way. You need some placeholders so that you can put the information in predefined places in a template. Unfortunately, there is no straightforward way to do this without using the regular mail merging facilities in Word.

You will therefore insert a number of fields into the Word template and give these fields convenient mnemonic names that correspond to the names of the C/SIDE record fields you are going to use.

To make this work, your C/AL code must make two extra calls to Word. You must call `ActiveDocument.Fields.Update` before starting to use the fields. After you have transferred all the information, you must call `ActiveDocument.Fields.Unlink`. This ensures that you can successfully use the Word fields as placeholders.

One more thing, while you can give the fields names like Customer or Address, you will have to reference them by indexing into the Fields collection of the document. This can make the C/AL code harder to understand.

### Creating the Template in Word

The first thing that you must do is prepare the template that you will use to create the letters to the customers that qualify for a discount.

To create the template:

- 1 Open Word and open a new document.
- 2 Click Insert, Field and add a field from the Mail Merge category called MergeField.
- 3 Enter *Contact* as the field name.


The simple template that you are going to use in this example should contain the following fields:

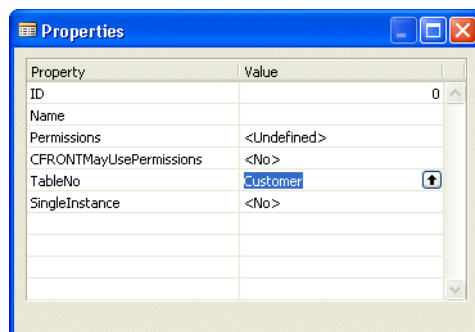
Field Name	Contains	Underlying Table
Name	The name of the customer.	Customer
Address	The address of the customer.	Customer
Sales (LCY)	The total amount that the customer has purchased from you.	Customer
Contact	The name of your contact person at the customer.	Customer
Company Name	The name of your company.	Company Information
User Name	The name of the person generating this letter.	User

- 4 Create these six fields and save the Word document as `Discount.dot`.

### Creating the Codeunit and Declaring the Variables

The next step is to create the codeunit that calls Word and creates the letter.

- 1 Open the Object Designer (SHIFT+F12), and click Codeunit, New to create a new codeunit.
- 2 Click View, Properties (SHIFT+ F4) to open the **Properties** window of the codeunit.
- 3 In the **TableNo** field, click the AssistButton  to open the **Table List** window.
- 4 In the **Table List** window, select the **Customer** table and click OK:



By setting the *TableNo* property to the **Customer** table, you get a very neat connection between the Customer card and this codeunit. Later, you will add the menu item that calls the codeunit to the Customer card. The codeunit will be called with the record that is currently selected in the Customer card as its current record. You do not have to do anything special to coordinate the two.

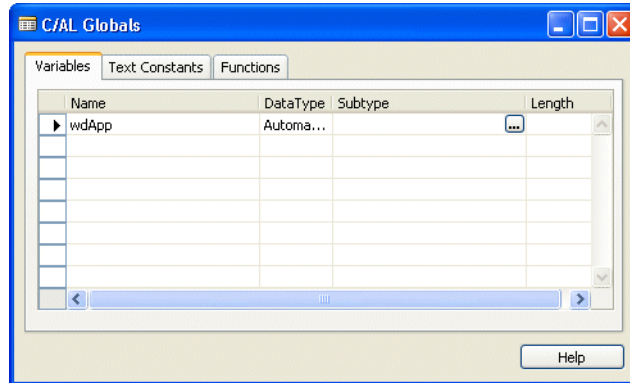


Declaring the variables

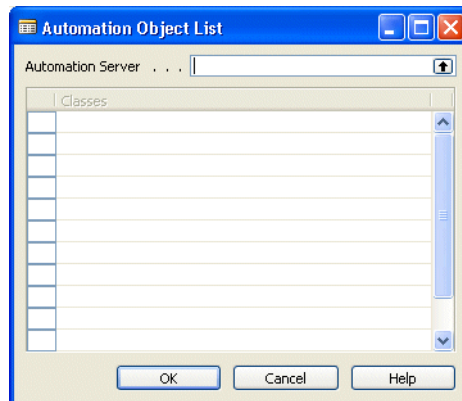
Now you must declare the variables that you need.

- 1 Click View, C/AL Globals.
- 2 First, you declare the top-level (creatable) class of Word. The name of this class is Application.

Enter *wdApp* as the name of a new variable, and select Automation as the data type.

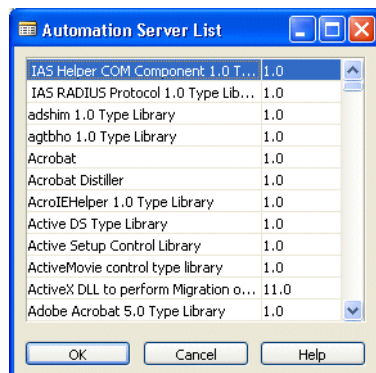


- 3 In the **Subtype** field, click the AssistButton ... and the **Automation Object List** window appears:



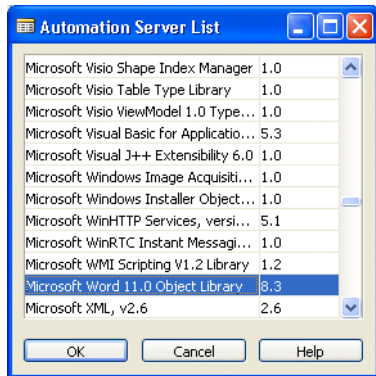
This window is used to select the class of the automation server that this variable refers to. However, you must first select an automation server.

- 4 In the **Automation Server** field, click the AssistButton ↑ and the **Automation Server List** window appears:

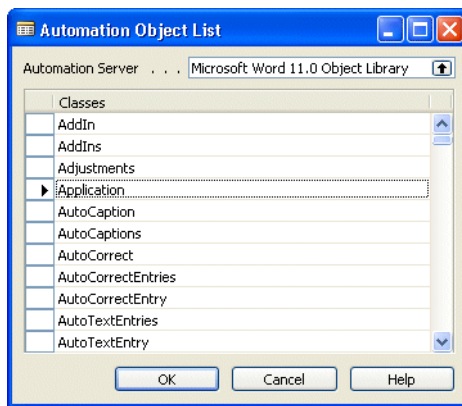


This is a list of the automation servers that are installed on the computer.

Scroll down to Microsoft Word, and select it:

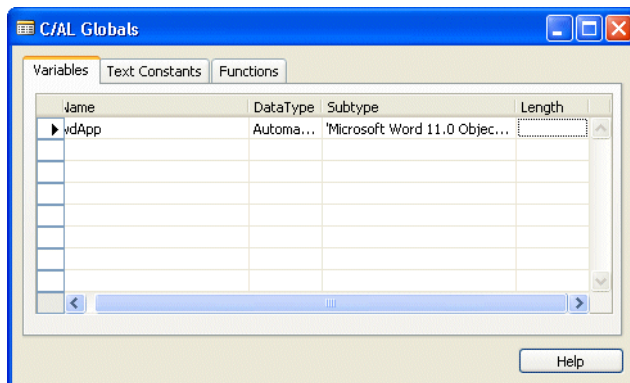


- 5 Click OK and the **Automation Server List** window closes. You can now see that the **Automation Object List** window has been filled in with a list of all the classes in the Microsoft Word 11.0 Object Library:



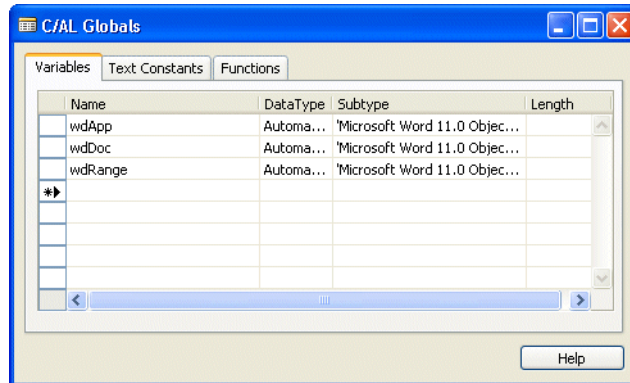
- 6 Select Application, and click OK.

You have now defined the creatable (top-level) class of Word as a variable:



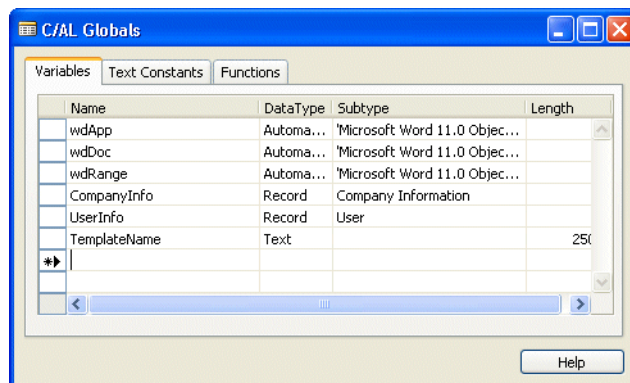
- 7 You need two more classes from Word for this example: Document and Range. Go ahead and declare the variables in the same way as you did for Application.

The **C/AL Globals** window should look like this when you are finished:



- 8 You also need to define a few more variables: two record variables that point to the **Company Information** and **User** tables respectively, and a text variable to hold the name of the template for the Word letter you are writing. Setting up these variables is straightforward.

The **C/AL Globals** window should look like this when you are finished:



Remember to change the length of the TemplateName text variable from 30 to 250.

## Writing the C/AL Code

Before you start writing the part of the C/AL code that uses automation, you must do some initial processing.

### Initial processing

You start by calculating the **Sales (LCY)** FlowField. Then you check whether or not the customer qualifies for a discount. Finally, you retrieve the information from the **Company Info** and **User** tables that you use to fill in some of the fields in the letter.

```
CALCFIELDS("Sales (LCY)");
IF ("Sales (LCY)" < 2500) THEN
    EXIT;
```

```
CompanyInfo.FIND;
UserInfo.GET(USERID);
```

Creating the automation server

You must create an instance of Word before you can use it. The C/AL function `CREATE` does exactly this. You call `CREATE` like this:

```
CREATE (wdApp) ;
```

Note that `CREATE` has an optional argument, `NewServer`, which by default is `FALSE`. This means that if an instance of the automation server is already running, it will be reused. If you had set `NewServer` to `TRUE`, as in `CREATE (wdApp, TRUE)`, you would have requested a new instance of Word. Note that ultimately the automation server itself can control whether or not it can be reused or not (see the documentation for the server in question if this aspect is important for your application.)

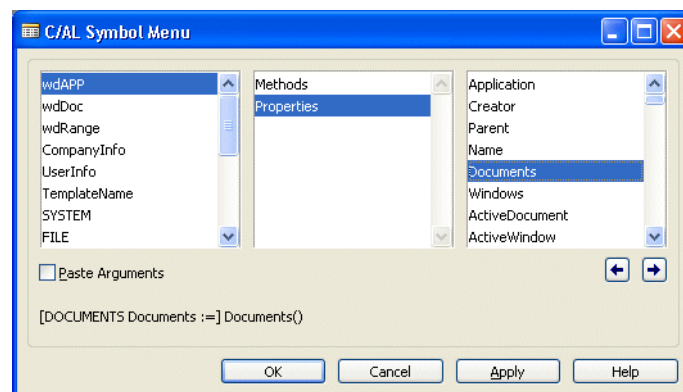
Adding a new document

Now you will add a new document to Word that uses the template you designed earlier:

```
TemplateName := 'C:\Documents and Settings\All Users\Discount.dot';
wdDoc := wdApp.Documents.Add(TemplateName);
wdApp.ActiveDocument.Fields.Update;
```

Because the `Add` method of the `Documents` collection requires that you pass the path to the template by reference, you must set up the `TemplateName` variable to hold this information. you will get a compilation error if you try to put the path into the call as a literal string.

Take a look at the syntax string for the `Documents` property of `wdApp` (the Microsoft Word Application class):



If you open the online Help of Microsoft Word Visual Basic for the `Documents` property, you can see that the `Documents` property returns a `Documents` collection that represents all the open documents. You can also see that the `Documents` collection object has an `Add` method, and that the `Add` method has this syntax:

```
expression.Add(Template, NewTemplate, Document Type, Visible)
```

where *expression* is a required argument, and it has to be an expression that returns a `Documents` object. All of the arguments are optional. You will use `Template` to open a new document based on your template.

Now look at the syntax in the C/AL Symbol Menu again. Note that the `Documents` property returns an object of type `DOCUMENTS`, a user defined type. This means that the property returns a `Documents` class (or `IDispatch` interface). This information helps the compiler perform a better type check during compilation.

It also means that the statement:

```
wdDoc := wdApp.Documents.Add(TemplateName);
```

succeeds and can pass both the compile-time and the runtime type checks.

Finally, the Add method returns a Document class. While you did not have to declare a C/AL variable for the "interim" Documents class, you have declared a variable for this return value, wdDoc.

The third line (`wdApp.ActiveDocument.Fields.Update;`) contains a call that must be made to ensure that the template works as intended.

Transferring data to  
Word

Now you are ready to transfer the actual data from the Customer record to the placeholder fields in the Word document.

You have set up the first three fields in the template so that they can contain the contact, name and address of the customer and you can therefore transfer the data with the following code:

```
wdRange := wdAPP.ActiveDocument.Fields.Item(1).Result;
wdRange.Text := Contact;
wdRange.Bold := 1;

wdRange := wdAPP.ActiveDocument.Fields.Item(2).Result;
wdRange.Text := Name;
wdRange.Bold := 1;

wdRange := wdAPP.ActiveDocument.Fields.Item(3).Result;
wdRange.Text := Address;
wdRange.Bold := 1;
```

Again, you are tweaking Word here. You cannot use the fields directly as variables (and do an assignment such as `...Fields.Item(3) := Address`). Instead, you use the *Result* property of the field. This property returns the result of the field as a range. You place this range in the third automation variable that you declared, wdRange.

Then you can set the *Text* property of the range to the desired values – the name of your contact person and the name and address of the customer. Finally, you turn on the bold formatting that the text would otherwise be normal.

### Using Default Members

.....

The documentation for Microsoft Word Visual Basic uses the following syntax:

```
wdApp.ActiveDocument.Fields(3).Result
```

As you can see you use a slightly different syntax in the examples:

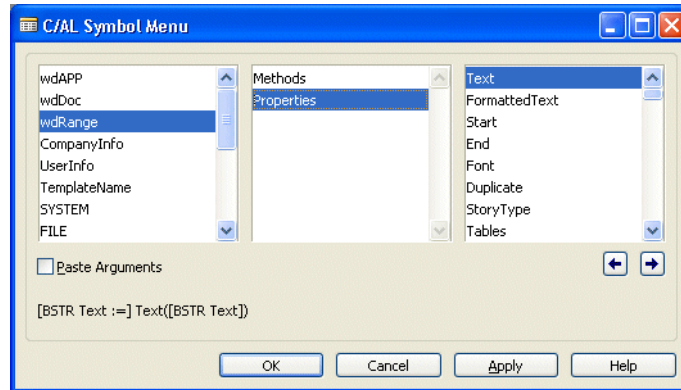
```
wdApp.ActiveDocument.Fields.Item(3).Result
```

This is because the Item method is the default member for the Fields collection. Visual Basic uses this method if the programmer does not specify an alternative method.

However, in C/SIDE you must explicitly specify the Item method.

.....

You must remember that the data you are transferring must be in text format. If it is not in text format, you get a compilation error. As you can see in the following picture, `wdRange.Text` expects its arguments to be of type `BSTR`, which maps to either `Text` or `Code` in C/SIDE:



This means that any data that is not `Text` or `Code` must be converted before it is passed to Word.

In this example you need to transfer the **Sales (LCY)** field, to Word. This field is a Decimal field, so you have to use the `FORMAT` function to convert it to `Text`. The `FORMAT` function has the following syntax:

```
String := FORMAT(Value [, Length] [, FormatNumber | FormatString])
```

You can transfer and format the data from the **Sales (LCY)** field with the following code:

```
wdRange := wdAPP.ActiveDocument.Fields.Item(4).Result;
wdRange.Text := FORMAT("Sales (LCY)");
wdRange.Bold := 0;
```

Now, all that remains is to transfer the data from the other tables. The following two statements transfer the information you retrieve from the **Company Information** and **User** tables:

```
wdRange := wdApp.ActiveDocument.Fields.Item(5).Result;
wdRange.Text := CompanyInfo.Name;

wdRange := wdApp.ActiveDocument.Fields.Item(6).Result;
wdRange.Text := UserInfo.Name;
```

Finishing the code

After you have transferred the data to Word, you must enter two more statements to finish the processing:

```
wdApp.Visible := TRUE;
wdApp.ActiveDocument.Fields.Unlink;
```

The first statement opens Word and shows you the letter that was created. The second statement completes the tweaking of Word and makes the fields work as placeholders.

Save and compile

Finally, save and compile the codeunit and give it a number and a name. In this example, you have used the name `Discount Letter`.

### To-do list

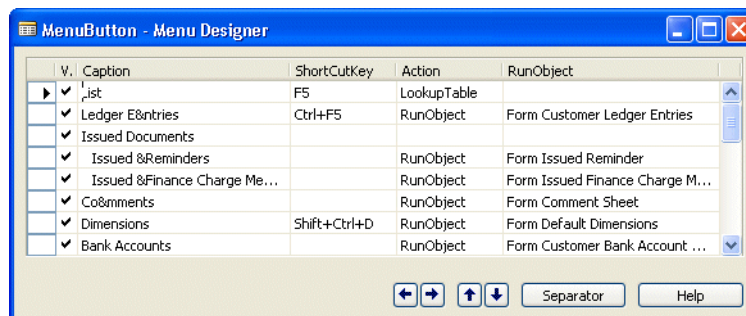
Although this code will work, you must add a few things to make it a ready for the real world:

- It is not a good idea to use a hard-coded template name. It should be kept in a table, and the user should select it from a form. You will probably have different templates for the different kinds of letters that you want to send to your customers.
- You should add some error-handling code. For example, the `CREATE` call fails if the user does not have Word installed or if the installation has been corrupted. You should check the return value of `CREATE` and give an appropriate message if it fails.
- The user should get a message if the customer does not qualify for the discount. In the example, the codeunit closes without further ado.


### Calling the Codeunit from the Customer Card

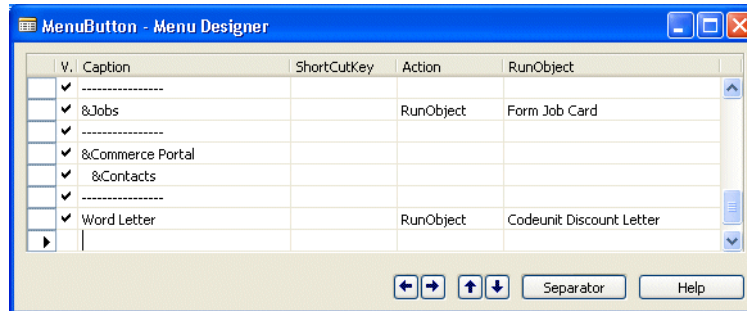
The final task is to ensure that you can call the codeunit from the Customer card. You will add a menu item to the Customer menu button.

- 1 Open the Object Designer (SHIFT+F12) and click Form.
- 2 Select the **Customer Card** form (Form 21) and click Design.
- 3 Right-click the Customer menu button and click Menu Items in the menu. Alternatively, you can click View, Menu Items to open the Menu Designer:



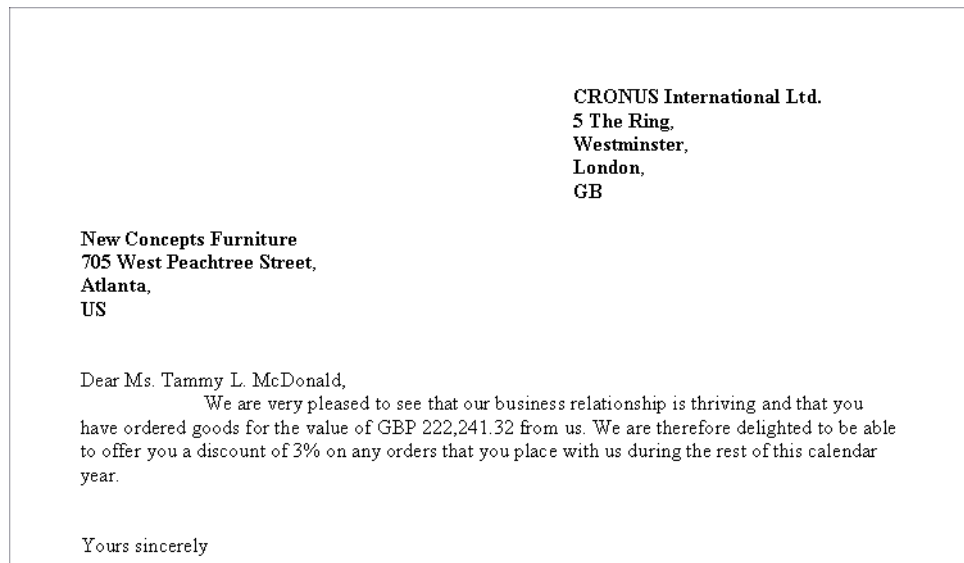
- 4 Scroll down to the bottom of the list of menu items.
- 5 Click Separator to insert a separator line.
- 6 Fill in the **Caption** field with the text you want to appear in the menu (here, you have used *Word Letter*).
- 7 In the **Action** field, click the AssistButton ▾ and select RunObject.

- In the **RunObject** field, click the AssistButton  and select the codeunit you have created:



- Save and compile the **Customer Card**.

However, the letter that you have just created only contains six fields and no body text. Before you can use this letter in a 'real' situation you will probably need to add some more fields, such as the name and address of your own company, the date and the currency code as well as the main text of the letter. It will also need some formatting to make it look more attractive.



However, if you alter the order in which the fields appear in the template, you must remember to change the numbering of the fields in the codeunit to ensure that the correct data is inserted into the appropriate fields.



## Graphing With Microsoft Excel

In this example, you will transfer data from the **G/L Entry** table to Microsoft Excel and create a graph. The main aim of this example is to show how to handle enumerations.

### Background Information about This Example

The aim of this example is to create a graph in Microsoft Excel that shows the distribution of personnel expenses by departments. In the Chart of Accounts, you can see that **Total Personnel Expenses** is the total of accounts 8700 to 8790. In the **Dimension Value** table, you can see that there are three departments: *ADM*, *PROD* and *SALES*.

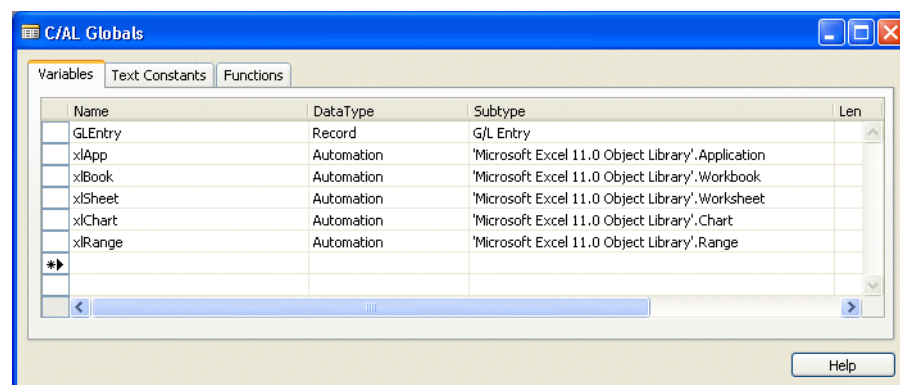
You will create a codeunit that retrieves this data from the **G/L Entry** table, transfers it to Excel and creates a graph. You will run the codeunit directly from the Object Designer, but in a real application you would call it from an appropriate place, for example, from a menu in the **Chart of Accounts** window.

### Creating the Codeunit: Declaring Variables

The first step is to define the necessary variables:

- 1 Open the Object Designer, and click Codeunit, New to create a new codeunit.
- 2 Click View, C/AL Globals.
- 3 Define a Record variable that has the **G/L Entry** table as Subtype. Here, you have called it *GLEntry*.
- 4 Define an Automation variable with Microsoft Excel 11.0 Object Library as Automation Server and Application as Automation Object. Call it *xlApp*.
- 5 Define an Automation variable with Microsoft Excel 11.0 Object Library as Automation Server and Workbook as Automation Object. Call it *xlBook*.
- 6 Define an Automation variable with Microsoft Excel 11.0 Object Library as Automation Server and Worksheet as Automation Object. Call it *xlSheet*.
- 7 Define an Automation variable with Microsoft Excel 11.0 Object Library as Automation Server and Charts as Automation Object. Call it *xlChart*.
- 8 Define an Automation variable with Microsoft Excel 11.0 Object Library as Automation Server and Range as Automation Object. Call it *xlRange*.

After these steps, the **C/AL Globals** window should look like this:



## Creating the Codeunit: Initial Steps

The code itself is quite simple. First, you set the key you need for the **G/L Entry** table and then use `SETFILTER` to select the accounts you are interested in:

```
GLEntry.SETCURRENTKEY("G/L Account No.", "Business Unit Code",
"Global Dimension 1 Code", "Global Dimension 2 Code",
"Close Income Statement Dim. ID", "Posting Date");

GLEntry.SETFILTER("G/L Account No.", '8700..8790');
```

Then, you create an instance of Excel:

```
CREATE(xlApp);
```

Next, you add a new workbook to Excel:

```
xlBook := xlApp.Workbooks.Add(-4167);
xlSheet:= xlApp.ActiveSheet;
xlSheet.Name := 'Personnel Expenses';
```

In the first line, you use the `Add` method of the `Workbooks` collection to return a new workbook. Then you use the `ActiveSheet` property of the `Application` class to make sure that what you do next affects the active sheet of the new workbook. In the third line you give the sheet a name.

Now you are probably wondering what the argument, `-4167`, in `Add` is. If you look in the Microsoft Excel Visual Basic online Help, you can see that the `Add` method as it applies to the `Workbooks` object has one argument, `Template`. It is of type `VARIANT`. The description says:

*If this argument is a constant, the new workbook contains a single sheet of the specified type. Can be one of the following: `XIWBATemplate constants: xlWBATChart, xlWBATExcel4IntlMacroSheet, xlWBATExcel4MacroSheet, or xlWBATWorkSheet.`*

You want to create a workbook with a single sheet. Judging from the description, you should give an `XIWBATemplate` constant with the value `xlWBATWorkSheet` as the `Template` argument.

Nevertheless, you are passing the number `-4167`. The following paragraphs explain why.

### Enumerations

As described on page 368, this particular `VARIANT` is an enumeration.

There are two types of enumerations: those that are `USERDEF` types, and those that are not. This is not a `USERDEF` type, so it looks like a `VARIANT` in the C/AL Symbol Menu. You have to look in the Microsoft Excel Visual Basic Help to figure out that it is actually an enumeration. The hint is that the arguments can be constants with names like `xl*` (in Microsoft Word, they would be `wd*`, and in Microsoft Outlook `ol*`).

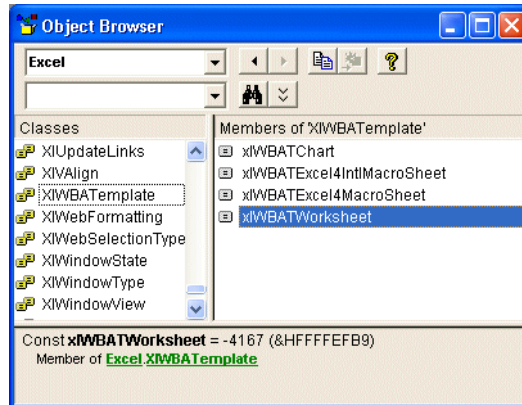
In C/SIDE, you cannot use the symbolic name (`xlWBATWorkSheet`). You have to use the enumerator (`-4167`).

To find this value:

### Finding an enumerator value

- 1 Open Excel.
- 2 Click Tools, Macro, Visual Basic Editor.

- 3 Click View, Object Browser.
- 4 Select *Excel* in the list box in the upper left-hand corner of the form.
- 5 Scroll down in the Classes list (to the left) until you can see XIWBATemplate, and select it.
- 6 In the Members of 'XIWBATemplate' list to the right, select xIWBATWorkSheet.
- 7 The value can now be seen in the information pane at the bottom of the form:



Alternatively, you can try this method:

A shortcut to the enumerator

Create a macro in Excel, and call the MsgBox function:

```
Sub x()
    MsgBox (xlWBATWorksheet)
End Sub
```

When you run the macro, a box will pop up with the value of the enumerator:



### Creating the Codeunit: Transferring Data

To transfer the data, you need to do two things: calculate the data and transfer the results of the calculation. To calculate the data, do the following:

```
GLEntry.SETRANGE("Global Dimension 1 Code", 'ADM');
GLEntry.CALCSUMS(Amount);
```

You use SETRANGE to filter the entries in the **G/L Entry** table on the **Global Dimension 1 Code** field. The first department is ADM (Administration). Then, you use CALCSUMS(Amount) to get the sum for the ADM department.

Now you can transfer the data to Microsoft Excel:

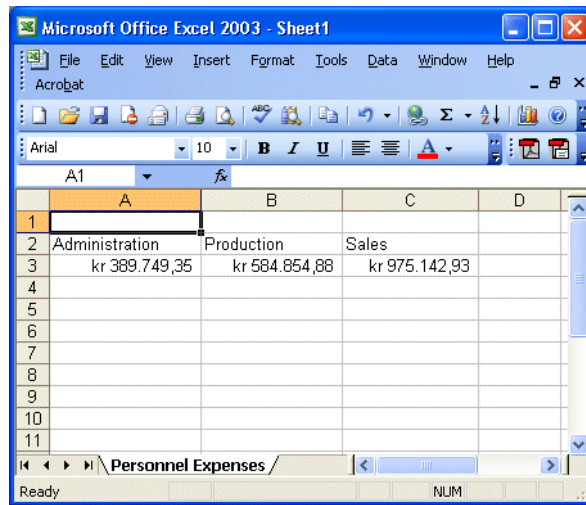
```
xlSheet.Range('A2').Value := 'Administration';
xlSheet.Range('A3').Value := GLEntry.Amount;
```

You repeat this for the other two departments, PROD and SALES:

```
GLEntry.SETRANGE("Global Dimension 1 Code", 'PROD');
GLEntry.CALCSUMS(Amount);
xlSheet.Range('B2').Value := 'Production';
xlSheet.Range('B3').Value := GLEntry.Amount;

GLEntry.SETRANGE("Global Dimension 1 Code", 'SALES');
GLEntry.CALCSUMS(Amount);
xlSheet.Range('C2').Value := 'Sales';
xlSheet.Range('C3').Value := GLEntry.Amount;
```

This is how the data looks once it is transferred to Microsoft Excel:



### Creating the Codeunit: Making the Graph

The final step is to create the graph. You will use the ChartWizard method to create a 3D pie chart. This is a fast and simple way to do it. You can more tightly control the design of the graph by setting it up using the methods and properties of the various Chart objects (ChartArea, Legend, and so on).

First, you must define a range for the data in the graph:

```
xlRange := xlSheet.Range('A2:C3');
```

Then, you add a new chart sheet and give it a name:

```
xlChart := xlBook.Charts.Add;
xlChart.Name := 'Personnel Expenses - Graph';
```

Finally, the following call creates the graph:

```
xlChart.ChartWizard(xlRange, -4102, 7, 1, 1, 0, 0, 'Personnel Expenses');
```

Use the first eight of the optional arguments of the ChartWizard method:

Argument	Description	Value
Source	The range that contains the source data for the new chart	xlRange – the object returned by xlSheet.Range('A2:C3').

Argument	Description	Value
Gallery	The chart type	-4102 – the enumerator for the xl3DPie XlChartType enumeration. See "Finding an enumerator value" on page 382.
Format	The option number for the built-in autoformats	7 – we found this one by trial and error, because the Excel documentation does not say much about it.
PlotBy	Specifies whether the data for each series is in rows or columns	1 – the enumerator for the xlRows XlRowCol enumerator.
CategoryLabels	An integer specifying the number of rows or columns within the source range that contain category labels.	1 – you have one row with category labels (the department names).
SeriesLabels	An integer specifying the number of rows or columns within the source range that contain series labels	0 – you do not have series labels in your data.
HasLegend	TRUE to include a legend	0 – this value works.
Title	VARIANT with the title of the chart	You pass a string, 'Personnel Expenses'. This works and the runtime conversion succeeds.

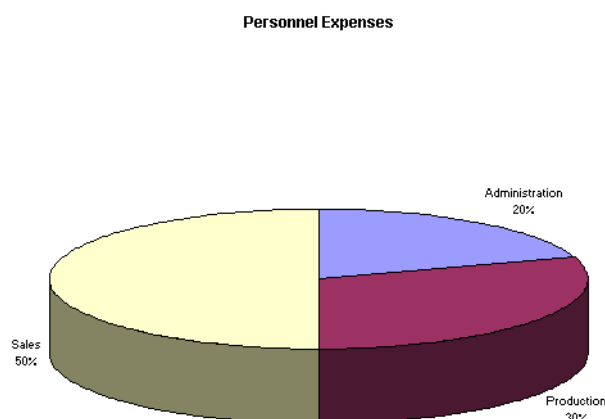
And, finally, you make Excel visible:

```
xlApp.Visible := TRUE;
```

Excel produces a General Protection Fault error when you close a new Excel worksheet created while Excel is invisible. To solve the problem, you can make Excel visible immediately after you create a new worksheet. Alternatively, you can make Excel visible just before you create a new Excel worksheet and then make it invisible again immediately after creating the new Excel worksheet. In this case you would write:

```
xlApp.Visible := TRUE;
xlBook := xlApp.Workbooks.Open(FileName);
xlApp.Visible := FALSE;
```

The graph looks like this:



## 19.4 Receiving Events in C/SIDE

C/SIDE can receive events from the components (automation servers and OCXs) that it controls. When you declare a global variable of data type Automation, you can specify whether you want to receive events. You do so by setting the *WithEvents* property for the variable to *Yes*. This automatically generates AL triggers for the events that the component provides. A trigger name consists of the name of the automation variable followed by "::." For example, if you declare an automation variable with the name *MyEventVar*, and the component provides the event *MessageReceived(...)*, the name of the trigger is *MyEventVar::MessageReceived(...)*. For information about the limitations on event triggers, see page 390.

In the following example, you enable C/SIDE to receive events from the Dynamics NAV Communication Component.

For more information about the Dynamics NAV Communication Component and the way in which it handles the exchange of data streams between NAV Application Server and a bus adapter, see the online Help *Development Guide for Communication Components* (*devguide.chm*). This is normally stored in `C:\Program Files\Common Files\Dynamics NAV\Communication Component`.

The Dynamics NAV Communication Component and the bus adapter are installed as part of the NAV Application Server. These two components must be installed before you can implement the following example.

### Receiving Notification of Inbound Documents

Before reading on, it will be helpful for you to read the example, "Writing a Letter In Microsoft Word" on page 370.

In the following example, you create two codeunits: one that enables C/SIDE to receive an event – notification of an inbound stream, and one that enables the client to send the stream. Needless to say, in a real life situation the same client would not both send and receive the stream.

To send and receive streams, C/SIDE depends on the existence of two external components, the Dynamics NAV Communication Component and a socket bus adapter.

### Receiving the Event

In this example you begin by creating the codeunit that allows the client to receive the event.

Declaring the variables for the external components

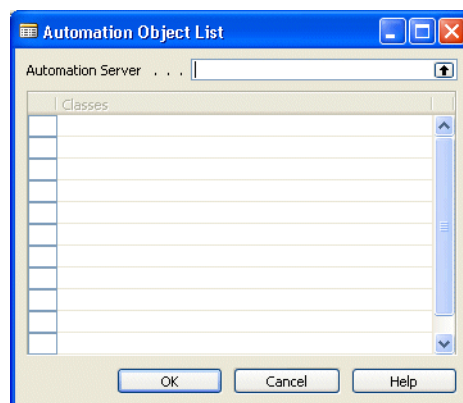
The first step is to create the codeunit and declare some variables of the data type Automation for both components. For each variable, you must also select an automation server and the class of the automation server that the variable refers to.

Declare the following variables:

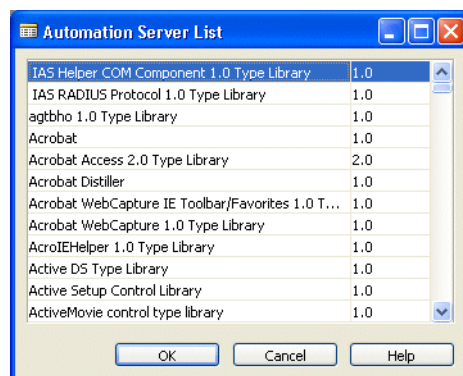
Variable Name	Data Type	Subtype
CC2	Automation	Dynamics NAV Communication Component version 2.CommunicationComponent
SBA	Automation	Dynamics NAV Socket Bus Adaptor.SocketBusAdapter
InMsg	Automation	Dynamics NAV Communication Component version 2.InMessage
InS	InStream	
txt	Text	

To declare these variables:

- 1 Click View, C/AL Globals to open the **C/AL Globals** window.
- 2 Enter CC2 as the name of the first variable and select Automation as the data type.
- 3 In the **Subtype** field, click the AssistButton ... to open the **Automation Object List** window.

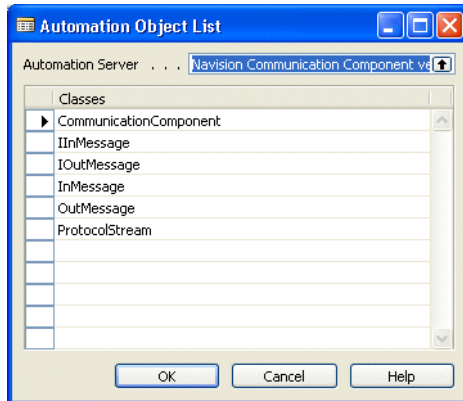


- 4 In the **Automation Server** field, click the AssistButton ↑ to open the **Automation Server List** window.

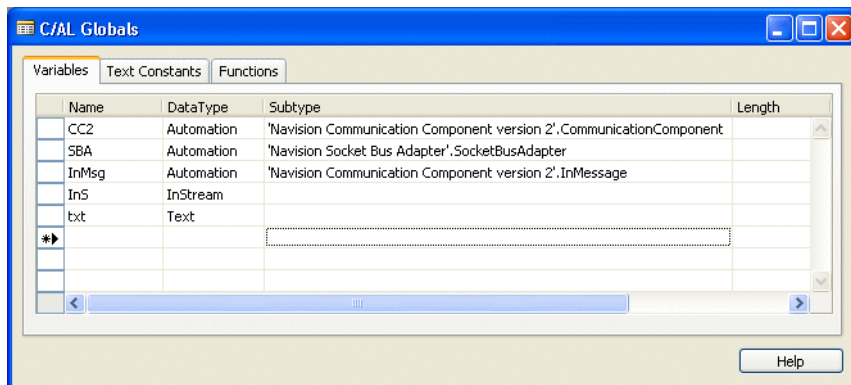


- 5 Select Dynamics NAV Communication Component version 2 and click OK.

- The classes of the Dynamics NAV Communication Component are now visible in the **Automation Object List** window.



- Select the class called CommunicationComponent.
- Declare all the other variables listed earlier and the **C/AL Globals** window should look something like this:



Setting the *WithEvents* property

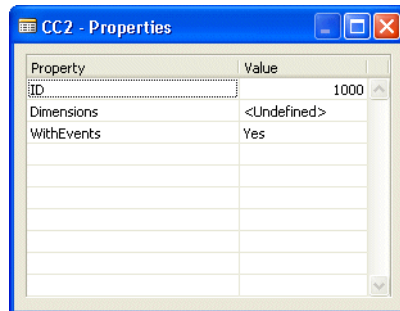
Now that you have declared the variables for the Dynamics NAV Communication Component and a bus adapter, you must set the *WithEvents* property for the Dynamics NAV Communication Component to *Yes*. By doing so, you subscribe to events from the Dynamics NAV Communication Component. This means that C/SIDE can receive notification of any inbound streams.

To set the *WithEvents* property:

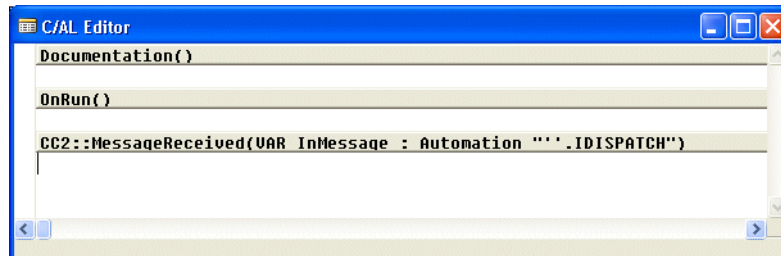
- Open the **C/AL Globals** window for this codeunit and select the variable you created earlier called CC2.



- Open the **Properties** window and click the AssistButton ▾ in the **Value** field of the *WithEvents* property and select **Yes**:



Setting the *WithEvents* property to *Yes* automatically creates a trigger in the codeunit for each event that the selected subtype of the global variable provides. In the case of the Dynamics NAV Communication Component, it creates the following trigger:



Creating the automation servers

Before you can use the Dynamics NAV Communication Component and a bus adapter, you have to create instances of them. To do this, you use the C/AL function `CREATE`. To call `CREATE`, enter the following code in the `OnRun` trigger of the codeunit:

```

OnRun()
IF ISCLEAR(CC2) THEN
    CREATE(CC2);
IF ISCLEAR(SBA) THEN
    CREATE(SBA);

CC2.AddBusAdapter(SBA, 0);
SBA.OpenSocket(8079, '');

```

This code creates an instance of the Dynamics NAV Communication Component and a bus adapter. The bus adapter now uses port 8079 on the local computer.

Writing code in the trigger

The next step is to write the code in the `MessageReceived` trigger that determines what happens when the event occurs.

Enter the following code in the MessageReceived trigger:

```
CC2::MessageReceived(UAR InMessage : Automation ""'.IDISPATCH'')
InMsg:= InMessage;

InS:= InMsg.GetStream();

WHILE NOT (InS.EOS) DO
BEGIN
    InS.READ(txt);
    MESSAGE(txt);
END;

InMsg.CommitMessage();
```

This code locates the input stream and uses the `InStream.EOS` function to find out whether or not the input stream has reached the End of Stream (EOS). It then uses the `InStream.READ` function to read the `InStream` object. The data is read in binary format. Finally, the code displays the text from the instream in a message.

Incidentally, the `InStream.READ` function has an optional parameter: `InStream.Read(Variable, [Length])`, that allows you to specify the number of bytes that you want to read from an `InStream` object. For more information about streams, see the *C/SIDE Reference Guide* online Help.

#### Important

.....

If you delete a global variable to which event triggers are associated, the event triggers and the code they contain are also deleted. Furthermore, if you change the Data Type or Subtype of a global variable, all the event triggers associated with this variable and their code are deleted.

.....

Save and compile the codeunit.

## Event Triggers

When you have enabled C/SIDE to receive events from a component that it controls, for example, an automation server, you should be aware of the following limitations and restrictions. The information is also relevant for component developers.

There are certain limitations on the triggers that are automatically generated for the events, which the component provides. Furthermore, incoming data is also subject to certain restrictions.

### Limitations on the Triggers

- C/SIDE only supports the default outgoing interfaces, that is, the source interfaces, which are defined by the automation variable. If more than one outgoing interface is defined by the automation server for a class, only event triggers for the default outgoing interface are generated in the C/AL code.
- Function calls can have a maximum of 39 parameters.
- Prototype text strings for functions can contain a maximum of 1024 characters.
- The connectable object strategy in COM is used to connect Dynamics NAV and the automation server. The Sink object defined in this strategy and implemented in

Dynamics NAV only supports the IDispatch interface (and IUnknown). It is therefore expected that the automation server calls on IDispatch when executing events.

- Parameter names are truncated to a maximum of 30 characters.
- There are no return values on event triggers.
- The variable name along with "::" and the event trigger name are truncated to a maximum of 30 characters.

### Restrictions on Incoming Data

All received data is copied to an internal data type, which can handle any data type that COM allows. No data is lost in this conversion and there is no check for valid C/AL data types. The data remains in this internal data type until it is used inside the trigger. When the data is used, it is converted to the appropriate C/AL data type.

However, if the data type is Variant, no conversion occurs. No data is lost in the conversion and all the required checks are made.

If the conversion cannot be performed successfully because there is an invalid data type, or because the data is outside the range, the event trigger causes an error message to be displayed and terminates execution. Note that if the data is never used in the event trigger, no checks for valid data, data type and data range are performed.

If a parameter is a VAR parameter (that is, called ByRef) and the data is used inside the event trigger, there is an implicit conversion just before the event trigger returns. A check is made to see if the conversion can be performed. If the conversion cannot be performed, an error message is shown and the event trigger terminates.

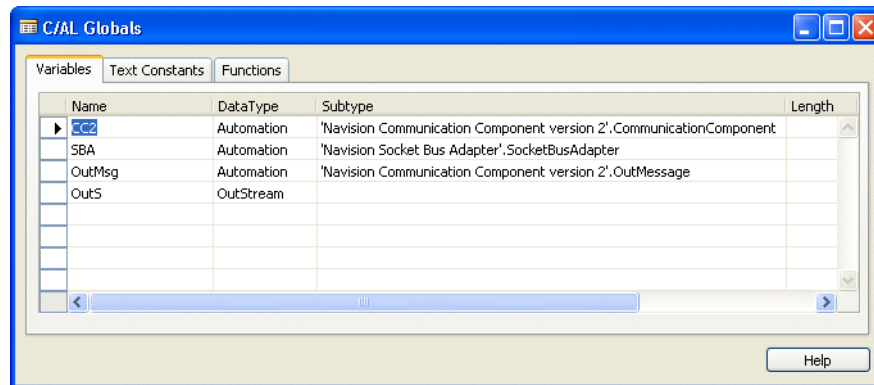
### Sending the Stream

The next step in this example is to create the codeunit that sends the stream.

- 1 Start by creating a new codeunit and declaring the following variables:

Variable Name	Data Type	Subtype
CC2	Automation	Dynamics NAV Communication Component version 2.CommunicationComponent
SBA	Automation	Dynamics NAV Socket Bus Adaptor.SocketBusAdapter
OutMsg	Automation	Dynamics NAV Communication Component version 2.OutMessage
outS	InStream	

The **C/AL Globals** window should look something like this:



Sending the message 2 Add the following code to the OnRun trigger of the codeunit:

```
OnRun()
IF ISCLEAR(CC2) THEN
    CREATE(CC2);
IF ISCLEAR(SBA) THEN
    CREATE(SBA);

CC2.AddBusAdapter(SBA, 0);
OutMsg:= CC2.CreateOutMessage('Sockets://localhost:8079');

OutS:= OutMsg.GetStream();
OutS.WRITE('Hello world');
OutS.WRITE('Using my First');
OutS.WRITE('Navision Socket bus adapter');
```

This code creates an instance of the Dynamics NAV Communication Component and a bus adapter. It then creates the outmessage. If the outmessage cannot be created it returns NULL.

The `ICommunicationComponent::CreateOutMessage` function has a destination parameter. This parameter has the following syntax:

Bus adapter type://destination

For example:

Named pipe://myServer/myPipe

Sockets://myserver::portnumber

Message queue://MyMessageQueueServer\MyQueue

The socket bus adapter is used to send the outmessage to Port 8097. The text of the outmessage is then specified.

3 Save and compile the codeunit.

## Running and Testing the Example

The last step in this example is to test the codeunits that you have just created.

- 1 Start a Dynamics NAV client and open the database that contains the codeunits that you have just created.
- 2 In the Object Designer, select the codeunit that receives the stream and click Run.

This is a single instance codeunit and is therefore loaded into memory and stored there.

- 3 Start another Dynamics NAV client and open the same database.
- 4 In the Object Designer, select the codeunit that sends the stream and click Run.
- 5 Switch back to the first client and you should see that it is receiving the stream and displaying the text that you defined earlier in the codeunit that sends the stream.

## 19.5 Using Custom Controls from C/SIDE

As mentioned in "Terminology and History" on page 364, *Custom Controls* are (or were) also known as OLE Controls and ActiveX Controls. They have also been called OCXs because they often have the file name extension `.ocx`.

Terminology in C/SIDE

In C/SIDE, the term *Custom Control* is used, for example, in the Tools menu. When you want to use a control, you define a variable (global or local) of type OCX and reference the control as the subtype of this variable.

### Simple Example

To show you how simple it is to use a custom control in C/SIDE, we will look at the C/Front OCX. C/Front comes with a Help file that contains help for both the methods and the properties for the control as well as help for the sample application.

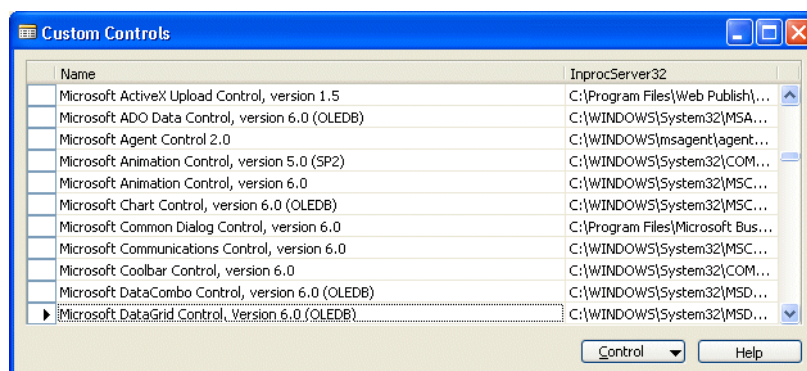
In this example, you will build a simple form that uses the C/Front OCX to connect to a server, open a database, open a company of your choosing and counts the number of records that exist in a table that you specify. This form will have access to the functions and properties supported by the C/Front OCX.

### Installing and Registering the Control

A control must be physically installed on the computer before you can use it. But a control must also be registered with the operating system before it can be used.

If the control has been installed physically by copying it to the hard disk, but has not yet been registered, you can follow this procedure to register the control from within C/SIDE:

- 1 Copy the control from the distribution media to the hard disk. The C/Front folder contains a `readme.txt` file with more information.
- 2 On the menu bar, click Tools, Custom Controls... to open the **Custom Controls** window:



- 3 Click Control, Browse to open a standard Windows dialog that you use to locate the OCX you want to register.
- 4 When you have located the control, select it and click Open.

The control is now registered with the system. You receive a confirmation message once the registration is complete. Click OK to return to the **Custom Controls** window and the control you have just added appears in the list.

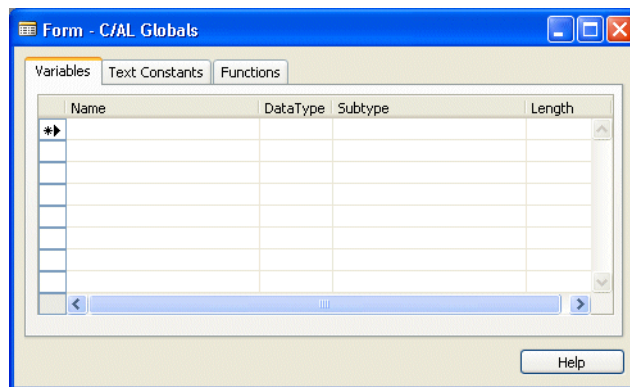
### Using the Control in C/AL

The control must be declared as either a global or a local variable before you can access its methods and properties from C/AL. Once you have done this, you can use the methods and set the properties, and you can see the methods and the properties in the Symbol Menu. If you press F1 while a method or a property is selected in the Symbol Menu, you will get context-sensitive Help for this method or property from the Help file of the control (provided there is a Help file. It is up to the creator of the control to provide such a file). C/Front comes with a Help file called `cfont.chm`. To open this Help file, double-click it.

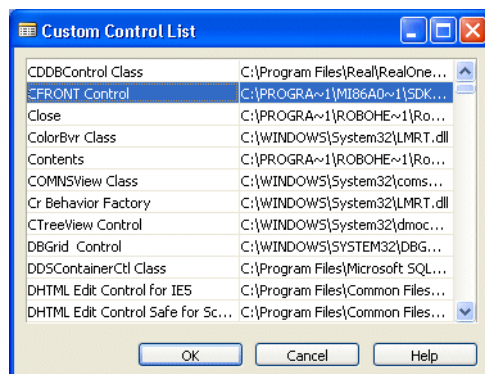
### Declaring the Control as a Variable

To declare the control as a variable:

- 1 In the Object Designer, create a new blank form that is not based on any table.
- 2 On the menu bar, click View, C/AL Globals to open the following window:

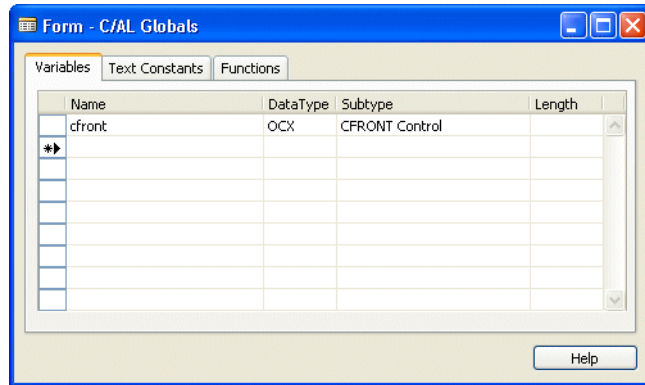


- 3 Give the variable a name (this example uses `cfront`), and select OCX as the data type. In the **Subtype** field, click the AssistButton  $\uparrow$  to open the following window:



- 4 Find the control on the list, select it and click OK.

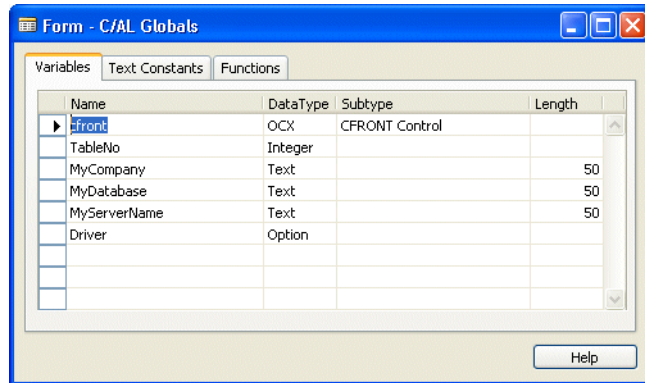
- 5 When you move the cursor out of the **Subtype** field the name of the control appears in the field and the window should look like this:



- 6 Declare the following variables:

Name	DataType	Subtype	Length
TableNo	Integer		
MyCompany	Text		50
MyDatabase	Text		50
MyServerName	Text		50
Driver	Option		

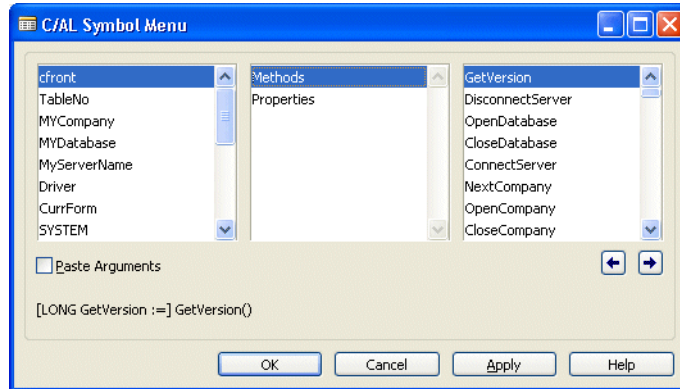
- 7 The **C/AL Globals** window should look something like this:



- 8 Select the variable called Driver and open the **Properties** window (Shift+F4). In the **Value** field of the *OptionString* property enter *NDBCS,NDBCN*.



