

# Application Designer's Guide

**MICROSOFT BUSINESS SOLUTIONS—NAVISION**



# APPLICATION DESIGNER'S GUIDE



## **DISCLAIMER**

This material is for informational purposes only. Microsoft Business Solutions ApS disclaims all warranties and conditions with regard to use of the material for other purposes. Microsoft Business Solutions ApS shall not, at any time, be liable for any special, direct, indirect or consequential damages, whether in an action of contract, negligence or other action arising out of or in connection with the use or performance of the material. Nothing herein should be construed as constituting any kind of warranty.

## **COPYRIGHT NOTICE**

Copyright © 2003 Microsoft Business Solutions ApS, Denmark.

## **TRADEMARK NOTICE**

Microsoft, Great Plains, bCentral and Microsoft Windows 2000 are either registered trademarks or trademarks of Microsoft Corporation or Great Plains Software, Inc. in the United States and/or other countries. Great Plains Software, Inc. and Microsoft Business Solutions ApS are wholly owned subsidiaries of Microsoft Corporation. Navision is a registered trademark of Microsoft Business Solutions ApS in the United States and/or other countries. The names of actual companies and products mentioned herein may be the trademarks of their respective owners. No part of this document may be reproduced or transmitted in any form or by any means, whole or in part without the prior written permission of Microsoft Business Solutions ApS. Information in this document is subject to change without notice. Any rights not expressly granted herein are reserved.

Published by Microsoft Business Solutions ApS, Denmark.

Published in Denmark 2003.

DocID: NA-370-DVG-001-v01.00-W1W1

## PREFACE

This manual provides information about the C/SIDE<sup>®</sup> development system. It is part of the documentation and Help materials for Microsoft<sup>®</sup> Business Solutions–Navision<sup>®</sup>.

When you create a C/SIDE application, you combine five types of application objects into a whole that solves a business problem. Each of the five types of application objects has its own part in this manual. The order in which the parts appear corresponds to the order in which you are most likely to need them when you design a new application.

The manual is divided into seven 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: Microsoft Business Solutions–Navision Database Server***

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

### ***Installation & System Management: Microsoft Business Solutions–Navision SQL Server Option***

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

### ***Installation & System Management: Microsoft Business Solutions–Navision Application Server***

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

### ***Development Guide for Communication Components***

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

## TABLE OF CONTENTS

<b>PART 1</b>	<b>FUNDAMENTALS</b>	<b>1</b>
	<b>Chapter 1 C/SIDE Fundamentals</b>	<b>1</b>
	The C/SIDE User Interface	2
	What Is a C/SIDE Application?	5
	The Physical and the Logical Database	8
	<b>Chapter 2 Designing a C/SIDE Application</b>	<b>11</b>
	Introduction to C/SIDE Application Design	12
<b>PART 2</b>	<b>TABLES</b>	<b>17</b>
	<b>Chapter 3 Table Fundamentals</b>	<b>19</b>
	What Is a Table?	20
	What Are Keys?	26
	Identifiers, Data Types and Data Formats in the SQL Server Option for Navision.	32
	Saving, Viewing, and Sorting Data	38
	Dividing the Database into Companies	41
	Special Table Fields	42
	<b>Chapter 4 Customizing and Maintaining Tables</b>	<b>49</b>
	Viewing and Modifying Properties	50
	Using Table and Field Triggers	58
	Setting Relationships Between Tables	60
	Changing Tables That Contain Data	65
	Linked Objects	66
	<b>Chapter 5 Special C/SIDE Tables</b>	<b>71</b>
	What Is a Temporary Table?	72
	What Is a System Table?	74
	What Is a Virtual Table?	79
	Overview of C/SIDE Virtual Tables	80
<b>PART 3</b>	<b>FORMS</b>	<b>97</b>
	<b>Chapter 6 Form Fundamentals</b>	<b>99</b>
	What Are Forms?	100
	Creating Forms	102
	Selecting, Moving and Adjusting Controls	108
	Saving, Compiling and Running Forms	113
	<b>Chapter 7 Designing Forms</b>	<b>115</b>
	Form and Control Properties	116
	Types of Controls	119

Adding Controls . . . . .	121
Tools for Customizing Controls . . . . .	124
Setting Control Properties . . . . .	125
How to Use Controls in Applications . . . . .	129
<b>Chapter 8 Extending the Functionality of Your Forms . . . . .</b>	<b>139</b>
Main Forms and Subforms . . . . .	140
Looking Up Values and Validating Entries . . . . .	143
Drilling Down to the Underlying Transactions . . . . .	147
Launching Another Form . . . . .	149
Designing Menu Buttons . . . . .	150
Form and Control Triggers . . . . .	154
<b>PART 4 REPORTS . . . . .</b>	<b>157</b>
<b>Chapter 9 Report Fundamentals . . . . .</b>	<b>159</b>
What Are Reports? . . . . .	160
What Happens When a Report Runs? . . . . .	164
The Report Designer . . . . .	167
Saving, Compiling and Running Reports . . . . .	170
<b>Chapter 10 Designing Reports . . . . .</b>	<b>173</b>
Report Properties . . . . .	174
Designing a Simple Report . . . . .	178
Designing a More Advanced Report . . . . .	186
<b>Chapter 11 Extending the Functionality of Your Reports . . . . .</b>	<b>191</b>
Grouping and Totaling . . . . .	192
Triggers in Reports . . . . .	198
Advanced Sample Reports . . . . .	200
<b>PART 5 CODEUNITS . . . . .</b>	<b>215</b>
<b>Chapter 12 Codeunit Fundamentals . . . . .</b>	<b>217</b>
What Is a C/SIDE Codeunit? . . . . .	218
Creating Codeunits . . . . .	220
Using Codeunits . . . . .	229
<b>Chapter 13 Introducing the C/AL Language . . . . .</b>	<b>233</b>
What Can You Do with C/AL? . . . . .	234
What Are Statements, Expressions, and Operators? . . . . .	235
Introducing the Elements of C/AL Expressions . . . . .	243
The C/AL Control Language . . . . .	252
<b>Chapter 14 Using C/AL . . . . .</b>	<b>261</b>
Overview . . . . .	262
System-Defined Variables . . . . .	264
Handling Runtime Errors . . . . .	265



	The Essential C/AL Functions . . . . .	266
	<b>Chapter 15 Debugging C/AL Code . . . . .</b>	<b>279</b>
	What Are Bugs? . . . . .	280
	Syntax Errors . . . . .	281
	Runtime Errors . . . . .	282
	Program Logic Errors . . . . .	287
	The Microsoft Business Solutions–Navision Debugger . . . . .	289
	The Code Coverage Tool . . . . .	297
	<b>Chapter 16 Extending C/AL . . . . .</b>	<b>299</b>
	What Is COM? . . . . .	300
	Using COM Technologies in C/SIDE . . . . .	302
	Using C/SIDE as an Automation Controller . . . . .	306
	Receiving Events in C/SIDE . . . . .	322
	Using Custom Controls from C/SIDE . . . . .	326
	Acquiring Controls . . . . .	333
<b>PART 6</b>	<b>DATAPORTS . . . . .</b>	<b>335</b>
	<b>Chapter 17 Dataports . . . . .</b>	<b>337</b>
	What Are Dataports? . . . . .	338
	Designing Dataports . . . . .	344
	Exporting Data . . . . .	350
	Importing Data . . . . .	357
<b>PART 7</b>	<b>MULTILANGUAGE FUNCTIONALITY . . . . .</b>	<b>367</b>
	<b>Chapter 18 Multilanguage Functionality . . . . .</b>	<b>369</b>
	Multilanguage Functionality . . . . .	370
	Developing Multilanguage-Enabled Applications . . . . .	376
	Learning the Code Base Language . . . . .	380
	Number Ranges for Text Constants . . . . .	383
<b>PART 8</b>	<b>BEYOND THE BASICS . . . . .</b>	<b>387</b>
	<b>Chapter 19 Type Conversion . . . . .</b>	<b>389</b>
	Type Conversion in Expressions . . . . .	390
	Type Conversion Mechanisms . . . . .	392
	<b>Chapter 20 SumIndexFields . . . . .</b>	<b>401</b>
	SumIndexFields . . . . .	402
	SIFT and the SQL Server Option for Navision . . . . .	404
	<b>Chapter 21 Numbering in Navision . . . . .</b>	<b>423</b>
	How Does Number Sorting Work? . . . . .	424

<b>Chapter 22</b>	<b>C/SIDE in Multiuser Environments</b>	<b>427</b>
	Ensuring Data Integrity in a Multiuser Environment	428
	Locking in Navision – a Comparison of the two Server Options	436
<b>Chapter 23</b>	<b>Caption Class Functionality</b>	<b>441</b>
	Syntax	442
	Function Code	448
<b>Chapter 24</b>	<b>Supporting Record Level Security</b>	<b>455</b>
	Record Level Security	456
<b>Chapter 25</b>	<b>Performance</b>	<b>457</b>
	The DBMS Cache	458
	The Commit Cache	460
	The Command Buffer	462
	Keys, Queries and Performance	464
	C/AL Database Functions and Performance on SQL Server	466
	Configuration Parameters	467
	Bulk Inserts	471
<b>PART 9</b>	<b>APPENDICES</b>	<b>473</b>
<b>Appendix A</b>	<b>C/SIDE Specifications</b>	<b>475</b>
	Specifications for the DBMS	476
	Specifications for C/SIDE Application Objects	477
<b>Appendix B</b>	<b>Report Flow Charts</b>	<b>479</b>
	Report Flow Charts	480
	Report.Run	481
	Dataltem.Run	482
	Section.Run	483
	Header.Run	484
	Footer.Run	485
	TransHeader.Run	486
	TransFooter.Run	487
	GroupHeader.Run	488
	GroupFooter.Run	489
	Body.Run	490
	NewPage	491
	GetRecord	492
<b>Appendix C</b>	<b>Dataport Flow Charts</b>	<b>493</b>
	Dataport Flow charts	494
	Dataport.Import/Export	495
	Dataltem.Export	496
	VariableRecord.Export	497
	FixedRecord.Export	498

DatalItem.Import. . . . . 499

VariableRecord.Import. . . . . 500

FixedRecord.Import. . . . . 501

**Appendix D   NDBCS – The Database Driver . . . . . 503**

NDBCS – the Database Driver . . . . . 504

A Brief History of Performance Improvements . . . . . 512







## Chapter 1

### C/SIDE Fundamentals

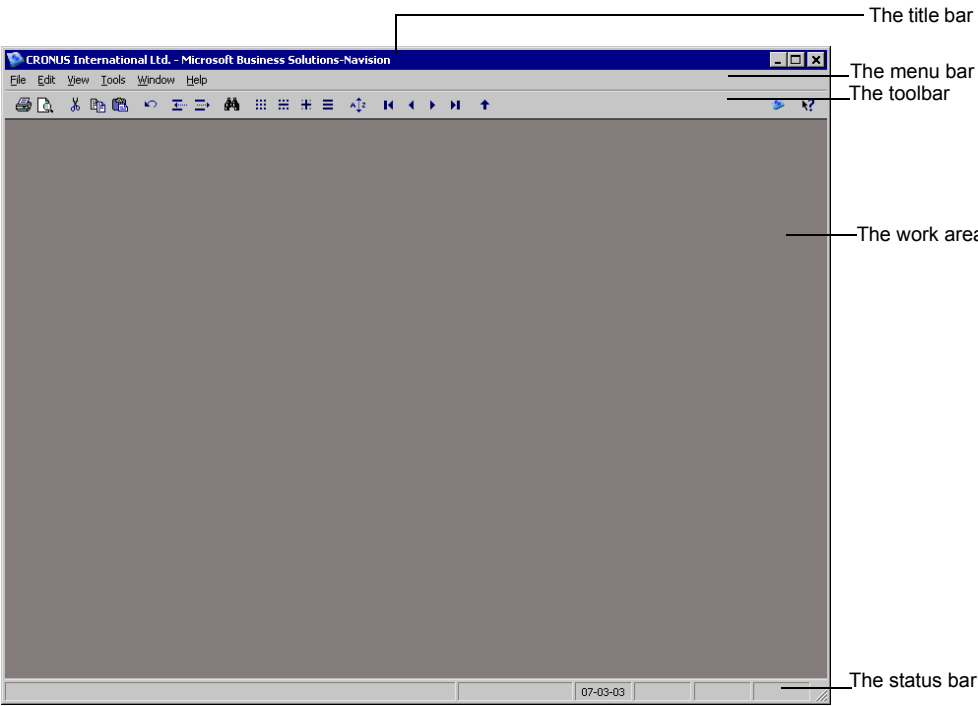
A C/SIDE® application is composed from five types of application objects. Each type of application object is created using a specific tool called a designer. The application objects you create using these designers are all based on some general concepts. A fundamental knowledge of these concepts speeds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

## 1.1 THE C/SIDE USER INTERFACE

This section introduces you to the user interface in C/SIDE. If you have not already installed C/SIDE, refer to the installation manual. If you have already installed C/SIDE, the installation program has created a new group that contains all the icons you need to work with C/SIDE. When the Integrated Development Environment (IDE) is running, your screen will look like this:



The user interface gives you access to a number of tools and functions. Some parts of the user interface also provide information about the current state of the system. The table below explains when to use the most important parts of the C/SIDE user interface.

To...	Use the...
get information about the name and path of the current database	title bar
access functions on drop-down menus	menu bar
access the most commonly used functions quickly	toolbar <sup>(A)</sup>
work with the application design tools	work area <sup>(B)</sup>
see basic status information about your system (such as the current date and your user ID)	status bar

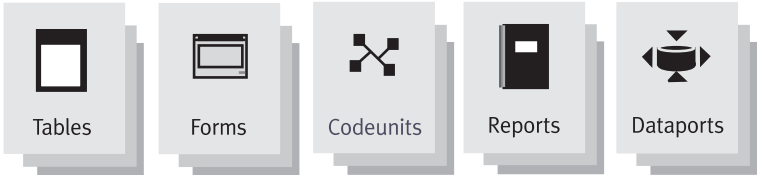
(A) DEPENDING ON THE TASK YOU ARE WORKING ON, THE SYSTEM AUTOMATICALLY CHANGES THE ICONS.

(B) THIS IS ALSO WHERE THE USER INTERACTS WITH YOUR APPLICATIONS.



Designing Application Objects

Any application designed in C/SIDE is based on five different types of application objects:



Tables are the fundamental objects that store the actual data

Tables are the fundamental objects that store the actual data; you need other application objects to insert, modify, delete or show data from tables. You will typically use a form to enter or retrieve data from the database and use a report to print data.

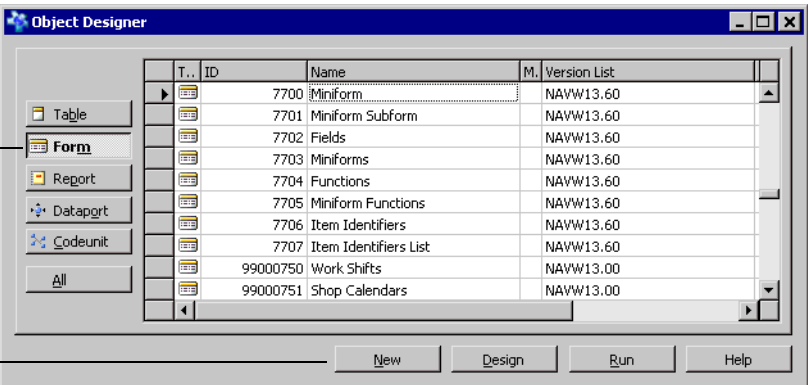
Note

All application objects are 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 NTR for more information.

The main tool used for developing applications in C/SIDE is the Object Designer (choose Tools, Object Designer). This is the tool you use to view and design tables, forms, reports, dataports and codeunits.

In the Object Designer you choose the type of application object you want to work on. From the Object Designer you can run an application object or start an application object designer to modify the design of an existing application object or create a new application object. The following picture shows how to use the Object Designer in more detail.

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 Object  
Designers

The table below lists the tools you can access via the Object Designer and when you should use them.

Use the...	When working on ...
Table Designer	tables
Form Designer	forms
Report Designer	reports
Dataport Designer	dataports
C/AL™ editor	codeunits

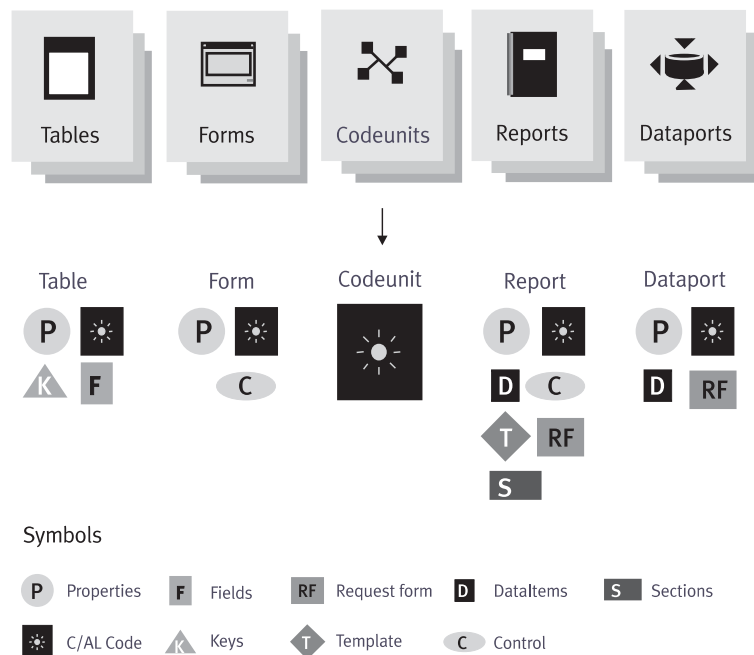
As you can see, there is a specific designer for each type of application object. When you create or modify an application, you can work on any number of application objects at the same time, and each application object is shown in its own designer. For example, if you work on three new forms at the same time, then each form will be displayed in its own form designer. The only designer that you cannot create more than one copy of is the Object Designer. You will learn more about how to use each of these designers as you read the following parts of this book.

## 1.2 WHAT IS A C/SIDE APPLICATION?

The C/SIDE Integrated Development Environment (IDE) is specially designed for creating accounting and business management applications. Any C/SIDE application consists of the same objects as a C/SIDE database. The difference between the term *database* and the term *application* is that when we speak about a database, we mean simply a collection of application objects, whereas when we speak about an application, we mean a set of application objects tied together to form a coherent whole.

### General C/SIDE Concepts

You have already learned that there are five different types of application objects in C/SIDE. All five types are based on some general concepts. Some of these concepts are restricted to one type of application object while others apply to several types. When you understand these fundamental concepts, you have a good foundation for creating your own applications. The following figure illustrates how the application objects are related to these general concepts.



The table below summarizes the information in the figure, and explains what each type of application object is used for.

Application Object Type	What is it used for?	Which concepts is it based on?
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
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

The terms in the third column have the following descriptions:

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

**C/AL** C/AL is the language used for writing functions in C/SIDE. In the table above, "C/AL" refers to functions written in this language.

**Triggers** When specific things happen to the application objects, the system automatically activates a trigger. Inside a trigger you can add your own C/AL code if you want to modify the default behavior of the application object or extend its functionality.

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

**Fields** A field is the smallest unit of information 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 decorate the form. Typical examples 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 you use for defining a model of your data when you create a report. By using a hierarchy of data items you define which data your report should include. 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. A section is where you place controls to display information. You will typically use sections defining the body, header, and footer in your report.

## 1.3 THE PHYSICAL AND THE LOGICAL DATABASE

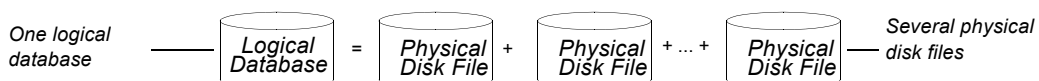
The previous section described the general concepts underlying all five types of application objects in C/SIDE. This section presents another view of C/SIDE applications. In this view we are concerned only with how the information in your application is structured.

As a typical database user, you are not concerned with where each piece of data is stored on the hard disk or what its size 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 you with 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 relations) which are 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 bits of information. That is, we do not deal with how these structures and relations are implemented. When we speak about the physical database we deal only with how the structures in the logical database and the search paths between them are implemented.

In this book the term database should be interpreted to mean the logical database unless otherwise noted.

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 figure below 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:

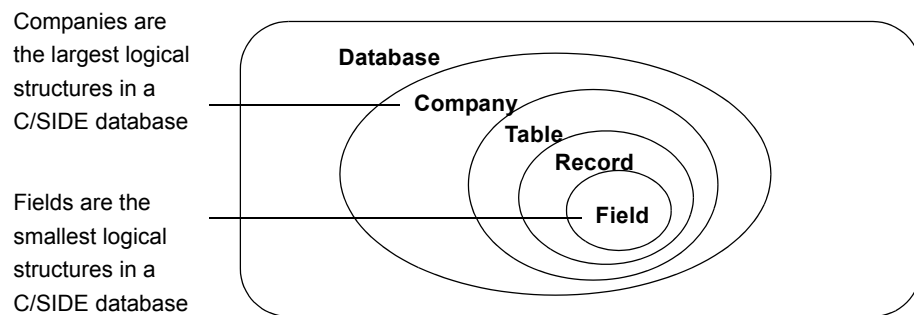
**Fields** A field is the smallest logical structure used in the C/SIDE database. A field is used to hold a single bit of information, such as a name, "Joe," or an amount, "2,352.00." Any particular field can hold information of only one specific type. (The C/SIDE database system distinguishes between 10 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. By assembling these small bits of information into records we get a much more flexible "information-holder" that is also better organized because it keeps together fields that belong together.

**Records** A record is a logical structure assembled from an arbitrary number of fields. It is used to store a single entry in the database. The fields in a record are used to

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 used in a C/SIDE database. A company may be considered as a subdatabase; its primary use is to separate and group large portions of data in a database. A company can contain private tables as well as tables shared with other companies.







## Chapter 2

### 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

## 2.1 INTRODUCTION TO C/SIDE APPLICATION DESIGN

In this section we will briefly outline the procedures involved in designing a C/SIDE database application. It usually includes the following 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 that you use to determine how the data will be 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 in the tables.
- 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 completed the design of the database tables, you are ready to begin designing the application itself. This involves:

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

The above steps depend on each other. When you go from one step to another you will 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 C/SIDE, you have to determine the purpose of the database and how it will be used. The easiest way to do this is to talk to the people who will use it. Involving the end user as early as possible eliminates problems that can stem from misunderstandings about the purpose of the database. Interviewing the end users will help you get a better understanding of the tasks they expect the system to be able to solve. Based on this, you can determine the data (tables) necessary for completing these tasks. This will often be the most difficult part of the design process and also the most important, as the usefulness of the entire application depends on whether the tables have been designed correctly.

Your interviews of the end users will give you a good knowledge of which questions the end users want answered and thus of the information that forms and reports should provide. This does not necessarily tell you how you should structure your tables, however.

## Designing the Tables

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

You begin by defining a data model. This model should describe:

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

A model suitable for this purpose is the ER model (Entity-Relationship model). An ER model is capable of mapping real-world situations to a relational database system such as C/SIDE.

Basically, an ER model divides all the elements of a real world situation into two categories: *entities* and *relations*. 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 will 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 entity type 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 subsections are not intended to serve as a description of all facets and implications of the ER model but are rather intended to give you an overview of the model and at the same time show you the benefits of applying a formalized design method.

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

A real world problem will usually contain groups of entity types that are similar. For example, consider a company having 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 that have the same properties. When you implement the abstract ER model in C/SIDE, you will 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, your next step is to 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

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

<b>Customer Table</b>	<b>Company Name</b>	<b>Contact Person</b>	<b>Phone</b>	<b>...</b>	<b>Payment Method</b>

Your analysis has shown that you need fields such as Company Name, Contact Person, Phone and Payment Method. When you implement the Customer table, you select the following field types:

Field Name	Description	Field Type
<b>Company Name</b>	This field is used to store the name of the customer (for example, "Microsoft Business Solutions ApS").	string
<b>Contact Person</b>	This field identifies the contact person in the company (for example, "JLJ").	string
<b>Phone</b>	This field contains the customer's phone number (for example, "45662111").	string
<b>Payment Method</b>	This field describes the payment method for the customer (for example, "pay in cash").	option

Refer to Choosing Data Types on page 23 for a description of the C/SIDE field types.

### Role of Keys in C/SIDE

The ER model places a very important constraint on the entities in an 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 whose values are distinct for each

individual entity. The table below 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

For C/SIDE to be able to operate efficiently on the data in tables, the records must be arranged according to some criterion (that is, a key). 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 above must hold for every record in the table. This constraint prevents any two records from having the same value for the key field. It is not a constraint on a specific record but a constraint for all records in the table, considered together.

Sometimes a key consists of several fields together; in this case the combination of the field values must be distinct for each record.

Sometimes you will be able to define several keys for a table. Refer to the section How to Define a Primary Key on page 26, which discusses the concepts of keys.

### Determining the Relationships

At this point in the design process, you have carefully planned a number of tables to store individual types of information. In your final application you want to be able to retrieve the information in a meaningful way. Very often an answer from your database will consist of information stored in several tables. To allow for such answers, C/SIDE uses relationships to chain related information together. In database terminology it is common to distinguish between three types of relationships:

**One-to-Many Relationships** In this type of relationship, 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** In this type of relationship, 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** In this type of relationship, 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 simply be avoided by combining the two tables.

## Assuring the Quality of the Design

In the process of defining the tables and setting up relationships, you will often have to select from among several possible solutions. To make sure that you select the most appropriate solutions, you need a way to measure design quality.

This is done using what is known as the *normalization process*. 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 can teach you how to obtain these normal forms. Some good starting points are the books mentioned on the last page of this chapter.

## Designing the Application

After you have completed your table design, you are ready to begin designing the application itself. From the analysis phase, you have an overview of which answers the application is expected to be able to provide. From the table design phase, you have a clear description 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. Reports allow more flexibility than forms do when you want to present summary information.

**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.

At this point you will have a useful application. If you took the time to plan all steps of the application design carefully, you also have an application that is fully documented. This will prove to be a great help when you need to make future adjustments and additions.

## 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.







## Chapter 3

### 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?
- What Are Keys?
- Identifiers, Data Types and Data Formats in the SQL Server Option for Navision
- Saving, Viewing, and Sorting Data
- Dividing the Database into Companies
- Special Table Fields

### 3.1 WHAT IS A TABLE?

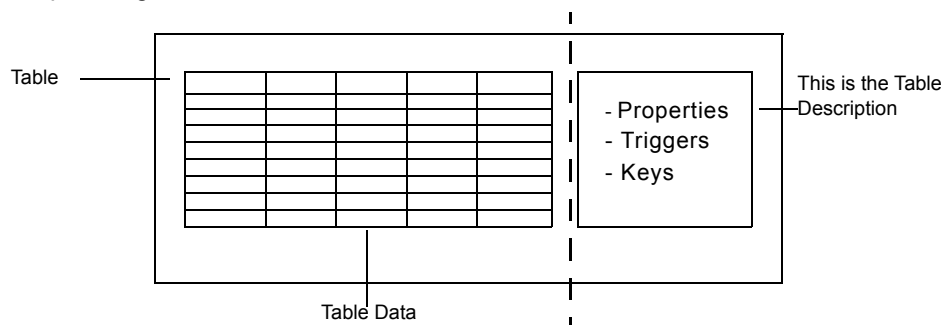
The records in the C/SIDE database are stored in tables. A C/SIDE table may be visualized as a two-dimensional matrix, consisting of columns and rows. The figure below shows a table with nine rows and eight columns. Each row is a record, and each column is a field.

Rows: Records

Columns: Fields

Entry No.	G/L Account	Date	Doc Type	Document No.	Description	Bal. Account	Amount
1	1110	C12/31/93	START		Opening Entry		11,344,104
2	1140	C12/31/93	START		Opening Entry		-2,915,721
3	1210	C12/31/93	START		Opening Entry		4,990,344
4	1240	C12/31/93	START		Opening Entry		-3,101,574
5	1110	C12/31/93	START		Opening Entry		1,322,682
6	1140	C12/31/93	START		Opening Entry		-538,741
7	1310	C12/31/93	START		Opening Entry		423,578
8	1340	C12/31/93	START		Opening Entry		-212,361
9	2120	C12/31/93	START		Opening Entry		2,311,205

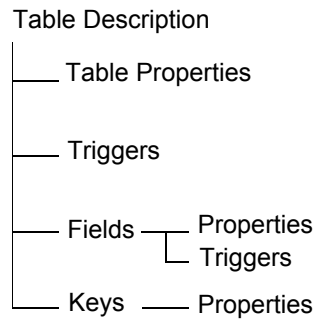
A table consists of two parts: the table data and a table description. The table data is the part users often think of as comprising the database, because it contains the actual records with their data fields. The layout and properties of those fields, however, are specified by the table description. The table description is not directly visible to the user. The next figure illustrates how the table data and the table description together form a table.



When you design a table, you assign it a number of characteristics, such as a name, an ID number and the fields it contains. You also assign a number of characteristics (such as name, ID number, data type and initial value) to each field. When you design a new table, you also specify which keys you want the system to maintain. All these characteristics are stored in the table description when you save your table design.

The information in the table description is used by the Database Management System (DBMS) and occasionally by database users who need information about the table structure. The table description makes the DBMS flexible, as it lets the system access tables with different structures. The DBMS can extract the definitions of the table structure from the table description and thereby correctly access any table.

The 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, while others are related to the fields in the table. Still other properties are related to the keys in the table. You can also see from the figure that triggers are defined both for the table and for the fields in the table.

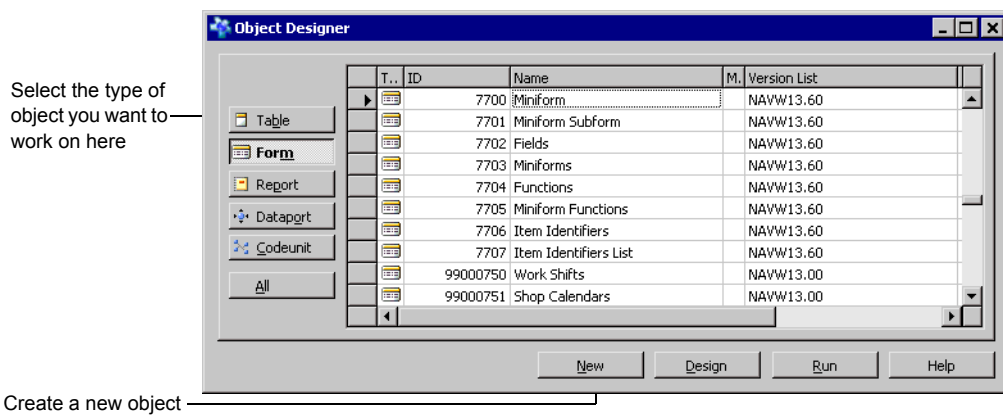
Don't worry if you are not familiar with these terms already. You'll learn more about them on the following pages, and the next chapter, Customizing and Maintaining Tables on page 49, provides a more detailed description of how to customize your tables by modifying the properties and creating triggers.

## Creating a Table

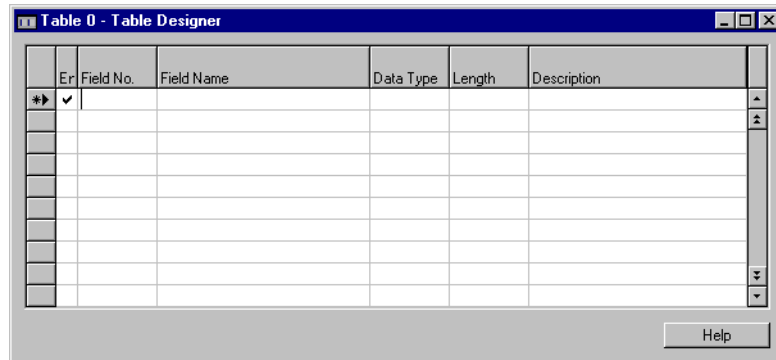
When you first create a table, it will not contain any data. When you create the table you have to decide which types of information you want to store in it.

To create a table:

- 1 Click Tools, Object Designer. C/SIDE will display:



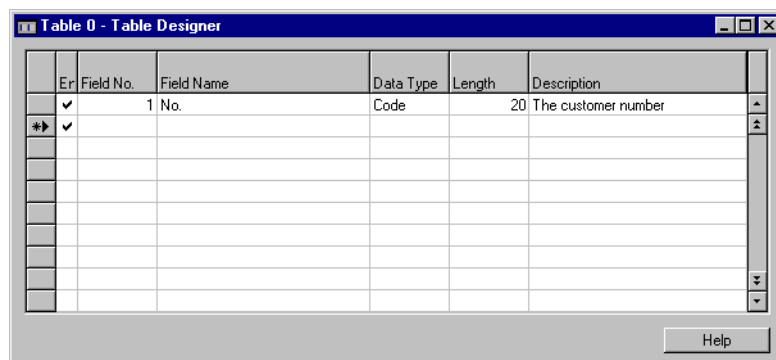
2 Click Table, New and the Table Designer appears:



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. The following subsections describe how to do this.

### Adding Fields to Your Table

Designing a field means assigning it a number of characteristics. These characteristics depend on what you intend the field to be used 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 will appear 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 the C/SIDE IDE or C/FRONT, this number uniquely identifies all tables and fields.
- A name (an alphanumeric string) serving 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 17 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	<p>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:</p> <p>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".</p> <p>The size of the corresponding SQL data type, <code>INTEGER</code>, is 4 bytes.<sup>(A)(B)</sup></p>	4 bytes
Integer	<p>Denotes an integer between -2,147,483,647 and 2,147,483,647.</p> <p>The size of the corresponding SQL data type, <code>INTEGER</code>, is 4 bytes.<sup>(A)(B)</sup></p>	4 bytes
Decimal	<p>A decimal number between <math>-10^{63}</math> and <math>10^{63}</math>. 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).</p> <p>The size of the corresponding SQL data type, <code>DECIMAL(38,20)</code>, is 17 bytes.<sup>(A)(B)</sup></p>	12 bytes
Text	<p>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.</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 + 1 byte (see note below).

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 Microsoft SQL Server Option for Navision, code fields work in a different way. You can use the SQL Data Type property to indicate whether code fields can contain integers or text strings. Refer to the online C/SIDE Reference Guide for information about the SQL Data Type property. Further, Appendix H 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 below).
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 below).
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.	
BigInteger	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 Navision. 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 NAVISION USES WHEN IT CREATES THE NAVISION DATA TYPE. FOR FURTHER INFORMATION, SEE PAGE 33.

### Note

.....  
In Navision 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 42.

## 3.2 WHAT ARE KEYS?

The DBMS keeps track of each field by means of the field number, described above, 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, the records are stored sequentially in ascending order, sorted according to the primary key. Before adding a new record to a table, the DBMS checks that the information in primary key fields in the record is unique, and only then inserts the record into its correct logical position. Because the records are sorted "on the fly," the database will always be 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; 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.

Some other database systems support unkeyed tables. An unkeyed table is one for which no key fields have been designated; in such a table, records are stored in the order in which they were entered in the table. The C/SIDE database system does not support unkeyed tables.

### How to Define a Primary Key

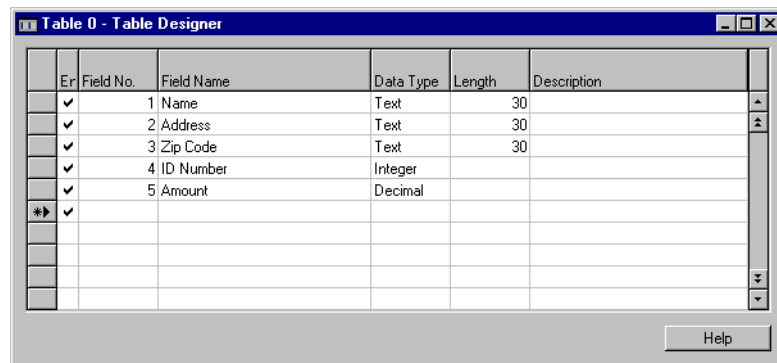
A maximum of 20 distinct fields can be used in the definition of the primary key. The number of fields in the primary key puts a limitation on the number of fields in the other (secondary) keys.

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



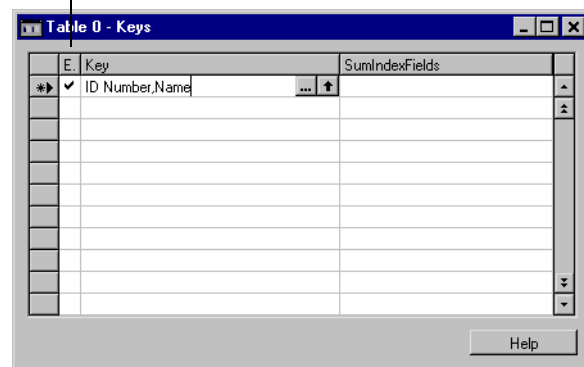
To define a primary key:

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



- 2** Choose Keys from the View menu to define a primary key. C/SIDE will display the following:

Define the  
primary key here



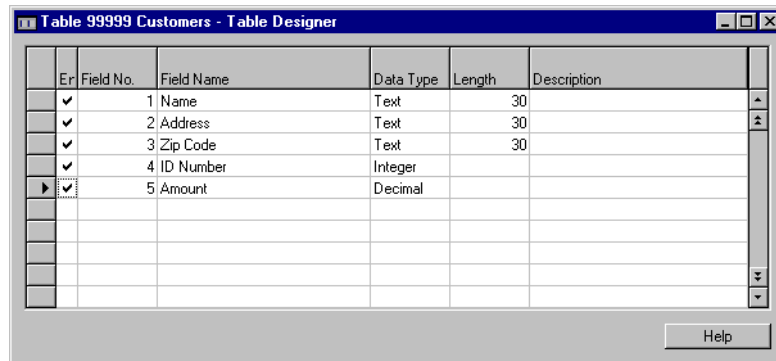
- 3** On the first line in the Key window, enter the primary key as a comma-separated list (for example: ID Number,Name).

## How to Create Secondary Keys

We have already mentioned that up to 40 keys can be associated to a table and that the first is the primary key. All other keys are secondary keys and optional. Secondary keys are used to view records in an order different from the one in which they are sorted according to the primary key fields.

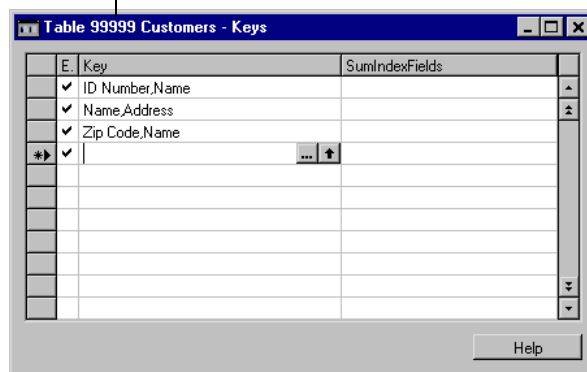
To create a secondary key:

- 1 Open your table in the Table Designer:



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

Enter the  
secondary keys  
here as comma-  
separated lists



- 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, then your secondary keys can include these four fields, and at most 16 others. 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 data structure called an index. The idea behind an index is similar to the idea behind the indexes used in common textbooks. A textbook index lists important terms at the end of the book in alphabetical order. Next to each term is a list of page numbers where the term appears. We can search the index to

find a list of page numbers (addresses) and easily locate the term in the textbook by searching the specified pages. Hence, the index is an exact indicator of 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. This means that the DBMS does not use time during updates to maintain its index. Furthermore, an inactive key doesn't occupy database space. Inactive keys can be reactivated; this process may consume some time, depending on the size of the table, 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; 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 solve this conflict. The example below shows how the primary key influences the sorting order when a secondary key has been activated:

We 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 will be based on the contents of the **Customer Name** field. As the contents of these fields are not unique, the records will have to be subsorted according to the primary key.

Customer Name	Customer No.
IBM	002
Lotus	003

Customer Name	Customer No.
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 the Working Speed of C/SIDE

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 flexible searches to be performed quickly. 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.
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 a few or many keys is not easy to discuss in general. 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 used only rarely.

The overall speed of C/SIDE will depend strongly upon 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 hardware, that is, the speed of your computer and its disk system.

### How Are the Keys Stored?

As illustrated in the figure on page 20, keys are stored in the Table Description, which contains a list of keys. The next figure illustrates a part of the key list for a **Cust. Ledger Entry** table.