All these variables have now been added to the C/AL Symbol Menu and you now have access to all the methods and properties contained in the C/Front OCX. To check this, click View, C/AL Symbol Menu (F5) to open the **C/AL Symbol Menu** window:

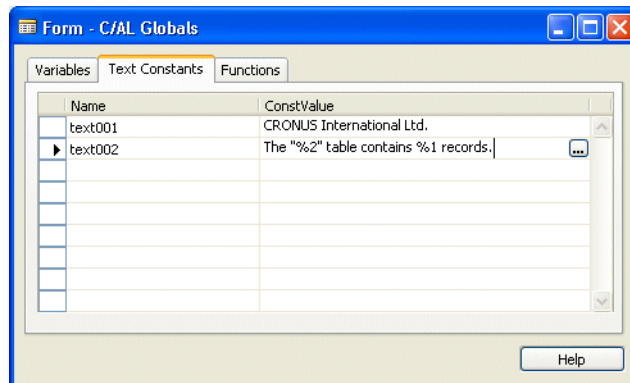


As you can see the C/Front methods and properties are now available. Select one of the methods or properties in the right hand panel and click F1. The C/Front online Help opens and displays the help for the method or the property that you selected.

Adding some text constants

The next step is to create the text constants that this form will use.

Open the **C/AL Globals** window and create the following text constants:



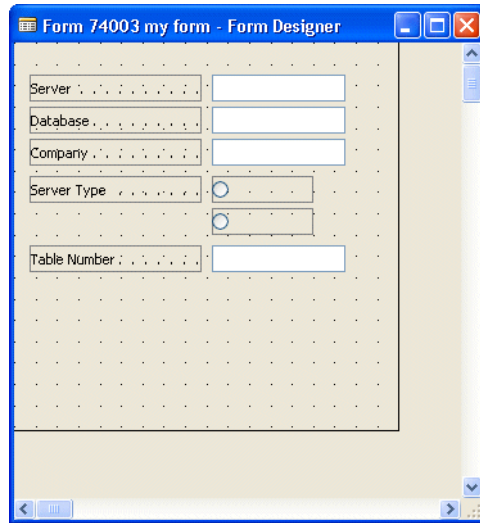
## Designing the Form

The first step in designing the form is to add the fields that you want the form to contain.

Adding the fields

- 1 Open the Toolbox and add four text boxes and two option buttons to the form.
- 2 Add labels to the text boxes and the first option button and name them *Server*, *Database*, *Company*, *Table Number* and *Server Type*, respectively.

3 The form should look something like this when you are finished:



You can always reposition the fields to suit your design.

- 4 Select the server text box and click View, Properties (Shift+F4) to open the **Properties** window.
- 5 In the **Value** field for the *SourceExpr* property, click the AssistButton ... and the C/AL **Symbol Menu** window opens.
- 6 Select *MyServerName* and ensure that the **Paste Arguments** check box is not selected and click OK.

The text box on the form should now contain the following text:  
 <=MyServerName>

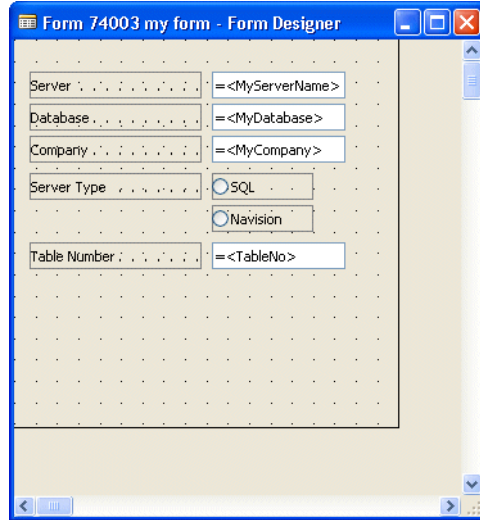
- 7 Repeat this procedure for the other three text boxes adding the *MyDatabase*, *MyCompany* and *TableNo* variables to the corresponding text box.
- 8 Select the first option button and in the **Properties** window enter the following values:

Property	Value
Caption	SQL
SourceExpr	Driver
OptionValue	NDBCS

- 9 Select the second option button and in the **Properties** window enter the following values:

Property	Value
Caption	Dynamics NAV
SourceExpr	Driver
OptionValue	NDBCN

The form should now look something like this:



### Adding the Code to the Form

Now that you have created the basic form it is time to add the code that will make it work.

In this example, you will place most of the code in the *OnPush* trigger of a command button at the bottom of the form.

- 1 Open the form in the form designer and add a command button to the form.
- 2 Select the command button and open the **Properties** window (SHIFT+F4) and enter *Connect* as the name and the caption.
- 3 Select the command button and click View, C/AL Code (F9) to open the C/AL Editor.
- 4 Enter the following code in the *OnPush* trigger:

```
CLEAR(cfront);

cfront.ConnectServerAndOpenDatabase(FORMAT(Driver),MyServerName,'tcp',
MyDatabase,0,FALSE,TRUE,'','');

cfront.OpenCompany(MyCompany);

IF cfront.OpenTable(TableNo,TableNo) THEN
MESSAGE(text002,cfront.RecCount(TableNo),cfront.TableName(TableNo));

cfront.CloseTable(TableNo);
MESSAGE('Table closed');

cfront.CloseCompany();
MESSAGE('Company Closed');

cfront.DisconnectServer();
MESSAGE('Disconnect')
```

To enter the C/Front functions open the **C/AL Symbol Menu** window and select the method or property that you want to use. Ensure that there is a check mark in the

**Paste Arguments** check box and click OK. The C/Front function is then pasted into the C/AL Editor and all you have to do is adjust the parameters.

This code uses the C/Front functions to connect to a server and open a database that you specify in the form. It uses the `TCPIP` protocol. It then opens the company that you specify in the **MyCompany** field. Next it opens the table whose number you specify and counts the number of records that the table contains. It then displays a message telling you how many records the table contains. When you click OK in the message window it closes the table, the company and disconnects from the server.

#### Note

.....

This form only uses Windows Authentication to connect to the server. This is specified in the 7th parameter of the `ConnectServerAndOpenDatabase` function. If you want to use Database Authentication you must set this parameter to `False` and enter your user ID and password in the 8th and 9th parameters respectively.

.....

The form could look something like this when it is finished:

#### Handling Exceptions

As mentioned earlier (see page 366), you cannot use the exception-handling mechanisms that are described in, for example, *Inside OLE*. The samples in the C/OCX Samples show you how to handle exceptions in another easy way (but only for controls you create yourself). The C/OCX samples are installed as part of the SDK/C/Front installation.

The control has two properties, `Error` and `ErrorCode`, that are used for exception handling. Every time a method is called, the `Error` property (a Boolean) is used to flag errors; it is set to `TRUE` if an error occurred and `FALSE` if no error occurred. What constitutes an error is defined by the methods in the control. In the control in the C/OCX Samples one error is, for example, that an arithmetic operation caused numeric overflow, another that an illegal (out of range) value was passed as a parameter.

When an error occurs (and `Error` is set to `TRUE`), the `ErrorCode` is set to a numeric code that can be used by the caller to decide what action to take (for example to display an appropriate message to the user).

Calls to methods in the control can then be wrapped like this in C/AL:

```
IF (Fin.Error) THEN
    // do error handling, for example:
    ErrorHandlerFunction(Fin.ErrorCode)
ELSE
    // continue processing
```

## 19.6 Acquiring Controls

**Buy** You can buy a third-party control and use it in C/SIDE, as long as the control fits within the restrictions imposed. These restrictions are described on page 366. In short, they are:

- Only non-visual controls are supported.
- Events are not supported.

**Develop** You need a suitable tool to develop a control yourself. Currently, the recommended tools are:

- Microsoft Visual C++ 4.0, or later
- Microsoft Visual Basic 5.0

There are other tools on the market, but the tools mentioned earlier have been tested with C/SIDE. Furthermore, they both have highly efficient wizards that make creating controls much easier (as opposed to programming by hand on top of the "raw" API).

On the other hand, it should be said that the controls that are created with the wizards are considerably larger than controls that you can create directly. They also require additional runtime libraries (for example, controls created with the wizards in Microsoft Visual C++ require the Microsoft Foundation Classes runtime library). This is a consideration of real importance for controls that are meant to be loaded over the Internet. This is probably less important for controls distributed as extensions to C/SIDE.

If you decide to create controls without using the wizards (in C++), you should study the recommended books, especially *Inside OLE*, and the documentation that comes with Microsoft Visual C++ very carefully before embarking on the project.

**Part 7**  
**Dataports**





## **Chapter 20**

### **Dataports**

Dataports are used to export data to external text files, and to import data from external text files.

- What Are Dataports?
- Designing Dataports
- Exporting Data
- Importing Data

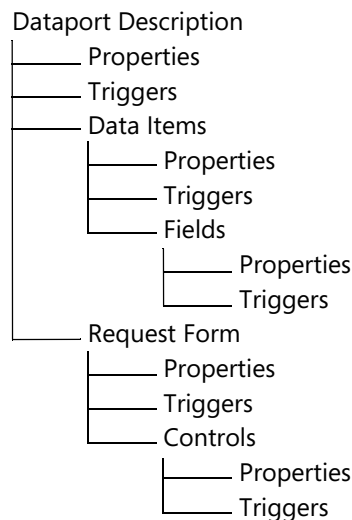
## 20.1 What Are Dataports?

Dataports are objects that you use to import data from and export data to external text files. There is a wide range of options for defining the format of the external file while you are importing and exporting the object.

When you are importing data, you can control what happens if a record in the file being imported has the same value in the key as an existing record in the database table. In addition, at field level, you can control whether or not the OnValidate trigger for each field should be run.

Dataports can be *dynamic*, meaning that when you execute the dataport it determines whether you are importing or exporting and the name of the file to read from or write to. This can be achieved either by defining options that the user sets in the request form or by programming.

The following diagram shows the components of the dataport object:



The diagram shows that a dataport is composed of a number of different components. Here is a short description of each component:

**Dataport Description** This is a complete description of the dataport including how data is collected, how data is formatted when written to the output file, and so on. The dataport description is stored in the database.

**Data Item** A data item corresponds to a table in the database. You define data items so that you can retrieve information from the tables in the database.

**Field** A dataport field can be a field in a data item (a database table), a field in a file from which data is to be imported, or a source expression to be executed during import or export. Fields in the external file are defined either as having a fixed length, or as being delimited by certain characters that you define.

**Request Form** A request form is a form that is run before the dataport is executed. The request form is used to gather requests and options from the user, for example, the name and location of the external file.

Dataports can be run without user interaction.

NAV Application Server only supports dataports that don't use request forms.

**Property** A property is an attribute of an object – dataport, data item, field, and so forth – that characterizes the object in some way. For example, this could be the length and position of a field in a line (when importing), or whether the OnValidate trigger of a field should be executed (when inserting imported data into a table). Properties are set in the **Properties** window of an object.

**Trigger** Certain predefined events that happen to a dataport cause the system to execute a user-defined C/AL function – the event *triggers* the function. As you can see in the diagram, the dataport itself, the data items, the fields, the request form and the controls on the request form all have triggers. You can edit triggers in the C/AL editor.

## Logical Design

Designing a dataport involves two distinct tasks: designing the *data model* and defining the layout of the *external file*.

### Designing the Data Model

You build the data model by designing *data items*. A data item corresponds to a table.

**Export** When you are exporting data, each data item is iterated for all the records in the underlying table, and you can set up sorting order, keys and table views to use. You can decide whether or not each individual record should be written to the external file.

**Import** When you are importing data, the records read from the external file can be inserted into tables that correspond to the data items. You can examine the records before inserting them and you can specify:

- whether or not the records should be inserted at all.
- whether or not the records should be inserted automatically.
- whether or not the records that are already in the database should be overwritten or updated when a record with the same primary key is read from the external file.

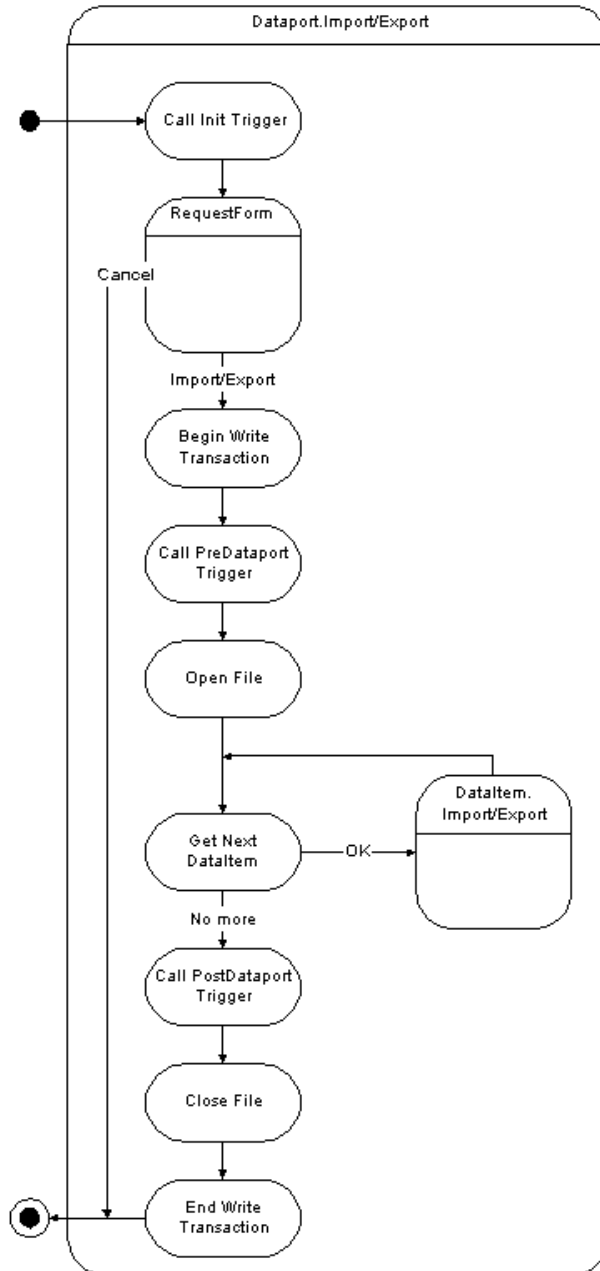
### External file

The layout of the external file is defined by means of a set of dataport properties. When you are importing, these properties describe how the input stream should be broken up into records and fields. When you are exporting, these properties describe how the fields and records should be written to the file.

The properties that you can set vary depending on the format of the external file.

### How a Dataport Is Run

The following flow chart is a simplified version of the full set of dataport flow charts in Appendix C, "Dataport Flow Charts", on page 595.



- 1 When you start the dataport run, the OnInitDataport trigger is run. This trigger can be used to initialize variables, but should not be used for general processing purposes.
- 2 When the OnInitDataport trigger has been executed, the request form for the dataport is run, if it is defined. You can cancel the dataport run from the request form.

- 3 If you choose to continue, the dataport enters a transaction (a Begin Write Transaction (BWT) is issued) and then the OnPreDataport trigger is called. No data has been processed at this stage.
- 4 The OnPreDataport trigger can be used to process the user input from the request form.
- 5 When the OnPreDataport trigger has been executed, the external file is opened, and the processing of the first data item begins.
- 6 When the first data item has been processed, the next data item, if there is one, is processed in the same way.
- 7 When there are no more data items, the OnPostDataport trigger is called. You can use this trigger to do any post processing that is necessary.
- 8 When the OnPostDataport has been processed, the external file is closed.
- 9 The transaction that was entered in step 3 ends with an End Write Transaction (EWT) being issued.

The processing of each data item (steps 5 and 6) is, of course, different for importing and exporting. For more detailed flow charts, see Appendix C. What is important to note in the overall chart is that the entire processing of a dataport takes place within a transaction. This means that if the processing is interrupted before the final EWT is issued, no trace is left of the interrupted run in the database (an external file will, however, often have been corrupted.)

### Saving, Compiling and Running a Dataport

After you have designed a dataport, you must save and compile it before it can be run. Normally, you do this when you have finished designing the dataport. However, you may want to save a dataport that is not yet finished and therefore cannot be compiled. You can also test-compile a dataport without closing or saving it.

### Saving and Closing a Dataport

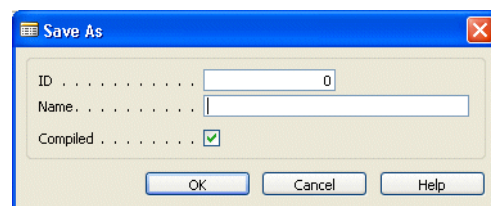
A dataport is closed when the **Dataport Designer** window is closed.

#### Note

If you enter ID and Name as dataport properties, these values will be used, and you will not be prompted for ID and Name when you close the dataport.

To save a dataport:

- 1 Click File, Save and give the dataport a name and an ID.



The ID must be unique and follow the rules for numbering objects. For more information about object ID numbers, contact your Microsoft Certified Business Solutions Partner.

- 2 The **Compiled** field is set to TRUE by default (displayed as a check mark). If you don't want to compile the dataport, click the field to remove the check mark.
- 3 Click OK to save the dataport.

You can save a dataport without closing it by clicking File, Save or Save As. You can use the Save As option to make a copy of an existing dataport.

### Compiling a Dataport

Dataports, like the other objects in C/SIDE, must be compiled before they can be run. As described earlier, you can compile a dataport whenever you save it.

When you are designing a dataport, you can test-compile it to find possible errors, by clicking Tools, Compile (F11).

### Running a Dataport

In a finished application, your dataports are generally incorporated into menus, or else they can be called, for example, from a command button on a form. However, when you are designing dataports, you often want to run them before they have been integrated into the application.

Test-running dataports When you are designing a dataport, you can test-run it by clicking File, Run (CTRL+R). The dataport is compiled and run in its current stage of development. It is not saved, which means that you can use this function to verify that the changes you are making work as intended before you save them.

**Note**

.....  
If the dataport is for importing data, no records are actually saved in the database table during the test-run.  
.....

Running dataports from the Object Designer To run a dataport from the list of dataports in the Object Designer, select it and click Run.

## 20.2 Designing Dataports

This section is a general description of the various elements that are involved in designing dataports. The tables that list the properties and triggers contain a brief description of what each property or trigger is used for. Full explanations can be found in the *C/SIDE Reference Guide* online Help.

Designing a dataport consists primarily of setting various properties. The following sections explain which properties to use, and how to use them.

### Dataport Properties

This set of properties describes the dataport in general, and this is also where you specify the format of the external file.

The *Import* and *FileName* properties can be set and reset dynamically. For example, you can create a dataport that allows the user to decide whether to import or export, or to select the name of the external file to read from or write to (or you can generate a filename automatically once the dataport is run).

Property	Meaning
ID	The ID of the dataport – must be unique among dataports.
Name	The name of the dataport.
Caption	The caption shown on the request form window. For example, the default value in English (United States) is the same as the name of the dataport.
CaptionML	The list of translations of the object's caption.
Import	Whether or not the dataport imports or exports data. It can be set dynamically in the OnPreDataport trigger. It cannot be changed after the OnPreDataport trigger has been run.
FileName	The name of the external file to write data to or read data from. This property can be set dynamically. If you reset the file name after a file has been opened, this file is closed and a new file is opened.
FileFormat	The format of the external file: <i>Variable</i> or <i>Fixed</i>
FieldStartDelimiter	When FileFormat is Variable, this property is used to define the string that marks the beginning of a field on input or output.
FieldEndDelimiter	When FileFormat is Variable, this property is used to define the string that marks the end of a field on input or output.
FieldSeparator	When FileFormat is Variable, this property is used to define the string that separates fields on input or output.
RecordSeparator	The string that separates records on input or output.
DataltemSeparator	The string that separates data items on input or output.
UseReqForm	Whether or not the request form should be run before the dataport is run.
ShowStatus	Whether a status window will be shown while the dataport is running. This window also has a Cancel button that you can use to interrupt the dataport run – otherwise this is only possible if you create a dialog yourself.

TransactionType	There are four basic transaction type options: Browse, Snapshot, UpdateNoLocks and Update. Each transaction type defines the behavior of a transaction in Dynamics NAV and takes effect from the beginning of a transaction.
Permissions	The permissions of the dataport to access database objects. (The dataport can have wider permissions than the individual user. This means that the user might be able to use dataports that retrieve information from tables that the user cannot normally access.)

The *FieldStartDelimiter* and the *FieldEndDelimiter* properties are used to place the contents of a field in quotation marks in situations where the data in the field contains the character that is defined as a separator (*FieldSeparator*, *RecordSeparator* or *DataItemSeparator*). These delimiters are not obligatory. This means that if only one field of a record needs to be placed in quotation marks, only this field has to be enclosed by the *FieldStartDelimiter* and *FieldEndDelimiter* characters. The other fields can optionally have the delimiters; it makes no difference when you are importing. When you are exporting, all the fields are written to the external file with all the delimiters and separators.

**File Format**

The *FileFormat* property determines the format of the external file and defines how a record is read from or written to the file. The *RecordSeparator* property defines how the file is broken up into records, and the *FileFormat* property then defines how to break each record up into fields. Finally, the *DataItemSeparator* property defines how data items should be separated if the dataport has more than one data item. Note that data items cannot be nested, although a dataport can have several data items that are processed sequentially.


**Note**

.....  
 The file format you set for a dataport determines which properties are available for that dataport. For example, if you set the file format to *Variable*, the *FieldEndDelimiter*, *FieldStartDelimiter* and *FieldSeparator* properties become active.  
 .....

FileFormat: Fixed      When the format of the external file is *Fixed*, the fields in a record have a fixed width. You can define the starting position and the width of each field in the record (if the record separator is a *newline* character, you can think of a record as a line of text). The positions and widths of the fields are properties of the fields and are described on page 414.

FileFormat: Variable      When the format of the external file is *Variable*, the fields in a record are delimited by characters that you define, and the fields can have varying widths. The fields are separated by the string defined as the *FieldSeparator* property.

**Data Item Properties**

 This set of properties describes the data items of the dataport. A data item is a table in the dataport.



Most of these properties are the same and have the same function as the corresponding properties of a data item in a report (see Chapter 11, page 219). Dataports have three special properties: *AutoSave*, *AutoUpdate* and *AutoReplace*.

Property	Meaning
DataltemIndent	How much the data item is indented. Can be set in the designer when creating data items.
DataltemTable	The table that the data item is based on. Can be set in the designer when creating data items.
DataltemVarName	The name of the data item as a variable. The default value is the value of DataltemTable.
DataltemTableView	The key, sort order and filters to apply.
ReqFilterHeading	The name of this tab on the request form. The default value is the name of DataltemTable.
ReqFilterHeadingML	The translations of ReqFilterHeading.
ReqFilterFields	The names of the fields that initially will be included in the ReqFilter form.
CalcFields	The names of the fields that will be calculated after a record has been retrieved.
DataltemLinkReference	The DataltemVarName of a less-indented data item that this data item will be linked to.
DataltemLink	The link between the current data item and the data item specified by DataltemLinkReference.
AutoSave	Whether or not the imported records are automatically inserted into a C/SIDE table.
AutoUpdate	Whether or not the imported records are initialized with values from an existing record with the same primary key.
AutoReplace	Whether or not imported records will automatically replace existing records with the same primary key.

### AutoUpdate, AutoReplace, AutoSave

The three *Auto* properties determine how the records that are read from the external file are handled. These properties are also used to resolve the conflict that arises when you import a record from an external file and it has the same primary key as a record that already exists in the database table.

*AutoSave* and *AutoReplace* are used to define how records are saved in the database table. Note that if *AutoSave* is No, the settings of *AutoReplace* and *AutoUpdate* have no effect. In this case, you have to handle any conflicts from your C/AL code. The

following table outlines the effects of various combinations of settings of these properties:

<b>Auto Save</b>	<b>Auto Update</b>	<b>Auto Replace</b>	<b>Record exists in database and in import file</b>	<b>Record exists only in import file</b>
No*	---	---	The record in the database is not automatically updated or replaced	The imported record is not automatically inserted in the database
Yes	No	No	A runtime error occurs, and the import is terminated	The imported record is automatically inserted in the database
Yes	No	Yes	The imported record replaces the existing record	The imported record is automatically inserted in the database
Yes	Yes	---	The imported record updates the existing record	The imported record is automatically inserted in the database

--- MEANS THAT THE SETTING DOES NOT MATTER.

\* IF AUTOSAVE IS NO, IT IS POSSIBLE TO INSERT AND MODIFY RECORDS BY USING INSERT OR MODIFY FROM C/AL.

AutoUpdate is useful in some particular situations. Its functionality is best explained with a brief example.

#### **Example**

Suppose you have a table that is an item list. You update the prices by exporting a list with item numbers (the primary key) and prices to an external file and then you do some calculations on the prices in a spreadsheet. Now, when the prices are calculated and you are ready to import the file with the new prices, it is obvious that the records read from the external file will have the same primary key as the records that already exist in the database. Using AutoSave and AutoReplace will not solve the problem. If you are replacing every record with the corresponding record from the import file, all the information except the item numbers and the prices will be lost (it is assumed that the table contains more information than just the item numbers and the prices, for example the names of the items, the stock level, and so forth).

AutoUpdate solves this dilemma. When a record is imported, it actually replaces the existing record, but any fields that are not present in the imported record are initialized with the data from the existing record instead of being left empty. The end result is that the existing record is *updated* with the information that was revised.

## **Field Properties**

This set of properties describes the fields of a record. If FileFormat is Fixed, you use the *StartPos* and *Width* properties of each field to define how a record that is read from the external file is broken into fields. When you are exporting, these properties determine how data from the database is written to the external file.

<b>Property</b>	<b>Meaning</b>
Enabled	Whether or not the field is enabled or disabled. A field that is disabled is imported from the external file, but will not be inserted into the record.

Property	Meaning
SourceExpr	The source expression of the field. When you are importing, this could be the name of the database table field where the value that is read from the external file should be stored, but it can be any valid C/AL variable. When you are exporting, this could be the value that is exported to the external file – for example the name of a database table field, but it can be any valid C/AL expression.
Caption	The caption in the currently selected language. The value is taken from the <i>CaptionML</i> property if this property is set. For example, the default value in English (United States) is the same as the name of the field.
CaptionML	The list of all the translations of the field's caption.
StartPos	The starting position of this field if FileFormat is Fixed. Positions are numbered from 1 upwards.
Width	If FileFormat is Fixed, this field contains the width of the field.
CallFieldValidate	Whether or not the OnValidate trigger is executed when the field is imported.
Format	How the field is formatted during exportation. For example, you can determine the number of decimal places, and so forth.
AutoCalcField	The system automatically calculates this FlowField.

When fields are exported, the data they contain is converted to text before being exported. If the format is Fixed and the width is smaller than the actual width of the data after conversion, the contents are truncated from the right until it has the defined width. That is, a number is not rounded or truncated as a number, but as text, from the right.

You get a warning at design-time if you have defined fields with starting positions and widths that could cause these fields to overlap. The error makes it impossible to compile (and subsequently execute) the dataport.

During importation, a value that is too large for the data type or defined width of the database table field where it is to be inserted causes an error and execution stops. As the whole dataport is within a transaction, no traces will be left of the aborted run in the database.

## Dataport Triggers

The following is a list of all the triggers that are executed when a dataport is run. These descriptions are only an overview. The *C/SIDE Reference Guide* online Help contains the full and most recent descriptions.

Dataport Trigger	Executed
OnInitDataport	When the dataport is loaded, and before the request form is run and table views and filters are set.
OnPreDataport	Before the dataport is run – but after the request form has been run. Table views and filters are set when this trigger is run.
OnPostDataport	After all the data items have been processed.

<b>Dataport Trigger</b>	<b>Executed</b>
OnPreDataItem	Before the data item is processed – but after the associated variable has been initialized and the table views and filters set.
OnBeforeExportRecord	When a record has been retrieved and is ready for export.
OnAfterExportRecord	After a record has been exported to the external file. You can use this trigger, for example, to do some processing on the external file before the next record is exported, such as moving the file pointer.
OnBeforeImportRecord	Before a record is imported from the external file. You can use this trigger, for example, to do some processing on the external file before importing the next record, such as moving the file pointer.
OnAfterImportRecord	After a record has been imported from the external file, but before it is inserted in the table. You can use this trigger, for example, to process the record before inserting it or to examine it in order to decide whether to insert it at all.
OnPostDataItem	When the data item has been iterated for the last time.
OnAfterFormatField	After the value of a field has been formatted, but before the text is written to the external file. This trigger gives you access to the formatted value in its text format.
OnBeforeEvaluateField	After a field has been read from the external file, but before the value has been evaluated and validated. This trigger gives you access to the imported field in text format.

## 20.3 Exporting Data

This section describes how to create dataports and walks you through the steps involved in creating the four fundamental types of dataports: importing and exporting, each with a fixed and a variable format of the external file. The last example tells you how to create a dataport that both exports and imports, and updates records in the database when importing.

### Exporting - Fixed Format

This dataport exports records in a fixed format to a file. The records in the file are separated by new lines, and within each record or line, a field has the same width in all the records, no matter how wide the actual data of the field is.

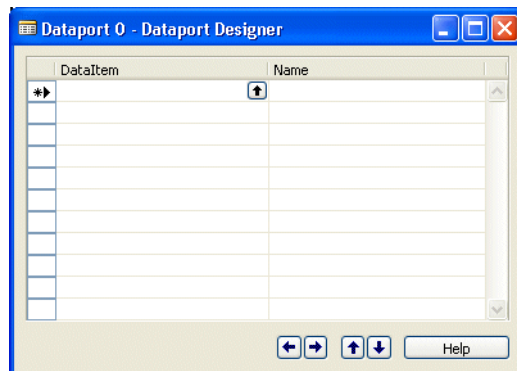
In this example, you use the **G/L Account** table (the Chart of Accounts). There are several FlowFields among the fields that are exported and these have to be calculated during the exportation.

### Simple Version

The first version is very basic. You will refine it in the next subsection.

To create a dataport:

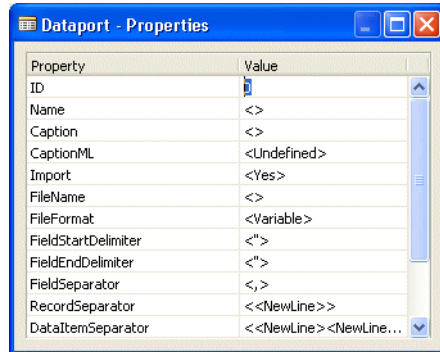
- 1 Click Tools, Object Designer (SHIFT+F12).
- 2 In the Object Designer, click Dataport, New and the **Dataport Designer** window appears:



- 3 In the first **DataItem** field, click the AssistButton ↑ and select the **G/L Account** table from the **Table List** window that appears.

The name is set by default to the name of the table. You do not have to change it in this dataport.

- Click an empty line in the Dataport Designer to select the dataport itself and click View, Properties (SHIFT+F4) to open the **Properties** window of the dataport (not for the data item). You can also select the dataport by clicking Edit, Select Object.



- In the **Properties** window, leave the default settings except for:

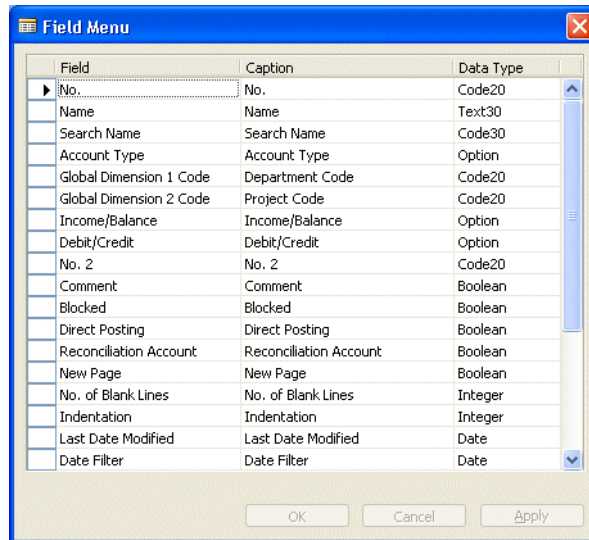
Import – set it to No to create a dataport that exports data.

FileFormat – set it to Fixed.

You have just created a dataport with a single data item and must now specify which fields from the underlying table will be used in the dataport.

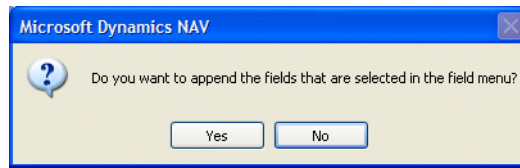
Adding dataport fields

- In the Dataport Designer, select the G/L Account data item.
- Click View, Dataport Fields and the **Field Designer** window appears.
- When the Field Designer is open, click View, Field Menu.

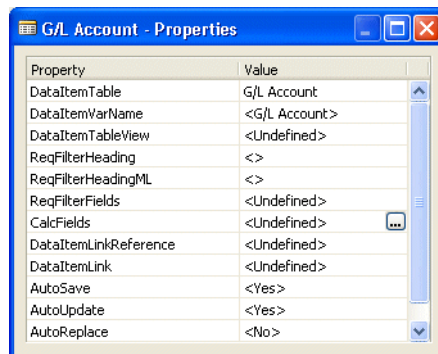


- In the **Field Menu** window, select the fields that you want to export. For this dataport, select the **No.**, **Name**, **Balance at Date** and **Net Change** fields.

10 Click the Field Designer. You are asked if you want to append the selected fields. Click Yes.

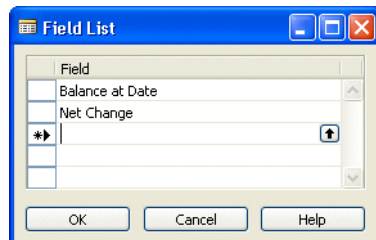


11 Select the G/L Account data item and open the **Properties** window (SHIFT+F4).



12 In the **Value** field of the *CalcFields* property, click the AssistButton ... to open the **Field List** window.

13 In the **Field List** window, select the fields from the data item that must be calculated when they are exported, that is, the FlowFields. In this example, select the **Balance at Date** and **Net Change** fields and click OK.

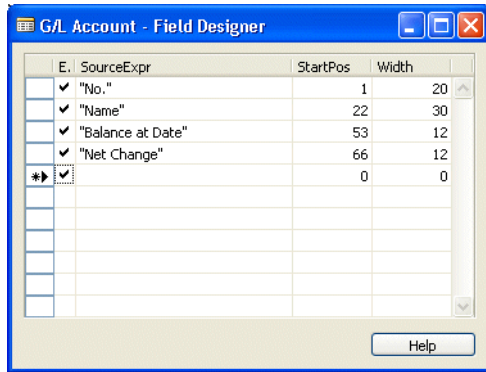


Now, you are ready to run the dataport. Click File, Run to run it before saving it. If you want to save the dataport first, you will be prompted to give it a name and a number.

StartPos and Width properties

After you have inserted the fields from the Field Menu, you can check the settings of the *StartPos* and *Width* properties of all the fields in the Field Designer.

Click View, Dataport Fields to open the **Field Designer** window:



As you can see, both StartPos and Width have been filled in for you. They are assigned values according to these rules:

Data type of field	Width assigned
Code	If actual length > 10, actual length is used, otherwise 10
Text	If actual length > 30, actual length is used, otherwise 30
Date	11
Time	10
Option	10
Decimal	12
Integer	7
Boolean	10

Running the dataport

When you run the dataport, you see the default request form. The first tab is of little interest, but the second tab, **Options**, is important because it is here that you enter the name of the external file to write to.

When you have entered the name of the file, click OK to run the dataport.



When the dataport run is finished, you can open the file in a text editor:

### Refined Version

This simple dataport could be more user-friendly. Depending on what the dataport is to be used for, several things could be changed. Here are some examples of what you can do:

- The first tab in the request form should not be shown because you do not want the user to set filters and keys.
- Only accounts where the Account Type is Posting or End-Total should be exported.
- The numbers must be formatted as thousands. There must be no thousand separators, no decimals, and the sign should be prefixed.

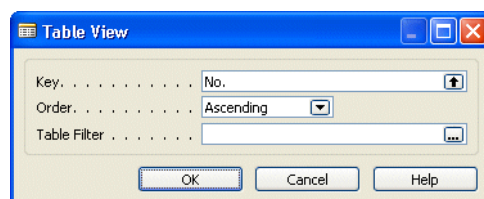
Changing the request form

You *must never* set the *UseReqForm* property to No to remove the request form. If you do, you will make it impossible for the user to enter the name of the file to write the data to. In fact, running the dataport will cause a runtime error because no filename has been set.

Instead, you can keep the **Options** tab and remove the data item tab by setting the *DataltemTableView* property of the data item. When this property is set, as opposed to being left undefined, the user cannot change key or sort order and cannot set a table filter, and the corresponding tab will be removed by the system.

To set the *DataltemTableView* property of the data item:

- 1 In the Dataport Designer, select the G/L Account data item and open the **Properties** window (SHIFT+F4).
- 2 In the *DataltemTableView* property, click the AssistButton ... to open the **Table View** window:

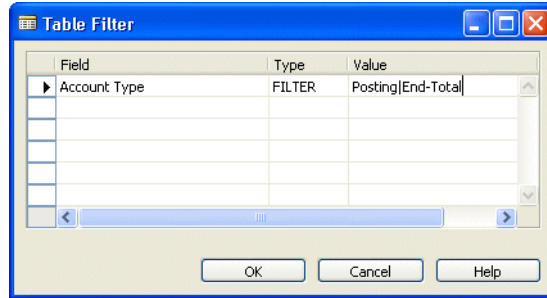


- Fill in at least one of the fields. In this example, the **Key** field has been set to No. and the **Order** field has been set to Ascending.

Selecting specific account types

To select only some account types, you must set a table filter. You can do this in the *DataltemTableView* property.

- In the **Table Filter** field, click the AssistButton ... to open the **Table Filter** window:



- Set the values of the **Field**, **Type** and **Value** fields as shown in this picture. This creates a table filter that selects records where the Account Type is either Posting or End-Total (the character between the two values is a | (pipe) and means OR.)
- Click OK to close the **Table Filter** and **Table View** windows.

Changing the formatting of numbers

To export the decimal fields, **Balance at Date** and **Net Change** as thousands (so that the number 1.444.723,67 is exported as 1444), you have to use the *SourceExpr* property of these fields:

To change the formatting of number fields:

- Select the G/L Account data item and click View, Dataport Fields to open the Field Designer.
- In the **Field Designer** window, select the **Balance at Date** field and then click View, Properties (SHIFT+F4).
- Enter the following expression as the *SourceExpr* property of this field:

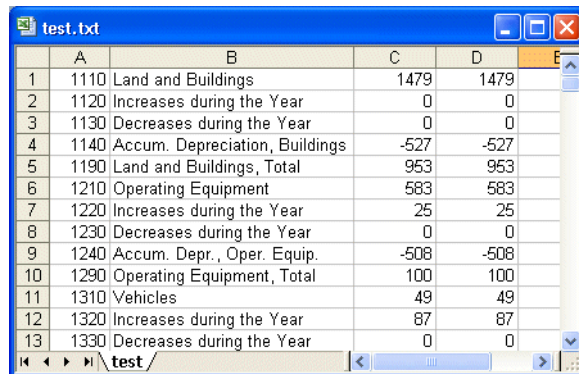
```
FORMAT(ROUND("Balance at Date"/1000,1,'='),0,1)
```

This C/AL statement ensures that the value of the **Balance at Date** field is divided by 1000. The result is rounded by the `ROUND` function, and then the `FORMAT` function is used to render the value returned by the `ROUND` in format 1 (which for a decimal value means <Sign> <Integer> <Decimals>).

- Enter this expression as the *SourceExpr* of the **Net Change** field:

```
FORMAT(ROUND("Net Change"/1000,1,'='),0,1)
```

The file that is created by this dataport looks like this when it is imported into a spreadsheet:

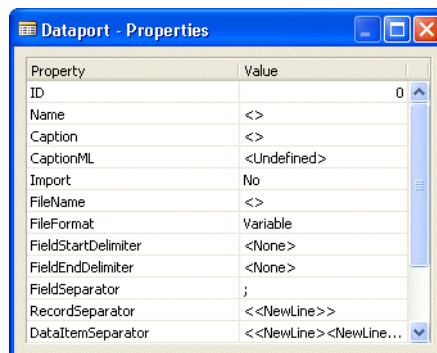


	A	B	C	D
1	1110	Land and Buildings	1479	1479
2	1120	Increases during the Year	0	0
3	1130	Decreases during the Year	0	0
4	1140	Accum. Depreciation, Buildings	-527	-527
5	1190	Land and Buildings, Total	953	953
6	1210	Operating Equipment	583	583
7	1220	Increases during the Year	25	25
8	1230	Decreases during the Year	0	0
9	1240	Accum. Depr., Oper. Equip.	-508	-508
10	1290	Operating Equipment, Total	100	100
11	1310	Vehicles	49	49
12	1320	Increases during the Year	87	87
13	1330	Decreases during the Year	0	0

### Exporting - Variable Format

This sample dataport exports the same records as the previous example, but in a variable format. Each field in a record will be delimited by characters that you define and will only have the width of the actual data of the field in each of the records.

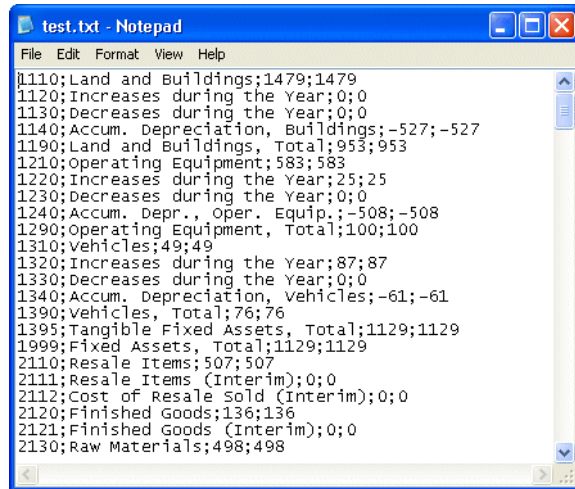
Very few changes are required to create this dataport instead of the dataport with fixed width format. You must specify the right set of options for the data item, as follows:



Property	Value
ID	0
Name	<>
Caption	<>
CaptionML	<Undefined>
Import	No
FileName	<>
FileFormat	Variable
FieldStartDelimiter	<None>
FieldEndDelimiter	<None>
FieldSeparator	;
RecordSeparator	<<NewLine>>
DataItemSeparator	<<NewLine>><NewLine...>

- Set the *FileFormat* property to Variable.
- Set the *FieldSeparator* property to ; (semicolon).
- Set both the *FieldStartDelimiter* and the *FieldEndDelimiter* properties to <None>.
- Let the *RecordSeparator* and the *DataItemSeparator* properties keep their default settings, which means that records will be separated by new lines, and data items by two new lines (which is not of real interest here, because you have only one data item in this dataport.)

The exported file could look as follows in a text editor:



```
test.txt - Notepad
File Edit Format View Help
1110;Land and Buildings;1479;1479
1120;Increases during the Year;0;0
1130;Decreases during the Year;0;0
1140;Accum. Depreciation, Buildings;-527;-527
1190;Land and Buildings, Total;953;953
1210;Operating Equipment;583;583
1220;Increases during the Year;25;25
1230;Decreases during the Year;0;0
1240;Accum. Depr., Oper. Equip.;-508;-508
1290;Operating Equipment, Total;100;100
1310;Vehicles;49;49
1320;Increases during the Year;87;87
1330;Decreases during the Year;0;0
1340;Accum. Depreciation, Vehicles;-61;-61
1390;Vehicles, Total;76;76
1395;Tangible Fixed Assets, Total;1129;1129
1999;Fixed Assets, Total;1129;1129
2110;Resale Items;507;507
2111;Resale Items (Interim);0;0
2112;Cost of Resale sold (Interim);0;0
2120;Finished Goods;136;136
2121;Finished Goods (Interim);0;0
2130;Raw Materials;498;498
```

The semicolon was used as FieldSeparator because the fields include both space characters and commas. Another solution would have been to use the delimiters. In that case, the FieldSeparator could also have been a comma, but what you choose should really depend upon the target application for the exported file, and upon the formats that the application supports when it imports text files.

## 20.4 Importing Data

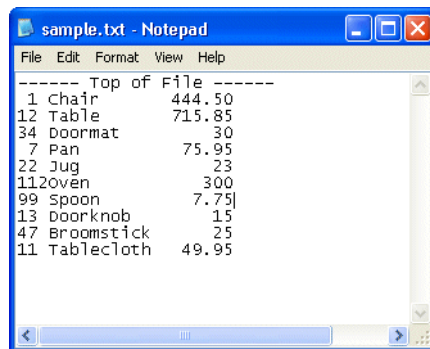
This section describes how to create dataports for importing data using each of the two formats for the external file: fixed and variable format. The last example tells you how to create a dataport that both exports and imports, and updates records in the database when importing.

### Importing - Fixed Format

Creating a dataport that imports data is not very different from creating one that exports data. However, you must consider how the imported records are going to be inserted in the database table that the data item is based on. This is especially relevant if the table already contains records that have the same primary key as some of the records that are going to be imported.

This first example assumes that the table is empty (or that other actions have been taken to ensure that no conflicts will occur). A later example will show you how to deal with any conflicts that might occur.

The file that you are going to import records from looks like this:

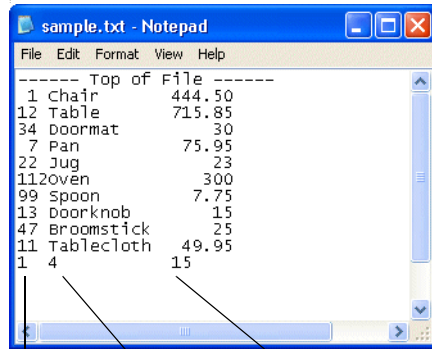


Setting up the table In this example, you will import the records into a newly created table. The table should have the following layout:

E.	Field No.	Field Name	Data Type	Length	Description
<input checked="" type="checkbox"/>	1	No.	Integer		
<input checked="" type="checkbox"/>	2	Name	Text	30	
<input checked="" type="checkbox"/>		Piece	Decimal		
<input type="checkbox"/>					
<input type="checkbox"/>					
<input type="checkbox"/>					
<input type="checkbox"/>					
<input type="checkbox"/>					
<input type="checkbox"/>					
<input type="checkbox"/>					

This means that you must decide how the lines in the file you are importing – each line will become a record in the data item – will be divided into fields. The lines have a fixed

format, and by carefully looking at the layout of the lines, the field starting positions can be deduced:



First field Second field Third field

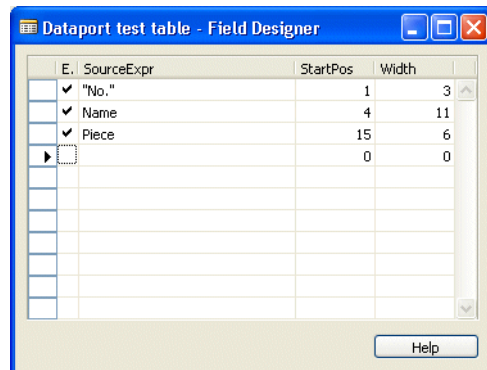
This tells us that the fields must have the following properties:

Field	StartPos	Width
No.	1	3
Name	4	11
Price	15	6

Once you have established how the lines in the import file are organized, creating the dataport is straightforward:

- 1 Open the Object Designer and click Dataport, New to create a dataport.
- 2 Open the **Properties** window of the dataport and set the *Import* property to Yes and the *FileFormat* property to Fixed.
- 3 In this example, you must create a data item based on the table that you just designed.
- 4 Click View, Dataport Fields to open the Field Designer and add all the fields from the table to the dataport.
- 5 You must change the setting of the *StartPos* and *Width* properties to values that are appropriate for the actual file you are going to import data from.

In this example, the values should be set as follows:



- 6 You can now run the dataport. Remember that if you run the dataport from inside the Dataport Designer (by clicking File, Run), the records will not be stored in the database (to have them stored, you must run the dataport from the Object Designer, or call it from a menu.) You may also want to remove the unnecessary tabs from the request form (see the description on page 421).

After running the dataport so that the records are actually imported into the database, the table will look like this:

No.	Name	Piece
1	Chair	444.50
7	Pan	75.90
11	Tablecloth	49.90
12	Table	715.80
13	Doorknob	1.00
22	Jug	2.00
34	Doormat	3.00
47	Broomstick	2.00
99	Spoon	7.70
112	Oven	30.00

Note that because **Field No.** is the primary key of the table, the order in which the records are displayed is determined by the primary key, which, incidentally, is not the same order as they appeared in the original import file.

#### Possible errors

It is easy to make errors when deciding how to "cut up" the lines in the file you are importing. In some cases, this will result in a runtime error when the dataport is run. For example, if you made an error in setting up the **Field No.** field and assigned it a width that is one character too wide. For most of the import file, this would make no difference at all; the resulting trailing space would be ignored. But the line beginning with "112Oven..." would provoke a runtime error when C/SIDE read 112O instead of 112. The "O" (upper case "o") cannot be inserted into an integer field.

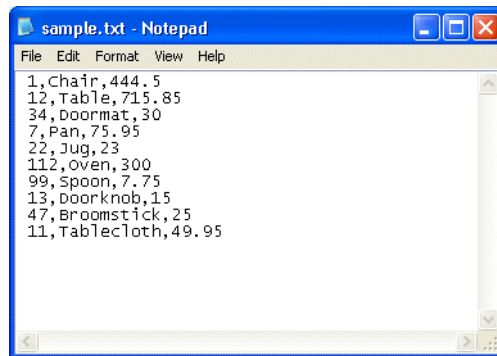
You might consider it fortunate that the error actually provoked a runtime error. In other cases, the error might not have been detected by C/SIDE, for example, if the "cut" between two text fields was placed incorrectly. If you have the opportunity, test your imports carefully before using them for production.

**Hint**

In some cases, the fields have a different order in your table that they do in the file you are importing. If this is the case, you can add the fields manually and ensure that the design of the data item reflects the order that you need. You will have to set the StartPos and Width manually.

**Importing - Variable Format**

It is not very difficult to change this dataport so that it can import an external file in variable format. Let us suppose that the file looks like this:



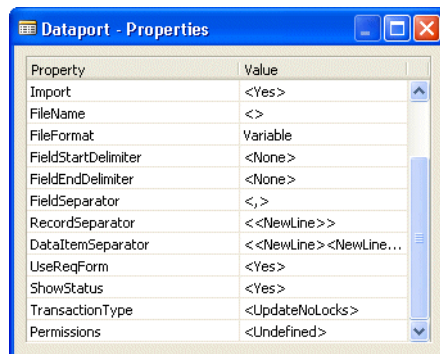
The data in this file is the same as in the file with fixed format that you imported earlier, but here the fields are separated by commas.

To change the dataport so that it can import an external file in variable format, you must alter some of the settings in the **Properties** window of the dataport.

You must make the following changes:

- 1 Set the *FileFormat* property to *Variable*.
- 2 Set the *FieldSeparator* property to a comma.
- 3 Set the delimiters to <None>.

The properties of the dataport should then look like this:







designer. However, this does not matter because these properties are not used in a dataport with a variable format.

### Importing or Exporting: A Dynamic Dataport

This final example is slightly more advanced than the previous ones. You will create a dataport that can be used to update prices in the **Item** table in an external program – you will use Microsoft Excel.

The dataport works as follows:

- 1 In the request form that opens when you run the dataport, you can select whether to import or export, and you can set a filename.
- 2 When you are exporting, only the records for the items where the **Gen. Prod. Posting Group** is RETAIL are exported.
- 3 Furthermore, only the records that have a value in the **No.** field between 1000 and 2000 are exported.
- 4 Only three fields are exported from the table – **No.**, **Description** and **Unit Price**.
- 5 The **Description** field (text) that corresponds to the **Tariff No.** field is retrieved from the **Tariff Number** table for each record in the Item data item. The text in the **Description** field is written as a fourth field when each record is exported.
- 6 The user is supposed to change the **Unit Price** field in the external program – or at least be able to do so. This means that the information in this field can be different in the external file than it is in the database. You therefore want the **Unit Price** fields from the external file to replace the values in the database when you import the file again. That is, you want the records in the database to be *updated* when you import the file.

**Note**

.....  
 One of the reasons for filtering on the **No.** field is to exclude records that contain a comma in the **Description** field. Excluding these fields makes managing this example easier in Excel.  
 .....

### Creating the Export Part

You start by creating the export part of the dataport. The first task is to determine the format that the external file should have. You are going to edit the file in Excel, so you should select a format that suits Excel. A semicolon-delimited format appears to be the best choice.

With this in mind, create the dataport:

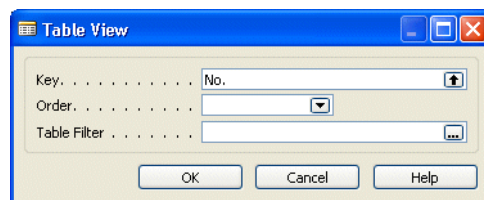
- 1 Open the Object Designer and click Dataport, New to create a new dataport.
- 2 Open the **Properties** window (SHIFT+F4) of the dataport and set the properties of the dataport as follows:

Property	Value
FileFormat	Variable – the default value

Property	Value
FieldStartDelimiter	" (quote) – the default value
FieldEndDelimiter	" (quote) – the default value
FieldSeparator	; (semicolon)

Leave the other properties at their default settings.

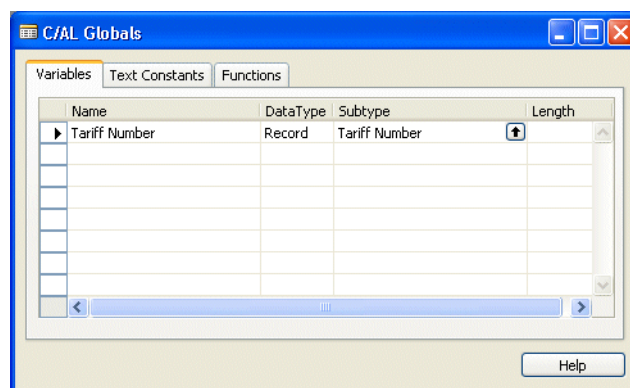
- 3 Create a data item based on the **Item** table.
- 4 Open the **Properties** window of the Item data item.
- 5 In the **Value** field of the *DataItemTableView* property, click the AssistButton ... to open the **Table View** window.



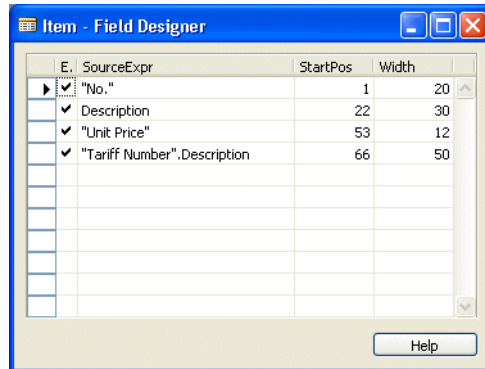
- 6 In the **Key** field, click the AssistButton ↑ and select the **No.** field in the **Key List** window.

Do not use the **Table Filter** field to set a filter even if you are only going to select a subset of the records in the **Item** table. The reason for not setting the filter here will become clear later.

- 7 Click View, C/AL Globals to open the **C/AL Globals** window and create a global variable called Tariff Number of data type Record, with the **Tariff Number** table as the subtype:



- 8 Select the Item data item and click View, Dataport Fields to open the **Field Designer** window.
- 9 Set up the dataport fields as follows:



Use the AssistButton  in the **SourceExpr** field to select the fields from the **Item** table.

However, you must enter the name of the field from the **Tariff Number** table manually. Remember to use the format shown in the picture.

To calculate the StartPos and Width of each field, open the tables in the Object Designer and look up the length of each field. The first field must always start at position 1 and not at 0.

The **Unit Price** field is of data type Decimal and its length must therefore be 12. For more information about the size of the various data types, see the section "Choosing Data Types" on page 63.

- 10 To set the filter that will select the records between No. 1000 and 2000 where the **Gen. Prod. Posting Group** is RETAIL, open the C/AL editor, and enter the following code in the *OnPreDataItem* trigger of the Item data item:

```
IF NOT CurrDataport.IMPORT THEN
    Item.SETRANGE("Gen. Prod. Posting Group", 'RETAIL');
    Item.SETRANGE("No.", '1000', '2000');
```

#### About filters

The filter is set only if the dataport is used to export (remember that the user can decide whether to import or export at runtime). The reason for using this construction instead of setting a table filter in the *DataItemTableView* property is that if you are placing the filter on a field that is not being exported with the other data. If you had used *TableFilter* the filter would always be set – also when the dataport is used to import data. However, because the **Gen. Prod. Posting Group** field is not in the file that you import, no records will be imported. If the field that you use for filtering is exported and imported, you could of course have used the **TableFilter** field in the **Table View** window to set the filter.