Key Description	1 (Entry No.)				Primary key
	3 (Customer No.)	4 (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 illustrates 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.

### 3.3 IDENTIFIERS, DATA TYPES AND DATA FORMATS IN THE SQL SERVER OPTION FOR NAVISION

#### 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 Navision 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 Navision 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 Ltd\_\$Customer\$0*. Any secondary keys in a Navision 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 Navision table with keys that contain *SumIndexFields*<sup>®</sup>, this causes additional tables to be created in SQL Server to support the SIFT<sup>™</sup> 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 Navision 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 Navision Data Types

Every available Navision data type is mapped to an appropriate SQL Server data type in the tables of the SQL Server Option for Navision. The following table shows which SQL Server data type is used for the corresponding Navision data type:

Navision 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 Navision data types. The following table shows the extended compatibility of SQL Server data types with Navision data types:

SQL Server Data Type	Navision Data Type
CHAR ( n )	Code(n) Text(n) DateFormula

SQL Server Data Type	Navision Data Type
NCHAR (n)	Text(n)
NVARCHAR (n)	Text(n)
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 Navision, you must be aware of the effect the data formats will have on the way your data is compared and sorted.

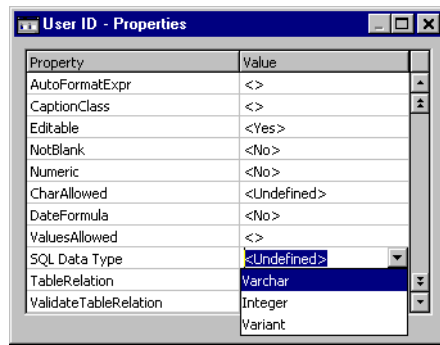
### Code Fields

In the SQL Server Option for Navision, 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 Navision, 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 Navision using external tools, they cannot be read into Navision.
- The value "0"(zero) is used to represent an empty string in Navision.
- 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 Navision 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, Navision 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 Navision 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. Navision uses only the time part and places a constant value for the date: 01-01-1754. The Navision undefined time is represented by the same value as an undefined date.

In order for Navision to interpret date and time values correctly, the formats mentioned above must be used when linking Navision table definitions to external tables or views. For more information about this, see page 66.

To reformat a DATETIME or SMALLDATETIME column that is to be used as a date field in Navision, 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 Navision Tables with External Tools

You can access data in Navision 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 Navision. 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).

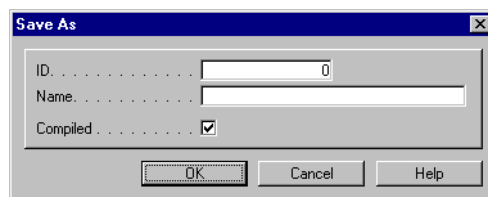
The view is updated by every change in the CaptionML values of a table. For more information about multilanguage, see Multilanguage Functionality on page 369.

### 3.4 SAVING, VIEWING, AND SORTING DATA

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

To save a table in the database:

- 1 Make sure that focus is on the Table Designer. Click File, Save. C/SIDE displays the following window:



- 2 Enter a number to serve as a unique identification of the table in the ID field. Notice that there are restrictions about which numbers you can use. Please contact your NTR for information.

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

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

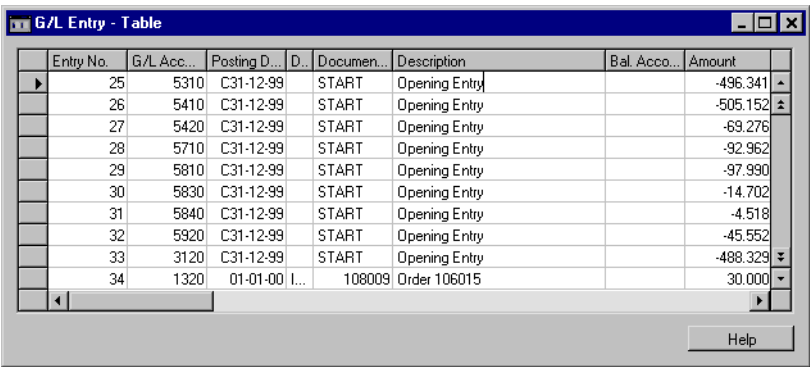
- 1 Open the Object Designer, select the table you want to view and click the Run button. C/SIDE displays the data in a tabular format:

Entry No.	G/L Acc...	Posting D...	D.	Documen...	Description	Bal. Acco...	Amount
25	5310	C31-12-99		START	Opening Entry		-496.341
26	5410	C31-12-99		START	Opening Entry		-505.152
27	5420	C31-12-99		START	Opening Entry		-69.276
28	5710	C31-12-99		START	Opening Entry		-92.962
29	5810	C31-12-99		START	Opening Entry		-97.990
30	5830	C31-12-99		START	Opening Entry		-14.702
31	5840	C31-12-99		START	Opening Entry		-4.518
32	5920	C31-12-99		START	Opening Entry		-45.552
33	3120	C31-12-99		START	Opening Entry		-488.329
34	1320	01-01-00	I...	108009	Order 106015		30.000

The order in which the information appears in the window above reflects 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 these keys define.

To view the table data in different sorting orders:

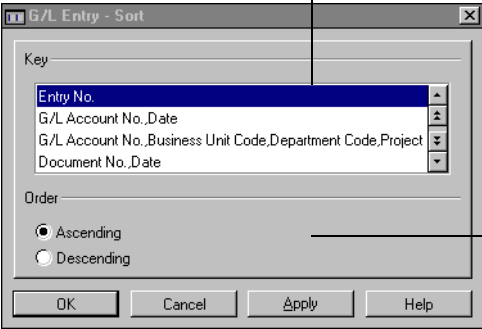
- 1 Run the table from the Object Designer. C/SIDE will display the following:



Entry No.	G/L Acc...	Posting D...	D...	Documen...	Description	Bal. Acco...	Amount
25	5310	C31-12-99	START		Opening Entry		-496.341
26	5410	C31-12-99	START		Opening Entry		-505.152
27	5420	C31-12-99	START		Opening Entry		-69.276
28	5710	C31-12-99	START		Opening Entry		-92.962
29	5810	C31-12-99	START		Opening Entry		-97.990
30	5830	C31-12-99	START		Opening Entry		-14.702
31	5840	C31-12-99	START		Opening Entry		-4.518
32	5920	C31-12-99	START		Opening Entry		-45.552
33	3120	C31-12-99	START		Opening Entry		-488.329
34	1320	01-01-00	I...	108009	Order 106015		30.000

- 2 Click View, Sort and C/SIDE displays:

Select the key here



Select ascending or descending order here

The 'G/L Entry - Sort' dialog box contains a 'Key' list with the following items: 'Entry No.', 'G/L Account No.,Date', 'G/L Account No.,Business Unit Code,Department Code,Project', and 'Document No.,Date'. The 'Entry No.' item is selected. Below the list is an 'Order' section with two radio buttons: 'Ascending' (which is selected) and 'Descending'. At the bottom are four buttons: 'OK', 'Cancel', 'Apply', and 'Help'.

- 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 the OK or the Apply button 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.

Data is normally entered in a table by using a form, but it is also possible to enter it directly in the table.

To add records to a table without using a form:

- 1 Open the Object Designer and select the table you want to add records to.

2 Click Run. C/SIDE will display the table in a tabular format:

Entry No.	G/L Acc...	Posting D...	D.	Documen...	Description	Bal. Acco...	Amount
2703	5530	25-01-01		2607	Packing Tape		4
2704	2910	25-01-01		2607	Packing Tape	8910	-24
2705	2910	25-01-01		2608	Repair and Upgrade of Spray-p...		-277
2706	1120	25-01-01		2608	Repair and Upgrade of Spray-p...		147
2707	5530	25-01-01		2608	Repair and Upgrade of Spray-p...		36
2708	8910	25-01-01		2608	Repair and Upgrade of Spray-p...		73
2709	5530	25-01-01		2608	Repair and Upgrade of Spray-p...		18
2710	6120	25-01-01	C...	2810	Auto-Günther KG	49633663	322
2711	2320	25-01-01	C...	2810	Auto-Günther KG	6120	-322
							0

3 Place the cursor in a blank 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 the fields.

4 When you have finished entering data, close the table. (You do not have to save it, as the records are saved and updated whenever you leave a field after entering a value in it.)

### 3.5 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. As mentioned on page 22, fields and tables are identified by a number. Companies are identified not by a number but by a name. A company "bundles" one or more data tables together into a logical superstructure that is identified by a 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 manual *Installation and System Management: Microsoft Business Solutions—Navision Database Server*.

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.

The idea of a company can be explained by an analogy with records in C/AL: When you work with records in C/AL, you can use a WITH statement in order to tell the system that whenever you refer to a field, you mean that field within a specific record. That is, within the scope of the WITH statement, you do not explicitly need to refer to <record>.<field> but just to <field>, as <record> is assumed as the default. Likewise, by opening a company you specify a default group of tables to which all your database accesses will be directed.

Even though you have selected a specific company, you can still access data in any table in any other company. To do so, you must 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 on page 432.

### 3.6 SPECIAL TABLE FIELDS

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

- FlowFields
- FlowFilter fields

#### What Are FlowFields?

FlowFields are a powerful feature of the C/SIDE database system. The FlowField is a fundamental concept that strongly influences the way a C/SIDE application is designed.

FlowFields and the underlying concept of SumIndexFields have been designed in order to increase the performance in such activities as calculating the balance of your customers, which in traditional database systems 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 and the chapter SumIndexFields on page 401, which deals with the underlying concept of SumIndexFields.

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. Actually the information in the FlowFields exists only at run time. The values in FlowFields are automatically initialized to 0 (zero). To update a FlowField, you must use the C/AL function <Record>.CALCFIELDS. Notice that 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.

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 figure below. 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 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

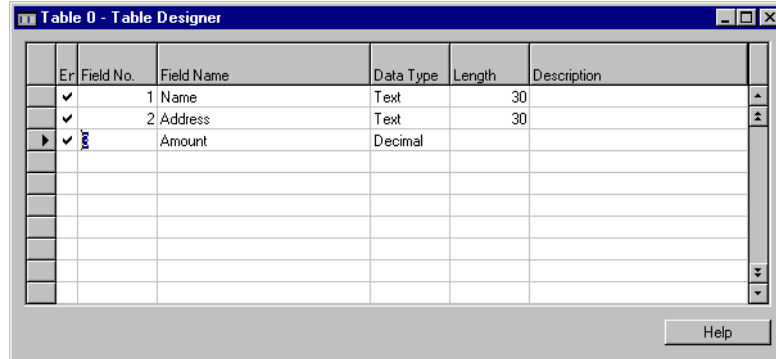
```
Sum("Customer Entries".Amount WHERE(CustNo=FIELD(CustNo)))
```

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

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

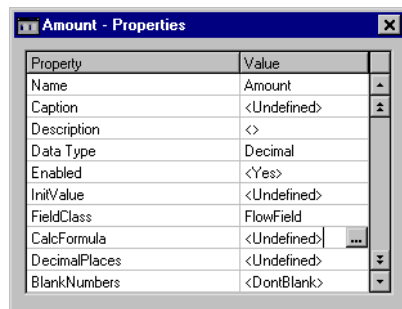
To create a FlowField:

- 1 Design the table in the Table Designer. C/SIDE will typically display:



- 2 Click on the line defining the field that you want to turn into a FlowField.

- 3 Click View, Properties. C/SIDE displays the property sheet:



- 4 Change the value of the *FieldClass* property from *Normal* to FlowField
- 5 Now you have to 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 value in the FlowField is calculated. Below is a description of the valid syntax for the *CalcFormula* property:

```
<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>)))

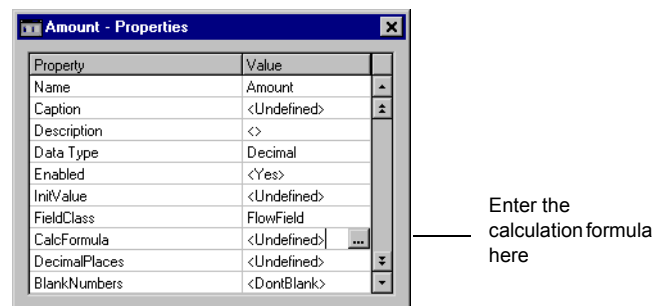
```

where...

Symbol	Explanation
<TableNo>	Specifies the table holding the information to be used in the FlowField.
<FieldNo>	Specifies the column from which you want to compute values.
<TableFilters>	A list of filters to be used in the computation of the FlowField.
<TableFilter>	A table filter can be one of the following: a constant expression, a filter expression, a value from ordinary fields or a FlowFilter field (FlowFilter 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.

To create, view, or edit a calculation formula:

- 1 Click the field for which you want to create, view, or edit the calculation formula.
- 2 Click View, Properties. Find the *CalcFormula* property in the property sheet:



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

The 'Calculation Formula' dialog box has a title bar with a close button. It contains several fields: 'Method' with a dropdown menu showing 'Sum'; 'Reverse Sign' with an unchecked checkbox; 'Table' with a dropdown menu showing 'G/L Entry' and an assist-edit button; 'Field' with a dropdown menu showing 'Amount' and an assist-edit button; and 'Table Filter' with a text field containing 'G/L Account No.=FIELD(No.),G/L Acc...'. At the bottom are 'OK', 'Cancel', and 'Help' buttons.

- 4 Click the drop-down button to select the appropriate calculation method. Click the Reverse Sign option if you want 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 you want 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 Filter field. C/SIDE displays the **Table Filter** window:

The 'Table Filter' window has a title bar with standard window controls. It contains a table with the following data:

Field	Type	Value	OnlyMax...	ValuesFil...
G/L Account No.	FIELD	No.		
G/L Account No.	FIELD	Totaling		✓
Business Unit Code	FIELD	Business Unit Filter		
Department Code	FIELD	Department Filter		
Project Code	FIELD	Project Filter		
Posting Date	FIELD	Date Filter		
**	CONST			

At the bottom are 'OK', 'Cancel', and 'Help' buttons.

Each line defines a field filter. Notice that there are implicit logical ANDs between the lines.

- 5 At each line in this window, you can define a field filter. For each field filter you must specify a field, a type, and a value and you can set the OnlyMaxLimit and the ValuesFilter 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 that contains the information to be summed. This field is defined as a **SumIndexField** in the key for the **G/L Entry** Table.

Entry No.	G/L Account	Date	Doc. Typ.	Document No.	Description	Bal. Account	Amount
1	1110	12/31/93	START		Opening Entry		11,344,104
2	1140	12/31/93	START		Opening Entry		-2,915,721
3	1210	12/31/93	START		Opening Entry		4,990,344
4	1240	12/31/93	START		Opening Entry		-3,101,574
5	1110	12/31/93	START		Opening Entry		1,322,682
6	1140	12/31/93	START		Opening Entry		-538,741
7	1310	12/31/93	START		Opening Entry		423,578
8	1340	12/31/93	START		Opening Entry		-212,361
9	2120	12/31/93	START		Opening Entry		2,311,205

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

Field	Type	Value	OnlyMaxLim	ValuesFilter
G/L Account No.	FIELD	No.		
G/L Account No.	FIELD	Totaling		✓
Business Unit Code	FIELD	Business Unit Filter		
Department Code	FIELD	Department Filter		
Project Code	FIELD	Project Filter		
Date	FIELD	Date Filter	✓	

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

No.	Name	Search Name	Account Type	Balance at Date	Project Code	Department Code	Inc.
5550	Electricity Tax	ELECTRICI	Posting	0.00			Bal
5560	Natural Gas Tax	NATURAL (	Posting	0.00			Bal
5570	Coal Tax	COAL TAX	Posting	0.00			Bal
5580	CO2 Tax	CO2 TAX	Posting	0.00			Bal
5590	Water Tax	WATER TA	Posting	0.00			Bal
5610	VAT Payable	VAT PAYAE	Posting	0.00			Bal
5690	VAT, Total	VAT, TOT	End-Tot	-2,407,590.15			Bal
5800	Personnel-related Items	PERSONNE	Begin-T	0.00			Bal
5810	Withholding Taxes Payable	WITHHOLC	Posting	-726,666.00			Bal

The **Balance at Date** FlowField

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 will be able to enter filter values in a FlowFilter form:

Field	Filter
Date Filter	
Department Filter	
Project Filter	
Budget Filter	
Business Unit Filter	

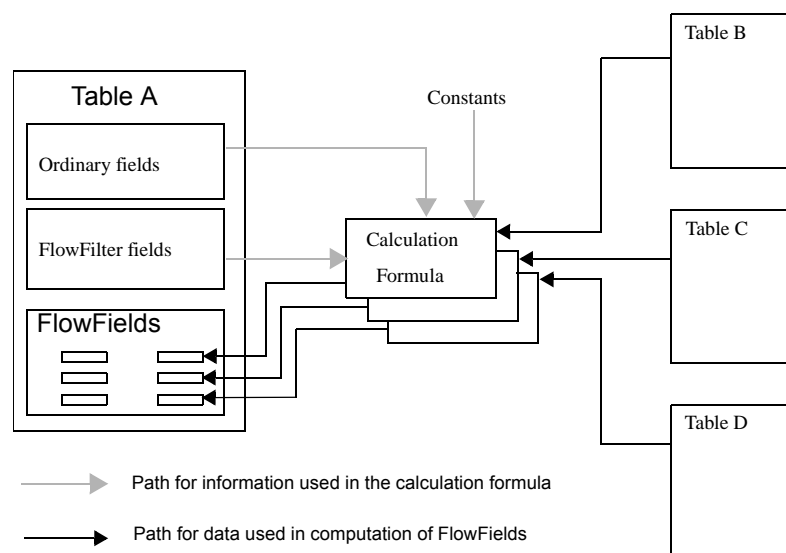
This means that if the user enters a date filter expression in the **Date Filter** field, it will be transferred via the table filter and used in the **Date** column in the **G/L Entry** table.

You can use the OnlyMaxLimit option to remove the lower bound 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 will transform the expression into ..y.

The ValuesFilter 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 ValuesFilter option will cause this value to be interpreted as a filter rather than a specific value.

### Using FlowFilter Fields in the Calculation Formula

End 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 up 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 above 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, of values from ordinary fields and of filters given as parameters in FlowFilter fields. FlowFilter fields are fields in which the end user can enter a filter value (via the user interface in a C/SIDE application) that will affect the calculation of a FlowField.

## Chapter 4

### 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.

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

## 4.1 VIEWING AND MODIFYING PROPERTIES

This section describes how you can use properties in your table design. As you have learned in the previous section the properties in a C/SIDE table can be divided into these categories

- Table Properties
- Field Properties
- Key Properties

### Viewing and Modifying Table Properties

A table in C/SIDE has a number of properties that describe the behavior of the table in your environment. When you create a table, C/SIDE automatically defines a number of default values for these properties. Depending on the purpose of the table 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.



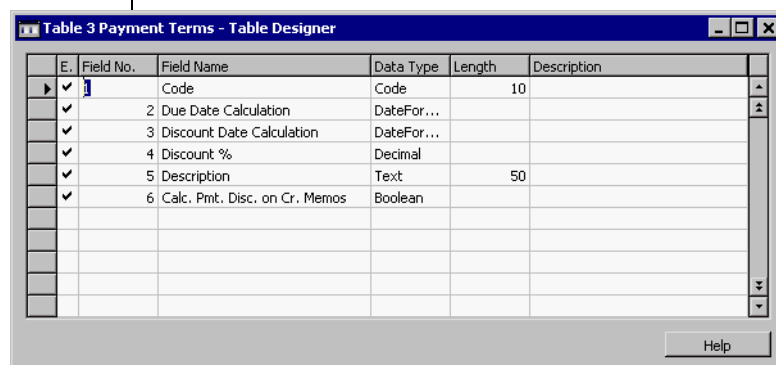
Property Name	Use this property to...
LinkedObject	determine whether this Navision table description is to be linked to an existing SQL Server object.
LinkedInTransaction	determine whether the linked object supports transactions and can be accessed within Navision transactions or does not support transactions and is not under transaction control. This property is only available when the value of the <i>LinkedObject</i> property is set to Yes. For more information, see the section See Linked Objects on page 66.

Refer to the online Reference Guide for additional information about any of these properties.

To view or modify table properties:

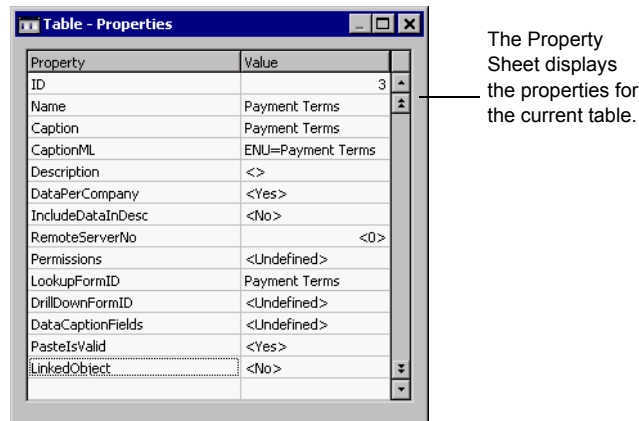
- 1 From the Tools menu, choose Object Designer.
- 2 Click the Table button in the Object Designer window to get a list of the tables.
- 3 Select a table and click the Design button. C/SIDE will display the table in the Table Designer:

The Table Designer



- 4 Place the cursor on an empty line in the Table Designer or click Edit, Select Object. (If you place the cursor on a line defining one of the fields in the table, you will get the properties for the field instead of those for the table.)

5 Choose Properties from the View menu. C/SIDE will display the Property Sheet:



6 If you want to modify the setting of a property, simply enter the new value on the Property Sheet. 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, point at it on the Property Sheet 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.

## Viewing and Modifying Field Properties

Just like tables, all fields in C/SIDE have a number of properties that describe 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 displays next to a control that is based on the field.
CaptionML	provide the text that will be used to identify a control or other object in the user interface.
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.

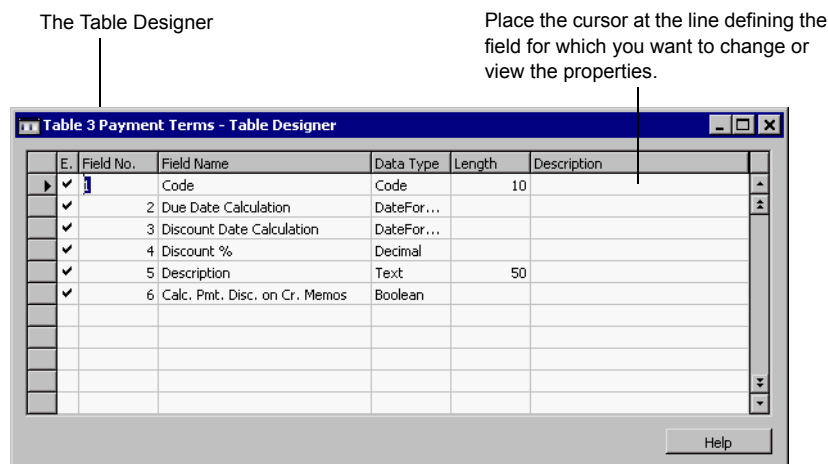
Property Name	Use this property to...
Data Length	specify the maximum length of the data stored in this field.
InitValue	define an initial value for a field.
FieldClass	define the class for a field (that is, specify whether it is a normal field, a FlowField or a FlowFilter field).
CalcFormula	define 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.
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. Refer to the section Setting Relationships Between Tables on page 60 for a detailed discussion about 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.
SubType	define the subtype of a BLOB field (for example a Bitmap or Memo).
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.

Property Name	Use this property to...
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 Navision.
ClearOnLookup	tell the system to delete the current contents of the field before it adds the value the user selects via the lookup.
SubType	provide additional information about what will be contained in this field. This property only applies to BLOB fields.
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.

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

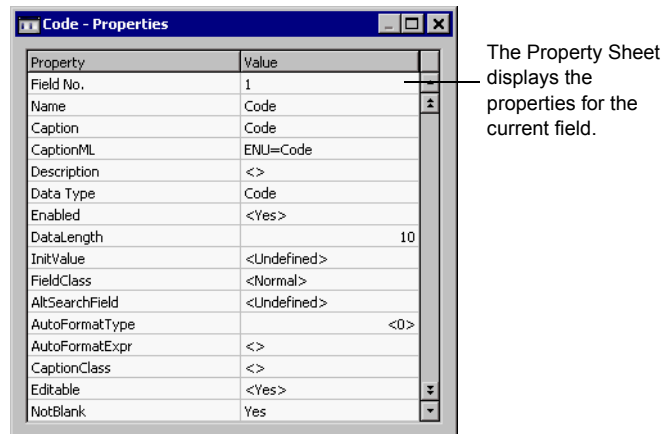
To view or modify field properties:

- 1 From the Tools menu, choose Object Designer.
- 2 Click the Table button in the Object Designer window to get a list of the tables.
- 3 Select a table and click the Design button. C/SIDE will display the table in the Table Designer:



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

5 From the View menu, choose Properties. C/SIDE will display the Property Sheet:



6 If you want to modify the setting of a property, simply enter the new value on the Property Sheet. 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, point at it on the Property Sheet 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.

### 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:

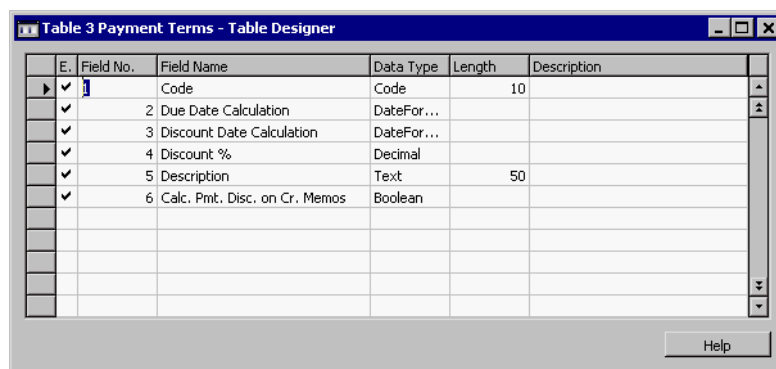
Property Name	Use this property to...
Enabled	determine whether the system will maintain an index for the key. You cannot use a key unless it is enabled.
Key	define the key.
SumIndexFields	determine the fields for which the system will maintain a SumIndex®.
KeyGroups	determine which keygroups the key is a member of.
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 Navision key should be created.

Property Name	Use this property to...
MaintainSIFTIndex	determine whether or not SIFT structures should be created in SQL Server to support the corresponding SumIndexFields for the Navision key.
SIFTLevelsToMaintain	specify which SIFT levels are maintained for a key.

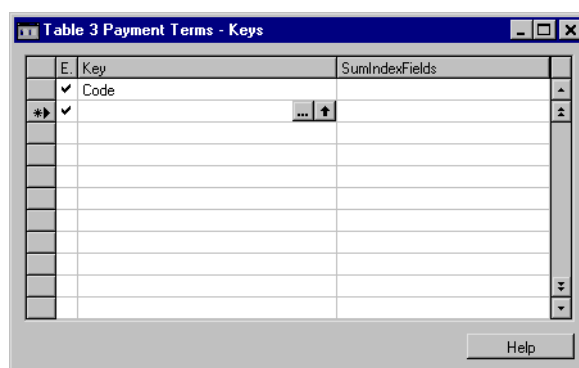
Refer to the online *C/SIDE Reference Guide* for additional information about these properties.

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

- 1 From the Tools menu, choose Object Designer.
- 2 Click the Table button in the Object Designer to get a list of the tables.
- 3 Select a table and click the Design button. C/SIDE will display the table in the Table Designer:

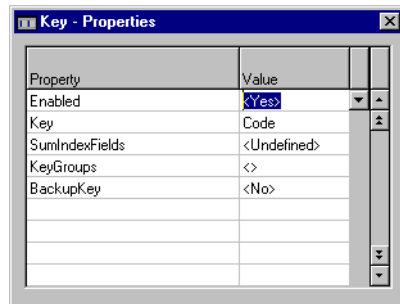


- 4 From the View menu, choose Keys. C/SIDE will display:



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

6 From the View menu, choose Properties. C/SIDE will display the Property Sheet:



The Property  
Sheet for a  
key

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

8 To get Help for a property, point at it on the Property Sheet and press F1.

## 4.2 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 can be thought of as predefined functions that are executed when certain things happen. The bodies of these functions are initially empty and must be defined by the developer. By defining C/AL code in triggers, you can 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:

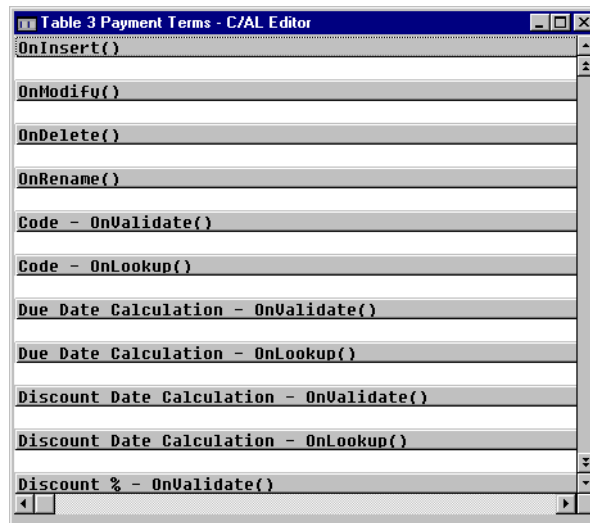
Field Trigger Name	Executed when...
<b>OnValidate</b>	data is entered in a field or when <Record>.VALIDATE is executed in C/AL code.
<b>OnLookup</b>	Lookup (F6) is activated.

If you are not familiar with C/AL programming, please refer to chapter 13, Introducing the C/AL Language, on page 233.

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

- 1 Open the Object Designer 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 will display the code for the table in the Table Designer. The system uses the position of the cursor in the Table Designer to determine what code to display. That is, 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. Notice, however, that 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.





4 Enter or modify the C/AL code in the relevant trigger(s).

## 4.3 SETTING RELATIONSHIPS BETWEEN TABLES

As mentioned in the section Introduction to C/SIDE Application Design on page 12, 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

Because the one-to-many relationship is the most commonly used, this section will focus on this type of relationship. If your database design model indicates that you need to set up a many-to-many relationship, you probably have a problem in your design – it may be 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 text books mentioned in the subsection Recommended Books on Database Design on page 16.

### 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, while 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

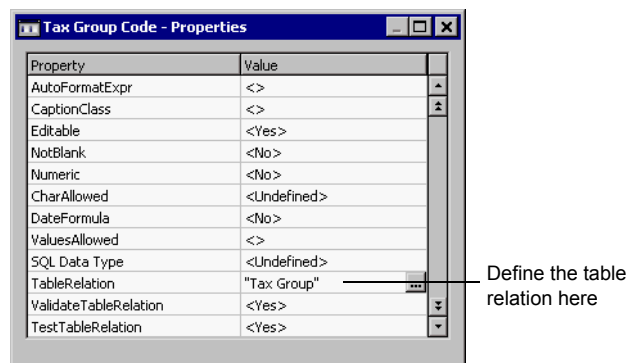
When you create table relations you can either enter them manually or use the assist-edit tool. You will usually enter basic table relations such as:

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

directly on the Property Sheet, whereas you will use assist-edit to enter the more advanced table relations that use conditions and filters. Below you will see how to create (basic) table relations by entering them directly on the Property Sheet. In the following section, you will see how to use the assist edit tool to do the same.

To create a basic table relation:

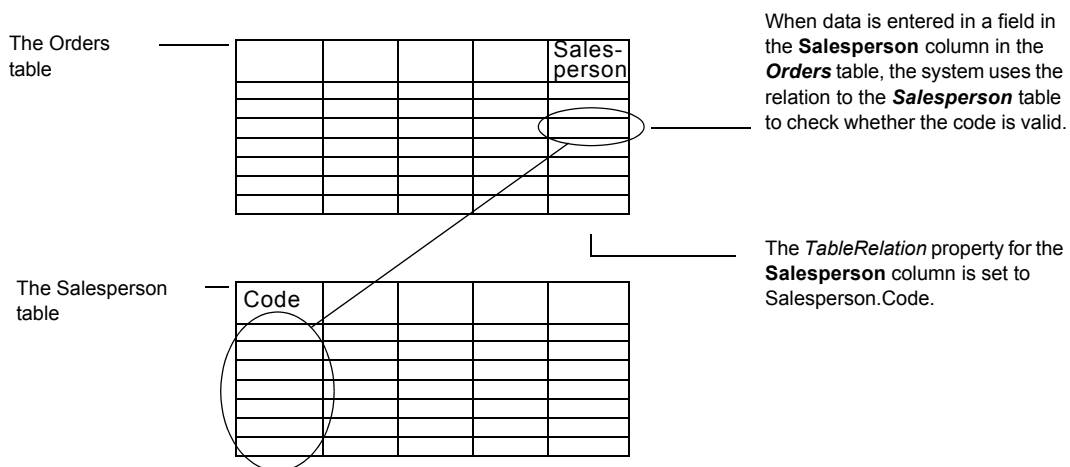
- 1 Open the Object Designer and click Table to see a list of the tables.
- 2 Select a table for which you want to create a relationship, and click Design. C/SIDE will display the table in the Table Designer.
- 3 Make sure that the cursor is placed in the field for which you want to set up a relation.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



- 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. Then 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.

#### EXAMPLE

Assume that you have a **Vendors** table and a **Currency Code** table as in the example above. 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 Assist-Edit

C/SIDE has an assist-edit tool to help you enter advanced table relations. By an advanced table relation, we mean a table relation that is prefixed with a conditional statement and uses filters.

To create a table relation using assist-edit:

- 1 Start exactly as if you are creating a basic table relation. Repeat steps 1 to 3 as described on page 61.



- 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:

- Deleting a table in the Object Designer.
- Restoring a database backup.
- Importing 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.  
.....

## 4.4 CHANGING TABLES THAT CONTAIN DATA

When you design the tables in your database, you determine which fields they contain. Sometime later, you may want to modify the design of some of the tables in your application. Typically you will want to add or delete fields, or make changes to field names or data types.

### Note

.....  
C/SIDE is designed to ensure that you never lose data when you modify the design of a table that contains data.  
.....

### Rules for Changing Tables

Whether it is possible to make changes to a table depends on a number of things. If you haven't added data to the table, you can modify it as you like, but when the table contains data, a number of restrictions apply. The table below gives some general rules:

Modification	Rules
Changing a field name	You can always change the name of a field.
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. Furthermore you must 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.

## 4.5 LINKED OBJECTS

With Navision, 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 Navision 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 Navision transaction control. This means that if a Navision 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 Access to Objects in Other Databases or on Linked Servers on page 69.

#### Note

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

### Creating a Navision Table Description

The following descriptions illustrate the different kinds of Navision 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. Navision will check that the corresponding SQL object exists and that it is compatible with the Navision 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. Navision will check that the corresponding SQL object exists and that it is compatible with the Navision table description when the table is accessed.

### Deleting a Navision 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 Navision 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 Navision 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 Navision transactions.
- If the *LinkedInTransaction* property is set to *No*, all access to the linked object will be performed outside transaction control – independent of Navision 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 Navision table.
- As is the case when creating regular Navision tables, you must be a member of the *db\_owner* fixed database role in the current database.

- As is the case with regular Navision 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 69.
- Navision will automatically grant the required SQL permissions on the object so that you can access it in the same way that regular Navision tables are accessed. It will then be subject to permissions assigned in the Navision security system.
- The object being linked must have a SQL Server table definition that is compatible with the Navision 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 Navision. 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 Navision.

### **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 Navision table definition. It is not necessary to name all the columns in the Navision table definition. For more information about type compatibility, see page 33.
- *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 Navision 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 Navision. 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, Navision will order the data as though a primary key was physically defined. Navision 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 Navision 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 Navision 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: Microsoft Business Solutions–Navision SQL Server Option*.

Once an object has been linked, Navision 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 Navision, and a SQL Server error message appears if you attempt to do this. The ability to redesign the object from within Navision is limited, and these restrictions are described in the next section.

### Redesigning the Navision Linked Object Table Definition

A Navision 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 Navision 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 Navision.
- If the *DataPerCompany* property of the Navision 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.

### Access to Objects in Other Databases or on Linked Servers

You can access objects outside the current database or server from Navision by linking to an appropriately defined view in the current database. You can create a view definition outside of Navision 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.

In order 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.
- All security permissions for linked servers must be granted outside Navision 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. It is not sufficient to grant permissions for the objects to the users that will be using the view. This constraint does not apply if all the users using the view are members of the *db\_owner* fixed database role in the current database.

## Chapter 5

### 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.

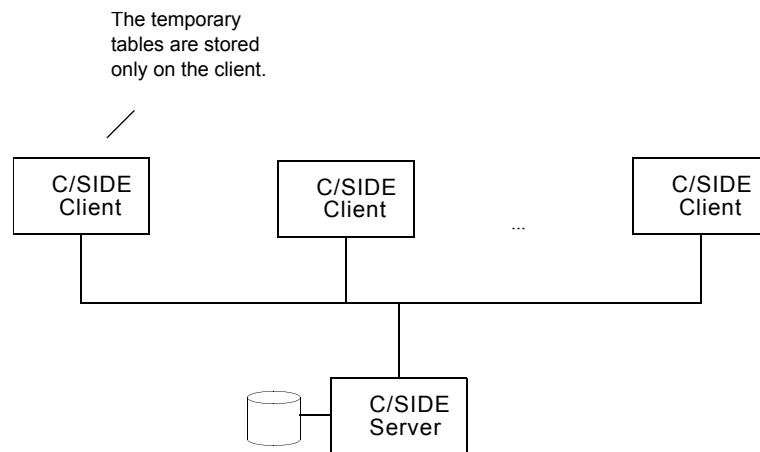
- What Is a Temporary Table?
- What Is a System Table?
- What Is a Virtual Table?
- Overview of C/SIDE Virtual Tables

## 5.1 WHAT IS A TEMPORARY TABLE?

A temporary table can be regarded as a temporary variable that is used to hold a table. A temporary table is intended to be used as a buffer for table data in your C/AL programs. If you are not familiar with C/AL, please refer to chapter 13, Introducing the C/AL Language, on page 233.

You can do almost anything with a temporary table that you can do with a normal database table; the only differences between a normal database table and a temporary table are that:

- Temporary tables aren't stored in the database but only held in memory on your workstation 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, please refer to the section Write Transactions and Recovery on page 428.



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

When you need to perform many operations on data in a specific table in the database, you can load the information into a temporary table while you modify it. Because all operations are local, this will speed up the process.

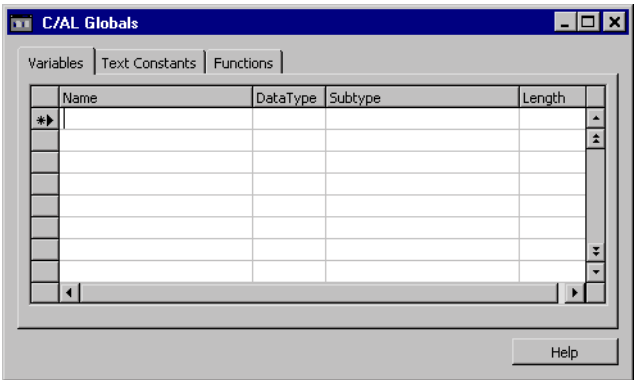
### Defining and Using a Temporary Table

Before you can use a temporary table in your C/AL code, you have to define it. The variable holding a temporary table is defined just like any other global or local variable.

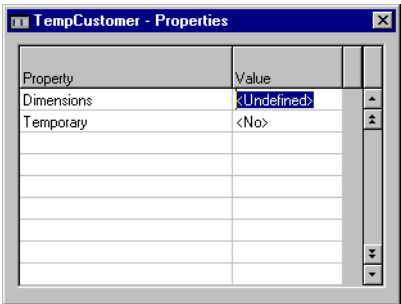
To define a temporary table:

- 1 We assume that you are working in the C/AL editor. From the View menu, choose 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, C/SIDE displays:



- 2 Enter a name for the temporary table variable and enter Record as data type. Use the lookup button in the Subtype field to select the table you want to make a temporary copy of.
- 3 With the cursor still on the line that defines the temporary table, choose Properties from the View menu to display the Property Sheet. C/SIDE displays:



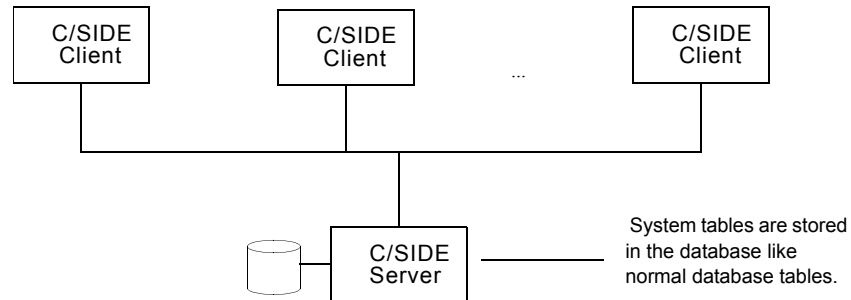
- 4 Change the Temporary property value to Yes.

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

## 5.2 WHAT IS A SYSTEM TABLE?

System tables are stored in the database just like normal database tables. However, system tables, unlike normal database tables, 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.

It is possible to read, write, modify and delete the information in system tables.



There are eight system tables in C/SIDE:

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

The first six tables in the list above all 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 you want to modify. This means that you cannot assign to other users or take away from them permissions that you do not have yourself.

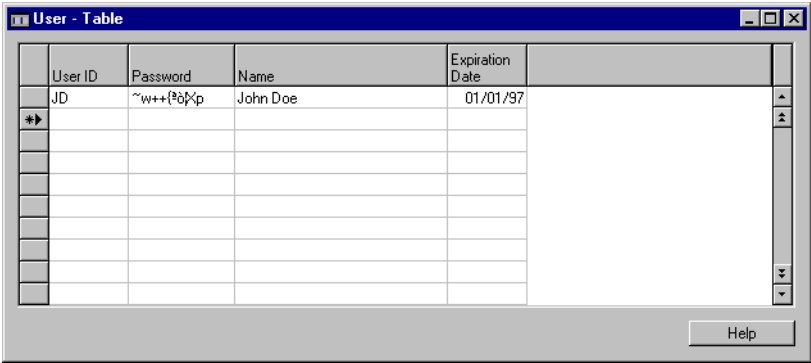
.....

The following subsections provide an overview of these system tables. For further information, see the manual *Installation & System Management: Microsoft Business Solutions–Navision Database Server*.



The User System Table

The **User** system table provides an overview of all the user IDs you have defined in your database for users with database logins. Each record in the **User** system table defines a single user ID. For information about creating database logins, see the manual *Installation & System Management: Microsoft Business Solutions–Navision Database Server*.



User ID	Password	Name	Expiration Date
JD	~w+++{?&Xp	John Doe	01/01/97

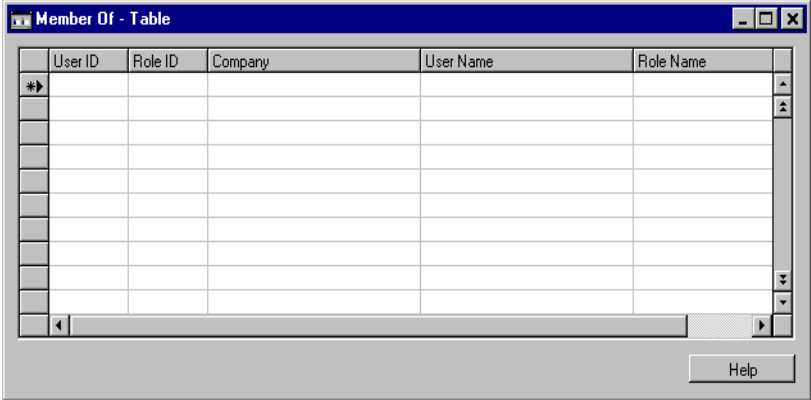
For each ID defined in your database, the **User** system table includes information about the password (displayed in encrypted form on your screen), the real name of the user and how long the user’s ID is valid. You can create new user IDs by entering appropriate data in this table. Correspondingly, you can 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 system will automatically remove the corresponding entries in the **Member Of** system table.  
.....

The Member Of System Table

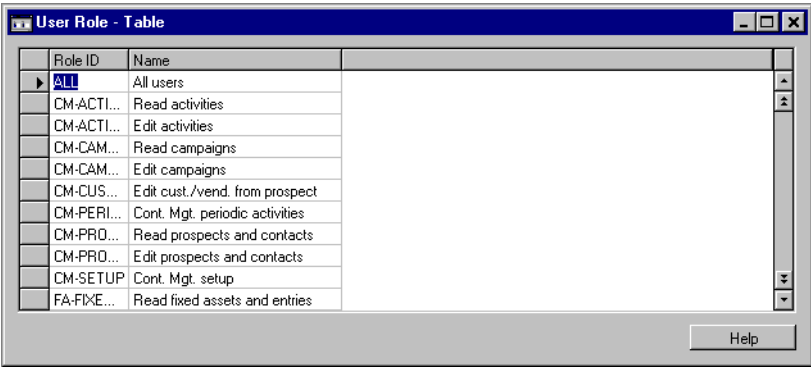
The **Member Of** system table provides an overview of which user groups (roles) a user is a member of. Each user (ID) can be a member of any number of user groups.



User ID	Role ID	Company	User Name	Role Name

The User Role System Table

The **User Role** system table provides 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.



Role ID	Name
ALL	All users
CM-ACTI...	Read activities
CM-ACTI...	Edit activities
CM-CAM...	Read campaigns
CM-CAM...	Edit campaigns
CM-CUS...	Edit cust./vend. from prospect
CM-PERI...	Cont. Mgt. periodic activities
CM-PRO...	Read prospects and contacts
CM-PRO...	Edit prospects and contacts
CM-SETUP	Cont. Mgt. setup
FA-FIXE...	Read fixed assets and entries

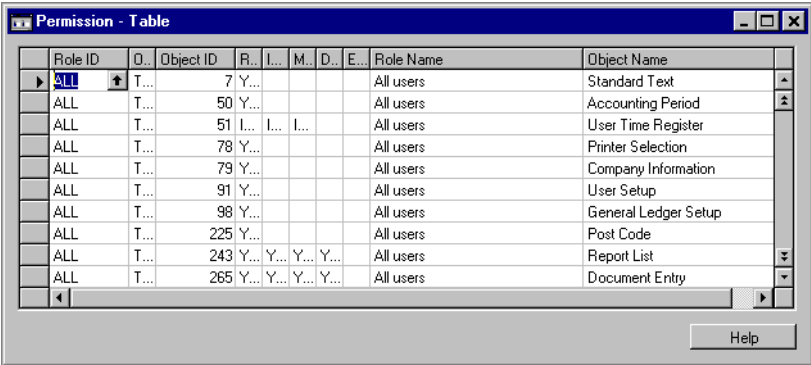
Deleting a Record

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

The Permission System Table

You can use the **Permission** system table to define what different user roles are allowed to do. Permissions are specified for objects; 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:

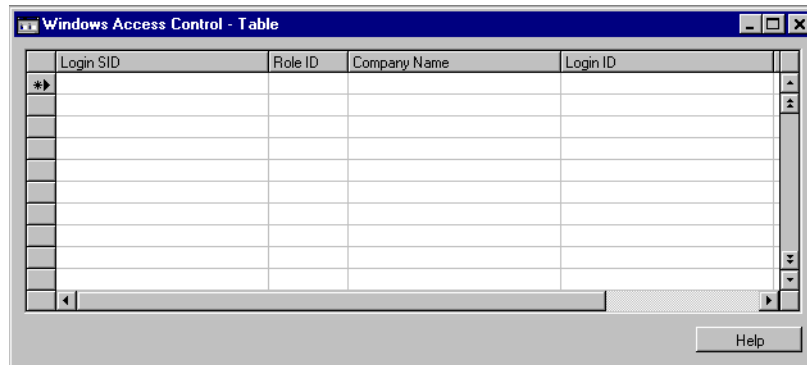
- Read
- Create/Insert
- Modify
- Delete
- Execute



Role ID	O...	Object ID	R...	I...	M...	D...	E...	Role Name	Object Name
ALL	T...	7	Y...					All users	Standard Text
ALL	T...	50	Y...					All users	Accounting Period
ALL	T...	51	I...	I...				All users	User Time Register
ALL	T...	78	Y...					All users	Printer Selection
ALL	T...	79	Y...					All users	Company Information
ALL	T...	91	Y...					All users	User Setup
ALL	T...	98	Y...					All users	General Ledger Setup
ALL	T...	225	Y...					All users	Post Code
ALL	T...	243	Y...	Y...	Y...	Y...		All users	Report List
ALL	T...	265	Y...	Y...	Y...	Y...		All users	Document Entry

### The Windows Access Control System Table

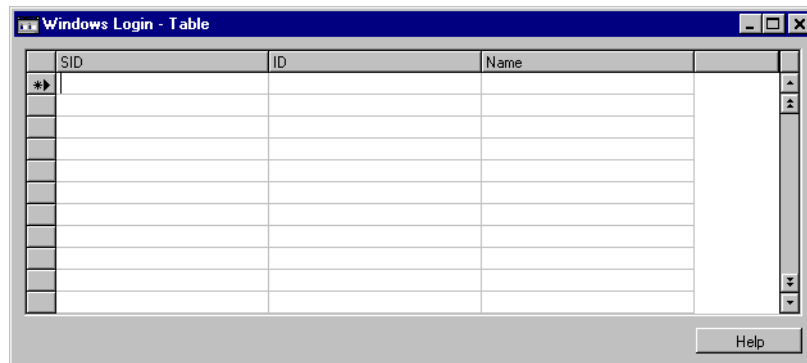
The **Windows Access Control** system table enables you to manage the access rights of a user or group of users to Windows 2000, and thereby to Navision. Each user's or group's Windows login has a unique security identifier (SID). Further, each user or group has a role ID, which relates to a set of permissions within a certain company in Navision. 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

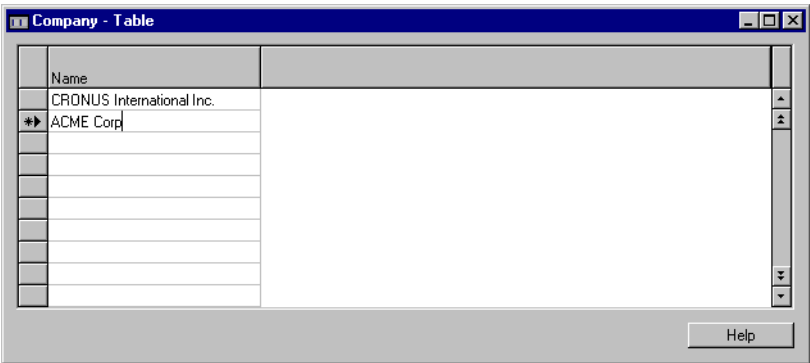
The **Windows Login** system table enables you to define which Windows users and groups can log on to the system. Only those Windows users or those who are members of a Windows group that are listed here can log on. Each Windows user or group has a unique security identifier (SID). The name of the user 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	

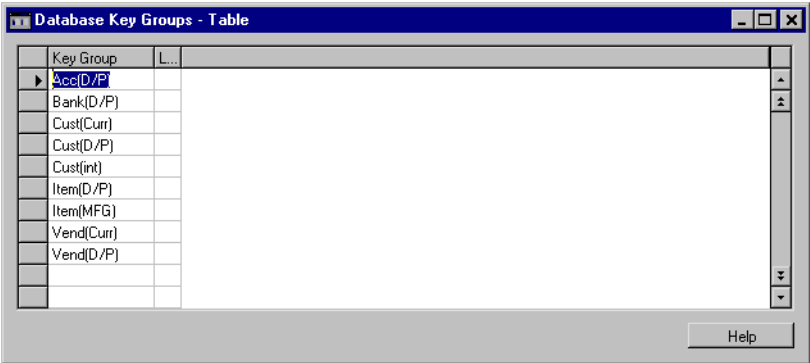
### The Company System Table

The **Company** system table provides 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 in your database by deleting the corresponding record in the **Company** table. (When you do that, you delete all the tables in the company. At the same time you also delete all permissions that include this company.)



The Database Key Groups System Table

The **Database Key Groups** system table provides 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 make use of key groups, select File, Database, Information and then click Tables. The **Database Information** window appears. You can now click Key Groups.

## 5.3 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 working with ordinary tables. For example, you can use filters to get subsets or ranges of integers or dates.

The virtual tables provide information such as:

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

## 5.4 OVERVIEW OF C/SIDE VIRTUAL TABLES

C/SIDE contains numerous virtual tables, including:

### Virtual Tables

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

### Using the Virtual Tables

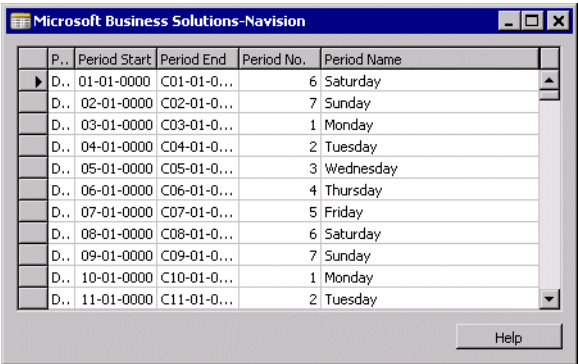
The first of these virtual tables give you easy access to dates, integers, information about your operating system files and the logical drives on your computer.

### The Date Virtual Table

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

Field	Comments
Period Type	Days, weeks, months, quarters or years
Period Start	The date of the first day in the period
Period End	The date of the last day in the period

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



The screenshot shows a window titled "Microsoft Business Solutions-Navision" displaying a table with the following columns: P., Period Start, Period End, Period No., and Period Name. The table contains 11 rows of data, each representing a day of the week from Saturday to Tuesday.

P.	Period Start	Period End	Period No.	Period Name
D..	01-01-0000	C01-01-0...	6	Saturday
D..	02-01-0000	C02-01-0...	7	Sunday
D..	03-01-0000	C03-01-0...	1	Monday
D..	04-01-0000	C04-01-0...	2	Tuesday
D..	05-01-0000	C05-01-0...	3	Wednesday
D..	06-01-0000	C06-01-0...	4	Thursday
D..	07-01-0000	C07-01-0...	5	Friday
D..	08-01-0000	C08-01-0...	6	Saturday
D..	09-01-0000	C09-01-0...	7	Sunday
D..	10-01-0000	C10-01-0...	1	Monday
D..	11-01-0000	C11-01-0...	2	Tuesday

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 **G/L Balance** form below is a typical example. You will learn how to design forms in part 2, Forms, on page 97.

Period	Net Change	Balance	Budgeted Net Change	Budgeted Balance	Net Change
1994	0.00	0.00	0.00	0.00	0.00
1995	0.00	0.00	0.00	0.00	0.00
1996	0.00	0.00	0.00	0.00	0.00
1997	0.00	0.00	0.00	0.00	0.00
1998	0.00	0.00	0.00	0.00	0.00
1999	0.00	0.00	0.00	0.00	0.00
2000	0.00	0.00	0.00	0.00	0.00
2001	0.00	0.00	0.00	0.00	0.00
2002	0.00	0.00	0.00	0.00	0.00

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 has 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 provides an overview of the files in a directory on your disk system. The **File** virtual table has the following fields:

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

### The Drive Virtual Table

This virtual table provides an overview of the logical drives on your computer. The **Drive** virtual table has 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 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

#### Note

.....  
 If there is no diskette 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, as they provide a lot of useful information about the users that are connected to the system. They also provide information about 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. You can get access to the **Monitor** virtual table directly from C/SIDE by clicking Tools, Client Monitor.

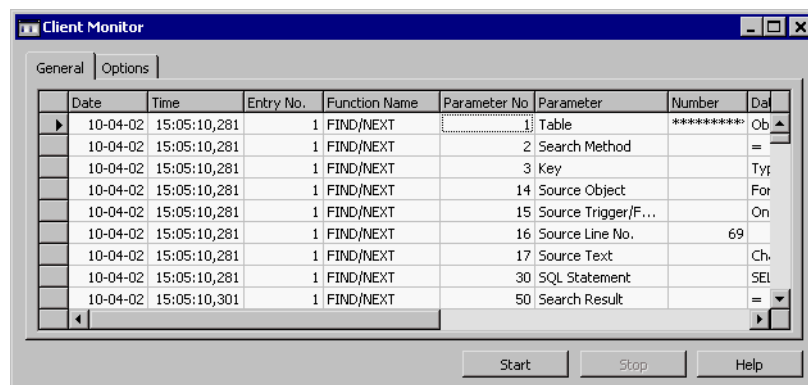
This virtual table is used by C/AL programmers to get an overview of the time consumption of specific operations. C/AL programmers can use the information in this virtual table to optimize the performance of their code. The **Monitor** virtual table contains the following fields:

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.

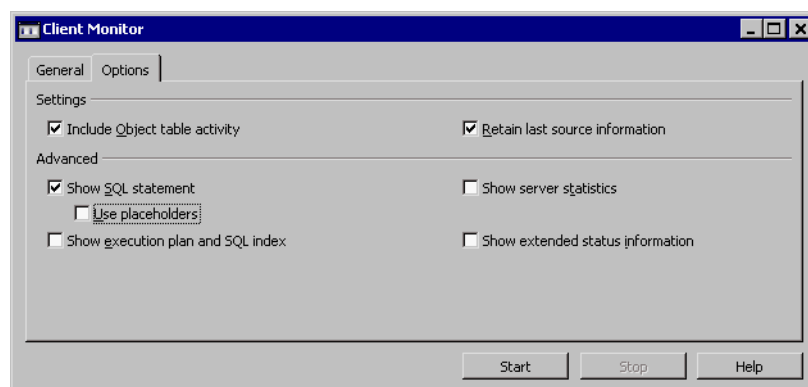


Field	Comments	Possible Values
<b>Parameter No.</b>	The number of the parameter	Depending on the number of parameters
<b>Parameter</b>	The name of the parameter	Table, Key, Order, Filter, Search Method, Search Result, Records Found, Sum, CPU (ms), Records Read, Sum Intervals, Records Deleted, Records 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

You can access the **Monitor** virtual table directly by clicking Tools, Client Monitor and the **Client Monitor** window appears:



The **Client Monitor** window contains two tabs and 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

New parameters have been added to the Client Monitor to improve troubleshooting and performance analysis when you are running the SQL Server Option for Navision. These new parameters can be configured dynamically and include status information about caching, SQL statements and execution plans.

Collecting server statistics is time-consuming. Therefore, 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 in order to use its graphical execution plan does not necessarily give the same plan or statistics as it does when the same statement is executed from within Navision. 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 Open the Client Monitor.
- 2 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 you want to investigate.
- 5 When you have completed these tasks, click Tools, Client Monitor to open the 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 Navision 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>	Shows 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>	Shows 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	
<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	

Field	Comments	SQL Server	NDS
<b>Blocked</b>	Shows 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

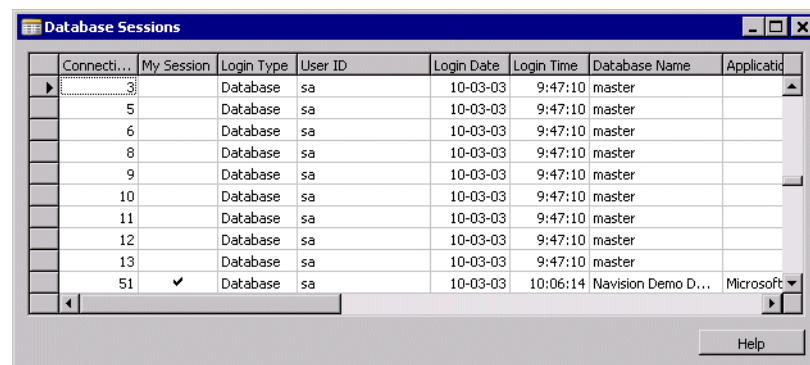
### Important

.....  
 If a solution uses any of the fields that are not available in the SQL Server Option for Navision, 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 will appear.  
 .....

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

To access this virtual table from C/SIDE, follow this procedure:

- 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 Option) appears:



An administrator can cancel 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. The administrator must be a member of either the *sysadmin* or *processadmin* SQL Server server rolls.

**Note**  
.....  
The **Database Sessions** window displays different fields from the **Session** virtual table depending on which server you are running.  
.....

The Database File Virtual Table

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

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

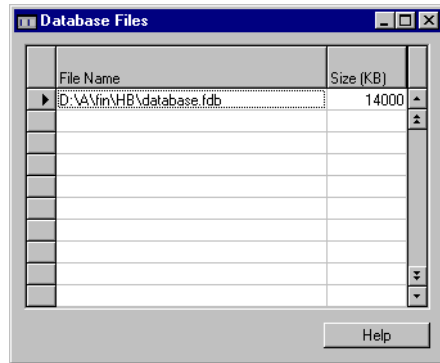
**Important**  
.....  
If a solution uses any of the fields that are not available in the SQL Server Option for Navision, it will have to be modified to run on SQL Server. These fields will not be created on SQL Server. If the program tries to access them, an error message will appear.  
.....

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

To access the **Database File** virtual table from C/SIDE, follow this procedure:

- 1 Click File, Database, Information.
- 2 Click the **Database** tab.

3 Click the AssistButton ↓ in the **Database Name** field. C/SIDE will display:



#### Note

.....  
 The **Database Files** window displays only some of the fields in the **Database File** virtual table. Note, this window does not appear in the Microsoft SQL Server Option for Navision.  
 .....

### The Table Information Virtual Table

This virtual table contains various information about database tables. The **Table Information** virtual table has 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 MICROSOFT SQL SERVER OPTION FOR NAVISION.

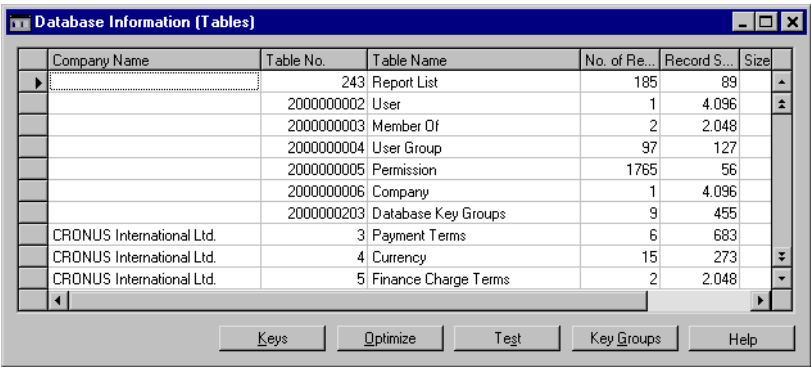
#### Important

.....  
 If a solution uses any of the fields that are not available in the SQL Server Option for Navision, it will have to 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 will appear.  
 .....

C/SIDE uses the **Table Information** virtual table to display information about database tables.

To access the **Table Information** virtual table from C/SIDE:

- 1 Click File, Database, Information.
- 2 Click Tables. C/SIDE will display:



**Note**

.....

This window does not display all the fields in the **Table Information** virtual table. Use the horizontal scroll bar to view the information in the hidden fields.

.....

The Field Virtual Table

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

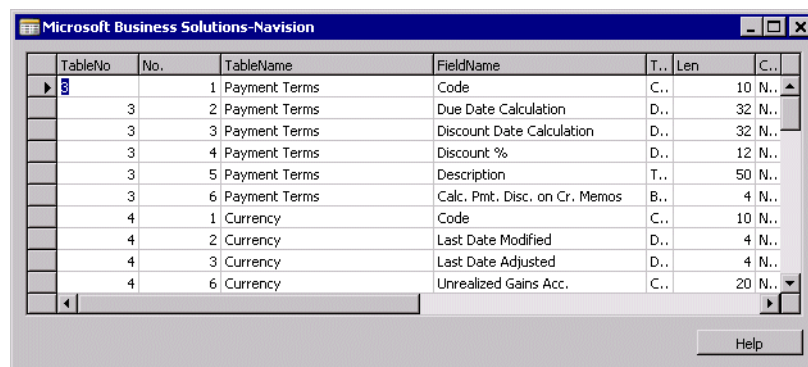
Field	Comments
TableNo	This field shows the ID number for the table.
No.	This field shows the number assigned to the field.
Table Name	This field shows the name of the table.
FieldName	This field shows the name of the field.
Type	The field indicates the data type assigned to the field, for example, decimal.
Len	This field shows the length of the field entry in bytes.
Class	This field indicates the class of the field, for example, FlowField.
Enabled	This field indicates whether the field is enabled.
Type Name	This field shows the data type assigned to the field. The length of the field entry in bytes is included for Code and Text data types.
Field Caption	This field shows the caption of the field in the language that has been selected.
RelationTableNo	This field shows the ID number for the table that the field is related to.



Field	Comments
<b>RelationFieldNo</b>	This field indicates the number of the field in another table that the field is related to.
<b>SQLDataType</b>	This field shows the data type assigned to code fields in the Microsoft SQL Server Option for Navision.

To access the **Field** virtual table from C/SIDE, follow this procedure:

- 1 Click Tools, Object Designer.
- 2 In the Object Designer, click Form.
- 3 Click New.
- 4 Create a tabular-type form based on the **Field** table. C/SIDE will display:



TableNo	No.	TableName	FieldName	T.	Len	C..
3	1	Payment Terms	Code	C..	10	N..
3	2	Payment Terms	Due Date Calculation	D..	32	N..
3	3	Payment Terms	Discount Date Calculation	D..	32	N..
3	4	Payment Terms	Discount %	D..	12	N..
3	5	Payment Terms	Description	T..	50	N..
3	6	Payment Terms	Calc. Pmt. Disc. on Cr. Memos	B..	4	N..
4	1	Currency	Code	C..	10	N..
4	2	Currency	Last Date Modified	D..	4	N..
4	3	Currency	Last Date Adjusted	D..	4	N..
4	6	Currency	Unrealized Gains Acc.	C..	20	N..

#### Note

.....  
 This window does not display all the fields in the **Field** virtual table. Use the horizontal scroll bar to view the information in the hidden fields.  
 .....

### The Navision Server Virtual Table

This virtual table contains information about the Navision Database Servers and where they reside on the network. You can see this information when the application runs on the Microsoft Windows 2000 operating system. Navision retrieves the data from Active Directory. See Microsoft's Windows 2000 documentation for information about Active Directory.

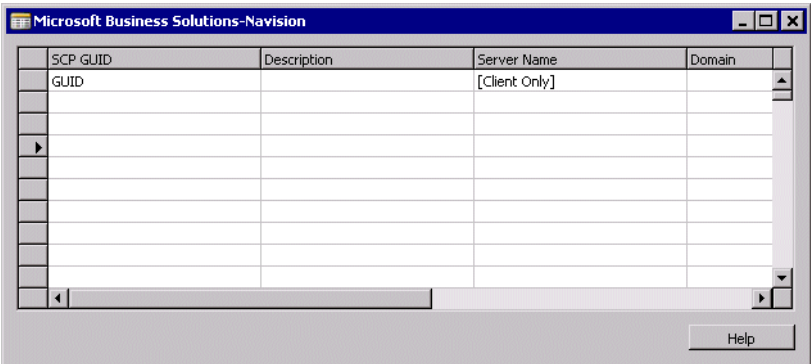
The **Navision Server** virtual table contains the following fields:

Field	Comments
<b>SCP GUID</b>	Shows the globally unique identifier (GUID) for the Navision Database Server's service connection point (SCP).
<b>Description</b>	Shows the name that has been given to the Navision Database Server.
<b>Server Name</b>	Shows the name of the computer on which the server is installed.

Field	Comments
Domain	Shows the name of the domain on which the server resides. A domain can consist of more than one physical location.
Net Type	Shows the network type (TCP/IP or NetBIOS) that is used to connect to the Navision Database Server.
Port Number	Shows the port number for the transfer of data to the server.
Distinguished Name	Shows the unique name of the Navision Database Server in Active Directory.
Online	This field indicates whether or not the server is running and thereby available.

To access the **Navision Server** virtual table from C/SIDE, follow this procedure:

- 1 Click Tools, Object Designer.
- 2 In the Object Designer, click Form.
- 3 Click New.
- 4 Create a tabular-type form based on the **Navision Server** table. C/SIDE displays:



**Note**

.....

This window does not display all the fields in the **Navision Server** virtual table. Use the horizontal scroll bar to view the information in the hidden fields.

.....

**The Server Virtual Table**

The **Navision Server** virtual table is only available when you are running on Navision Database Server. If you are running on the SQL Server Option for Navision, you can access the **Server** table.

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

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

### The Windows Object Virtual Table

This virtual table provides an overview of Windows users and Windows groups, which can be integrated in the Navision security system. You can see this information when the application is running on a client with the Microsoft Windows 2000 operating system. Navision retrieves the data from Active Directory. See Microsoft's Windows 2000 documentation for information about Active Directory.

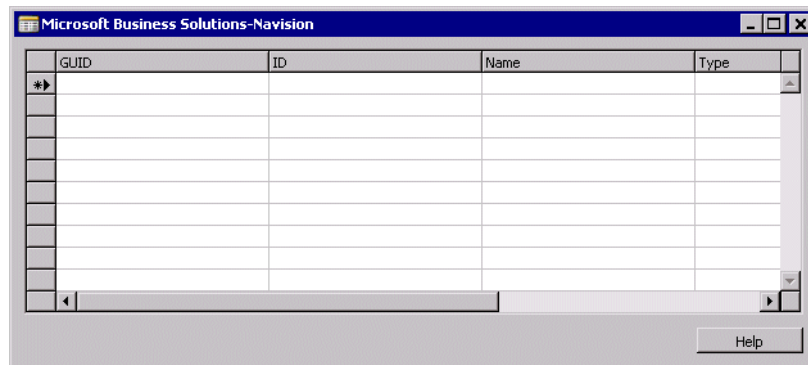
The **Windows Object** virtual table has the following fields:

Field	Comments
<b>GUID</b>	This field shows the globally unique identifier (GUID) for the Windows user or group.
<b>ID</b>	This field shows 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>	This field shows 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>	This field indicates whether the object is a Windows user or a Windows group.
<b>SID</b>	This field shows the unique security identifier (SID) for the Windows user or group.
<b>Distinguished Name</b>	The distinguished name identifies the domain that holds the Windows object as well as the complete path by which the object is reached. Every object in the Active Directory has a unique distinguished name.

To access the **Windows Object** virtual table from C/SIDE, follow this procedure:

- 1 Click Tools, Object Designer.
- 2 In the Object Designer, click Form.
- 3 Click New.

- 4 Create a tabular-type form based on the **Windows Object** virtual table. C/SIDE will display:



### The Windows Group Member Table

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

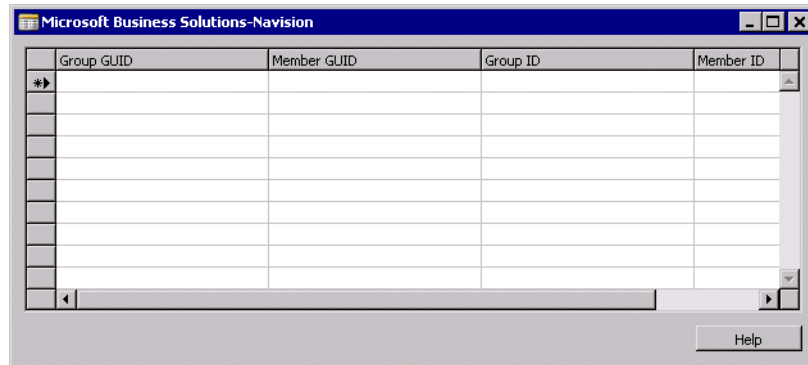
You can access this virtual table if you are running Navision on the Microsoft Windows 2000 operating system. The **Windows Group Member** virtual table has the following fields:

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

To access the **Windows Group Member** virtual table from C/SIDE, follow this procedure:

- 1 Click Tools, Object Designer.
- 2 In the Object Designer, click Form.
- 3 Click New.

- 4 Create a tabular-type form based on the **Windows Group Member** virtual table. C/SIDE displays:



### The SID - Account ID Virtual Table

This virtual table can convert the security identifier (SID) for a Windows object into an ID. It can also convert an ID for 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>	This field shows the security identifier (SID) for the Windows user or group.
<b>ID</b>	This field shows 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.  
.....

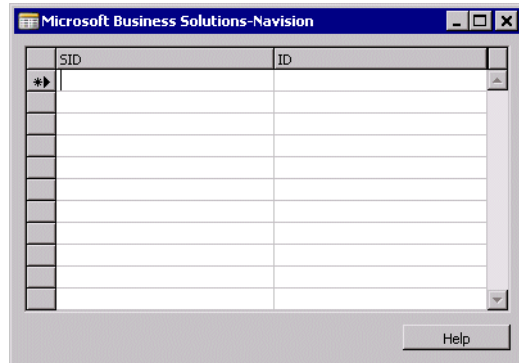
### The User SID Virtual Table

This virtual table shows the security identifiers (SIDs) and IDs for 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>	This field shows the security identifier (SID) for the groups that the user who is logged on to the system is a member of.
<b>ID</b>	This field shows 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.

To access the **User SID** virtual table from C/SIDE, follow this procedure:

- 1 Click Tools, Object Designer.
- 2 In the Object Designer, click Form.
- 3 Click New.
- 4 Create a tabular-type form based on the **User SID** virtual table. C/SIDE displays:



The screenshot shows a window titled "Microsoft Business Solutions-Navision". Inside the window is a table with two columns: "SID" and "ID". The table has a header row and several empty rows below it. A vertical scrollbar is on the right side of the table. At the bottom right of the window is a "Help" button.

SID	ID







## Chapter 6

### 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.

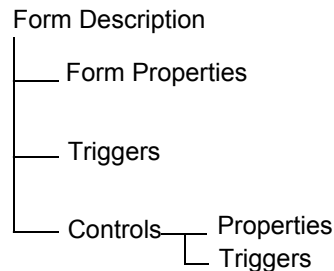
- What Are Forms?
- Creating Forms
- Selecting, Moving and Adjusting Controls
- Saving, Compiling and Running Forms

## 6.1 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: forms are used for entering information into database tables and for retrieving and displaying information from database tables. It is through forms that users 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 figure below shows the components of a form and how they are related. This and the following chapters will explore each component in depth.



Forms are created and edited in the Form Designer.

### What are Controls?

All information on 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. An example is a frame. The frame itself does not display data or information, but it can contain a number of other controls that you want to group. A powerful container control is the tab control. A tab control really is a number of frames or pages that are placed on top of each other. The user can switch between the pages by clicking the tabs that have captions. Tab controls make it possible to group information on a form so that each page is not cluttered with information, and it is very fast and easy to switch between pages.

Another concept is control branches, which consist of a parent control and subordinate or child controls. The best 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 on the form.

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

Typically, a form is related to a database table and will be used to enter information into the table and to display information from the table. The form is said to be *bound* to the 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 the user can choose other forms or reports to run.

The controls on a form that is bound to a table are usually bound to fields in the same table. There need not be a control for every field in the table, nor do all controls on the form need to be bound to table fields: controls that aren't bound to fields are called unbound controls. An example is a command button that causes the information on the form to be printed; another is a control that contains a descriptive text. An important category of unbound controls includes controls displaying information – based on the underlying table or user-entered values – that is calculated as the form is displayed.

### What Are Form and Control Properties?

Properties describe how a control is placed on the form, what field it is related to and what happens when information is entered into the field, among other things. Different types of controls have different sets of properties. For example, a text box, the control type that is typically used to display the contents of a database field, has more properties than a picture box, a control used to display bitmap pictures.

The form itself also has properties. For example, you can specify whether the form is to be used only for displaying information or whether it will be possible to insert new records or update existing ones.

Properties are defined on the Property Sheet that can be edited when the form is opened in the Form Designer.

### What Are Triggers?

Certain predefined events that happen to a form or a control cause the system to execute a user-definable C/AL function – the event *triggers* the function. The event and the function are together called a trigger. Form triggers include OnOpenForm, containing statements that will be executed when the form is opened, and OnModifyRecord, containing statements that will be executed before the system accepts changes the user makes to a record. Triggers are edited in the C/AL editor, which can be opened from the Form Designer.

## 6.2 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. C/SIDE offers an alternative method that is fast and easy to use and therefore preferable when you are just beginning to create forms: you can use a *form wizard*. A form wizard prompts you for the minimum amount of information needed to create a form and then does the rest of the work for you. The automatically created form can be changed later on in the Form Designer.

Forms that display one record at a time are called card forms, while forms that show several records at a time are called tabular forms. The form wizard will help you create either type.

A Card Form

No. ....	1	Phone ....	11223344
Name. ....	John Doe	Company ....	ACME Corp
Address. ....	17 Riverside Drive	Title. ....	Sales Clerk

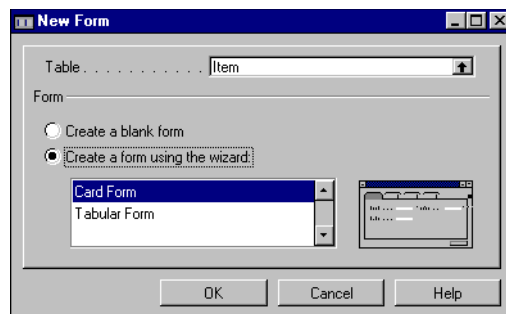
A Tabular Form

No.	Name	Search N...	Name 2	Address
10000	The Cannon Group PLC	THE CAN...		192 Market Square
20000	Selangorian Ltd.	SELANG...		153 Thomas Drive
30000	John Haddock Insurance Co.	JOHN HA...		10 High Tower Green
40000	Deerfield Graphics Company	DEERFIE...		10 Deerfield Road
50000	Guildford Water Department	GUILDFO...		25 Water Way
60000	Blanemark Hifi Shop	BLANEM...		28 Baker Street
61000	Fairway Sound	FAIRWA...		159 Fairway
62000	The Device Shop	THE DEVI...		273 Basin Street
01121212	Spotsmeyer's Furnishings	SPOTSM...		612 South Sunset Dri
01445544	Progressive Home Furnishings	PROGRE...		3000 Roosevelt Blvd.

## Creating Forms with a Form Wizard

To create a form using the form wizard:

- 1 From the Tools menu, choose the Object Designer.
- 2 Click the Form button In the Object Designer.
- 3 Click the New button. C/SIDE will display this form:

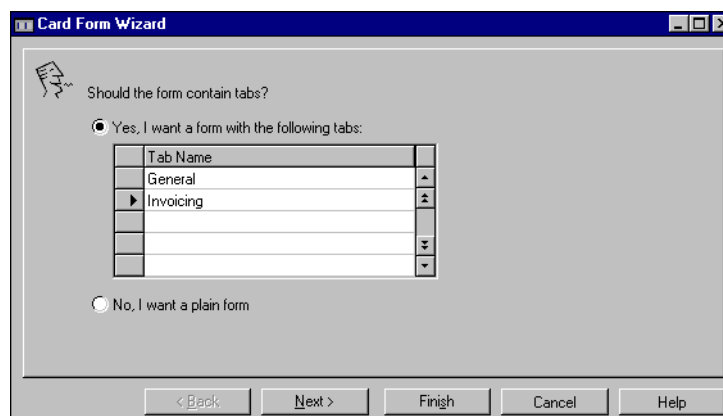


- 4 If you are creating a form that is related to a table, type the name of the table in the Table field. You can also click the Lookup button and choose the table from a list. Once you have entered the table name in the field, press ENTER.
- 5 Select the option called "Create a form using the wizard." Then you must select the type of form you want the wizard to create: card form or tabular form. After this, click OK.

## Creating a Card Form

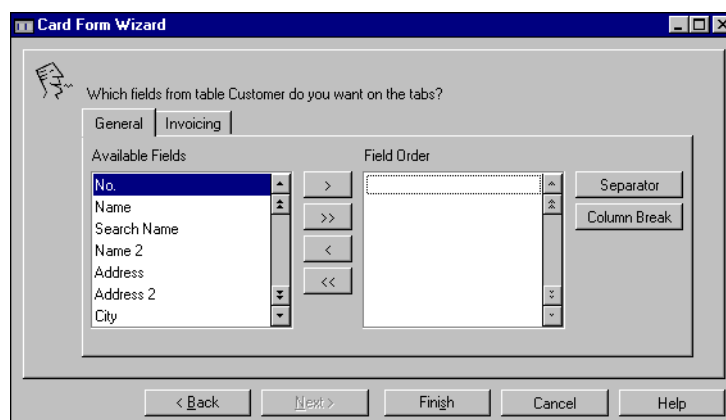
To create a card form, follow the steps outlined above. Then proceed as follows:

- 1 The system asks whether you want the form to have tabs or whether it should be a plain form. A form with tabs is a multi-page form where the user can switch between pages by clicking the tabs.