- 11 To retrieve the text from the **Tariff Number** table and to export it, enter the following code in the *OnBeforeExportRecord* trigger of the data item:

```
IF "Tariff No." <> '' THEN
BEGIN
    "Tariff Number"."No." := "Tariff No.";
```

```

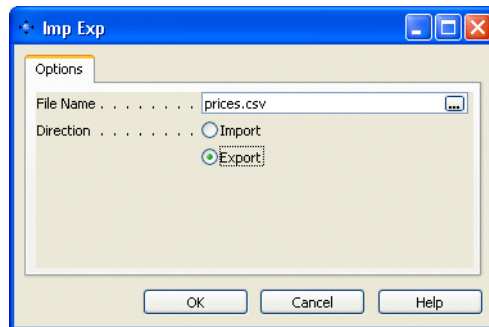
"Tariff Number".FIND;
END
ELSE
"Tariff Number".Description := 'NO TARIFF NUMBER';

```

This is enough to create the part of the dataport that exports the records.

12 Save and compile the dataport.

When you run the dataport, a request form appears and you can fill it out as follows:



When you click OK, the dataport is run and the records that match the criteria you have set up are written to a file called `prices.csv`.

When you open this file in Excel, it could look like this:

	A	B	C	D	E	F
1	1000	Bicycle	4,100	NO TARIFF NUMBER		
2	1001	Touring Bicycle	4,200	NO TARIFF NUMBER		
3	1100	Front Wheel	1,001	NO TARIFF NUMBER		
4	1110	Rim	50	NO TARIFF NUMBER		
5	1150	Front Hub	500	NO TARIFF NUMBER		
6	1200	Back Wheel	1,200	NO TARIFF NUMBER		
7	1250	Back Hub	1,100	NO TARIFF NUMBER		
8	1300	Chain Assy	800	NO TARIFF NUMBER		
9	1310	Chain	0	NO TARIFF NUMBER		
10	1700	Brake	600	NO TARIFF NUMBER		
11	1710	Hand rear wheel Brake	0	NO TARIFF NUMBER		
12	1896-S	ATHENS Desk	649.4	Desks		
13	1900-S	PARIS Guest Chair, bla	125.1	Other chairs, upholstered		
14	1906-S	ATHENS Mobile Pedest	281.4	Closets with door/drawers		
15	1908-S	LONDON Swivel Chair, t	123.3	Swivel chairs, upholstered		
16	1920-S	ANTWERP Conference	420.4	Other office furniture		
17	1924-W	CHAMONIX Base Stora	136.4	Other office furniture		
18	1928-S	AMSTERDAM Lamp	35.6	Desk lamps		
19	1928-W	ST.MORITZ Storage Uni	342.1	Other office furniture		
20	1936-S	BERLIN Guest Chair, ye	125.1	Other chairs, upholstered		
21	1952-W	OSLO Storage Unit/She	158.5	Other office furniture		
22	1960-S	ROME Guest Chair, gre	125.1	Other chairs, upholstered		
23	1964-S	TOKYO Guest Chair, bli	125.1	Other chairs, upholstered		
24	1964-W	INNSBRUCK Storage Ui	292	Closets with door/drawers		
25	1968-S	MEXICO Swivel Chair, b	123.3	Swivel chairs, upholstered		
26	1968-W	GRENOBLE Whiteboard	974.8	Other office furniture		
27	1972-S	MUNICH Swivel Chair, y	123.3	Swivel chairs, upholstered		
28	1972-W	SAPPORO Whiteboard,	974.8	Other office furniture		
29	1976-W	INNSBRUCK Storage Ui	256.1	Closets with door/drawers		

Now you should change the prices of some of the items and save the spreadsheet as `newprices.csv`. This is the file that you will import in the next section.

### Creating the Import Part

Now, you will create the part of the dataport that can import the new prices. This part is very easy to create. When you run the dataport, you decide whether it should export or import, and you select the file that it should write to or read from.

To specify that the imported records must update the existing records with the new prices:

- 1 Open the dataport in the Dataport Designer and open the **Properties** window of the data item.
- 2 Set the *AutoSave* property of the data item to Yes.
- 3 Set the *AutoUpdate* property of the data item to Yes.

The net effect of these settings is to update the existing records with the data that is different in the imported records, in this case, the **Unit Price**.

Before the dataport is used to import the converted data, the records could look like this:

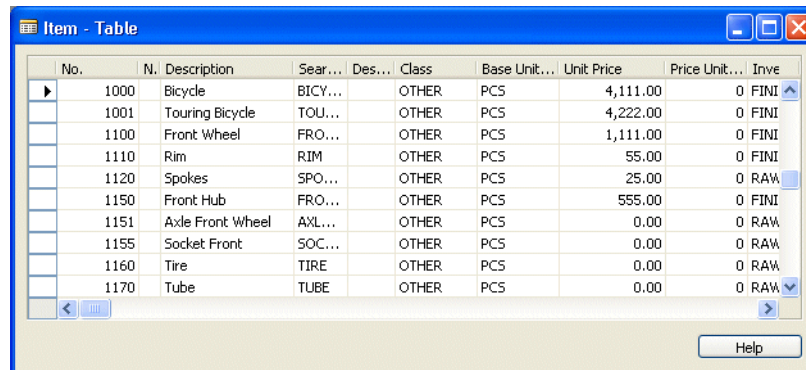
No.	N.	Description	Search...	Description	Class	Base Unit...	Unit Price	Price Unit...	Inve
1000		Bicycle	BICY...		OTHER	PCS	4,000.00	0	FINI
1001		Touring Bicycle	TOU...		OTHER	PCS	4,000.00	0	FINI
1100		Front Wheel	FRO...		OTHER	PCS	1,000.00	0	FINI
1110		Rim	RIM		OTHER	PCS	0.00	0	FINI
1120		Spokes	SPO...		OTHER	PCS	0.00	0	RAW
1150		Front Hub	FRO...		OTHER	PCS	500.00	0	FINI
1151		Axle Front Wheel	AXL...		OTHER	PCS	0.00	0	RAW
1155		Socket Front	SOC...		OTHER	PCS	0.00	0	RAW
1160		Tire	TIRE		OTHER	PCS	0.00	0	RAW
1170		Tube	TUBE		OTHER	PCS	0.00	0	RAW

The **Unit Price** field has been repositioned so that it is included in the screen shot.

To import the new data, run the dataport and fill out the request form as follows:

Use the AssistButton ... to browse to the `newprices.csv` file. Remember that the records are not actually imported if you run the dataport from inside the Dataport Designer. You must run it from, for example, the Object Designer.

After the records have been imported, the records in the *Item* table could look like this:



No.	N.	Description	Sear...	Des...	Class	Base Unit...	Unit Price	Price Unit...	Inve
1000		Bicycle	BICY...		OTHER	PCS	4,111.00	0	FINI
1001		Touring Bicycle	TOU...		OTHER	PCS	4,222.00	0	FINI
1100		Front Wheel	FRO...		OTHER	PCS	1,111.00	0	FINI
1110		Rim	RIM		OTHER	PCS	55.00	0	FINI
1120		Spokes	SPO...		OTHER	PCS	25.00	0	RAW
1150		Front Hub	FRO...		OTHER	PCS	555.00	0	FINI
1151		Axle Front Wheel	AXL...		OTHER	PCS	0.00	0	RAW
1155		Socket Front	SOC...		OTHER	PCS	0.00	0	RAW
1160		Tire	TIRE		OTHER	PCS	0.00	0	RAW
1170		Tube	TUBE		OTHER	PCS	0.00	0	RAW

Notice that the prices of the top six items have been changed.

### Further Work

This dataport has some shortcomings. For example, the price of each unit is used to calculate the **Profit %** field in the *Item* table when the **Unit Price** field is validated. You have not used the *CallFieldValidate* property to enforce that evaluation, but have left it at the default setting of No.

Getting the validation to work as intended is not so easy, because the code that is triggered uses values from other fields that are not part of this dataport. At the time of the validation, these fields do not contain any values because they are updated later. One way to solve this problem would be to export all the fields but this would make it much harder to manage in Excel.

This problem shows that you must give careful consideration to all the interdependent data when you update a table from a dataport. The solution to the problem will, however, be different for each table and for each set of fields that are imported.





**Part 8**  
**XMLports**



## **Chapter 21**

### **XMLports**

XMLports are conceptually related to dataports as they are object types that can import and export data. The difference is that the data is encapsulated in XML format. This makes it possible to exchange information between different computing systems in a streamlined way.

- XMLport Fundamentals
- Designing XMLports
- XMLport Examples
- Validating Data
- XMLports and Business Notifications

## 21.1 XMLport Fundamentals

XMLports enable you to import data received in XML format to the Dynamics NAV database and export data in XML format from the Dynamics NAV database. You only need a basic knowledge of XML to design and work with XMLports.

You design XMLports in the XMLport Designer, which you open from the Object Designer.

### Saving, Compiling and Running an XMLport

When you have designed an XMLport, you must save and compile it before you can use it. Normally, you do this when you have finished designing the XMLport. However, you may want to save an XMLport that is not yet finished and therefore cannot be compiled. You can also test-compile an XMLport without closing or saving it.

### Saving and Closing an XMLport

When you close the XMLport Designer, you close the XMLport.

To save an XMLport:

- 1 When you close an XMLport, C/SIDE will ask whether or not you want to save your changes. If it is a new XMLport (an XMLport that has not been saved before), and you choose to save, you will have to assign an ID and a name to the object. The ID must be unique and must follow the rules for numbering objects. Your Microsoft Business Solutions Partner will provide you with this information.

Hint: If you set up the ID and Name XMLport properties, their values will automatically be used, and you will therefore not be prompted for ID and Name information when you close the XMLport.

- 2 The option field **Compile** is by default set to True (displayed as a check mark). If the XMLport that you have designed is not yet ready to be compiled, remove the check mark by clicking the field.
- 3 Click OK to save the XMLport.

You can save an XMLport without closing it by choosing Save or Save As from the File menu. By using Save As, you can rename an existing XMLport. Note that you can copy an XMLport by opening and saving it with a new name.

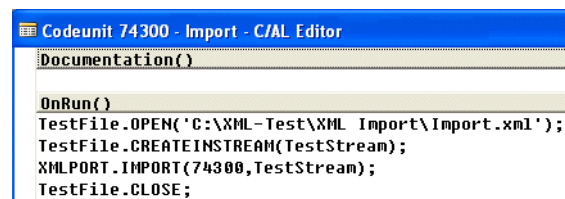
## Compiling an XMLport

XMLports, like other objects in C/SIDE, must be compiled before they can be run. As described previously, you can choose to compile an XMLport whenever you save it.

While you are designing an XMLport, you may want to test-compile it to find possible errors. You can test-compile an XMLport during design by choosing the Compile option in the Tools menu.

## Running an XMLport

XMLports are called from codeunits. While you are designing an XMLport, you will often want to run it before it has been integrated into an application to check that it functions as you intended. To do so, you can create a test-run codeunit that calls the XMLport and either streams data to or from a file depending on whether you are importing or exporting data. Here is an example of a test-run codeunit that calls an XMLport to import data from an XML file:



```
Codeunit 74300 - Import - C/AL Editor
Documentation()
OnRun()
TestFile.OPEN('C:\XML-Test\XML Import\Import.xml');
TestFile.CREATEINSTREAM(TestStream);
XMLPORT.IMPORT(74300,TestStream);
TestFile.CLOSE;
```

The four functions in the OnRun section perform the following:

- Open the Import.xml file.
- Create an InStream object so that the XML data can be streamed from the file.
- Load a specific XMLport object and give it the source from which it shall read and parse the incoming XML data stream.
- Close the Import.xml file.

For more information about streaming data to and from files or from automation components, see the *C/SIDE Reference Guide* online Help and the *Development Guide for Communication Components* online Help. The latter is available on the Dynamics NAV product DVD and is installed as part of the NAV Application Server.

## 21.2 Designing XMLports

An XML document contains XML tags, which identify the nature of the content that they contain. To create an XMLport to import the data in an XML document, you specify all the XML tag names and indicate the type of each, that is, whether it represents an element or an attribute. You then map these tag names to corresponding data structures (tables, records or fields) in the Dynamics NAV database. At runtime, when an XMLport object is called to handle an incoming XML document, it will read the incoming data stream and perform the processing and database actions.

When you want to create an XMLport to export data from the Dynamics NAV database in XML format, the XMLport Designer enables you to build the tag structure of the XML document and map the data. At runtime, the XMLport object will read the required data from the database, add the necessary XML tags to form the XML document and write the document to a data stream.

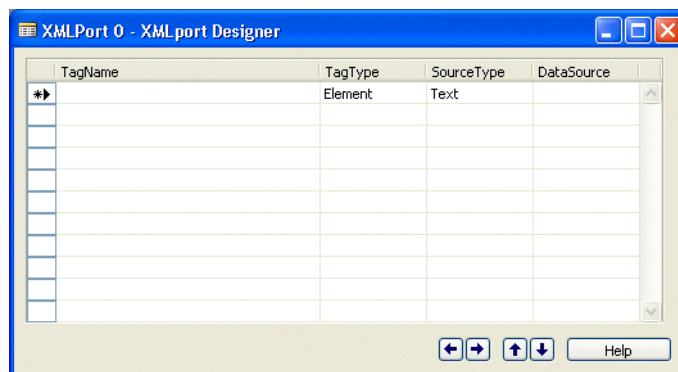
XMLports do not handle XML documents that:

- modify existing data in the database.
- find and delete data in the database.
- query the database for data (item catalog information, for example).

If you need to work with an incoming document of one of these types, you can do so using C/AL code and by carrying out the database manipulation necessary to achieve the desired result.

### The XMLport Designer

The XMLport Designer contains four column fields: **TagName**, **TagType**, **SourceType** and **DataSource**. The type of information that you enter in these column fields is:



#### TagName

In a **TagName** field, you enter the XML tag name of the XML element or attribute. You must enter tag names in the order in which they appear in the XML document. Parent elements must precede their child elements. You indent the tag names of child elements under their parent elements using one indentation per level. You list attributes under the elements that they define and you indent them to the child level.

### TagType

You use this field to specify whether the name in the **TagName** field represents data of the type element or attribute. The drop-down list in a **TagType** field contains two options: Element and Attribute. The default setting is Element.

### SourceType

You use this field to specify the data structure that the tag name corresponds to in the Dynamics NAV database. The **SourceType** field contains a drop-down list containing three options: Text, Table and Field. The default setting is Text.

**Text:** This is the option you select when the XML data cannot be mapped directly to the database or when the database does not need the information. The value of the **Text** field will be put into a text variable with the name you have specified in the **VariableName** property (otherwise the tag name will be used by default). The text variable acts like a normal global C/AL text variable. You can also turn the text source type into a big text variable by setting the **TextType** property to **BigText**.

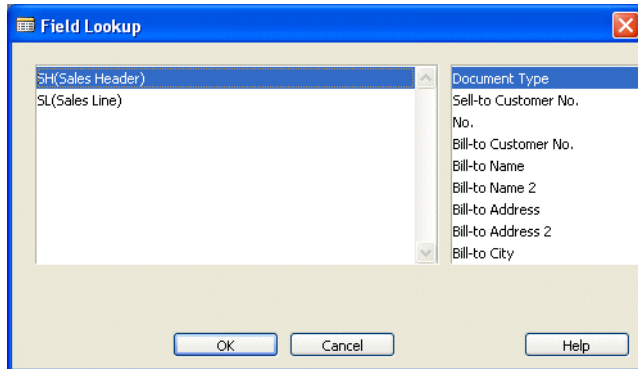
**Table:** You select this option to indicate that the tag name is equivalent to a table or that a table record must be initialized. As with the Text option, you can specify a variable name for the table, which also acts like a global record variable. By default, the variable name is the name of the table.

**Field:** You select this option to indicate that the tag name is equivalent to a field in the database. However, for this selection to be valid, you must first have declared a table as the parent of the field. Failure to do so will cause an error to occur when you try to compile the XMLport object.

A **DataSource** field has the following interactions with the **SourceType** field:

- If you have specified *Table* as the source type, clicking the **AssistButton** in the **DataSource** field will open the **Table List** window. You can also select a table by setting the **SourceTable** property. If you have defined a variable name for the table to be used, the format of the value shown in the **DataSource** field will be `tablevariablename(tablename)`. If you have not defined a variable name, the format of the value will be shown as `<tablename>(tablename)`.
- If you have specified *Field* as the source type, clicking the **AssistButton** in the **DataSource** field opens the **Field Lookup** window. Here you can select a field from one of the tables you have specified in the XMLport. The format of the value is shown as `tablevariablename::fieldname`.

In the following **Field Lookup** window, fields can be selected from the **Sales Header** and **Sales Line** tables:



- If you have specified *Text* as the source type, the text's variable name (or tag name if you have not specified a variable name) will be shown in the **DataSource** field.

### XMLport Properties, Functions and Triggers

There are a set of properties, functions and triggers that you can use to manipulate an XMLport. There are properties, functions and triggers for the object level, and for the element level, which consists of Field, Table and Text. The properties that are available at the element level depend on the selections you make in the **TagType** and **SourceType** fields in the XMLport Designer. See the online *C/SIDE Reference Guide* for descriptions of all the properties, functions and triggers for XMLports.



## 21.3 XMLport Examples

This section contains five examples of how you can use an XMLport to import data from an XML document into the Dynamics NAV database.

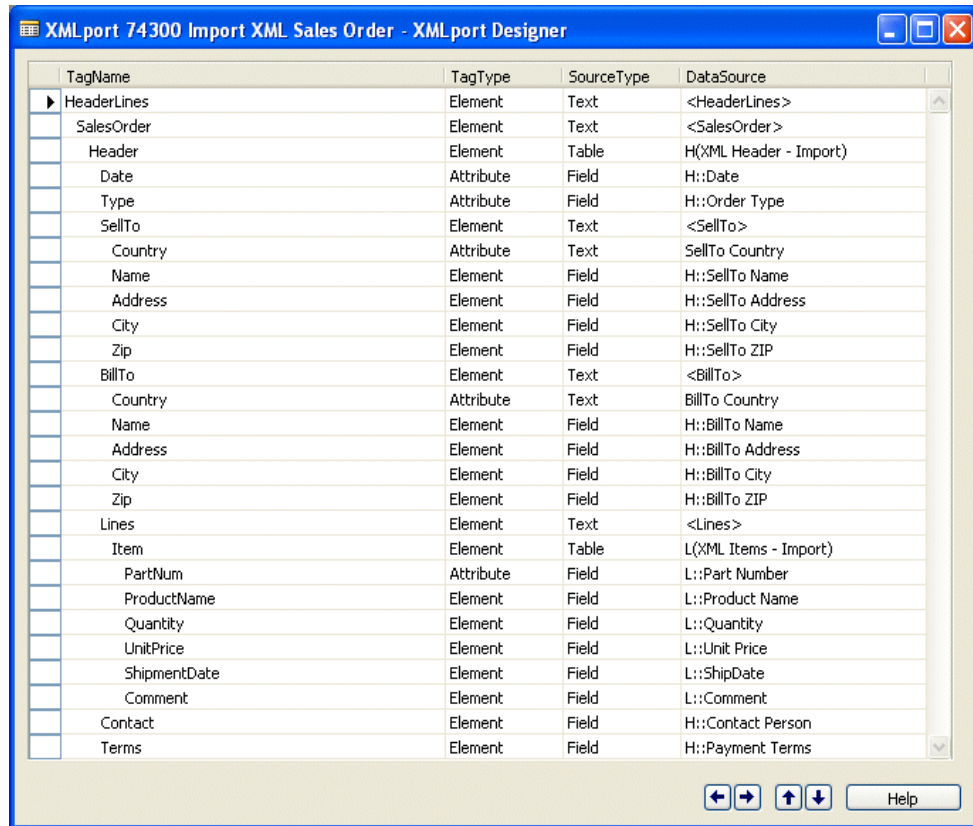
### Example 1

In this example you design an XMLport to import data from an XML sales order document, `XML Sales Order.xml`, to the Dynamics NAV database. The sales order is shown here:

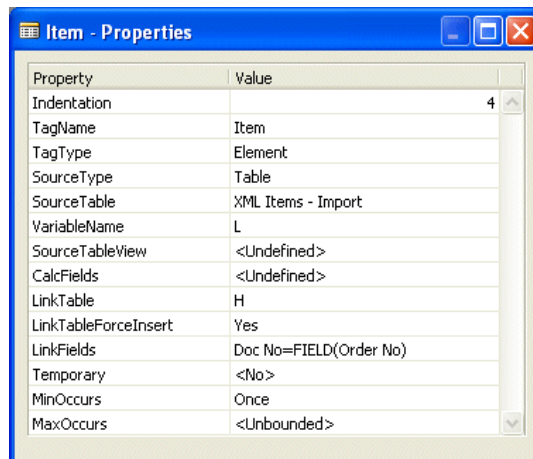
```
<?xml version="1.0" ?>
- <HeaderLines>
- <SalesOrder>
  - <Header Date="17-02-04" Type="Quote">
    - <SellTo Country="GB">
      <Name>The Canon Group PLC</Name>
      <Address>192 Market Square</Address>
      <City>Birmingham</City>
      <Zip>B27 4KT</Zip>
    </SellTo>
    - <BillTo Country="DK">
      <Name>The Cannon Group PLC</Name>
      <Address>Frydenlunds Alle</Address>
      <City>Vedbaek</City>
      <Zip>2950</Zip>
    </BillTo>
    - <Lines>
      - <Item PartNum="LS-150">
        <ProductName>Loudspeaker, Cherry, 150W</ProductName>
        <Quantity>8</Quantity>
        <UnitPrice>129,00</UnitPrice>
        <ShipmentDate />
        <Comment>Confirm the voltage is 75W</Comment>
      </Item>
      - <Item PartNum="LS-MAN-10">
        <ProductName>Manual for Loudspeakers</ProductName>
        <Quantity>20</Quantity>
        <UnitPrice />
        <ShipmentDate />
        <Comment />
      </Item>
      - <Item PartNum="LS-2">
        <ProductName>Cables for Loudspeakers</ProductName>
        <Quantity>10</Quantity>
        <UnitPrice>21,00</UnitPrice>
        <ShipmentDate />
        <Comment />
      </Item>
    </Lines>
    <Contact>Mr. Andy Teal</Contact>
    <Terms>14 days</Terms>
  </Header>
</SalesOrder>
```

After analyzing the XML document, you can see that the data belongs in two database tables; ***XML Header - Import*** and ***XML Items - Import***. We therefore design an XMLport that can insert the data into these tables.

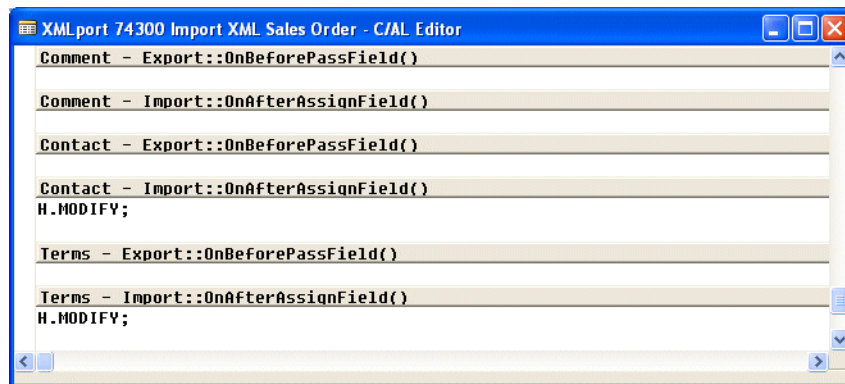
The XMLport has the following design:



The design of the XMLport shows that the data in the XML document has been mapped to database tables which have a header-line relation. The XMLport must insert the header information before inserting the line information. We have therefore set the `LinkedTableForcedInsert` property to `Yes` for the `<Item>` tag to ensure that this happens. The `LinkTable` and `LinkFields` properties for this tag have also been set to indicate the relationship between the **XML Header - Import** and **XML Items - Import** tables. It is important to specify this binding information to ensure that the XMLport enters the data into the correct tables. Note that you have given the **XML Items - Import** table the variable name "L" and that the **XML Header - Import** table has the variable name "H". The **Property** window for the `<Item>` tag looks as follows:



After all the records have been inserted into the *XML Items - Import* table, the **Contact Person** and **Payment Terms** fields in the *XML Header - Import* table have to be updated. We do this using the following C/AL code:



### Example 2

The following example shows how to handle a situation in which an XML document contains multiple data values that are represented by only one data structure in the Dynamics NAV database.

We have an *XML Sales Lines* table in the database that contains one field of the data type DateTime: the **Shipment** field. However, the XML sales order document from which you want to import data into the table contains two tags for this data, <ShipDate> and <ShipTime>, as shown in the following:

```
<?xml version="1.0" encoding="utf-8" ?>
- <SalesOrder>
- <OrderHeader OrderDate="17-09-04">
- <SellTo Country="GB">
  <Name>The Cannon Group PLC</Name>
  <Address>192 Market Square</Address>
  <City>Birmingham</City>
  <Zip>GB-CV6 1GY</Zip>
</SellTo>
- <BillTo Country="DK">
  <Name>Jens Ole</Name>
  <Address>Frydenlunds Alle 6</Address>
  <City>Vedbaek</City>
  <Zip>DK-2950</Zip>
  <PaymentTerms>14days</PaymentTerms>
</BillTo>
</OrderHeader>
- <Items>
- <Item PartNum="LS-75">
  <ProductName>Loudspeaker, Cherry, 75W</ProductName>
  <Quantity>10</Quantity>
  <UnitPrice>79</UnitPrice>
  <ShipDate>12-10-04</ShipDate>
  <ShipTime>09:50</ShipTime>
  <Comment>Confirm the voltage is 75w</Comment>
</Item>
- <Item PartNum="1908-S">
  <ProductName>LONDON Swivel Chair, blue</ProductName>
  <Quantity>12</Quantity>
  <UnitPrice>190,926</UnitPrice>
  <ShipDate>12-10-04</ShipDate>
  <ShipTime>09:50</ShipTime>
</Item>
</Items>
</SalesOrder>
```

One solution to the problem is to cache the values in the <ShipDate> and <ShipTime> tags, format the two values into a DateTime value, and assign the formatted value to the **Shipment** field in the

**XML Sales Lines** table. In the XMLport Designer, you would define the <ShipDate> and <ShipTime> tags as being of the Text source type:

TagName	TagType	SourceType	DataSource
ShipDate	Element	Text	<ShipDate>
ShipTime	Element	Text	<ShipTime>

In the OnAfterAssignVariable trigger for the <ShipTime> tag, you would write code that both formats the two text variables into one DateTime variable, and assigns the value held by that variable to the **Shipment** field of the **XML Sales Lines** table.

**Example 3**

In a business document, there is normally no need to state the same information more than once. In a relational database, however, certain information has to be repeated in various tables for data linking purposes. In the following XML sales order document, the <Items> tag has the attribute Type="Order":

```
<?xml version="1.0" ?>
- <SalesOrder>
  - <OrderHeader OrderDate="17-01-04">
    - <SellTo Country="GB">
      <Name>The Cannon Group PLC</Name>
      <Address>192 Market Square</Address>
      <City>Birmingham</City>
      <Zip>GB-CV6 1GY</Zip>
    </SellTo>
    - <BillTo Country="DK">
      <Name>Jens Ole</Name>
      <Address>Frydenlunds Alle 6</Address>
      <City>Vedbaek</City>
      <Zip>DK-2950</Zip>
      <PaymentTerms>14days</PaymentTerms>
    </BillTo>
  </OrderHeader>
  - <Items Type="Order">
    - <Item PartNum="LS-75">
      <ProductName>Loudspeaker, Cherry, 75W</ProductName>
      <Quantity>10</Quantity>
      <UnitPrice>79</UnitPrice>
      <Comment>Confirm the voltage is 75w</Comment>
    </Item>
    - <Item PartNum="1908-S">
      <ProductName>LONDON Swivel Chair, blue</ProductName>
      <Quantity>12</Quantity>
      <UnitPrice>190,926</UnitPrice>
      <ShipDate>12-02-04</ShipDate>
    </Item>
  </Items>
</SalesOrder>
```

The problem here is that the Type="Order" information has to be written to the **Document Type** field of the relevant table, in this example, the **XML Sales Lines** table. To cache the "Order" value, you would declare the Type attribute as a Text source type.

TagName	TagType	SourceType	DataSource
Type	Attribute	Text	Type

To assign the Type value to each record in the *XML Items - Import* table, you could write the following code in the Import::OnAfterInitRecord trigger for the <Item> tag:

```
Item - Import::OnAfterInitRecord()  
L.Document Type := Type;
```

#### Example 4

Sometimes an incoming XML document may not contain sufficient information to allow you to insert certain data as a record. To solve the problem, you can preassign the values of the fields for which you do not have data. You do so in the Import::OnAfterInitRecord and Import::OnBeforeInsertRecord triggers.

If, for example, you receive an XML sales order document that does not contain Line No. information, you can assign a value for it in the following way:

```
Item - Import::OnBeforeInsertRecord()  
IF not SL.Line No. then  
SL.Line := 1001;
```

#### Example 5

An XML document can be described as having a tree structure with many levels. A database table, however, can only store data in a flat structure — at the field level. A situation can therefore arise where an incoming XML document contains data in different scopes, whereas the data belongs in the same scope in the database.

For example, the <BillTo> tag of an XML sales order document could contain a <PaymentTerms> child element. The value of the <PaymentTerms> tag might sometimes need to be assigned to all sales items, and sometimes to only one of them. If you declare the <PaymentTerms> tag as being of the Element tag type and of the Text source type, the XMLport can keep the value alive and therefore when it starts to process the sales item information, you can specify which record(s) the value is to be assigned to.

## 21.4 Validating Data

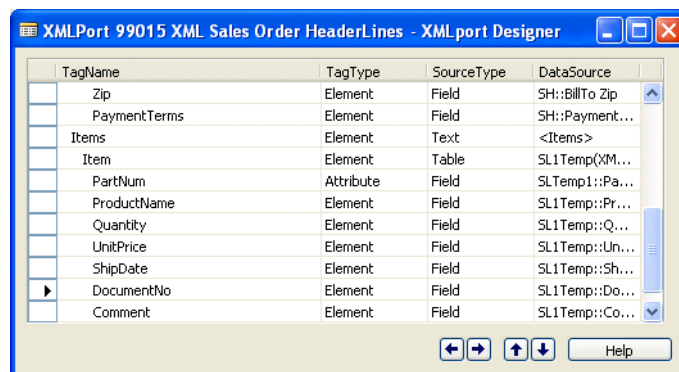
Data validation is essential to ensure the integrity of information used by the application. When an XMLport writes data to the Dynamics NAV database, it is therefore important that corrupt data is not inserted. To help prevent this, the default setting for the `DefaultFieldsValidation` property for an XMLport object is `Yes`. However, the insertion of data follows the order in which you have listed the XML tag names in the XMLport, not the validation order of the respective table. Therefore, if the insertion order violates the table's validation order, it is your responsibility to ensure that the data is not written directly to the table, or that you change the order of the XML tag names.

To avoid writing data directly to a table, you can apply a stylesheet to the XML document to ensure that the XML tag names are transferred in the correct order. Alternatively, you could design an XMLport that writes the data to a temporary table. You could then write code that inserts the data into the relevant table using the correct order. For example, an XML sales order document might contain a `<DocumentNo>` tag as shown in the following:

```
- <Items>
- <Item PartNum="LS-75">
  <ProductName>Loudspeaker, Cherry, 75W</ProductName>
  <Quantity>10</Quantity>
  <UnitPrice>79</UnitPrice>
  <Comment>Confirm the voltage is 75w</Comment>
</Item>
- <Item PartNum="1908-S">
  <ProductName>LONDON Swivel Chair, blue</ProductName>
  <Quantity>12</Quantity>
  <UnitPrice>190,926</UnitPrice>
  <ShipDate>12-05-04</ShipDate>
  <DocumentNo>9999</DocumentNo>
</Item>
</Items>
```

The value of this tag, the number of the document, needs to be written to the **XML Sales Header** table in the database. Before this can happen though, the value needs to be checked to ensure that it is valid. However, the XMLport design will not work because the position of the `<DocumentNo>` tag violates the validation order of the **XML Sales Lines** table.

To solve the problem, you could, for example, assign the XML tag names that were mapped to the **XML Sales Lines** table to a temporary table:



You could then write code that assigns the value of the `<DocumentNo>` tag to the correct table, and in the correct validation order, after the value has been assigned to a

temporary table record. This means that the <DocumentNo> value must be copied from the temporary record to the real record before any of the other fields are copied to the ***XML Sales Lines*** table.

By using temporary tables, you will be able to solve most of the cases where you can see that an incoming XML document contains tags in a sequence that would violate the database validation order.

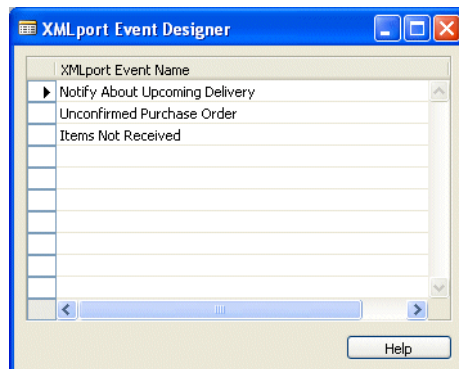
## 21.5 XMLports and Business Notifications

If you are running Microsoft Business Notifications with Dynamics NAV, you can associate XMLport events with an XMLport. Users who run Business Notifications can then subscribe to these events. For example, you could define the event "Sales Order Created" for a Sales Order XMLport. You will usually design special XMLports for the Business Notifications system.

XMLport events are called from codeunits to raise an event to Business Notifications. The XMLport then sends an XML document to the Business Notifications system. Based on the content of the XML document and the event raised, the Business Notifications system will send event information to subscribers. For example, in the case of the "Sales Order Created" event, it will check to see who has subscribed to this event and send them e-mail notification that a sales order has been created.

The events that you define can be more detailed than simply "Sales Order Created". For example, you can define an event that is raised when the total amount of a sales order exceeds a certain amount.

You define events in the XMLport Event Designer, which you open by selecting the XMLport Events option in the View menu when you are designing an XMLport. The following screenshot shows the XMLport Event Designer in which three events have been defined for a Purchase Order XMLport:



### Example

You want to raise an event that provides notification when a customer is blocked, that is, when Blocked changes from False to True. You therefore design an XMLport that contains the necessary data, for example, the ID and Name of the customer, and the date when the change occurred. You define an event called "Blocked" in the XMLport Event Designer and you declare the XMLport as a local variable called "CustXMLDoc". Finally, you add the following code to the OnModify trigger in the **Customer** table:

```
OnModify()
IF Blocked = TRUE THEN
  IF Blocked <> xRec.Blocked THEN
    CustXMLDoc.Blocked();
```

Refer to the Business Notifications online Help for further information.



**Part 9**  
**MenuSuite Objects**



## Chapter 22

### MenuSuite Objects

This chapter describes the MenuSuite object, which contains the menu suite content that is displayed in the Navigation Pane and in the Navigation Pane Designer.

The chapter contains information about the following subjects:

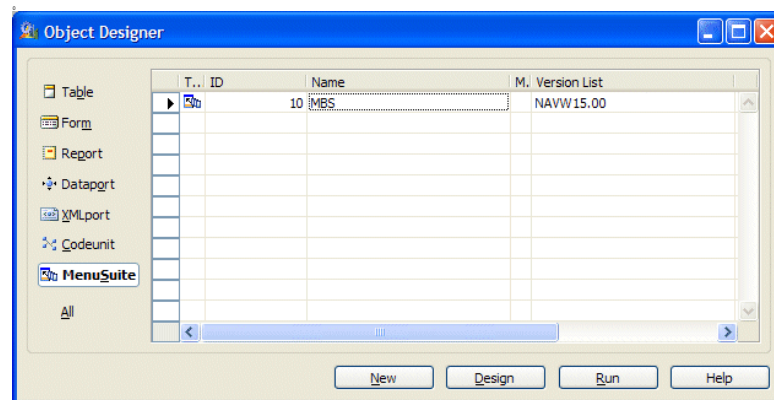
- Menu Suite Fundamentals
- Customizing a Menu Suite
- Exporting a MenuSuite Object
- Upgrading Menu Suite Content

## 22.1 Menu Suite Fundamentals

The MenuSuite object contains the main menu content that is displayed in the Navigation Pane and in the Navigation Pane Designer. A menu suite is a set of menus. Each menu contains content for a specific departmental area, for example, Finance or Manufacturing.

### Creating and Designing MenuSuite Objects

To create a new MenuSuite object or design an existing one, you click the MenuSuite button in the Object Designer, and then click New or Design.



**Clicking New** This opens a dialog asking you to specify which design level you want to create an object for. If you have already created a MenuSuite object for all the levels you have permission to, a message will appear informing you of this. Once you have made a selection, the Navigation Pane Designer opens. The MenuSuite object level that you are working on is shown in the header section of the Navigation Pane Designer.

**Selecting a MenuSuite object and then clicking Design** This opens the Navigation Pane Designer with the chosen menu suite content displayed. The MenuSuite object level that you have selected is shown in the header section of the Navigation Pane Designer

You also have the following options:

- You can select the File, Import/Export, and Tools, Translate, Import/Export options.
- You can compile MenuSuite objects (F11 or Tools, Compile). It is the object references that are compiled. If the MenuSuite object contains a reference to a non-existing form, report, batch job, dataport or codeunit, a compilation error occurs.

The Run button in the Object Designer is disabled when you click the MenuSuite button because a MenuSuite object cannot be run.

## 22.2 Customizing a Menu Suite

You customize a menu suite in the Navigation Pane Designer. In the following, you can read about the design options that are available for developers. You access design options by right-clicking on menu buttons, menu groups, menu items or anywhere in the content pane area.

Refer to the Dynamics NAV online Help for information about the design options for administrators, which can also be used by developers.

### Creating Menu Items

To create a menu item, you right-click in the content pane area or on a menu group or menu item, and select Create Item. The **Create Item** window opens in which you specify the menu item's object type, object ID, caption and captionML (multilanguage caption). An example is shown in the following:

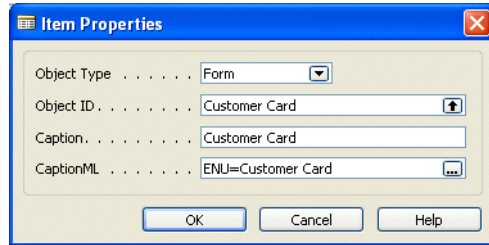
For example, if you have created a new report and want to add it as a menu item to a particular menu, then you select Report in the dropdown box in the **Object Type** field, and use the lookup in the **Object ID** field to select the report from the Report List. When you then click the **Caption** field, the name of the report is automatically filled in. You can rename it here if necessary.

The **CaptionML** field will by default show the code for the language that you use on your computer and the name of the menu item. If you have designed a menu item that will be used in different languages by different users, you must give the menu item a name in each language. When you click the AssistButton in the field, the **Multilanguage Editor** window opens where you can enter a menu item name for each language code:

Language	Value
Danish	Min nye rapport
English (United States)	My New Report

## Setting Properties

There are four properties that you can set for a menu item in the **Item Properties** window; Object Type, Object ID, Caption and CaptionML. You open this window by right-clicking a menu item and selecting Properties.



Property	Description
Object Type	In the <b>Object Type</b> field, you can make a selection in the dropdown box. The following object types are available: Table, Form, Report, Batch Job, Codeunit, Dataport.
Object ID	You use the lookup in the <b>Object ID</b> field to select a specific object from a list. For example, if you had selected Report as the Object Type, clicking the lookup will display the Report List.
Caption	By default, the <b>Caption</b> field contains the name that has been assigned to the menu item. You can rename it here if you want.
CaptionML	The <b>CaptionML</b> field contains the code for the language that you use on your computer and the name of the menu item, for example, ENU=General Journals. If the menu item will be used in different languages by different users, you must give the menu item a name in each language. When you click the AssistButton in the <b>CaptionML</b> field, a window opens where you can enter a menu item name for each language code.

## MenuSuite Object Levels

Microsoft Business Solutions provides a generic MenuSuite object, which we call the MBS level. This object is changed in various ways before end users see its content. These changes are applied at different levels, which are categorized as Country/Region, Add-on, Partner and Company. For example, the MenuSuite object is changed when the application undergoes localization changes, which takes place at the Country/Region level. If you are a developer working at a Microsoft Business Solutions partner, you customize a MenuSuite object at the Partner level. You can also configure a MenuSuite object at the Company level, which is the level that administrators work on.

Changes that are made to a MenuSuite object are stored as the differences between the previous MenuSuite object level and the current one. For example, when a company administrator configures a MenuSuite object at the Company level, the modifications are stored as the differences between the Company level and the Partner level, which was the previous level. If you export a MenuSuite object in text format and

then open the text file, you will see information about the changes that you have made seen in relation to the previous level.

With the exception of the Add-on MenuSuite object for which a maximum of 10 instances is allowed, there can only be one MenuSuite object for each level.

## 22.3 Exporting a MenuSuite Object

You can export a MenuSuite object in Dynamics NAV object format (\*.fob) or in text format. The text file contains information about the changes that have been made to the object since the previous level. In the following you can see an example of a text file that contains the changes that have been made to a MenuSuite object at the Company level.

### Example

When an administrator makes configuration changes to a menu suite, these changes are saved at the Company level. When this Company-level MenuSuite object is exported in text format, the text file contains information about the changes that have been made to the object since the previous level, which in this case would be the Partner level. Let us suppose that an administrator has made the following changes to his company's menu suite:

- In the Finance menu, he has deleted a menu group called Setup, which contained one menu item called Accounting Periods.
- He has added a new menu item, G/L Account Card, to the General Ledger menu group in the Finance menu.
- He has moved the G/L Account Card menu item so that it is placed between the Chart of Accounts and Bank Accounts menu items in the General Ledger menu group.

The text file would contain the following information:

```
OBJECT MenuSuite 70 Company
{
OBJECT-PROPERTIES
{
Date=19-04-04;
Time=16:04:10;
Modified=Yes;
Version List=;
}
PROPERTIES
{
}
MENUNODES
{
  { MenuItem ;[{FA8395A4-7A0B-4524-B0FD-20436D9711A4}] ;Name=G/L Account
    Card;
      CaptionML=ENU=G/L Account Card;
      MemberOfMenu=[ {F8D2429D-034B-4C58-9B5E-81BE962DB1BC} ] ;
      RunObjectType=Form;
      RunObjectID=17;
      ParentNodeID=[ {B12180CF-0EFB-43AD-9118-7765E953AAFD} ] ;
      Visible=Yes;
      NextNodeID=[ {7A5B6DDE-B44B-41A0-95DA-69B7109F0E32} ] }
  { ;[{1BB06483-AFFD-4750-BF62-3ABA035E11B7}];
    Deleted=Yes}
  { ;[{DF601C8A-07F7-4841-8929-9F2065BCB302}];
    Deleted=Yes}
  { ;[{8AC7917D-2C91-457D-80D6-A24B42F71AE7}];
    NextNodeID=[ {FA8395A4-7A0B-4524-B0FD-20436D9711A4} ] }
}
```



The content of a MenuSuite object can be described as having the following characteristics:

- It consists of a set of menus.
- A menu contains a collection of menu nodes, which are displayed in the Navigation Pane/Navigation Pane Designer in a tree structure.
- A menu node can be either a menu group or a menu item.
- A menu node has a globally unique identifier (GUID) and various properties.
- A menu group contains a collection of menu nodes.
- A menu item is the lowest level in the tree. When you click a menu item, its associated form, report, batch job, dataport or codeunit is run.

## 22.4 Upgrading Menu Suite Content

Menus in a menu suite that are inherited from the previous MenuSuite object level have the symbol ">>" on the menu button to the left of the menu name. These menus will be upgraded when you upgrade the corresponding MenuSuite object. Any changes that you have made to these menus will be merged into the new menu suite when you upgrade. For example, a menu item that you have added to a menu group – but where the menu group's contents now have changed after an upgrade – will be placed as the last menu item in the menu group.

However, there will be cases where a straightforward merge cannot be carried out. For example, if you have added a menu item to a menu group that is no longer a part of the menu after an upgrade, then no merge can be made. Instead, the menu item will be placed in a Lost Items group at the bottom of the menu tree. Another example would be if you had added one or more menu items to a menu that is no longer present in the menu suite after an upgrade. In such a situation, the menu items will be placed in a Lost Items menu.

After an upgrade, you can take action on the menu items in a Lost Items group or menu – either by inserting them somewhere in the current menu (in the case of the Lost Items group), in another menu or by deleting them. A Lost Items group or menu is not visible in the Navigation Pane at runtime.

New menus that you have created are not affected by an upgrade. If there are new menu items available after the upgrade that you would like to insert in these menus, you will have to add them manually.

**Part 10**  
**Multilanguage Functionality**



## **Chapter 23**

### **Multilanguage Functionality**

This chapter explains certain aspects of the multilanguage functionality of Dynamics NAV.

The chapter contains information about the following subjects:

- Multilanguage Functionality
- Learning the Code Base Language
- Number Ranges for Text Constants

## 23.1 Multilanguage Functionality

Everything to do with multilanguage functionality in C/SIDE in Dynamics NAV runs automatically. Note that to take advantage of this multilanguage functionality, you must upgrade your application to the multilanguage-enabled Dynamics NAV. For more information, see the manual *Upgrade Toolkit* on the Dynamics NAV product DVD.

### Defining the Current Application Language

C/SIDE executes the `ApplicationLanguage` function (trigger), with ID 4, on Codeunit 1 to determine the current language of the application. This trigger must return an integer (language ID). The trigger is not allowed to access the database. If the trigger does not contain a language code, C/SIDE reads the value from the `fin.stx` file, which contains general texts used by C/SIDE.

An algorithm has been built into C/SIDE to handle the hierarchy of languages that are available. This algorithm defines which language to show if one or more text strings are missing from the current application language. For more information, see the section "Displaying Text" on page 470.

For more information about language ID, see the section "The Windows Language Virtual Table" starting on page 469.

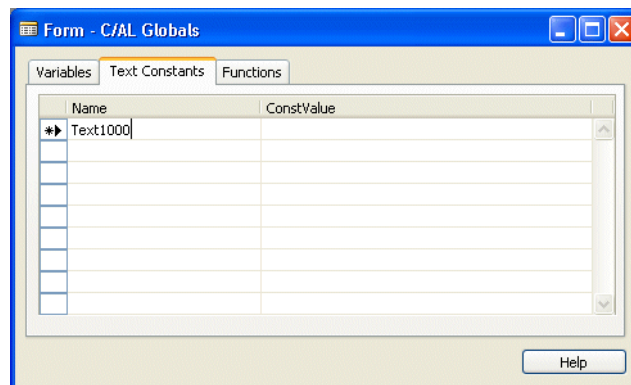
### Selecting a Language from the User Interface

In a multilanguage-enabled database, if the user click Tools, Language, the code generated by the `SetGlobalLanguage` trigger opens Form 534. In this **Application Languages** window, users can select the language in which captions in windows, on command buttons, and so on are displayed.

### Text Constants

The **C/AL Globals** and **C/AL Locals** windows have a **Text Constants** tab with a hidden column, **ConstValueML**, which displays all the languages for a text constant.

Text constants replace the use of hard-coded language dependent text strings.



For more information about the multilanguage use of text constants, see page 472, and for more information about creating text constants, see the section called "Defining Variables, Text Constants and Functions in Codeunits" on page 282.

## Language Modules

A language module contains the same information as the Translate Import/Export data files. However, a language module contains text for only one language layer. Language modules are binary files that you cannot modify with external tools.

You can import a language module by clicking Tools, Language Module, Import, and you can export one by clicking Tools, Language Module, Export.

## Installing \*.STX, \*.ETX, \*.CHM and \*.HH files for Multilanguage

You must install the \*.stx, \*.etx, \*.chm and \*.hh files for each language that the users will have access to in subdirectories. The name of a subdirectory must be the three-letter language code (Abbreviated Name) used by Windows for the particular language. For more information, see the section called "The Windows Language Virtual Table" on page 469.

If you create a subdirectory for a language and then install the \*.stx, \*.etx, \*.chm and \*.hh files while Dynamics NAV is running, the language will not be available until you restart the program.

## Adding a Language Layer

To let the user select a certain language from the Tools menu, that language must be represented as a granule in the license file.

The application must also be translated to that language, so that you can import it into the database by clicking Tools, Translate, Import. You can either export all the text strings and translate them in a translation tool, such as the Dynamics NAV Localization Workbench, or you can enter the translation of the text strings directly into the Multilanguage Editor.

To add a translation in the Multilanguage Editor:

- 1 Open the Object Designer and open the object that you want to add a translation to.
- 2 Open the **Properties** window (SHIFT+F4).
- 3 In the *CaptionML* property, click the AssistButton ... and the **Multilanguage Editor** window appears.
- 4 In the **Language** field, click the AssistButton ↑ and the **Windows Language List** window appears.
- 5 Select the language that you want to add from the list.

You can also simply enter the three-letter code for the language that you want to add and move the cursor to the **Value** field. The system automatically replaces the abbreviation with the full language description.

- 6 In the **Value** field, enter the correct name for this object in this language.
- 7 Click OK to save the information you have entered.

In order for the new language layer to work with the application, you must place the relevant `fin.stx` file in the subdirectory for that language. In other words, to allow the user to select a specific language from the Tools menu, the following must be true:

- The application must have the correctly named subfolder.
- The subfolder must contain the correct `fin.stx` file.
- The text strings in the database are marked with the correct language ID.
- The license file contains the correct granule.

### The Language Subfolder

Each language that the user will have access to must be represented by a subfolder in the Dynamics NAV directory structure.

Each language subfolder must contain the following:

- `fin.stx` file
- `fin.etx` file
- online Help files (`*.chm` and `*.hh`).

#### Note

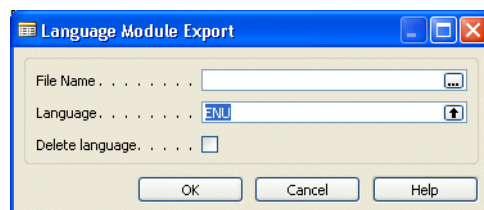
If you are installing a Swiss add-on to the application, and there is online Help for the add-on in German (Swiss) only, it must be installed in the DES subfolder. All Help files, such as `*.hh`, `*.chm` and `*.hlp` files, are placed in language-specific subfolders.

### Deleting a Language Layer

Once a language has been introduced into a database, there is only one way to delete it again.

To delete a language layer:

- 1 Click Tools, Language Module, Export and the **Language Module Export** window appears:



- 2 In the **Language** field, click the AssistButton  $\uparrow$  and select the unwanted language in the **Windows Language List** window and click OK.
- 3 In the **Language Module Export** window, place a check mark in the **Delete language** field and click OK.



## The Windows Language Virtual Table

The virtual, read-only **Windows Language** table displays the languages that Windows supports. You can view its contents by designing a tabular-type form based on the table.

The **Windows Language** virtual table contains the following fields:

Field Name	Description
<b>Language ID</b>	This field is the primary key. It displays the standard Windows language ID for a specific language. C/AL supports the setting of language using the GLOBALLANGUAGE, WINDOWSLANGUAGE and object. LANGUAGE properties. The values of these properties are taken from this field.
<b>Primary Language ID</b>	Windows languages are grouped. A group consists of a primary language and zero or more secondary languages. The <b>Primary Language ID</b> field contains the Windows Language ID of the primary language.
<b>Name</b>	This field contains the standard Windows name for the language.
<b>Abbreviated Name</b>	This field is a secondary key. It contains the standard Windows three-letter code for the language.
<b>Enabled</b>	A check mark indicates that the language is either globally enabled, form enabled, report enabled or dataport enabled. Your license file determines how a specific language can be used.
<b>Globally Enabled</b>	A check mark indicates that the license file allows you to set the language in question as the global language for the whole application.
<b>Form Enabled</b>	A check mark indicates that the license file allows forms to be shown in a language other than the global language.
<b>Report Enabled</b>	A check mark indicates that the license file allows reports to be printed in a language other than the global language.
<b>Dataport Enabled</b>	A check mark indicates that the license file allows dataports to be shown in a language other than the global language.
<b>Primary Code Page</b>	The code page for a language defines the character set available for that language. If you mix text by using multiple code pages, you may not obtain the expected result.
<b>STX File</b>	A check mark indicates that an *.stx file is installed for the language in question. An *.stx file contains general texts used by C/SIDE, for example, menu labels and system table names.
<b>ETX File</b>	A check mark indicates that an *.etx file is installed for the language in question. An *.etx file contains error messages.
<b>Help File</b>	A check mark indicates that an *.hlp or a *.chm file is installed for the language in question.

## Tab Controls

If you create a tab control without setting the *PageNames* property, C/SIDE will use the names 0, 1, 2, and so on as names for pages containing visible controls. Pages that do not contain controls or that do not contain visible controls are not displayed.

## Maintaining SQL Views

In the SQL Server Option for Dynamics NAV, you can set the option Maintain SQL Views. This setting determines whether SQL Server will create and maintain a view for each language ID that is added to a table or field in Dynamics NAV.

If you select this option, external tools are able to use these views to gain access to the *Caption ML* property of the object in the required languages rather than the name supplied in the table. For more information, see the section "Accessing Dynamics NAV Tables with External Tools" on page 82.

## NAV ODBC

NAV ODBC is multilanguage enabled. A NAV ODBC user can retrieve the application data from Dynamics NAV in different languages independent of the current Dynamics NAV application language.

NAV ODBC covers the following multilanguage features:

- Table name
- Field name
- OptionString value
- Date Formula

For more information, see the manual *Microsoft Dynamics NAV ODBC Driver 5.0 Guide*.

## Displaying Text

Whenever C/SIDE needs to display a text, it searches in the current language. If C/SIDE cannot find the text, it searches for the text in another language.

If, for example, the user wants to use German (Swiss) and the user wants to see a form that contains strings that do not exist with the language ID for German (Swiss), the algorithm will tell the system to look for a string with the language ID for German (Standard). This is because German (Standard) is the primary language for German (Swiss).

The algorithm that tells C/SIDE how to search for the right text uses the following order:

- 1 The object language
- 2 The primary language of the object language
- 3 The global language selected by the user
- 4 The primary language of the global language selected by the user
- 5 The application language

## 6 The primary language of the application language

**Multiple Document Languages**

You could run multiple document languages before you had a multilanguage-enabled database. But the multiple document languages functionality benefits from multilanguage because you now get the languages automatically.

If you have documents that you want to print in the language of the recipient rather than in your own working language, you can add a single line of code in the document to handle this. This functionality is already enabled for most printing reports in the standard Dynamics NAV database. The document is printed in the language that is specified in the **Language Code** field in the **Customer Card** window.

In reports that need the multiple document languages functionality, you must insert the following C/AL code as the first line in the OnAfterGet Record() trigger:

```
CurrReport.LANGUAGE := Language.GetLanguageID("Language Code")
```

Secondly, for each of these reports, you must create a new variable, Language, with the data type Record pointing to the **Language** table (table 8).

When you have compiled the object, it will no longer print in the user's working application language if another language has been specified in the **Customer Card** window.

## 23.2 Developing Multilanguage-Enabled Applications

When you are developing in a multilanguage-enabled environment, it is important to remember the following three rules of thumb:

- Everything has a *Name* property in English (United States).
- Text constants replace text strings such as error messages.
- Everything that the user will see must have a *Caption* property.

Before you start working in a multilanguage-enabled database, you should set the application language as English (United States). You do this by clicking Tools, Languages and selecting English (United States).

### Name Property

In Dynamics NAV, the code base is English (United States). This means that the *Name* property of, for example, an object must always be in English (United States).

The code base in English (United States) includes, among other things, the following:

- Object names
- Field names
- Function and variable names
- Comments
- Option strings
- Control names

### Text Constants

Error messages and other text strings must be entered as text constants so that they can be easily translated.

Text constants are automatically assigned unique IDs by C/SIDE. You can see the ID for a text constant by opening the **C/AL Globals** window, selecting a text constant and opening its **Properties** window.

In a single-language database, you can code error messages as text strings directly in the code. In the new multilanguage-enabled database, this must now be entered as:

```
IF FileName = '' THEN
    ERROR(Text000);
```

In this example, *Text000* is an available name for a text constant in that object. The text constants must then be created in the **C/AL Globals** window. For more information about creating text constants, see the section called "Accessing Dynamics NAV Tables with External Tools" on page 82.

When you add new text constants to existing objects, you can name them according to your needs. C/SIDE assigns unique IDs according to the number ranges listed in section "Number Ranges for Text Constants" on page 479, which makes it easier for you to upgrade customized objects.

When you are working in the C/AL Editor and place the cursor on a text constant, the content of the text constant will be shown in the message line in the language you have chosen as the application language. For more information, see the section called "C/AL Scanner" on page 477.

### Caption Property

Everything that is displayed to the user must have a *Caption* property. The *Name* property is always English (United States), so the *Caption* property is used to show the user the name in their own language.

For example, if you want to call on a field from the code in connection with an error message, you will call it by its *Name* property but make sure that the *Caption* property is displayed:

```
VATPostingSetup.FIELDSCAPTION("VAT Calculation Type")
```

where `VATPostingSetup` is the *Name* property in English (United States), and `FIELDSCAPTION` makes sure that the *Caption* property in the relevant language is used rather than the *Name* property.

When you are programming, you should always remember the difference between the *Name* property and the *Caption* property to ensure that you get the expected result when running the code.

C/SIDE can help you in a number of ways as described in the section "Learning the Code Base Language" on page 476.

### CaptionML Property

The *CaptionML* property makes it possible to change languages. Everything must have a *CaptionML* property where the value is set to the correct term in English (United States). This value is followed by whatever translations there may be of that object or field. The *Caption* property copies the value for the current application language from the *CaptionML* property.

#### Example

Table 37, Field 1 has the following *CaptionML* values:

```
ENU=Document Type;FRC=Type document
```

In the **CaptionML Value** field, you can either enter the value for English (United States) directly, or you can click the AssistButton ... to open the Multilanguage Editor and enter the value there.

If you are creating a new field, you must enter the value for English (United States) in the **Name** field to get started.

You must click OK to save the information when you exit the **Multilanguage Editor** window.

**Note**

.....

If you have created a new field in a form, the content of the *Caption* property will not be shown on the form until the form has been compiled. If you have copied another field on that form and modified the properties, the content of the *Caption* property will be shown on the form even in the Form Designer. This rule includes request forms for reports.

But if you enter the caption directly in the **Value** field, you will not have to compile the form to see the *Caption* property.

.....

**Creating Captions**

If your application does not have *Caption* properties for everything that is translated, you must insert these properties. You can do this manually or use the `make-ml` tool. For more information about this tool, see the manual *Upgrade Toolkit* for Dynamics NAV.

You should pay special attention to the following:

- Option buttons
- Option strings
- Option variables

**Option Buttons**

For options buttons, you must make sure that the *CaptionML* property is correct.

**Note**

.....

The value in the **OptionValue** field are always in English, because this value is used by the corresponding global variable, and code must always be in English (United States). You must make sure that the value for English (United States) in the **CaptionML** field is the same as in the **OptionValue** field.

.....

**Option Strings**

For option strings, for example, a control in a request form, you must make sure that the *OptionsCaptionML* property is correct.

**Note**

.....

The *Name* property must remain the number of the control, for example, *Control 9*.

.....

**Option Variables**

For option variables, for example, in a source expression for a `FORMAT`, `STRSUBSTNO`, `ERROR`, `MESSAGE`, or `CONFIRM`, you must insert a `SELECTSTR` string to select an option from a text constant. You should then let the text constant contain the options from the option string.

## Date Formulas

When you are creating a field in a table and you want this field to contain a date formula, you must apply the `DateFormula` data type to the field. This data type is non-language dependent and gives the `CALCDATE` function multilanguage capabilities.

You achieve a similar result if you apply the `Code` or `Text` data type to the field and then set the `Dateformula` property to `Yes`. However, this solution makes the data language-dependent, which means that users with different application languages cannot use the same data.

## Usage in C/AL Code

When you use the `CALCDATE` function to calculate dates, you must enter the date formula in English (United States) but with angle brackets (< >) around the date formula. Date formulas are translated but if you place angle brackets around them, the code will be valid regardless of the application language. In this way, the calculation will be the same no matter which application language the user has selected.

### Example

```
EndOfMonth := CALCDATE(' <CM>', TODAY);
```

## 23.3 Learning the Code Base Language

If you are not used to working with English as the code base language, C/SIDE can help you get used to working in the new environment in a number of ways.

### Generating a Dictionary

You may want to generate a translation of the variables so that you can compare the English name for the variable to the name in your local language.

You can use the Dynamics NAV Localization Workbench (NLW) to generate this translation. Use NLW to create a project based on the translation your local Microsoft Certified Business Solutions Partner created for the previous version of Dynamics NAV, and export the relevant fields to a comma-separated file. You can then open this file in, for example, Microsoft Excel and use it as a dictionary.

The relevant fields are:

- Source Text
- Target Text
- SourceResourceID

#### Note

You must first set a filter to only show variables before you export the file from NLW.

For more information about the Dynamics NAV Localization Workbench, see the manual *Dynamics NAV Localization Workbench User's Guide*, which is included in the `NLW.cab` file.

### How to See Both Captions and Names

In a number of different C/SIDE windows, you can see both the *Name* property and the *Caption* property of the selected item, as described in the following sections.

### Zoom Functionality

When you use the Zoom functionality on an object, you can choose to see both the local language and English (United States) for the fields.

To see both the local language and English (United States):

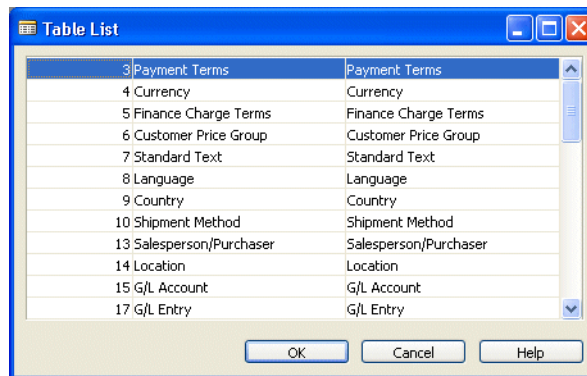
- 1 Open the object, for example, a purchase order, and place the cursor in the relevant field.
- 2 Click Tools, Zoom (CTRL+F8) to zoom in.
- 3 In the **Zoom** window, right-click one of the column headers and select Show Columns from the list.
- 4 In the **Show Columns** window, place a check mark next to **Field Name** and click OK.



The **Field** column will show the caption values for the current application language and the **Field Name** column will show the name properties.

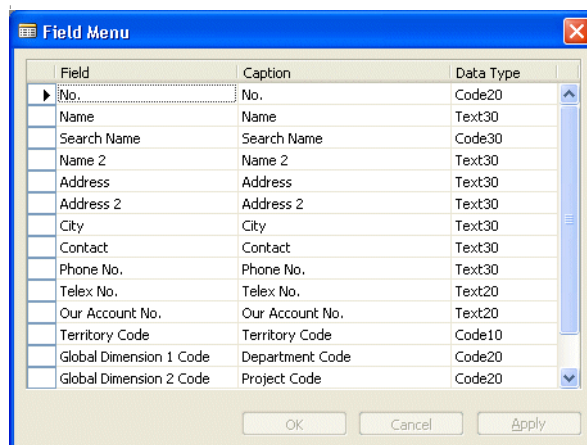
### Table List, Form List, Field List, Object List and Field Menu

In the **Object Designer**, **Table List**, **Form List**, **Field List**, **Object List** and **Field Menu** windows, you can see both the *Name* property and the *Caption* property for the items on the list:



The first column from the left contains the object number, the second column contains the *Name* property, and the third column contains the *Caption* property in the current application language.

In the **Object Designer** and **Field Menu** windows, you can hide the **Caption** column by right-clicking the **Caption** column header and selecting Hide Column from the list.



You can show the column again by following the procedure described in the section "Zoom Functionality".

For more information about the use of captions, see page 473.

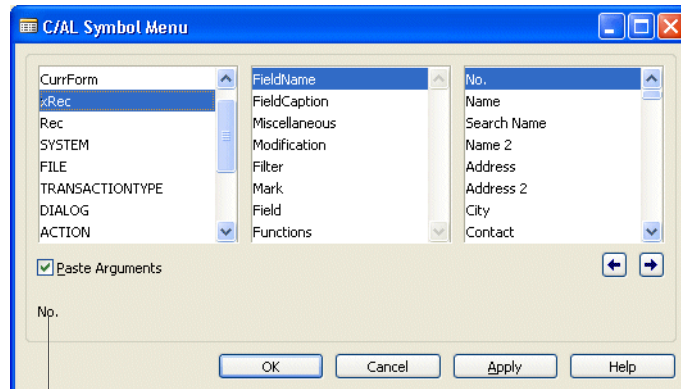
### C/AL Scanner

In the C/AL Editor, a scanner can show you captions for objects, fields and text constants. Since the code base should be in English, the scanner can help you read the code correctly.

When you place the cursor on an object, field or text constant, the C/AL scanner will look for the caption property in the current application language for the object, field or text constant. The scanner then displays this information in the status bar at the bottom of the Dynamics NAV window.

### C/AL Symbol Menu

A new subcategory has been added to the **C/AL Symbol Menu** window: *FieldCaption*. You can see this when you have selected a variable that relates to a table record:



This is the *Caption* property of the field whose *Name* property you have selected

If you have selected a field name, you can see the caption for that field in the current application language in the bottom left-hand corner of the window. In other words, in the third column in the **C/AL Symbol Menu** window, you can see the *Name* property, and in the bottom left-hand corner of the window, you can see the *Caption* property. In the picture, the current application language is English (United States) so the two properties are the same.

If you have selected a caption, you can see the corresponding field name for that caption in English (United States) in the bottom left-hand corner of the window. In other words, in the third column, you can see the *Caption* property, and in the bottom left-hand corner of the window, you can see the *Name* property.

For more information about the **C/AL Symbol Menu** window, see the section called "Using the C/AL Symbol Menu" on page 287.

## 23.4 Number Ranges for Text Constants

C/SIDE assigns unique IDs to text constants according to the following table of number ranges:

<b>Developer</b>	<b>From</b>	<b>To</b>
Microsoft Business Solutions HQ	000	9,999
Microsoft Business Solutions Netherlands	1,000,000	1,009,999
Microsoft Business Solutions Belgium	1,010,000	1,019,999
Microsoft Business Solutions USA	1,020,000	1,029,999
Microsoft Business Solutions Canada	1,030,000	1,039,999
Microsoft Business Solutions United Kingdom	1,040,000	1,049,999
Microsoft Business Solutions Iceland	1,050,000	1,059,999
Microsoft Business Solutions Denmark	1,060,000	1,069,999
Microsoft Business Solutions Sweden	1,070,000	1,079,999
Microsoft Business Solutions Norway	1,080,000	1,089,999
Microsoft Business Solutions Finland	1,090,000	1,099,999
Microsoft Business Solutions Spain	1,100,000	1,109,999
Microsoft Business Solutions Portugal	1,110,000	1,119,999
Microsoft Business Solutions France	1,120,000	1,129,999
Microsoft Business Solutions Italy	1,130,000	1,139,999
Microsoft Business Solutions Germany	1,140,000	1,149,999
Microsoft Business Solutions Switzerland	1,150,000	1,159,999
Microsoft Business Solutions Austria	1,160,000	1,169,999
Microsoft Business Solutions Poland	1,170,000	1,179,999
Microsoft Business Solutions Lithuania	1,180,000	1,189,999
Microsoft Business Solutions Latvia	1,190,000	1,199,999
Microsoft Business Solutions Estonia	1,200,000	1,209,999
Microsoft Business Solutions Russia	1,210,000	1,219,999
Microsoft Business Solutions Czech Republic	1,220,000	1,229,999
Microsoft Business Solutions Slovenia	1,230,000	1,239,999
Microsoft Business Solutions Australia	1,240,000	1,249,999
Microsoft Business Solutions New Zealand	1,250,000	1,259,999
Microsoft Business Solutions Singapore	1,260,000	1,269,999
Microsoft Business Solutions South Africa	1,270,000	1,279,999
Microsoft Business Solutions India	1,280,000	1,289,999
Microsoft Business Solutions Argentina	1,290,000	1,299,999

<b>Developer</b>	<b>From</b>	<b>To</b>
Microsoft Business Solutions Brazil	1,300,000	1,309,999
Microsoft Business Solutions Mexico	1,310,000	1,319,999
Microsoft Business Solutions Croatia	1,320,000	1,329,999
Microsoft Business Solutions North Africa/Middle East	1,330,000	1,339,999
Microsoft Business Solutions Thailand	1,340,000	1,349,999
Microsoft Business Solutions Malaysia	1,350,000	1,359,999
Microsoft Business Solutions Hungary	1,360,000	1,369,999
Microsoft Business Solutions Ireland	1,370,000	1,379,999
General customer modifications	1,000,000,000	1,000,999,999
Add-on	1,100,000,000	1,199,999,999

If you are converting an object with general customer modifications using the *conv-ml* tool, specify a random number between 1,000,000,000 and 1,000,999,999 as the start number of the number range.

**Part 11**  
**Beyond the Basics**



## **Chapter 24**

### **SumIndexFields**

This chapter describes SumIndexFields™, which are the basis for the FlowFields in a C/SIDE application. This chapter describes how SIFT™ works on C/SIDE Database Server as well as some details of the way that SIFT is implemented in the SQL Server Option for Dynamics NAV. This information will help programmers develop efficient applications that use SumIndexField Technology.

- SumIndexFields
- SIFT and the SQL Server Option for Dynamics NAV

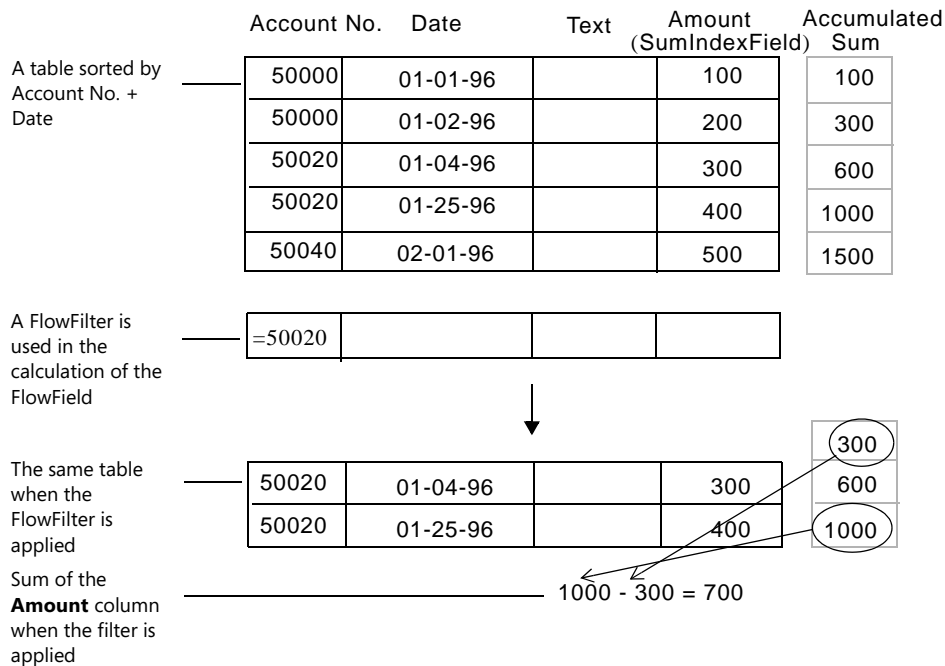
## 24.1 SumIndexFields

SumIndexField Technology (SIFT™) has been designed to improve performance when carrying out such activities as calculating customer balances. In traditional database systems this involves performing a series of database calls and calculations before arriving at a result. The power and efficiency of SIFT on C/SIDE Database Server makes calculating sums for numeric columns in tables extremely fast, even in tables that contain thousands of records. This powerful feature is used throughout the Dynamics NAV application and has also been implemented in the SQL Server Option.

### SIFT and C/SIDE Database Server

A SumIndexField is a fundamental feature which is the basis of FlowFields. A SumIndexField is associated with a key; each key can have at most 20 SumIndexFields. During database design, a field of the decimal type can be associated with a key as a SumIndexField. This tells the DBMS to create and maintain a structure which contains the accumulated sum of values in a column. When a new current key is selected, any SumIndexField associated with it becomes accessible.

The following figure illustrates a table where the **Amount** field (column) is defined as a SumIndexField in the Account No + Date key. This enables the DBMS to automatically maintain the accumulated sum of the column. Every time a change is made to a field in the column, the accumulated values are updated.



To the right of the table is shown an area in the database where the accumulated sums for the **Amount** column are kept. In the previous figure, the third field in the column, holding the accumulated sum, contains the value 600 because the first three Amount values are 100, 200 and 300, respectively – a total of 600. The fourth virtual field contains 1000, the total of the first four values in the **Amount** column, and so on. If the table contained a second SumIndexField, its values would be accumulated in the same way.



What advantages do SumIndexFields offer? Sums (of columns) can be quickly calculated and the result displayed in FlowFields. Let us say you want the sum of all the values in the Amount fields. In a conventional system, the DBMS is forced to access every record and add each value in the Amount field, a very time-consuming operation in a database with thousands of records. Here, you would create a FlowField, define the calculation formula of this FlowField to sum the Amount field, and the DBMS only needs retrieve the value from the SumIndexField.

Operations with SumIndexFields are as fast when FlowFilter fields are applied. The second table in the previous figure shows a group of records selected by a FlowFilter field in the **Account No.** field. Two records fulfil the conditions of the calculation filter. Only two accesses are needed to sum the Amount for these records: one access to get the accumulated sum associated with the last record before the specified range, and one access to get the accumulated sum associated with the last record in the specified range.

The value 300 is subtracted from the value 1000 to produce the correct sum (700). No matter how many records there are in the selected range, the system will always need to perform only two accesses in order to compute the sum.

SIFT has been built into the index structure used on C/SIDE Database Server and the more SumIndexFields that are added the larger the index becomes. However, the time used to maintain the accumulated sum for SumIndexFields is negligible due to a special index structure used in the DBMS.

## 24.2 SIFT and the SQL Server Option for Dynamics NAV

As mentioned earlier, SIFT has also been implemented in the SQL Server Option for Dynamics NAV. This section describes in some detail the way that SIFT is implemented in the SQL Server Option.

### SIFT Components

A SumIndexField is always associated with a key and each key can have a maximum of 20 SumIndexFields associated with it. In this document we will refer to a key that has at least one SumIndexField associated with it as a SIFT key.

When you set the *MaintainSIFTIndex* property of a key to *Yes* Dynamics NAV will regard this key as a SIFT key and create all the SIFT structures that are needed to support it. However, disabling the SIFT key by setting the *MaintainSIFTIndex* property to *No* can improve performance in certain circumstances. Setting this property to *No* means that the SIFT functionality is implemented by calculating the totals online instead of using the precalculated sums that are maintained by SIFT.

Any field of the *Decimal* data type can be associated with a key as a SumIndexField. Dynamics NAV then creates and maintains a structure that stores the calculated totals that are required for the fast calculation of aggregated totals.

In the SQL Server Option for Dynamics NAV this maintained structure is a normal table (a SIFT table). The layout of a SIFT table is described later in this article. As soon as the first SIFT table is created for a base table, a dedicated SQL Server trigger is also created and is then automatically maintained by Dynamics NAV. This is known as a SIFT trigger. A base table is a standard Dynamics NAV table, as opposed to an extra SQL Server table that is created to support Dynamics NAV functionality.

One SIFT trigger is created for each base table that contains SumIndexFields. This dedicated SQL Server trigger supports all the SIFT tables that are created to support this base table. The purpose of the SIFT trigger is to implement all the modifications that are made on the base table in every SIFT table that is affected. This means that the SIFT trigger automatically updates the information in all the existing SIFT tables after every modification of the records in the base table.

### SIFT and Cache

If you ask Dynamics NAV to calculate a total (*CALCSUMS*), SIFT will calculate all the totals for all the SumIndexFields that are related to that key in the base table. You will receive the total you requested and all the aggregations will be stored in cache. These totals can be reread from the cache to answer any subsequent requests provided that the cache is still valid. SIFT will do this without issuing any statement to SQL Server.

### Naming Conventions

This section describes the naming conventions that are used when generating the SIFT components on the SQL Server Option for Dynamics NAV.

## SIFT Triggers

The body of the SIFT trigger is generated by Dynamics NAV and is maintained automatically so that it reflects every change that is made to the design of the base table as well as its fields, keys and SumIndexFields.

The name of the SIFT trigger has the following format:

*<base Table Name>\_TG.*

For example, the SIFT trigger for table 17, **G/L Entry** is named:

*CRONUS International Ltd\_\$G/L Entry\_TG.*

## SIFT Tables

A SIFT table is a SQL Server table that is created and maintained automatically by Dynamics NAV and used to store precalculated totals based on values that are stored in SumIndexFields in base tables. A SIFT table is created for every base table key that has at least one SumIndexField associated with it. No matter how many SumIndexFields are associated with a key, only one SIFT table is created for that key.

The name of the SIFT table has the following format:

*<Company Name>\$<base Table ID>\$<Key Index>.*

For example, one of the SIFT tables created for table 17, **G/L Entry** is named:

*CRONUS International Ltd\_\$17\$0.*

The Key Index is a calculated integer value starting from 0. This means that the first SIFT key in the base table is given the value 0, the next is 1 and so on. These values are updated if any changes are made to the base table.

For example, table 17, **G/L Entry** has the following key layout:

Enabled	Key	SumIndexFields	MaintainSIFTIndex
YES	Entry No.		YES
YES	G/L Account No., Posting Date	Amount, Debit Amount, Credit Amount, Additional-Currency Amount, Add.-Currency Debit Amount, Add.-Currency Credit Amount	YES
YES	G/L Account No., Business Unit Code, Global Dimension 1 Code, Global Dimension 2 Code, Close Income Statement Dim. ID, Posting Date	Amount, Debit Amount, Credit Amount, Additional-Currency Amount, Add.-Currency Debit Amount, Add.-Currency Credit Amount	YES
YES	Document No., Posting Date		YES
YES	Transaction No.		YES
YES	Close Income Statement Dim. ID		YES

This table has two SIFT keys because only two keys have SumIndexFields associated with them.

The SIFT key that is composed of the **G/L Account No., Posting Date** fields has the Key Index value 0. Therefore, the SIFT table with the name *CRONUS International Ltd\_\$17\$0* is created for it on SQL Server.

The SIFT key that is composed of the **G/L Account No., Business Unit Code, Global Dimension 1 Code, Global Dimension 2 Code, Close Income Statement Dim. ID, Posting Date** fields has the Key Index value 1. Therefore, the SIFT table with the name *CRONUS International Ltd\_\$17\$1* is created for it on SQL Server.

The column layout of the SIFT tables is based on the layout of the SIFT key along with the SumIndexFields that are associated with this SIFT key. But the first column in every SIFT table is always named "bucket" and contains the value of the bucket or SIFT level for the precalculated sums that are stored in the table. Buckets are discussed in the following section.

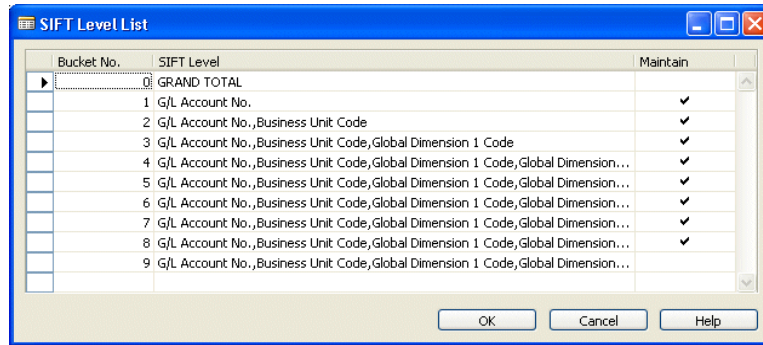
After the bucket column, comes a set of columns with names that start with the letter "f". These are also known as f- or key-columns. Each of these columns represents one field of the SIFT key. The name of these columns has the following format: *f<Field No.>*, where Field No. is the integer value of the *Field No.* property of the represented SIFT key field. For example, column *f3* in *CRONUS International Ltd\_\$17\$1* represents the **G/L Account No.** field (it is field number 3 in the base table **G/L Entry**) of the SIFT key with Key Index = 1 (see the previous example).

And finally, there is a group of columns with names that start with the letter "s" followed by numbers. These are also known as s-columns. These columns represent every SumIndexField that is associated with the SIFT key. The name of these columns has the following format: *s<Field No.>*. Field No. is the integer value of the *Field No.* property of the represented SumIndexField. The precalculated totals of values for the corresponding SumIndexFields are stored in these fields of the SIFT table. For example, the first s-column in *CRONUS International Ltd\_\$17\$1* is *s17*. This column represents the **Amount** SumIndexField (it is field number 17 in the **G/L Entry** table) because the **Amount** field is associated with the SIFT key.

## Buckets and SIFT Levels

Understanding the relationship between buckets and SIFT levels is crucial to understanding the way that SIFT is implemented in the SQL Server Option for Dynamics NAV. The precalculated totals or sums for each SumIndexField column are stored in buckets in SIFT tables. The buckets correspond to the SIFT levels that are maintained and each SIFT level can generate many records that are stored with the same bucket number in the SIFT tables on SQL Server. The higher the bucket number the more detailed the SIFT level. The buckets and their corresponding SIFT levels can also be seen

from within Dynamics NAV, even though they only exist in the SQL Server tables that are created to support SIFT:



The precalculated totals from the different buckets are retrieved and aggregated to generate the sums or totals that are stored in the SumIndexFields. For information about how to open this window, see "Configuring the SIFT Levels" on page 500.

**What are SIFT Levels?**

As mentioned earlier, every row in a SIFT table stores precalculated totals in s-fields. These totals are based on the values in the corresponding SumIndexField column in the base table. The f-fields in each record in a SIFT table contain the conditions which are constant for every row in the base table, and which contribute to the value of the total that is stored in that record in the SIFT table. In other words, a SIFT level is the set of values that are stored in the key fields that are used to generate the stored total of the SumIndexField values. A SIFT level or bucket can be regarded as a hash value or a key value that uniquely specifies the totals that are stored in it. A bucket is similar to the concept of a cube that is used in OLAP systems.

Every bucket in a SIFT table has a bucket number that corresponds to its SIFT level. The SIFT level's bucket number is stored in the bucket field of each record in the SIFT table. Also, records in SIFT tables are sorted according to their bucket numbers (because the bucket field is part of the primary clustered index of every SIFT table).

**Note**

The records that store the grand totals of SumIndexFields have bucket number 0 corresponding to SIFT level 0. Although only one record with SIFT level 0 can exist (because only one grand total value can exist for each SumIndexField), this SIFT level is not maintained as a default. However, you can activate this SIFT level if you want to. It is important to remember that this grand total must be updated every time that a record is added or altered in the base table. This can have a bad affect on performance because each user must wait until the grand total has been updated by the previous operation before their update can be performed.

Furthermore, the most detailed bucket level is not maintained as a default value. This bucket level can also be activated. For information about maintaining bucket levels, see "SIFT and Performance" on page 502.

SIFT level 1 means that only one field (the first one) from the SIFT key makes up the buckets at this level. In other words, the number of records in the SIFT table that have

SIFT level 1 is equal to the number of different values that are stored in the first field in the SIFT key in the base table. The number of records in the SIFT table that have SIFT level 2 is defined by the number of different combinations of values that are stored in the first and second fields of the SIFT key in the base table, and so on.

Here is a simple example:

Base Table:

Rec. No.	Item No.	Location Code	Amount	Qty.
1	ITM001	BLUE	100	10
2	ITM002	BLUE	400	20
3	ITM001	YELLOW	450	30
4	ITM003	YELLOW	1200	40
5	ITM001	RED	1000	50

Base Table Keys:

Enabled	Fields	SumIndexFields	MaintainSIFTIndex
YES	Rec. No.		YES
YES	Item No., Location Code	Amount, Qty.	YES

This table has one SIFT key that has two SumIndexFields associated with it.

According to the data stored in the base table the following buckets and predefined sums will be calculated and stored in the SIFT Table:

SIFT Table:

bucket	f2	f3	s4	s5	
1	ITM001		1550	90	SIFT level 0 is not maintained as a default. The number of records at SIFT level 1 depends on the number of different values that are stored in the <b>Item No.</b> column (alias <i>f2</i> ) of the base table.
1	ITM002		400	20	
1	ITM003		1200	40	
2	ITM001	BLUE	100	10	The number of records at SIFT level 2 depends on the number of different possible combinations that can be composed from the values stored in the <b>Item No.</b> (alias <i>f2</i> ) and <b>Location Code</b> (alias <i>f3</i> ) columns of the base table.
2	ITM001	RED	1000	50	
2	ITM001	YELLOW	450	30	
2	ITM002	BLUE	400	20	
2	ITM003	YELLOW	1200	40	

As you can see, the highest bucket number in this SIFT table is 2 (in the example it is the number of fields, included in the SIFT key) and there are therefore only 2 SIFT levels maintained in this table. The number of records at each SIFT level is determined by the

data stored in the base table. On each SIFT level this number can be calculated as the number of possible combinations that can be made from the values in the key columns that compose this bucket. Finally, the s-fields of every record contain the precalculated sums of the values stored in the SumIndexFields **Amount** and **Qty**. The corresponding fields in the SIFT table are s4 and s5. These sums are calculated according to the SIFT level that they belong to.

Therefore, the s4 and s5 fields of the record with SIFT level 1 where f2 (Item No.) is ITM001 contain the totals of the values stored in the **Amount** and **Qty**. fields in the base table where the Item No. is ITM001.

Base Table:

Rec. No.	Item No.	Location Code	Amount	Qty.	
1	ITM001	BLUE	100	10	The records from the base table that contribute to the sums stored for SIFT level 1 in the SIFT table. This record in the SIFT table has bucket number 1 and Item No. ITM001.
3	ITM001	YELLOW	450	30	
5	ITM001	RED	1000	50	

SIFT Table:

bucket	f2	f3	s4	s5	
1	ITM001		1550	90	The record in the SIFT table that stores precalculated sums for this SIFT level. This SIFT level is composed of a single column f2 ( <b>Item No.</b> ). (1550 = 100 + 450 + 1000, 90 = 10 + 30 + 50)

These precalculated totals will be used to produce the sums that are requested in the following C/AL code:

```
SETCURRENTKEY("Item No.");
SETRANGE("Item No.", 'ITM001');
CALCSUMS("Amount", "Qty.");
```

### SIFT Levels and Fields of the Date Data Type

From the previous example, it might be assumed that the maximum value of a SIFT level is always defined by the number of fields included in the SIFT key. However, this is not always the case. If one or more fields of the *Date* data type are included in the SIFT key, the number of SIFT levels increases. This is because each field of the *Date* data type in the SIFT key causes not one but three SIFT levels to be created. The system was designed this way to answer requests for totals that are based on dates.

Instead of having one SIFT level per date, there is one per year, one per month of the year and one per day of the month of the year. This allows us to calculate totals that are based on dates more efficiently.

In the following example, the base table contains a new column of the *Date* data type, called **Invoice Date**. This field is included in the SIFT key **Item No.**, **Location Code**, **Invoice Date** and two SumIndexFields **Amount** and **Qty.** are associated with this key. Let's take a look at the SIFT table and analyze the bucket structure that is created for this SIFT key.

Base Table:

Rec. No.	Item No.	Location Code	Invoice Date	Amount	Qty.
1	ITM001	BLUE	12 Jan 2000	100	10
2	ITM002	BLUE	23 Feb 2001	400	20
3	ITM001	YELLOW	17 Mar 2001	450	30
4	ITM003	YELLOW	19 Mar 2001	1200	40
5	ITM001	RED	28 Mar 2001	1000	50

Base Table Keys:

Enabled	Fields	SumIndexFields	MaintainSIFTIndex
YES	Rec. No.		YES
YES	Item No., Location Code, Invoice Date	Amount, Qty.	YES

SIFT Table:

Bucket	f2	f3	f4	s5	s6	
1	ITM001		01 Jan 1753	1550	90	SIFT level 0 is not supported as a default.
1	ITM002		01 Jan 1753	400	20	
1	ITM003		01 Jan 1753	1200	40	
2	ITM001	BLUE	01 Jan 1753	100	10	The date 01 Jan 1753 is interpreted as an undefined date ('0D') on SQL Server.
2	ITM001	RED	01 Jan 1753	1000	50	
2	ITM001	YELLOW	01 Jan 1753	450	30	
2	ITM002	BLUE	01 Jan 1753	400	20	



Bucket	f2	f3	f4	s5	s6
2	ITM003	YELLOW	01 Jan 1753	1200	40
3	ITM001	BLUE	01 Jan 2000	100	10
3	ITM001	RED	01 Jan 2001	1000	50
3	ITM001	YELLOW	01 Jan 2001	450	30
3	ITM002	BLUE	01 Jan 2001	400	20
3	ITM003	YELLOW	01 Jan 2001	1200	40
4	ITM001	BLUE	01 Jan 2000	100	10
4	ITM001	RED	01 Mar 2001	1000	50
4	ITM001	YELLOW	01 Mar 2001	450	30
4	ITM002	BLUE	01 Feb 2001	400	20
4	ITM003	YELLOW	01 Mar 2001	1200	40
5	ITM001	BLUE	12 Jan 2000	100	10
5	ITM001	RED	28 Mar 2001	1000	50
5	ITM001	YELLOW	17 Mar 2001	450	30
5	ITM002	BLUE	23 Feb 2001	400	20
5	ITM003	YELLOW	19 Mar 2001	1200	40

As you can see, the number of records in the SIFT table has increased dramatically. The upper part of the SIFT table that contains the records at SIFT levels 1 and 2 has exactly the same layout as it had in the first example (if you don't count the new column – f4). All the changes are visible at the bottom of the SIFT table. Three more bucket numbers corresponding to 3 more SIFT levels have been created – 3, 4 and 5.

To generate the records at SIFT level 3 in the SIFT table, all the values stored in the Invoice Date column of the base table are converted to the "first-day-of-year" date. This date has the format: 01 Jan XXXX, where XXXX is the year of the date that is converted. For example, 17 Mar 2001 is converted to 01 Jan 2001 and 12 Jan 2000 is converted to 01 Jan 2000. After this conversion the records at SIFT level 3 are generated. They contain totals for the SumIndexFields for all the possible combinations of Item No., Location Code and the converted dates from the Invoice Date column. In other words, SIFT level 3 represents the Item No., Location Code, Invoice YEAR(Date) buckets.

To generate the records at SIFT level 4 in the SIFT table, all the values stored in the Invoice Date column of the base table are converted to the "first-day-of-month-of-year" date. This date has the format: 01 Mmm XXXX, where Mmm is the month and XXXX is the year of the date that is converted. 17 Mar 2001 is converted to 01 Mar 2001 and 12 Jan 2000 is converted to 01 Jan 2000. After this conversion the records at SIFT level 4 are generated. They contain totals for the SumIndexFields for all the possible combinations of Item No., Location Code and the converted dates from the Invoice

Date column. In other words, SIFT level 4 represents the Item No., Location Code, Invoice MONTH-OF-YEAR(Date) buckets.

Finally, to generate the records at SIFT level 5 in the SIFT table, all the values stored in the Invoice Date column of the base table are used as they are, without any conversions. The records at SIFT level 5 contain totals for the SumIndexFields for all the possible combinations of Item No., Location Code and the dates stored in Invoice Date column. In the other words, SIFT level 5 represents the Item No., Location Code, Invoice DATE(Date) buckets.

This configuration of SIFT levels makes calculating totals based on dates faster. If you want to calculate the total amount of the item ITM001 that are stored in the BLUE location and have been invoiced during the year 2000, this sum is precalculated and stored in the s5 field of the following record in the SIFT table:

Bucket	f2	f3	f4	s5	s6
3	ITM001	BLUE	01 Jan 2000	100	10

These precalculated totals will be used to produce the sums that are requested in the following C/AL code:

```
SETCURRENTKEY("Item No.", "Location Code", "Invoice Date");
SETRANGE("Item No.", 'ITM001');
SETRANGE("Location Code", 'BLUE');
SETRANGE("Invoice Date", 010100D, CLOSINGDATE(311200D));
CALCSUMS("Amount", "Qty");
```

A more advanced request wants to calculate the total amount of the item ITM001 that is stored in the BLUE location and were invoiced between 07 Mar 1998 and 14 Jun 2001. The algorithm used to make this calculation is more complicated and includes several steps. However, the sum will still be calculated more efficiently in this way than by directly searching the base table for the relevant records and aggregating them, especially when the number of records is greater than it is in this simple example.

Generally, calculating sums using SIFT tables gets more efficient the greater the amount of records that fall within the parameters specified in the filter.

### SIFT Levels and Fields of the DateTime Data Type

SIFT keys can also contain fields of the *DateTime* data type. Fields of *DateTime* data type can generate up to seven SIFT levels; one to support each level of detail that is contained in a datetime field: year, month, day, hour, minute, second and millisecond.

However, Dynamics NAV only supports three of these levels by default: year, month and day.

Furthermore, we recommend that if a SIFT key contains a field of the *DateTime* data type, this is the last field in the SIFT key. If another field comes after a datetime field in a SIFT key, the most detailed SIFT level of the datetime field is automatically maintained as part of the SIFT level that is created for the last field in the SIFT key. The most detailed level is milliseconds, and this means that the SIFT table will contain a bucket for each millisecond. The SIFT table will therefore contain as many buckets as there are records in the base table because it is almost impossible to enter two records into the base table at the same millisecond. There is therefore no point in maintaining

this SIFT level as no performance benefit can be gained from calculating sums based on a SIFT table that contains as many buckets as there are records in the base table.

### Important

.....  
 If a field SIFT key contains a field of the *DateTime* data type, this field must be the last field in the SIFT key.  
 .....

## SIFT Tables

### Indexes

It is important to know that each SIFT table has its own primary clustered index. This index is composed of the bucket column and all the f-columns in the SIFT table. The name of this index has the following format: *<SIFT Table Name>\_idx*.

For example, one of the SIFT tables supported by table 17, **G/L Entry** is *CRONUS International Ltd\_\$17\$1* and its primary clustered index is called *CRONUS International Ltd\$17\$1\_idx*. The fields, included in this index are: bucket, *f3*, *f45*, *f23*, *f24*, *f71* and *f4*.

Sometimes you can improve performance by creating non-clustered secondary index for a SIFT table. The name of this index has the following format: *<SIFT Table Name>\_hlp\_idx*. A non-clustered secondary index always consists of a single field.

For example, the SIFT table *CRONUS International Ltd\$17\$1* has the non-clustered secondary index called *CRONUS International Ltd\$17\$1\_hlp\_idx* and this index consists of field *f4*.

### Layout estimation

Before you create a SIFT key, you might want to estimate the layout of the SIFT table that will be created and maintained to support this key. This will help you understand the amount of support that the new SIFT key will require.

### Extended key

However, before you can estimate the layout of a SIFT table you must understand the concept of the extended key. It is a standard feature of database design to add all the fields in the primary key to the secondary keys to facilitate sorting.

This extended key is not visible in Dynamics NAV but can be seen in SQL Server:

Index	Clustered	Index_Keys
Cronus International Ltd_\$G/L Entry\$0		Entry No.
\$1	No	G/L Account No., Posting Date, Entry No.
\$2	No	G/L Account No., Business Unit Code, Global Dimension 1 Code, Global Dimension 2 Code, Close Income Statement Dim. ID, Posting Date, Entry No.
\$3	No	Document No., Posting Date, Entry No.
\$4	No	Transaction No., Entry No.
\$5	No	Close Income Statement Dim. ID, Entry No.

As you can see from this table, the field in the primary key has been added to all the secondary keys – compare it to the table on page 487. Furthermore, the fields in the primary key are only added to the secondary keys if they are not already part of the secondary key.

The following rules will help you calculate the number of columns and SIFT levels that will be supported by SIFT during the SIFT key design phase:

- 1 The first column in a SIFT table is always the bucket column. This is where the bucket number is stored.
- 2 The f-columns are next. To estimate the number of f-columns, use the following formula:

If the last field in the SIFT key is of the *Date* data type, the number of f-columns in the SIFT table is equal to the number of fields in the SIFT key.

If the last field of the SIFT key is of any other data type, the number of f-columns in the SIFT table is equal to the number of fields in the SIFT key minus 1 (the f-field representing the last field in this kind of SIFT key will not appear in the SIFT table).

A SIFT key based on a non-primary key in the Base table has a composite layout. This means that after the user has included the fields in the key, all the fields in the primary key that are still not a part of this SIFT key are also included in it. Here is an example:

Table 17, G/L Entry - Keys (fragment):

Enabled	Key	SumIndexFields
YES	Entry No.	
YES	G/L Account No., Posting Date	Amount, Debit Amount, Credit Amount, Additional-Currency Amount, Add.-Currency Debit Amount, Add.-Currency Credit Amount

These are the first two keys in the **G/L Entry** table. The first key is the primary key and consists of a single field: **Entry No.** The second key is one of the secondary keys in this table and it has SumIndexFields associated with it. This secondary key consists of two fields: **G/L Account No.**, **Posting Date.**

**Entry No.** is the field in the primary key and is added at the end of every secondary key if the user hasn't already added it to this key. In other words, all the fields in the primary key are always included in the SIFT key. So the extended secondary key consists of three fields: **G/L Account No.**, **Posting Date** and **Entry No.**

Therefore, the extended SIFT key also consists of three fields: **G/L Account No.**, **Posting Date** and **Entry No.** As stated previously – *all the fields in the primary key are always included in the SIFT key.*

The last field in this SIFT Key is not of the *Date* data type (and the corresponding f-columns are not included in the SIFT table). That is why the number of f-columns in the SIFT table in this example is equal to the number of fields in the SIFT key minus 1. The SIFT Table *CRONUS International Ltd\_ \$17\$0* has two f-columns *f3* and *f4*, corresponding to the **G/L Account No.** and **Posting Date** fields in the base table, respectively.

- 3 Finally, in every SIFT table there are the s-columns or sum-columns. The number of s-columns is always equal to the number of SumIndexFields, associated with the SIFT key. In this example six SumIndexFields are associated with the SIFT key: **Amount**, **Debit Amount**, **Credit Amount**, **Additional-Currency Amount**, **Add.-Currency Debit Amount** and **Add.-Currency Credit Amount**. Therefore, the SIFT table contains six s-columns named *s17*, *s53*, *s54*, *s68*, *s69* and *s70* after each of the SumIndexFields.

The number of SIFT levels that are supported can be calculated by analyzing the data types of the fields that are included in the SIFT key:

- Every field of the *Date* and *DateTime* data types generate three SIFT levels.
- Key fields of any other data type generate only one SIFT level each.
- To optimize performance, SIFT level 0 (grand total sums) and the last SIFT level (the so-called most detailed bucket level) are not included in SIFT tables.
- All the fields in the primary key that are not specified as part of the SIFT key are also included in the SIFT key.

Therefore, the number of SIFT levels that are supported can be calculated as:

- the number of fields of the *Date* and *DateTime* data type that exist in SIFT key multiplied by three, plus the number of fields of any other data type in the SIFT Key minus one.

Furthermore, if the first field in the SIFT key is a field of the Boolean or Option data type, this field does not generate a SIFT level. Therefore, the calculated number of SIFT levels should be reduced by 1. In this case the first SIFT level in the table will be 2 (because SIFT level 0 is not used and the first SIFT level is ignored because the first field in the SIFT key is Boolean). Therefore, all the records belonging to the first SIFT level in the SIFT table will have value 2 in the bucket field. If any of the other fields in the SIFT key is a field of the Boolean data type, it does produce a SIFT level.

Let's take another look at our example:

It has a SIFT key that is composed of three fields: **G/L Account No.**, **Posting Date** and **Entry No.** There is one field of the *Date* data type in this key and there are two other fields. Therefore, the number of different SIFT levels in the SIFT table *CRONUS International Ltd\_\$17\$0* is:

$$1 \times 3 + 2 - 1 = 4.$$

In this table, the first SIFT level is 1 because the first field in the key is of the *Code* data type (neither *Option* nor *Boolean*). You can easily check these calculations by using tools like SQL Query Analyzer or SQL Server Enterprise Manager to inspect the *CRONUS International Ltd\_\$17\$0* table in your database.

### Updating the Base Table

Every time you insert, delete or update data in a base table that can change the precalculated sums that are stored in the SIFT tables for this particular base table, all of the affected SIFT tables must also be updated. The SIFT trigger manages this procedure automatically. However, the important thing to understand is that every single record that is inserted into a base table can cause hundreds of records to be updated in the SIFT tables.

The following example illustrates how this works. The base table contains the following records:

Rec. No.	Item No.	Location Code	Invoice Date	Amount	Qty.
1	ITM001	BLUE	12 Jan 2000	100	10
2	ITM002	BLUE	23 Feb 2001	400	20
3	ITM001	YELLOW	17 Mar 2001	450	30
4	ITM003	YELLOW	19 Mar 2001	1200	40
5	ITM001	RED	28 Mar 2001	1000	50

If you, for example, insert the following record into the base table:

6	ITM002	BLUE	12 Feb 2001	2000	80
---	--------	------	-------------	------	----

The SIFT trigger will update the following rows in the SIFT table:

At SIFT level 1, one record is updated:

bucket	f2	f3	f4	s5	s6	
1	ITM002			2400	100	The totals for the <b>Amount</b> and <b>Qty.</b> fields for ITM002 are affected by adding the new record.
		Updated values:		(400)	(20)	

At SIFT level 2, one record is updated:

bucket	f2	f3	f4	s5	s6	
2	ITM002	BLUE		2400	100	The totals for the <b>Amount</b> and <b>Qty.</b> fields for ITM002 in the BLUE location are affected by adding the new record.
		Updated values:		(400)	(20)	

At SIFT level 3, one record is updated:

Bucket	f2	f3	f4	s5	s6	
3	ITM002	BLUE	01 Jan 2001	2400	100	The totals for the <b>Amount</b> and <b>Qty.</b> fields for ITM002 in the BLUE location, posted in the year 2001 are affected by adding the new record.
		Updated values:		(400)	(20)	

At SIFT level 4, one record is updated:

Bucket	f2	f3	f4	s5	s6	
4	ITM002	BLUE	01 Feb 2001	2400	100	The totals for the <b>Amount</b> and <b>Qty.</b> fields for ITM002 in the BLUE location, posted in February 2001 are affected by adding the new record.
		Updated values:		(400)	(20)	

At SIFT level 5, one new record is inserted:

Bucket	f2	f3	f4	s5	s6	
5	ITM001	YELLOW	17 Mar 2001	2400	30	
5	ITM002	YELLOW	12 Feb 2001	2000	80	This record is inserted at SIFT level 5.
5	ITM002	BLUE	23 Feb 2001	400	20	
5	ITM003	YELLOW	19 Mar 2001	1200	40	

The SIFT trigger automatically performs all these modifications when the record is inserted into the base table. As you can see, inserting this single record in the base table causes modifications to be made to multiple records in the SIFT table. In this example only a few records were affected by the changes to the base table.

Some of the tables in Dynamics NAV contain many large SIFT keys. This means that updating the SIFT tables can take a long time. This decrease in performance when updating the base tables is the price that must be paid if the system is to contain SumIndexFields that facilitate rapid calculations. That is why it is crucial that you choose the right configuration of table keys when you are designing a table.

Keeping your keys as short and as selective as possible can dramatically reduce the complexity of the layout of the SIFT tables and reduce the time required by the SIFT trigger to update the SIFT tables. Keeping the primary key of the table as short as possible is particularly important because all of the fields in the primary key are always included in every SIFT key that you create in that table.

### Deleting Records from the Base Table

When you delete a record from a base table, the SIFT table is updated in the normal way and all of the aggregated totals are updated. However, the record is not deleted from the SIFT table; its corresponding totals in the SIFT table are set to zero. The entries in the SIFT table are not removed because there is a performance benefit to be gained for future updates by keeping them.

### Configuring the SIFT Levels

As stated earlier, both SIFT level 0 (the Grand Total) and the most detailed SIFT level are not maintained as a default. However, you can decide to maintain these SIFT levels if you need them. Furthermore, you can also decide not to maintain any of the other SIFT levels if you do not need them.

In the following example the *MaintainSIFTIndex* property of the key is set to *Yes*, indicating that SIFT structures necessary for maintaining the SumIndexFields associated with this key have been created on SQL Server.

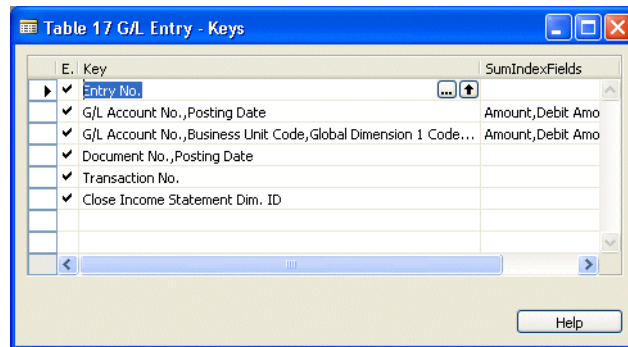
If you no longer want to maintain these SIFT structures, you must set the *MaintainSIFTIndex* property for this key to *No*. For more information about the factors that must be taken into consideration before deciding whether or not to maintain these structures, see page 502.

To change the SIFT levels that are maintained:

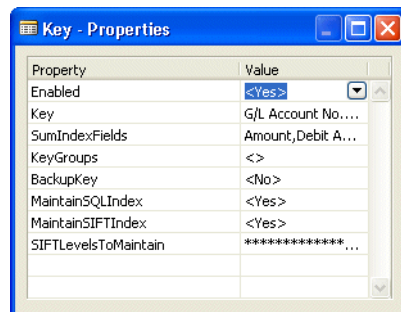
- 1 Open the Object Designer and click Table.
- 2 Select the table that contains the SIFT indexes that you want to modify and click Design. In this example, use table 17, **G/L Entry**.



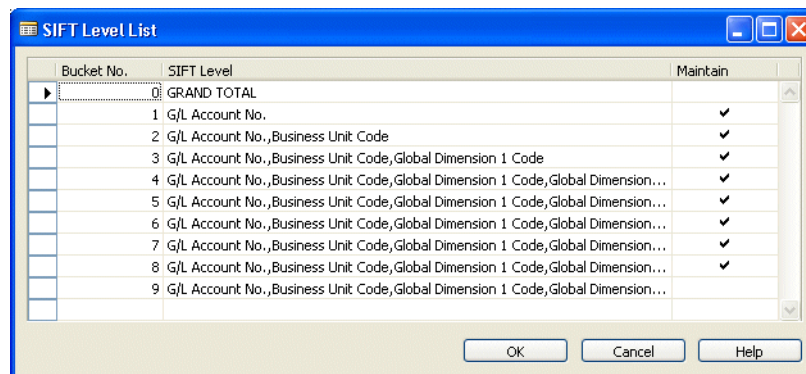
3 Click View, Keys to open the **Keys** window for this table:



4 Select the key that you want to modify and click View, Properties. The **Key – Properties** window for this key appears:



5 In the **Value** field of the *SIFTLevelsToMaintain* property, use the AssistButton ... to open the **SIFT Level List** window.




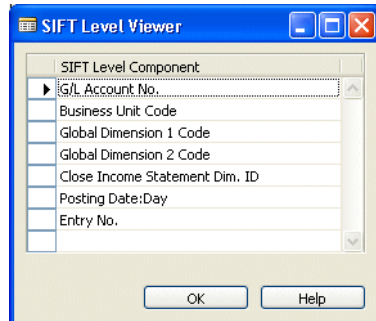
This window lists all the SIFT levels and their components that have been created to support the SumIndexFields associated with this key.

6 Enter or remove a check mark from the **Maintain** field to specify whether or not you want to maintain a particular SIFT level.

**Important**

Adding or removing a SIFT level will mean that parts of the corresponding SIFT table will have to be rebuilt. This could be time-consuming.

- 7 If the SIFT **Level** field contains so much information that it cannot be displayed in the window, use the AssistButton  in the **SIFT Level** field to open the **SIFT Level Viewer** window.



This window lists all of the components that make up this SIFT level.

## SIFT and Performance

As explained earlier, every time you update a key or a SumIndexField in a base table all of the SIFT tables that are associated with the base table must also be updated. This means that the number of SIFT tables that you create, as well as the number of SIFT levels that you maintain, affects performance.

If you have a very dynamic base table that constantly has records inserted, modified and deleted, the SIFT tables that are associated with it will constantly have to be updated.

The SIFT tables can get very large, both because of the new records that are entered and because the records that are deleted from the base table are not removed from the SIFT tables. This can also badly affect performance, particularly when the SIFT tables are queried to calculate sums.

Factors to consider The factors that you must take into consideration when you deal with any performance problems that arise include:

- Have you designed your SIFT indexes optimally?

Supporting too many SIFT indexes will affect performance.

Having unnecessary date fields in the SIFT indexes of the base table will affect performance because they create three times as many entries as an ordinary field.

Supporting too many fields in the SIFT indexes will also affect performance.

The fields in the SIFT index that are used most regularly in queries must be positioned to the left in the SIFT index. As a general rule, the field that contains the greatest number of unique values must be placed on the left with the field that contains the second greatest number of unique values on its right and so on. Integer fields generally contain the greatest number of unique values and Option fields contain a relatively small number of values.

- Are there too many SIFT levels?

Maintaining the Grand Total (SIFT level 0) can affect multiuser performance and lead to concurrency problems because this total must be updated every time a record is entered or modified in the base table.

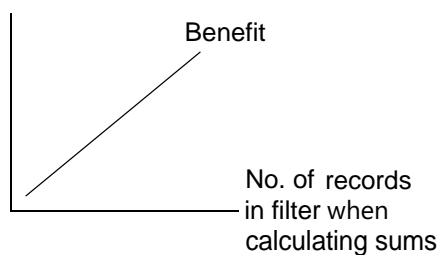
Maintaining the most detailed SIFT level is not recommended because you will not need totals that are this detailed often enough to justify the cost to performance.

You must regularly use the totals generated by the SIFT levels that are maintained for a particular SIFT index to justify the cost in performance of maintaining these SIFT levels. If the filtered set of records that the totals are based on is large, it is generally worthwhile maintaining the SIFT structures. If the filtered set of records that the totals are based on is small, do not maintain the SIFT structures.

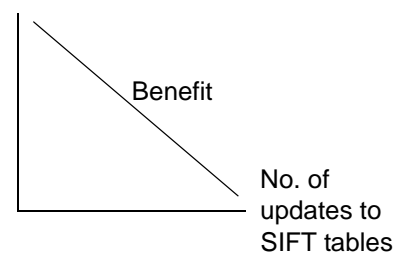
Consider the costs and the benefits of maintaining SIFT tables and SIFT levels:

Cost	Benefit
Updates to the SIFT tables	Fast calculation of sums
Potential locking conflicts	

Maintain SIFT structures



Maintain SIFT structures



These graphs illustrate some of the factors that must be taken into consideration before deciding to maintain the SIFT structures and determining how many SIFT levels to maintain.

- You can prevent the SIFT tables from being updated by setting the *MaintainSIFTIndex* property of the index in the base table to *No*. This means that you no longer benefit from SIFT's ability to calculate sums quickly. However, the SIFT functionality is still available.
- You can reduce the cost of updating the SIFT table by not maintaining all of the SIFT levels that are generated by a particular index. This means that some totals are not readily available and will have to be calculated when you need them.
- You can reduce the cost of updating and limit the size of the SIFT table by optimizing it and removing the records that contain zero values in all the *SumIndexFields*.
- If the base table doesn't grow or only grows slowly, there is no need to set the *MaintainSIFTIndex* property of any indexes that contain *SumIndexFields* to *Yes*. If the base table does grow, you ought to set the *MaintainSIFTIndex* property of any indexes that contain *SumIndexFields* to *Yes*.

**Important**

.....

It is important that you remember to carry out some tests every time you make any changes to the SIFT structures in Dynamics NAV. You must make sure that the changes that you have made do not cause problems in any other areas of the application. You must also ensure that your changes do not have a negative affect on performance.

.....

**Note**

.....

If you set the *MaintainSIFTIndex* property to *No*, you should not set the *MaintainSQLIndex* to *No*.

.....

**Optimizing SIFT Tables**

If one of your SIFT tables becomes very large you might want to determine whether or not it should be optimized.