- 2 If you choose to create tabs, type a caption for each tab you want. In either case, click the Next button when you are ready to continue.

- 3 In the next form the wizard displays, you must choose those fields from the database table you want on your form.



If you are creating a form with tab controls, begin by choosing the page on which you want certain fields to appear. You can switch between pages by clicking on the tabs, which have the captions you have defined.

- 4 The form that the wizard displays contains two lists: the **Available Fields** list, which contains all fields in the table, and the **Field Order** list, which contains the fields that have been 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

.....

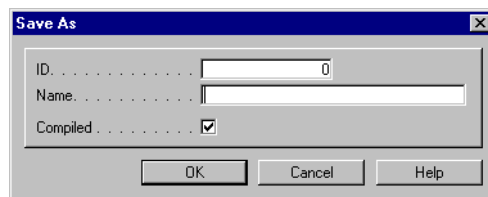
The contents of the **Available Fields** list are the *Caption* properties of the fields available to you – not the *Name* properties. For more information about captions, see Chapter 18.

.....

- 5 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 a different order, move a field by removing it from the **Field Order** list and inserting it again in the position you want.
- 6 By clicking Separator, you can insert a small amount of extra vertical space between the controls; this allows you to group information together in a logical and visually pleasing way. Note that tabs provide a more powerful way of grouping information together on a form. The separator will be inserted after the field that is currently selected on the **Field Order** list; you can remove a separator by selecting it and clicking <.
- 7 To insert a column break, click Column Break. The rules for insertion and deletion are the same as for a separator. If you feel you need to create three or more columns, you should consider using tabs instead.

- 8 When you are satisfied, click Finish. The form wizard will create your form, and the **Form Designer** window will open containing the new form. You can test-run the form by clicking File, Run.
- 9 Close this window and answer Yes to save the form. You will be 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.

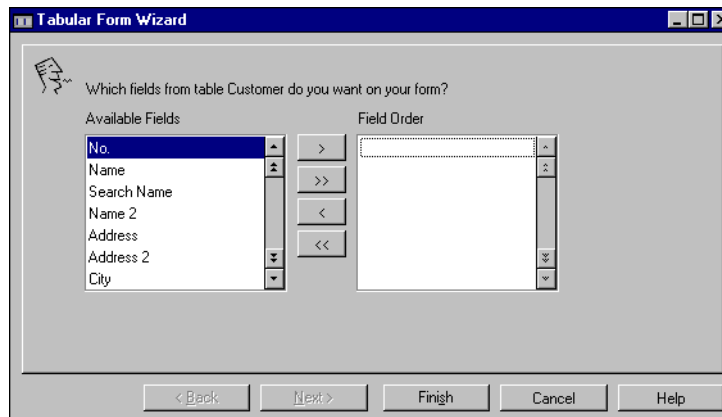
You can also do it later, by selecting the form in the **Object Designer** window and clicking Tools, Compile.



- 10 When the form has been saved and compiled, it can be run. Select the form from the **Object Designer** window and click Run.

### Creating a Tabular Form

To create a tabular form, follow the initial steps outlined in "Creating Forms with a Form Wizard" on page 103. Then proceed as follows:

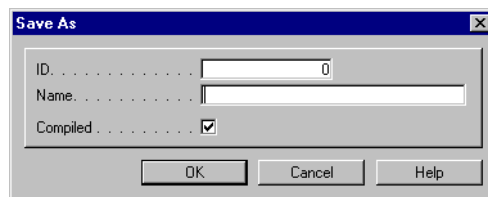


- 1 You are prompted to choose the fields from the database table that you want on your form. The form that the wizard displays contains two lists: the **Available Fields** list, which contains all fields in the table, and the **Field Order** list, which contains the fields that have been selected. To 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. For more information about captions, see Chapter 18.  
 .....

- 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 a different order, move a field by removing it from the **Field Order** list and inserting it again in the position you want.
- 3 When you are satisfied, click Finish. The form wizard will create your form, and the **Form Designer** window will open containing the new form. You can test-run the form by clicking File, Run.
- 4 Close this window and answer Yes to save the form. You will be 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 do it later, by selecting the form in the **Object Designer** window and clicking Tools, Compile.

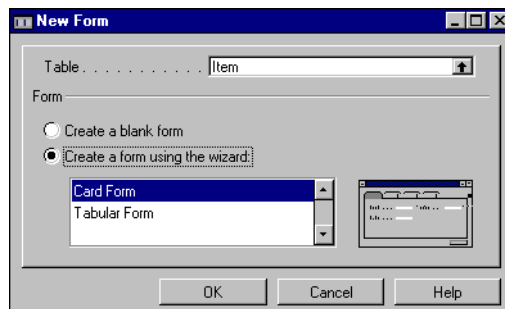


- 5 When the form has been saved and compiled, it can be run. Select the form in the **Object Designer** window and click Run.


### Creating Forms Without a Wizard

To create a form without using a wizard:

- 1 Click Tools, Object Designer.
- 2 In the **Object Designer** window, click Form.
- 3 Click New. C/SIDE will display this form:





- 4 If you are creating a form that is related to a table, type the name of the table in the **Table** field. You can also click the AssistButton  and select the table from a list.
- 5 Select Create a blank form, and click OK.
- 6 The Form Designer will open, displaying an empty form. Chapter Chapter 7, Designing Forms, describes in detail how to design the form by adding controls, changing properties and so forth.
- 7 You can test-run the form at any point by clicking File, Run.
- 8 When you have finished designing the form, close the **Form Designer** window, and answer Yes to save the form. You will be 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 do it later, by selecting the form in the **Object Designer** window and clicking Tools, Compile.

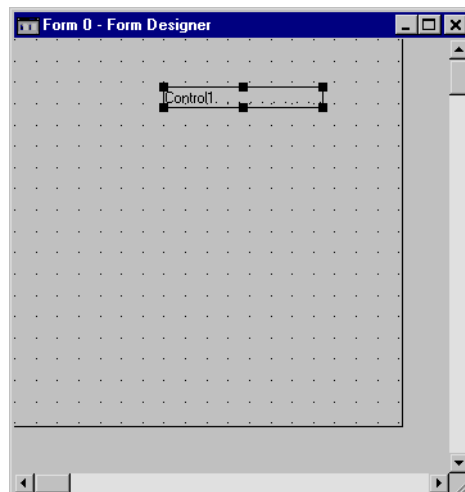
## 6.3 SELECTING, MOVING AND ADJUSTING CONTROLS

This section describes how to select and move controls, and how to adjust controls by aligning and sizing them.

### Selecting Controls

To move or adjust a control, you must first 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 pointing at it and clicking the mouse. In order 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 will be surrounded by a box with sizing handles.

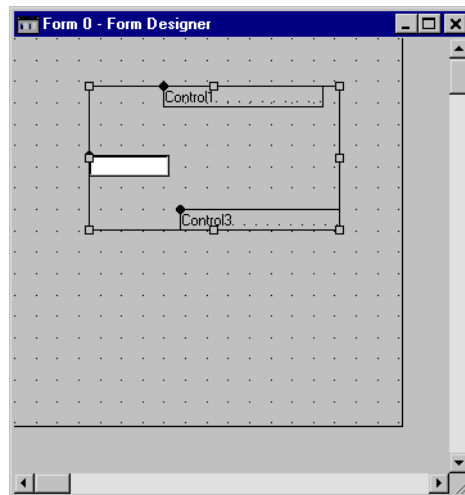


### Multiple Selections

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 will be described below).

### Adding to a Selection

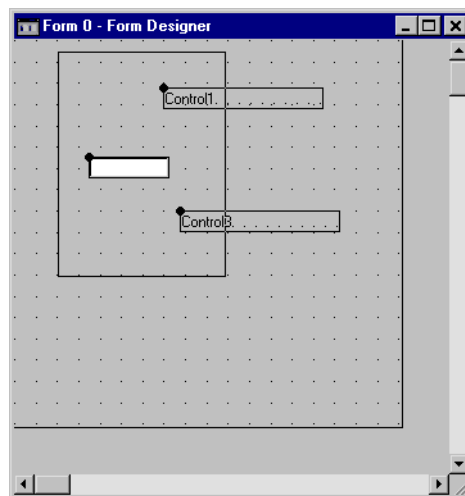
When one control has been selected, you can add other controls to the selection by holding down the CTRL key when clicking to select them. As you add controls, a box will appear around the complete selection and each control in the selection will be marked by a circle in the upper left corner of its own bounding box.



### Marquee Selection

Another way to make a multiple selection is by *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. When you drag the mouse, a rectangle will appear – a marquee.

As the rectangle expands, any control that it overlaps, completely or partly, will be selected; there will be a circle in the upper left corner of its bounding box.



Release the mouse button when you have finished selecting controls.

Controls can be added to the selection individually (as described above), 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. In the Tools menu, choose Options. The Marquee Full Selection option can be Yes or No. Only if the option is set to No does the marquee selection work as described above. If it is set to Yes, a control will be selected only if the marquee overlaps the control completely – not just partly.

.....

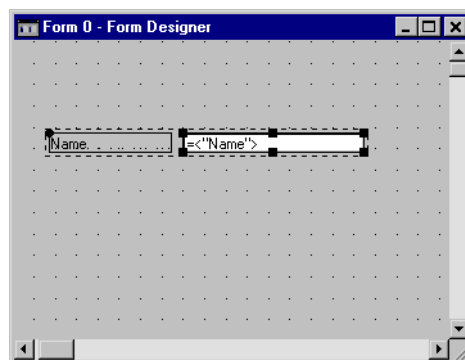
## Selection and Container Controls

When you select a container control, all the contained controls 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 will be 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 will be 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 will look 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 accurately than it is convenient to do freehand with the mouse. C/SIDE provides two methods for aligning controls easily and accurately:

- 1 You can turn on the option Snap to Grid in the Format menu. When you move a control as described above you will notice that it is not moving 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, select the controls as a multiple selection and click Format, Align. From the submenu that follows, select one of 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 – if you inadvertently choose to top align a columnar group, for example, the system will indeed do just this, placing 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 will be surrounded by a bounding box with sizing handles.
- 2 Place the mouse cursor on a sizing handle. The cursor will change into a sizing cursor.
- 3 Press the left mouse button and drag the control to the size you want. If Snap to Grid is on, the sizing takes place in fixed increments, in a way similar to the one described above for moving a control.

## 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 contained child of the container.

Beware that it is quite possible to reduce the size of the containing control so that a contained control seems to be outside the container. However, it is still considered part of the container. As no part of it is inside the control, however, it cannot be selected. The remedy for this is to enlarge the container so all contained controls are inside it.

## 6.4 SAVING, COMPILING AND RUNNING FORMS

After you have designed a form, you must save and compile it before it can be run. Normally, you will do this when you are done designing the form. However, you may want to save a form that is not yet finished and thus cannot be compiled, if the form is more complex than the forms described so far and contains C/AL code. You can also test-compile and test-run a form without closing or saving it.

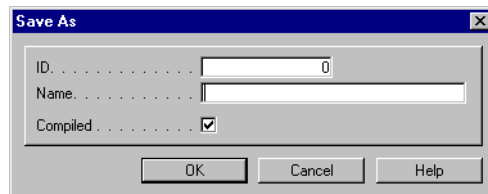
### Saving and Closing a Form

A designed form is closed when the **Form Designer** window is closed. You can close this window in the same ways that you can close any other window.

To save a form:

- 1 When you are closing a form, C/SIDE will ask whether you want the form to be saved. If it is a new form (a form that has not been saved before) you will have to assign 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 form properties, these values will be used, and you will not be prompted for ID and Name when you close the form.



- 2 The option field Compiled is by default set to TRUE (displayed as a check mark). If your form is not yet ready to be compiled, remove the check mark by clicking in the field.
- 3 Choose OK to save the form.

You can save a form without closing it by choosing Save or Save As... from the File menu. You can use Save As... to give a form a new name.

### Compiling a Form

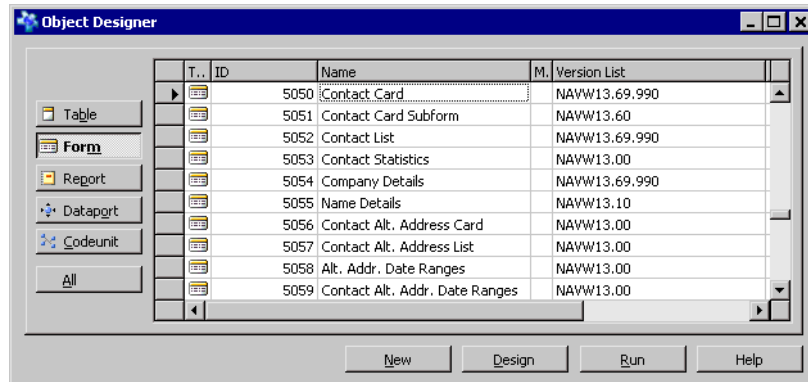
Forms, like other objects in C/SIDE, must be compiled before they can be run. As described above, you can choose to compile a form whenever you are saving it.

While you are designing a form, you may want to test-compile it to find possible errors. This is useful when the form contains C/AL code in triggers, as described in Chapter 9. You can test-compile a form during design by clicking Tools, Compile.

## Running a Form

In a finished application, your forms will be incorporated into menus or they will be called from other forms. However, while you are designing forms, you will often want to run them before they have been integrated into an application.

You can run a form from the list of forms in the **Object Designer** window by selecting it and clicking Run. Note that forms can also be run from inside the Form Designer by clicking File, Run.





## Chapter 7

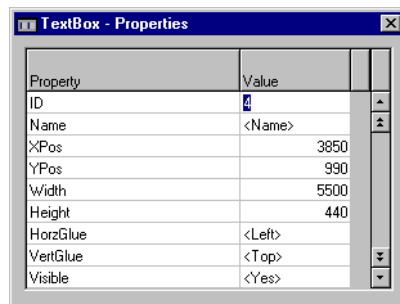
### Designing Forms

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
- Tools for Customizing Controls
- Setting Control Properties
- How to Use Controls in Applications

## 7.1 FORM AND CONTROL PROPERTIES

As described in Chapter 1 "C/SIDE Fundamentals", properties are a system-wide feature and every application object has some properties. Properties for forms and controls are edited by opening the Property Sheet in the Form Designer (by clicking View, Properties). As you select the form or a control in the form, the Property Sheet will display the properties for the selected object. The title bar of the Property Sheet window shows what kind of object (form, text box, label, and so forth) is currently selected. The first line of the sheet shows the ID if the object has one. Note that the Property Sheet can be scrolled vertically.



Each field on the Property Sheet contains a value that you can set by entering a value in the Value field on the Property Sheet. As soon as you leave the field (by pressing Enter or by using the arrow keys) the property is updated. If what you entered contains an error (for example, if you accidentally changed the ID of one control to be the same as that of another control), the update will not be accepted.

Default values are displayed in angle brackets (<>). If a property has a default, you can reset it to the default by deleting the current value and then moving out of the field. Notice that some properties do not have defaults – mainly those that describe the position of the control within the form. These properties are constantly updated by C/SIDE when the control is moved.

### How Properties Are Inherited

Controls that have a direct relationship to table fields will inherit the settings of those properties that are common to the field and the control. You can still change the settings of these properties for the control, but you cannot overrule the settings of certain field properties, namely those 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 – as 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 the field level or at the control level. The advantage of using the lowest level (the field level) is that whenever the field is used as the data source of a control, these settings will be used as defaults. This ensures consistency.

## Form Properties

The table below briefly describes some of the more important form properties. All properties are described in detail in the online C/SIDE Reference Guide. You can get context-sensitive Help for a property by opening the Property Sheet for a form, 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 C/SIDE dealer will inform you about the numbers you can use.
Name	give the form a descriptive name. The name does not have to be unique, but you should give your forms unique names anyway, as they will be a lot easier to identify and find by name than by ID.
Minimizable	specify whether the user can minimize the form window.
Maximizable	specify whether the user can maximize the form window.
Sizeable	specify whether the user can resize the form window.
SavePosAndSize	specify whether information about the user-made changes to the size and placement of a form window will be saved. If it is set to Yes, this information will be saved, and the next time the window is opened, these values will be used. Otherwise, the designed values will be used.
Editable	specify whether the user is allowed to edit controls in the form. If it is set to No, no controls may be edited, even when their individual Editable 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 will have specified the table when you first created the form. If you have created a form without an underlying table, however, you can enter a table name here to bind the form to a table.
SourceTableView	create a view (what the user 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 table below briefly describes those properties that are common to several types of controls. All properties are described in detail in the online Reference Guide. You can get context-sensitive Help for a property by opening the Property Sheet for a form, selecting a control, placing the cursor on a property and pressing F1.

Property Name	Use This Property to...
ID	set the numeric ID of the control. The system assigns a sequential number by default. If, however, 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 will be 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 will be resized when the form is resized.
Visible	specify whether the control will be 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.

## 7.2 TYPES OF CONTROLS

This section provides a brief overview of all controls that can be added to a form. The list below is structured according to the broad categories into which controls can be grouped.

### Static controls

Static controls are controls that cannot change contents at run time.

**Label** A label is used for displaying text, most commonly for displaying 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 bitmap picture.

**Shape** A shape is a graphical element (line, circle, rectangle).

### Data controls

Data controls are controls that can display the value of a C/AL expression, for example, the value of a table field or a variable (perhaps the simplest expression is just the name of a table field or a variable) or of a "real" expression. The valid combinations of data control and data type are as follows

Control	Valid data types
Check Box	Booleans and BLOBs
Option Button	All, except BLOBs
Text Box	All
Picture Box	Boolean, option, integer and bitmap BLOBs
Indicator	Integer, decimal, date, time

Data controls *must* have a relation to data, defined as their *SourceExpr* property.

### Containers

Container controls are used for grouping other controls. Some properties of the container overrule the same property in the contained controls: if the container is not editable, no single contained control can be edited (even if it individually has the *Editable* property set to TRUE).

**Frame** A frame is simply a rectangle into which other controls can be "dropped". While you are designing, the frame and its contained controls can be moved together, and a frame can have different border styles and colors.

**Tab Control** A tab control can be thought of as a kind of book with several pages, or as several frames, where only one is visible at a time. The user can switch between pages by clicking on tabs with captions.

## Data Containers

**Table Box** A table box is a container, too, but a special kind. It contains 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

**Command Button** A command button is not related to data — it performs an action when it is "pushed", that is, when it is clicked, or when ENTER or the spacebar is pressed while the button has 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: it can perform an action when you click it.

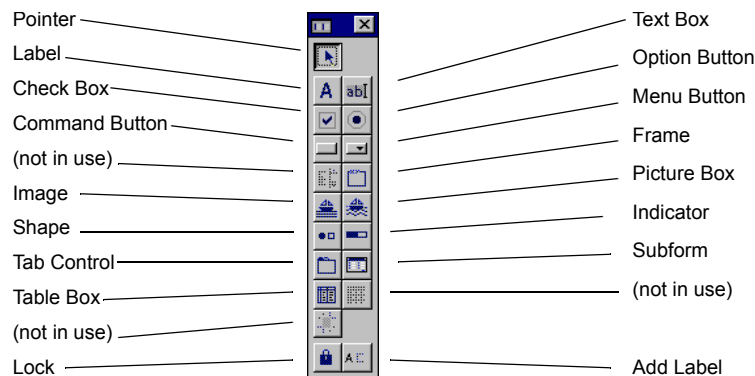
**Subform** A subform control is used to display a second form in a control on a form (a main form), in order to show data from two different tables. For example, the main form could be a card form and show records from a customer table, while the subform could be a tabular form and show details about purchases the customer has made.

## 7.3 ADDING CONTROLS

This section gives a few examples of how to add controls without using the form wizards.

### The Toolbox

You use the Toolbox to insert controls. The Toolbox is opened by choosing Toolbox from the View menu. You select a specific tool by clicking the corresponding icon.



Note that some of these tools are not implemented in the present version of C/SIDE, but that the icons are already present—they will, however, always appear disabled.

When you click the Pointer tool, the state changes from insertion to selection. You can use this if, for example, you change your mind about inserting a control.

The Lock tool locks the current control selection. Normally, after you have inserted a control, you have to select the type of control again before inserting the next control. You can continue inserting controls of the same type without having to select the tool again and again by turning Lock on (it is a toggle).

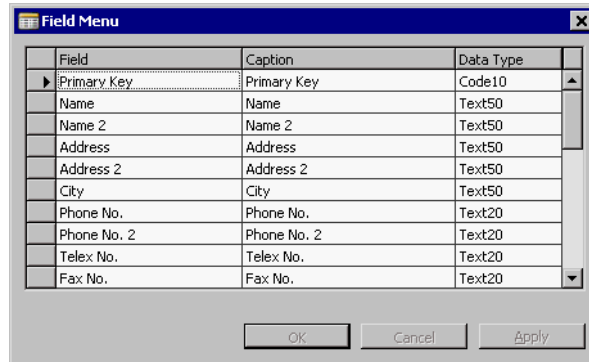
If Add Label is on (it is a toggle), all controls will have a label when you insert them.

### Adding a Text Box

If text boxes are related to database table fields, the easiest way to add them to a form is to use the *Field Menu*. The Field Menu is a list of all fields in the table that is defined as the SourceTable for the form. You can open the Field Menu in the Form Designer by clicking View, Field Menu. A text box that has a specific field in the table as its SourceExpr (that is, has this field as its underlying table field) can be added to the form as described in the following section.

To add a text box:

- 1 Open the form in the Form Designer.
- 2 Click View, Field Menu to open the **Field Menu** window:



- 3 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 18, "Multilanguage Functionality".



When you move the cursor into the design area, it will change into the Control Insertion cursor.

You need to activate the **Form Designer** window, for example, by a mouse click, before the cursor will change into the Control Insertion cursor.

- 4 Click in the design area for each selected field to insert a text box at the cursor position.

If you selected more than one field, the text boxes will be inserted and aligned in a column below the mouse position.

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, all properties that are both field properties and text box properties have the value of the field property in the underlying table as a 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 naming and properties are consistent.

Beware that if the data type is anything but boolean, a text box will be created automatically. If the data type is in fact boolean, a check box will be 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 way 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 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.

The subsection Changing the Properties of a Control on page 125 explains how you can bind the text box to a table field and add a label, and the subsection Displaying a Calculated Value on page 129 tells how you can use the 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 a descriptive text on the form – it could be instructions about using the form or other information that is static and not related to any database table field.

To add a label:

- 1 Open the form in the Form Designer.
- 2 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.
- 5 As the label is not part of a control branch, it will be given a default name and caption (like *Control4*). You can change the name and the caption on the Property Sheet for the label (see Changing the Properties of a Control on page 125).

## 7.4 TOOLS FOR CUSTOMIZING CONTROLS

In addition to the Property Sheet itself, there are two special tools available for setting various properties of controls. They are: the Color tool, for selecting color properties and border styles, and the Font tool, for setting font properties.

### Using the Color Tool

To start using the Color tool, click View, Color. The tool looks like this:



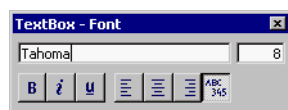
When a control that has color properties is selected, you can pick colors for foreground (text), background and border by clicking in the palette. The corresponding properties are ForeColor, BackColor and BorderColor.

The check boxes Background and Border are used to toggle the display of background color and display of the border on and off. The corresponding properties are BackTransparent and Border (if these options are *off*, a background or border color will not have any effect).

If the control has a border, the nine buttons at the bottom can be used to select border style and border width.

### Using the Font Tool

To start using the Font tool, click View, Font. The tool looks like this:



When a control that can display text is selected, you can set the font properties by using the tool. 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* meaning that text is left aligned and numbers are right aligned).

## 7.5 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 will often need to change some properties of these controls. Even if you did use a wizard, you may want to change some 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 property. The SourceExpr – source expression – is a C/AL expression. It can just be the name of a table field, or it can be a more 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, Name and Caption will be set by default to the name of the table field (unless the table field has a Caption: in this 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 will be set to this value as a default (you will see that it is displayed in angle brackets). If you change the Caption property of the text box again, the Caption property of the label will also be changed again.
- On the other hand, if you change the Caption property of the label directly, you will notice that the value that you enter is displayed without angle brackets, signifying that it is no longer a default value. This means that if you change the property of the text box, the value here will no longer be updated.

### Changing an Unbound Control into a Bound Control

An unbound text box – or other data control – can be changed into a bound control quite easily.

All it takes is to change the SourceExpr into what 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 for a bound control, that is, they default to the name of the table field.

This will not automatically add a label to the text box, but you can add one, as described below.

### Adding a Label to a Text Box

If you have created a bound text box by changing the SourceExpr for an unbound text box, the bound text box will not automatically get an attached label. You can add a label by adding a label control to the form and then change the ParentControl property of the label from the default (undefined) to the ID of the text box (which you can see on the first line of the Property Sheet for the text box).

The control branch resulting from this operation can be selected and moved as described in section 6.3, Selecting, Moving and Adjusting Controls.

### Display Properties

Controls that you add to a form – either by using a wizard or manually – will 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 provide for all 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 properties that control the display of numbers. Refer to the online C/SIDE Reference Guide for full descriptions.

**DecimalPlaces** This property (whose setting specifies both the minimum and maximum allowed values) determines how many decimals are displayed and how many can and must be entered. A typical situation where this property would be used is when amounts are stored in the database with 5 decimal places for higher precision, while you want the user to see only the customary number of decimal places for the currency in question, for example, 2. The table field would then have the DecimalPlaces property set to 2:5, while the DecimalPlaces property of the text box should be set to 2:2.

**BlankNumbers** You can choose from an option list whether a range of numbers will be displayed or they will be blanked.

**BlankZero** The default is No. If you change it to Yes, zero values *and* booleans that would have been displayed as a No will be blanked out.

**Divisor** The default is Undefined. If a number is entered, numeric values will be divided by this number when they are displayed. Any remainder will be 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 properties that control formatting of data. Refer to the online C/SIDE Reference Guide for full descriptions.

**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. Finally, you can build your own formats to serve special needs.

**HorzAlign and VertAlign** These properties define how data in a text box or a caption on a label will be aligned horizontally and vertically, respectively.

**MultiLine** If this property is set to Yes, labels and text boxes can have 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. See the subsection Displaying More Than One Line of Text on page 129 for details.

**PadChar** This property specifies the character to 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 there will be leading dots before the data. The dots are placed according to the horizontal alignment of the data: if left aligned, the dots are placed to the right, if right aligned, the dots are placed to the left – and if centered, there will be dots both before and after the data.

If this property is set to Yes, the setting of PadChar will be overruled.

### Properties That Control Input

This group of properties can be used to control user input, that is, restrict user input to certain values or a certain length.

**Numeric** This property restricts input to numeric values only if it is set to Yes.

**MinValue, MaxValue** Sets a minimum or maximum value that the user can enter.

**ValuesAllowed** Here you can specify the values that the user is allowed to enter. Enter the values separated by semicolons, like `1;7;4711` or `a;b;c`.

**CharAllowed** Here you can enter characters that the user can enter. You can enter a range, for example, `AZ`, to limit entry to uppercase characters only, or several ranges, for example, `amot`, specifying two ranges: `a` to `m` and `o` to `t`.

**NotBlank** If this is set to Yes, then an entry consisting of nothing, one blank or several blanks (spaces) will not be accepted – though a blank can be part of string that contains other characters.

**MaxLength** The maximum number of characters that can be entered in a text box.

**AutoEnter** If this is set to Yes, the system will accept a user 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, the user does not have to press ENTER.

**PasswordText** If this is set to Yes, user input will not be displayed, but shown as asterisks (`*****`).

## Assisting the User

The **ToolTip** property allows you to assist the user by displaying text that describes a control. You can also control the captions used in the title bar of a form window.

**ToolTip** If you enter a text here, it will be displayed in a small pop-up window whenever the mouse cursor rests on 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** As mentioned in the table of form properties above, you can control the label that is displayed on the caption bar of the form window by using the *Caption* property. This is a static caption, usually the name of the underlying table. By using **DataCaptionField** (either at table level or at form level), you can select fields from the record whose contents are displayed (and updated) in the caption bar as the user pages 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 the user selects a new record or the present record is changed. The online C/SIDE Reference Guide contains further explanations.

### Note

.....  
If the user-selectable option Status Bar (click View, Options) is set to Yes, the caption of text boxes and check boxes will be displayed in the status bar together with the current data contents of the control (if any) when the control gets the focus.  
.....

## 7.6 HOW TO USE CONTROLS IN 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 and enlarge it vertically by resizing.
- 3 Open the Property Sheet for the text box and set the MultiLine option 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 Move the cursor into the design area.
- 4 Click to add the control.
- 5 Open the Property Sheet 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)
```

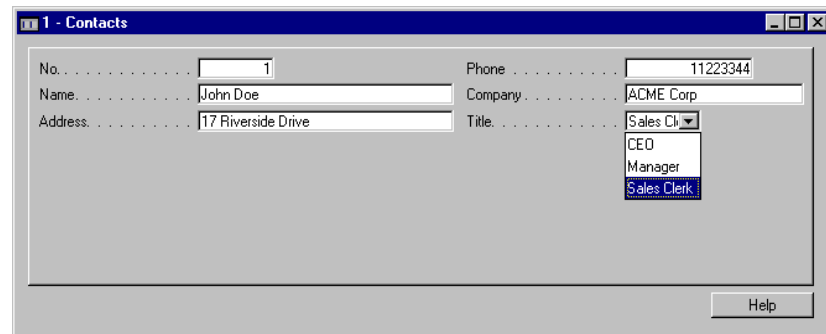
### Presenting a Set of Options

A recurring task in application programming is to present the user of the application 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. The field or variable that is the *SourceExpr* of the control must have type *Option*, and the options must have been entered as the *OptionString* property of the field or variable.

In C/SIDE, you can present these options in several ways. The following sections show two different approaches.

### Creating a Drop-Down List of Options

The list of options can be presented as a drop-down text box, as in the picture below:



To create a drop-down option list:

- 1 Open the form in the Form Designer.
- 2 If the option text box is based on a table field, open the Field Menu and highlight the field. Otherwise, proceed to create an unbound text box.
- 3 Select the text box tool; then click in the design area to create the text box (if the text box is based on a field that you have selected on the Field Menu, just click in the design area).
- 4 If the text box is unbound, bind it to the variable now by entering the name of the variable as the *SourceExpr* of the text box.



When you run the form, the text box will have the AssistButton ▼ attached, and you will be able to open the list by clicking this AssistButton.

#### Note

.....  
 You can enter only options that have been defined on the Property Sheet of the field or variable. The first option will be displayed in the text box. If the OptionString property has a blank as the first option, the text box will accordingly be blank. This does not mean that options that are not in the OptionString can be entered.

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 will not be any different from 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.

To add an option button group:

- 1 The OptionString property of the field or variable must be defined, as described above.
- 2 Open the form in the Form Designer.
- 3 Add an option button for each option in the OptionString of the field or variable (the option buttons must be added as unbound controls).
- 4 Enter the field or variable as SourceExpr in the Property Sheet for each button.
- 5 Enter one of the options from the OptionString as the OptionValue property for each button.

Each button has the OptionValue 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

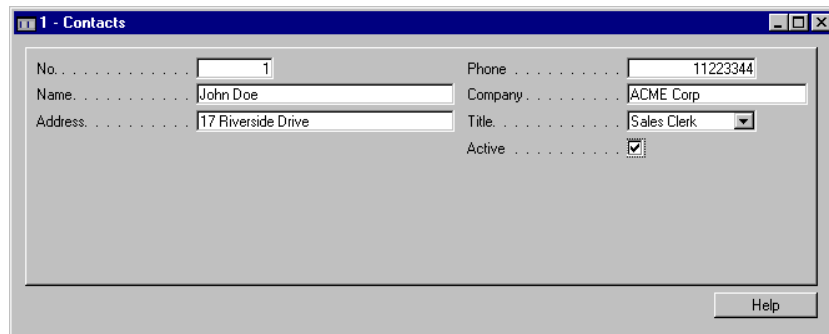
an option by clicking on the button, any previously-chosen button will be marked as *not chosen*.

#### EXAMPLE

In the previous illustration, the option button group has been embellished by adding two frames: the group is contained by a frame with a raised border and no caption. The other frame is actually the "Title" caption – with the TopLineOnly property set to Yes, and Caption property set to Title.

### 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 will be displayed as a check mark, while No will be displayed as a blank.



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

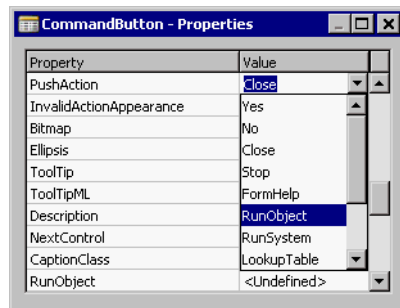
Command buttons are useful for a number of purposes. If you have used a wizard to create forms, you will have noticed that a Help button has been added to all forms. Other common uses are Yes and No buttons in contexts where the user must decide whether a certain task will be performed or not. Still 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 Property Sheet 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. Choose RunObject.
- 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

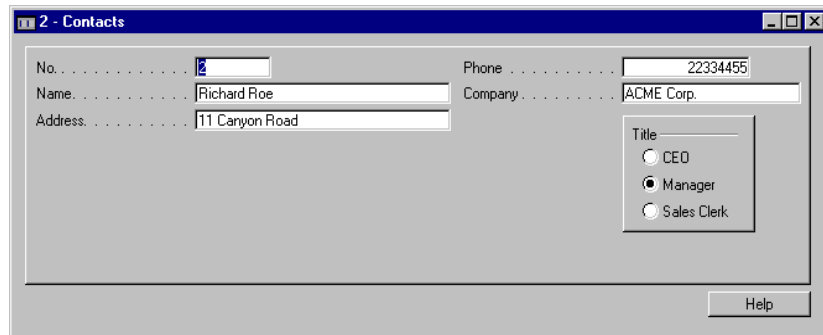
Not all settings of the PushAction property require additional information. Some do, such as RunSystem, while others, such as Yes or No, do not.

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 8, "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 contained controls will be 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 Choose the Frame tool; then click and drag in the design area to create a frame.
- 3 Create the controls you want 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 will be inhibiting display of the caption 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 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 will no longer be considered to be contained by the frame.

If a container is resized and thereby overlaps existing controls completely, these controls will be contained.

If a frame is deleted, all controls contained by it are deleted. See Sizing and Resizing Controls on page 111 for details on resizing container controls.

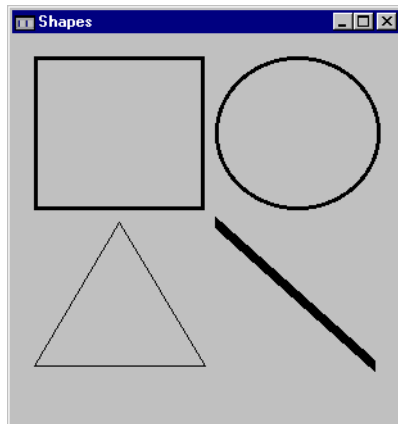
## Adding Shapes and Pictures

You can add shapes (graphical elements) and bitmap pictures to forms in order to provide information (pictures of products, for example), in order 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 the color of the

lines that the shapes are composed of by changing the `BorderWidth` and `BorderColor` properties.

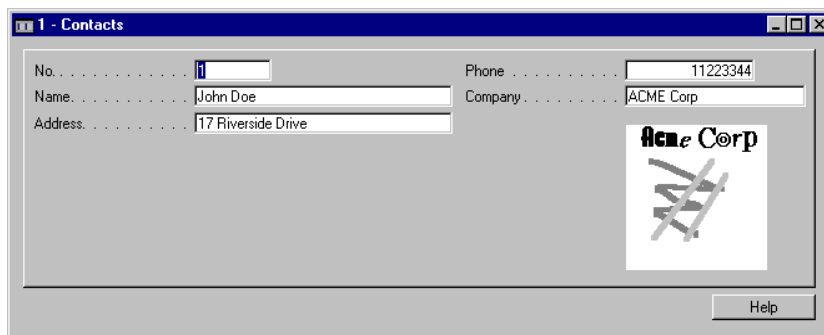


To create a shape:

- 1 Open the form in the Form Designer.
- 2 Choose the Shape tool from the Toolbox
- 3 Click and drag in the design area to add a shape of the desired size.
- 4 Choose a `ShapeStyle` and appropriate width and color in the Property Sheet for the shape.

### Adding a Static Picture as an Image

The simple way of adding a bitmap picture is to add it as a control of type image.



To add an image:

- 1 Open the form in the Form Designer.
- 2 Select the Image tool from the Toolbox.
- 3 Click and drag in the design area to create an image control.
- 4 Open the Property Sheet for the image control, and enter the file name of the bitmap in the `Bitmap` property.

Beware that the bitmap is not referenced but actually imported. This has the advantage that you don't need any external files for your application. On the other hand, if you make any changes to the bitmap during application development, you must update the imported copy by opening the Property Sheet, selecting the Value field of the Bitmap property and pressing F2. This will cause a reevaluation of the field, thus forcing the bitmap to be imported again. The bitmap can be up to 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 provides some advanced possibilities. While the image control is static, the picture box control is dynamic: provided 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).

A picture box provides another advantage: 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** Select the Picture Box tool from the Toolbox.
- 3** Click and drag in the design area to create a picture box control.
- 4** Open the Property Sheet for the picture box control.

To create a list of bitmaps from which one can be selected at run time for display, follow these steps:

- 1** Enter a comma-separated list of the file names of the bitmaps you want to use. The system provides a series of standard bitmaps that can be chosen by entering a number – see the online C/SIDE Reference Guide entry on Bitmap for details.
- 2** The value of SourceExpr determines which bitmap will be chosen by the system: the first in the list has number 0, the second number 1, and so forth. If SourceExpr evaluates to a value outside the range of bitmaps, no bitmap will be displayed.

To display a picture stored in a BLOB field, do this:

Enter the field name of the BLOB field as the SourceExpr property. 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 all have the capability of displaying a bitmap picture instead of – or in combination with – a caption.

They all have a property called Bitmap. Here you can enter the filename of a bitmap. The maximum size of the bitmap is 32 Kb, and it is actually imported, not referenced (thus if you change the original bitmap, you will have to reimport it).

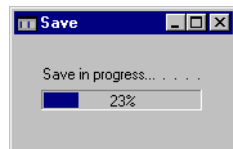
They also have a property called `BitmapPos`, where you can select the alignment of the bitmap within the control. This is especially useful when you are combining a caption and a bitmap: then you could right align the caption and left align the bitmap, like the command button in the picture below.



Using a bitmap on a button or a check box can in some situations make the user interface more intuitive: a well-chosen picture may be easier to remember and use than a label.

### Using an Indicator to Display Values

The indicator control provides a way of displaying values graphically, as an analog gauge. A minimum and a maximum value must be defined, to make it possible for the system to 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 “23%” shown in the picture above will be displayed – otherwise, it will 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 Property Sheet 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.
- 2 Select the table box tool, and click and drag in the design area to create a table box.
- 3 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.
- 4 Open the Field Menu.
- 5 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 8

### 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

8.1 MAIN FORMS AND SUBFORMS

As described in Chapter 2, Designing a C/SIDE Application, 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, suppose 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, for example 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, another, a lines table, with the information about each item. There is a one-to-many relationship between the tables.

However, the users of the application need to view information from both tables at the same time: the header information together with the lines, like this

T...	No.	Description	Location ...	Quantity	Reserve...	Unit of M...	Unit Pric...
I...	1980-5	MOSCOW Swivel Chair, red	GREEN	6		PCS	190,036

Although this looks like a normal form it is, in fact, two forms.

The main form is the one side of the one-to-many relationship; in the example, it is based on the **Sales Order Header** table. The subform is the many side of the relationship; in the example, it is based on the **Sales Order Line** table. When the user selects a sales order header in the main form, the subform is updated to display only sales order lines pertaining to this sales order header. There is therefore a link between the main form and the subform that keeps the information synchronized.

Designing the Main Form

There are no special procedures involved in designing a form that will be used as the main form in a main form/subform relationship: it is designed as any other form.

In order to add a subform, you will add a subform control. The subform control establishes the link between the main form and the subform, but it is not a form in itself. 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 below. If you are going to create a new form to use as a subform, it may be more convenient to create the subform first. How to do 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 Select the Subform tool, and click and drag in the design area to create the subform control.
- 3 Open the Property Sheet for the subform control.
- 4 Enter the name of the form that you want to use as a subform as the SubFormID property (or use the lookup button to choose from a list of all forms).
- 5 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 assist you (click the AssistButton ... to open the assist-edit window). Choose the field name from the *many* side of the relationship (the subform table) as the Field. Then choose FIELD as the Type of the relationship. Finally, choose the field from the one side of the relation (the main form table) as the Value.
- 6 In the SubFormView property, you can specify the key, sort order and table filter you want the system to apply to the table when it is displayed in the subform (it is not mandatory to enter anything).

#### EXAMPLE

In the SubFormLink property, you can choose other types of link than FIELD. 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 (as, for example, 10|30..40).

### Designing the Subform

There are no special requirements for a subform, that is, it is exactly like any other form in the system. However, the form is going to be used to display the many side of a one-to-many relationship, and not all forms are equally useful for this purpose.

The typical choice is a tabular form, that is, a form with a table box. See Creating a Table Box on page 138. 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. In this way, the subform and the table box will be resized when the main form is resized.

### Hints and Advice

Even if creating a form with a subform is not different from creating controls on forms in general, you may have to perform some experiments before you find the best way to do it.

Here are some hints and advice to help you along:

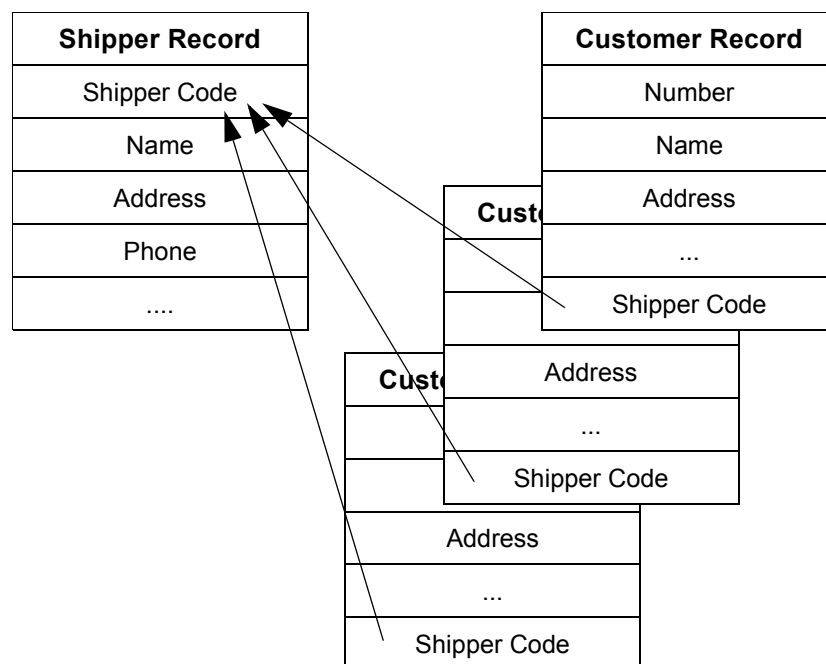
- It can be difficult to get the sizing of the subform control on the main form and the size of the subform itself right. You should finish the design of the subform first. Get the values for width and height of the form from the Property Sheet. Then, in the main form, click and drag a subform control of any size. In the Property Sheet, 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 will look better if you let the table box completely fill out the form vertically – this way there won't be extra space around the table inside the subform control – and set the HorzGlue and VertGlue properties to Both.
- If the subform is a tabular form, you should size the form to show only a few records at a time. Then, in the main form, set the VertGlue and HorzGlue properties of the subform control to Both. The user can resize the main form vertically and horizontally, and the subform will be resized along with it: more records and fields will be displayed.

## 8.2 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; the subform was bound to the *many* side.

Suppose, instead, that you are designing a form that is bound to a table containing information about customers. Some of this information is unique for each customer, while other information is not. The names and addresses of the customers are unique, but suppose 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 addresses of these shippers in the customer records.

Instead, you would create a **Shipper** table, and use a **Shipper Code** field to create a link between this table and the **Customer** table, storing only the shipper code in each customer record, and storing all other information about the shippers in the Shipper table.



This is, in fact, the many side of a one-to-many relationship: while each customer can be associated with only one shipper, a shipper can be associated with many different customers. A main form/subform is not applicable here (it would be, though, if you were to design a form to display information about the shippers. The subform could then display a list of customers that use each shipper).

There are two things you must consider when creating the **Customer** form and table:

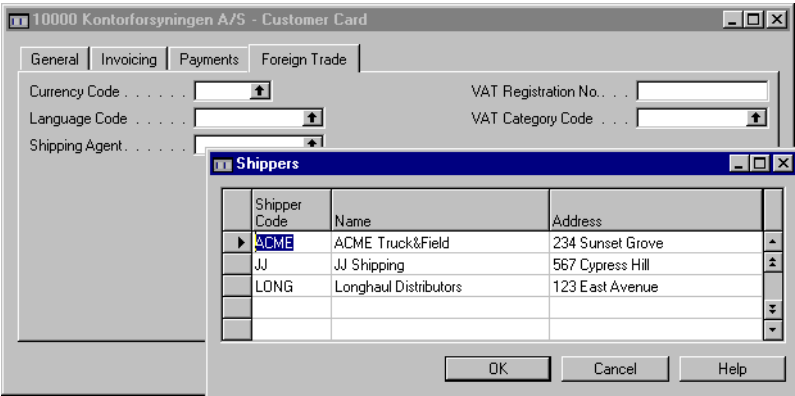
- Do you want to provide the user with an easy way of entering the shipper code?
- Do you want to validate the **Shipper Code** field in the **Customer** table against the **Shipper** table? That is, do you want the system to verify that the contents of the **Customer** table field are present in the **Shipper** table?

If you do not establish a relationship to the **Shipper** table, the users will have to memorize the shipper codes, and they may 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 the user can press F6 or click a lookup button (↕) and select the 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 by defining the relationship, either at the table level, as a property of the **Shipper Code** field in the **Customer** table, or at the form level, as a property of the text box displaying the shipper code on the **Customer** (main) form.

If you want to make certain that the user does not enter non-existent shipper codes into the **Customer** table, the system can validate the entries against the **Shipper** table. The `ValidateTableRelation` property, either of the field (at table level) or of the text box (at form level), governs whether entries are required to exist in the **Shipper** table.

Apart from simply asserting 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 at all in the customer's country). To do this, you will 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 field looks like this when the lookup function has been activated:



Defining the Table Relation

As mentioned above, the relationship to a table can be defined in two different places. In both places, as part of a table description or as part of a form description, the relationship is defined in the `TableRelation` property of the field or control. For the application user, there will be no functional difference between a table relationship defined at the table level and a table relationship defined at the form level. There is a difference, though, when you are designing an application. If the relationship is defined at the table level, all text boxes in 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 the function by setting the `Lookup` property of the text box explicitly to `No`.

To define a table relationship:

- 1 Open the Property Sheet for the field or the control.
- 2 In the Value field of the TableRelation property, click the assist-edit button.
- 3 In the assist-edit window, enter the name of the table to lookup into in the **Table** field (or choose from the list that appears when you click the lookup button).
- 4 In the **Field** field, enter the name of the field in the table (or choose from the lookup 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, you can, for example, lookup to 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 be validated against the contents of a field in a related table quite easily. 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. In this way you can bypass the default lookup function completely and write your own.

The rules for determining which lookup function is performed are these: 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 will have to 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 defined that will be used for looking up into the table, by setting the LookUpFormID table property – or a form can be defined by setting the LookUpFormID of the control for which the lookup is provided. If both properties are set, the form defined as a control property will be used.

Pay attention to the fact that if no lookup form is defined (either at table level or at form level), then although the text box will have the lookup button (↗) attached, a lookup will not be performed when the button is clicked.

If you are writing your own lookup function in the OnLookup trigger, you will have to explicitly run a form by using the RUNMODAL C/AL function.

**Hint**

.....  
If you always design a basic tabular form (fast and easy, using the wizard) 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 on 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 it is set to Yes, the lookup button will be permanently displayed; otherwise, it will be 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 provide a lookup to the same table (the source table of the form, that is). This is intended to provide the user with an easy way of selecting a record to work with. In effect, the lookup provides a list of all records in the table; the user can select a record from the list, which will then become the current record.

A lookup form must be defined, either at table or at form level, just as when the lookup is to another table. You cannot set conditions and filters, however (as you can when the lookup is to another table). The default behavior is to display all records in the table. If you need to change this, you will have to 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 Lookup to Yes will be overruled. On the other hand, if Lookup is (explicitly) set to No (as opposed to its default value <No>), no lookups, including to related tables, will be 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.



## 8.3 DRILLING DOWN TO THE UNDERLYING TRANSACTIONS

FlowFields were introduced in Chapter 3, Table Fundamentals. When a text box is based on a FlowField, you will see that a *drill-down* button (↕) automatically is attached to the text box. When the user clicks this button (or presses SHIFT F6), the transactions that the system used to calculate the value of the FlowField will be displayed.

In the first picture below, 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 next picture shows the form that is displayed when that particular drill-down is performed – a detailed list of the transactions:

No.	Name	I...	A...	Totaling	G...	G...	G...	Net Change	Balance
7495	Total Cost of Resources	I...	E...	7405.7495					
7620	Job Costs	I...	P...						
7995	Total Cost	I...	E...	7100.7995				3.802.149,33	3.802.149,33
8000	Operating Expenses	I...	B...						
8100	Building Maintenance E...	I...	B...						
8110	Cleaning	I...	P...		P...	N...	M...	26.665,75	26.665,75
8120	Electricity and Heating	I...	P...		P...	N...	M...	35.528,08	35.528,08
8130	Repairs and Maintenance	I...	P...		P...	N...	M...	234.008,75	234.008,75
8190	Total Bldg. Maint. Expe...	I...	E...	8100.8190				296.202,58	296.202,58
8200	Administrative Expenses	I...	B...						

Posting D...	D...	Documen...	G/L Acc...	Description	G...	G...	G...	Amount	B...	Bal.
01-01-00	2000-1		8130	Entries, January 2000	P...	N...	M...	2.110,93	G...	
01-01-00	2000-1		8130	Entries, January 2000	P...	N...	M...	3.166,39	G...	
01-01-00	2000-1		8130	Entries, January 2000	P...	N...	M...	5.277,32	G...	
01-01-00	2000-1		8130	Entries, January 2000	P...	N...	M...	2.234,74	G...	
01-01-00	2000-1		8130	Entries, January 2000	P...	N...	M...	3.352,10	G...	
01-01-00	2000-1		8130	Entries, January 2000	P...	N...	M...	5.586,84	G...	
01-02-00	2000-2		8130	Entries, February 2000	P...	N...	M...	1.977,24	G...	
01-02-00	2000-2		8130	Entries, February 2000	P...	N...	M...	2.965,86	G...	
01-02-00	2000-2		8130	Entries, February 2000	P...	N...	M...	4.943,10	G...	
01-02-00	2000-2		8130	Entries, February 2000	P...	N...	M...	2.093,20	G...	

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 (as for lookups – see Defining a Lookup Form on page 145), either at the table level or at the form level.

Drill-downs resemble lookups in many ways, and with them you can do most of the things that you can do with lookups – one exception being that drill-downs pertain only to FlowFields, which have to be defined when the table is designed.

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 the form level.
- You can decide whether the drill-down button should be displayed permanently or only when the text box has the focus, by setting the PermanentAssist property of the text box. Yes means that the button will always be displayed, and No means that the button will be displayed only when the text box has the focus.
- You can change the drill-down behavior altogether by writing C/AL code in the OnDrillDown trigger of the table field or the control. In this case you have to run a form explicitly from your trigger code, as the system does not perform any part of the default drill-down and does not display a form automatically.

## 8.4 LAUNCHING ANOTHER FORM

When you provide a lookup function for selecting values in a related table, you will typically display only a subset of the fields in the lookup table.

In some situations, however, it would be convenient to be able to update other fields than those displayed on the lookup form in the related table without closing the current form. Suppose a user is taking orders by phone. It is convenient to use a lookup function on the sales order forms to find the customer numbers as customers call in. But what if a customer calls in to order something – and mentions that he has moved to a new address? It would be 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 is to provide a way 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 to select the correct record from the **Customer** table whenever the user "pushes" the control. You can use command buttons, menu items, check boxes or option buttons.

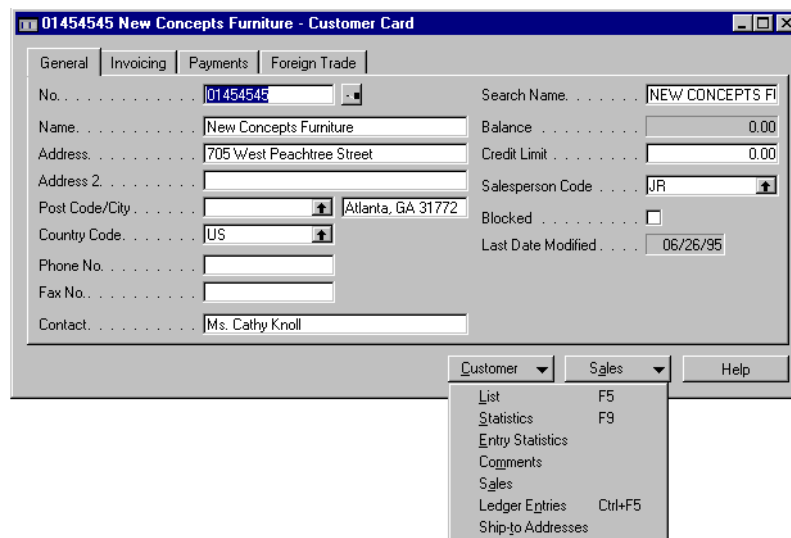
To add a command button that launches a form, follow these steps:

- 1 Add a command button (the procedure is described in Chapter 7).
- 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. As you can use RunObject to run any object, you have to specify the type of object (Form, Codeunit, and so forth). You can choose the object from the lookup list that is provided (in this case, the type of the object is inserted automatically).
- 4 Set the RunFormLink property to establish the link to the form you want to launch. Use assist-edit (click ... ) to create the expression. First, select a field from the table underlying the form you are going to launch. Choose FIELD as the type of the relationship. Finally, as the Value parameter, select the field in the table underlying the current form that must match the value in the other table.

## 8.5 DESIGNING MENU BUTTONS

While command buttons are a convenient way of adding functionality to forms, an excess of buttons will clutter the forms and impair their visual design. If you need to use many command buttons, you should consider creating menu buttons instead.

When a menu button is pushed, a menu is opened:



Each line in a menu is called a *menu item*. A menu item can:

- Perform an action when clicked. This can be an action from the same set of actions as command buttons (see the online C/SIDE Reference Guide for a list), or it can be an action written in C/AL, as menu items have OnPush triggers just 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.

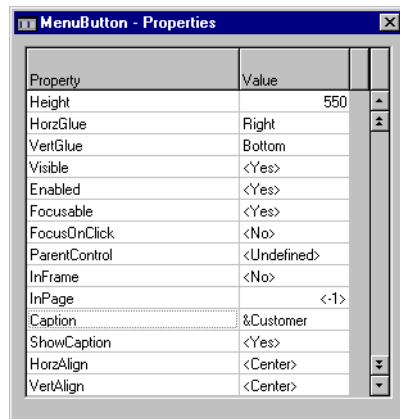
A menu is created in two steps. First you add a menu button to your form. This part is exactly the same procedure as adding a command button. Then you open the Menu Designer for the menu button and create the menu items.

### Adding a Menu Button to a Form

To add a menu button:

- 1 Open the form in the Form Designer.
- 2 Choose the Menu Button tool, then click in the design area to add the menu button.

- 3 Select the menu button and open the Property Sheet for the menu button. As a menu button does not have a relation to data – field or variable – Name and Caption are set to default values (like Control7). Change the Caption to an appropriate text. If the text contains an ampersand (&), the system interprets the following letter as an access key.

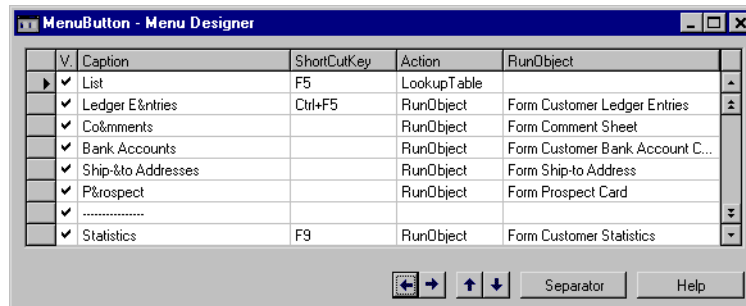


### Adding a Menu Line to a Menu

When you have created and modified a menu button as described above, you can add lines to the menu that is displayed when the button is pushed.

To add a menu line:

- 1 Select the button and open the Menu Designer while the button is selected. (Choose View, Menu Items.)



- 2 The first field, Visible, is by default set to Yes. Leave it like this.
- 3 Add lines by filling out the **Caption** field. If you do not create an access key yourself (by embedding an ampersand in the Caption text), the system will automatically use the first letter of each Caption as an access key. If you add menu lines where some captions start with the same letter, you must set the access keys yourself to avoid overloading certain ones.

- 4 If you want, you can define a shortcut key (accelerator key) by entering the name of the key in the **ShortcutKey** field. Keys are entered as follows:

Key	Entered as
Function keys	F1, F2, F3, ...
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

An accelerator key is active as long as the focus is on the form that the menu button is a child of. Beware of accidentally overloading some key combinations so that they perform different actions when different forms have the focus – this could confuse the user. Also beware of using accelerator keys that the system already uses.

- 5 Enter the action for the menu item in the **Action** field. You can use the drop-down list that is available to choose from among the same actions as for a command button. You can also write C/AL code in the OnPush trigger of the menu item.
- 6 If you have chosen RunObject, you can define the object (form, report, codeunit) in the Object field. There is a lookup function available to help you select the object. For other parametrized actions (for example, RunSystem) you have to set the parameters in the Property Sheet of the menu item (see below for details).

### Adding Other Menu Items

In addition to lines that perform actions, menus can contain separators and lines that are submenus, that is, lines that open up another menu when you click them.

**Separators** A separator is a horizontal line in a menu that cannot be selected or perform any action. It helps you group items on a menu.

To add a separator to a menu, click the Separator button in the Menu Designer. The separator will be inserted after the currently selected line.

**Submenus and Menu Levels** Menu items can be nested, that is, when you click a line on a menu, another menu can open.

Submenus are defined in the Menu Designer. When an item is selected, you can indent it by clicking the right-arrow button. An indented item becomes a menu item on a submenu. If you open the Property Sheet for a menu item, you will see that when first created, menu items have the MenuLevel property set to a default value of zero. As items are indented, the MenuLevel is set to 1, 2, 3 and so forth – 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 you must follow when creating submenus:

- If there are any items at all in a menu, there must be at least one item with MenuLevel 0 (zero).
- Each MenuLevel can be at most one 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 becomes a submenu, and the item with MenuLevel 1 becomes an item on this submenu. A menu line that is 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 (less indentation), menu items will from then on become items in the previous menu at the level indicated by the MenuLevel.
- Separators cannot be submenus and separators have to separate items at the same level, that is, you cannot put a separator as the first or last (or only) item in a menu or submenu.

### Check Marks on Menu Items

Menus in Windows programs habitually employ a special feature: for menu items that act as toggles, the on/off state is indicated by the presence or the absence of a check mark next to the item in the menu.

The SourceExpr property of a menu item is used for controlling whether a check mark is displayed or not. Initially, the SourceExpr property is undefined. You can define it to a valid C/AL expression that evaluates to a boolean. The check mark appears when the value is TRUE.

You can see how this feature can be used in C/SIDE itself. In the Format menu, the Snap to Grid menu item is a typical example: it can either be on or off. When it is On, the check mark is displayed.

## 8.6 FORM AND CONTROL TRIGGERS

While the system interprets and acts upon many events in a predefined way, certain actions – such as opening a form or pushing a command button – cause the system to execute a user-definable C/AL function (the event *triggers* the function). You will typically use triggers to do 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 default behavior does not suit your purpose.

### Overview of Form Triggers

These triggers pertain to forms in C/SIDE

Form trigger name	Executed when...
OnInit	the form is loaded, but before controls are available.
OnOpenForm	the form has been initialized (controls are available).
OnQueryCloseForm	the form is about to close, but before OnCloseForm. If this trigger returns FALSE, the form is not closed. The intended use is for asking the user if he or she really wants to close the form.
OnCloseForm	the form is about to close, and after OnQueryCloseForm.
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. OnAfterGetRecord 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.

The table only sketches out the main purpose of each trigger. Refer to the online C/SIDE Reference Guide for extensive descriptions and details.



**Note**

.....

The last three triggers in the table – OnInsertRecord, OnModifyRecord, OnDeleteRecord – correspond to triggers at table level. If you use triggers at both form and table level, the triggers at form level will be executed first, then the triggers at table level.

.....

**Overview of Control Triggers**

Depending on the type of a control (see Chapter 7), controls have a varying number of triggers. Static controls and container controls do not have any triggers at all, while text boxes have a full range. Other data controls and data container controls have a subset of the possible triggers. Controls that can be *pushed* – such as a command button or menu item, and also a check box – have a special trigger to handle this.

The following table outlines the full range of triggers. The column at the right indicates the controls for which the trigger is relevant.

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.	1,3,4,9
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

The table provides a sketch of the main purpose of each trigger. Refer to the online C/SIDE Reference Guide for concise descriptions and details.

### Note

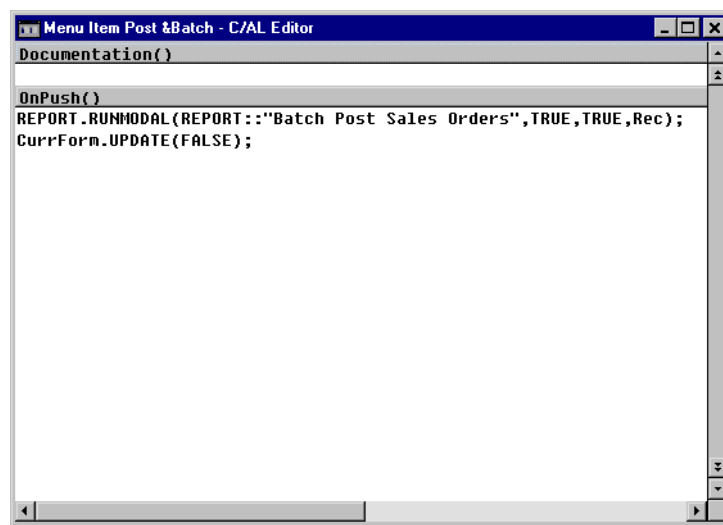
.....  
OnValidate is also a field trigger at the table level. If both triggers (field and control) are defined, the field trigger is executed before the control trigger (and the system default validation before anything else).

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 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

When you want to define a function to be triggered by a form or control event – or modify an existing function – follow these steps:

- 1 Open the form in the Form Designer.
- 2 Select the form itself or the control (or menu item) in question.
- 3 Open the C/AL Editor (choose C/AL Code from the View menu).



- 4 In the editor, you will only have access to those triggers that are relevant for the object that you selected. Enter C/AL code in those triggers you want to use, or modify those existing triggers you want to.
- 5 You can test-compile the form, thus including the code, by choosing Compile from the Tools menu.

If you are not familiar with the C/AL programming language, you should read Part 4, Codeunits, in this guide.





## Chapter 9

### 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.

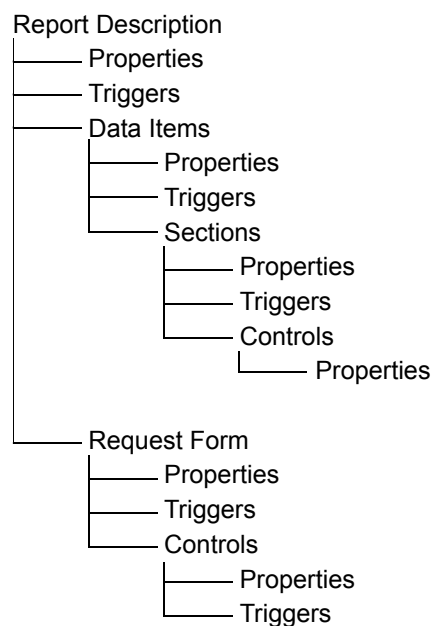
- What Are Reports?
- What Happens When a Report Runs?
- The Report Designer
- Saving, Compiling and Running Reports

## 9.1 WHAT ARE REPORTS?

Reports in C/SIDE have several purposes:

- Reports are used to print information from a database in a structured way. For example, in a sales order application, you can create a report that contains a list of all customers and for each customer lists all orders placed by that customer.
- All documents pertaining to an application must be created as reports. For example, in order to print an invoice, you will create a report that is automatically filled out with the relevant information.
- Reports can be non-printing. While this may sound like a contradiction in terms, it is not. A report can be used to automate many recurring tasks such as updating all prices in an item list. This could be performed entirely from C/AL code in a codeunit, but using a report makes it a lot easier because you can use the powerful data modeling available for report design.

The following diagram shows the components of a report and their relationship. This and the following chapters will explore all components in depth.



### The Report Components

The diagram above outlines how a report is composed from a number of different components. Below you will find a short description of each component.

**Report Description** This is the total description of the report: how data is collected, and how data is presented on paper when the report is run. The report description is stored in the database.

**Data Item** A data item corresponds to a table. In order to retrieve information from the tables in the database, you define data items. When a report uses more than one table, you set relations between the data items in order to retrieve and organize data in the way that you want.

**Section** In a printing report, each data item has one or more sections. A section can be thought of as a block of information to print on the paper. The complete report is composed of a number of sections, some that are printed only once, for example a header, and some that are printed for each record that is retrieved from the database.

**Control** The information that is 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 complex calculations), *labels* for printing static text such as a caption for a column of data, and *shapes*, *images* and *picture boxes*, for printing graphical elements (lines, circles) and bitmap pictures in a report.

**Request Form** A request form is a form that is run before the actual report begins execution. It is used to gather requests and options from the user of the report—for example, sort order or level of detail.

**Property** A property is an attribute of an object – report, data item, section, and so forth—that characterizes the object in some way: color, size, whether it is displayed, and much else. Properties are set on the Property Sheet of an object.

**Trigger** Certain predefined events that happen to a report cause the system to execute a user-definable C/AL function – the event *triggers* the function. As you can see in the diagram, the report itself, the data items, the sections, the request form and the controls on the request form all have triggers. Triggers are edited in the C/AL editor.

## Logical and Visual Design

There are two sides to designing a report: defining the logical structure, the *data model*, and designing the visual layout.

Defining the data model means defining how the data for the report is collected. This includes

- defining the tables the report will use by creating data items.
- defining relationships between data items if the report uses more than one table.
- defining the key, sort order and filters to use with the involved data items.
- defining how data is to be grouped.
- defining how subtotals and totals are to be calculated.
- possibly writing C/AL code in data item triggers to obtain advanced functionality.

### Data Items

The data model of a report is built from *data items*. A data item corresponds to a table. When the report is run (see the diagram on page 164) each data item is iterated for all records in the underlying table. When a report is based on more than one table, you establish a hierarchy of data items to control how the information is gathered by *indenting* data items.

### EXAMPLE

In order to make a report that prints out a list of customers and for each customer lists sales orders placed by that customer, you will define two data items: one that corresponds to the Customer table and one that corresponds to the Sales Order table. The second data item is indented: as the report works its way through the records in the Customer table, for each customer all sales orders that are related to this customer must be 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 have to print anything), one or more sections are attached to each data item. There are several types of sections, each having a specific function. Normally, the bulk of the data is printed out in the body section of a data item, while the header section of the data item is used to print information before any record of the data item is printed (for example, column captions), but there are reports – like some of the examples in this guide – where the body section is not used at all, and all information is printed in other sections.

The following picture shows a finished report.

Sales Statistics						
CRONUS International Inc.						
Customer: No.: 10000..30000						
No.	Name	...before	09/28/95 10/27/95	10/28/95 11/27/95	11/28/95 12/27/95	after...
<b>10000</b>	<b>Kontorforsyningen A/S</b>					
	Sales (LCY)	57,509.00	0.00	0.00	0.00	0.00
	Profit (LCY)	12,655.00	0.00	0.00	0.00	0.00
	Profit %	22.0	0.0	0.0	0.0	0.0
	Inv. Discounts (LCY)	0.00	0.00	0.00	0.00	0.00
	Pmt. Discounts (LCY)	0.00	0.00	0.00	0.00	0.00
<b>20000</b>	<b>Ravel Mobler</b>					
	Sales (LCY)	1,525.00	0.00	0.00	0.00	0.00
	Profit (LCY)	335.00	0.00	0.00	0.00	0.00
	Profit %	22.0	0.0	0.0	0.0	0.0
	Inv. Discounts (LCY)	0.00	0.00	0.00	0.00	0.00
	Pmt. Discounts (LCY)	0.00	0.00	0.00	0.00	0.00
<b>30000</b>	<b>Lauritzen Kontormøbler A/S</b>					
	Sales (LCY)	13,676.20	0.00	0.00	0.00	0.00
	Profit (LCY)	2,444.20	0.00	0.00	0.00	0.00
	Profit %	17.9	0.0	0.0	0.0	0.0
	Inv. Discounts (LCY)	0.00	0.00	0.00	0.00	0.00
	Pmt. Discounts (LCY)	0.00	0.00	0.00	0.00	0.00
<b>Total</b>						
	Sales (LCY)	72,710.20	0.00	0.00	0.00	0.00
	Profit (LCY)	15,434.20	0.00	0.00	0.00	0.00
	Profit %	21.2	0.0	0.0	0.0	0.0
	Inv. Discounts (LCY)	0.00	0.00	0.00	0.00	0.00
	Pmt. Discounts (LCY)	0.00	0.00	0.00	0.00	0.00



The report above prints sales statistics information and retrieves all its data from one table. It demonstrates a range of the features that are available for designing reports.

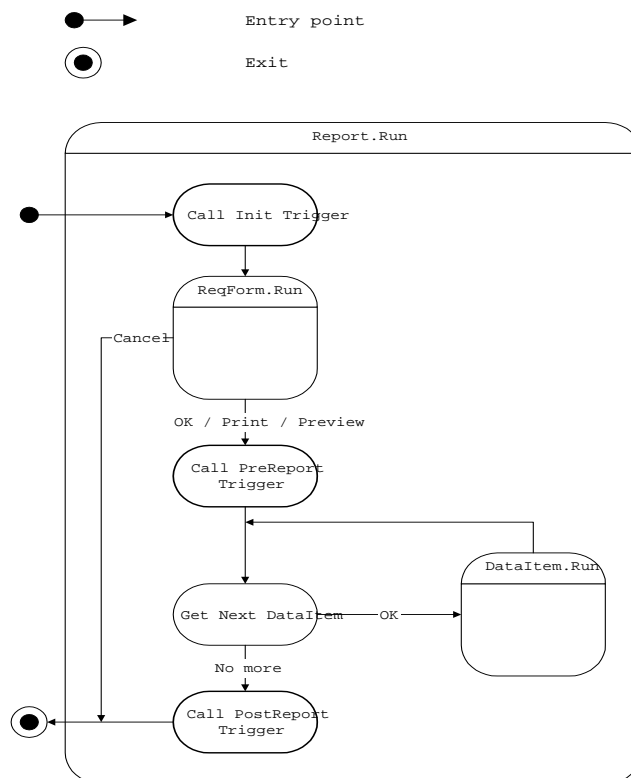
- Before any record from the table is printed, there is a header – containing a title and information about the filter that was used on the customer numbers.
- Each body section prints information about a customer on several lines. The "Profit %" lines are calculated as the report is run.
- After all records (all records that were selected by the filter, that is) 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 has been applied to create columns where data are collected and totalled for different periods.

## 9.2 WHAT HAPPENS WHEN A REPORT RUNS?

The two flow charts in this chapter are simplified versions of the flow charts in Appendix B, Report Flow Charts, on page 480. If you want to acquaint yourself with all details—including why and when triggers are executed—you should consult that appendix. The focus here will be on describing in general terms the way a report is run.

### The Report Run

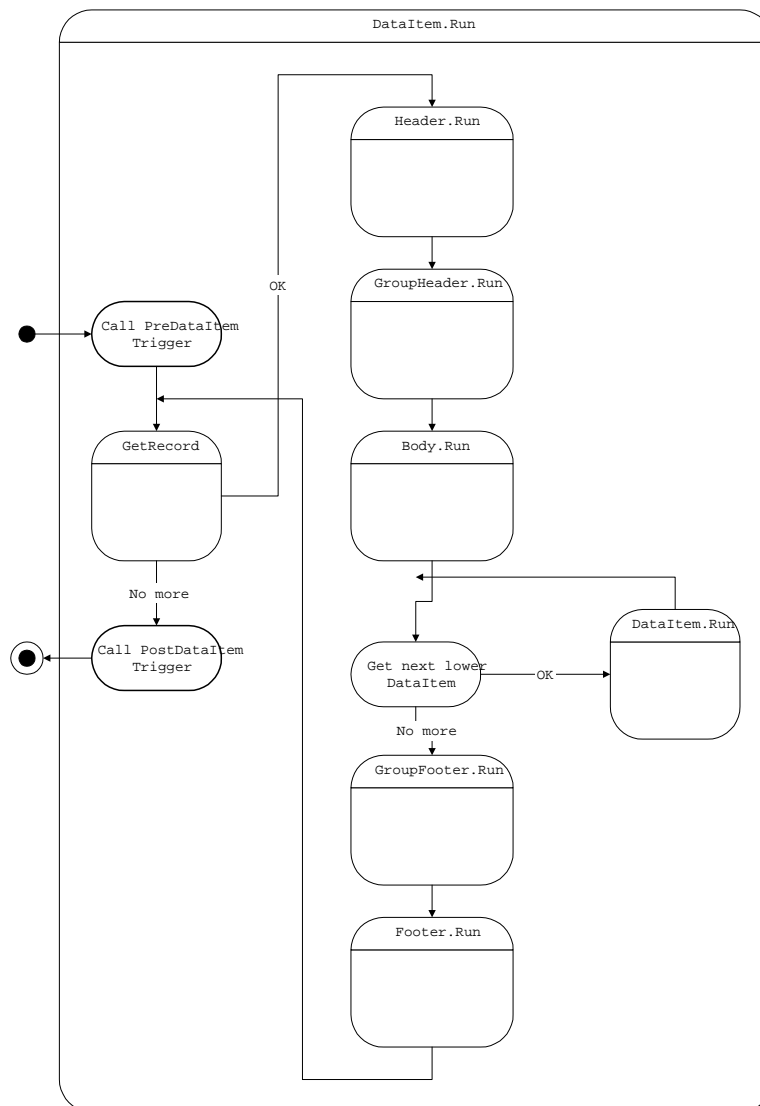
The flow chart below illustrates the events that take place when a report runs.



- 1 When the user initiates the report run, the OnInitReport trigger is called. This trigger can perform processing that would be necessary before any part of the report is run—or 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, the user can choose to cancel the report run.
- 3 If the user chooses 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 (if any) data item will be processed in the same way.
- 6 When there are no more data items, the OnPostReport trigger is called. You can use this trigger to do any post processing that is necessary, for example cleaning up by removing temporary files.

The flow chart below further explores step 4 – how a data item is processed:



- 1 Before the first record is retrieved, the OnPreDataItem trigger is called, and after the last record has been processed, the OnPostDataItem trigger is called.
- 2 Between these two triggers, the records of the data item are processed. Processing a record means executing the record triggers and outputting sections. C/SIDE also determines whether the current record should cause outputting of a special section: header, footer, group header or group footer.

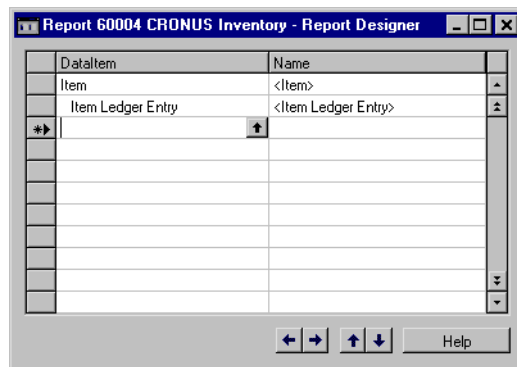
- 3** If there is an indented data item, a data item run will be initiated for this data item (data items can be nested 10 levels deep).
- 4** When there are no more records to be processed in a data item, control returns to the point from which the processing was initiated. For an indented data item this will be to the next record of the data item on the next higher level. If the data item is already on the highest level (indentation is zero) control will return to the report – as shown in the first flow chart (Report.Run).

## 9.3 THE REPORT DESIGNER

The Report Designer contains two additional designers: the Section Designer, used for designing the layout of reports, and the Request Options Form Designer, used for designing request options forms.

### The Report Designer

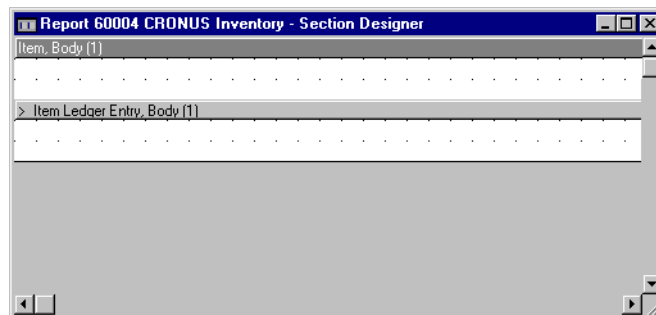
In the Report Designer window, you define the data model by adding data items and indenting them appropriately:



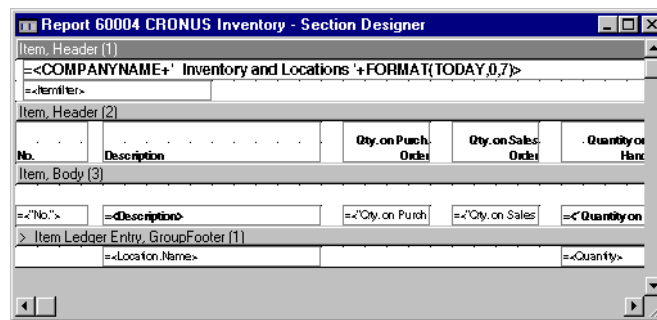
Properties and triggers for each of the data items can be edited by opening the Property Sheet or the C/AL editor, respectively, while the data item is selected. Properties and triggers for the *report* can be edited by selecting an empty line in the Report Designer window and then opening the Property Sheet or the C/AL editor, or by choosing Select Object from the Edit menu.

### The Section Designer

When one or more data items have been defined, 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 as described for forms on page 121. In the picture below, a number of text boxes and labels have been placed in four sections.



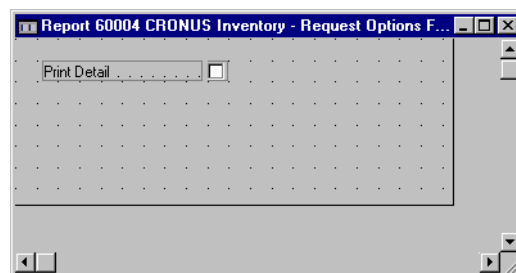
You can think of each section as one or more lines on the paper that the report will eventually be printed on. A header section is printed only once, while a body section typically will be printed several times as the report loop is iterated. You can control whether the header will be printed when a page break occurs while body sections of the same data item are being printed.

You can edit properties and triggers for each section by opening the Property Sheet 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 (as not all properties are relevant on 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 on the Property Sheet, and you can read about them in chapter 7, Designing Forms, or in the online C/SIDE Reference Guide.

### The Request Options Form Designer

The Request Options Form Designer is used to create a form with fields that prompt the user for options before the report is run. This designer works exactly like the Form Designer.



You only have to use this designer if you want to prompt the user to select options. When a report is run, the request form looks like this:

The screenshot shows a window titled "CRONUS Inventory" with three tabs: "Item", "Item Ledger Entry", and "Options". The "Item" tab is active, displaying a table with two columns: "Field" and "Filter". The table contains two rows: "No." and "Description". To the right of the table are vertical scroll bars. Below the table is a small button with an upward-pointing arrow. At the bottom of the window are four buttons: "Print...", "Preview", "Cancel", and "Help".

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 they are 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 screenshot shows the same "CRONUS Inventory" window, but with the "Options" tab selected. The main area of the window is a large, empty rectangular box. In the top-left corner of this box, the text "Print Detail . . . . . " is displayed next to a small icon of a document with a magnifying glass. The "Item" and "Item Ledger Entry" tabs are visible but not active. The "Print...", "Preview", "Cancel", and "Help" buttons remain at the bottom.

The form has the same properties and triggers as any other form, and the same controls can be placed on it.

## 9.4 SAVING, COMPILING AND RUNNING REPORTS

After you have designed a report, you must save and compile it before it can be run. Normally, you will do this when you are done designing the report. However, you may want to save a report that is not yet finished and thus cannot be compiled, for example, if the report is more complex than the reports described so far and contains C/AL code. You can also test-compile a report without closing or saving it.

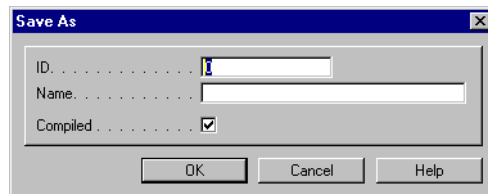
### Saving and Closing a Report

A report is closed when the Report Designer window is closed. You can close this window in the same ways that you can close any other window.

To save a report:

- 1 When you close a report, C/SIDE will ask whether the report should be saved. If it is a new report (a report that has not been saved before) you will have to assign an ID and a name. The ID must be unique and follow the rules for numbering objects – your C/SIDE dealer will provide you with this information.

Hint: if you enter ID and Name as report properties, these values will be used, and you will not be prompted for ID and Name when you close the report.



- 2 The option field Compiled is by default set to TRUE (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 Choose OK to save the report.

You can save a report without closing it by choosing Save or Save As from the File menu. By using Save As, you can rename an existing report (thereby in effect copying it).

### Compiling a Report

Reports, like other objects in C/SIDE, must be compiled before they can be run. As described above, you can choose to compile a report whenever you are saving it.

While you are designing a report, you may want to test-compile a report, to find possible errors (this possibility will be more important if the report contains C/AL code in triggers, as described in chapter 11). You can test-compile a report during design by choosing the Compile option from the Tools menu.

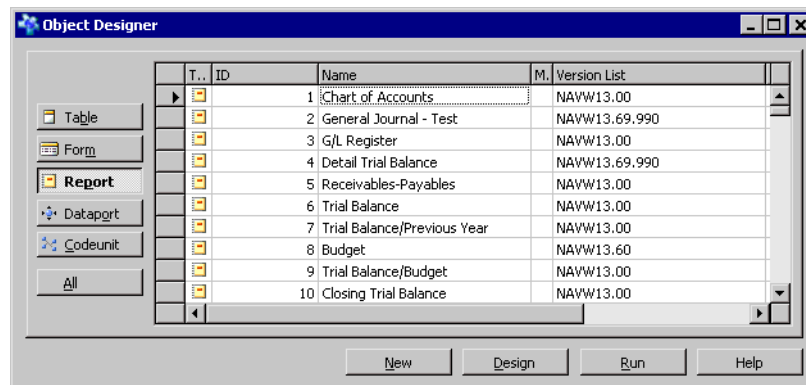


## Running a Report

In a finished application your reports will be incorporated into menus, or they will be called from, for example, a command button on a form. However, while you are designing reports, you will often want to run them before they have been integrated into an application.

**Test-running reports** While designing a report, you can test-run the report by choosing Run from the File menu. In this way, the report will be compiled and run in its current stage of development. It will not be 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 main window by selecting it and clicking the RUN button.





## Chapter 10

### Designing Reports

This chapter describes the properties of reports, and then, by creating two examples, shows the basic steps involved in designing reports.

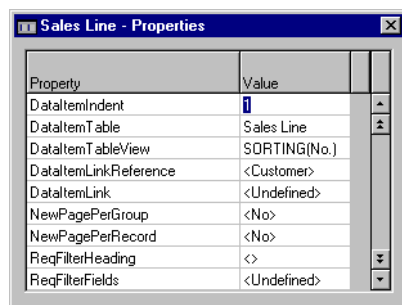
- Report Properties
- Designing a Simple Report
- Designing a More Advanced Report

## 10.1 REPORT PROPERTIES

As described in Chapter 1, C/SIDE Fundamentals, properties are a system-wide feature and every application object has some properties. All objects in a report have properties:

- The report itself
- The data items
- The sections
- The controls in the section
- The request form
- The controls on the request form

Properties for reports can be set by opening the Property Sheet (choose Properties from the View menu) while an object is selected.



You select objects as follows:

- Select a data item in the Report Designer window by clicking it.
- Select the report itself by clicking an empty line or by clicking Edit, Select Object.
- Select a section in the Section Designer by clicking either the section bar or somewhere in the section (not on a control, though).
- Select a control by clicking it.

Use the Value field to set the value of each property. As soon as you leave this field (by hitting ENTER or by moving with the arrow keys), the property will be updated. If what you entered contains an error, the update will not be accepted.

Default values are displayed in angle brackets (<>). You can reset any property (for which there is a default) to the default by deleting the current value and then moving out of the field.

### How Properties Are Inherited

Controls that have a direct relationship to table fields will inherit the settings of those properties that are common to the field and the control. For example, in an accounting application you will want to store some calculated amounts with five decimal places, to obtain a high degree of precision. However, on a printed report, you will only want to display currency amounts with the customary 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, obviously).

## Report Properties

The table below briefly describes the report properties. All properties are described in detail in the online C/SIDE Reference Guide. You can get context-sensitive Help for a property by opening the Property Sheet for a report, placing the cursor on a property and pressing F1.

The Property Sheet for a report is opened by choosing View, Properties while an empty line is selected in the Report Designer window, or by choosing Edit, Select Object.

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).
ShowPrintStatus	Should the printing status window be displayed during printing (with the opportunity to cancel printing)?
UseReqForm	Should the request form be run before the report is run?
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	No printing—only processing. If Yes, the report cannot have sections.
Description	Description—for internal purposes, as it is not user-visible.
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.)
Orientation	Use this property to set the page orientation for this report. Values are Portrait and Landscape.
PaperSize	Use this property to set the paper size for this report.
PaperSource	Use this property to specify which paper source to use when printing this report.

Property	Meaning
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 table below briefly describes the data item properties. All properties are described in detail in the online C/SIDE Reference Guide. You can get context-sensitive Help for a property by opening the Property Sheet for a Data Item, placing the cursor on a property and pressing F1.

Property	Meaning
DataltemIndent	Indentation level (can be set in the designer when creating data items).
DataltemTable	Table of item (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 Dataltem will be linked to.
DataltemLink	Link between the current Data Item and the Data Item specified by DataltemLinkReference.
NewPagePerGroup	Should each group be printed on a separate page?
NewPagePerRecord	Should each record be printed on a separate page?
ReqFilterHeading	Tab caption for this item on request form (default is name of DataltemTable).
ReqFilterFields	Names of the fields that will be included in the ReqFilter 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 name of DataltemTable).
PrintOnlyIfDetail	Print item only if sublevels generate output.

### Section Properties

The table below briefly describes the section properties. All properties are described in detail in the online C/SIDE Reference Guide. You can get context-sensitive Help for a property by opening the Property Sheet for a Section, placing the cursor on a property and pressing F1.

Property	Meaning
PrintOnEveryPage	Should header and footers be printed on all pages?

Property	Meaning
PlaceInBottom	Should footer be placed below last line or at bottom of page?
SectionWidth	Width in 1/100 mm.
SectionHeight	Height in 1/100 mm.

### Control Properties

Controls in reports have exactly the same properties as controls on forms – that is, those properties that it makes sense to set in a report. The Property Sheet of a control shows the properties, and chapter 7, "Designing Forms", describes each property, as does the online C/SIDE Reference Guide.



## 10.2 DESIGNING A SIMPLE REPORT

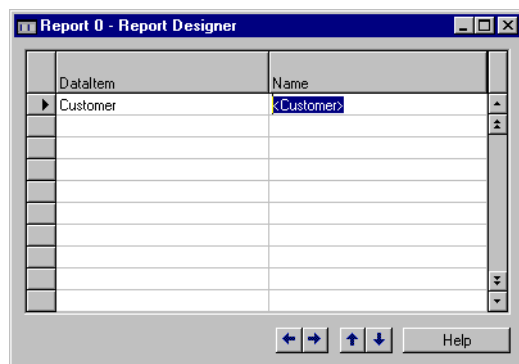
This section goes through the steps required to create a very simple report, in which a list of customers is created, based on one table that contains customer information.

### Defining the Data Model

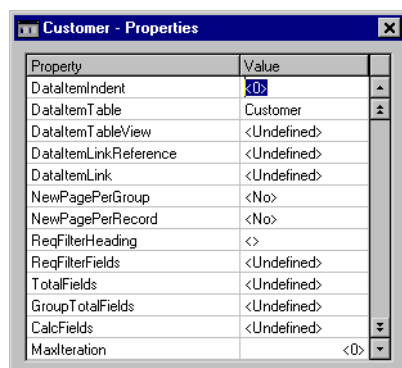
The first step is to define the data model by creating the data items that you want to use.

To create a data item:

- 1 Click Tools, Object Designer.
- 2 In the Object Designer, click Report.
- 3 Click New. C/SIDE opens the Report Designer.
- 4 In the **New Report** window, in the **Table** field, click the AssistButton  to select a table from the Table List window.
- 5 In the **Report** field, click Create a blank report and then click OK.
- 6 In the **Report Designer** window, in the first **Data Item** field, click the AssistButton  and select a table from the **Table List** window. The Name is by default set to the name of the table. You do not have to change it in this report. In this example, the Customer table has been chosen, and the default for Name is "Customer".

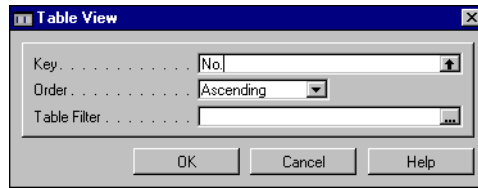


- 7 Click View, Properties to open the **Properties** window for the data item.

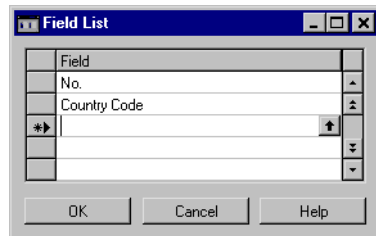




- 8 Select the `DataItemTableView` property and click the `AssistButton ...` to open the **Table View** window:

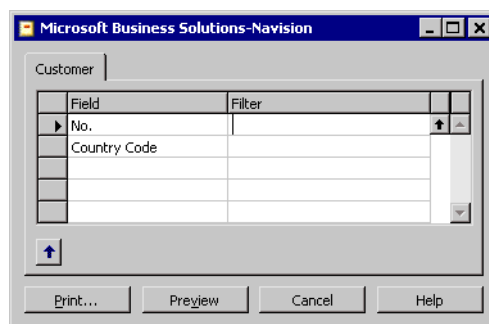


- 9 Select the key, sort order and filters that you want to use and then click OK. In the example, a key (previously defined during table design) consisting only of the **No.** field has been chosen, and the sort order is set to Ascending. The **Table Filter** field has been left empty, meaning that a permanent filter is not defined on the table.
- 10 Select the `ReqFilterFields` property, and click the `AssistButton ...` to open the following window:



- 11 Select the fields on which the user will often need to set filters. You can use the lookup function to select them. In the example, the fields **No.** and **Country Code** have been selected. When you have selected the fields, press OK.

The picture below shows the request form the user will see when the report is run (with the various choices made as in the steps above).



As the key and sort order were established during report design, the only choice left for the user involves setting filters. The fields that were defined as `ReqFilterFields` are shown, but the user can also choose to put a filter on other fields by adding lines below those that are already used.

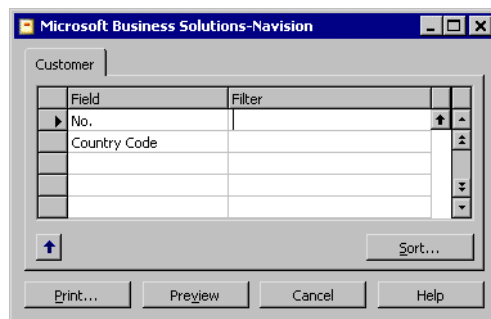
Concerning `ReqFilterFields`, you should be aware that the user can choose to set filters on other fields than those you specify. However, it will still be a good idea to add the fields that those who use the report will often want to set filters on. If the table has

a lot of fields, the casual user may find it difficult to find the relevant fields to filter from a lookup list of all the fields in the table.

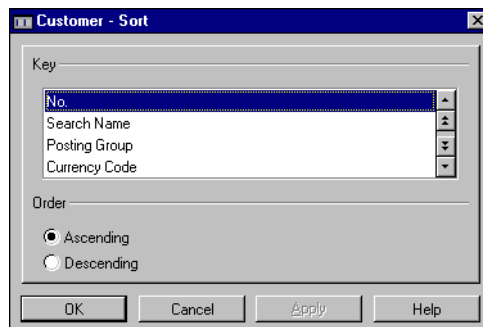
You can remove the filter-selection tab altogether by not defining any ReqFilterFields for the data item and by setting the DataItemTableView to define a sort order. If you create a request options form, it will still be shown.

If there is no request options form, an empty form will be displayed. On this, the user can choose Print, Cancel, and so forth. Finally, if you set UseReqForm to No, the report will start printing as soon as it is run. In this case, the user will not be able to change his or her mind and cancel the report run altogether. (It will still be possible to cancel printing, but some pages will probably be printed).

If a DataItemTableView is not defined, the user will be able to select key and sort order at runtime. Then, the request form will look like this:



When the user clicks Sort, they can choose the key and sort order from this form:



### Be careful...

.....  
 about what you allow the user to change. In a more complex report, where you work with data from several tables, the functionality may well depend on a specific key and sort order. On the other hand, letting the user choose filters freely will not interfere with the logic of the report. In a very simple report like this one, 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 wizard will guide you through the steps of selecting the fields that the report will be based on and the sorting order.

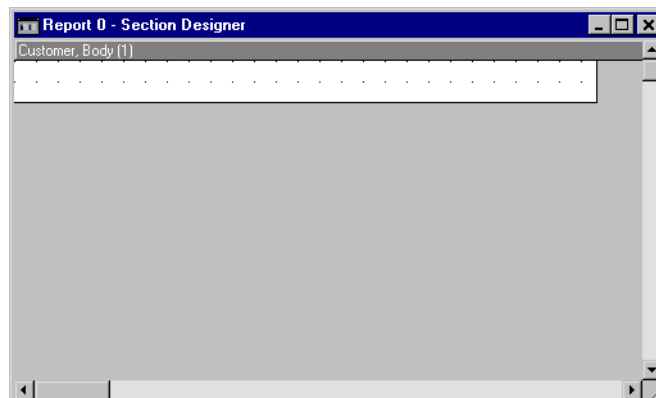
Note that on the first page of the wizard, the contents of the **Available Fields** field are the *Caption* properties of the fields – not the *Name* property. For more information about captions, see Chapter 18.

## Designing the Sections

So far, only the data model of the report has been defined. So far, nothing will be printed. The next step, therefore, is to design the sections.

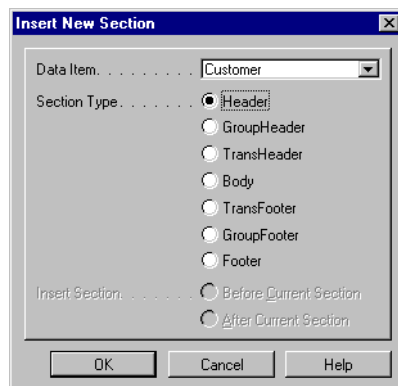
To design report sections:

- 1 Open the Section Designer by clicking View, Sections, while the Report Designer window has the focus. Having created a data item as described above, the Section Designer will look like this:



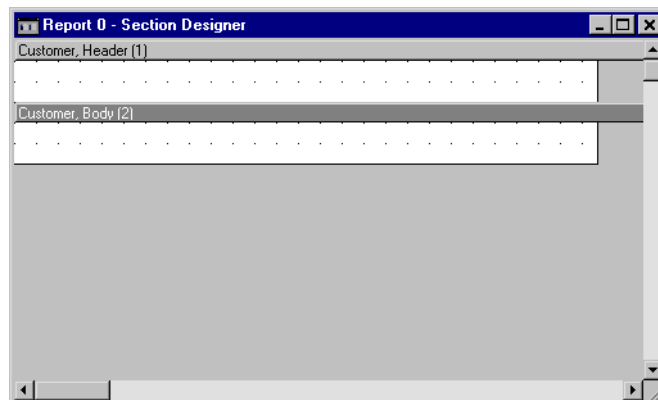
As you can see, a section named "Customer - Body" ("1" means that this is presently the first section of this data item) has been inserted. By default, a Body section will be inserted for each data item that has been created; these sections will be in the same order as the data items in the Report Designer.

- 2 Now, insert a header section for the Customer data item. Choose Insert New from the Edit menu. The following form appears:



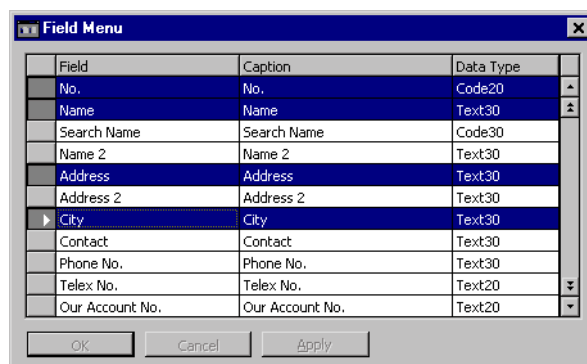
The 'Insert New Section' dialog box has a title bar with a close button. It contains three main sections: 'Data Item' with a dropdown menu set to 'Customer'; 'Section Type' with radio buttons for Header (selected), GroupHeader, TransHeader, Body, TransFooter, GroupFooter, and Footer; and 'Insert Section' with radio buttons for Before Current Section (selected) and After Current Section. At the bottom are OK, Cancel, and Help buttons.

- 3 Choose Header as the Section Type and Before Current Section. Then press OK. The Section Designer now looks like this:



The 'Report 0 - Section Designer' window shows a design area with two sections: 'Customer, Header (1)' at the top and 'Customer, Body (2)' below it. The design area is a large rectangle with a grid pattern. The title bar includes standard window controls.

- 4 Open the field menu by choosing Field Menu from the View menu. Select the fields that you want in the report. (You can select multiple fields by holding down CTRL while clicking in the selection bar.) In the following example, four fields have been selected.



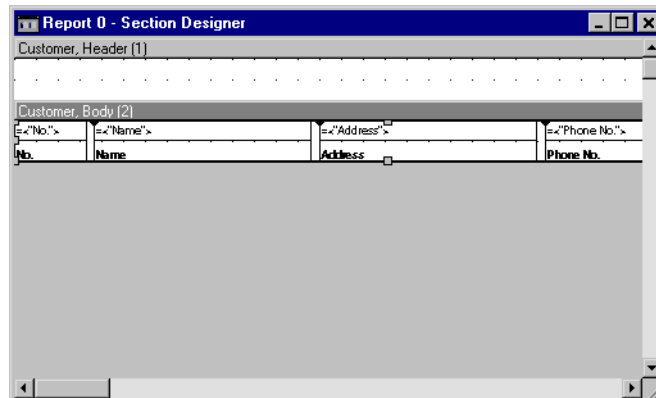
The 'Field Menu' dialog box displays a table of fields. The first four rows are selected (highlighted in blue). The table has three columns: Field, Caption, and Data Type. At the bottom are OK, Cancel, and Apply buttons.

Field	Caption	Data Type
No.	No.	Code20
Name	Name	Text30
Search Name	Search Name	Code30
Name 2	Name 2	Text30
Address	Address	Text30
Address 2	Address 2	Text30
City	City	Text30
Contact	Contact	Text30
Phone No.	Phone No.	Text30
Telex No.	Telex No.	Text20
Our Account No.	Our Account No.	Text20

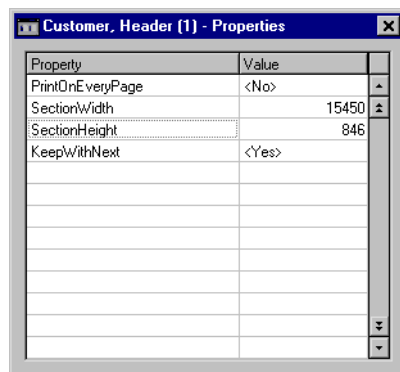
- 5 Move the mouse cursor into the Body section of the data item. Click once to activate the window—the cursor changes into the Control Insertion cursor. Place the



cursor at the left side of the section and click again. A text box with an attached label will be inserted for each selected field.

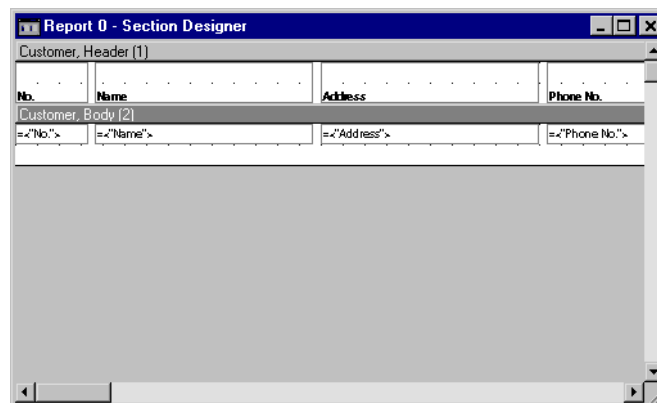


- 6 Open the Property Sheet by choosing Properties from the View menu. Click in the Header section. Look at the setting of SectionWidth (the unit of measure is 1/100 mm).



All sections' widths have been modified to make room for the inserted controls (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 – that paper is 21 cm wide.

- 7 Select all the labels as a multiple selection (hold down CTRL while clicking), and move them all into the Header section in one move (in this way, the alignment of labels and text boxes will be preserved).



Now, the report is ready to be printed, but it still needs some work before it will look good on paper.

- 8 If the report, when it is run, ends up consisting of more than one page, you will want the Header section – containing the labels – to appear on every page. Open the Property Sheet for the Header section and set the PrintOnEveryPage property to Yes (see the picture in step 6 above).
- 9 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 mouse 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.
- 10 Save and close the report, and run it from the Object Designer. The example described so far gives this result with sample data:

No.	Name	Address	Phone No.
10000	Kontorforsyningen A/S	Carl Blochs Gade 7	11223344
20000	Ravel Møbler	Parkvej 44	22334455
30000	Lauritzen Kontormøbler A/S	Jomfru Ane Gade 56	33445566
01121212	Spotsmeyer's Furnishings	612 South Sunset Drive	9998887771
01445544	Progressive Home Furnishings	3000 Roosevelt Blvd.	8887776661
01454545	New Concepts Furniture	705 West Peachtree Street	7776665551

As you can see, the label and the data in **No.** 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.

- 11 Select the **No.** label and open the Property Sheet. Set the HorzAlign property to Right.

Now the report looks like this:

No.	Name	Address	Phone No.
10000	Kontorforsyningen A/S	Carl Blochs Gade 7	11223344
20000	Ravel Mobler	Parkvej 44	22334455
30000	Lauritzen Kontormobler A/S	Jomfru Ane Gade 56	33445566
01121212	Spotsmeyer's Furnishings	612 South Sunset Drive	9998887771
01445544	Progressive Home Furnishings	3000 Roosevelt Blvd.	8887776661
01454545	New Concepts Furniture	705 West Peachtree Street	7776665551

## 10.3 DESIGNING A MORE ADVANCED REPORT

The report designed in section 10.2 was very simple: it ran through one table and printed out the records. In this section, you will learn to design reports that use more than one table.

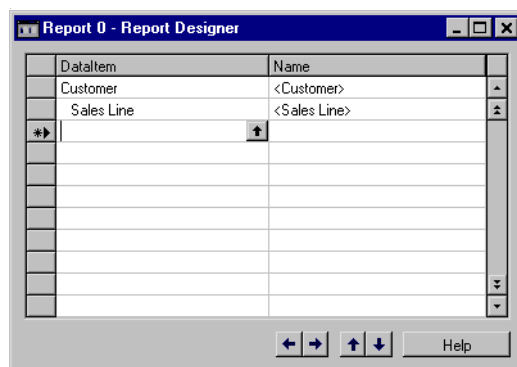
The sample report that will be created uses two tables: one is the customer table, as in the preceding example. The other table contains sales lines, lines from not-yet-posted sales orders that contain information about the actual items that have been ordered. There is a one-to-many relationship between the two tables: while one customer can have many items on order, a sales line can pertain to only 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 in section 10.2 above.

To define the data model:

- 1 In the Report Designer window, choose the Customer table as the first data item, and the Sales Line table as the second.
- 2 Indent the Sales Line data item by clicking the right-arrow button once while the data item is selected:



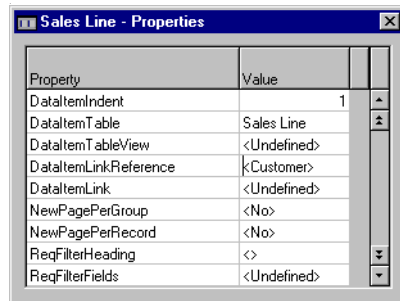
The data model defined thus far will work like this:

- The report will run through the Customer data item.
- For *each* record in the Customer data item, the report will run through the *entire* Sales Line data item.

This is clearly not the purpose of the report – you need a way to select only those Sales Line records that are related to the current customer. This is accomplished by 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 will be selected from the Sales Line table only when the Sell-to Customer No. is the same as the No. in the Customer table.



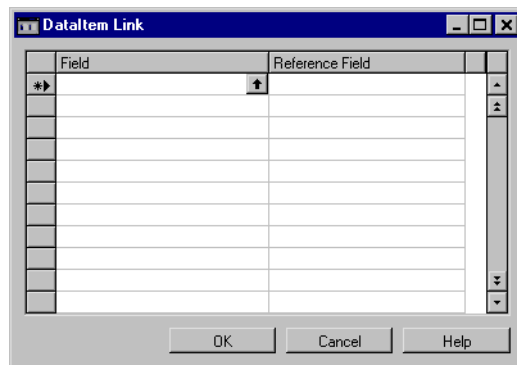
- 3 Open the Property Sheet for the Sales Line data item.
- 4 Set the `DataItemLinkReference` property to the name of the less-indented data item (Customer) that the more-indented data item (Sales Line) must be related to. In most cases, including this one, this is the default.



The 'Sales Line - Properties' dialog box shows a list of properties and their values. The 'DataItemLinkReference' property is set to '<Customer>'.

Property	Value
DataItemIndent	1
DataItemTable	Sales Line
DataItemTableView	<Undefined>
DataItemLinkReference	<Customer>
DataItemLink	<Undefined>
NewPagePerGroup	<No>
NewPagePerRecord	<No>
ReqFilterHeading	<>
ReqFilterFields	<Undefined>

- 5 In the value field of the `DataItemLink` property, open the form shown below by clicking the AssistButton ... :

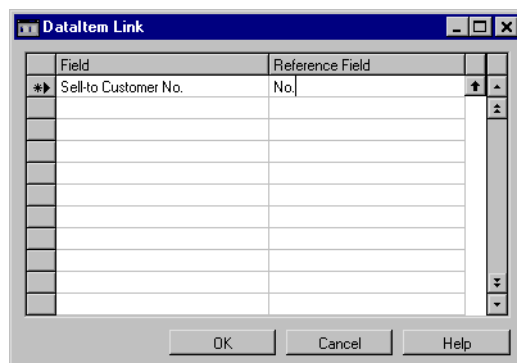


The 'DataItem Link' dialog box is shown with empty 'Field' and 'Reference Field' columns.

Field	Reference Field

Buttons: OK, Cancel, Help

- 6 In the **Field** field, enter the name of the field from Sales Line (the more-indented data item) that must correspond to a field from Customer (the less-indented data item). You can use the lookup function to select the field.
- 7 In the **Reference Field** field, enter the name of the field from Customer that must correspond to the field from Sales Line. Again, you can use the lookup function to select the field. In the example below, the **Sell-to Customer No.** field from the Sales Line data item and the **No.** field from the Customer data item have been chosen.



The 'DataItem Link' dialog box is shown with 'Sell-to Customer No.' in the 'Field' column and 'No.' in the 'Reference Field' column.

Field	Reference Field
* Sell-to Customer No.	No.

Buttons: OK, Cancel, Help

- 8 Finally, open the Property Sheet for the Customer data item, and set the PrintOnlyIfDetail property to Yes. This will cause the Customer body sections to be printed only if there is data to print from Sales Line.

The data model now works like this:

- The Customer data item will be run through.
- For each record in the Customer data item, records in the Sales Line data item will be 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 will be printed – not even the information from the Customer data item.

## Designing The Sections

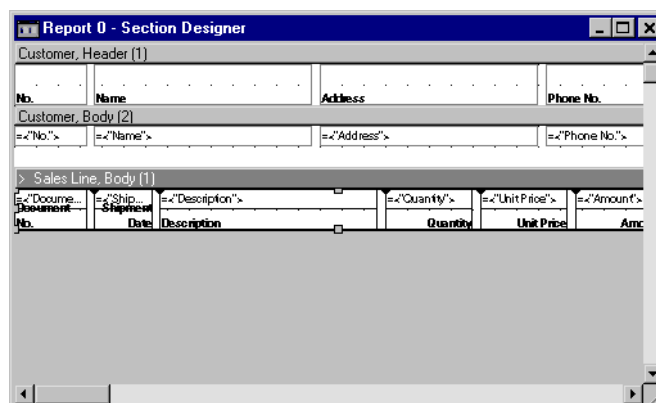
Presuming that you already know how to design the sections for a report with just one data item, the description here concentrates on showing how to handle a situation with two data items.

To design the sections:

- 1 When you first open the Section Designer, there will already be a Body section for each data item. Add a Header section for the Customer data item.
- 2 Add fields to the Customer body section. Move the labels up into the Header section.

So far, the procedure has been exactly the same as for creating the first, simple report. Now, continue like this:

- 3 Add fields to the Sales Line body section.



- 4 At this point, you need to make a decision about the labels for the controls of Sales Line: if they stay where they are right now, 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, which is for each record of the Customer data item. As neither of these solutions seems very good, you can take a

third approach: you can move the labels into the header section of the Customer data item, like this:

The screenshot shows the 'Report 0 - Section Designer' window. It displays two sections: 'Customer, Header (1)' and 'Customer, Body (2)'. The 'Customer, Header (1)' section contains a table with columns: No., Name, Address, and Phone No. The 'Customer, Body (2)' section contains a table with columns: Document No., Shipment Date, Description, Quantity, Unit Price, and Amount. The 'Sales Line, Body (1)' section contains a table with columns: Document No., Shipment Date, Description, Quantity, Unit Price, and Amount. The labels for the Sales Line columns are in normal font weight, while the labels for the Customer columns are bold.

No.	Name	Address	Phone No.		
Document No.	Shipment Date	Description	Quantity	Unit Price	Amount

Document No.	Shipment Date	Description	Quantity	Unit Price	Amount
Document No.	Shipment Date	Description	Quantity	Unit Price	Amount

- 5 Now, labels for both the Customer records and the Sales Line records will be printed as column captions in the Customer Header section (remember to set the PrintOnEveryPage property of this section to Yes). In order 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. Also, the text boxes of the Customer data item can be changed to bold, to make these records stand out among the lines that are printed. (There are bound to be a lot more records from Sales Line than from Customer.) Furthermore, the Sales Line labels have been resized to occupy only one line, and an empty line has been added to the header section.

The screenshot shows the 'Report 0 - Section Designer' window. It displays two sections: 'Customer, Header (1)' and 'Customer, Body (2)'. The 'Customer, Header (1)' section contains a table with columns: No., Name, Address, and Phone No. The 'Customer, Body (2)' section contains a table with columns: Document No., Shipment Date, Description, Quantity, Unit Price, and Amount. The 'Sales Line, Body (1)' section contains a table with columns: Document No., Shipment Date, Description, Quantity, Unit Price, and Amount. The labels for the Sales Line columns are in normal font weight, while the labels for the Customer columns are bold. The Sales Line labels are resized to occupy only one line, and an empty line has been added to the header section.

No.	Name	Address	Phone No.		
Document No.	Shipment Date	Description	Quantity	Unit Price	Amount

Document No.	Shipment Date	Description	Quantity	Unit Price	Amount
Document No.	Shipment Date	Description	Quantity	Unit Price	Amount

- 6 Save and close the report, and run it from the Object Designer. The example described gives this result with sample data:

No.	Name	Address	Phone No.		
Document No.	Date	Description	Quantity	Unit Price	Amount
<b>10000</b>	<b>Kontorforsyningen A/S</b>	<b>Carl Blochs Gade 7</b>	<b>11223344</b>		
941016	05/11/95	ANTWERP Conference Table	1	3,599.00	3,599.00
<b>20000</b>	<b>Ravel Mobler</b>	<b>Parkvej 44</b>	<b>22334455</b>		
941017	05/11/95	ST.MORITZ Storage Unit/Drawers	2	2,929.00	5,272.20
941017	05/11/95	INNSBRUCK Storage Unit/G.Door	1	2,500.00	2,250.00
941017	05/11/95	INNSBRUCK Storage Unit/W.Door	1	2,193.00	1,973.70
<b>30000</b>	<b>Lauritzen Kontormøbler A/S</b>	<b>Jomfru Ane Gade 56</b>	<b>33445566</b>		
941023	02/01/95	ANTWERP Conference Table	4	3,599.00	13,676.20
941023	02/01/95	BERLIN Guest Chair, yellow	23	1,265.00	20,366.50

## Chapter 11

### Extending the Functionality of Your Reports

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

## 11.1 GROUPING AND TOTALING

Grouping and totaling data are crucial to creating useful reports. By grouping and totaling data your reports can provide information that is not otherwise readily available.

The second report created in chapter 10 listed customers and those entries from the **Sales Line** table that pertained to each customer. You can use grouping and totaling to enhance this information in several ways.

First of all, if the report is to provide statistical information, it will be more useful to have the sales lines grouped according to the items (grouping on the item number, which identifies each item) instead of printing all lines from all sales documents. Each line should then provide 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 all in all.

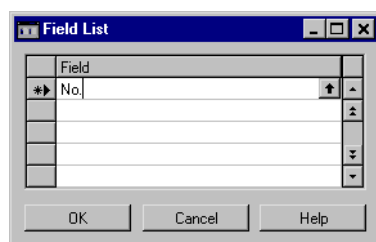
This is the report that will be created below.

### Defining the Data Model

The first steps in creating this report involve designing a report similar to the report created in Designing a More Advanced Report on page 186.

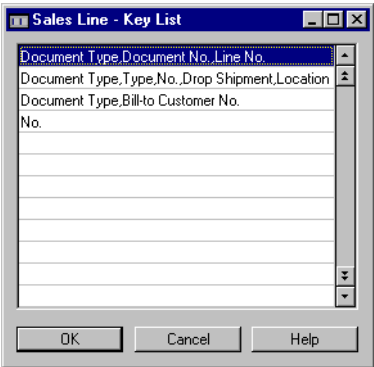
To add the grouping and totaling, follow these steps:

- 1 In the Property Sheet of Sales Line (the indented data item), enter as the value of the GroupTotalField property, the name of the field you want to be used for grouping the records. You can use the AssistButton ... to help you select the field. Here, the **No.** field is used:

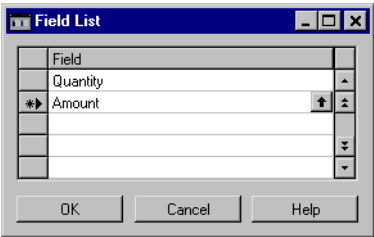


- 2 Use the AssistButton ... for the DataItemTableView property, and then select a key. You have to select a key that contains the field you want to group by.
- 3 If the key you select is a composite key, the grouping can fail if there are other fields in the key before the grouping field, and the contents of one of these fields change. In other words: 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 of the **No.** field only, was created for the **Sales Line** table:



- 4 Enter the names of those fields for which totals should be calculated as the value of the TotalFields property. You can use the AssistButton ... to help you select the fields. Here, the fields named **Quantity** and **Amount** are selected:



This data model is now defined. This is what has been accomplished:

- For each record in the Customer data item, those records in the Sales Line data item that are related to this customer are selected.
- The 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 the report being designed, a hierarchy of data items have been established, where Customer is the highest level data item and Sales Line is an indented data item. Further, the records of Sales Line will be grouped on the **No.** field, and totals will be calculated for the **Amount** and **Quantity** fields.

What, then, is the relationship between these totals and the sections – that is, how can these totals be printed?

Until now, only Header and Body sections have been used. To print totals, you will need to use some new sections. The table below gives an overview of all types of sections:

Section Name	Output
Header	Before a data item loop begins and (if the PrintOnEveryPage 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 PrintOnEveryPage property of the section is Yes), also on each new page. Moreover, if the PlaceInBottom 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	A page break occurs during a data item loop. Printed at top of the new page. This section is printed after a possible Header section of the data item.
TransportFooter	A page break occurs during a data item loop. Printed before the page break, This section is printed before a possible Footer section of the data item.

In order to print out the totals, you will need to use both a GroupFooter and a Footer section for the Sales Line (indented) data item.

In the GroupFooter section, the totals for **Quantity** and **Amount** will be for the defined group – 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 will be 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 will be printed. Therefore, this section can be used to print summary information about the customers.

That is, 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, while 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, of course, but



this information is not too useful, since it will be a total of quantities for all kinds of different items.)

**Note**  
.....  
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.  
.....

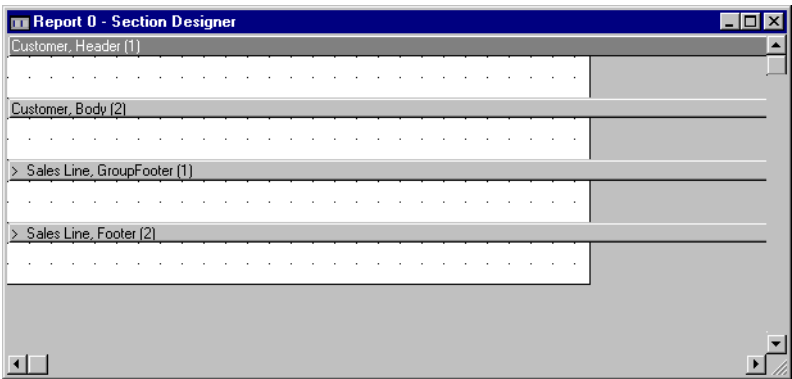
**Designing the Sections**

As usual, when you open the Section Designer, a Body section for each defined data item has been inserted.

To design the sections:

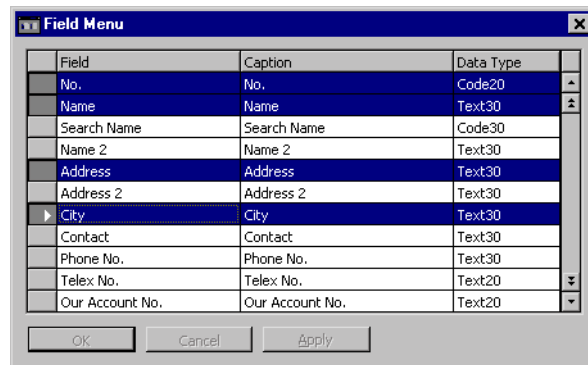
- 1 Add a Header section for the Customer data item. This section will be used to print headings for the columns in the report.
- 2 Add a GroupFooter section for the Sales Line data item. This section will be used to print the summary information about each item.
- 3 Add a Footer section for the Sales 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. Therefore, this section should be deleted. (You delete a section by clicking on the section bar, then choosing Delete from the Edit menu. You will be prompted to confirm the deletion.) The Section Designer will now look like this:

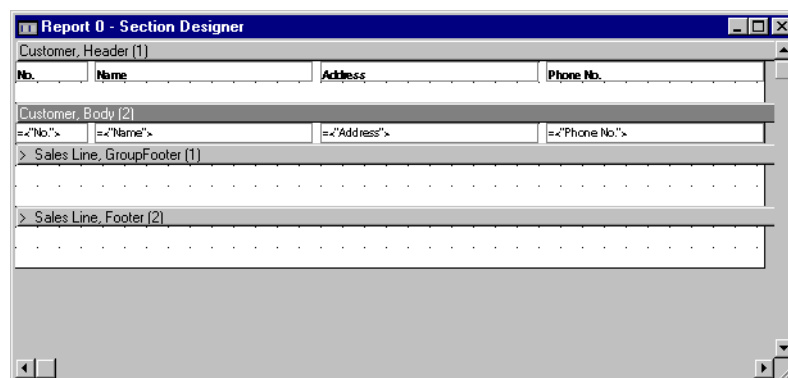


- 4 Select the Body section of the Customer data item (by clicking the section bar). Then open the field menu by choosing Field Menu from the View menu.

- 5 Select fields from the field menu. Here, four fields have been selected:



- 6 Click once in the Section Designer window to activate the window, then move the cursor into the Body section of the Customer data item. Click to insert text boxes and labels for the four selected fields.
- 7 Move the labels up into the Header section of the Customer data item and adjust the vertical size of the Body section. Resize the labels vertically and move them up to the top of the Header section. The Section Designer now looks like this:



- 8 Select the GroupFooter section of the Sales Line data item (by clicking the section bar) and open the field menu by clicking View, Field.
- 9 Select fields from the field menu. Here, the **No.**, **Description**, **Quantity** and **Amount** fields have been selected. Insert the fields in the GroupFooter section of the Sales Line data item. Delete the labels, and resize the section vertically.
- 10 Select the Footer section of the Sales Line data item. Insert the **Amount** field here and remove the label. Let the section have its default size – this way, there will be some empty space before each new customer.

11 Save, close and run the report.

So far, the report will look like this when printed:

No.	Name	Address		Phone No.
10000	Kontorforsyningen A/S	Carl Blochs Gade 7		11223344
1920-S	ANTWERP Conference Table		9	32,391.00
				32,391.00
20000	Ravel Møbler	Parkvej 44		22334455
1928-W	ST.MORITZ Storage Unit/Drawers		2	5,272.20
1964-W	INNSBRUCK Storage Unit/G.Door		1	2,250.00
1976-W	INNSBRUCK Storage Unit/W.Door		1	1,973.70
				9,495.90
30000	Lauritzen Kontormøbler A/S	Jomfru Ane Gade 56		33445566
1920-S	ANTWERP Conference Table		4	13,676.20
1936-S	BERLIN Guest Chair, yellow		23	20,366.50
				34,042.70

Obviously, this report needs some work before it looks good and is truly functional. For example, we need to devise a way to place captions for the columns from the indented data item. But 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 is to add a line at the end of the report where the grand total for all customers is printed. To do this, however, it is necessary to use C/AL code in a report trigger. The Advanced Sample Reports section on page 200 gives examples of how to do it.

## 11.2 TRIGGERS IN REPORTS

While 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 (depending, for example, on the value in a field, or a choice the user made in the request form). Perhaps the most important point about triggers, however, is that using them allows you to control how data is selected and retrieved in a more complex ways than using properties would.

### Report Triggers

These triggers pertain 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.

The table only describes the main purpose of each trigger. Refer to the online C/SIDE Reference Guide for concise descriptions and details.

### Data Item Triggers

The following triggers pertain 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.

The table only sketches out the main purpose of each trigger. Refer to the online C/SIDE Reference Guide for concise descriptions and details.

### Section Triggers

These triggers pertain 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.

The table only sketches out the main purpose of each trigger. Refer to the online C/SIDE Reference Guide for concise descriptions and details.

## 11.3 ADVANCED SAMPLE REPORTS

This final section will give examples of reports that are slightly more advanced than those previously described. The examples are not intended to be complete or ready to run, but are meant to give some ideas that you can use when designing reports.

### Using Virtual Tables

C/SIDE includes a number of *virtual tables*, such as the **Integer** table and the **Date** table. They are described in chapter 5, Special C/SIDE Tables. This section will show you how to use one of these tables, the **Date** table, in a report.

### Using the Date Table

The **Date** table consists of three fields, **Period Type**, **Period Start** and **Period End**. **Period Type** can be Date, Week, Month, and so forth, 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*.)

The **Date** table will be used to create a report that prints information from the **Cust. Ledger Entry** table (the **Customer Ledger Entry** table, but the word Customer has been abbreviated). For each day in a range of dates (that can be chosen by the user), the report summarizes entries made on that date. For each type of document (Invoice, Payment, and so forth), a line will be printed containing the number of documents of this type and the sum of the amounts on these lines. After each date, the total number of entries made on that day, along with the total amount of all these entries, will be printed. Finally, the total number of entries and the total amount for the selected date range will be printed at the end of the report.

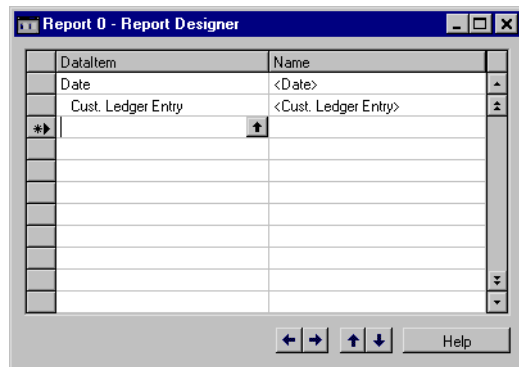
You could create the report by grouping according to the **Cust. Ledger Entry** table alone, but the field that contains the posting date in that table is not part of any key. Creating a special key just for this report is not desirable, because it would slow down all other transactions involving this table – in fact, all entries concerning sales would be affected.

### Defining the Data Model

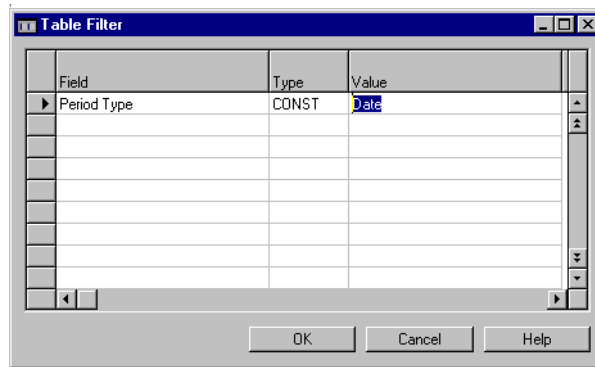
There are two data items in this data model: one that is related to the **Date** table and one that is related to the **Cust. Ledger Entry** table.

To define the data model:

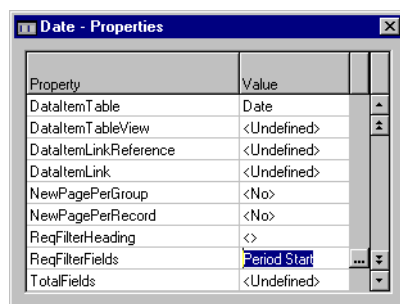
- 1 Open the Report Designer and create two data items with the **Date** and the **Cust. Ledger Entry** tables as the underlying tables. Indent the Customer Ledger Entry data item.



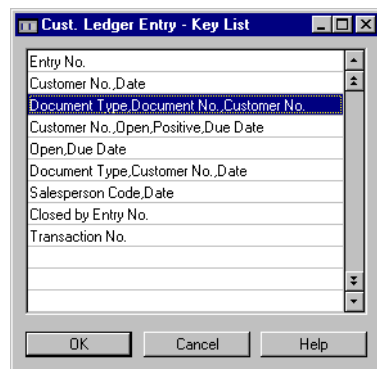
- 2 Open the Property Sheet for the Date data item. Then use the assist-edit button to help you set the value of the DataItemTableView property so that it selects records whose **Period Type** is Date. This is an important step, as the iteration of the Date data item would otherwise run through all records, including those for Weeks, Quarters, Months and Years.



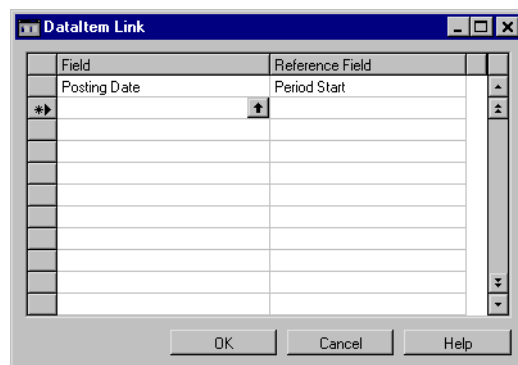
- 3 Enter the **Period Start** field as the ReqFilterFields property of the Date data item in order to let the user select a range of dates at run time.



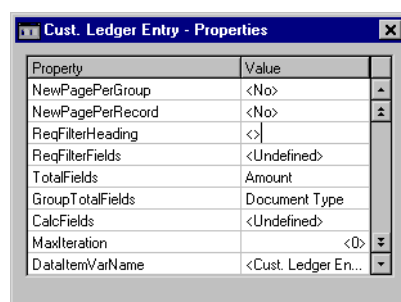
- 4 Open the Property Sheet for the Cust. Ledger Entry data item. Then use the assist-edit button to help you set the value of the DataItemTableView property to an appropriate key. An appropriate key in this case is a key containing the **Document Type** field, as the definition of a group will be based on this field. Here, a key is selected that has this field as its first component. As the report is going to show only summarized information, rather than all the entries, the other fields that are included in the key are not significant. (No individual entries will be printed, so they do not need to be sorted in any specific order.)



- 5 In the Property Sheet for the Cust. Ledger Entry data item, set the `DataltemLinkReference` property to point to the Date data item (this is the default). Then use the assist-edit button of the `DataltemLink` property to specify the field that establishes the link between the two data items. Choose the **Posting Date** field from the Cust. Ledger Entry data item, and the **Period Start** field from the Date data item.



- 6 Enter the **Document Type** field as the value of the `GroupTotalFields` property of the Cust. Ledger Entry data item, and enter the **Amount** field as the value of the `TotalFields` property.



Thus far, the report will work like this:

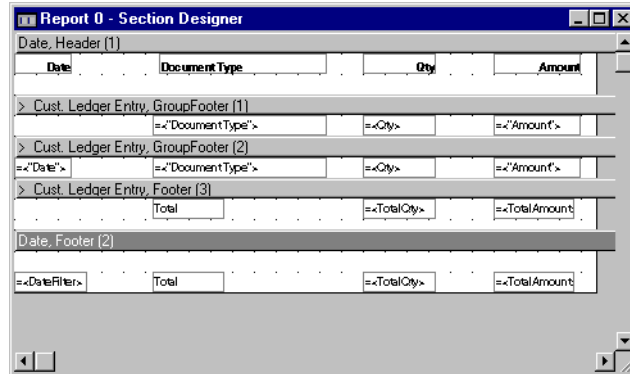
- The user can select a range of dates from the request form of the report.
- The report will run through the **Date** table, with a constant filter on the **Period Type** field that selects only records whose type is Date. If the user selected a range, only dates in the range will be selected; otherwise all dates will be used.



- For each selected date, records in the Cust. Ledger Entry data item that were posted on that date will be selected.
- The records of the Cust. Ledger Entry data item will be grouped according to the value of the **Document Type** field, and totals will be maintained for the **Amount** field.

## Designing the Sections

The design of the sections is fairly straightforward. The final screen looks like this:



The sections contain some controls that do not have fields from the data items as source expressions (Qty, TotalQty, TotalAmount and DateFilter). The purpose of these controls will be explained below.

Otherwise, the design of the sections is as follows:

- There is a Header section for the Date data item, containing captions for the columns of data in the report.
- There is a Footer section for the Date data item, used for printing totals for all printed records.
- There are *two* GroupFooter sections for the Cust. Ledger Entry data item, one with a text box for the **Date** field, one without. The reason for this construction—and how to use it—will be explained below. Both sections will print summarized information about the groups of this data item (remember that the **Document Type** field was used for grouping here).
- There is a Footer section for the Cust. Ledger Entry data item, used for printing totals.

Neither data item has a Body section.

## Refining the Design by Using Triggers

To make this report work as desired, we will need to write a small amount of C/AL code in triggers and to define a few variables. The report still needs four things to make it work the way we want:

- 1 The number of entries must be counted for each document type, for each date and for the complete range of dates in the report.

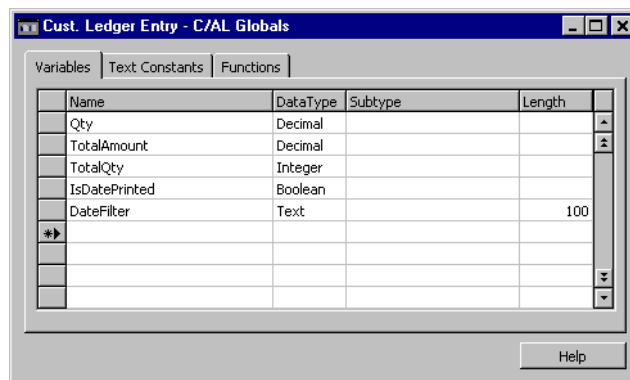
- 2 The total amount for all entries in the report must be calculated.
- 3 The date for a group of entries (with different document types) should be printed only once, when the first record in the group is printed.
- 4 At the end of the report, when the total number of entries and the total amount are printed, the date range that was selected by the user should be printed.

### Counting the Number of Entries

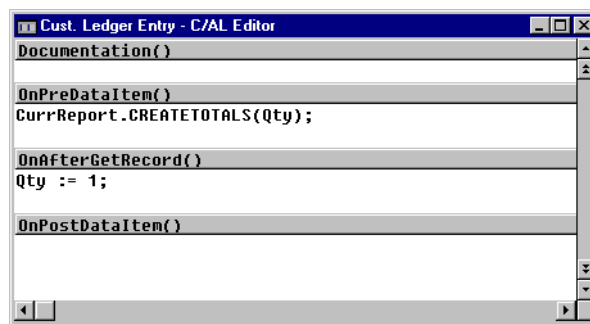
The records in the Cust. Ledger Entry data item contain an amount – which is totaled by using properties, as described in step 6 on page 202. The number of entries cannot be calculated in the same way, as no field in the data item record contains this information.

However, each record corresponds to exactly one entry. This means that the number of entries can be counted by simply counting the records. It can be done in this way

- 1 Create a global variable (here it is called Qty) of type Decimal.



- 2 Add the following C/AL code to the triggers of the Cust. Ledger Entry data item:



The statement in the OnPreDataItem trigger causes totals to be maintained for the Qty variable in the same way as when you use the TotalFields property to specify that totals will be 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.

The CREATETOTALS function will 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 will contain the sum of all entries with the same document type each time the Cust. Ledger Entry GroupFooter section is printed. When the Footer section is printed, Qty will contain the sum of all entries (that were selected, that is, all entries that pertain to the same date).

The argument of the CREATETOTALS function must be of the Decimal type (because the function will usually be used to sum amounts). Therefore, the Qty variable was declared to be of Decimal type rather than Integer (which perhaps would have been the intuitive choice). This means that the Qty text boxes in the sections have to be formatted *not* to show any decimal places, as they by default will have the format <2:2> when the SourceExpr is of the Decimal data type.

### Calculating the Total Amount and the Total Quantity

You want to print the total amount for all selected records from the Cust. Ledger Entry data item at the end of the report, in the Footer section of the Date data item. The value you want to print is, of course, the sum 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 to do the totaling.

Correspondingly, the total number of posted entries should be printed in the Footer section of the Date data item.

The solution is simple:

- 1 Declare two global variables: TotalAmount and TotalQty (refer to the picture on page 204).
- 2 Add these lines to the OnAfterGetRecord trigger of the Cust. Ledger Entry data item:

```
TotalQty := TotalQty + 1;
TotalAmount := TotalAmount + Amount;
```

The first line simply adds one to the TotalQty variable whenever a record is retrieved, while the second line adds the retrieved Amount to the TotalAmount. When the Date data item loop ends and the Footer section is printed, TotalQty and TotalAmount will contain the wanted values.

### Printing the Date in the First Iteration Only

For each iteration of the Date data item loop, you want to print the Date to which the information selected from the Cust. Ledger Entry data item pertains. You could, of course, just create a Body section for the Date data item and print the date there (the **Period Start** field of the data item), but this date would be printed on a line by itself. Instead, the date should be printed along with other information on the first line that comes from the Cust. Ledger Entry data item.

One solution is not to print a Body section for the Date data item at all, but to print the **Date** field from the Cust. Ledger Entry data item. This creates another problem, though: if it is added to the GroupFooter section, the date will be printed on every line. While this could be an easy way to solve the problem, the finished report will not be very attractive. Besides, it will be difficult to read the report if it is 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.

To do so, follow these steps:

- 1 Design two GroupFooter sections as shown in the picture on page 203.
- 2 Declare a global variable of type Boolean and call it `IsDatePrinted`.
- 3 Add the following line to the `OnPreDataItem` trigger of the Cust. Ledger Entry data item in order to initialize the `IsDatePrinted` variable before each iteration of the data item loop:

```
IsDatePrinted := FALSE;
```

- 4 Add the following lines to the `OnPreSection` trigger of the *first* GroupFooter section of the Cust. Ledger Entry section:

```
IF IsDatePrinted THEN
    CurrReport.SHOWOUTPUT(TRUE)
ELSE
    CurrReport.SHOWOUTPUT(FALSE);
```

- 5 Add the following lines to the `OnPreSection` trigger of the *second* GroupFooter section of the Cust. Ledger Entry section:

```
IF IsDatePrinted THEN
    CurrReport.SHOWOUTPUT(FALSE)
ELSE BEGIN
    CurrReport.SHOWOUTPUT(TRUE);
    IsDatePrinted := TRUE;
END
```

What happens is:

- 1 When a new iteration of the Cust. Ledger begins, a date has not yet been printed.
- 2 If the loop generates any output at all, only the second GroupFooter section (containing the **Date** text box) will be included as output in the first iteration.
- 3 If additional output is generated, only the first GroupFooter section (without the **Date** text box) will be printed.

### Printing the Selected Range of Dates

The final touch to the report is to add a line at the end of the report that shows the total number of entries and the total amount of these entries. How to calculate the figures has already been described. The posting dates are used as a kind of header in the left margin of the report, however, so the report would look good if the final line of this header could display the range of dates that the user selected. This is easy to implement:

**1** Create a variable of type Text, with a length of 100, and call it DateFilter.

**2** Add the following line 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 that has DateFilter as source expression.

When the OnPreReport trigger is executed, the RequestForm will already have been run. The GETFILTER function returns any filters on the field that is passed as an argument as a text string.

### The Final Report

The report will look like this with sample data:

Date	Document Type	Qty	Amount
01/15/95	Payment	3	-1,555,252.48
	Invoice	1	1,906.25
	Credit Memo	1	-2,640.00
	Total	5	-1,555,986.23
01/16/95	Invoice	1	10,101.25
	Total	1	10,101.25
01/17/95	Credit Memo	1	-6,950.00
	Total	1	-6,950.00
01/18/95	Invoice	1	74,658.58
	Total	1	74,658.58
01/19/95	Invoice	1	12,162.65
	Total	1	12,162.65
01/20/95	Invoice	2	420,327.98
	Credit Memo	2	-12,109.57
	Total	4	408,218.41
01/22/95	Invoice	1	27,027.30
	Total	1	27,027.30
01/23/95	Payment	1	-17,095.25
	Invoice	2	149,167.22
	Credit Memo	1	-2,230.00
	Total	4	129,841.97
01/15/95..01/23/95	Total	18	-900,926.07

## Creating a Simple Document

This section will describe how to create a document by using the Report Designer. The example is a skeletal sales invoice, that is, an invoice that does not take the complexities of VAT calculations into account and does not test for a number of conditions that will 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 posted sales invoice, while the **Sales Invoice Line** table contains the individual lines that are part of each invoice. The tables are related through a field that is called **No.** in the header table (and is the primary key of this table) and **Document No.** in the lines table.

In order to define the data model, follow these steps:

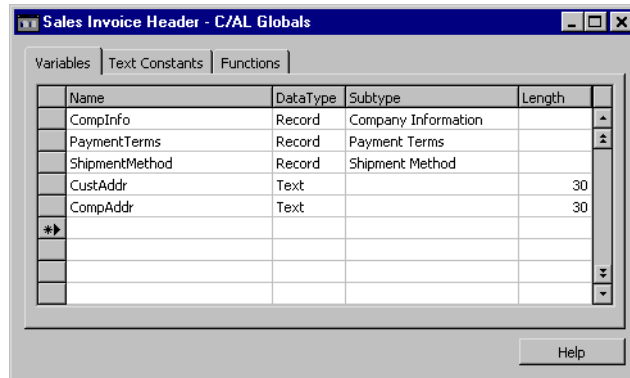
- 1 Create a data item, based on the **Sales Invoice Header** table.
- 2 Create another data item, based on the **Sales Invoice Line** table, and indent this data item one level.
- 3 By default, the `DataItemLinkReference` of the Sales Invoice Line data item points to the Sales Invoice Header data item. Leave it like this, and set the value of `DataItemLink` property to `Document No.=FIELD(No.)`.
- 4 Enter the **Amount** field as the value of the `TotalFields` property of the Sales Invoice Line data item, in order to calculate the total amount for all lines on the invoice.
- 5 Finally, in order to let the users of the report select a posted invoice to print, enter the **No.** field as the value of the `ReqFilterFields` of the Sales Invoice Header data item.

This completes the definition of the data model itself. In this report, some supporting variables are needed in order to access information from tables that cannot be fitted into the data model.

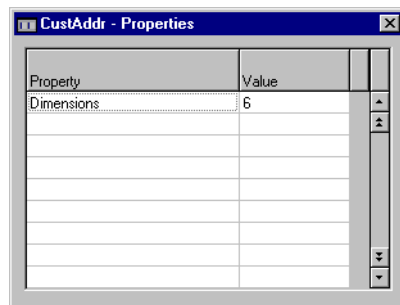
To create the variables, follow these steps:

- 1 Choose C/AL Globals from the View menu. This will open the form where you can declare variables.

2 Declare the variables like this:



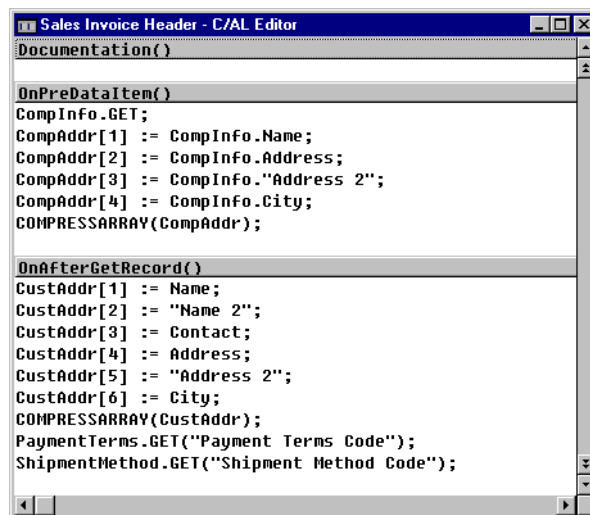
3 The two last variables must be declared as arrays. Open the Property Sheet for each variable and set Dimensions to 6 for the variable called CustAddr, and to 4 for the variable called CompAddr.



This concludes the definition of the data model. Next, a small amount of C/AL code must be added to the report triggers.

## Using the Triggers

This report, in this basic version, needs a very limited amount of C/AL code in order to function. The picture below actually contains all the code that is needed:



The entire code is in the triggers of the first data item, Sales Invoice Header. In the OnPreDataItem trigger, the statements work like this:

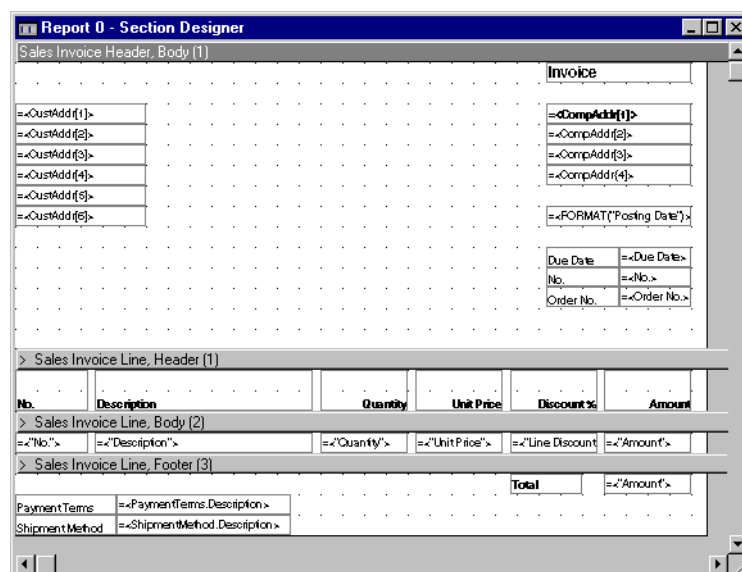
- The first line, `CompInfo.GET`, retrieves a record – in fact, the only record – from the **Company Information** table.
- The next four lines assign the contents of a field in the record in the **Company Information** table to an element of the `CompAddr` array.
- The final line of that trigger uses the `COMPRESSARRAY` function with the `CompAddr` array as an argument, in order to eliminate empty elements from the array. The reason for doing this is that you cannot be certain that all fields in the retrieved record have values assigned. 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` 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 for the reasons described above.
- 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 picture below 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 these points:

- Six text boxes have been inserted with CustAddr[1]..CustAddr[6] as source expressions. If you compare it with the document reproduced below, you will see that in this particular invoice, only four of these array elements contain data. Using COMPRESSARRAY has moved the data up, so to speak.
- Likewise, in the invoice shown below 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 January 19, 1995.
- In the Footer section of the Sales Invoice Line data item, the **Amount** field is a totaled field, containing the total of all amounts printed in the Body sections.
- In the same section, the full text descriptions of **Payment Terms** and **Shipment Method** are printed using `PaymentTerms.Description` and `ShipmentMethod.Description` as source expressions, respectively.

This is how the invoice document looks when sample data is used:

		<b>Invoice</b>			
Englunds Kontorsmöbler AB		CRONUS International Inc.			
Box 3319		5 The Ring			
Kungsgatan 18		9999 Kugleby			
600 03 Norrköping					
		January 19, 1995			
		Due Date	01/31/95		
		No.	943028		
		Order No.	941013		
<b>No.</b>	<b>Description</b>	<b>Quantity</b>	<b>Unit Price</b>	<b>Discount %</b>	<b>Amount</b>
1952-W	OSLO Storage Unit/Shelf	1	1,589.62771	15	1,351.19
1928-W	ST.MORITZ Storage Unit/Drawers	2	3,431.11242	15	5,832.89
1964-W	INNSBRUCK Storage Unit/G.Door	2	2,928.56984	15	4,978.57
<b>Total</b>					12,162.65
Payment Terms		Current month			
Shipment Method		Cost Insurance and Freight			

## A Nonprinting Report

To complete the examples, we will create 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 (prompting for options, setting filters) is achieved with little effort, while recreating this functionality in a codeunit is a considerable task.
- Using the features of the Report Designer to prompt for options and to set filters ensures consistency, not only in the application that you are creating, but also with the Navision application and add-on products.
- Instead of writing C/AL code to open tables and retrieve records, you just define a data item.

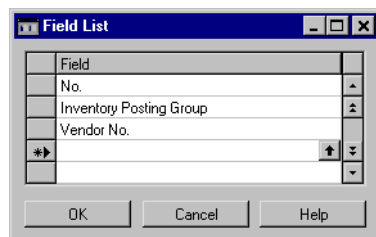
The report we will create is a simple one: it adjusts prices in the Item table. The user can set filters on some of the fields in the table in order to select a range of items by number, by posting group or by vendor, and, of course, can choose the factor to adjust the prices by.

## Defining the Data Model

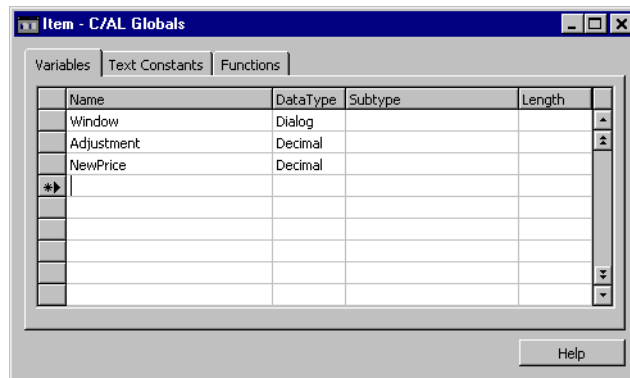
This report has one data item, based on the Item table.

To define the data model:

- 1 Create a data item based on the Item table.
- 2 Set the value of the ProcessingOnly property of the report to Yes.
- 3 Set the value of the DataItemTableView property of the Item data item to **No**. (by using the assist-edit button). Though this is not strictly necessary for the functionality of the report, it does serve one purpose: it removes the Sort ... button from the request form that will be presented to the user of the report. As the report will not print anything, the order in which data items will be run through is irrelevant.
- 4 Select the fields that the user will be able to filter, by using the assist-edit button in the ReqFilterFields property:



5 Declare three variables, like this:



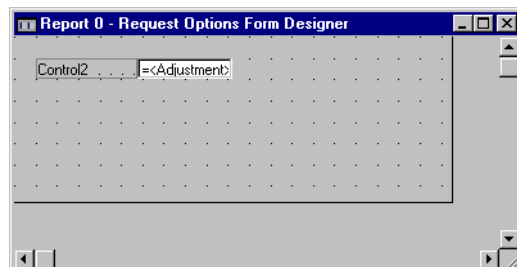
- The Window variable, declared as a Dialog type, will be used for printing a message on the screen while the report runs.
- The Adjustment variable will be used for the value that the user enters in the request form.
- NewPrice will be used to store an intermediate result.

### Creating the Request Form

Step 4 of Defining the Data Model above has already taken care of creating a request form with a tab where the user can set filters on some of the fields of the data item. You must add an Options tab, where the user can define the adjustment factor.

To create an Options tab:

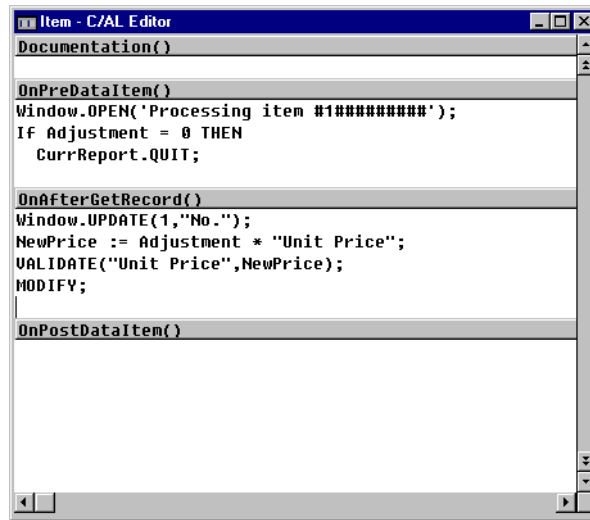
- 1 Open the Request Options Form Designer by choosing Request Form from the View menu.
- 2 Add a text box with a label to the form (to have the label added automatically, press the Add Label button in the Toolbox before selecting the Text Box tool).
- 3 In the Property Sheet of the text box, set the source expression to Adjustment, the newly created variable.



### Using the Triggers

Now that you have defined the data model and designed the request form, you must add a small amount of C/AL code to the triggers of the Item data item in order to

perform the actual price adjustment. The picture below shows all the C/AL code that is necessary:



The code functions like this:

- The first statement in the OnPreDataItem trigger opens a window, intended to show the progress of the report as it is run. (Because the report is non-printing, the usual window that shows printing progress is not shown. If the table is very large, the report may run for a while. Therefore, it is a good idea to indicate to the user that something is actually happening.)
- 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 causes the report to end without doing any processing if the adjustment factor is 0 (zero). If the adjustment factor were allowed to be zero, then all prices in the table would be set to zero, which would certainly never be the intention. The statement used here is a very crude way of handling this situation: in a more polished version, the user should, for example, have an opportunity to reenter the adjustment factor (or at least be notified of the reason for quitting the report run).
- 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 other fields that are related to this field) in the OnValidate trigger of the table field will be performed. Finally, the MODIFY function is used to commit the change.





## Chapter 12

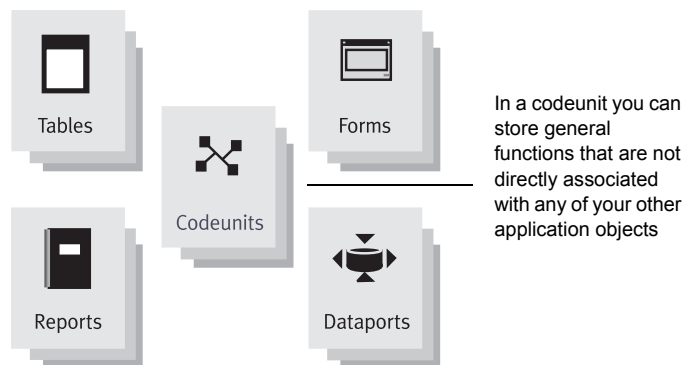
### Codeunit Fundamentals

This chapter explains what a 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

## 12.1 WHAT IS A C/SIDE CODEUNIT?

In the previous parts of this book you have seen examples of C/AL code in forms and reports. This code was always stored in a form or report object. In simple applications the normal approach is to place the code in the object that calls the functions, but as your application grows you will often find that you use the same functions repeatedly. 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. Think of a codeunit as a container for C/AL code that you want to use in many application objects.



In codeunits you can define:

**Functions** A function is a sequence of C/AL statements, which you define in order to create new functionality. Within each function you can define local variables, that is, variables whose scope is limited to the function in which they are defined.

**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 mainly used as structured variables that hold data temporarily while you work on it. Refer to *Defining and Using a Temporary Table* on page 72 for a description of how to create a temporary table.

Each function you add to a codeunit will be 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('Initializing company...');

AccountingSetup.INIT;
AccountingSetup.INSERT;

IF NOT (SourceCodeSetup.FIND('-') OR SourceCode.FIND('-')) THEN
    WITH SourceCodeSetup DO BEGIN
        INIT;
        InsertSourceCode(Sales, 'SALES', 'Sales');
        InsertSourceCode(Purchases, 'PURCHASES', 'Purchases');
        InsertSourceCode('Inventory Post Cost', 'INUTPCOST', 'Inventory Post Cost');
        InsertSourceCode('Exchange Rate Adjmt.', 'EXCHRATEADJ', 'Exchange Rate Adjmt. (Report)');
        InsertSourceCode('Post Recognition', 'POSTRECOG', 'Post Recognition (Report)');
        InsertSourceCode('Post Value', 'POSTVALUE', 'Post Value (Report)');
        InsertSourceCode('Close Income Statement', 'CLSINCOMESTAT', 'Close Income Statement (Report)');
        InsertSourceCode('Consolidation', 'CONSOLID', 'Consolidation (Report)');
        InsertSourceCode('General Journal', 'GENJNL', 'General Journal (Form)');
        InsertSourceCode('Sales Journal', 'SALESJNL', 'Sales Journal (Form)');
    END;
```

When you add your own functions they will be shown here

All codeunits include two default sections called Documentation and OnRun. In the Documentation section, you can add optional information about the code such 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 will be run.

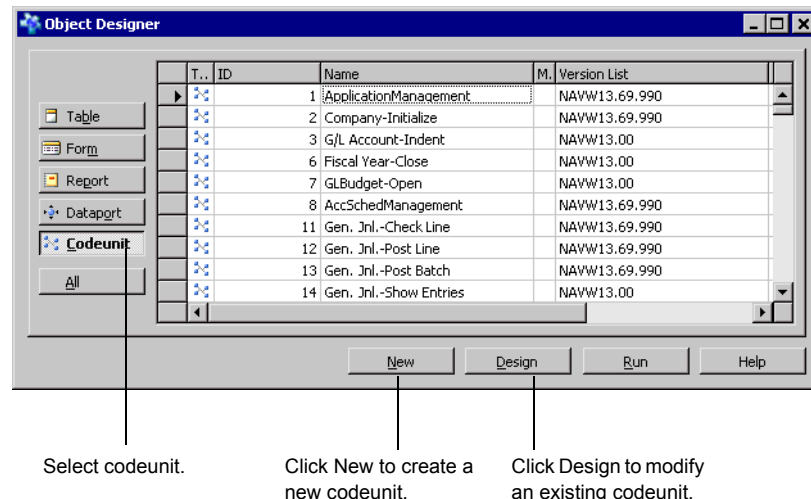
.....

## 12.2 CREATING CODEUNITS

You create a new codeunit or modify an existing codeunit in the same way you create and modify other application objects, that is, by using the Object Designer.

To create a codeunit:

- 1 From the menu bar, choose Tools, Object Designer. C/SIDE will open the Object Designer.



- 2 Click the Codeunit button in the Object Designer.
- 3 Click New to create a new codeunit. C/SIDE will open the C/AL Editor, where you can create functions.

To modify an existing codeunit:

- 1 Click the Codeunit button in the Object Designer.
- 2 Select the codeunit you want to modify.
- 3 Click the Design button. C/SIDE will open the C/AL Editor, where you can modify the codeunit by changing existing functions or adding new functions.

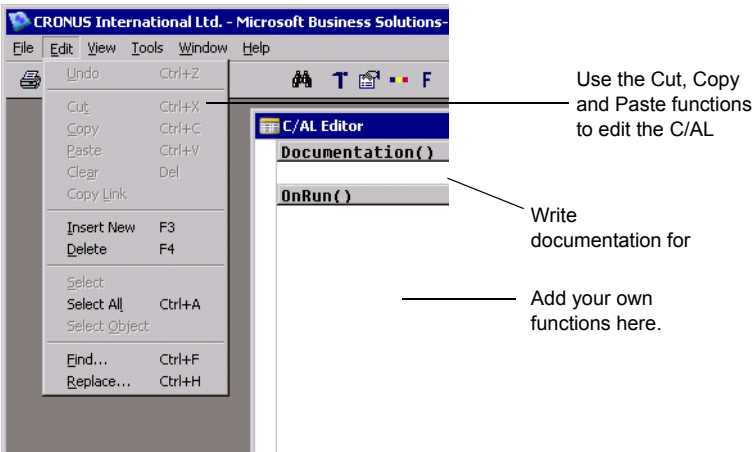
### Using the C/AL Editor

The C/AL Editor is where you view and edit your code. This editor is designed to make it easy for you to create and modify C/AL code. When you are in the C/AL Editor, you have access to the C/AL Symbol Menu that helps you define C/AL functions. When you use the C/AL Symbol Menu, you can get help about all 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 226.

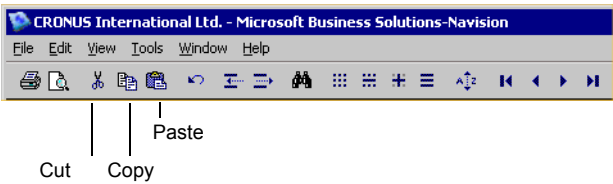
When you create a codeunit, the window shows the two default sections described above (the Documentation and the OnRun section).

From the Object Designer you can open as many codeunits as you like. Each time you create a new codeunit or open an existing one, it will be displayed in a separate window. This makes it easy to cut and paste lines of code between the codeunits.

If you have tried to use other Windows editors, you'll find the C/AL Editor easy to use. You can access the editing functions either from the Edit menu:



Or you can access the editing functions from the toolbar:



When you are working in the C/AL Editor, you can use a number of shortcut keys:

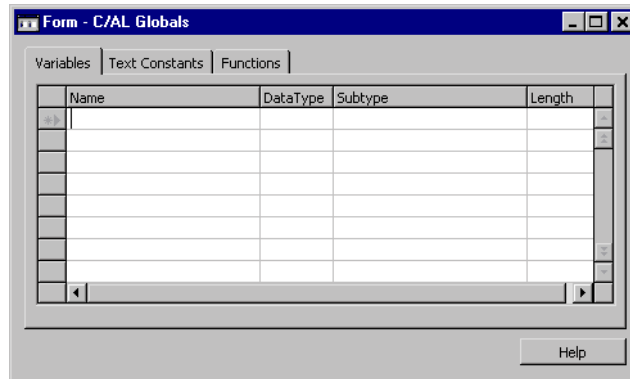
To...	Press...
cut the selected text to the clipboard.	CTRL+X
copy the selected text to clipboard.	CTRL+C
paste the text at the clipboard into the codeunit at the cursor position.	CTRL+V
open the <b>Find</b> dialog to search for trigger names.	CTRL+F

Defining Variables, Text Constants and Functions in Codeunits

When you have created a new codeunit, the next step is to define the global variables, text constants and functions you need in the codeunit. You use the C/AL Globals tool for this.

To access the C/AL Globals tool:

Make sure that focus is on the C/AL Editor. From the View menu, choose C/AL Globals. C/SIDE will display the **C/AL Globals** window:



In the **C/AL Globals** window, you select whether you want to add a global variable, a text constant or a function.