Run a SQL query on the SIFT table to find out how many records there are with zero values in all the sum fields in the table. If there are a large number of these records, you can initiate the optimization process in Dynamics NAV and remove them.

The optimization process removes any entries that contain zero values in all numeric fields from each SIFT table. The removal of these redundant entries frees up space and makes updating and summing SIFT information more efficient.

To initiate the optimization process click File, Database, Information, Tables, Optimize.

For more information about optimizing SIFT tables, see the chapter "Working with Databases" in the manual *Installation & System Management: SQL Server Option for Microsoft Dynamics NAV*.

## **Chapter 25**

### **Type Conversion**

This appendix describes all possible type conversions in C/AL expressions. The appendix is divided into the following sections:

- Type Conversion in Expressions
- Type Conversion Mechanisms

## 25.1 Type Conversion in Expressions

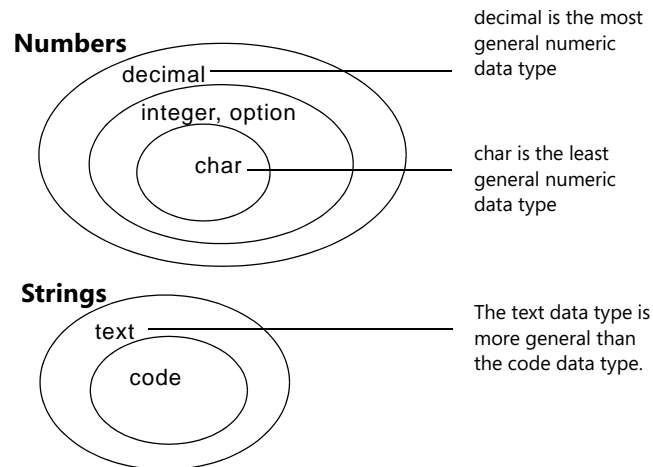
Consider the following statements:

```
CharVar := 15; // A char variable
integerVar := 56000; // An integer variable
Sum := CharVar + integerVar;
```

The last statement involves one or two type conversions. The right-hand side of the statement involves the evaluation of the expression `CharVar + integerVar` (`char + integer`). In order to evaluate this expression, the first operand (`CharVar`) will have to be converted from `char` to `integer`. The addition operator will then return an `integer` result. But if the type of the left-hand side variable has been declared as, for example, `decimal`, the result must be converted from `integer` to `decimal` before its value can be assigned to `Sum` (this kind of conversion is discussed in "Assignment and Type Conversion" on page 308.)

This appendix describes the type conversions which sometimes take place when expressions are evaluated. First, some general rules:

- When asked to evaluate an expression of mixed data types, the system will (if possible) always convert at least one of the operands to a more general data type.
- The data types in the two main groups, numbers and strings, can be ranked from "most general" to "least general."



- The most general data types include all the possible values from the less general data types: a decimal is more general than an integer, which again is more general than a char.
- Type conversion can take place in some cases even though two operands have the same type.

These rules can be illustrated by some examples.

**Example 1**

Evaluation of a numeric expression:

```
integer + decimal
```

This expression contains two sub-expressions of different data types. Before it can add these two sub-expressions, the system must convert the left-hand side sub-expression to decimal:

```
decimal + decimal
```

When the left-hand side sub-expression has been converted, the expression can be evaluated, and the resulting data type will be decimal:

```
decimal + decimal = decimal
```

**Example 2**

Evaluation of a string expression:

```
text + code
```

This expression contains two sub-expressions that must be concatenated. To do this, the system must convert the sub-expression of the least general data type (code) to the most general data type (text).

```
text + text
```

When the right-hand side argument has been converted, the expression can be evaluated, and the resulting data type will be text.

```
text + text = text
```

## 25.2 Type Conversion Mechanisms

This section discusses the type conversion mechanisms for the C/AL operators in more depth. The starting point is to divide the operators into some main categories:

- Relational operators
- Logical operators
- Arithmetic operators

The following subsections discuss the properties of operators in C/AL: for each category of operators, there are descriptions of the valid data types for the arguments and the data types that result when expressions are evaluated.

The relational operators will be treated first, as they are common to most of the C/AL data types.

### Relational Operators

The relational operators are used to compare expressions. The following table defines the evaluation rules for relational operators. The rules assume that the data types of the expressions can be compared. Refer to the next section "Valid Uses of Relational Operators" for a complete overview of comparable data types.

Operator	Operator Name	Expression	Resulting Data Type
>	Greater than	Expr > Expr	boolean
<	Less than	Expr < Expr	boolean
<=	Less than or equal	Expr >= Expr	boolean
<>	Not equal to	Expr <> Expr	boolean
=	Equal to	Expr = Expr	boolean
IN	In range	Expr IN [Valueset]	boolean

**Note**

When using relational operators, upper and lower case letters in strings are significant. Furthermore, the comparison is based on the built-in character comparison table of the system, that is, not by comparing "true" ASCII characters.

### Valid Uses of Relational Operators

The following table describes the valid uses of the relational operators and the data types that result when expressions are evaluated. The invalid combinations of types for relational operators are indicated by a dash. All relational operators are binary infix operators; that is, they take a left and a right argument and are placed between the arguments.

The rows in the table show the type of the left argument and the columns show the type of the right argument. The cells show the resulting data type.



From the table you can see that a valid use of the relational operators is, for example, text compared with text or code, while boolean cannot be compared with anything other than boolean, and so forth.

Relational Operators	bool	char	option	integer	decimal	date	time	text	code
bool	bool	-	-	-	-	-	-	-	-
char	-	bool	bool	bool	bool	-	-	-	-
option	-	bool	bool	bool	bool	-	-	-	-
integer	-	bool	bool	bool	bool	-	-	-	-
decimal	-	bool	bool	bool	bool	-	-	-	-
date	-	-	-	-	-	bool	-	-	-
time	-	-	-	-	-	-	bool	-	-
text	-	-	-	-	-	-	-	bool	bool
code	-	-	-	-	-	-	-	bool	bool

### Boolean (Logical) Operators

The logical operators can only be used with arguments that can be evaluated to boolean.

Operator	Name	Expression	Resulting Data Type
NOT	Logical negation	NOT bool	bool
AND	Logical and	bool AND bool	bool
OR	Logical or	bool OR bool	bool
XOR	Exclusive logical or	bool XOR bool	bool

As this table shows, the NOT operator is a unary prefix operator. This means that it takes only one argument and is placed in front of the argument. The AND, OR and XOR operators, on the other hand, are binary infix operators; that is, they take two arguments and are placed between the corresponding arguments.

### Arithmetic Operators

Here are some examples of how to use the type conversion rules for arithmetic operators. The examples illustrate how the operators are supposed to be used and the effect of the type conversion made automatically by the C/AL compiler. The examples have been divided into groups corresponding to the data types in C/AL.

For a full description of the type conversion rules in C/AL, refer to the tables in the section "Complete Overview of Type Conversion Rules" on page 511, which provide a full description of all the possible uses of C/AL operators and the resulting data types.

**Example**

This table illustrates type conversion in integer operator expressions

Operator	Name	Expression	Resulting Data Type
+	Unary plus	+ integer	integer
-	Unary minus	- integer	integer
+	Addition	integer + integer	integer
-	Subtraction	integer - integer	integer
*	Multiplication	integer * integer	integer
/	Division	integer / integer	decimal
DIV	Integer division	integer DIV integer	integer
MOD	Modulus	integer MOD integer	integer

Note that the same rules apply to option operator expressions as well.

**Example**

This table illustrates type conversion in decimal operator expressions:

Operator	Name	Expression	Resulting Data Type
+	Unary plus	+ decimal	decimal
-	Unary minus	- decimal	decimal
+	Addition	decimal + decimal	decimal
-	Subtraction	decimal - decimal	decimal
*	Multiplication	decimal * decimal	decimal
/	Division	decimal / decimal	decimal
DIV	Integer Division	decimal DIV decimal	decimal
MOD	Modulus	decimal MOD decimal	decimal

**Example**

This table illustrates type conversion in date operator expressions:

Operator	Name	Expression	Resulting Data Type
+	date addition	date + Number	date
-	date subtraction	date - Number	date
-	date difference	date - date	integer

In the "date addition" and "date subtraction" examples, a runtime error will occur if date is a closing date or if date is undefined (OD).

**Example**

This table illustrates type conversion in time operator expressions:

Operator	Name	Expression	Resulting Data Type
+	time addition	time + integer	time
-	time difference	time - time	integer

The time unit is milliseconds. If time is undefined (0T), a runtime error will occur.

**Example**

This table illustrates type conversion in text and code (String) operator expressions:

Operator	Name	Expression	Resulting Data Type
+	Concatenation	text + text	text
+	Concatenation	text + code	text
+	Concatenation	code + text	text
+	Concatenation	code + code	code

**Complete Overview of Type Conversion Rules**

The following tables provide a complete overview of type conversion rules for the arithmetic operators.

**The Unary Arithmetic Operators**

The unary arithmetic operators in C/AL are so-called prefix operators, whose syntax is:

`PrefixExpression = PrefixOperator Expression`

This table shows the data types for which the unary operators in C/AL are defined, and the resulting data types.

Unary Operator	option	integer	decimal
+	integer	integer	decimal
-	integer	integer	decimal

**The Binary Arithmetic Operators**

This table shows the data types for which the binary arithmetic operators are defined. The binary arithmetic operators in C/AL are all infix operators, that is:

`InfixExpression = LeftExpression InfixOperator RightExpression`

Operator	boolean	char	option	integer	decimal	date	time	text	code
+	○	●	●	●	●	●	●	●	●
-	○	●	●	●	●	●	●	○	○

	boolean	char	option	integer	decimal	date	time	text	code
*	<input type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
/	<input type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
DIV	<input type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
MOD	<input type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

- Yes, the operator can take at least one operand (left, right or both) of the given type.
- No, the operator cannot be used with the given type.

The following tables define the valid uses of the binary arithmetic operators, and the resulting data types.

### Definition of Type Conversion Rules for the "+" Operator

The "+" operator	boolean	char	option	integer	decimal	date	time	text	code
<b>boolean</b>	-	-	-	-	-	-	-	-	-
<b>char</b>	-	integer	integer <sup>(C)</sup>	integer <sup>(C)</sup>	decimal <sup>(C)</sup>	-	-	-	-
<b>option</b>	-	integer <sup>(C)</sup>	integer <sup>(C)</sup>	integer <sup>(C)</sup>	decimal <sup>(C)</sup>	-	-	-	-
<b>integer</b>	-	integer <sup>(C)</sup>	integer <sup>(C)</sup>	integer <sup>(C)</sup>	decimal <sup>(C)</sup>	-	-	-	-
<b>decimal</b>	-	decimal <sup>(C)</sup>	decimal <sup>(C)</sup>	decimal <sup>(C)</sup>	decimal <sup>(C)</sup>	-	-	-	-
<b>date</b>	-	date <sup>(A)</sup> (C)	date <sup>(A)</sup> (C)	date <sup>(A)</sup> (C)	date <sup>(A)</sup> (C) (D)	-	-	-	-
<b>time</b>	-	time <sup>(B)</sup> (C)	time <sup>(B)</sup> (C)	time <sup>(B)</sup> (C)	time <sup>(B)</sup> (C) (D)	-	-	-	-
<b>text</b>	-	-	-	-	-	-	-	text	text
<b>code</b>	-	-	-	-	-	-	-	text	code

- (A) the operation is not defined for the date 0D.
- (B) the operation is not defined for the time 0T.
- (C) Overflow may occur.
- (D) the operation is not defined if decimal has a fractional part.

**Definition of Type Conversion Rules for the "-" Operator**

The "-" operator	boolean	char	option	integer	decimal	date	time	text, code
<b>boolean</b>	-	-	-	-	-	-	-	-
<b>char</b>	-	integer	integer <sup>(C)</sup>	integer <sup>(C)</sup>	decimal <sup>(C)</sup>	-	-	-
<b>option</b>	-	integer <sup>(C)</sup>	integer	integer	decimal <sup>(C)</sup>	-	-	-
<b>integer</b>	-	integer <sup>(C)</sup>	integer	integer	decimal <sup>(C)</sup>	-	-	-
<b>decimal</b>	-	decimal <sup>(C)</sup>	decimal <sup>(C)</sup>	decimal <sup>(C)</sup>	decimal <sup>(C)</sup>	-	-	-
<b>date</b>	-	date <sup>(A)(C)</sup>	date <sup>(A)(C)</sup>	date <sup>(A)(C)</sup>	date <sup>(A)(C)(D)</sup>	integer <sup>(A)</sup>	-	-
<b>time</b>	-	time <sup>(B)(C)</sup>	time <sup>(B)(C)</sup>	time <sup>(B)(C)</sup>	time <sup>(B)(C)(D)</sup>	-	integer <sup>(B)</sup>	-
<b>text</b>	-	-	-	-	-	-	-	-
<b>code</b>	-	-	-	-	-	-	-	-

(A) the operation is not defined for the date 0D.

(B) the operation is not defined for the time 0T.

(C) overflow may occur.

(D) the operation is not defined if decimal has a fractional part.

**Definition of Type Conversion Rules for the "\*" Operator**

*	char	option	integer	decimal
<b>char</b>	integer <sup>(C)</sup>	integer <sup>(C)</sup>	integer <sup>(C)</sup>	decimal <sup>(C)</sup>
<b>option</b>	integer <sup>(C)</sup>	integer <sup>(C)</sup>	integer <sup>(C)</sup>	decimal <sup>(C)</sup>
<b>integer</b>	integer <sup>(C)</sup>	integer <sup>(C)</sup>	integer <sup>(C)</sup>	decimal <sup>(C)</sup>
<b>decimal</b>	decimal <sup>(C)</sup>	decimal <sup>(C)</sup>	decimal <sup>(C)</sup>	decimal <sup>(C)</sup>

(C) overflow may occur.

**Definition of Type Conversion Rules for the "/" Operator**

/	char	option	integer	decimal
<b>char</b>	decimal	decimal	decimal	decimal
<b>option</b>	decimal	decimal	decimal	decimal
<b>integer</b>	decimal	decimal	decimal	decimal
<b>decimal</b>	decimal	decimal	decimal	decimal

Note that overflow may occur in all cases in this table.

A runtime error will occur if the right operand is equal to zero (0).

**Definition of Type Conversion Rules for the 'MOD' and 'DIV' Operators**

<b>MOD and DIV</b>	<b>char</b>	<b>option</b>	<b>integer</b>	<b>decimal</b>
<b>char</b>	integer	integer	integer	decimal
<b>option</b>	integer	integer	integer	decimal
<b>integer</b>	integer	integer	integer	decimal
<b>decimal</b>	decimal	decimal	decimal	decimal

A runtime error will occur if the right operand is equal to zero (0).

## **Chapter 26**

### **Numbering in Dynamics NAV**

This chapter explains how items, such as documents, are numbered in Dynamics NAV. The information is helpful if you use or are planning to use the SQL Server Option for Dynamics NAV. We also recommend that you read this chapter if you use C/SIDE Database Server and want Dynamics NAV to sort numbers correctly when you view data with external programs.

This chapter contains the following section:

- How Does Number Sorting Work?

## 26.1 How Does Number Sorting Work?

Code fields in Dynamics NAV can contain both numerical values and text strings. Dynamics NAV ensures that numbers kept in code fields on C/SIDE Database Server are sorted in the correct numerical order. However, this does not necessarily happen when you use external programs to access the same data. External programs may view and sort these numbers as text. This means that when Dynamics NAV sorts the data, comparisons are made character by character, and not by comparing the numeric content of the strings.

Numbers that you keep in code fields on SQL Server using the *Varchar* SQL data type are not sorted in the correct numerical order. The SQL Server Option for Dynamics NAV sorts the numbers as if they were text strings. The following table illustrates the differences that occur:

Numerical Sorting	Text Sorting
1	1
2	10
3	100
4	2
10	3
100	4

To avoid this problem, we recommend that you use a numerical series that has a fixed length. You can do this in three ways:

- Define a numerical series as consisting of a predefined number of digits that start with a digit other than zero, for example, 100-399 (300 numbers). If this numerical series is too short for your requirements, you can start a new numerical series, for example, 40,000-69,999 (30,000 numbers). If this numerical series is too short, you can start a new one such as 7,000,000-9,999,999 (3,000,000 numbers). Users will quickly get used to entering numbers that have a fixed length, and the numbers will be sorted correctly.

```

1001
-----
1002
-----
1003
-----
.
-----
.
-----
9999
-----

```

This is the solution that we recommend because you can now define the SQL data type as being either *Varchar* or *Integer* and the sorting will still be correct.

- Define a numerical series that consists of a predefined number of digits and that starts with a letter, such as A001-A999. This series will be sorted correctly. When the



series is complete, you can define a new series by starting with a different letter. The users will quickly get used to entering numbers that have a fixed length, and the numbers will be sorted correctly.

A001
A002
A003
.
.
A999

- Define a numerical series as consisting of a predefined number of digits that start with zeros, for example, 001-999.

We do not recommend this solution because there are several inherent drawbacks. Firstly, users tend to ignore the zeros and to refer to the first number as 1. Users may, therefore, omit the zeros when entering numbers. Secondly, the numerical series feature in Dynamics NAV does not permit numbers that start with zero.

Furthermore, the SQL Server Option for Dynamics NAV will not allow you to save numbers that are defined according to this system as the *Integer* SQL data type.

**Important**

As a general rule, data types used in fields that are related to each other must be compatible. Therefore, when you use a SQL data type in a field, you will normally have to change the SQL data type settings of related fields in other tables. For example, in the General Ledger application area, if you change the SQL data type of the **No.** field in the **G/L Account** table from *Varchar* to *Integer* (or if you change the data type from *Code* to *Text*), you must change the data type of the **G/L Account No.** fields in the **G/L Entry** and **G/L Budget Entry** tables to the corresponding data type. Failure to do so results in the display of incorrect totals, based on these tables, in the chart of accounts and elsewhere.

**Numbering Principles**

To ensure that numbers kept in code or text fields are sorted correctly, irrespective of which database server you are using, you must use the following principles:

- *Always* use a numerical series that has a fixed length, for example, 100-399.
- *Never* use a numerical series such as 1-999 in code or text fields.
- *Never* use a numerical series such as 001-999 in code or text fields.

**Filters**

If you do not follow the numbering principles, problems will arise when you apply filters that involve numbers in Dynamics NAV.

Here is an example:

- If you have not used a numerical series that has a fixed length, when you apply a filter, for example, 10..20, the result will be 10,100.....20.

When you follow the numbering principles, you must remember to use these for filters that you apply. Here are two examples:

- If you do not follow the numbering principles when you apply a filter, for example, 2..10, the result will contain no records. This is because 2 comes after 10.
- You have followed the numbering principles and are using three-digit numbers. If you forget to follow the same principles when you apply a filter, for example, 10..20, the result will be 100,101,102.....199.

## **Chapter 27**

### **C/SIDE in Multiuser Environments**

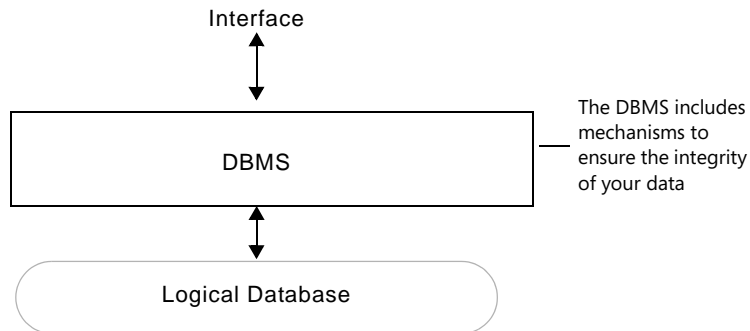
This chapter explains how the Database Management System handles data integrity in a multiuser environment. It describes how the C/SIDE system handles situations where more than one user or process try to change the same object.

This chapter contains the following section:

- Ensuring Data Integrity in a Multiuser Environment
- Locking in Dynamics NAV – a Comparison of the two Server Options

## 27.1 Ensuring Data Integrity in a Multiuser Environment

Data integrity deals with the reliability of the data stored in the database, that is, the requirement that the database must describe the real world as accurately as possible. All access to the data in your database goes through the DBMS (Database Management System) as illustrated in the following figure:



The DBMS controls the interaction of the user with the database to ensure that a number of data integrity constraints are observed. By observing these constraints, the DBMS protects your data from damage or corruption. The DBMS is a very complex unit in your database system, but the methods that it uses to maintain data integrity are based on a few fundamental concepts:

- write transactions and recovery
- read consistency and concurrency
- table locking
- deadlock detection
- committing updates

These concepts are discussed and explained in the following sections.

### Write Transactions and Recovery

A write transaction in C/SIDE is defined as an atomic unit of work on the database which is either completed entirely or not at all. In other words, a transaction is a way to encapsulate a sequence of read and write operations on the database in order to ensure that either all or none of the operations is performed. The concept of write transactions is a general C/SIDE facility that is used both in single- and multiuser environments.

When a transaction has been submitted to the C/SIDE DBMS, the system is responsible for making sure that:

- all the transaction operations are completed successfully and their effect is recorded permanently in the database.
- or
- the transaction has no effect at all on the database.

The DBMS must prevent the following situation from occurring: some of the transaction operations are applied to the database while the other operations in the

same transaction are not applied. A situation like this could occur if a transaction fails while executing.

Some typical reasons for a transaction to fail are:

- The user decides to abort the transaction.
- The transaction cannot be completed because some information is missing.
- The system crashes, due to hardware or software errors.
- There are operation errors, such as overflow or division by zero.

If the transaction is aborted, all the tables are restored to the state they were in before the transaction started.

A typical example of a write transaction would be transferring \$100 from one account to another. This involves two operations in a single database:

- 1 Subtract \$100 from account A.
- 2 Add \$100 to account B.

If a power failure or some other fatal error interrupts the program after the first operation, the database is not in a consistent state, because the second operation has not been completed. However, bundling both operations into a single transaction, ensures that either none or both of the operations are executed and the data will always be consistent.

### More on Write Transactions

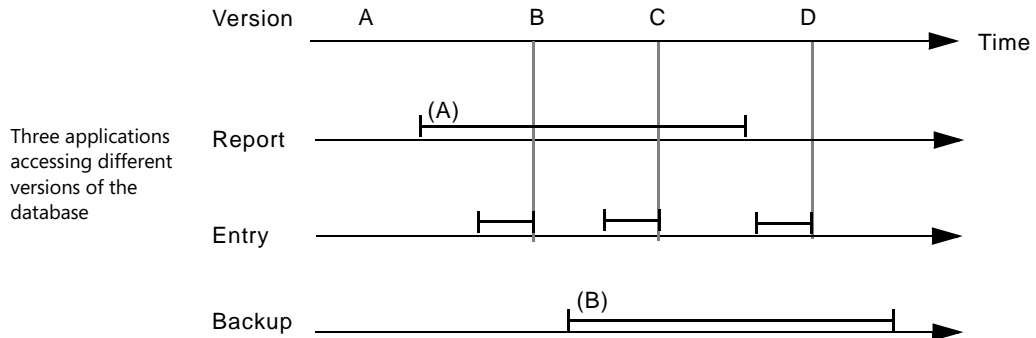
The previous section explained that the database will always be consistent regardless of whether a transaction is committed or aborted. The way C/SIDE handles write transactions and keeps the database consistent is different from traditional approaches. Traditionally, database systems contain a facility that automatically maintains a log file which records all changes to the database. This log file contains images of a record before it is modified and after it is modified, "before" and "after" images. The changes recorded in the log file can be used to recover the database if failures occur.

Assume that an application program aborts because of power failure or is aborted by the operator. The database is now in an inconsistent state, and all the modifications that were already made to the database must be cancelled. In common database systems this is achieved by so-called *rollback recovery*, that is, by backing out the updates of the application program. This rollback is performed by reading the log file backwards and undoing the recorded changes to the database, until the point where the application program started. This restores the modified records to their original contents.

The C/SIDE DBMS does not need to use a log file because the C/SIDE database is *data-version oriented*. This means that each time a transaction is committed, a new version of the database is created. While you enter new data in the database your changes are private; not until you commit the changes, does the new data become public and establish the newest version. The DBMS allows different applications to access and modify the database concurrently by letting them work on individual versions that are snapshots of the database taken at the point in time where the applications start to access the database. The advantages of the data version approach will become clear as you read through the following sections.

### Read Consistency and Concurrency

C/SIDE is data-version oriented, meaning that each time a write transaction is performed, a new version of the data in the database is created. The following figure shows three applications accessing the same database. Imagine that the first access is made by a report. The second access is made by a user who inserts new entries in the database, and the third access is made by a backup procedure.



In this example, generating the report is a time-consuming process, and while the report is generated, another user enters or modifies records in the database. As each entry is committed, a new version of the database is created, but when the report started, a snapshot of the database was made and the report continues to work on version A of the database. A third user starts a backup procedure. When the backup starts, a snapshot of the current (most recent) version of the database, B, is made, and the backup works on this version, uninfluenced by new data that the second user continues to enter into the system. Working with data versions makes it possible for many users to access the database without interfering with each other.

The implications of the data-version approach are many; most important is that different applications may be reading different versions of the same database. These versions are snapshots of the database at the time when the application started to access the database. In this way the DBMS allows for concurrency while still maintaining read consistency. If the accesses involve only data retrieval and no changes, then the newest version will persist for all applications. There will be no new version until a write transaction is performed.

When you update the database, your modifications are private and only become public after you commit your updates. Your newly-committed updates plus the part of the database which was not modified make up the newest version.

### What is a Data Version?

The data in the database is stored in a well known data structure that resembles a tree. A tree data structure consists of nodes. Each node in the tree, except for a special node called the root, has one parent node and one or more child nodes. The root node has no parent. A node that has no child node is called a leaf; a non-leaf node is called an internal node. The level of a node is defined as one more than the level of its parent, with the level of the root node being zero.

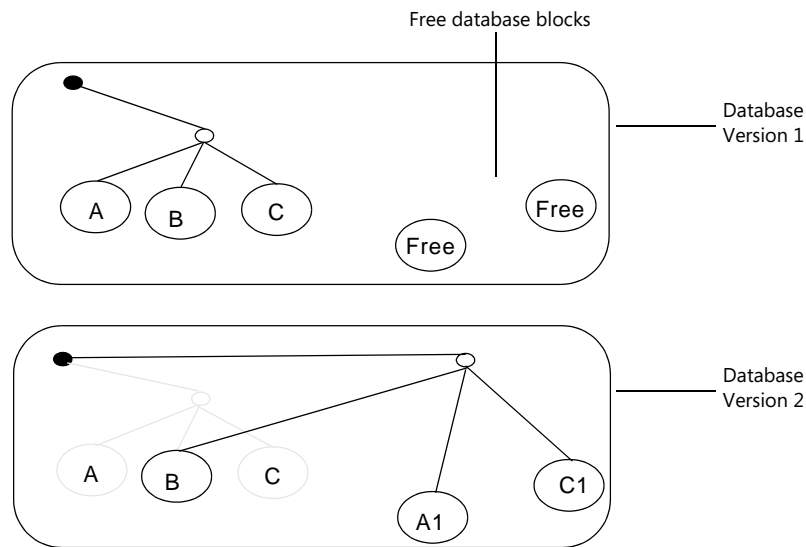
The data structure used in the C/SIDE database is called a B+ tree. This means that the tree structure is balanced and that the data (records) are stored only in the leaf nodes, not in the internal nodes. A balanced tree has the advantage that it always contains the minimum number of levels necessary to contain the nodes, so all search paths will be

the shortest possible. A B+ tree structure is an efficient data structure that enables fast searches to be performed.

### Example

Imagine that the tree structure in your database contains a branch with customers A, B and C and that there are two free database blocks available.

Assume that you need to modify customer A and C. When you update the records, the DBMS makes a copy of the original. As illustrated in the following figure, the copies will use two free database blocks. You will then modify the copies, and the system will create a new internal node.



If an error occurs during the transaction, or the user decides to abort the changes, the database blocks occupied by the copied branch will be freed and be available for new database updates.

If the transaction is committed, this new internal node will replace the old node, and the database blocks used by the old versions of customer A and C will now be available as free database blocks that can be used by database updates.

The database contains a number of historical versions. Gradually, as the free area in the database is consumed by succeeding historical versions, new versions begin to replace the oldest versions.

Slow operations can run into trouble in this environment. Suppose Application A is reading data from the latest version, while generating a very time-consuming report. In the meantime, Application B begins performing write transactions which consist of order entries.

Newer versions of the database are created as the order entries are added to the database. The maximum number of historical versions in your database depends upon the space in the database that currently is not used by the newest version, that is:

$$\frac{\text{The maximum defined size of the database}}{\text{The amount of space currently used to hold the newest version}} = \text{Space available for historical versions}$$

At some point the data version accessed by A becomes the oldest complete data version. But B needs a database block from this version to complete its modifications.

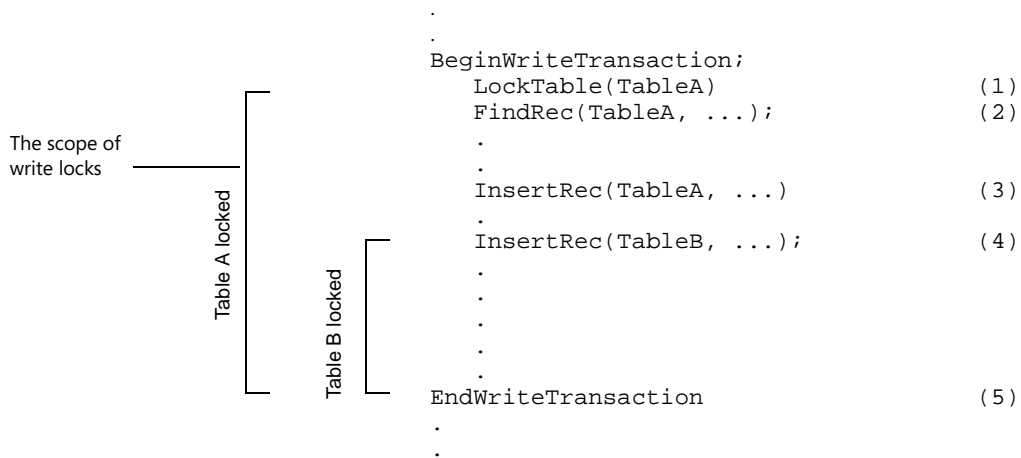
This conflict is solved by the DBMS by giving priority to the write transaction and ejecting application A. A runtime error message is sent to A on its next read operation – "Data version is no longer valid" – and it is forced to restart the process with the newest version. But as long as B continues and the space in the database available for historical versions remains the same, there is little hope that A will be able to generate the report. (Enlarging the database could solve the problem.)

### What is Table Locking?

In multiuser environments, the DBMS ensures the integrity of the data by setting write locks on all the tables you are updating. This prevents other users from making changes to the same tables.

While write operations automatically lock a table during updates, you can explicitly lock a table, even if you are not performing a write operation. By locking a table immediately before accessing a record, you are assured that the data you might change in the record conforms to the data you have read, even if some time has elapsed. A write lock does not influence data retrieval. This means that locking a table does not prevent other users from gaining read access to the records in the table.

With C/SIDE Database Server, a write lock is active until the write transaction is either aborted or committed. This figure uses pseudo-language syntax to illustrate the scope of write locks.



This illustrates both an *explicit lock* and an *automatic lock*. Line (1) in the write transaction explicitly locks table A. If this explicit lock was not set on table A, the DBMS would automatically lock this table when a record is inserted (3). Table B is not locked explicitly, but is locked automatically by the DBMS when a record is inserted (4). Both locks are active until the `EndWriteTransaction` command is executed in line (5).

### What Is Deadlock Detection?

table locking must be carefully coordinated if a multiuser system is to function correctly.

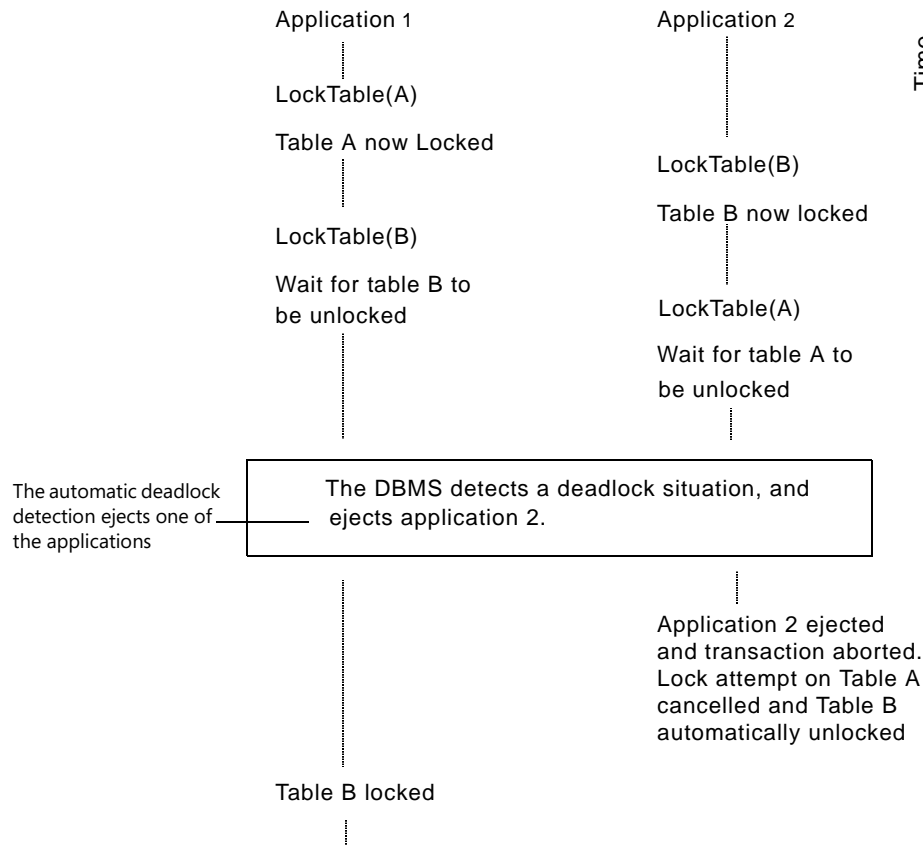
If different transactions require write access to several tables at once, you must be careful to avoid a *Deadlock* situation. A deadlock occurs when one transaction has



access to and locks some of the tables it needs and another transaction has locked some other tables that it needs and neither transaction can continue before the other has finished and releases the tables that it has locked.

This means that each transaction must wait for the other to finish. As a result both processes will have to wait forever. This situation is also known as Deadly Embrace.

In order to avoid deadlocks, the DBMS has an automatic deadlock detection mechanism, which detects these situations and ejects one of the write transactions. The following figure illustrates how a deadlock can arise:



The DBMS will always eject the application that causes the deadlock – as in the previous example. This rule applies no matter how many applications are involved in a deadlock.

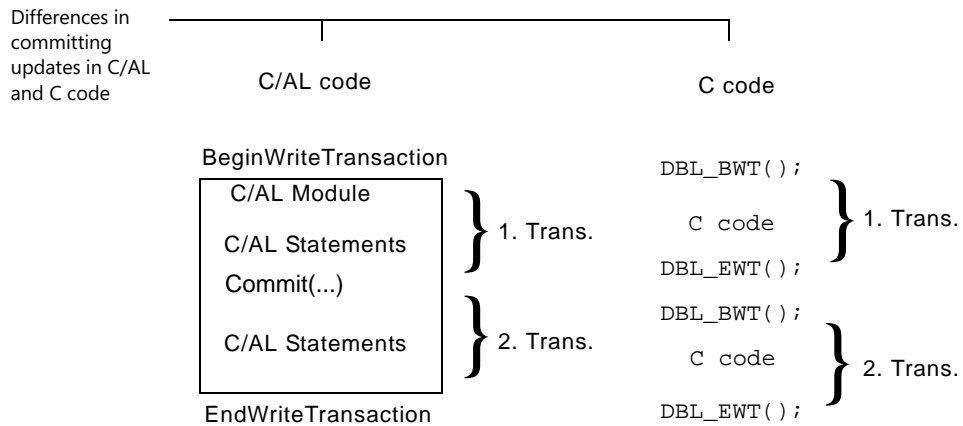
### Are There Any Differences between Commit in C/AL and C?

Although the concept of committing an update is the same whether you are using C/AL code or C/FRONT (the toolkit that allows you to develop applications, in the C programming language, that access a C/SIDE database), there are some minor differences. This subsection explains these differences in detail.

When you want to perform an update using C/FRONT, the first thing you must do is to tell the system explicitly that you want to perform a write transaction (use `DBL_BWT`, `BeginWriteTransaction`.) Likewise you must use `DBL_EWT` (`EndWriteTransaction`) to explicitly tell the system when your write transaction ends.

When you use C/AL code to update a C/SIDE database, these `BeginWriteTransaction` and `EndWriteTransaction` statements are handled implicitly by the system. That is, the system automatically executes these commands before and after the C/AL code is executed. This means that if you only need to perform a single write transaction you do not have to commit your update explicitly: it is done automatically by the system. If, however, you need to perform more than one write transaction, you have to use `COMMIT()` in order to separate the transactions.

The next figure illustrates these differences:



The C/AL code contains two write transactions. When the C/AL code starts to execute, the write transactions are automatically enabled. By issuing the `COMMIT()` command, you tell the system that the first write transaction has ended, and you prepare the system for the second. As the execution of the C/AL code is completed, the system automatically ends the second write transaction. When you use C code to perform the same transactions, each transaction must be explicitly encapsulated within the `DBL_BWT()` and `DBL_EWT()` commands.

## 27.2 Locking in Dynamics NAV – a Comparison of the two Server Options

This section explains the differences and similarities in the way that locking is carried out in the two database options for Dynamics NAV: C/SIDE Database Server and the SQL Server Option.

### Important

The following information only covers the default transaction type `UpdateNoLocks` for the SQL Server Option for Dynamics NAV. For information about the other transaction types, see the *C/SIDE Reference Guide* online Help.

### Both Server Options

Locking	In the beginning of a transaction, the data that you read is not locked. This means that reading data does not conflict with transactions that modify the same data. If you want to ensure that you are reading the latest data from a table, you must lock the table before you read it.
Locking single records	Normally, you must not lock a record before you read it even though you may want to modify or delete it afterwards. When you try to modify or delete the record, you get an error message if another transaction has modified or deleted the record in the meantime. You receive this error message because C/SIDE checks the timestamp that it keeps on all the records in a database and detects that the timestamp on the copy you have read is different from the timestamp on the modified record in the database.
Locking record sets	Normally, you lock a table before reading a set of records in that table if you want to read these records again later to modify or delete them. You must lock the table to ensure that another transaction does not modify these records in the meantime.  You will not receive an error message if you do not lock the table even though the records have been modified as a result of other transactions being carried out while you were reading them.

### Minimizing Deadlocks

To minimize the amount of deadlocks that occur, you must lock records and tables in the same order for all transactions. You can also try to locate areas where you lock more records and tables than you actually need, and then diminish the extent of these locks or remove them completely. This can prevent conflicts from occurring that involve these records and tables.

If this does not prevent deadlocks, you can, as a last resort, lock more records and tables to prevent transactions from running concurrently.

If you cannot prevent the occurrence of deadlocks by programming, you must run the deadlocking transactions separately.

### Locking in C/SIDE Database Server

Data that is not locked is read from the same snapshot (version) of the database. If you call a modifying function (for example, `INSERT`, `MODIFY` or `DELETE`), on a table, the whole table is locked.

**Locking record sets** As mentioned earlier, you normally lock a table before reading a set of records in that table if you want to read these records again later and modify or delete them. With C/SIDE Database Server, you can choose to lock the table with `LOCKTABLE(TRUE, TRUE)` after reading the records for the first time instead of locking with `LOCKTABLE` before reading the records for the first time.

When you try to modify or delete the records, you receive an error message if another transaction has modified the records in the meantime.

You also receive an error message if another transaction has inserted a record into the record set in the meantime. However, if another transaction has deleted a record from the record set in the meantime, you will not be able to notice this change. The purpose of locking with `LOCKTABLE(TRUE, TRUE)` after reading the records for the first time is to improve concurrency by postponing the table lock that C/SIDE Database Server puts on the table.

## Locking in SQL Server

When data is read without locking, you get the latest (possibly uncommitted) data from the database. If you call `Rec.LOCKTABLE`, nothing happens right away. However, when data is read from the table after `LOCKTABLE` has been called, the data is locked.

If you call `INSERT`, `MODIFY` or `DELETE`, the specified record is locked immediately. This means that two transactions, which either insert, modify or delete separate records in the same table do not conflict. Furthermore, locks are also placed whenever data is read from the table after the modifying function has been called.

When you call `INSERT`, `MODIFY` or `DELETE`, only one record is locked when no `SumIndexFields` are maintained in the table or when calling `INSERT`, `MODIFY` or `DELETE` doesn't require any `SumIndexFields` to be updated. If you place a lock on a sum, you prevent other transactions from updating that sum.

It is also important to note that even though SQL Server initially puts locks on single records, it can also choose to escalate a single record lock to a table lock. It will do so if it determines that the overall performance can be improved by not having to set locks on individual records. The improvement in performance must outweigh the loss in concurrency that this excessive locking causes.

If you specify what record to read, for example, by calling `Rec.GET`, that record is locked. This means that two transactions, which read specific, but separate, records in a table do not cause conflicts.

If you browse a record set (that is, read sequentially through a set of records), for example, by calling `Rec.FIND(' - ')` or `Rec.NEXT`, the record set (including the empty space) is locked as you browse through it. However, the locking implementation used in SQL Server means that the record before and the record after this record set are also locked. This means that two transactions, which just read separate sets of records in a table will cause a conflict if there are no records between these two record sets. When locks are placed on a record set, other transactions cannot put locks on any record within the set.

Note that C/SIDE decides how many records to retrieve from the server when you ask for the first or the next record within a set. C/SIDE then handles subsequent reads with no additional effort, and fewer calls to the server give better performance. In addition

to improving performance, this means that you cannot precisely predict when locks are set when you browse.

The SQL Server Option for Dynamics NAV only supports the default values for the parameters of the `LOCKTABLE` function – `LOCKTABLE(TRUE, FALSE)`.

#### Note

Dynamics NAV tables that have keys defined for `SumIndexFields` cause additional tables to be created in SQL Server to support SIFT functionality. One table is created for each key that contains `SumIndexFields`. When you modify a Dynamics NAV table that has keys defined for `SumIndexFields`, modifications can be made to these SQL Server tables. When this happens, there is no guarantee that two transactions can modify different records in the Dynamics NAV table without causing conflicts.

### Locking Differences in the Code

A typical use of `LOCKTABLE(TRUE, TRUE)` in C/SIDE Database Server is shown in the first column of the following table. The equivalent code for the SQL Server Option is shown in the second column. The code that works on both servers is shown in the third column. The `RECORDLEVELLOCKING` property is used to detect whether record level locking is being used. If this is the case, then you are using the SQL Server Option for Dynamics NAV. This is currently the only server that supports record level locking.

C/SIDE Database Server	SQL Server
<pre>IF Rec.FIND('-') THEN   REPEAT     UNTIL Rec.NEXT = 0; Rec.LOCKTABLE(TRUE, TRUE); IF Rec.FIND('-') THEN   REPEAT     Rec.MODIFY;   UNTIL Rec.NEXT = 0;</pre>	<pre>Rec.LOCKTABLE; IF Rec.FIND('-') THEN   REPEAT     UNTIL Rec.NEXT = 0; IF Rec.FIND('-') THEN   REPEAT     Rec.MODIFY;   UNTIL Rec.NEXT = 0;</pre>

#### Both Server Types

```
IF Rec.RECORDLEVELLOCKING THEN
  Rec.LOCKTABLE;
IF Rec.FIND('-') THEN
  REPEAT
    UNTIL Rec.NEXT = 0;
IF NOT Rec.RECORDLEVELLOCKING THEN
  Rec.LOCKTABLE(TRUE, TRUE);
IF Rec.FIND('-') THEN
  REPEAT
    Rec.MODIFY;
  UNTIL Rec.NEXT = 0;
```



## **Chapter 28**

### **Caption Class Functionality**

This chapter describes the caption class functionality and explains how the CaptionClassTranslate function trigger (ID 15) in Codeunit 1 deals with it.

The chapter covers the following subjects:

- Syntax
- Function Code

## 28.1 Syntax

If the *CaptionClass* property of a field or a control is defined, the function trigger `CaptionClassTranslate` (ID 15) in Codeunit 1 (ApplicationManagement) is called by the system every time the field or control is shown. The purpose of this function is to replace the caption, as defined in the design of the field or control, with another string.

The syntax of this procedure is:

```
CaptionClassTranslate (<LANGUAGE>;<CAPTIONEXPR>)
LANGUAGE
    <DataType>    := [Integer]
    <DataValue>   := .....
CAPTIONEXPR
    <DataType>    := [String]
    <Length>      <= 80
    <DataValue>   := <CAPTIONAREA> , <CAPTIONREF>
```

As you can see, two parameters are passed to this function:

- LANGUAGE
- CAPTIONEXPR

**LANGUAGE** The LANGUAGE parameter is automatically mentioned by the system as is the Windows Language ID of the active language in Dynamics NAV.

### Example

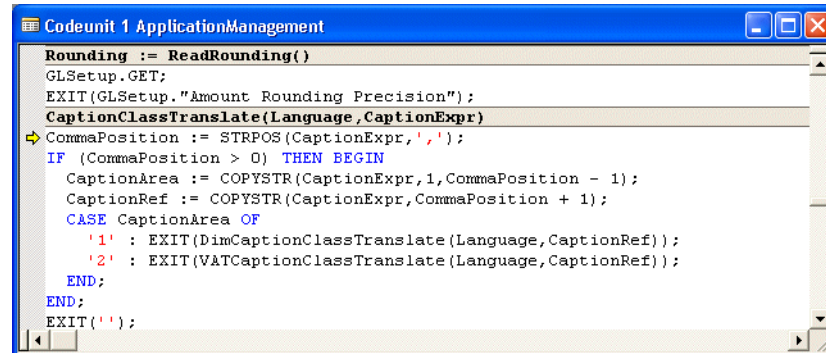
If the active language in Dynamics NAV is English (United States), LANGUAGE will hold the value 1033.

**CAPTIONEXPR** The CAPTIONEXPR parameter holds the content of the *CaptionClass* property of the field or control.



### Example

In Table 13 (*Salesperson/Purchaser*), the **Global Dimension 1 Code (5050)** field has the string '1,1,1' as its *CaptionClass*. Open the *Salespeople/Purchasers* form, activate the debugger and open the **Zoom** window (Ctrl + F8) and the *CaptionClassTranslate* function trigger pops up in the Debugger:

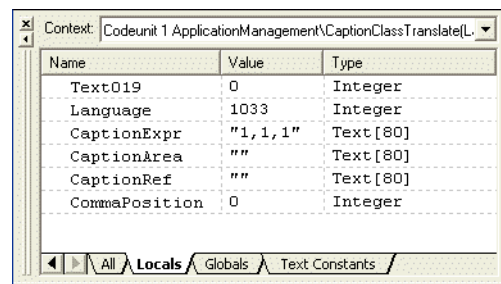


```

Rounding := ReadRounding()
GLSetup.GET;
EXIT(GLSetup."Amount Rounding Precision");
CaptionClassTranslate(Language,CaptionExpr)
CommaPosition := STRPOS(CaptionExpr,',');
IF (CommaPosition > 0) THEN BEGIN
CaptionArea := COPYSTR(CaptionExpr,1,CommaPosition - 1);
CaptionRef := COPYSTR(CaptionExpr,CommaPosition + 1);
CASE CaptionArea OF
'1' : EXIT(DimCaptionClassTranslate(Language,CaptionRef));
'2' : EXIT(VATCaptionClassTranslate(Language,CaptionRef));
END;
END;
EXIT('');

```

In the *C/AL Locals* window, you can see that the *CaptionExpr* parameter holds the string '1,1,1'.



Name	Value	Type
Text019	0	Integer
Language	1033	Integer
CaptionExpr	"1,1,1"	Text [80]
CaptionArea	""	Text [80]
CaptionRef	""	Text [80]
CommaPosition	0	Integer

### Note

In the **Zoom** window, you will not find the **Global Dimension 1 Code** field. Instead you find the **Department Code** field – as a result of the *CaptionClass* property and the *CaptionClassTranslate* function trigger.

### Function Code

In a way, the function trigger *CaptionClassTranslate* (ID 15) is a system trigger. A programmer can intervene here every time the *CaptionClass* property – if it is defined – is evaluated by the system.

If we take a look at this trigger in the standard CRONUS database, we see that it already contains some code:

```

CaptionClassTranslate(Language : Integer;CaptionExpr : Text[80])
CommaPosition := STRPOS(CaptionExpr,',');
IF (CommaPosition > 0) THEN BEGIN
CaptionArea := COPYSTR(CaptionExpr,1,CommaPosition - 1);
CaptionRef := COPYSTR(CaptionExpr,CommaPosition + 1);
CASE CaptionArea OF
'1' : EXIT(DimCaptionClassTranslate(Language, CaptionRef));

```

```

    '2' : EXIT(VATCaptionClassTranslate(Language, CaptionRef));
  END;
END;
EXIT(");

```

This standard code analyzes and unravels the CaptionExpr parameter. As we saw earlier, this parameter has the following syntax:

```
CAPTIONEXPR := <CAPTIONAREA>, <CAPTIONREF>
```

Depending upon the value of the CAPTIONAREA, different procedures are called. Look at the CASE ... OF statement, either:

```
DimCaptionClassTranslate(Language, CaptionRef)
```

or

```
VATCaptionClassTranslate(Language, CaptionRef)
```

Here is a detailed description of these functions.

**CAPTIONAREA** The first part of the CaptionExpr parameter, up to the first comma, is the CAPTIONAREA, and has the following syntax:

```

CAPTIONAREA
  <DataType> := [SubString]
  <Length>   <= 10
  <DataValue> := 1..9999999999
  // 1 for Dimension Area
  // 2 for VAT

```

#### Note

.....

In the standard functionality, only two CAPTIONAREA values are defined: 1 for Dimension Area and 2 for VAT.

.....

**CAPTIONREF** The second part of the CaptionExpr parameter, after the first comma, is the CAPTIONREF, and has the following syntax:

```

CAPTIONREF
  <DataType> := [SubString]
  <Length>   <= 10
  <DataValue> :=
  IF (<CAPTIONAREA> = 1)
    <DIMCAPTIONTYPE>, <DIMCAPTIONREF>
  IF (<CAPTIONAREA> = 2)
    <VATCAPTIONTYPE>, <VATCAPTIONREF>

```

As you can see, depending upon the value of the CAPTIONAREA, CAPTIONREF can consist of either one (VATCAPTIONTYPE) or two references (VATCAPTIONTYPE, VATCAPTIONREF or DIMCAPTIONTYPE, DIMCAPTIONREF – and even more than two as we will see in the following).

**Note**

.....

This is the way the standard functionality in Dynamics NAV deals with the *CaptionClass* property. Every field or control with a defined *CaptionClass* has a string in this property with the syntax described earlier. For new functionality, a programmer could define other syntaxes and add code to the function trigger *CaptionClassTranslate* (ID 15) to handle these syntaxes.

.....

**Syntax for CAPTIONREF**

As described earlier, the CAPTIONREF part of the CaptionExpr parameter can have the following syntax:

```
CAPTIONREF := < DIMCAPTIONTYPE > , < DIMCAPTIONREF >
```

if CAPTIONAREA equals 1, or

```
CAPTIONREF := < VATCAPTIONTYPE > , < VATCAPTIONREF >
```

if CAPTIONAREA equals 2. In the following, we find the syntax for these different sub references.

**Dimension Area**

If the CAPTIONAREA equals 1, the caption of the field or control should be retrieved from the dimensions information.

**DIMCAPTIONTYPE** This reference determines where the main part of the new caption should be retrieved from. The syntax is:

```
DIMCAPTIONTYPE
  <DataType>   := [SubString]
  <Length>     <= 10
  <DataValue>  := 1..6
  // 1 to retrieve Code Caption of Global Dimension
  // 2 to retrieve Code Caption of Shortcut Dimension
  // 3 to retrieve Filter Caption of Global Dimension
  // 4 to retrieve Filter Caption of Shortcut Dimension
  // 5 to retrieve Code Caption of any kind of Dimensions
  // 6 to retrieve Filter Caption of any kind of Dimensions
```

**DIMCAPTIONREF** DIMCAPTIONREF consists of a number of sub references:

```
DIMCAPTIONREF := < number > , < DIMOPTIONALPARAM1 > ,
  < DIMOPTIONALPARAM2 >
```

The following syntax describes what < number > can be and what < DIMOPTIONALPARAM1 >, and < DIMOPTIONALPARAM2 > are:

```
DIMCAPTIONREF
  <DataType>   := [SubString]
  <Length>     <= 10
  <DataValue>  :=
```

```

IF (<DIMCAPTIONTYPE> = 1)
  1..2,<DIMOPTIONALPARAM1>,<DIMOPTIONALPARAM2>
IF (<DIMCAPTIONTYPE> = 2)
  1..8,<DIMOPTIONALPARAM1>,<DIMOPTIONALPARAM2>
IF (<DIMCAPTIONTYPE> = 3)
  1..2,<DIMOPTIONALPARAM1>,<DIMOPTIONALPARAM2>
IF (<DIMCAPTIONTYPE> = 4)
  1..8,<DIMOPTIONALPARAM1>,<DIMOPTIONALPARAM2>
IF (<DIMCAPTIONTYPE> = 5)
  [Table]Dimension.[Field]Code, <DIMOPTIONALPARAM1>,
  <DIMOPTIONALPARAM2>
IF (<DIMCAPTIONTYPE> = 6)
  [Table]Dimension.[Field]Code, <DIMOPTIONALPARAM1>,
  <DIMOPTIONALPARAM2>

```

**DIMOPTIONALPARAM1**

```

DIMOPTIONALPARAM1
  <DataType>   := [SubString]
  <Length>     <= 30
  <DataValue>  := [String]
  // a string added before the dimension name

```

**DIMOPTIONALPARAM2**

```

DIMOPTIONALPARAM2
  <DataType>   := [SubString]
  <Length>     <= 30
  <DataValue>  := [String]
  // a string added after the dimension name

```

**VAT**

If the CAPTIONAREA equals 2, the caption of the field or control should be replaced by its original caption plus an extra string. This string should state either 'Excl. VAT' or 'Incl. VAT'. The syntax is:

**VATCAPTIONTYPE**

```

VATCAPTIONTYPE
  <DataType>   := [SubString]
  <Length>     := 1
  <DataValue>  := '0' -> <field caption + 'Excl. VAT'>
  '1' -> <field caption + 'Incl. VAT'>

```

**VATCAPTIONREF**

VATCAPTIONREF contains the caption of the field or control:

```

VATCAPTIONREF
  <DataType>   := [SubString]
  <Length>     <= 30
  <DataValue>  := field caption

```

## 28.2 Function Code

### DimCaptionClassTranslate (ID 7)

After `CaptionClassTranslate` has sifted the contents of the `CaptionClass` property (passed in the `CaptionExpr` parameter) in a `CAPTIONAREA` and a `CAPTIONREF`, `DimCaptionClassTranslate` will be called (if `CAPTIONAREA` equals 1). It passes the Language ID and the `CAPTIONREF` part of the `CaptionClass` property.

This function can be split into three main parts:

- 1 Collect the G/L Setup data, if not done yet.
- 2 Sift out the comma separated subparts of the `CAPTIONREF` (see the previous description of the `CAPTIONREF` syntax.)
- 3 Determine what the caption should be, depending on the `DIMCAPTIONTYPE` and `DIMCAPTIONREF`.

### Code

```
DimCaptionClassTranslate(Language : Integer;CaptionExpr : Text[80]) : Text[80]
```

Begin (1)

```
IF NOT GLSetupRead THEN BEGIN
```

```
IF NOT GLSetup.GET THEN
```

```
EXIT("");
```

```
GLSetupRead := TRUE;
```

End (1)

```
END;
```

Begin (2)

```
CommaPosition := STRPOS(CaptionExpr,',');
```

```
IF (CommaPosition > 0) THEN BEGIN
```

```
DimCaptionType := COPYSTR(CaptionExpr,1,CommaPosition - 1);
```

```
DimCaptionRef := COPYSTR(CaptionExpr,CommaPosition + 1);
```

```
CommaPosition := STRPOS(DimCaptionRef,',');
```

```
IF (CommaPosition > 0) THEN BEGIN
```

```
DimOptionalParam1 := COPYSTR(DimCaptionRef,CommaPosition + 1);
```

```
DimCaptionRef := COPYSTR(DimCaptionRef,1,CommaPosition - 1);
```

```
CommaPosition := STRPOS(DimOptionalParam1,',');
```

```
IF (CommaPosition > 0) THEN BEGIN
```

```
DimOptionalParam2 := COPYSTR(DimOptionalParam1,CommaPosition + 1);
```

```
DimOptionalParam1 := COPYSTR(DimOptionalParam1,1,CommaPosition - 1);
```

```
END ELSE BEGIN
```



```

        DimOptionalParam2 := ";
    END;
END ELSE BEGIN
    DimOptionalParam1 := ";
    DimOptionalParam2 := ";
END;
CASE DimCaptionType OF
'1': // Code Caption - Global Dimension using No. as Reference
    BEGIN
        CASE DimCaptionRef OF
            '1':
                BEGIN
                    IF Dim.GET(GLSetup."Global Dimension 1 Code") THEN
                        EXIT(DimOptionalParam1 + Dim.GetMLCodeCaption(Language) + DimOptionalParam2)
                    ELSE
                        EXIT(
                            DimOptionalParam1 +
                            GLSetup.FIELDCAPTION("Global Dimension 1 Code") +
                            DimOptionalParam2);
                    END;
                '2':
                    BEGIN
                        (same as case '1' for field "Global Dimension 2 Code")
                    END;
                END;
            END;
'2': // Code Caption - Shortcut Dimension using No. as Reference
    BEGIN
        CASE DimCaptionRef OF
            '1':
                BEGIN
                    IF Dim.GET(GLSetup."Shortcut Dimension 1 Code") THEN

```

End (2)      ↑

Begin (3)      ↓

```

EXIT(DimOptionalParam1 + Dim.GetMLCodeCaption(Language) + DimOptionalParam2)
ELSE
EXIT(
DimOptionalParam1 +
GLSetup.FIELDCAPTION("Shortcut Dimension 1 Code") +
DimOptionalParam2);
END;
'2':
BEGIN
(same as case '1' for field "Shortcut Dimension 2 Code")
END;
'3':
BEGIN
(same as case '1' for field "Shortcut Dimension 3 Code")
END;
'4':
BEGIN
(same as case '1' for field "Shortcut Dimension 4 Code")
END;
'5':
BEGIN
(same as case '1' for field "Shortcut Dimension 5 Code")
END;
'6':
BEGIN
(same as case '1' for field "Shortcut Dimension 6 Code")
END;
'7':
BEGIN
(same as case '1' for field "Shortcut Dimension 7 Code")
END;
'8':
BEGIN

```

(same as case '1' for field "Shortcut Dimension 8 Code")

```

    END;

    END;

    END;
'3': // Filter Caption - Global Dimension using No. as Reference
BEGIN
CASE DimCaptionRef OF
'1':
    BEGIN
        IF Dim.GET(GLSetup."Global Dimension 1 Code") THEN
            EXIT(DimOptionalParam1 + Dim.GetMLFilterCaption(Language) + DimOptionalParam2)
        ELSE
            EXIT(
                DimOptionalParam1 +
                GLSetup.FIELDCAPTION("Global Dimension 1 Code") +
                DimOptionalParam2);
        END;
    '2':
        BEGIN

```

(same as case '1' for field "Global Dimension 2 Code")

```

    END;

    END;

    END;
'4': // Filter Caption - Shortcut Dimension using No. as Reference
BEGIN
CASE DimCaptionRef OF
'1':
    BEGIN
        IF Dim.GET(GLSetup."Shortcut Dimension 1 Code") THEN
            EXIT(DimOptionalParam1 + Dim.GetMLFilterCaption(Language) + DimOptionalParam2)
        ELSE
            EXIT(
                DimOptionalParam1 +

```



```

        GLSetup.FIELDCAPTION("Shortcut Dimension 1 Code") +
        DimOptionalParam2);
    END;
'2':
    BEGIN
(same as case '1' for field "Shortcut Dimension 2 Code")
    END;
'3':
    BEGIN
(same as case '1' for field "Shortcut Dimension 3 Code")
    END;
'4':
    BEGIN
(same as case '1' for field "Shortcut Dimension 4 Code")
    END;
'5':
    BEGIN
(same as case '1' for field "Shortcut Dimension 5 Code")
    END;
'6':
    BEGIN
(same as case '1' for field "Shortcut Dimension 6 Code")
    END;
'7':
    BEGIN
(same as case '1' for field "Shortcut Dimension 7 Code")
    END;
'8':
    BEGIN
(same as case '1' for field "Shortcut Dimension 8 Code")
    END;
END;
END;

```

```

'5': // Code Caption - using Dimension Code as Reference
BEGIN
  IF Dim.GET(DimCaptionRef) THEN
    EXIT(DimOptionalParam1 + Dim.GetMLCodeCaption(Language) + DimOptionalParam2)
  ELSE
    EXIT(DimOptionalParam1);
  END;
'6': // Filter Caption - using Dimension Code as Reference
BEGIN
  IF Dim.GET(DimCaptionRef) THEN
    EXIT(DimOptionalParam1 + Dim.GetMLFilterCaption(Language) + DimOptionalParam2)
  ELSE
    EXIT(DimOptionalParam1);
  END;
END;
END;
EXIT("");

```



End (3)

### VATCaptionClassTranslate (ID 9)

If CAPTIONAREA equals 2, CaptionClassTranslate passes the CaptionExpr parameter CAPTIONREF, which is actually the VATCAPTIONTYPE, and calls VATCaptionClassTranslate. VATCaptionClassTranslate also passes the Language ID and the CAPTIONREF part of the *CaptionClass* property.

This function can be split into two main parts:

- 1 Sift out the comma separated subparts of the CAPTIONREF (see the previous description of the CAPTIONREF syntax.)
- 2 Determine what the caption should be, depending on the VATCAPTIONTYPE. In either case, the original caption is replaced by its original caption plus the string:
  - 'Excl. VAT' if VATCAPTIONTYPE equals 1.
  - 'Incl. VAT' if VATCAPTIONTYPE equals 2.

**Code**

```

VATCaptionClassTranslate(Language : Integer;CaptionExpr : Text[80]) : Text[30]

Begin (1)      CommaPosition := STRPOS(CaptionExpr, ',');

                IF (CommaPosition > 0) THEN BEGIN

                    VATCaptionType := COPYSTR(CaptionExpr,1,CommaPosition - 1);

End (1)        VATCaptionRef := COPYSTR(CaptionExpr,CommaPosition + 1);

Begin (2)      CASE VATCaptionType OF

                '0' : EXIT(COPYSTR(STRSUBSTNO('%1
                %2',VATCaptionRef,Text016),1,30));

                '1' : EXIT(COPYSTR(STRSUBSTNO('%1
                %2',VATCaptionRef,Text017),1,30));

End (2)        END;

                END;

                EXIT(");

```



## **Chapter 29**

### **Supporting Record Level Security**

This chapter explains how to implement record level security in the SQL Server Option for Dynamics NAV.

- Record Level Security

## 29.1 Record Level Security

The SQL Server Option for Dynamics NAV allows you to limit the access that users have to the information stored in the database by specifying that they can only access records that fulfill certain criteria in specific tables. This is called record level security and you implement it by placing security filters on the roles and permissions that you assign to your users.

One important thing to remember when using security filters is that the filters and the business logic of your application must go hand in hand if you are to avoid problems. Record level security has no influence on the business logic of your application but the business logic of your application can have a great influence on record level security and the filters that you can apply. Record level security is only useful when it compliments/supplements your business logic.

### Implementing Record Level Security

Once you have set up your record level security filters, Dynamics NAV automatically applies them in most situations. This means that the user only receives an error message if they manually attempt to access data that is outside the range of the security filters that have been defined for them.

When a form is opened from a command button or menu item, C/SIDE automatically applies record level security filters to the main record variable used in the form, provided that the command button or menu item in question uses properties to run the form, and not code.

Similarly, when a report or dataport is opened from a command button or menu item, C/SIDE automatically applies record level security filters to all the record variables used in the request filter tabs, provided that the command button or menu item in question uses properties to run the report or dataport, and not code.

C/SIDE does not apply record level security filters to user defined global and local variables. So to help users stay within the defined security filters you must include the appropriate statements in the code that applies the filters. Security filters are applied on a record variable by using the `SETPERMISSIONFILTER` function that is available for the record variable.

#### Example

This example finds the first record within your read permission security filter:

```
Customer.SETPERMISSIONFILTER;
Customer.FIND( '- ' );
```

#### Note

Record level security filters affect performance in the same way as any other filters that are applied by the user. It is important that the record level security filters have matching keys in tables that contain a large number of records, and that these keys are used. C/SIDE does not automatically select the most effective key to use. If, for example, a security filter specifies that a user is only allowed to see records that he

created himself by placing a filter on a field called **User ID**, the matching key must start with User ID. Furthermore, to ensure the best performance, the user must select the User ID key when opening the form for the first time. In some situations you can choose to change the default sorting in the form, or in the command buttons, menu items and code that opens the form.



### Forms, Reports and Dataports

As stated earlier, when a form, report or dataport is opened from a command button or menu item, C/SIDE automatically applies record level security filters provided that the command button or menu item in question uses properties to run the form, and not code. This means that as long as the form in question is a simple one the security filters that you apply at user level will be enough to ensure that the degree of security that you want is implemented.

Find('+') and  
calcsms

However, if the report or form contains some complex C/AL code – for example, Find('+') – or contains a calcsms, the security filters that you have applied at user level will probably not be enough to ensure that the form or report can run successfully. In which case, the user will probably receive an error message informing them that they do not have read permission to a particular table.

This situation generally arises because the form or report in question contains functionality that requires access to *all* the records in the table and this conflicts with the user's security filters that only give them access to *some* of the records in the table. This can happen, for example, if the form needs to read the last record in the table before it can insert a new record and the last record in the table lies outside the range of the security filter that has been applied to the user.

There is, however, a relatively simple method of overcoming this difficulty. The user has already been assigned a role that gives them permission to access the records in the table. This role was then limited to some of the records in the table by applying the security filter that conflicts with the functionality of the form. To resolve this conflict, you must augment the users permissions by assigning them a new role that gives them *indirect* read access to all the records in the table.

Now, when the user runs the form, the code will be able to read the last record in the table even though it is outside the range specified by the user's security filter. The fact that the user has been given indirect permission to read the entire table does not compromise the integrity of the security system that you are enforcing by implementing record level security. The user is still unable to see the records that are outside the range of the security filter, but the application can perform the calculations and read all the data that is necessary for it to function correctly and give the user the information they require.

This solution works because security permissions are added together and are not mutually exclusive.

### Codeunits

A typical Dynamics NAV application also contains forms and reports that include more complex functionality that requires the form or report to run individual codeunits or chains of codeunits. These codeunits can contain code that accesses many different tables.

In Dynamics NAV, codeunits are independent objects that have their own permissions. They are not assigned roles but rather given permission to access various tables. A codeunit's permissions are part of its properties. These permissions are indirect because codeunits can only access an object after they have been activated by a user who has permission to run the form or report that calls the codeunit.

When a user runs a form that calls a codeunit there is no guarantee that the security filter(s) that have been applied to the user's permissions are compatible with the permissions that have been defined for the codeunit. If they are incompatible the user will receive an error message informing them that they don't have permission to access to a particular table.

To resolve this conflict, you must ensure that the user's security filters are compatible with the permissions defined for the codeunit. To do this, you must assign the user a new role that gives them *indirect* read access to all the records in the table and then open the codeunit in the Object Designer, open the **Properties** window and give the codeunit read permission to the table in question.

However, the error message will not always be so informative. In this case, you must use the Debugger to identify the point at which the code errors. This will help you identify the table involved and you can then give the codeunit read permission to it. You should then run the form again. If you receive another error message, you can repeat this procedure until the form runs successfully.

It is also possible that your license does not allow you to modify the codeunits in question. In this case, you must contact your local Microsoft Business Solutions Partner.

### Guidelines for Implementing Record Level Security

With all this in mind we can formulate some general rules for implementing record level security:

- Record level security and the business logic of the application must work together.
- For forms, reports and dataports that are run by properties and not code, the security filters that you apply at user level are sufficient.
- For forms, reports and dataports that run code, assign an extra indirect read permission at user level.
- For codeunits, assign the extra indirect read permissions at user level and at object level.
- Permissions are cumulative and not mutually exclusive.

**Important:**

.....  
Record level security filters do not support wildcards. This means that you cannot use \* and ? in the filters. You can use the other symbols, delimiters and operators, such as <, >, |, &, .. and =.

The maximum length of a security filter is 250 characters, but all of the delimiters, symbols and operators such as <, >, |, &, .. and = also count as characters and can considerably reduce the length of the security filters that you can enter.

Furthermore, security filters are concatenated and therefore the sum of all the security filters applied to a user or a role cannot exceed 250 characters.

.....



### Applying Security Filters in a Posting Scenario

As explained earlier, applying security filters in situations where the functionality involves running one or more codeunits is quite tricky. A typical example of this would be a posting procedure. The security filters are applied to the user in the normal way and must then be supplemented with extra indirect read permissions at user level and at object level.

The following example demonstrates how to set security filters that limit the permissions that a user has to certain functionality and illustrates the complexity of this process.

This example involves a user that can post in the G/L Journal. The user has been given the following roles:

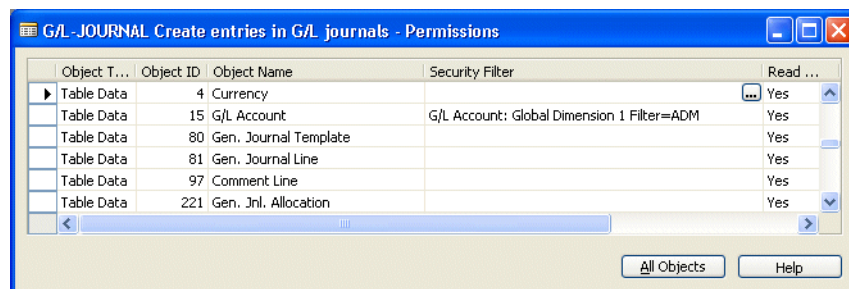
Role	Description
All	All Users
G/L-Journal	Create entries in G/L journals
G/L-Journal, Post	Post G/L journals
Payment Terms	A role created for this example that gives access to the <b>Payment Terms</b> table.

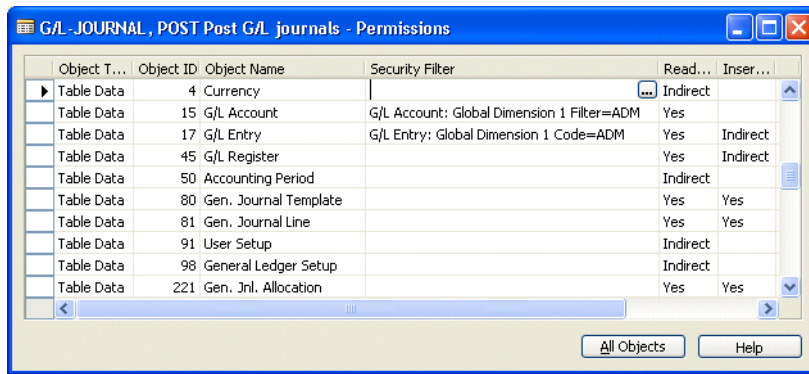
The first three roles are standard Dynamics NAV roles and the last one is a new role that has been created for this example. For this example we have created a user called *TestUser* who has been assigned these roles.

**Note**

To complete this example you must be able to log on as two different users – *TestUser* who wants to post in the G/L Journal and an administrator who modifies the roles and permissions that *TestUser* has been assigned. The administrator has been given the SUPER role. It is also important to remember that *TestUser* must log off and on again every time the administrator modifies the permissions system.

In this scenario, you only want *TestUser* to be able to post entries that relate to the Administration department (Global Dimension 1). You must therefore apply security filters to the roles you have given *TestUser*. You must apply these filters to all the roles that give *TestUser* permission to access the **G/L Account** and **G/L Entry** tables:



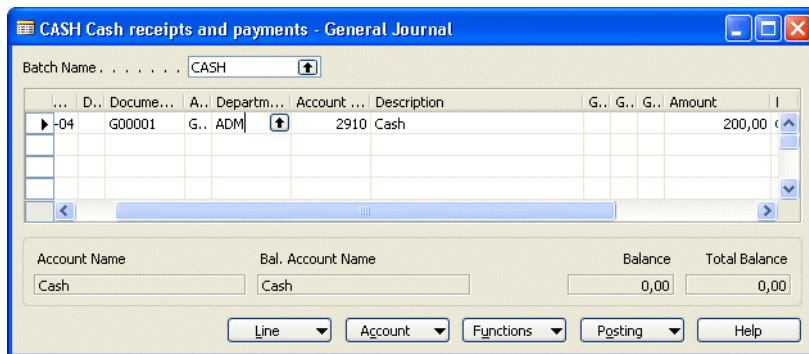


Let's see what happens when *TestUser* tries to post an entry in the General Journal.

**Note**

The aim of this example is to ensure that *TestUser* can only post entries in the General Journal that refer to the Administration department. You must therefore make sure that the **Department Code** field is visible in the **General Journal** window.

- 1 Log on to the database as *TestUser* and open the General Ledger and click General Journals to open the **General Journal** window.
- 2 In the **Batch Name** field select CASH and enter the information that you want to post. Remember to select ADM in the **Department Code** field:



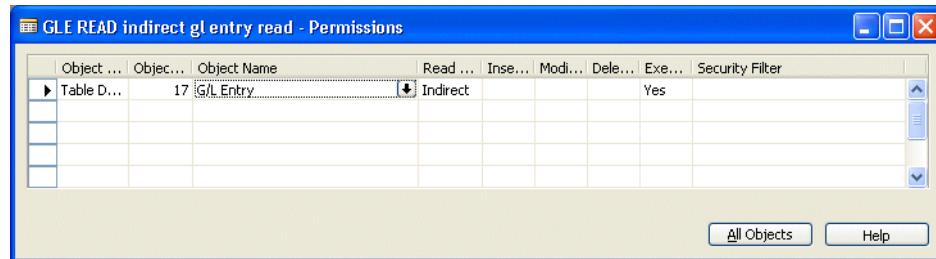
If the **Department Code** field is not visible, click View, Show Column to open the **Show Column** window and enter a check mark beside the **Department Code** field.

- 3 Click Posting, Post (F11) to post the entry. You receive an error message informing you that you don't have read permission to the **G/L Entry** table.

This occurs because each time you post a new entry it must be given a unique ID. Before it can create this ID, Dynamics NAV must know the ID of the last record in the **G/L Entry** table. If the last entry in the table is not for the ADM department, *TestUser* is not allowed to read it and this is why the error message says that *TestUser* does not have read permission to the **G/L Entry** table. If the last record was for the ADM department *TestUser* would be able to read it and would not receive this error message.

To overcome this problem, you must give *TestUser* indirect read permission to the entire **G/L Entry** table.

To do this, log on to the database as the administrator and create an extra role that only contains this permission and give it to *TestUser*:

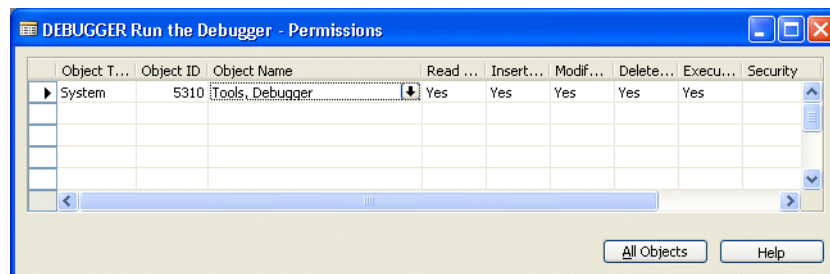


Remember to synchronize the security system after altering the permission system.

Log on to the database as *TestUser* and try to post the entry again.

Unfortunately, you still receive the same error message telling you that you don't have read permission to the **G/L Entry** table. This occurs because the functionality of the form requires it to run a few codeunits and one or more of these codeunits needs to be able to read all the records in the **G/L Entry** table. To identify which codeunits are causing the error, you must use the Debugger to debug the code as the form is run.

You must therefore give *TestUser* permission to run the Debugger. To do this create an extra role that only contains this permission and give it to *TestUser*:



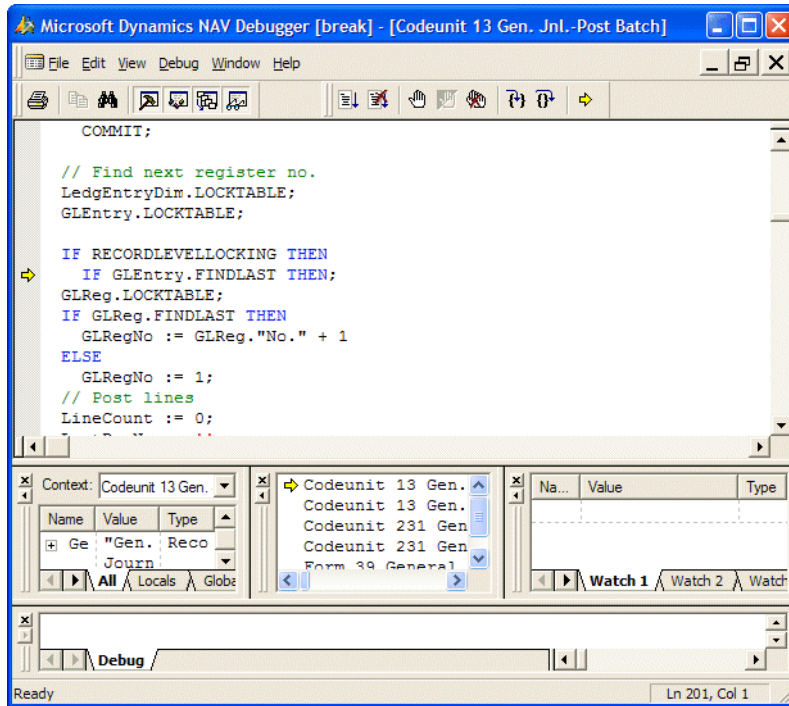
Remember to synchronize the security system after altering the permission system.

Debugging the code To debug the code and identify the codeunits that are causing the error:

- 1 Log on to the database as *TestUser* and open the **General Journal** window.
- 2 In the **Batch Name** field select CASH and enter the information that you want to post. Remember to select ADM in the **Department Code** field. Don't post the entry yet.
- 3 Click Tools, Debugger, Breakpoint on Triggers to remove the check mark from this option. The debugger will only stop when the code encounters an error.
- 4 Click Tools, Debugger, Active to activate the Debugger.
- 5 In the **General Journal** window, click Posting, Post.

The application runs a little slower now because the Debugger is running in the background.

After a while the Debugger opens and shows you where it encountered the first error:

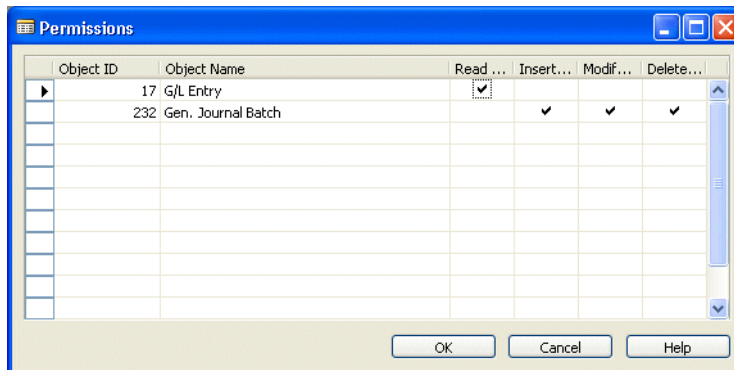


As you can see from this picture, the first error occurred in codeunit 13, **Gen. Jnl.-Post Batch** when the code tried to read the last record – **FINDLAST** – in the **G/L Entry** table.

To solve this, you must give codeunit 13 permission to read all the records in the **G/L Entry** table.

To give the extra permission to the codeunit:

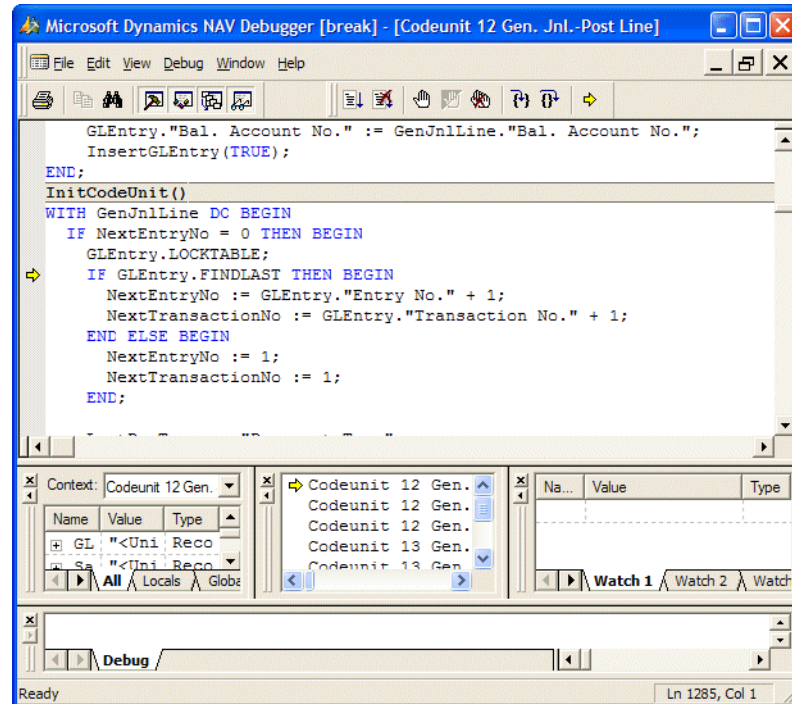
- 1 In the Object Designer, open codeunit 13, **Gen. Jnl.-Post Batch** and open the **Properties** window.
- 2 In the **Value** field of the **Permissions** property, click the AssistButton ... to open the **Permissions** window for the codeunit.
- 3 Add the **G/L Entry** table to the list and make sure that the codeunit only has read permission to this table:



- 4 Click OK to commit the changes that you have made in this window.
- 5 Close, save and compile the codeunit.

You do not need to synchronize the security system after altering the properties of the codeunit as it is not part of the permission system.

Now log on to the database as *TestUser*, activate the Debugger and try to post the entry again. The code encounters another error:



In the **Debugger** window, you can see that this time the error occurred in codeunit 12, **Gen. Jnl.-Post Line** when the code tried to read the last record – **FINDLAST** – in the **G/L Entry** table.

To solve this, you must repeat the procedure described earlier and give codeunit 12, **Gen. Jnl.-Post Line** permission to read all the records in the **G/L Entry** table.

Log on to the database as *TestUser*, activate the Debugger and try to post the entry again. This time the entry was posted correctly.

Turn off the Debugger and try to post an entry for another department to see what happens. Now you receive a different error message informing you that you don't have **write** permission to the **G/L Entry** table.

Well, you know that this is not quite true because you have just posted the previous entry. However, if the message was more detailed, it might reveal too much. What it means is that your security filters work and you can only post entries for the **ADM** department.

### Security Filters and Complex Forms

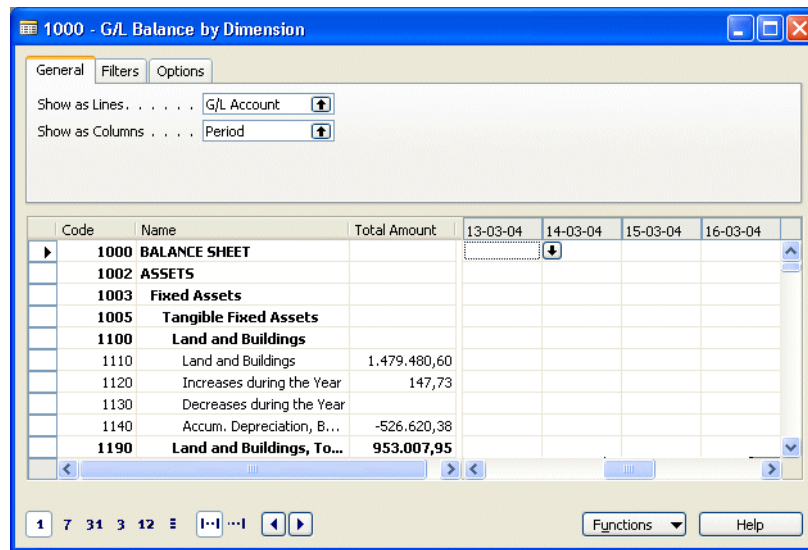
As mentioned earlier, C/SIDE does not apply record level security filters to user defined global and local variables. This means that if your application contains complex forms that use variables to access tables containing sensitive data, the security filters that you set will not be applied to these variables. To ensure that users cannot access the sensitive data, you must modify the code that these forms contain.

In the example that you have just created, you know that *TestUser* can only post entries for the *ADM* department. However, you still need to investigate whether or not *TestUser* can read any data from the other departments.

Log on to the database as *TestUser* and open some of the forms in the General Ledger and check the data that is displayed. Remember, it is the complex forms that are most likely to use variables.

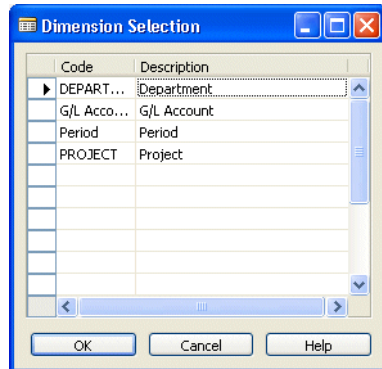
To test one such form:

- 1 In the General Ledger, click Chart of Accounts to open the **Chart of Account** window. This window only shows data for the *ADM* department.
- 2 In the **Chart of Account** window, click Balance, G/L Balance by Dimension to open the **G/L Balance by Dimension** window:



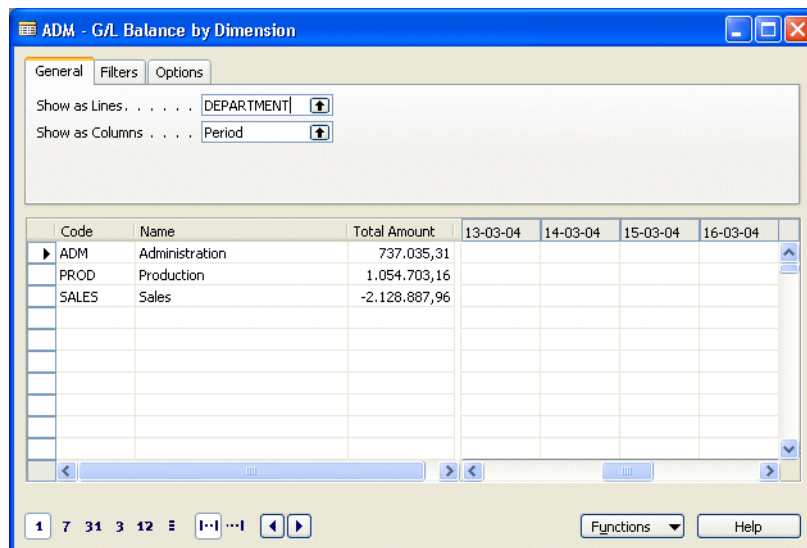
This is a matrix form that displays a summary of the balances for all the accounts in the chart of accounts. Furthermore, the list of balances can be presented in several ways by applying different filters.

- In the **Show as Lines** field, use the AssistButton ↑ to open the **Dimension Selection** window:



In this window, you select the different dimensions that you use to filter the data displayed in the **G/L Balance by Dimension** window.

- Select Department and click OK to return to the **G/L Balance by Dimension** window:



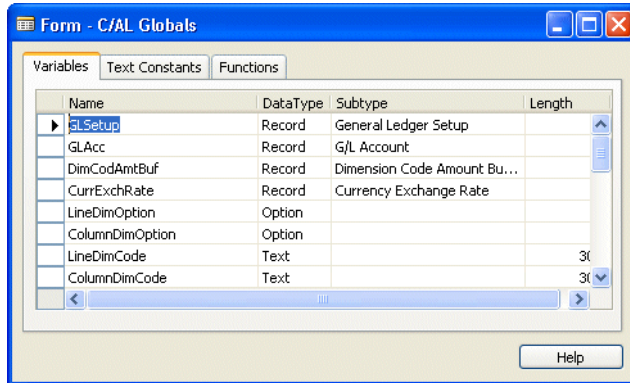
As you can see this window now displays three totals – one for each department. Unfortunately, *TestUser* is only supposed to be able to see the total for the *Administration* department. However, the situation is not as bad as it might seem.

In the **Total Amount** field for the *Sales* department, click the AssistButton ↓ and the **General Ledger Entries** window opens. This window is empty which means that some of the security filters you have implemented are being applied and *TestUser* cannot see the G/L entries for the other departments.

Nonetheless, *TestUser* is not supposed to be able to see all the totals in the **G/L Balance by Dimension** window. To remedy this, you must investigate the code that this form contains to see if there are any places where the code should be improved so that it can support security filters.

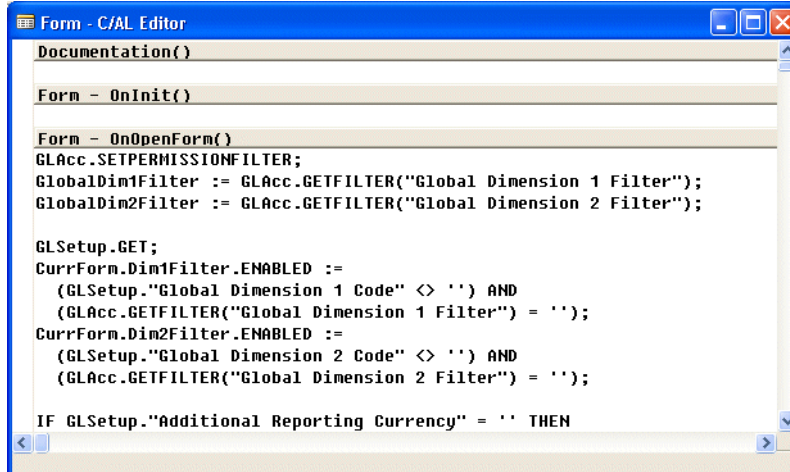
Looking at the code To investigate the code that this form contains:

- 1 Log on to the database as an administrator and open the **G/L Balance by Dimension** window.
- 2 Click CTRL + F2 to open the form in the Designer.
- 3 Click View, C/AL Globals to open the **C/AL Globals** window:



As you can see, this window contains a global variable called *GLAcc* that refers to the **G/L Account** table.

- 4 Click View, C/AL Code to open the C/AL Editor and see the code that the form contains:



The first line of code in the *OnOpenForm* trigger is:

```
GLAcc.SETPERMISSIONFILTER;
```

This ensures that any security filters that have been set up for the current user are applied to this global variable in the code.

However, the matrix form that we are investigating contains functionality that allows you to change the filters that are used on the data. This means that the filters are cleared and reset. Furthermore, the C/AL language contains a function called `RESET` that is used to clear any filters that are set so that new filters can be set.



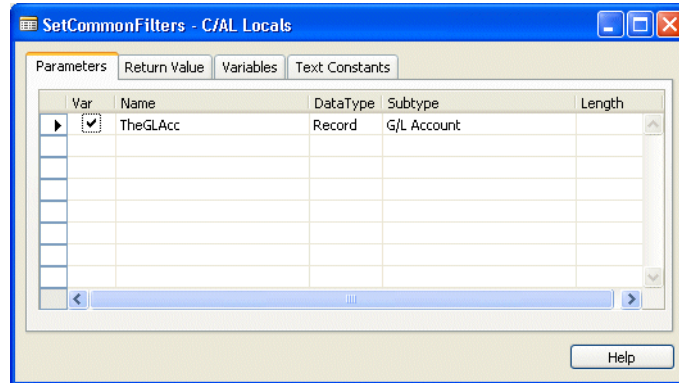
With this in mind, it might be a good idea to see if the code in the **G/L Balance by Dimension** form uses the `RESET` function.

- 5 Use the **Find** window to locate all the places where the `RESET` function is called.

It occurs in two places – in the *SetCommonFilters* function and in the *CalcAmount* function. Both of these are user-defined functions.

- 6 Take a look at the local variables that have been declared for these functions.

The *SetCommonFilter* function seems to be the most promising:



As you can see, it contains a parameter that calls the **G/L Account** table by reference.

- 7 Take a closer look at the code in these functions:

```

SetCommonFilters(VAR TheGLAcc : Record "G/L Account")
CLEAR(TheGLAcc);
WITH TheGLAcc DO BEGIN
  IF DateFilter = '' THEN
    DateFilter2 := ExcludeClosingDateFilter
  ELSE BEGIN
    IF AmountType = AmountType::"Net Change" THEN BEGIN
      DateFilter2 := DateFilter;
    END ELSE BEGIN
      SETFILTER("Date Filter",DateFilter);
      DateFilter2 := STRSUBSTNO('..%1',GETRANGEMAX("Date Filter"));
    END;
    IF ExcludeClosingDateFilter <> '' THEN
      DateFilter2 := DateFilter2 + '&' + ExcludeClosingDateFilter;
  END;
  RESET;
  IF GLAccFilter <> '' THEN
    SETFILTER("No.",GLAccFilter)
  ELSE
    SETRANGE("No.");
  IF GLAccFilter <> '' THEN

```

```

TextBox - C/AL Editor
PeriodInitialized := FALSE;

CalcAmount(SetColFilter : Boolean) : Decimal
IF SetColFilter THEN
    ColumnCode := CurrForm.Matrix.MatrixRec.Code
ELSE
    ColumnCode := '';
IF DimCodAmtBuf.GET(Code,ColumnCode) THEN
    EXIT(DimCodAmtBuf.Amount);

GLAcc.RESET;
SetCommonFilters(GLAcc);
SetDimFilters(GLAcc,0);
IF SetColFilter THEN
    SetDimFilters(GLAcc,1);
    
```

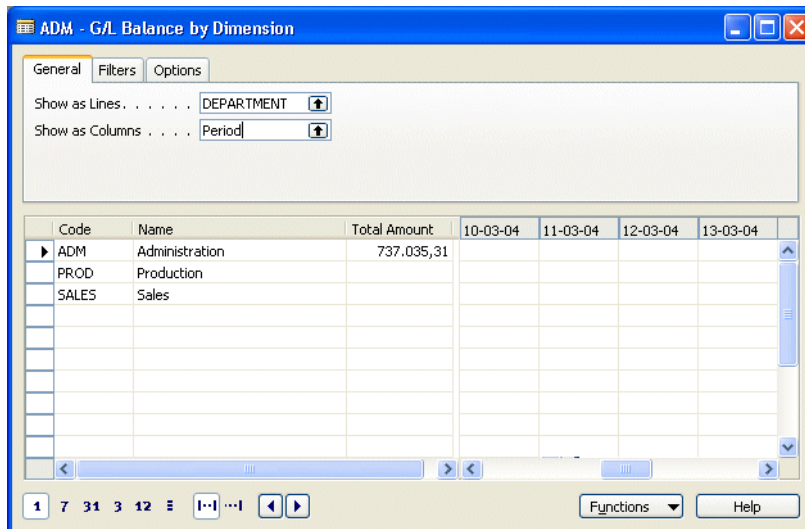
In the *CalcAmount* function, you can see that immediately after the code calls the `RESET` function on the *GLAcc* global variable, it calls the *SetCommonFilters* function.

All of this suggests that the instance of the `RESET` function that is most relevant to us is the one that occurs in the *SetCommonFilters* function.

- 8 Enter the following code immediately after the `RESET` function in the *SetCommonFilters* trigger:

```
GLAcc.SETPERMISSIONFILTER;
```

- 9 Close, save and compile the form.
- 10 Log on to the database again as *TestUser* and open the **G/L Balance by Dimension** window. Remember to select *Department* in the **Show as Lines** field:



The form no longer displays data that belongs to the other departments.

You have now successfully implemented record level security in a posting situation and ensured that the user only has read access to the data specified by their security filters.

## **Chapter 30**

### **Performance**

This chapter covers features built into C/SIDE to increase performance, such as the DBMS cache, the commit cache and the command buffer. It also contains a section on how keys and queries can affect performance.

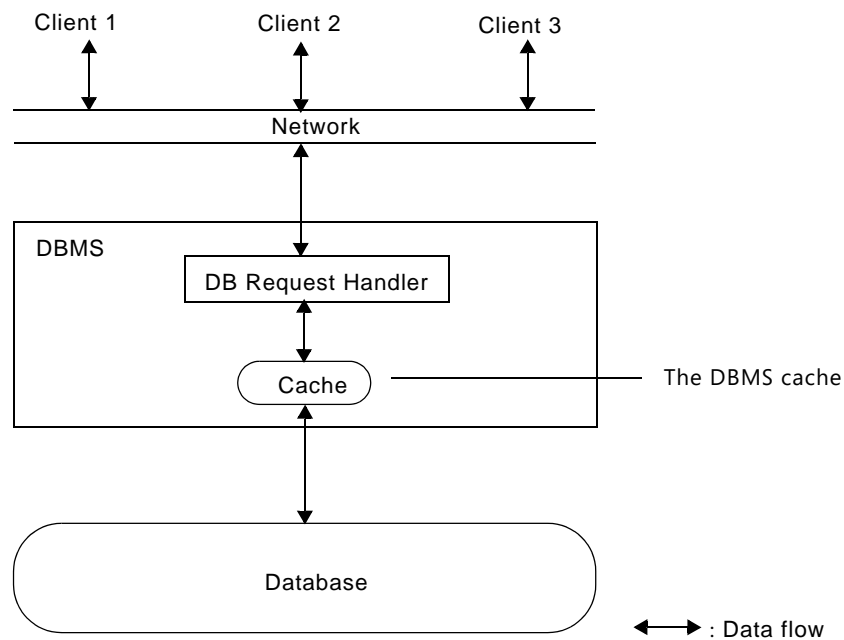
- The DBMS Cache
- The Commit Cache
- The Command Buffer
- Keys, Queries and Performance
- C/AL Database Functions and Performance on SQL Server
- Configuration Parameters
- Login Stored Procedure on the SQL Server Option

### 30.1 The DBMS Cache

The Database Management System (DBMS) is a memory buffer that stores copies of portions of the database that the DBMS is currently using. Reading from memory is much faster than reading from the disk. The DBMS therefore returns a record more quickly if it is already stored in cache. As long as the required data is stored in cache, the data appears to be immediately available. When the required data is not stored in cache, it must be copied from the disk and then stored in cache.

The DBMS cache is transparent to the user. For example, when a client or user requests data, the data is automatically copied into the cache and stored there. If the data is modified, it is automatically copied back to the physical disk(s). These data transfers take place automatically. The user does not need to know about the cache.

The following figure illustrates three clients that send requests to the DBMS. When, for example, Client 2 sends a request to read data from the database, the request handler determines whether the desired data can be fetched directly from the cache or whether it must be fetched from a disk.



At the same time, another client can be modifying a record in a table in the database. The modified data will be written to the DBMS cache, and not to the disk. When this client completes the write transaction (that is, commits the changes), the data in the cache that was modified during the transaction is written to the disk. The cache is then said to be *flushed*.

The DBMS cache always contains the most recently used data. The cache is continually updated with the relevant data from the database.

The size of the cache greatly affects performance. When you set the size of the cache, you must remember two simple rules:

- The more memory you assign to the cache, the more efficient it becomes. (Of course, there is no reason to assign more memory to the cache than the total size of your database.)
- The size of the cache must not exceed the amount of physical memory available on your system. This is because it may cause the operating system to swap the cache memory in and out of the disk. This will considerably slow down the overall speed of the C/SIDE system.

**Note**

.....

You must remember to specify the `commitcache=yes` server parameter in the command line to enable the caching of write transactions. See the next section for more information.

.....

See the section "C/SIDE Specifications" on page 577 for information about the maximum cache size.

### 30.2 The Commit Cache

The commit cache is a special write buffer for the disk(s) in the system. The commit cache has been designed to:

- quickly absorb committed transactions from the DBMS. This frees the DBMS to perform other tasks.
- enable asynchronous disk writes.
- enable parallel disk read and write operations when multiple disks are used.
- guarantee that the disk file is always consistent.

The commit cache is placed between the DBMS and the database. It absorbs committed transactions from the DBMS. When the commit cache receives a committed transaction, it writes the data to the disk(s). Thus the DBMS can perform other tasks while the commit cache writes to the disk. The data is said to be written asynchronously to the disk. This is because the disk write does not occur at the same time as the DBMS commits the transaction.

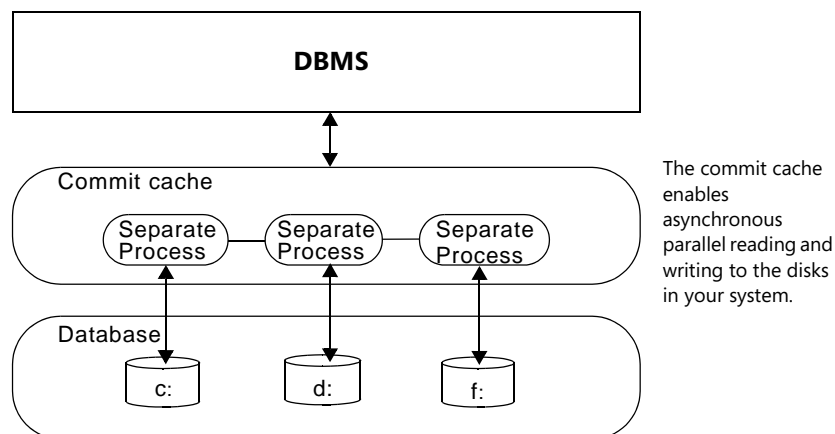
As described in the section "The Physical and the Logical Database" on page 49, the logical database can be stored in several distinct disk files (which can be stored on separate disks). When more than one disk is used to store the database, each of these disks is controlled by separate commit cache processes, which are linked together to both enable and control (asynchronous) parallel read and write operations.

The commit cache ensures that the database file is consistent even if a power failure occurs during a write operation to the disk. However, if a power failure occurs, you lose all the committed transactions that are currently contained in the commit cache.

**Note**

.....  
 You should not use advanced disk caches with delayed write back (sometimes called lazy write). The use of such cache systems may corrupt your database file(s).  
 .....

The following figure illustrates a database that is stored on three physical disks. Each disk is controlled by its own commit cache process. These processes are connected to enable parallel reading and writing.



## 30.3 The Command Buffer

The command buffer only applies to C/SIDE Database Server, and is placed as a link between your application and the DBMS. It is a temporary storage that can hold requests (C/AL database commands) sent from your application to the DBMS. The command buffer has been designed to reduce the number of network transfers when using C/SIDE in local area network (LAN) environments.

When an application performs a write transaction, some requests such as inserting a record in a table (using `record.INSERT()`) need not be sent to the DBMS at once. They can be temporarily stored in a command buffer. In general, the commands that don't have to be sent to the DBMS immediately are the ones that don't have to return a value.

### Note

.....

The contents of the command buffer are sent to the DBMS when the buffer is full or when a command requires an immediate response from the DBMS.

.....

The advantage of assembling DBMS commands into packages is that the number of network transfers is reduced (that is, the load on the LAN is reduced). This is because the time required to send one DBMS request is comparable to the time used to send an entire package.

The following C/AL code sample illustrates how the command buffer affects the number of network transfers.

```
WHILE Record.FIND('-') DO
    Record.DELETE();
```

Two commands are executed for each record in the table. However, each record causes only one request to be sent to the DBMS. This is because the `DELETE` command is stored in the command buffer until the `FIND` command is executed.

### Debugging

The system automatically turns off the command buffer when you activate the C/AL debugger. This can lead to some confusion if you are not aware of this fact.

The following statements (the complete contents of an imaginary codeunit) illustrate the difference between running code with and without the debugger:

```
Customer."No." := '12';
Customer.DELETE();
First := 7;
Second := 0;
Ratio := First / Second;
```

If there is no Customer with the number 12, a runtime error occurs irrespective of whether the debugger is active or not. However, the error that occurs will not be the same. There are two errors here: since the Customer cannot be found, the `DELETE` will also fail. Furthermore, the last statement is a division by zero.

When the debugger is *inactive*, the `DELETE` command is stored in the command buffer for execution at a later time. Therefore, a runtime error will occur when the last statement tries to divide by zero.

When the debugger is *active*, the `DELETE` command is executed immediately. This causes a runtime error when the record for Customer number 12 cannot be found.



## 30.4 Keys, Queries and Performance

When you write a query that searches through a subset of the records in a table, you should always carefully define the keys both in the table and in the query so that Dynamics NAV can quickly identify this subset. For example, the entries for a specific customer will normally be a small subset of a table containing entries for all the customers.

If Dynamics NAV can locate and read the subset efficiently, the time it will take to complete the query will only depend on the size of the subset. If Dynamics NAV cannot locate and read the subset efficiently, performance will deteriorate. In the worst case scenario, Dynamics NAV will read through the entire table and not just the relevant subset. In a table containing 100,000 records, this could mean taking a few milliseconds or several seconds to answer the query.

To maximize performance, you must define the keys in the table so that they facilitate the queries that you will have to run. These keys must then be specified correctly in the queries.

For example, you would like to retrieve the entries for a specific customer. To do this, you apply a filter to the **Customer No.** field in the **Cust. Ledger Entry** table. In order to run the query efficiently on SQL Server, you must have defined a key in the table that has **Customer No.** as the first field. You must also specify this key in the query.

The table could have these keys:

```
Entry No.
Customer No., Posting Date
```

The query could look like this:

```
SETCURRENTKEY("Customer No.");
SETRANGE("Customer No.", '1000');
IF FIND('-') THEN
  REPEAT
  UNTIL NEXT = 0;
```

You should define keys and queries in the same way when you are using C/SIDE Database Server. However, C/SIDE Database Server can run the same query almost as efficiently if **Customer No.** is not the first field in the key. For example, if you have defined a key that contains **Country/Region Code** as the first field and **Customer No.** as the second field and if there are only a few different country/region codes used in the entries, it will only take a little longer to run the query.

The table could have these keys:

```
Entry No.
Country/Region Code, Customer No., Posting Date
```

The query could look like this:

```
SETCURRENTKEY("Country/Region Code", "Customer No.");
SETRANGE("Customer No.", '1000');
IF FIND('-') THEN
  REPEAT
  UNTIL NEXT = 0;
```

But SQL Server will not be able to answer this query efficiently and will read through the entire table.

In conclusion, SQL Server makes stricter demands than C/SIDE Database Server on the way that keys are defined in tables and on the way they are used in queries. You should therefore define your keys and queries with SQL Server in mind, as this will ensure that your application can run just as efficiently on both server options.

## 30.5 C/AL Database Functions and Performance on SQL Server

The fastest SQL statement that Dynamics NAV sends to SQL Server runs slower than most database functions on C/SIDE Database Server. However, one SQL statement can sometimes cover several database server calls. The following section describes the relationship between some basic database functions in C/AL and SQL statements.

Each `GET` (or `FIND( '=' )`) requires a separate SQL statement, unless the client has already retrieved the record in question during a recent operation. This means that if the client reads the same record several times, SQL Server will only be called the first time that the client needs to read the record.

Each `FIND( '-/+ ' )` requires a separate SQL statement, unless the client has executed the same query (filters etc.) in a recent operation.

Each `NEXT` (or `FIND( '>/< ' )`) requires at least one, but often several, SQL statements. However, when `NEXT` is used together with `FIND( '-/+ ' )` to read a set, as shown in the following example, one SQL statement can cover the needs of all the `NEXT` function calls in the loop:

```
IF FIND( '-' ) THEN
  REPEAT
  UNTIL NEXT = 0;
```

Reading the set backwards with `FIND( '+' )/NEXT(-1)` or using `"ASCENDING := FALSE"` is equally efficient. You should not read record sets by using `"WHILE FIND( '-/+ ' ) DO"` or any similar constructions.

Each `CALCFIELD/CALCSUMS` that calculates a sum requires a separate SQL statement, unless the client has calculated the same sum or another sum that uses the same `SumIndex`, filters etc., in a recent operation. In other words, totals for all the `SumIndexFields` in a `SumIndex` are calculated when a sum is required for one of them, and all the sums are stored in the client's cache.

Each `INSERT/MODIFY/DELETE` requires a separate SQL statement. If the table that you modify contains `SumIndexes`, the operations will be considerably slower. As a test, select a table that contains `SumIndexes` and execute a hundred of these `INSERT/MODIFY/DELETE` operations to measure how long it takes to maintain the table and all its `SumIndexes`.

`LOCKTABLE` does not require any separate SQL statements. It only causes any subsequent reading from the table to lock the table or parts of it.

### Database Administration, Object Design and Performance on SQL Server

It is much slower to create tables and companies on SQL Server than on C/SIDE Database Server. Similarly, translating and renaming tables and fields are slower on SQL Server.

## 30.6 Configuration Parameters

You can configure a Dynamics NAV database by creating a SQL Server table configuration parameter table and entering parameters into the table that will determine some of the behavior of Dynamics NAV when it is using this database.

In the database create a table, owned by dbo:

```
CREATE TABLE [${ndo$dbconfig}] (config VARCHAR(512) NOT NULL)

GRANT SELECT ON [${ndo$dbconfig}] TO public
```

(You can add additional columns to this table, if necessary. The length of the config column should be large enough to contain the necessary configuration values, as explained in the following, but need not be 512.)

There is one record in this table for each parameter that is required.

The following sections describe the parameters that you can enter into this table.

### Index Hinting

It is possible to force SQL Server to use a particular index when executing queries for `FIND( '-' )`, `FIND( '+' )`, `FIND( '=' )` and `GET` statements. This can be used as a workaround when SQL Server's Query Optimizer picks the wrong index for a query.

Index hinting can help avoid situations where SQL Server's Query Optimizer chooses an index access method that requires many page reads and generates long-running queries with response times that vary from seconds to several minutes. Selecting an alternative index can give instant 'correct' query executions with response times of milliseconds. This problem usually occurs only for particular tables and indexes that contain certain data spreads and index statistics.

In the rare situations where it is necessary, you can direct Dynamics NAV to use index hinting for such problematic queries. When you use index hinting, Dynamics NAV adds commands to the SQL queries that are sent to the server. These commands bypass the normal decision making of SQL Server's Query Optimizer and force the server to choose a particular index access method.

#### Note

.....

This feature should only be used after all the other possibilities have been exhausted, for example, updating statistics, optimizing indexes or re-organizing column order in indexes.

.....

The index hint syntax is:

```
IndexHint=<Yes,No>;Company=<company name>;Table=<table
name>;Key=<keyfield1,keyfield2,...>; Search Method=<search method
list>;Index=<index id>
```

Each parameter keyword can be localized in the "Driver configuration parameters" section of the `.stx` file.

The guidelines for interpreting the index hint are:

- If `IndexHint=No`, the entry is ignored.
- All the keywords must be present or the entry is ignored.
- If a given keyword value cannot be matched the entry is ignored.
- The values for the company, table, key fields and search method must be surrounded by double-quotes to delimit names that contain spaces, commas etc.
- The table name corresponds to the name supplied in the Object Designer (not the Caption name).
- The key must contain all the key fields that match the required key in the **Keys** window in the Table Designer.
- The search method contains a list of search methods used in `FIND` statements, that must be one of '-', '+', '=', or '!' (for the C/AL `GET` function).
- The index ID corresponds to a SQL Server index for the table: 0 represents the primary key; all other IDs follow the number included in the index name for all the secondary keys. Use the SQL Server command `sp_helpindex` to get information about the index ID associated with indexes on a given table. In this example we are looking for index information about the **Item Ledger Entry** table:

```
sp_helpindex 'CRONUS International Ltd_$Item Ledger Entry'
```

When Dynamics NAV executes a query, it checks whether or not the query is for the company, table, current key and search method listed in one of the `IndexHint` entries. If it is, it will hint the index for the supplied index ID in that entry.

Note that:

- If the company is not supplied, the entry will match all the companies.
- If the search method is not supplied, the entry will match all the search methods.
- If the index ID is not supplied, the index hinted is the one that corresponds to the supplied key. This is probably the desired behavior in most cases.
- If the company/table/fields are renamed or the table's keys redesigned, the `IndexHint` entries must be modified manually.

Here are a few examples that illustrate how to add an index hint to the table by executing a statement in Query Analyzer:

#### Example 1

```
INSERT INTO [$ndo$dbconfig] VALUES
('IndexHint=Yes;Company="CRONUS International Ltd.";Table="Item
Ledger Entry";Key="Item No.", "Variant Code";Search Method="-
+";Index=3')
```

This will hint the use of the 3 index of the *CRONUS International Ltd\_\$Item Ledger Entry* table for `FIND( '-')` and `FIND( '+')` statements when the *Item No., Variant Code* key is set as the current key for the **Item Ledger Entry** table in the CRONUS International Ltd. company.

#### Example 2

```
INSERT INTO [$ndo$dbconfig] VALUES
```

```
( 'IndexHint=No;Company="CRONUS International Ltd.";Table="Item
Ledger Entry";Key="Item No.", "Variant Code";Search Method="-
+";Index=3' )
```

The index hint entry is disabled.

### Example 3

```
INSERT INTO [$ndo$dbconfig] VALUES

( 'IndexHint=Yes;Company="CRONUS International Ltd.";Table="Item
Ledger Entry";Key="Item No.", "Variant Code";Search Method="-
+";Index=' )
```

This will hint the use of the *Item No., Variant Code* index of the *CRONUS International Ltd\_ \$Item Ledger Entry* table for `FIND( '-' )` and `FIND( '+' )` statements when the *Item No., Variant Code* key is set as the current key for the ***Item Ledger Entry*** table in the CRONUS International Ltd. company.

This is probably the way that the index-hinting feature is most commonly used.

### Example 4

```
INSERT INTO [$ndo$dbconfig] VALUES

( 'IndexHint=Yes;Company=;Table="Item Ledger Entry";Key="Item
No.", "Variant Code";Search Method="-+";Index=3' )
```

This will hint the use of the \$3 index of the *CRONUS International Ltd\_ \$Item Ledger Entry* table for `FIND( '-' )` and `FIND( '+' )` statements when the *Item No., Variant Code* key is set as the current key for the ***Item Ledger Entry*** table for all the companies (including a non-company table with this name) in the database.

### Example 5

```
INSERT INTO [$ndo$dbconfig] VALUES

( 'IndexHint=Yes;Company="CRONUS International Ltd.";Table="Item
Ledger Entry";Key="Item No.", "Variant Code";Search Method=;Index=3' )
```

This will hint the use of the \$3 index of the *CRONUS International Ltd\_ \$Item Ledger Entry* table for every search method when the *Item No., Variant Code* key is set as the current key for the ***Item Ledger Entry*** table in the CRONUS International Ltd. company.

## Lock Granularity

When Dynamics NAV is reading data from tables it places forced `ROWLOCK` hints, by default. These rowlock hints prevent SQL Server from automatically determining the granularity (row, page or table) of the locks that it places. This can lead to a high locking overhead on the server, even though concurrency is optimum.

To allow SQL Server to determine the granularity of the locks that it places, the `DefaultLockGranularity` parameter can be used in the database configuration table.

The syntax of the `DefaultLockGranularity` parameter is:

```
DefaultLockGranularity=<Yes,No>
```

When the parameter is `Yes`, SQL Server will choose the granularity of the locks that it places. When the parameter is `No`, Dynamics NAV will override SQL Server and place `ROWLOCKS`.

## 30.7 Login Stored Procedure on the SQL Server Option

A login stored procedure is a stored procedure that you can use to perform predefined functions after a user logs on to the SQL Server Option. A typical function would be to generate a message informing the user that the database is currently in single-user mode so that an administrator can perform some database maintenance tasks and is therefore inaccessible.

The login stored procedure is run immediately after the user has logged on to SQL Server and opened a database and before Dynamics NAV carries out any tasks including executing any C/AL triggers. The user must have successfully logged on to the server and have access to the database before the stored procedure is run.

### Creating the Stored Procedure

The stored procedure is created in the database and has a predefined name and a list of parameters.

The stored procedure is called [`$ndo$dbproperty`] and has the following characteristics:

- It takes two `VARCHAR` parameters: the name of the application and the C/SIDE version number. These parameters *must* be declared as part of the stored procedure but do not have to be used.
- It can perform transactions. Dynamics NAV uses a `COMMIT` to flush any outstanding transactions after the stored procedure has finished executing.
- The `RAISERROR` statement can be used to display an error message in Dynamics NAV and prevent the user from accessing the database.
- The `PRINT` statement can be used to display a warning in Dynamics NAV and allow the user to access the database.
- If the stored procedure returns a value, it is ignored.
- If the stored procedure does not exist, no action is taken by Dynamics NAV and the login process continues normally.

The following examples show how to create a login procedure in the Query Analyzer tool. The database must be selected before these statements are executed.

#### Example 1

Displaying a warning message in Dynamics NAV and permitting the login:

```
IF EXISTS (SELECT name FROM sysobjects
           WHERE name = 'sp_$ndo$loginproc' AND type = 'P')
  DROP PROCEDURE [sp_$ndo$loginproc]
GO
CREATE PROCEDURE [sp_$ndo$loginproc]
  @appname    VARCHAR(64) = NULL,
  @appversion VARCHAR(16) = NULL
AS
BEGIN
  PRINT 'The system will be unavailable on Sunday 1st April.'
END
```



```
GO
GRANT EXECUTE ON [sp_$ndo$loginproc] TO public
GO
```

**Example 2**

Displaying an error message in Dynamics NAV and disallowing the login

```
IF EXISTS (SELECT name FROM sysobjects
           WHERE name = 'sp_$ndo$loginproc' AND type = 'P')
  DROP PROCEDURE [sp_$ndo$loginproc]
GO
CREATE PROCEDURE [sp_$ndo$loginproc]
  @appname    VARCHAR(64) = NULL,
  @appversion VARCHAR(16) = NULL
AS
BEGIN
  IF SUSER_SNAME() IN ('ACCOUNTS\jim', 'SALES\bill')
    RAISERROR ('Please contact the system administrator.', 11, 1)
END
GO
GRANT EXECUTE ON [sp_$ndo$loginproc] TO public
GO
```



**Part 12**  
**Appendixes**



## **Appendix A**

### **C/SIDE Specifications**

This appendix provides the technical specifications of C/SIDE. Use this information to get an overview of maximum sizes and other limitations that may affect your application design.

- Specifications for the DBMS
- Specifications for C/SIDE Application Objects

## A.1 Specifications for the DBMS

These are the specifications for the C/SIDE DBMS (Database Management System).

Maximum number of physical disk files	16
Database file size	256 GB
Maximum number of objects in a database	Infinite
Maximum number of characters in application object names	30
Maximum number of characters in a password	10
Maximum number of concurrent users (the actual limit depends on your hardware and the workload)	500
Maximum cache size	1GB

## A.2 Specifications for C/SIDE Application Objects

This section lists specifications for the five types of application objects in a C/SIDE database.

### Specifications for Tables

Range for table object ID numbers	1 - 999,999,999 <sup>(A)</sup>
Maximum number of characters in a table name	30
Maximum table size	Infinite
Maximum number of records in a table	Infinite
Maximum record size	4KB (C/SIDE Database Server), 8KB (SQL Server)
Maximum number of fields in a record	500
Range for field numbers	1 - 999,999,999
Maximum number of keys for a table	40
Maximum number of distinct fields per key	20 for a primary key. The number of fields in the primary key + the number of fields in a secondary key which do not occur in the primary key must always be less than or equal to 20.
Maximum number of SumIndexFields per key	20
Maximum number of characters in a text or code field	250
Maximum size of a BLOB field	2 GB
Maximum number of characters in a field name	30

(A) all application objects are identified by an ID number. there are restrictions, however, on the numbers you can use when you create your own application objects. Please contact your NTR for more information.

### Specifications for Forms and Reports

Range for form or report object ID numbers	1 - 999,999,999 <sup>(A)</sup>
Maximum form width	100000 x 1/100 mm
Maximum form height	100000 x 1/100 mm
Maximum number of nested forms	1
Maximum number of controls on a form	32767
Maximum number of characters in a label	254
Maximum number of characters in a text box	250
Maximum bitmap size in bitmap property	32500 bytes
Maximum number of levels in drop-down menus	10

(A) all application objects are identified by an ID number. there are restrictions, however, on the numbers you can use when you create your own application objects. Please contact your NTR for more information.

### Specifications for Codeunits

Range for table object ID numbers	1 - 999,999,999 <sup>(A)</sup>
Maximum number of characters in variable names	30
Maximum number of dimensions in array variables	10
Maximum number of elements in an array variable	1,000,000
Maximum physical size of a codeunit	2 GB
Lower bound of index in an array	1

(A) all application objects are identified by an ID number. there are restrictions, however, on the numbers you can use when you create your own application objects. Please contact your NTR for more information.



## **Appendix B**

### **Report Flow Charts**

This appendix illustrates the flow of control for reports in C/SIDE.

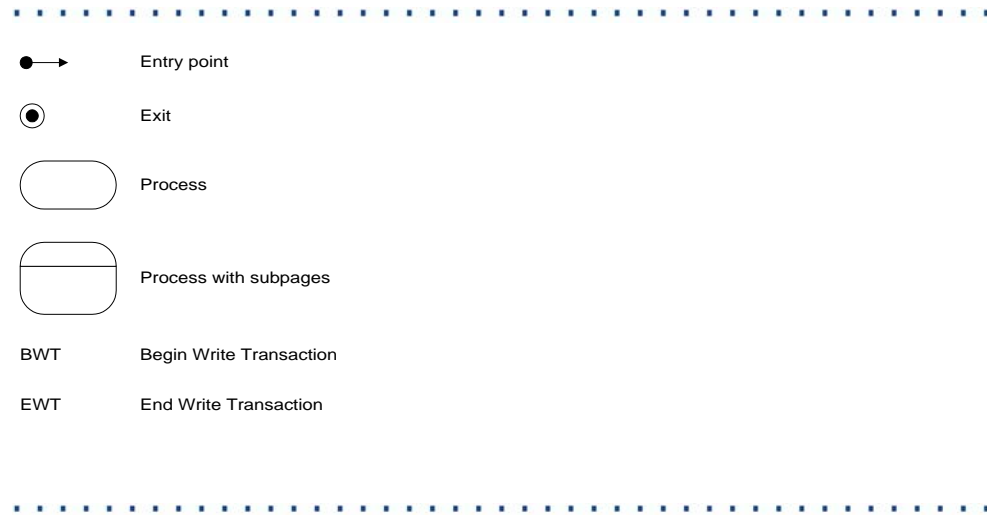
- Report Flow Charts
- Report.Run
- DataItem.Run
- Section.Run
- Header.Run
- Footer.Run
- TransHeader.Run
- TransFooter.Run
- GroupHeader.Run
- GroupFooter.Run
- Body.Run
- NewPage
- GetRecord

## B.1 Report Flow Charts

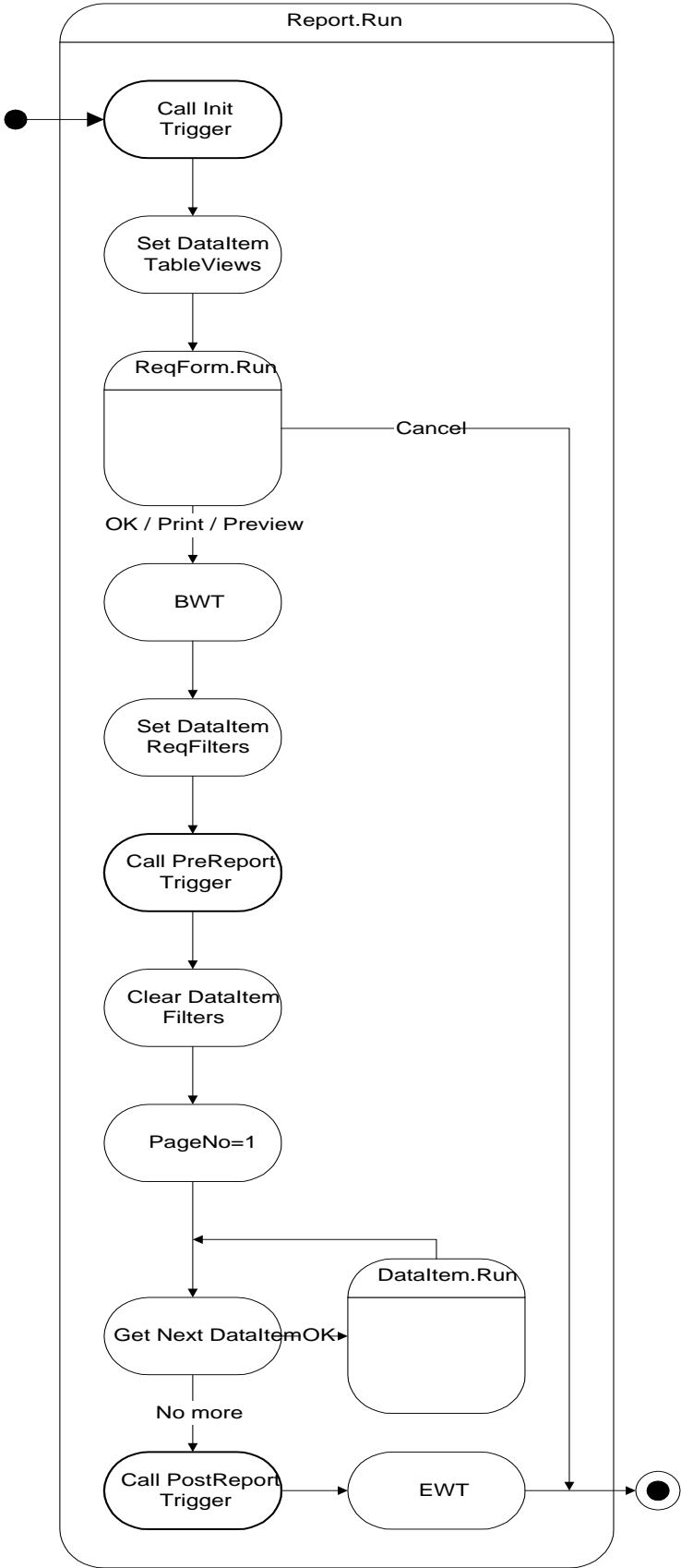
The following sections contain flow charts that show the flow of control for reports in C/SIDE.

As indicated by the following legend, some processes in one flow chart are "exploded" in the following pages in order to show more details.

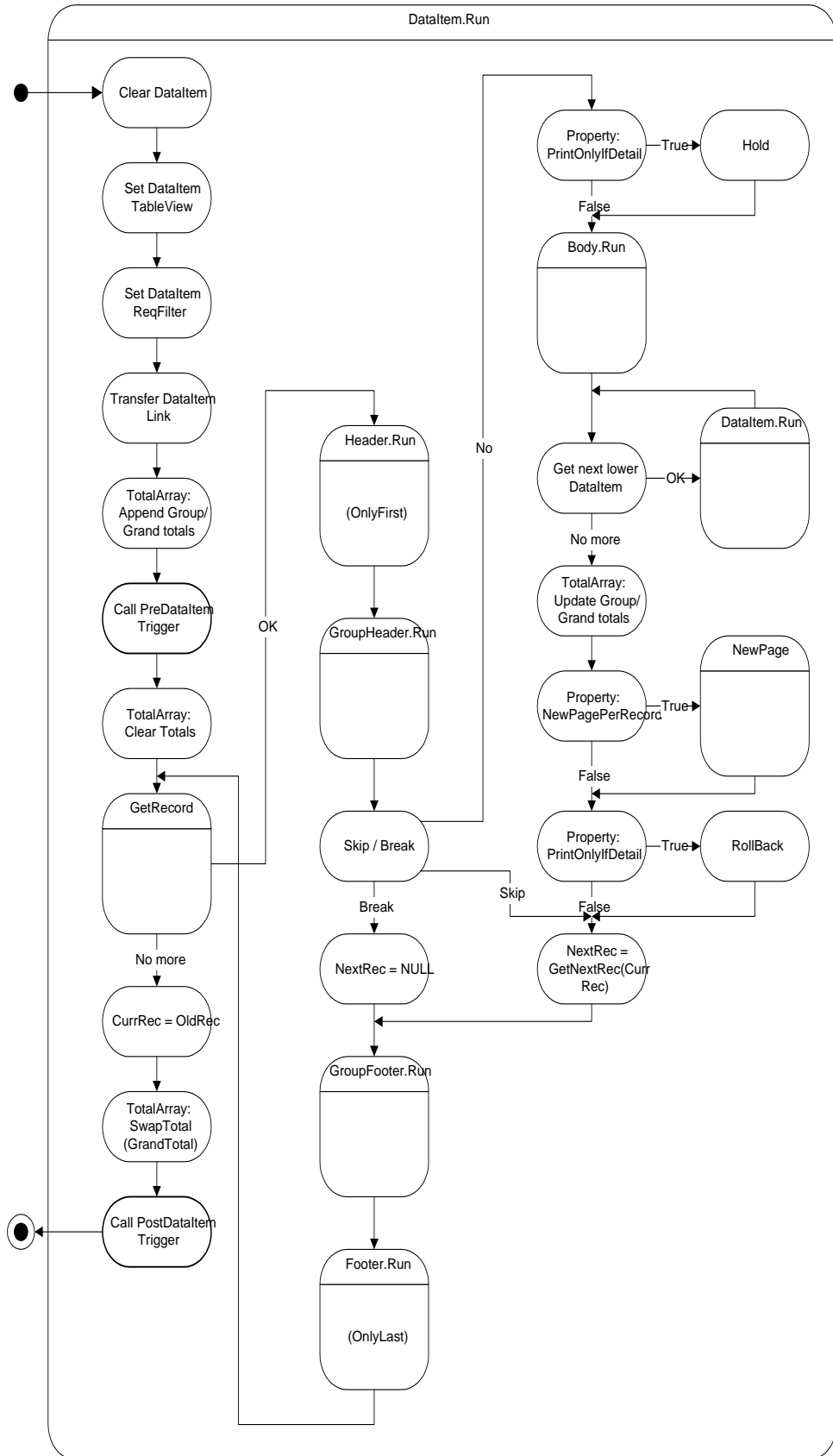
### Legend



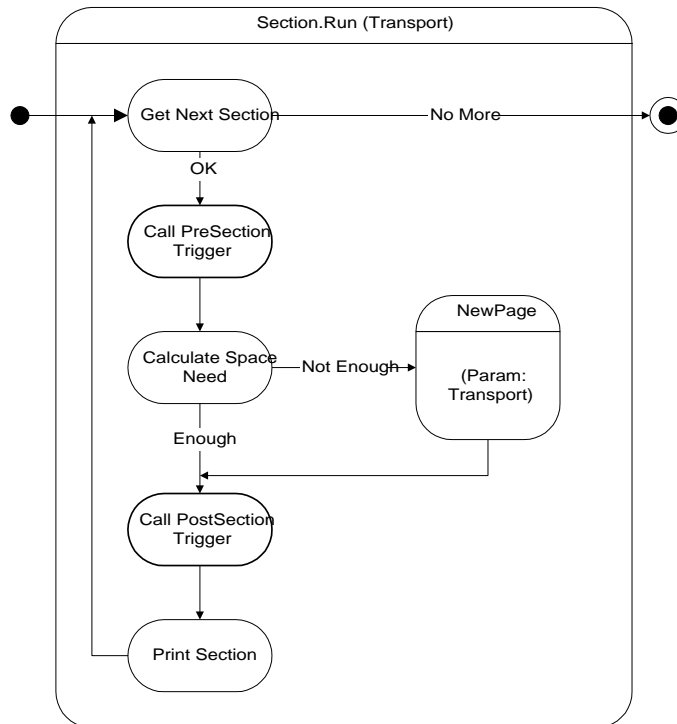
### B.2 Report.Run



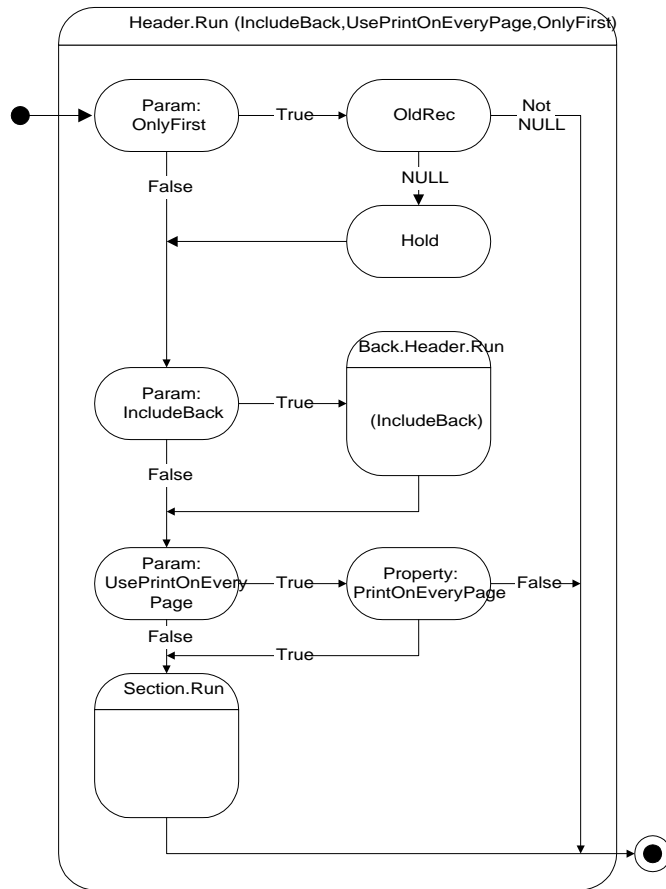
### B.3 Dataltem.Run



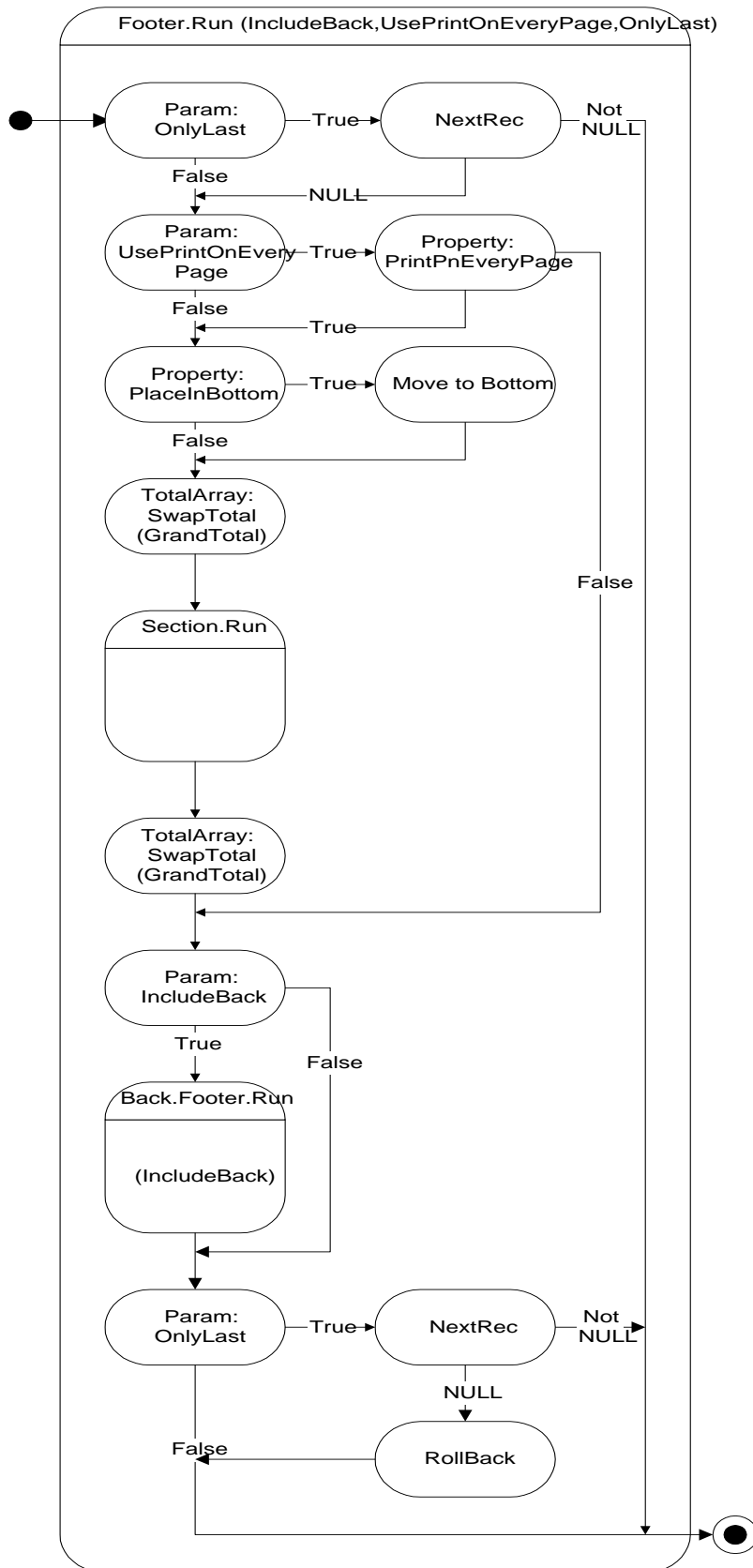
## B.4 Section.Run



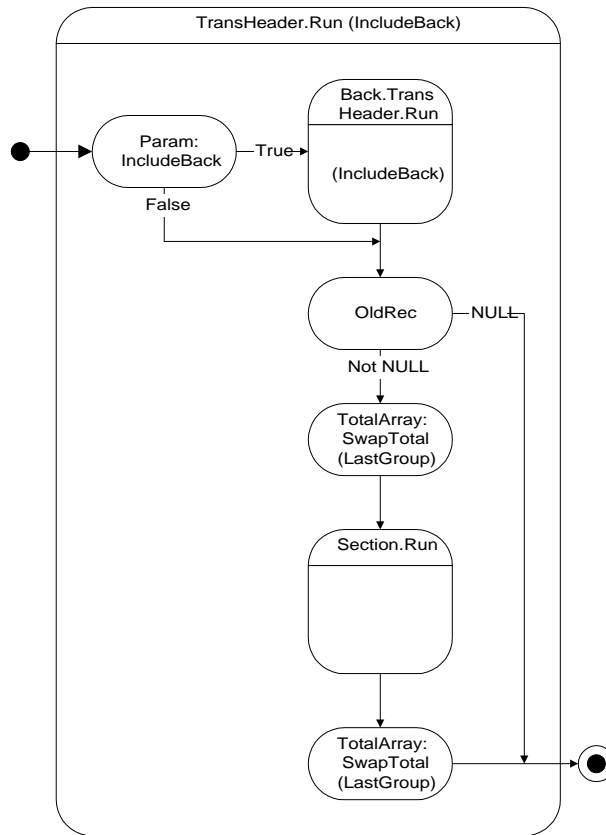
## B.5 Header.Run



## B.6 Footer.Run

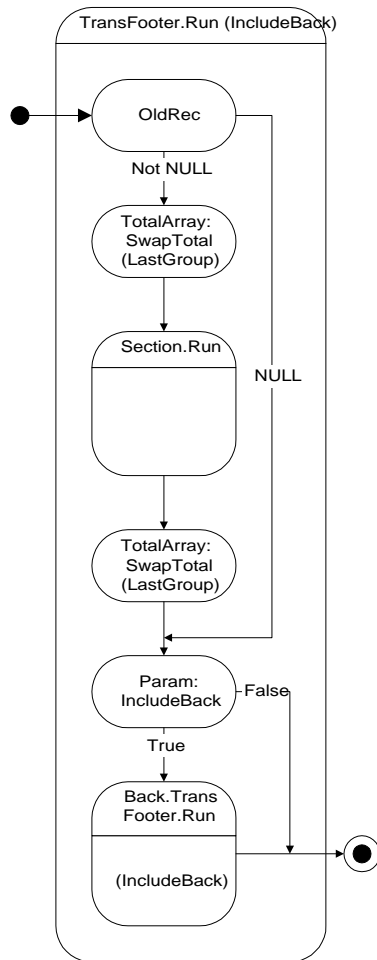


## B.7 TransHeader.Run

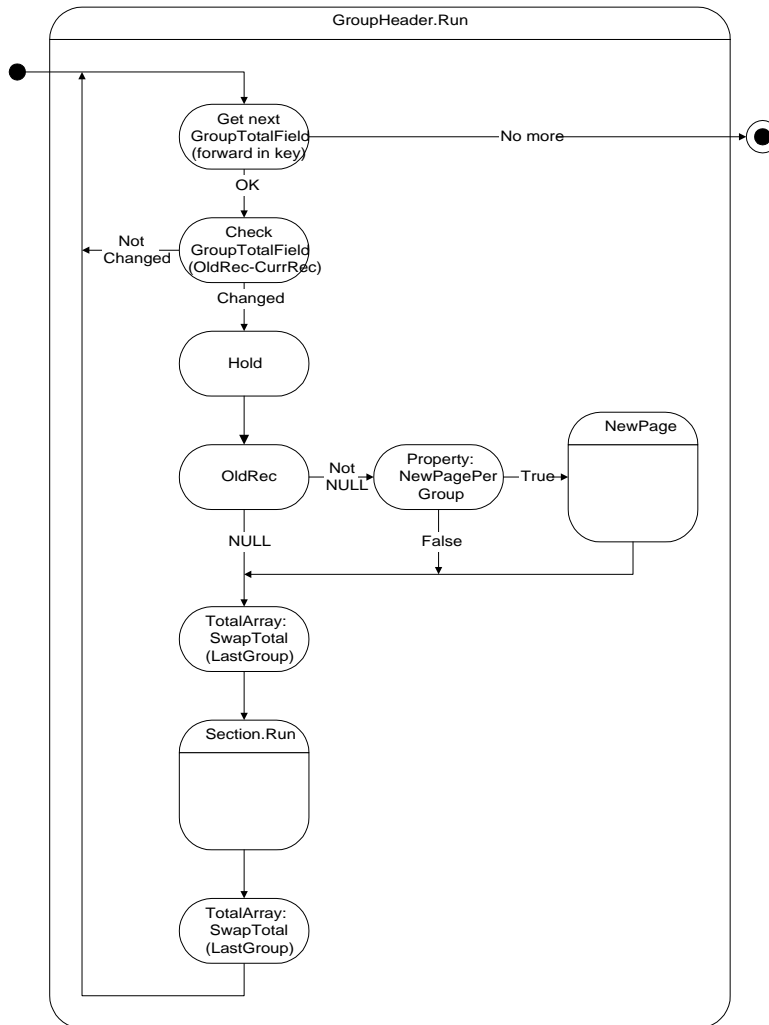




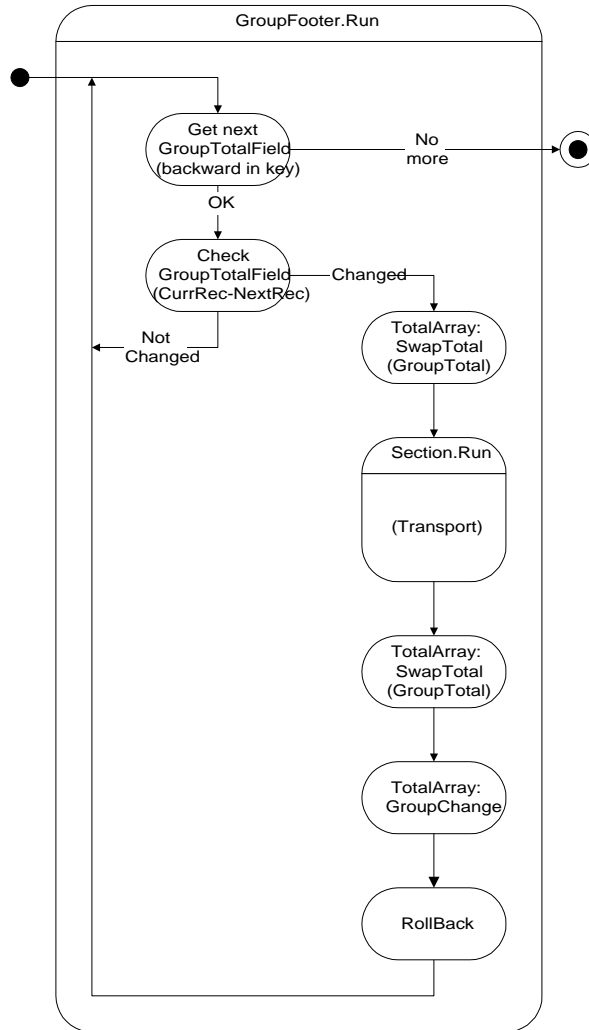
## B.8 TransFooter.Run



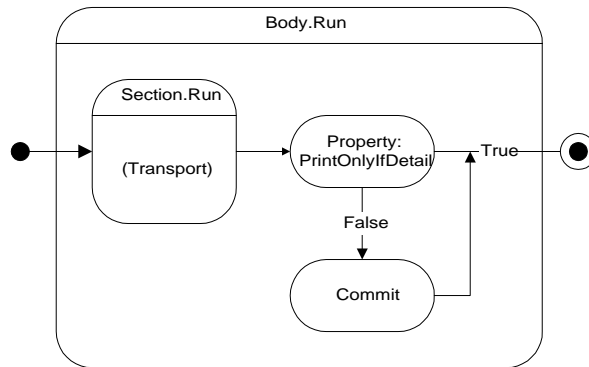
## B.9 GroupHeader.Run



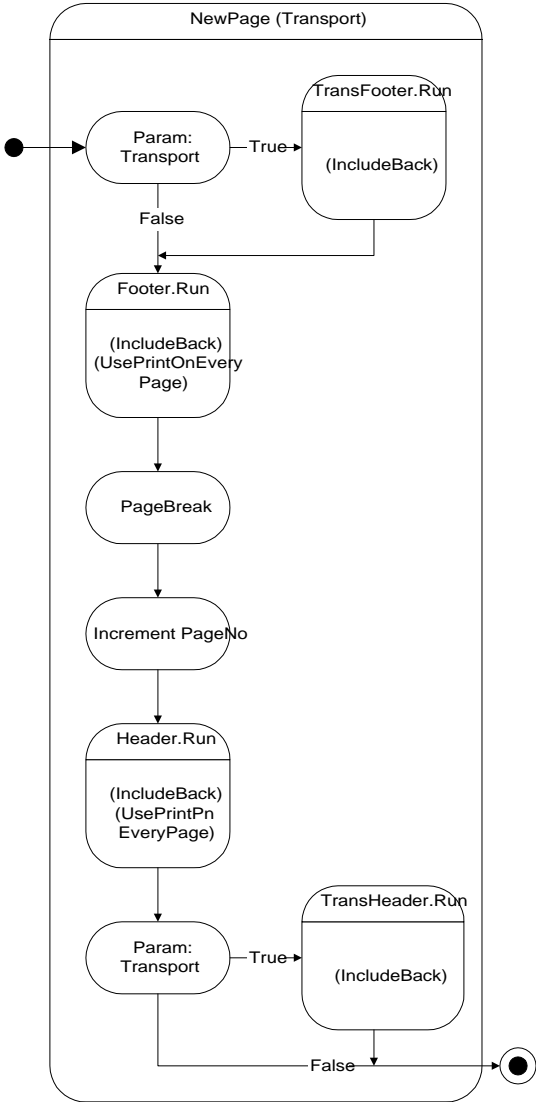
## B.10 GroupFooter.Run



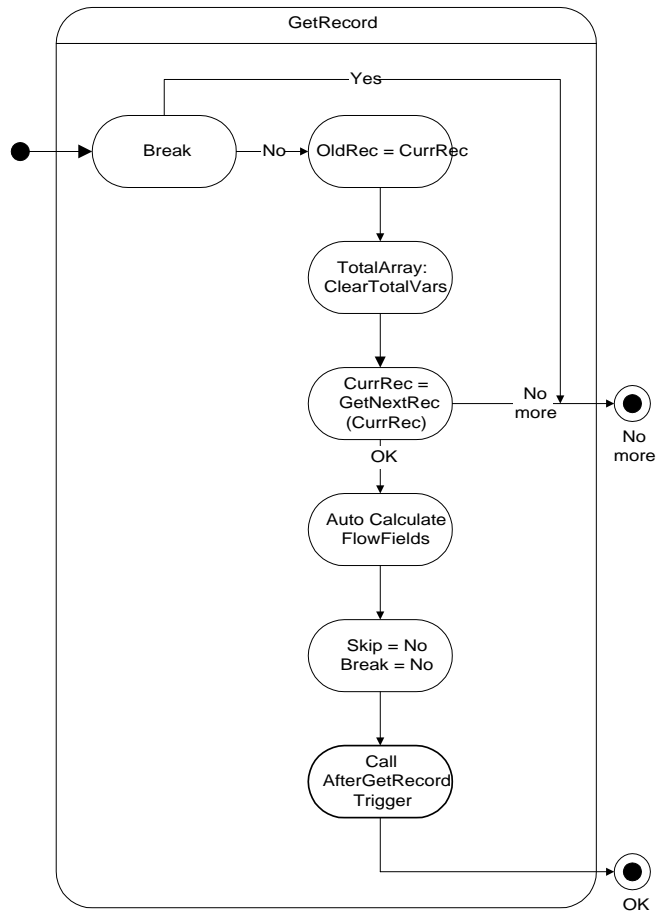
## B.11 Body.Run



### B.12 NewPage



## B.13 GetRecord



## **Appendix C**

### **Dataport Flow Charts**

This appendix illustrates the flow of control for dataports in C/SIDE.

- Dataport Flow charts
- Dataport.Import/Export
- DataItem.Export
- VariableRecord.Export
- FixedRecord.Export
- DataItem.Import
- VariableRecord.Import
- FixedRecord.Import




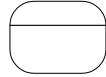
## C.1 Dataport Flow charts

The following sections contain flow charts that show the flow of control for dataports in C/SIDE.

As indicated by the legend, some processes in one flow chart are "exploded" in the following pages in order to show more details.

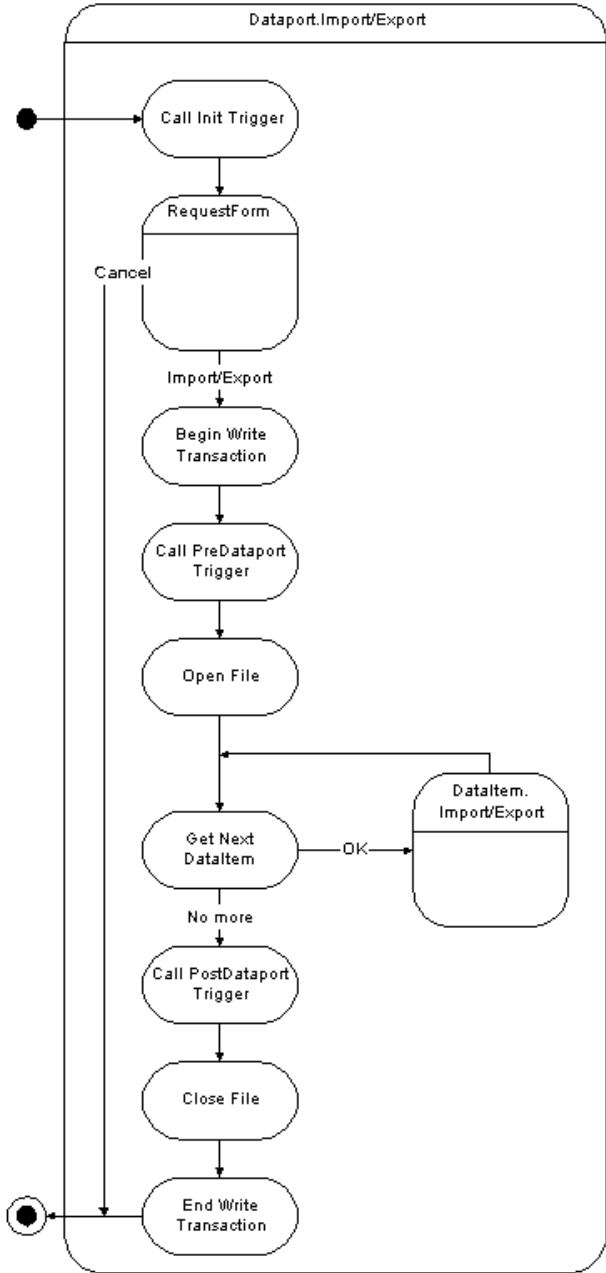
### Legend



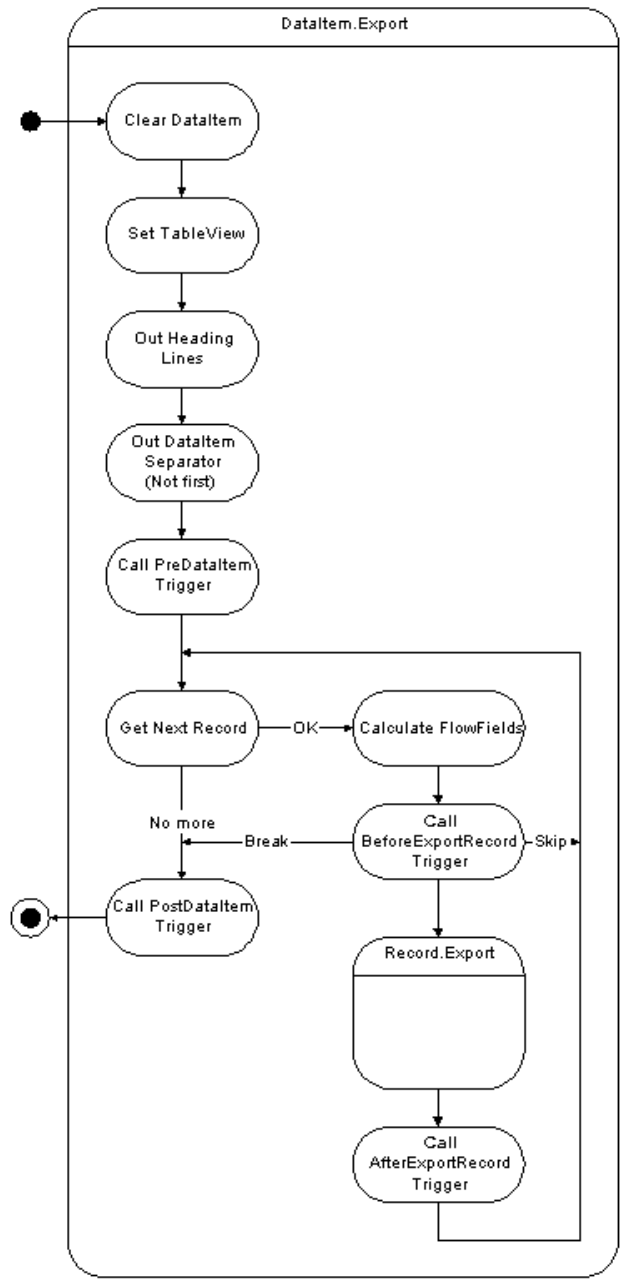
-  Entry point
-  Exit
-  Process
-  Process with subpages
- BWT      Begin Write Transaction
- EWT      End Write Transaction



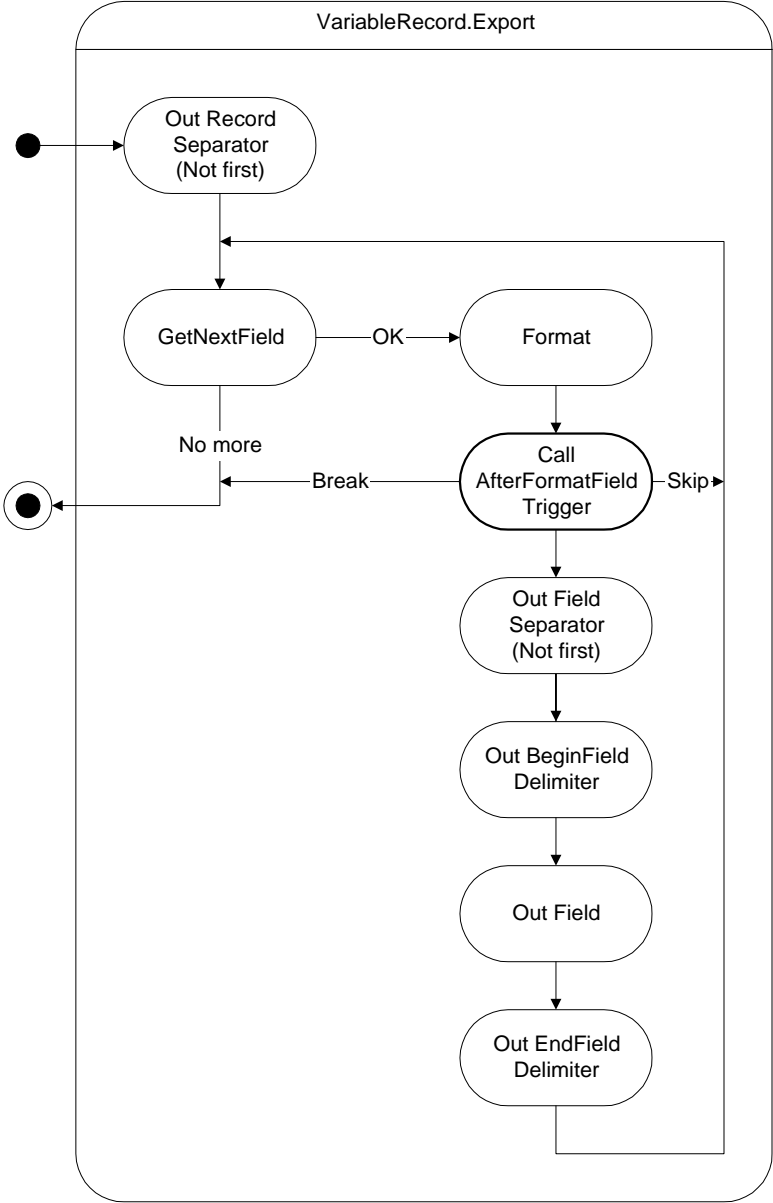
### C.2 Dataport.Import/Export



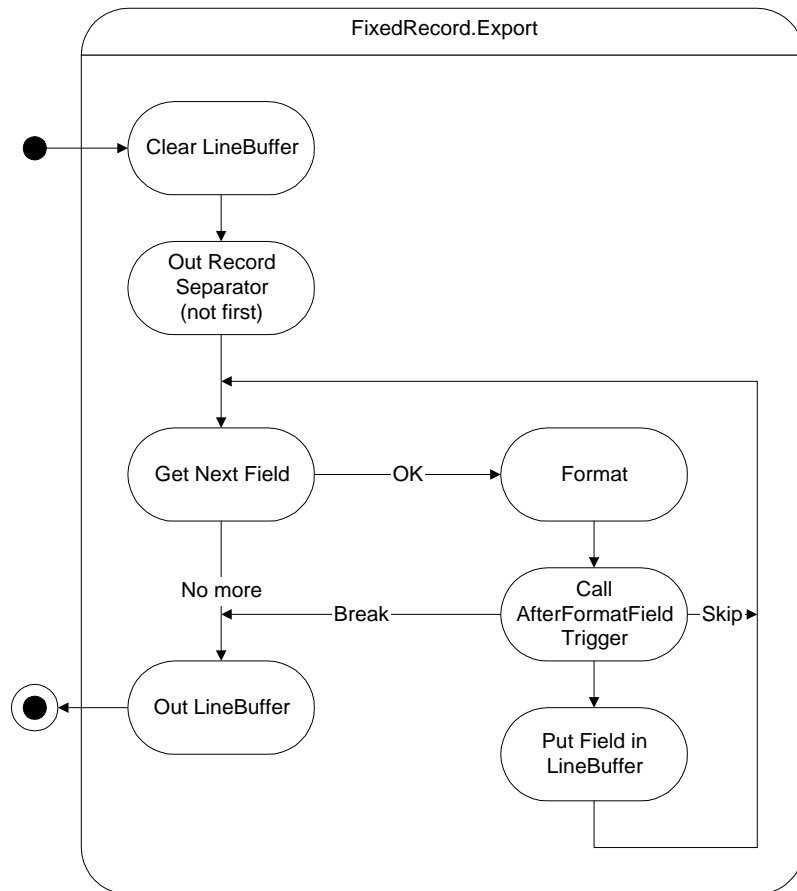
### C.3 Dataltem.Export



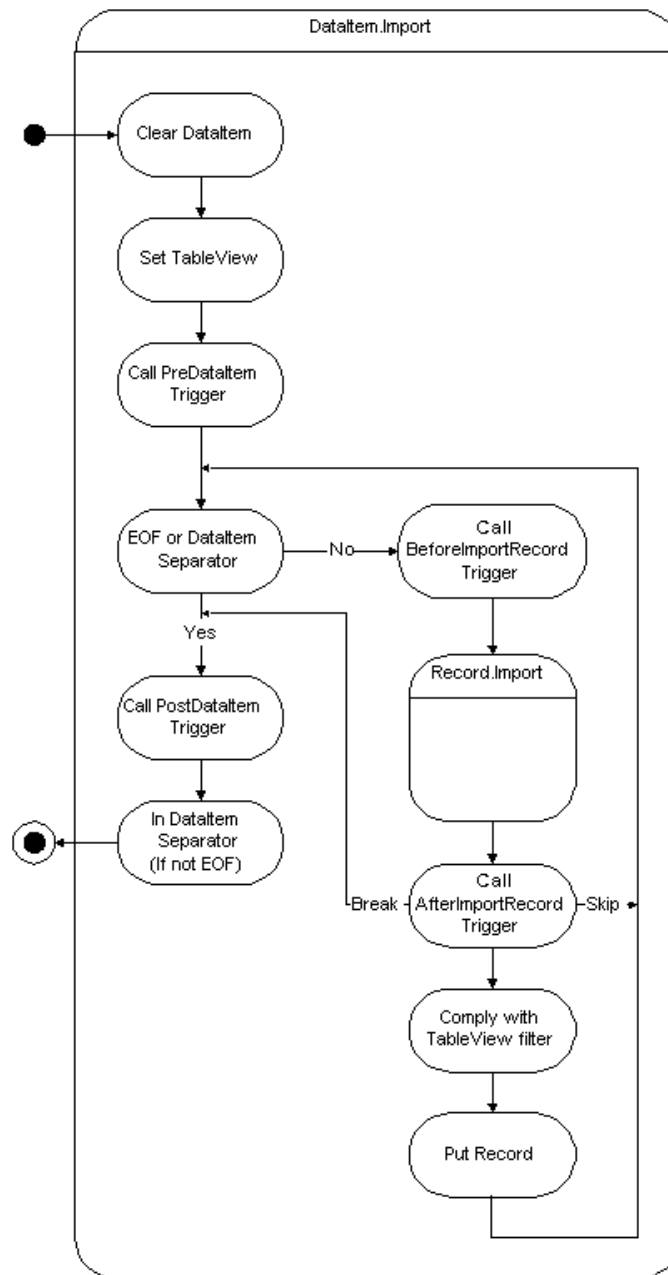
### C.4 VariableRecord.Export



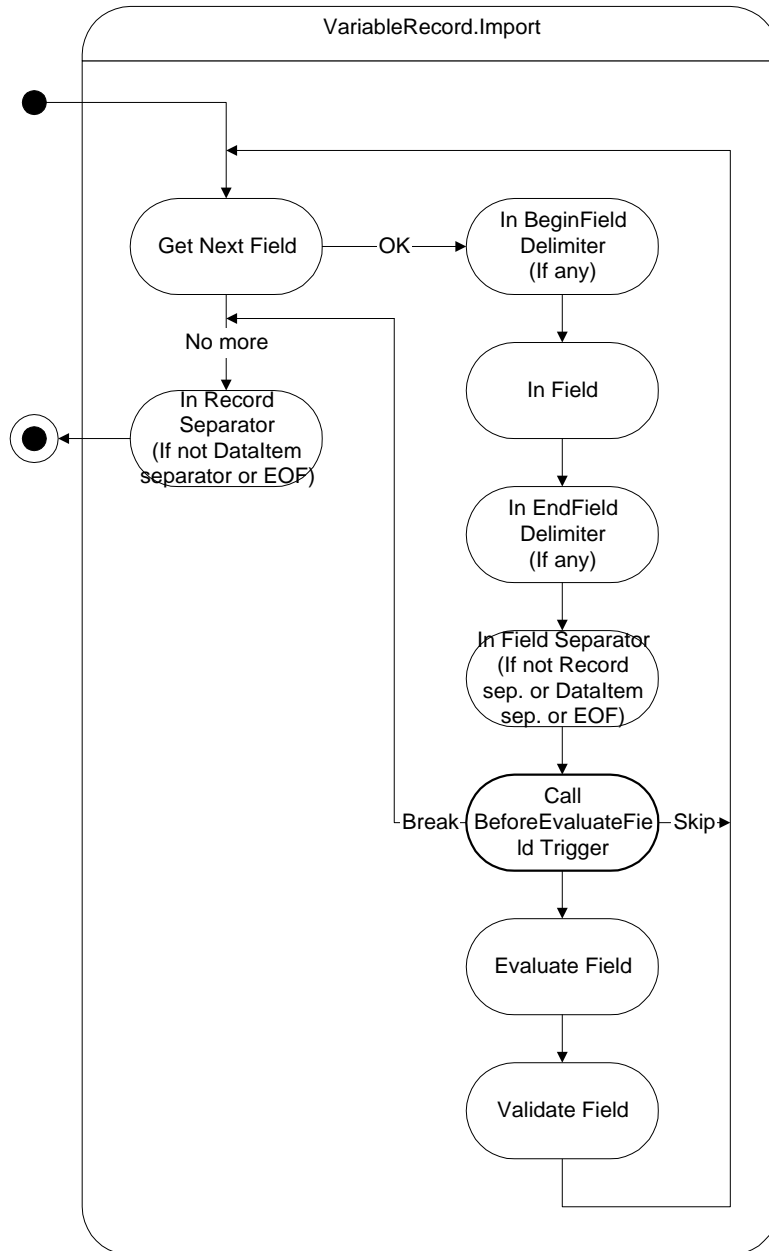
## C.5 FixedRecord.Export



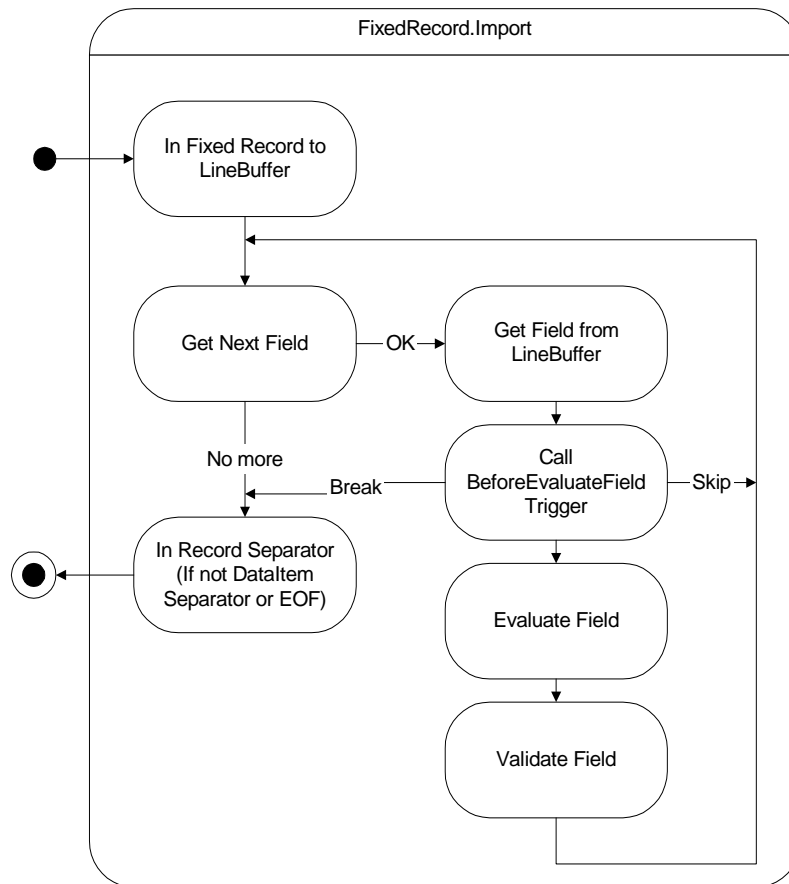
## C.6 Dataltem.Import



## C.7 VariableRecord.Import



## C.8 FixedRecord.Import







## **Appendix D**

### **NDBCS – The Database Driver**

This appendix describes some details of the way that the database driver module (NDBCS) for the SQL Server Option for Dynamics NAV has been implemented. Although it is not a guide for C/AL development, it can help you understand the way Dynamics NAV uses SQL Server. This appendix also contains a brief history of the performance improvements that have been implemented for the SQL Server Option.

This appendix contains the following sections:

- NDBCS – the Database Driver
- A Brief History of Performance Improvements

## D.1 NDBCS – the Database Driver

The database driver maps internal database requests, that have been formulated for the architecture used by C/SIDE Database Server, to SQL-based requests to SQL Server. This is done for all the types of requests that must communicate with the database server, including:

- Connecting, setting connection properties and disconnecting from the server.
- Opening, creating and altering databases.
- Redesigning tables and managing linked objects such as views.
- Reading data for all the objects in the form, report, and dataport engines.
- C/AL functions such as `FIND`, `MODIFY` and so on.
- FlowFields.
- Statistics for databases, sessions and tables.
- Sort Order, Character Set and Code Page considerations (Collations).

Most of the SQL statements that are used to achieve this mapping are constructed in a dynamic manner where everything but the basic syntax of the statement is unknown until runtime. For example, the table name, field list, lock type, filter parameters and the ordering are all dependent on the C/SIDE area or application area that is being used. In some cases, such as database redesign, table redesign and SIFT queries, the syntax itself varies considerably.

This is in contrast to the majority of SQL applications that use pre-defined business logic in the form of query repertoires, statement batches and stored procedures. Although these elements can be parameterized, they are essentially static in nature and allow a great deal of optimization to be incorporated, both at the time they are designed - by fully exploiting the power of the SQL language - and at the time they are executed - by allowing the server to pre-build internal structures such as compilation plans, execution plans, intermediate working tables and buffers for caching.

The Dynamics NAV Client Monitor can be used to display the SQL statement that is used for the current database operation, regardless of its origin. The SQL statement could originate from, for example, a C/AL function or a form. The Client Monitor displays the SQL statement in a slightly more readable layout than that used internally. When the driver issues more than one statement for an operation, only the first statement is displayed in the Client Monitor. However, this is not very common.

The SQL Profiler can also be used to display the SQL statements being received by the server in more detail. Although the SQL Profiler gives you more information, it is not easy to track the statements back to database behavior in Dynamics NAV, and in many cases internal stored procedures, and other mechanisms, are being used (by both the SQL Server ODBC driver and the server itself) in place of the original SQL statements.

If you want to understand how SQL Server is being utilized by Dynamics NAV, or why there may be a functional or performance problem, you should use one of these tools to analyze database activity.

The following sections contain details about some of the more important areas of the database driver. These areas are particularly concerned with performance and the ability to use SQL Server as optimally as possible, given the nature of the C/AL application language that must be used for both server platforms.

## Database Driver Concepts

This section explains some of the most important database driver concepts and terms.

### Command

In this context a command is a driver object that is used for executing any SQL statement and has built in error handling and can use parameterization.

### Direct and Prepared Execution

When a SQL statement is executed it can use either a direct execution or a prepare-execute model.

#### The Prepare-Execute Model

The prepare-execute model is a general model that allows for the optimization of statements that are frequently executed. The preparation stage is performed once, and this establishes server-specific data structures – typically compilation or execution plans. The execute stage is then performed repeatedly, using the created data structures. For example, the preparation of an `INSERT` statement is followed by multiple executions of the prepared statement, with different parameter values (different records being inserted into the table).

#### The Direct Execution Model

Direct execution performs all the work necessary for preparing and executing on the server, in one step. Therefore, it takes longer to issue the statement several times because it must be prepared and executed each time. SQL Server has increased the performance of direct execution by internally matching its data structures and re-using them. However, it is still faster to use the prepare-execute model when you know that a statement will be re-executed.

### Result Set

A result set is the set of records returned from the server to a client application, such as Dynamics NAV, in response to a query. The query is usually a SQL `SELECT` statement or a stored procedure. The set can include 0, 1 or more records. A default result set is the fastest and most efficient way for the server to send the results. This is sometimes referred to as a fire hose, taken from the analogy of water being sprayed at high power onto the 'client'. A cursor can also be used for sending a result set, but is less efficient because it supports additional features on top of the result set.

SQL Server places an important limitation on the use of default result sets:

- There can be only one default result set active on a given client connection, for example, a single instance of Dynamics NAV that has opened a database.

This means that once the server starts sending a particular result set to a client, the client must read the entire set to the end, close the set before reading to the end, or cancel the request. The client cannot partially read from the set and then perform another activity, such as request a new set for a different query or make modifications to a table. This makes the use of default result sets quite limited for the database driver because it must track many result sets for different clients at the same time, in response to read requests for different record variables, for example. The database driver uses cursors to do this. For queries that are known to produce 0 or 1 record only (singleton queries) such as a `GET` or `SIFT` queries, the driver always uses a default result set since it

can be opened, the data read, and closed within the same Dynamics NAV database operation and does not remain active.

## Cursor

In general, a cursor is a data structure that allows the result set of a query to be navigated and manipulated, with some additional features other than that of merely reading the results from beginning to end. The cursor can be viewed as an additional layer on top of the result set. When a cursor is used to retrieve data, the result set is no longer said to be a default result set. Cursors were designed primarily to allow applications that deal with single record retrieval (such as Dynamics NAV) to use result-based SQL databases.

The most commonly used features are the ability to:

- Maintain a current record position in the results.
- Scroll backwards or jump around in the set.
- Modify or delete the record at the current position.

In the driver, it is essential to use cursors to overcome the limitation imposed by SQL Server of having only one active default result set on a client connection (see Result Set). Cursors allow many result sets to be active (in an open state) so that many read requests on different tables, with various keys and filters, can be serviced efficiently when running a codeunit, for example. Otherwise the driver does not need to make use of the variety of features that cursors offer.

The following cursor types are available in SQL Server and are listed in order of their reading efficiency (the fastest being the default result set, which is not classed as a cursor):

Type	Properties	Basic Requirements
Fast Forward	Read-only, forward-only, latest data at fetch time	No locking, no BLOB columns
Dynamic	Updateable, scrollable, latest data at fetch time	An index must match ordering
Keyset	Updateable, scrollable, snapshot at creation time and partially latest data at fetch time	A primary key
Static	Updateable, scrollable, snapshot at creation time	None

There are several requirements that must be met when requesting a particular cursor type (the driver requests the fastest possible cursor for the given inputs), in order for the server to provide the cursor type. If the requested cursor type cannot be created, an alternative type is offered that has less requirements but is often less efficient.

In some cases the driver does not know if a cursor type cannot be provided, because it is too costly to determine if all the requirements can be met, such as the existence of the correct indexes. In other situations it knows in advance that a particular cursor type cannot be used. For example, the driver never requests a Fast Forward cursor for a table that has been locked because this will never be provided. In this case, a dynamic cursor is used instead. The server can always supply a dynamic cursor for Dynamics NAV (non-

linked) tables, provided that the *MaintainSQLIndex* key property is set to *Yes* and the indexes have not been modified outside of the program.

The driver never uses the scrollable or updateable properties of cursors. Only a limited number of cursors can co-exist in the driver because they are a more expensive client and server-side data structure in terms of memory-usage, and sometimes disk-usage than default result sets.

## Rowset

A rowset is an internal driver object that is used for data retrieval. A rowset is based on a driver command object. It always encapsulates a result set and can also use a cursor, depending on the database operation that caused the rowset to be created.

The rowset contains:

- The SQL `SELECT` statement to be executed.
- The table, output field list, filter, parameter values and ordering.
- The data for the records that are returned.
- The status of each record (normal, deleted, and so on).
- The current record position in the record buffers.
- The result set state (open, closed, read full set, performed a `NEXT`, and so on).
- Statistics about the usage of the rowset since it was created.
- The cursor type, locking, and other attributes.
- Caching information.

When a rowset is created, the following attributes are fixed:

- The table.
- The output field list.
- The ordering of the results (based on the current key).
- The filter fields and operators (but not the filter values).
- The search method being used for the database request.
- The locking status.
- The cursor type.
- The number of initial result set record buffers allocated in memory (the number of actual buffers can grow, and later shrink back to this initial size).

Rowsets are maintained both for every table and for every connection. When searching for an existing rowset to be used for a request, only the rowsets for the table involved are examined. When searching for rowsets that should be deleted because they have expired or to allow new rowsets to be created, all the rowsets for the connection are examined. When a table handle is closed by `C/SIDE`, for example, because of error conditions or table re-designs, the rowsets it owns are also deleted.

## Transaction Type

See the *C/SIDE Reference Guide* online Help.

## Reading Data: Rowset Usage

The database driver uses a rowset object to read data from the database. This involves creating a new rowset object or utilizing an existing one. After it has been created, the rowset object usually undergoes the following operations:

- The current filter parameter values are used and the data is converted from a C/SIDE format to a SQL format.
- If the result set is open, it is closed.
- It is determined whether the rowset is caching data or not and if this cache can be used instead of executing the statement.
- It is determined whether the filter will give an empty set or not.
- The SQL statement is executed. It might need to be prepared first.
- It is determined whether or not the result set field list is compatible with the C/SIDE field list.
- A number of records are fetched from the network buffers or from the server (this step is often performed as part of the execution phase) and placed in the record buffers.
- If no records are found, this situation must be handled. Finding no records might be the expected result of the database operation or it might be an error. The result set is closed here.
- If records were found, the required record position based on the database operation must be obtained. This may require more fetches.
- If the required record was not found with this rowset and there are more records available in the logical set, a new rowset is used to continue the search and the current rowset is deleted.
- If the required record is found, the data is converted from a SQL format to a C/SIDE format.
- The status of the database operation, and the record data, is now available to the C/SIDE database layer, and to the area of Dynamics NAV that made the request.

After a rowset has been created and executed in this way, it can be re-used for subsequent operations. For example, a rowset created to satisfy a `FIND( ' - ' )` will be used to satisfy the subsequent `NEXT`, provided that all the required rowset attributes are compatible (for example, the table lock status for the `FIND` is the same as the table lock status for the `NEXT`). The remaining records might have been fetched into memory already, or further fetches might be required. If the current record in the rowset matches the input record of the `NEXT`, the next record is provided as output, and so on until no more records are found.

This mechanism of re-using rowsets is essential in the driver. It allows existing server execution plans and statement handles to be used when re-executing a rowset statement using prepared statements, and allows fetch operations on open result sets to be used thereby avoiding having to re-execute statements. In many situations it also allows cached data in the rowsets to be used without having to visit the server at all. When a filter is used in a request, the filter is parameterized in the rowset and the SQL statement so that different filter values can re-use the same rowset by re-executing the same statement with the new values. If anything other than the filter values are changed (for example, an operator is changed from `=` to `>`), the rowset cannot be reused. Therefore, when a `NEXT` operation is being performed, the filter for the rowset

that is being re-used must match exactly the active filter for the `NEXT` operation or the rowset cannot be reused.

Almost every rowset can be re-used to exploit the set-based behavior of SQL Server. The Client Monitor can display additional SQL Status information that shows if a rowset has been re-used for a particular operation and how many times it has been used since it was created. For more information about the Client Monitor, see "The Monitor Virtual Table" on page 121.

Executing even the simplest query in SQL Server to obtain a record is more expensive than retrieving a single record in C/SIDE Database Server. This is mainly due to the power of the SQL Query Optimizer, which carries additional baggage when executing simple queries because it is able to efficiently handle complex queries. Dynamics NAV generally executes simple queries. There is no SQL optimizer in C/SIDE Database Server because the server does not support the SQL language and therefore this added overhead is not present. The performance section of this document presents the mechanisms that are employed to reduce the expense of retrieving records on SQL Server.

## Modifying Data

When data is to be modified in the database, the appropriate SQL statement (`INSERT`, `DELETE` or `UPDATE`) is issued using a driver command object. This means that either a new object must be created or an existing object utilized, as is the case with a rowset. The re-using of command objects for modifications allows the prepare-execute model to be employed in a parameterized statement. The prepare-execute model is an optimal mechanism for issuing these statements. The driver creates and re-uses the following commands (maintained within the table):

C/AL Function	SQL Statement	Properties
<code>INSERT</code>	<code>INSERT</code>	1 per table
<code>DELETE</code>	<code>DELETE</code>	per-connection limit
<code>MODIFY</code>	<code>UPDATE</code>	per-connection limit
<code>DELETEALL</code>	<code>DELETE</code> with filter	per-connection limit
<code>MODIFYALL</code>	<code>UPDATE</code> with filter	per-connection limit
	<code>INSERT</code> (bulk)	1 per table, batched, used during bulk inserts

When modifications are performed on a table with `SumIndexFields`, the `SIFT` trigger on the SQL table is fired to update the accompanying `SIFT` tables.

When performing a `MODIFY`, the record to be modified is first read from the database table (or a client cache). This allows a comparison to be made between this record and the record values that are being modified. Only those fields that have been changed will be included in the SQL `UPDATE` statement thereby improving performance.

The timestamp field in the table is used when an optimistic concurrency check must be performed to determine if the record in the table has been changed since the driver read it. Timestamp fields are assigned a unique value when a record is inserted into a table in SQL Server, and the timestamps are updated automatically whenever the record is changed. The driver always reads the timestamp value when it reads a record.

The driver reads the timestamp when performing a `DELETE` or `MODIFY` but this check is not performed when performing `DELETEALL` or `MODIFYALL`. If the timestamp is greater than it was when the driver read the record, a standard Dynamics NAV error message is displayed.

## Transactions

C/SIDE tracks transactions at several levels and these can vary from the points at which C/AL code may begin, commit or rollback transactions. Most of the additional complexity in transactions has been implemented to optimize the point at which the server really needs to begin or end a transaction boundary, and therefore avoid creating unnecessary transactions.

The driver manages transactions in the following way:

- A SQL Server setting is enabled so that every SQL statement will begin a transaction implicitly. This avoids having to send manual begin markers.
- Different C/SIDE Transaction Types use different transaction isolation levels. In SQL Server, isolation levels determine the default locking behavior of all the data accessed, but the driver sometimes overrides the locking behavior when executing particular SQL statements.
- Isolation levels are not changed until it is necessary. For example, if there is no locking in a transaction, no change in isolation level takes place.
- No commit or rollback is issued to the server if no locking has been performed in a transaction.
- Cursors that have no locks placed (that is, cursors belonging to tables that have not been locked) are left open when a transaction is committed.
- C/SIDE makes use of outer and inner transactions. An outer transaction is the first transaction that takes place when the Transaction Type is changed from Browse, for example when running a report. Inner transactions are those that end with a `COMMIT`, for example, within the report, before the end of the outer transaction. The driver is given information about the outer and inner transactions in order to determine when rowsets should be closed and when data caches should be purged.

## SIFT

SIFT stands for sum index field technology. `SumIndexFields` allow sums of numbers that are stored in columns in tables to be calculated quickly – even when the table contains several thousand records. Each time you change the contents of a field in a column, the accumulated value is updated. The sum is updated continuously, so the program does not need to add all the entries together – it can simply add the newest figure to the sum that is already calculated. The updated sum can be seen every time you open a window, which contains a `FlowField` or set a filter on a balance field.

`FlowFields` are used to display amounts and quantities that must always be up-to-date. The calculation can be based on information that is stored in tables other than the one that contains the `FlowField`. `FlowFilters` are used to determine how much information the system will include when it calculates the contents of `FlowFilters`.

However, SIFT has been implemented very differently in the SQL Server Option for Dynamics NAV. This implementation involves creating a new table on SQL Server for every `SumIndexFields` that exists in a Dynamics NAV database table. The totals in the



SumIndexFields are therefore calculated in SQL Server tables. This means that there are more tables that must be updated and more filters that must be applied.

This can in turn result in poor performance. Therefore you should not create any FlowFields unless they are necessary and you should also give serious consideration to the design of any indexes and filters that you are going to implement. For example, you must give the SumIndexFields a unique name because SQL Server will create a table that is named after this field. No two objects can have the same name in Dynamics NAV.

For more information, see the section SIFT and the SQL Server Option for Dynamics NAV on page 486.

## D.2 A Brief History of Performance Improvements

The database driver has become increasingly complex because of the continuing need to improve performance.

### The Features and The Versions

This section contains details of the features that have been introduced to optimize performance, including the version of Dynamics NAV in which the changes were introduced.

#### Parameterization (2.50)

C/SIDE filters are not parameterized because the auto-parameterization feature in SQL Server 7.0 is believed to provide the necessary parameterization on the server. The development effort that is required to achieve the parameterization is quite high. All `INSERT`, `UPDATE` and `DELETE` statements are parameterized, along with some rowsets for navigating cursors. Subsequent tests have shown that the auto-parameterization feature in SQL Server 7.0 does not work – or at least works very conservatively – and it would therefore be necessary to do this in the driver.

#### Prepared Statements (2.50)

Prepared statements are used for all modifications except for the `DELETEALL` and `MODIFYALL` functions. Statements used for `GET` and `FIND( '= ' )` are also prepared.

#### Statement re-use (2.50)

Modification statements for `INSERT`, `DELETE` and `MODIFY` are re-used; however only two versions of `DELETE` and `MODIFY` are persistent; one with and one without the timestamp check. Cursor-based rowsets for all `FINDs` are re-used, along with those for `GET`. `SIFT` query rowsets are not re-used due to the isolated nature of the `SIFT` system.

#### Fetch Buffer Growth (2.50)

The buffering of rows for block rowsets is done by setting an initial buffer size based on the width (in bytes) of the table and some threshold values. As records are read from a rowset, the buffer grows steadily to reduce the number of fetches. This is not done immediately because the reading pattern for a particular rowset is unknown. Once the rowset is closed, the buffer is restored to its original size.

#### Paging in the User Interface (2.50)

When paging up or down in a regular table window, the form system makes requests both forwards and backwards even though you are only paging in the same direction. It also uses different records as reference points for requests for further records. This disturbs the basic sequential reading from a cursor and causes several rowsets to be executed when paging is being carried out. To avoid this, the rowset buffer layout has been extended to give a scroll window that can be read backwards, like a history of the current window. An additional anchor record is also maintained as well as the usual current record, to cater for the dual reference points used in the form. This allows a rowset to perform pure fetching when paging with the form system, utilizing the history buffer and current records.

**Preserving Rowsets during Modifications (2.50)**

When modifying a record, for example, in a loop, it is best to allow as many cursors as possible to remain open, including the cursor being used for the loop itself. This is possible for fast-forward and dynamic cursors, provided that the modification is not to a field in the current key, in which case the ordering of the record could change. Although these cursor types retrieve fresh data at fetch time, they maintain a memory buffer, which is not updated when the modification is performed. To allow modifications and deletions of a record and to keep these cursors open, the buffer is flagged for the record so that it cannot be visited again, but further records can be read. If the modification is to a key field, the cursor must be closed.

**Providing the `ISEMPTY` Alternative to `FIND` (2.50)**

The new `ISEMPTY` function utilizes an existing driver function and allows a less expensive, non-cursor, SQL statement to be used for determining whether or not a filtered set is empty.

**Client Caching (2.50)**

Records are cached on the client for `GET`, `FIND( '=' )`, `NEXT`, `SIFT` queries and `BLOB` data. This improves performance when re-reading these items but means that the data is not necessarily the most recent.

**Minimizing unnecessary Transactions (2.50)**

Status information is maintained by the driver to minimize the amount of server calls for transaction end blocks and isolation level changes. This significantly reduces the number of server calls, which can otherwise be made many times by `C/SIDE` without any logical necessity on the server.

**Using optimal `SIFT` queries (2.50)**

It was discovered that many `SIFT` queries that use the `OR` operator for bucket comparisons are using fairly expensive execution plans on the server. Tests showed that using the `UNION ALL` operator (with the necessary restructuring of the SQL statement) gives a much faster execution plan.

**Bulk Fetching during a Backup, and Batch Inserting during a Restore (2.50)**

Two internal functions are implemented to improve backup and restore performance. A bulk fetching function is built on the existing rowset functions to perform mass record fetching. A batch insert function is created to utilize batch inserts in the SQL Server ODBC driver, thereby reducing the number of server calls that must be made when many inserts are performed.

**Extended Parameterization (2.60.A)**

`C/SIDE` filters are parameterized giving significant performance benefits throughout the client. `SIFT` queries are also parameterized but they are still not re-used.

**Extended Preservation of Rowsets during Modifications (2.60.A)**

Modifications are made to extend the cursor types that can remain open during modifications.

### **A New Algorithm for Deleting Rowsets (2.60.A)**

The LRU algorithm that is used for deleting rowsets that are using cursors, when new rowsets are created, is replaced with a more complex algorithm. The new algorithm is introduced to prevent reports that have several loops, from deleting rowsets and using new rowsets to continue the looping. The problem is also related to having cursors used for `FIND( '- ' )` operations that only request one row. The new scheme looks at the state and usage of the cursor to determine if it should be deleted. This improves performance for reports that have several loops.

### **Using Single-row Rowsets for FIND (2.60.A)**

When a `FIND( '- ' )` is issued, the default of initially fetching several rows is changed so that only a single row is fetched. This is useful if the `FIND` will not retrieve further rows. If it does retrieve further rows, the fetch size is set to the normal initial size.

### **Modifying Fewer Fields (2.60.A)**

For a `MODIFY`, the SQL statement is changed so that it only updates those fields that have been changed. This means reading the modified record in advance, but gives a more efficient update especially where `SIFT` is concerned.

### **Client Analysis of Filters (2.60.D, 3.00)**

To avoid some specific problems with the SQL Server query optimizer, the `C/SIDE` filter is examined to determine if it defines an empty set. This analysis is done only for particular operators. As a result of this analysis, many such queries are not executed on the server.

### **Extended Statement Re-use (3.00)**

All modification statements including `DELETEALL` and `MODIFYALL` are now re-used. Rowsets that implement the `ISEMPTY` function, `BLOB` retrieval and `SIFT` queries are also re-used now.

### **Modified Threshold Values (3.00)**

The thresholds for buffer sizes, and the numbers of command and rowsets are adjusted after performance testing.

### **Client Caching of SIFT Queries on Base Tables (3.00)**

Sums performed on base tables, where the `MaintainSIFTIndex` property is set to `No`, are obtained and cached in a single server call.

### **Change to Prepared Statements (3.01)**

Statements that are used in cursor rowsets when performing `FIND( '- / + ' )` are now prepared, depending on the cursor type used. A known bug in the SQL Server ODBC driver means that preparing statements with certain cursor types, while using an auto-fetch feature, returns incorrect information to the client application. Since the problem does not occur with Fast Forward cursors, these can be prepared as long as the cursor type does not change after the first execution. Additionally, the statements for `ISEMPTY`, `BLOB` retrieval and `SIFT` queries are all prepared.

### **Change to Single-row Rowsets (3.01)**

The rules for using single-row rowsets are modified by using table and rowset statistics. If the table has recently experienced any modifications to key fields, or a `FIND( '- / + ' )`

rowset has not experienced a `NEXT` operation, single-row fetching will be used for the rowset instead of buffered fetching.

### Change to Rowset Closure and Cache Purging in Transactions (3.01)

The information about outer and inner transactions that is maintained by the C/SIDE database layer is now passed onto the driver at the end of the transaction. All caches are purged at the beginning of the first inner transaction, for example when a codeunit is run.

The driver now allows non-locking cursors, which are used by tables that are not locked, to remain open after a commit (but not a rollback). This improves batch job performance when commits are issued during the batch job because cursors that are used by looping tables that are not locked can continue to be used.

### Utilizing Faster SQL Statements (3.01)

Rowsets continue to optimize for the situation where a result set is opened because of a `FIND( '-/+ ' )` and the set is fully read to the end. However, rowset statistics are used to determine if a faster more efficient SQL statement can be used to satisfy the request that a rowset is currently servicing. The following schemes have been introduced for statements implementing `FIND( '-/+ ' )`, which replace the use of the original cursor in the rowset:

- If a rowset is mainly producing empty results, a SQL statement that implements the `ISEMPTY` function is used.
- If a rowset is mainly reading the first record only with no subsequent `NEXT` operations, a single-row default result set is used.
- If a rowset is reading records to the end of a set and the set is small, a buffered default result set is used.

### Extended Client Caching (3.01)

The results for many rowsets are now cached, including the situation where no record is found. Original cursor rowsets are not cached. They must be replaced by buffered default result sets in order to be cached.

### Change to Rowset Deletion (3.01)

The algorithm for deleting cursor rowsets is simplified to an LRU (least recently used) system as for non-cursor types. Testing found this to be the best overall scheme and it replaces the more complex system introduced in 2.6A. Since the number of rowset objects has significantly increased, the original problems seen in 2.6 will no longer occur for most typical batch jobs.

### Change to Firehose Rowsets (3.10)

The rules for using a firehose rowset are slightly modified. A single-row rowset is never converted to a firehose rowset. Furthermore, when a key is modified, all the existing firehose rowsets for the table are deleted so that they will not be re-used. As before, a firehose rowset will not be created when the table is undergoing key modifications.

### Change to Rowset Closure Due to Modifications (3.10)

More use is made of single-row rowsets to allow them to remain open after modifications are made to the table, and avoid having to create and execute new SQL statements. A single-row rowset can now survive an `INSERT`, `DELETEALL` or `MODIFYALL`

operation. These operations are treated in the same way as key modifications and deny the use of a firehose rowset.

### **Change to Rowset Memory Usage (3.10)**

The number of rowsets and commands available for a connection is based on the available physical memory as was the case in 3.01B, but now the number can change dynamically as the program runs. If the amount of memory available is reduced to a lower performance level than the current threshold, the command and rowset resources are deleted to stay within the new limits. If more memory is available, the performance level can increase to accommodate more resources. This memory checking is performed within the usual resource expiry sweep – every 5 minutes.

### **Change to Rowset Expiry (3.10)**

Rowsets are no longer checked for expiry prior to use. They can expire only within the resource expiry sweep, but only after 30 minutes of inactivity. If the status of a rowset is Open, it will never expire.

### **Change to Transaction End Markers in the User Interface (3.10)**

The user interface often begins a transaction when performing a lookup. Sometimes the transaction is ended with a rollback, and this closes all the active rowsets for this connection. These rollbacks are now changed to commits in order to preserve the open rowsets. The transaction itself has performed no work that needs to be committed or rolled back, so the actual type of the transaction end is not important.

### **Non-locked Rowsets Persist Beyond a Transaction (3.10)**

Rowsets that are created for a non-locked table (i.e. non-locked rowsets) can persist beyond a transaction, even when the table was locked within the transaction. Previously, once a table was locked, all its rowsets were closed on commit, including non-locked rowsets that were opened before the table lock.

### **Automatic Bulk Inserts (3.10)**

The driver automatically buffers record insertions and sends them in batches to the server, in a similar manner as it does when restoring a database. Special operations are performed for tables that contain SIFT keys to further optimize the use of the SIFT triggers. There are various criteria that must be met before automatic bulk inserts are available.

## INDEX

- A**
- ActiveX .....364
- ANSI NULL default (database option) ... 20
- application
  - application object ..... 47
  - design ..... 52
  - design (reference to books) ..... 56
- Application Area Name (field) ..... 469
- array .....304
- ascending order ..... 85
- Auto close (database option) ..... 20
- Auto shrink (database option) ..... 21
- automation ..... 364, 370
  - using Microsoft Word .....370
  - where to put code ..... 371
- AutoReplace ..... 413
- AutoSave ..... 413
- AutoUpdate ..... 413
  
- B**
- backup and restore facilities
  - Dynamics NAV Database Server ..... 23
  - SQL Server Option ..... 24
- bigint
  - SQL Server data type ..... 79
- BigInteger
  - C/AL data type ..... 79
  - field data type ..... 65
- binary
  - C/AL data type ..... 79, 80
  - SQL Server data type ..... 80
- binary (field data type) ..... 64
- bit
  - SQL Server data type ..... 80
- BLOB ..... 64
  - C/AL data type ..... 79, 80, 302
  - field data type ..... 64
- boolean
  - C/AL data type ..... 79, 80
  - displaying ..... 177
  - field data type ..... 64
- bound control ..... 141, 165
- bound form ..... 140
- breakpoints ..... 352
  - in the C/AL Editor ..... 356
  - storage in XML file ..... 357
- Breakpoints virtual table ..... 356
- bugs ..... 346
  
- C**
- C/AL
  - bugs ..... 346
  - comments ..... 321
  - constants ..... 305
  - control language ..... 314
  - data types ..... 299
  - debugging ..... 346
  - defined ..... 48
  - dialogs ..... 340
  - editor ..... 45, 280
  - essential functions ..... 328
  - expressions ..... 297
  - function calls ..... 312
  - Globals ..... 285
  - Locals ..... 286
  - operator hierarchy ..... 312
  - operators ..... 310
  - program logic errors ..... 351
  - repetitive statements ..... 316
  - reusing code ..... 325
  - run-time errors ..... 347
  - statements ..... 297
  - Symbol Menu ..... 287, 478
  - syntax errors ..... 346
  - where to place ..... 324
- C/AL Editor
  - shortcut keys ..... 282
  - using ..... 282
- C/AL functions
  - CALCDATE ..... 475
  - CALCFIELDS ..... 335
  - CALCSUMS ..... 336
  - CLEAR ..... 292
  - CLEARALL ..... 293
  - CONFIRM ..... 342
  - DELETE ..... 333
  - DELETEALL ..... 334
  - ERROR ..... 342
  - FIELDERROR ..... 336
  - FIELDNAME ..... 338
  - FIND ..... 328
  - GET ..... 328
  - GETRANGEMAX ..... 331
  - GETRANGEMIN ..... 331
  - INIT ..... 338
  - INSERT ..... 332
  - LOCKTABLE ..... 335
  - MESSAGE ..... 341
  - MODIFY ..... 332
  - MODIFYALL ..... 333
  - NEXT ..... 329
  - overview ..... 328
  - SETCURRENTKEY ..... 329
  - SETFILTER ..... 330
  - SETRANGE ..... 330
  - STRMENU ..... 343
  - TESTFIELD ..... 338
  - VALIDATE ..... 339

- C/AL run-time errors ..... 349
- cache
  - commit cache ..... 562
  - DBMS cache ..... 560
- CalcFormula (property) ..... 90
- calculation formula ..... 90
- CAPTIONAREA ..... 534
- CaptionClassTranslate trigger ..... 533
- CAPTIONREF ..... 534
  - syntax ..... 535
- card form ..... 142
  - creating ..... 143
- CASE ..... 316
- char
  - SQL Server data type ..... 79
- char (C/AL data type) ..... 300
- check box to display booleans ..... 177
- Client Monitor ..... 122
  - additional parameters for SQL Server Option ..... 123
- client/server environment
  - Dynamics NAV ..... 4
- code
  - C/AL data type ..... 79, 301
  - field data type ..... 64
- Code Coverage Tool ..... 360
- code examples
  - DimCaptionClassTranslate ..... 537
  - VATCaptionClassTranslate ..... 543
- code fields ..... 80
- codeunit
  - accessing functions ..... 289
  - assigning ..... 290
  - compiling ..... 287
  - creating ..... 282
  - defined ..... 280
  - limitations ..... 294
  - running ..... 281
  - saving ..... 287
  - single instance ..... 293
  - specifications ..... 580
- codeunit (C/AL data type) ..... 302
- collations
  - SQL Server Option ..... 18
- Color tool ..... 164
- column ..... 60
- COM ..... 364
  - and C/SIDE ..... 366
  - automation ..... 370
  - CREATE ..... 376
  - custom controls ..... 394
  - declaring variables for external components ..... 386
  - default members ..... 377
  - enumerations ..... 368
  - exceptions ..... 400
  - limitations on event triggers ..... 390
  - Microsoft Excel ..... 381
  - OCX ..... 394
  - receiving events in C/SIDE ..... 386
  - restrictions on incoming data ..... 391
  - terminology ..... 364
  - the WithEvents property ..... 388
  - USERDEF ..... 368
  - using Microsoft Word ..... 370
- commit
  - in C ..... 525
  - in C/AL ..... 525
- commit cache ..... 562
- COMMIT() ..... 526
- company ..... 94
- company level security ..... 32
- Company system table ..... 115
- complex data types ..... 302
- compound statement ..... 314
- concurrency ..... 522
- conditional statements ..... 314
- configuration parameters
  - SQL Server Option ..... 568
- consistency ..... 522
- constant ..... 305
  - text ..... 283, 305
- container control ..... 140, 155
  - frame ..... 179
- control
  - adding ..... 157
  - aligning ..... 162
  - bitmaps ..... 182
  - bound ..... 141, 165
  - branch ..... 140
  - changing caption ..... 165
  - changing name ..... 165
  - check box ..... 177
  - Color tool ..... 164
  - command button ..... 156, 178
  - container ..... 140, 155, 179
  - container control selection ..... 161
  - control branch selection ..... 161
  - data ..... 155
  - data types ..... 155
  - defined ..... 48
  - display properties ..... 166
  - displaying numbers ..... 166
  - Font tool ..... 164
  - formatting data ..... 166
  - frame ..... 155, 179
  - image ..... 155
  - in reports ..... 213
  - indicator ..... 182
  - input ..... 167
  - label ..... 155
  - menu button ..... 156, 196
  - menu item ..... 156
  - moving ..... 162
  - multiple selection ..... 160
  - option button group ..... 175



- option drop-down .....172
- picture box .....181
- properties of, in forms .....165
- shape ..... 155, 180
- sizing .....163
- static .....155
- subform .....156
- tab control ..... 155, 183
- table box ..... 156, 183
- text box .....157, 158, 165, 171
- toolbox .....157
- tools .....164
- ToolTip .....167
- types of .....155
- unbound ..... 141, 165
- control types .....155
- CREATE
  - automation variable ..... 370, 376
- creating a database
  - Dynamics NAV Database Server ..... 14
  - SQL Server Option ..... 17
- custom control .....364
  - developing .....402
  - using in C/SIDE .....394
- D**
- data .....63
- data container .....156
- data controls .....155
- data integrity ..... 520, 524
- data item ..... 212, 213
  - data model ..... 228, 236, 242
  - dataport .....406
  - defined .....48
  - properties .....226
- data model .....53
  - dataport .....407
- data type
  - BigInteger (field) .....65
  - binary (field) .....64
  - BLOB (C/AL) .....302
  - BLOB (field) .....64
  - boolean (field) .....64
  - char (C/AL) .....300
  - code (C/AL) .....301
  - code (field) .....64
  - codeunit (C/AL) .....302
  - complex (C/AL) .....302
  - controls and data types .....155
  - date (C/AL) .....300
  - date (field) .....64
  - DateFormula .....475
  - dateformula (C/AL) .....302
  - dateformula (field) .....65
  - DateTime (field) .....65
  - datetime (SQL) .....64
  - decimal (C/AL) .....300
  - decimal (field) .....63
  - decimal (SQL) .....63
  - dialog (C/AL) .....302
  - Duration (field) .....65
  - field data type .....65
  - fieldref (C/AL) .....303
  - file (C/AL) .....302
  - form (C/AL) .....302
  - fundamental .....299
  - GUID (C/AL) .....302
  - GUID (field) .....65
  - image (SQL) .....64
  - instream and outstream (C/AL) .....303
  - integer (field) .....63
  - integer (SQL) .....63
  - keyref (C/AL) .....303
  - mixing .....506
  - option (C/AL) .....299
  - option (field) .....63
  - recordID (C/AL) .....303
  - recordref (C/AL) .....303
  - report (C/AL) .....302
  - RowID (field) .....65
  - table fields .....63
  - tablefilter (C/AL) .....303
  - text (field) .....63
  - time (field) .....64
  - tinyint (SQL) .....64
  - varbinary (SQL) .....64
  - varchar (SQL) .....63, 64
  - variant (C/AL) .....303
- data version
  - defined .....521
  - historical .....523
  - storage of .....522
- database
  - defined .....49
  - design .....52
  - design (reference to books) .....56
  - logical .....49
  - physical .....49
- Database File virtual table .....128
- Database Key Groups system table ....116
- database level security .....30
- database logins .....32
- database options
  - ANSI NULL default .....20
  - Auto close .....20
  - Auto shrink .....21
  - Recovery model .....19
  - Recursive triggers .....20
  - SQL Server Option .....19
  - Torn page detection .....21
- dataport
  - AutoReplace .....413
  - AutoSave .....413
  - AutoUpdate .....413
  - combining export and import .....430
  - data item .....406

- data item properties ..... 413
- data model ..... 407
- description ..... 406
- designing ..... 411
- dynamic dataport example ..... 430
- export examples ..... 417
- export, fixed format ..... 417
- export, variable format ..... 423
- external file ..... 407
- field ..... 406
- field properties ..... 414
- FileFormat property ..... 412
- fixed format export ..... 417
- fixed format import ..... 425
- import examples ..... 425
- import, fixed format ..... 425
- import, variable format ..... 428
- logical design ..... 407
- property ..... 407, 411
- request form ..... 406
- running ..... 408, 410
- trigger ..... 407, 415
- variable format export ..... 423
- variable format import ..... 428
- Dataport Designer ..... 45
- date
  - C/AL data type ..... 79, 80, 300
  - field data type ..... 64, 82
- Date virtual table ..... 118
- DateFormula
  - C/AL data type ..... 79
- dateformula
  - C/AL data type ..... 302
  - field data type ..... 65
- DateTime
  - C/AL data type ..... 79
  - field data type ..... 65
- datetime
  - SQL Server data type ..... 79
- DBL\_BWT ..... 525
- DBL\_EWT ..... 525
- DBMS ..... 520
  - cache ..... 560
  - specifications ..... 578
- deadlock detection ..... 524
- debugger
  - activating ..... 352
  - breakpoints stored in XML file ..... 357
  - code coverage ..... 360
  - interface ..... 353
  - menus ..... 353
  - overview of shortcut keys ..... 358
  - running on Dynamics NAV Application Server ..... 353
  - setting breakpoints in the C/AL Editor .. 356
  - storage of information in the fin.zup file 358
  - the Breakpoints virtual table ..... 356
  - toolbar ..... 354
  - windows ..... 355
- decimal
  - C/AL data type ..... 79, 80, 300
  - field data type ..... 63
  - SQL Server data type ..... 79
- default members (COM) ..... 377
- delayed write back ..... 562
- deleting language ..... 468
- descending order ..... 85
- Designer
  - Dataport ..... 45
  - Form ..... 45
  - Navigation Pane ..... 45
  - Report ..... 45
  - Table ..... 45
  - XMLport ..... 45
- dialog ..... 340
  - C/AL data type ..... 302
- DimCaptionClassTranslate ..... 537
- DIMCAPTIONREF ..... 535
- DIMCAPTIONTYPE ..... 535
- dimension area ..... 535
- document ..... 264
- Documentation section ..... 280
- drill-down
  - form ..... 193
- Drive virtual table ..... 120
- Duration
  - C/AL data type ..... 79
  - field data type ..... 65
- Dynamics NAV
  - client/server environment ..... 4
  - functional areas and granules ..... 36
  - initiating the security system ..... 34
  - installing ..... 6
  - installing Dynamics NAV Database Server 8
  - installing multiple Dynamics NAV Database Servers ..... 11
  - installing SQL Server Option ..... 10
  - installing the client ..... 7
  - security, different levels ..... 30
- Dynamics NAV C/SIDE client
  - installing ..... 7
- Dynamics NAV Database Server
  - backup and restore facilities ..... 23
  - creating a database ..... 14
  - installing ..... 8
  - installing multiple instances ..... 11
  - locking in ..... 527
  - number sorting ..... 515
  - running as a service ..... 9
  - SIFT ..... 25, 484
  - troubleshooting the database connection 11

- Dynamics NAV debugger .....352
- Dynamics NAV license
  - granules ..... 36
- Dynamics NAV security system
  - different levels ..... 30
  - initiating ..... 34
  - things to remember ..... 34
- E**
- editor
  - purpose ..... 45
  - using ..... 280
- Entity-Relationship (ER) model ..... 53
- enumerations ..... 368
- Event Designer (XMLports) ..... 452
- Events
  - receiving ..... 386
- EXIT ..... 319
- expression
  - basic elements of ..... 305
  - defined ..... 297
  - evaluation ..... 298, 506
- extended stored procedure
  - adding to SQL Server Option ..... 10
- external tools
  - accessing table data with ..... 82
- F**
- Field
  - virtual table ..... 130
- field
  - dataport ..... 406
  - defined ..... 48
  - illustration ..... 60
  - property ..... 68
  - trigger ..... 96
- Field Menu
  - in reports ..... 218
- Field Virtual Table ..... 130
- Field virtual table ..... 130
- fieldref
  - C/AL data type ..... 303
- file (C/AL data type) ..... 302
- File virtual table ..... 120
- FileFormat property ..... 412
- filters
  - and number sorting ..... 517
- float
  - SQL Server data type ..... 80
- FlowField
  - calculation formula ..... 90
  - table filter ..... 92
- FlowFilter field ..... 87
- Font tool ..... 164
- FOR TO/DOWNTO ..... 317
- Form ..... 469
- form
  - bound ..... 140
  - card form ..... 142, 143
  - closing ..... 150
  - compiling ..... 150
  - creating ..... 142, 143, 147, 149
  - description ..... 140
  - design ..... 140, 187
  - drill-down ..... 193
  - Form Wizard ..... 142
  - lookup ..... 192
  - main form ..... 186
  - running ..... 150, 195
  - saving ..... 150
  - specifications ..... 580
  - subform ..... 186
  - tabular form ..... 142
  - test-compiling ..... 150
  - unbound ..... 140
- form (C/AL data type) ..... 302
- form design
  - Color tool ..... 164
  - Font tool ..... 164
  - toolbox ..... 157
  - tools ..... 164
- Form Designer ..... 45, 140
- Form Wizard ..... 142, 143
  - creating a card form ..... 143
  - creating a tabular form ..... 147
- function
  - accessing in codeunit ..... 289
  - C/AL ..... 280
  - creating ..... 283
- functional areas and granules ..... 36
- fundamental data types ..... 299
- G**
- Globals ..... 285
- granules
  - Dynamics NAV license ..... 36
- granules and functional areas ..... 36
- graph
  - creating with Microsoft Excel ..... 381
- groups in reports ..... 242
- GUID
  - C/AL data type ..... 79, 302
  - field data type ..... 65
- H**
- HotCopy
  - server based backup ..... 24
- I**
- ID number ..... 44
- IF THEN ELSE ..... 315
- image
  - SQL Server data type ..... 79
- index hinting
  - SQL Server Option ..... 568
- installing
  - Dynamics NAV ..... 6
  - Dynamics NAV C/SIDE client ..... 7

- Dynamics NAV Database Server ..... 8
- multiple Dynamics NAV Database Servers  
11
- SQL Server Option ..... 10
- instream and outstream
  - C/AL data type ..... 303
- integer
  - C/AL data type ..... 79, 80
  - field data type ..... 63
  - SQL Server data type ..... 79, 80
- Integer virtual table ..... 119
- integration options
  - Maintain Relationships ..... 22
  - Maintain Views ..... 22
  - SQL Server Option ..... 22
  - Synchronize Table Relationships ..... 22
- integrity ..... 520
- K**
- key
  - defined ..... 48
  - discussed ..... 71
  - groups ..... 116
  - in ER model ..... 54
  - list ..... 71, 74, 75
  - performance ..... 74
  - primary ..... 71
  - property ..... 75
  - secondary ..... 72
  - SumIndexFields ..... 484
- Key virtual table ..... 130
- keyref
  - C/AL data type ..... 303
- L**
- Language ..... 469
- language
  - adding ..... 467
  - deleting ..... 468
  - multiple document ..... 471
- language ID ..... 469
- language layer ..... 467
- lazy write ..... 562
- license file
  - per database ..... 22
- limitations
  - event triggers ..... 390
- linked objects
  - description ..... 105
  - requirements ..... 106
- Locals ..... 286
- lock granularity
  - SQL Server Option ..... 570
- locking
  - a comparison of Dynamics NAV Database  
Server and SQL Server ..... 527
  - in Dynamics NAV Database Server ..... 527
  - in SQL Server ..... 528
- locks ..... 524
- log file ..... 521
- logical database ..... 49
- lookup ..... 189, 191
  - form ..... 192
  - table relation ..... 191
- looping (C/AL) ..... 316
- M**
- main form ..... 186
  - design ..... 187
- Maintain Relationships (integration option)  
22
- Maintain Views (integration option) .... 22
- Member Of system table ..... 113
- menu
  - design ..... 196, 197
  - menu button ..... 196
  - menu item ..... 196
  - menu line ..... 197
  - shortcut key ..... 197
- MenuSuite object ..... 456
  - customizing ..... 457
  - exporting ..... 460
  - levels ..... 458
  - Navigation Pane ..... 456
  - Navigation Pane Designer ..... 456
  - upgrading ..... 462
- Microsoft Enterprise Manager ..... 82
- Microsoft Excel ..... 381
- Microsoft Word ..... 370
- money
  - SQL Server data type ..... 80
- Monitor virtual table ..... 121
- multilanguage ..... 465
  - C/ODBC ..... 470
  - date formulas ..... 475
  - multiple document languages ..... 471
  - SQL views ..... 82
  - text constants ..... 466, 472
- N**
- Navigation Pane ..... 456
- Navigation Pane Designer ..... 45, 456
- nchar
  - SQL Server data type ..... 79
- nonprinting report ..... 270
- ntext
  - SQL Server data type ..... 80
- Number ..... 302
- number sorting
  - a definition of ..... 516
  - and filters ..... 517
  - differences between Dynamics NAV  
Database Server and SQL Server .... 516
  - principles ..... 517
  - recommendations ..... 516

- numbering principles .....517
- numeric
  - SQL Server data type ..... 80
- nvarchar
  - SQL Server data type ..... 79
- O**
- object level security ..... 32
- OCX .....364
  - developing .....402
  - registering .....394
  - using in C/SIDE .....394
- OLE ..... 364
- OnRun section .....280
- operators
  - arithmetic (type conversion) .....509
  - hierarchy .....312
  - logical (type conversion) .....509
  - relational (type conversion) .....508
  - using in C/AL .....310
- option
  - C/AL data type ..... 79, 80, 299
  - field data type ..... 63
- option (C/AL data type) ..... 79
- order
  - ascending ..... 85
  - descending ..... 85
- P**
- performance
  - C/AL functions and SQL Server . 565, 567
  - command buffer .....563
  - commit cache .....562
  - DBMS cache .....560
  - keys ..... 74
  - keys and queries .....565
  - measuring .....121
- performance monitoring
  - both server options .....26
- Permission system table .....114
- physical database ..... 49
- program logic errors ..... 351
- Properties window ..... 141
- property
  - CalcFormula ..... 90
  - control .....141, 152, 165, 166, 167
  - control, general properties .....154
  - dataport ..... 407, 411
  - defined ..... 48
  - fields ..... 68
  - form ..... 141, 152
  - inheriting .....152
  - key ..... 75
  - list of, in forms ..... 153
  - list of, in reports .....213
  - list of, in tables ..... 66
  - Properties window .....141
  - report ..... 224, 225
  - TableRelation ..... 98
- R**
- read consistency ..... 522
- real
  - SQL Server data type ..... 80
- record
  - defined ..... 60
- record level security ..... 34
  - applying in a posting scenario ..... 549
  - applying to complex forms ..... 554
  - applying to user-defined variables .. 554
  - calcsums and Find('+') ..... 547
  - codeunits ..... 547
  - debugging the code ..... 551
  - entering the filters ..... 549
  - forms reports and dataports ..... 547
  - giving a codeunit indirect read permission 552
  - giving the user indirect read permission . 550
  - implementation guidelines ..... 548
  - matching filters with keys ..... 546
  - performance considerations ..... 546
  - setpermissionfilter ..... 546
  - supporting on SQL Server option ... 546
  - the RESET function and filters ..... 557
  - using setpermissionfilter in C/AL .... 556
  - using with variables ..... 546
- recordID
  - C/AL data type ..... 303
- recordref
  - C/AL data type ..... 303
- Recovery model (database option) ..... 19
- Recursive triggers (database option) ... 20
- registering an OCX ..... 394
- relationship ..... 98
- REPEAT UNTIL ..... 318
- report
  - closing ..... 221
  - control ..... 213
  - controlling output ..... 259
  - data item ..... 212, 213, 228, 236, 242
  - data item properties ..... 226
  - data item triggers ..... 249
  - data model .....228, 236, 242
  - definition ..... 212
  - designing sections .....231, 238, 245
  - documents ..... 264
  - execution ..... 215
  - Field Menu ..... 218
  - flow chart ..... 215
  - grouping ..... 242
  - nonprinting ..... 270
  - properties ..... 213, 224
  - property, description of ..... 225
  - report description ..... 212
  - report triggers ..... 249
  - request form ..... 213
  - Request Options Form Designer 218, 219

- running .....221
- saving .....221
- section ..... 212, 214, 231
- Section Designer .....218
- section properties .....227
- section triggers ..... 249
- specifications ..... 580
- totaling .....242
- totals .....244
- triggers ..... 213, 249
- types and naming .....274
- virtual tables in reports .....250
- report (C/AL data type) .....302
- report description .....212
- Report Designer ..... 45
- request form .....213
  - dataport .....406
- Request Options Form Designer .. 218, 219
- RequestForm .....213
  - defined ..... 48
- row .....60
- RowID
  - C/AL data type ..... 79
  - field data type ..... 65
- run-time errors ..... 327, 347
  - avoiding .....349
  - finding and correcting .....351
- S**
- section ..... 212, 214
  - defined ..... 48
  - designing ..... 231, 238, 245
  - properties .....227
  - triggers .....249
- Section Designer .....218
- security
  - company level .....32
  - database level .....30
  - database logins .....32
  - initiating the security system .....34
  - object level .....32
  - record level .....34
  - Windows logins .....31
- server based backup
  - HotCopy .....24
- Server virtual table .....131
- Session virtual table .....125
- shortcut key .....197
- shortcut keys
  - in the debugger .....358
- SID - Account ID virtual table .....133
- SIFT
  - Dynamics NAV Database Server ..25, 484
  - SQL Server Option ..... 25, 486–504
  - tables on SQL Server .....78
- smalldatetime
  - SQL Server data type ..... 80
- smallint
  - SQL Server data type ..... 80
- smallmoney
  - SQL Server data type ..... 80
- specifications
  - codeunit ..... 580
  - DBMS ..... 578
  - forms ..... 580
  - reports ..... 580
  - tables ..... 579
- SQL Server
  - enabling a trace flag ..... 15
  - locking in ..... 528
  - SIFT tables ..... 78
- SQL Server Option
  - adding extended stored procedure .. 10
  - additional parameters in the Client Monitor ..... 123
  - backup and restore facilities ..... 24
  - collations ..... 18
  - configuration parameters ..... 568
  - creating a database ..... 17
  - database options ..... 19
  - how code fields work in ..... 64
  - index hinting ..... 568
  - installing ..... 10
  - integration options ..... 22
  - linked objects ..... 105
  - lock granularity ..... 570
  - maintaining table relationships ..... 101
  - SIFT .....25, 486–504
  - SIFT buckets ..... 488
  - SIFT table ..... 487
  - SIFT table, extended key ..... 495
  - SIFT table, indexes ..... 495
  - SIFT table, layout ..... 495
  - SIFT table, optimizing ..... 504
  - SIFT Trigger ..... 487
  - SIFT, costs and benefits ..... 503
  - SIFT, Date fields ..... 491
  - SIFT, DateTime fields ..... 494
  - SIFT, deleting records ..... 500
  - SIFT, updating the base table ..... 498
  - sorting numerical values in code fields .. 515
  - supporting record level security ..... 546
- statement
  - compound ..... 314
  - conditional ..... 314
  - defined ..... 297
- String ..... 302
- subform ..... 156, 186
  - design ..... 187
- SumIndexField
  - and FlowFields ..... 87
  - and SQL Server ..... 529
  - defined ..... 484

- Symbol Menu ..... 287, 478
  - Synchronize Table Relationships (integration option) ..... 22
  - syntax errors ..... 346
  - system table ..... 112
    - Company ..... 115
    - Database Key Groups ..... 116
    - Member Of ..... 113
    - Permission ..... 114
    - User ..... 113
    - User Role ..... 114
    - Windows Access Control ..... 115
    - Windows Login ..... 115
  - system-defined variable ..... 306, 326
- T**
- tab control ..... 155
  - table
    - accessing data with external tools .... 82
    - adding records ..... 86
    - defined ..... 60
    - description ..... 60
    - modifying the design of ..... 104
    - property ..... 66
    - relationship ..... 98
    - saving ..... 84
    - specifications ..... 579
    - system ..... 112
    - temporary ..... 110
    - trigger ..... 96
    - validating relationship ..... 191
    - viewing data in ..... 84
    - virtual ..... 117
  - Table Designer ..... 45, 62
  - Table Information virtual table ..... 129
  - table property
    - LinkInTransaction ..... 105
    - LinkableObject ..... 105
    - list ..... 66
    - viewing and modifying ..... 66
  - table relation ..... 191
    - and assist edit ..... 100
    - example ..... 100
  - TableFilter
    - C/AL data type ..... 79
    - data type (field) ..... 65
  - tablefilter
    - C/AL data type ..... 303
  - TableRelation property ..... 98
  - tabular form ..... 142
    - creating ..... 147
  - template ..... 48
  - temporary table ..... 110
  - text
    - C/AL data type ..... 79, 80
    - field data type ..... 63
    - SQL Server data type ..... 80
  - text box
    - adding ..... 157, 158
    - adding a label ..... 165
    - calculation ..... 171
    - multiple lines ..... 171
  - text constant ..... 283, 305
  - Text Constants ..... 466
  - time
    - C/AL data type ..... 79
    - field data type ..... 64, 82
  - tinyint
    - SQL Server data type ..... 79, 80
  - toolbox ..... 157
  - ToolTip ..... 167
  - Torn page detection (database option) . 21
  - totals
    - and sections ..... 244
  - totals in reports ..... 242
  - Trace flag
    - enabling on SQL Server ..... 15
  - transaction ..... 520
  - trigger
    - control ..... 141, 201
    - data item ..... 249
    - dataport ..... 407, 415
    - defined ..... 48
    - field ..... 96
    - form ..... 141, 200
    - overview of control triggers ..... 201
    - overview of form triggers ..... 200
    - overview of report triggers ..... 249
    - report ..... 213, 249
    - table ..... 96
  - triggers
    - CaptionClassTranslate ..... 533
  - type conversion ..... 506
- U**
- unbound control ..... 141
    - changing to bound ..... 165
  - unbound form ..... 140
  - uniqueidentifier
    - SQL Server data type ..... 79, 80
  - User Role system table ..... 114
  - User SID virtual table ..... 134
  - User system table ..... 113
  - USERDEF ..... 368
  - user-defined variable ..... 306
- V**
- varbinary
    - SQL Server data type ..... 79
  - varchar
    - SQL Server data type ..... 79
    - SQLdata type ..... 64
  - variable
    - arrays ..... 304
    - assignment ..... 308
    - creating ..... 282

CurrForm .....	326
CurrReport .....	326
initialization .....	308
naming conventions .....	306
Rec .....	326
system-defined .....	306, 326
user-defined .....	306
xRec .....	326
variant	
C/AL data type .....	303
VATCAPTIONREF .....	536
VATCAPTIONTYPE .....	536
virtual table .....	117
Breakpoints .....	356
Database File .....	128
Date .....	118
Drive .....	120
Field .....	130
File .....	120
Integer .....	119
Key .....	130
Monitor .....	121
Server .....	131
Session .....	125
SID - Account ID .....	133
Table Information .....	129
User SID .....	134
Windows Group Member .....	133
Windows Object .....	132
virtual tables .....	250
<b>W</b>	
WHILE DO .....	318
Windows Access Control system table ..	115
Windows Group Member virtual table ..	133
Windows Login system table .....	115
Windows logins .....	31
Windows Object virtual table .....	132
WITH .....	320
write locks .....	524
write transaction .....	520
<b>X</b>	
XMLport Designer .....	45
XMLports	
designing .....	442
Event Designer .....	452
examples .....	445
saving, compiling and running .....	440
validating data .....	450
xp_ndo.dll	
adding .....	10