Global variables

To add a global variable:

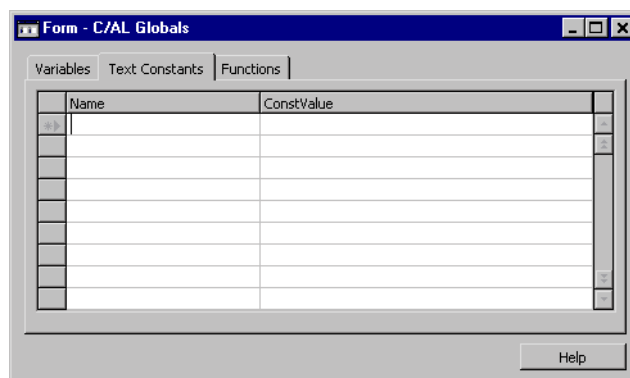
- 1 Click the Variables tab in the **C/AL Globals** window.
- 2 Add a name and a type. If the type you select corresponds to an application object, you also have to add a subtype, that is, the name of a specific object in the database. If you select text or code you have to define a length for the variable (the default length is 10 characters for code, and 30 for text). If you select OCX or Automation, you also have to add a subtype as described in the chapter *Extending C/AL*. Refer to Defining and Using a Temporary Table on page 72 for information about how to create temporary tables.

Text constants

To add a text constant:

When you are about to create a message for the user in the C/AL Editor you must do the following:

- 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**

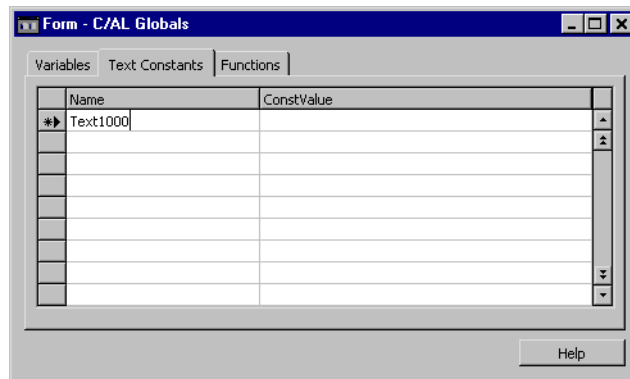
.....  
 There is no naming convention for the text constants. Using the unique ID for the name is a suggestion but not a requirement.  
 .....

- 3 Open the property sheet for 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 *Text1000*, if the ID number in the **ID** field is 1000.

The **C/AL Globals** window will look like this:



- 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 will represent.
- 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(Text1000);
```

Text1000 is an available number in the text constants number series for that object.

When you move the cursor over the new text constant, you will see its contents on the message line.

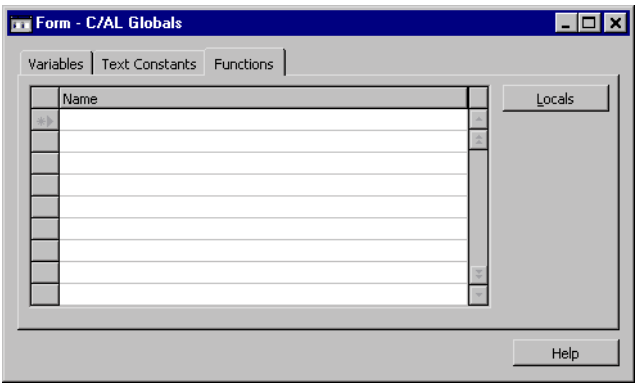
**Note**

.....  
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).  
.....

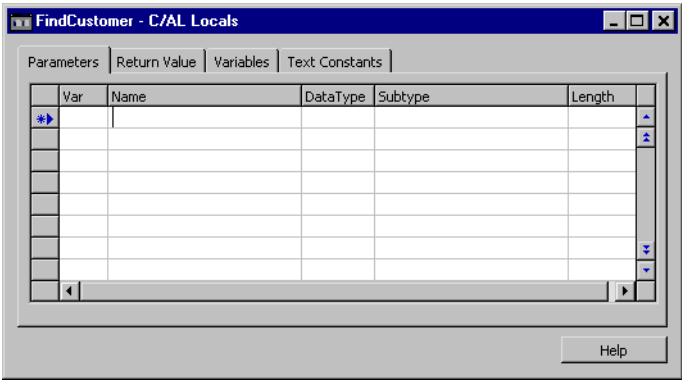
Functions

To add a function:

- 1 Click the Functions tab in the **C/AL Globals** window. C/SIDE will open the following window:



- 2 Enter a name for each function you want to add.
- 3 Click Locals to define the parameters, return value, local variables and text constants for each function. C/SIDE will display the **C/AL Locals** window:



- 4 For each parameter you have to specify the calling method, a name, and a data type. You can 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 affects only the copy and not the variable itself.

If the type you select corresponds to an application object, you also have to 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. C/SIDE will display:

The screenshot shows the 'FindCustomer - C/AL Locals' dialog box with the 'Return Value' tab selected. The dialog has four tabs: 'Parameters', 'Return Value', 'Variables', and 'Text Constants'. Under the 'Return Value' tab, there are three input fields: 'Name' (with a text entry field), 'Return Type' (with a drop-down arrow), and 'Length' (with a text entry field). A 'Help' button is located at the bottom right.

- 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.

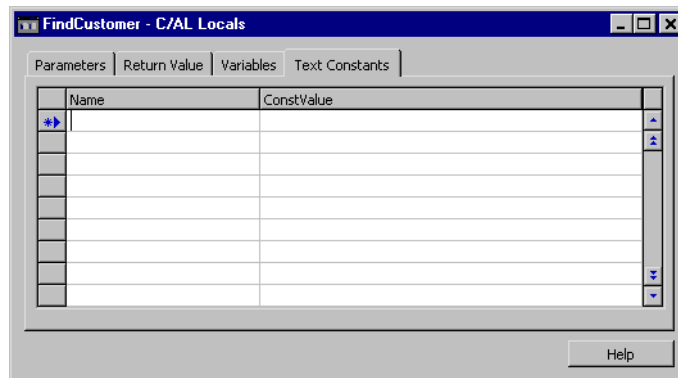
- 7 Click the Variables tab in order to define local variables. C/SIDE will display:

The screenshot shows the 'FindCustomer - C/AL Locals' dialog box with the 'Variables' tab selected. The dialog has four tabs: 'Parameters', 'Return Value', 'Variables', and 'Text Constants'. Under the 'Variables' tab, there is a table with four columns: 'Name', 'DataType', 'Subtype', and 'Length'. The table has several empty rows for adding variables. A 'Help' button is located at the bottom right.

Name	DataType	Subtype	Length

- 8 For each local variable, you must add a name and a type. If the type you select corresponds to an application object, you also have to 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 the variable (the default length is 10 characters for code and 30 for text).

- 9 Click the Text Constants tab in order to define text constants for the function.  
C/SIDE will display:



See page 222 about text constants for a description on how to fill in the **Name** field and the **ConstValue** field.

### 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 the following:

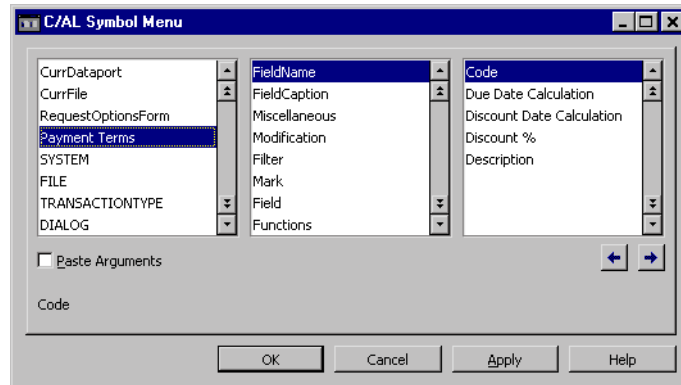
- All variables defined in the codeunit
- All C/AL functions

The information in the **C/AL Symbol Menu** window is divided into three or more columns:

- The column to the left shows variable names (if you have defined any) and function categories.
- The second column shows the subcategories.
- The third column shows the functions in the category you have 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 382.



Click OK or Apply to paste the syntax description into the editor

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. You can use the right-arrow button to scroll the symbol menu columns to the right and the left-arrow button to scroll them back.

Click OK or Apply to make C/SIDE insert this string in the C/AL Editor. If you click OK, the **C/AL Symbol Menu** window is closed automatically; if 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 online Help: C/SIDE Reference Guide.

## Compiling and Saving Codeunits

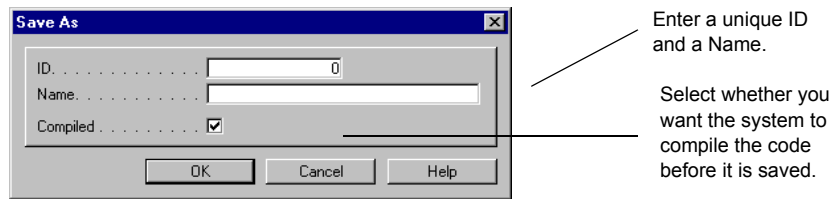
Before the functions in a codeunit can be run, the code has to be compiled and saved. When you compile the code, the system checks the syntax of the statements. If the compiler finds any errors in the code it will display a message.

To compile the code in a codeunit:

- 1 From the Tools menu, choose Compile.
- 2 If the system finds any errors in your code it will display a message. Correct the errors and choose Compile from the Tools menu again.

To save the codeunit:

- 1 From the File menu, choose Save. C/SIDE will display:



- 2 Enter an ID number and a name. The number is used as a unique identification, while the name serves as a label. If you save the codeunit without compiling it, you won't be able to run it or call any of its functions.

### Why Save without Compiling?

.....  
If you are working on a large and complicated codeunit, you will want to save you 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.  
.....

## 12.3 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 as:

```
<CodeunitName>.<FunctionName>
```

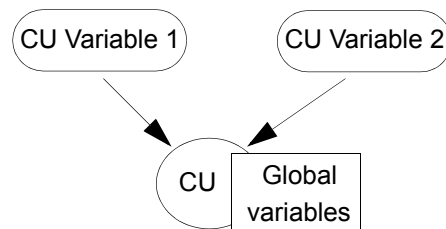
### EXAMPLE

Assume that you have created a codeunit containing two statistical functions named F and G. The illustration below shows how to access these functions from a form.

Codeunit named StatFun
F(x:integer) Begin ...  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 prefixing the function name with the name of the application object containing 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 may refer to the same codeunit as shown in the following figure:



Codeunits contain internal variables that are defined as global variables. These variables are not accessible 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.

#### EXAMPLE

Codeunit 50000 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.

```
VAR
    CoMIC1: Codeunit 50000;
    CoMIC2: Codeunit 50000;
//codeunit 50000 is a standard codeunit

BEGIN
    CoMIC1.Set(1);
    CoMIC2.Set(2);
//both codeunit variables are instantiated - they have different
//instances
    CoMIC1.Get();
    CoMIC2.Get();
//CoMIC1 returns 1; CoMIC2 returns 2
    CoMIC2 := CoMIC1;
//CoMIC2 is assigned to CoMIC1 and they now both use the same
instance
    CoMIC1.Get();
    CoMIC2.Get();
//both codeunit variables return 1
END;
```

**CLEAR on codeunits** When you use the function *CLEAR* 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 codeunit instance. 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 a codeunit instance, you must clear all references to the codeunit with the function *CLEAR*. If you need to clear the internal variables in a codeunit, you must call *CLEARALL* from a user-defined function within the codeunit. When a local codeunit variable goes out of scope, meaning that it is no longer used by the codeunit, it is automatically cleared.

Single Instance  
Codeunit

In some cases, the situation requires that only one instance of a codeunit exists. This means that all codeunit variables of a particular codeunit will use the same set of variables. By setting the *SingleInstance* property for the codeunit to *Yes*, all codeunit variables of that codeunit will use the same instance, thus allowing the developer to create global variables.

**Note**

.....

It is 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**

Codeunit 50001 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. Codeunit 50001 has the `SingleInstance` property set.

```
VAR
    CoSIC1: Codeunit 50001;
    CoSIC2: Codeunit 50001;
//codeunit 50001 is a single instance codeunit

BEGIN
    CoSIC1.Set(7);
//a codeunit instance is created if one did not exist
    CoSIC2.Get();
//returns 7 - CoSIC2 uses the same instance as CoSIC1, so //they use
the same internal variables
END;
```

**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.

All C/AL functions can be used in a codeunit. Notice, however, that you cannot create a function with the same name as a built-in function. Neither can two or more user-defined functions have the same name (unless they are part of different application objects).



## Chapter 13

### Introducing the C/AL Language

This chapter introduces the C/AL language. It describes how you can use the language to create functions, and it describes the syntactical rules 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

## 13.1 WHAT CAN YOU DO WITH C/AL?

The previous parts of this book have shown you how to design various database objects such as tables, forms and reports. But simply getting these individual objects up and running is not enough. To turn these objects into a coherent application you have to make the database objects work together. C/AL code is the glue that does this for you. When you are designing professional applications you will often need specialized functions. C/AL makes it possible to go beyond what C/SIDE does automatically. For example, you can create special functions for use anywhere in the database.

Here are the most important things C/AL lets you do:

**Design Your Own Functions** Although C/SIDE has a large number of intrinsic functions, you will sometimes find it convenient or necessary to add your own functions, for example, if 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.



## 13.2 WHAT ARE STATEMENTS, EXPRESSIONS, AND OPERATORS?

In this section the following terms are introduced:

- 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 table below 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. All valid C/AL expressions can be evaluated to a specific value.
:=	The assignment operator. When the right-hand side expression has been evaluated, this operator is used to assign (store) the value in the variable Amount.
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 of a C/AL function. Consider the C/AL statement below:

```
Date := DMY2DATE(31, 12, 1996);
```

This function takes three simple expressions as arguments, 31, 12 and 1996.

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 expressions in C/AL are built from:

- Constants
- Variables
- Operators
- Functions

Depending on the elements in the expression, the evaluation will result in a value with a C/AL data type. The table below 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.

The above 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.

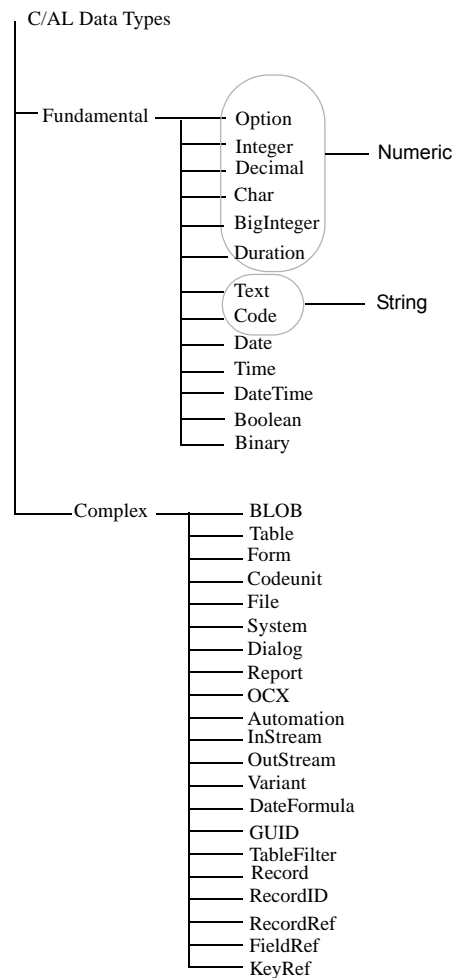
### Introducing the C/AL Data Types

As you have already seen, variables can be used to store data of various types. By declaring variables to be of the proper type, you:

- create faster code.
- save space.
- avoid runtime errors due to overflow or impossible type conversions.

For example, if you know that a variable will always contain an integer number between 0 and 700, you should use an integer variable instead of a decimal variable. All calculations 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 will have to use a data type that can hold all possible values 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

In C/AL, there are a number of fundamental data types, which are designed to store boolean values, numbers, text, time and dates.

**boolean** The values 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 statement below 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. Thus a closing date is 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** Use this data type to denote the date and 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 Navision. 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`

- Navision 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 with a number type variable.

#### 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 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 maximum 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 type char. The length of a variable of type text corresponds to the number of characters in the text. An empty text string thus has the length 0.

The table below illustrates some typical examples of text strings. In these examples it is assumed that the variable t is of 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 code type variable, the text is transformed to uppercase, and 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 type char. The maximum length of a code type variable ranges from 1 to 250 characters. The length of a code type variable always corresponds to the number of characters in the text without leading and trailing spaces.

#### EXAMPLE

The table below shows some typical examples of code string assignments. In the examples, it is assumed that the variable `c` has the type `code`, and the maximum length 4.

Assignment	The variable <code>c</code> now contains...	The length is...
<code>c := 'AbC';</code>	'ABC'	3
<code>c := '1';</code>	'1'	1
<code>c := '';</code>	" (empty string)	0 (zero)
<code>c := ' 2';</code>	'2'	1
<code>c := '1 2';</code>	'1 2'	3

### Descriptive Data types

In order to describe the syntax of the C/AL language, some descriptive data types are used. It is important to stress that these are not real system data types, but are used in the C/SIDE documentation for descriptive purposes only.

The table below 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 of 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 which 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 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 types Automation and OCX.

**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. For more information about this data type, see the online C/SIDE Reference Guide.

**OCX and Automation** See the chapter *Introducing the C/AL Language*, on page 299.

### **Creating Arrays of Variables**

It is possible to create 10-dimensional variables, using the simple and complex data types presented above. 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 will occur.

#### **EXAMPLE**

Assume that Foo is a one-dimensional array variable of the Integer type, 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 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].



### 13.3 INTRODUCING THE ELEMENTS OF C/AL EXPRESSIONS

The previous sections have introduced you to C/AL expressions and data types. The aim of this section is to present the basic elements of C/AL expressions. The following subsections will briefly discuss:

- Constants
- Variables
- Operators
- Functions

We start by defining ranges and properties of constants in C/AL.

#### 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 examples below, 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 will occur.  
 .....

**boolean constant** A boolean constant may 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 "." 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 table below illustrates different types of C/AL constants:

Constant	Description
TRUE	boolean constant
50000	integer constant
-23.7	decimal constant
122196D	date constant (December 21, 1996)
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 are ones you define when you create new C/AL code. You can define variables that are global to all functions within a codeunit and variables that are local to each 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.

In addition, a number of predefined variables are provided by the system. These variables are automatically maintained by the system and are called system-defined variables. 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.

During the execution of triggers and functions, 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

Variable names must be unique, that is, two user-defined variables with the same name are not allowed in a C/AL codeunit. Furthermore, you cannot have user-defined and system-defined variables with the same name. Uppercase and lowercase letters are not distinct, that is Smith and SMITH refer to the same variable. In standard Pascal notation, a variable name (an identifier) can only be written as an unbroken

word. This restriction is relaxed in C/AL: here it is also possible to use special characters (for example, spaces) in a variable name.

Observe the following basic restrictions:

- The maximum length of a variable name is 30 characters.
- A variable name must not be the same as the name of a C/AL function name 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 the following:

- Control characters (ASCII 0-31, 255)
- The character " (ASCII 34)

When naming 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, it is also possible to include one or more special characters (spaces, and so on) in a variable name in C/AL, but then the entire variable name must be put 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 a compile-time error.  
 .....

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 and code) 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 previously mentioned, 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

Assignment of values can be performed in one of two ways:

- As parameter assignment, for example FUNCTION(Expression). The resulting data type of the evaluation of the expression must correspond to a specific data type or have a type which can be converted automatically to the correct type. (For a detailed discussion about evaluation and type conversion in expressions, refer to the chapter Type Conversion on page 389.)
- By using the assignment operator ":=" (for example Variable := Expression). Generally, the resulting data type of the evaluation of the right-hand side expression must be of the same type as the variable (left operand) or have a type which can be converted automatically to the type of the left operand.

Automatic type conversion in assignments takes place when:

- A parameter in a function call does not have the correct type. This occurs 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 type that differs from the type of the variable on the left-hand side.

The automatic type conversion in assignments can freely take place between the following numeric data types, provided overflow does not occur:

**char ↔ integer ↔ decimal**

The automatic type conversion in assignments can also freely take place between the String data types:

**code**  $\longleftrightarrow$  **text**

All of the above has been 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.

**Note**

.....  
The type conversion that takes place in assignments can cause runtime errors even though the 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**

Let the variable A be 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 below:

```
A[2] := 'Enter your name';
```

**EXAMPLE**

Result is an option variable, while Amount and Total both are decimal variables. Consider the following assignment statements:

```
Amount := 10;  
Total := 4;  
...  
Result := Amount + Total;
```

The above code can always be compiled, but a runtime error will occur if the result of the right-hand side expression "Amount + Total" exceeds the valid range of 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
text	⦿	⦿
code	⦿	⦿

● THE ASSIGNMENT IS VALID

⦿ THE ASSIGNMENT IS VALID, BUT OVERFLOW MAY OCCUR

## 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 table below 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
/	Division
DIV	Integer division
MOD	Modulus
>	Greater than

C/AL operator	Meaning
>=	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 only as an unary operator, while all other operators are binary.

Most of the above operators can be used on different data types. The action of these operators may depend upon the data type of expression they are used on. Below 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
```

You can read more about the function of each operator in the chapter Type Conversion, on page 389, which explains the type conversion mechanisms in C/AL.

## Operator Hierarchy

The operators just discussed are organized in a hierarchy that determines the order in which the operands in a given expression will be evaluated. The following list shows the precedence order 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 example below illustrates the effect of the operator hierarchy. The expressions, which apparently are the same, will 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 has a number of functions for different purposes, such as string handling, text formatting, database handling and so on. Some of these functions differ from standard Pascal, as it is possible to use a variable 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 the function has, for instance, 3 optional parameters, then it is not possible to 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 which 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.

## 13.4 THE C/AL CONTROL LANGUAGE

This section describes the basic structures in the control language in C/AL and how to use them. All the C/AL programs 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. You may have to repeat the execution of one or more statements 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

### Using Compound Statements

In some cases, the C/AL syntax will only allow use of a single statement. If you have to execute more than one simple statement in such a case, the statements can be turned into a compound statement, by "encapsulating" 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 above `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 to 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

This statement type has 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 to be executed, depending on the evaluation of the <Condition>.

It is possible to 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.

Several nested IF THEN ELSE statements may seem 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

Illustration of 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 for the C/AL newcomer is to put an extraneous semicolon at the end of a line before an ELSE (line 4). As mentioned above, 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 the above definition, the `<Expression>` cannot be a record, and the `<Value set>` must be an expression or a range.

CASE statements are also called multiple option statements and are typically used when a selection between more than two different actions is to be made. The function of the CASE statement is as follows:

- The `<Expression>` is evaluated, and the first matching value set causes the associated statement, if any, to be executed.
- If none of the value sets matches the value of the expression, and the `ELSE` part has been omitted, no action will be taken; but if the optional `ELSE` part is used, then the associated statement will be executed.

The type of the value sets must be the same as the type of `<Expression>` or at least convertible to the same type.

### Note

.....

The data type of the value sets will be converted to the same data type as the evaluated `<Expression>`, if necessary. This type conversion may cause an overflow at run time if the resulting data type cannot hold the values of the value sets.

.....

### EXAMPLE

This C/AL code sample will print 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 will be used as 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 while 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>
```

<Control Variable>, <Start Number> and <End Number> must be boolean, number, time or date data types.

FOR statements are used when a code block is to be executed a specific number of times. A control variable is used to control the number of times the code block is executed. The <Control Variable> may be increased or decreased by one, according to whether TO or DOWNTO is used.

### When declaring the type of the <Control Variable>...

.....  
it should be noticed that when the system executes the FOR statement, the <Start Number > and <End Number> will be converted to the same datatype as <Control Variable>, if necessary. This type conversion may 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

The following FOR loop uses an integer control variable named Count.

```
FOR Count := 1000 TO 10000000000000000 DO
```

When the above statement is executed, a runtime error will occur because the system tries to convert the start and end values to the same type as the control variable; but as Count has been

declared as an Integer variable, an error will occur when 10000000000000000 is to be converted, because this end value is outside the valid range for Integers.

**EXAMPLE**

This example illustrates nesting of FOR statements:

```
FOR I := 1 TO 5 DO
    FOR J := 1 TO 7 DO
        A[I,J] := 23;
```

The two FOR statements above could be used to initialize every element in a 5 x 7 array with the value 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.

When a block of code is to be repeated as long as an expression is TRUE, the WHILE DO statement may come in handy.

**EXAMPLE**

The C/AL code below increases the variable i until it equals 1000:

```
WHILE i < 1000 DO i := i + 1;
```

**The REPEAT UNTIL Control Structure**

The syntax for the REPEAT UNTIL statement is:

```
REPEAT <Statements> UNTIL <Condition>
```

<Statements> will be executed repeatedly until <Condition> is TRUE.

This might at first glance seem to function just like a WHILE control structure, but as the REPEAT UNTIL statement is executed from left to right, it is easily seen that the <Statements> always will be 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—implying that if the first evaluation of <Condition> returns FALSE, then no statements will be executed.

**EXAMPLE**

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;

```

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 and the system exits the loop.

### 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 will take place even when the execution is inside a loop or a similar structure. The EXIT statement is also used when a local function is to 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 will occur 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 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 for the 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, it is only necessary to specify the record name once.

Within the scope of <Statement>, fields in <Record> may be addressed without specification of the record name.

Several nested **WITH** statements may be used. In case of identical names, the inner **WITH** will overrule the outer **WITH**-statements.

**EXAMPLE**

This example shows two ways of writing the same code:

```
CustomerRec.No := '1234';
CustomerRec.Company := 'Windy City Solutions';
CustomerRec.Manager := 'Joe Blow';
CustomerRec.Address := '1241 East Druid Avenue';
CustomerRec."State and Zip" := 'Chicago, IL 60079';
```

Another way of expressing the same is:

```
WITH CustomerRec DO
BEGIN
    No := '1234';
    Company := 'Windy City Solutions';
    Manager := 'Joe Blow';
    Address := '1241 East Druid Avenue';
    "State and Zip" := 'Chicago, IL 60079';
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.

Any number of nested comments may occur. 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! *
```

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 14

### 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 may or may not 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

## 14.1 OVERVIEW

This chapter describes how to use C/AL. The first sections concentrate on giving some advice on the things you should consider when you use C/AL—the most important subject being where you place the code.

The concepts of system-defined variables and runtime errors are explained, and the final, larger, section describes and gives examples on how to use a subset of C/AL functions – a subset that experience has shown will be the set of functions that developers will use most often.

### Where to Write C/AL Code

As described in previous chapters, almost every object in C/SIDE has triggers where C/AL code can be placed. In summary, you have triggers for:

- Tables
- Table fields
- Forms, including request options forms
- Form controls
- Reports
- Data items
- Sections

The execution of C/AL can also be initiated 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 above.

As you can see, you can put C/AL code in a large number of places and initiate or trigger its execution in many ways. You should not, however, choose a location for your C/AL code at random. A few simple guidelines should be followed:

- In general, place the code as close as possible to the object it operates on. This implies that code that modifies records in the database should normally be placed in 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 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 controls or whether the section should be printed. On the other hand, you should never place data-manipulating code 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. Users do not have direct access to tables with sensitive data – such as the general ledger entry and register tables. 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 the right 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 other reasons than security for putting a posting function like the one described in the item above in a codeunit. A function that is placed in a codeunit can be called from many places in the application – perhaps including 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 it faster and easier to develop applications, but this alone 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.

## 14.2 SYSTEM-DEFINED VARIABLES

C/SIDE automatically declares and initializes a number of variables for use in application development. These are the system-defined variables

Variable	Comments
Rec	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).
xRec	
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

You could put the Rec/xRec pair of records to use in a situation like this: 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. On the form where the user enters information in the header table there are fields that contain the customer's address. These fields are related to a **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 any further.

You could do something along these lines:

- When validating the customer number field, check whether the order has been shipped.
- If it has, 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.

## 14.3 HANDLING RUNTIME ERRORS

In the chapter on debugging (chapter 15, Debugging C/AL Code), the section Other Runtime Errors on page 284 describes how to handle functions that return a boolean value that can be processed or ignored.

When you use these functions, four different scenarios are possible, as depicted in this table:

	Return value is ignored	Return value is processed
Function succeeds	Execution continues	Execution continues
Functions fails	A runtime occurs	Execution continues, and you must handle the situation yourself

A typical example of a function that will or will not 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 will be TRUE if the record is found and FALSE otherwise. If GET is used as below, and no record is found,

```
Customer.GET("Customer Number");
```

a runtime error will occur. If, on the other hand, GET is used as below, and no record is found,

```
IF Customer.GET("Customer Number") THEN
    ....
ELSE
    ...
```

execution will continue. In this case you will need to handle the situation yourself in the ELSE part of the statement.

## 14.4 THE ESSENTIAL C/AL FUNCTIONS

Although there are more than 100 functions in C/AL, you will find that you use a limited set of these functions repeatedly, while you use the rest of the functions only occasionally. That is to say: during basic application development you use perhaps no more than 20 different functions. This does not mean that the rest of functions are obsolete or that you will never use them – but it does mean that if you are comfortable with this set of essential functions, you will be able to go a long way in C/AL programming. As you need to add more specialized functionality to your applications—or you want to round them off by adding "bells and whistles"—you can familiarize yourself with the full set of functions.

Below are some examples of how to use this set of essential functions. You should, however, always refer to the online C/SIDE Reference Guide for full and updated information on any C/AL function.

### Searching For Records

The three functions described in this section are used to search for records. When you are going to search for records, it is important to remember the difference between `GET` and `FIND` – and how you can use `FIND` and `NEXT` in conjunction.

`GET` retrieves one record, based on the value of the primary key. That is, if the **No.** field is the primary key of the **Customer** table, `GET` can be used like this:

```
GET(Customer, '4711');
```

The result will be that the record of customer 4711 will be retrieved. `GET` is one of those functions that will produce a runtime error if it fails and the return value is not inspected by the code, and otherwise not. This means that your actual code would probably look more like this:

```
IF GET(Customer, '4711') THEN
    .... // do some processing
ELSE
    .... // do some error processing
```

`GET` searches for records, regardless of current filters, and it does not change any filters. In other words: `GET` always searches among all records in a table.

`FIND` An important difference between `GET` and `FIND` is that `FIND` respects (and is limited by) the current setting of filters. Further differences are that `FIND` can be instructed to look for records where the key value is equal to, larger than or smaller than the search string, and finally, `FIND` can find the first or the last record (given the sorting order defined by the current key).

You can use these features in various ways. When developing applications under a relational database management system, you will often have one-to-many relationships between tables. An example could be the relations between an **Item** table, which registers items, and a **Sales Line** table, which registers the detail 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 deleted while there are still open sales order records that include the item. You can use `FIND` to check this.

To do this, insert the following code in the OnDelete trigger of the Item table:

```
SalesOrderLine.SETCURRENTKEY("Document Type",Type,"No.");
SalesOrderLine.SETRANGE(
    "Document Type",SalesOrderLine."Document Type"::Order);
SalesOrderLine.SETRANGE(Type,SalesOrderLine.Type::Item);
SalesOrderLine.SETRANGE("No.", "No.");
IF SalesOrderLine.FIND('-') THEN
    ERROR(
        'You cannot delete because there are one or more outstanding
        sales orders that include this item.');
```

`NEXT` `NEXT` is often used with `FIND` to step through records of a table, 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

These 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. You will also find a description of how to change the *sorting* of the records in a table.

`SETCURRENTKEY` This function is used to select a key for a record, thereby setting the sorting order that will be used for the associated table. `SETCURRENTKEY` has this syntax:

```
[Ok :=] Record.SETCURRENTKEY(Field1, [Field2],...)
```

You should have these points in mind when you use `SETCURRENTKEY`:

- 1 Fields that are not active are ignored
- 2 When searching for a key, C/SIDE selects the first occurrence of the specified field(s).

For example, even if you specify only one field as a parameter when calling `SETCURRENTKEY`, the key that is actually selected may consist of more fields; if several keys have as their first component the field that you specified, you may not get the key that you think you will.

If no keys can be found that include the specified field(s), a runtime error will occur unless you test the boolean return value of `SETCURRENTKEY` in your code. If you do test the return value, you will have to decide what to have the program do if the function returns `FALSE`, because without a runtime error, the program will continue to run even though no key has been found.

**SETRANGE** This function is used to set a delimitation on a field – that is, a simple filter. The syntax is:

```
Record.SETRANGE(Field [,From-Value] [,To-Value]);
```

as in this example:

```
Customer.SETRANGE("No.", '10000', '90000');
```

which would limit the **Customer** table by selecting only those records where the **No.** field has a value between 10000 and 90000.

`SETRANGE` will remove previous filters. If used without the From-Value/To-Value parameters, the function can be used to remove any filters that might already be set. And, finally, if only From-Value is used, To-Value will be set to the same value as From-Value.

**SETFILTER** `SETFILTER` sets a filter in a more general way than `SETRANGE`. `SETFILTER` has this syntax:

```
Record.SETFILTER(Field, String [, Value], ...);
```

where **Field** is the name of the field to set a delimitation on. **String** is a filter expression that may contain %1, %2 and so on to indicate locations where the system will insert values (but not operators) given as the **Value** parameter(s) in a filter expression.

Here are two examples of using `SETFILTER`:

```
Customer.SETFILTER("No.", '>10000 & <> 20000');
```

This statement would select records where the **No.** is larger than 10000 and not equal to 20000.

```
Customer.SETFILTER("No.", '>%1&<>%2', Value1, Value2);
```

If the variables V1 and V2 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 delimitation currently in effect for a field. `GETRANGEMIN` has this syntax:

```
Record.GETRANGEMIN(Field);
```

GETRANGEMIN will cause a runtime error if the filter currently in effect is not a range. That is, if a filter has been set like this:

```
Customer.SETFILTER( "No.", '10000|20000|30000' );
```

then

```
BottomValue := Customer.GETRANGEMIN( "No." );
```

will fail, since the filter is not a range.

**GETRANGEMAX** GETRANGEMAX works like GETRANGEMIN, except that it retrieves the maximum value of the delimitation currently in effect.

### Inserting, Modifying and Deleting Records

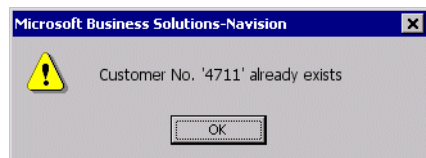
These function are used to *maintain* the database by adding, modifying and removing records.

Generally, these functions return a boolean value that indicates whether they succeed. If you do not handle the return value in your code, a runtime error will occur 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, as in this example:

```
Customer.INIT;
Customer."No." := '4711';
Customer.Name := 'John Doe';
Customer.INSERT;
```

These statement will insert a new record, with No. and Name having the assigned values, while other fields will have their default values. Supposing that **No.** is the primary key of the **Customer** table, the record will be inserted in the **Customer** table unless there already is a record in the table with the same primary key. In that case, as the return value is not tested, this error message would be displayed:



**MODIFY** This function is used to modify an already-existing record. Like INSERT, it returns a boolean – TRUE, if the record to be modified exists, FALSE otherwise.

MODIFY is used like this:

```
Customer.GET( '4711' );
Customer.Name := 'Richard Roe';
Customer.MODIFY;
```

The statements above would change the name of customer 4711 to Richard Roe.

#### MODIFYALL

This function is used to do a bulk update of records. `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.

You could use `MODIFYALL` like this:

```
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 these records to have **Salesperson Code** set to JR.

#### DELETE

This function is used to delete a record from the database. The record to delete must be specified (using the value(s) in the primary key fields) before calling the function. (This means that `DELETE` does take filters into consideration.) Here is an example in which `DELETE` is used to delete the record with customer number 4711:

```
Customer."No." := '4711';  
Customer.DELETE;
```

`DELETE` returns a boolean value: `TRUE` if the record could be found, `FALSE` otherwise. Unless you test this value yourself, a runtime error will occur when `DELETE` fails (returns `FALSE`).

When developing your application, you should consider this scenario:

- 1 You retrieve a record from the database.
- 2 You perform various checks to determine whether the record should be deleted.
- 3 You delete the record, if step 2 indicated that you should.

Now, this can cause problems if, in a multiuser environment, another user modifies or deletes the record between 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 runtime 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 below) – but `LOCKTABLE` should be used as sparingly as possible, since this function effectively short-circuits the concept of optimistic concurrency, thereby degrading performance.

#### DELETEALL

This function is used to delete all records that are selected by the filter settings – if no filters are set, all records in the table will be deleted.

The following statements would delete all records where **Salesperson Code** is PS from the **Customer** table:

```
Customer.SETRANGE("Salesperson Code", 'PS', 'PS');
Customer.DELETEALL;
```

## Transactions

Normally, you do not need to be concerned with transactions and table locking when developing applications in C/SIDE. Chapter 22, C/SIDE in Multiuser Environments, explains the details.

There are, however, some situations where you will have to lock a table explicitly. For example, if you, in the beginning of a function, inspect data in a table, then use this data to perform various checks and calculations and finally want to write back a record, based upon the result of this processing, you will want the values that you retrieved at the beginning to be consistent with the data in the table now. In short, you cannot allow other users to update the table while your function is busy doing its calculations.

**LOCKTABLE** The solution is to lock the table yourself, at the beginning of your function, by using the **LOCKTABLE** function.

## Working with Fields

These functions perform various actions on fields.

**CALCFIELDS** The **CALCFIELDS** function is used to update FlowFields. As described in Form and Control Properties on page 116, FlowFields are automatically updated when they are direct source expressions of controls, but they must be explicitly calculated when they are not (that is, are part of a more complicated expression).

When you use FlowFields in C/AL functions, you have to update them yourself, and this is what you use the **CALCFIELDS** function for. In the statements below, the **SETRANGE** function sets a filter, and then **CALCFIELDS** is called. **CALCFIELDS** will calculate the Balance and Balance Due fields by taking account of the filter setting and performing the calculations that are defined as the CalcFormula properties of the FlowFields.

```
SETRANGE("Date Filter",0D,TODAY);
CALCFIELDS(Balance,"Balance Due");
```

**CALCSUMS** The **CALCSUMS** function is used to calculate the sum of one or more fields that are SumIndexFields in the record. 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.

In the statements below, an appropriate key is selected. Then filters are set, and finally the summation is performed.

```
SETCURRENTKEY("Customer No.")
SETRANGE("Customer No.", '10000', '50000');
SETRANGE(Date, 0D, TODAY);
CALCSUMS(Amount);
```

#### FIELDERROR

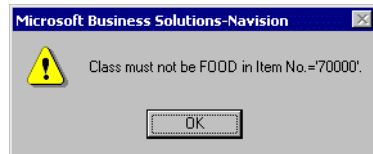
The `FIELDERROR` function triggers a runtime error after having displayed a field-related error message. The function is very similar to the `ERROR` function, described on page 276, but it has some benefits. For one thing, it is easier to use. The more important reason, however, is that if the name of 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 as:

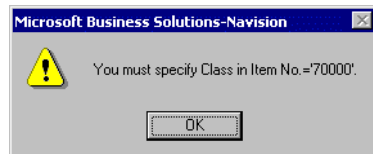
```
Item.GET('70000');
IF Class <> 'HARDWARE' THEN
    FIELDERROR(Class);
```

This will cause an appropriate message to be displayed, depending on whether **Class** currently is empty or has a value.

A message like this will appear when a field has a "wrong" value:



You will see a message like this when a text or code field contains the empty string:

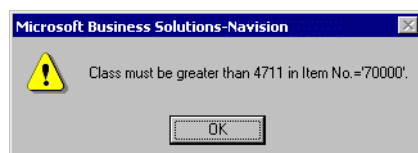


(When a numeric field is empty, it is considered as having the value 0 (zero) – and will produce a message like the first one shown, with "0" instead of "FOOD".)

Finally, you can add your own text if the default texts don't suit your application. Then, you call `FIELDERROR` like this:

```
IF Class < '4711' THEN
    FIELDERROR(Class, 'must be greater than 4711');
```

and the message will look like this:



**FIELDNAME** The **FIELDNAME** function returns the name of a field. Again, you could simply use the name, as you probably know it when you are writing the code, but by using **FIELDNAME**, you can create messages that will still be meaningful if the field name is later 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. If a default value for a field has been defined (by the **InitValue** property), this value will be used for the initialization – otherwise, there is a default value for each data type (see the online C/SIDE Reference Guide entry for **INIT**).

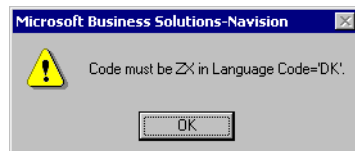
Note that **INIT** *does not* initialize the fields of the primary key.

**TESTFIELD** This function is used to test a field against a value. If the test fails, that is if the values are not the same, an error message is displayed, and a runtime error is triggered, meaning that any changes made to the record will be discarded. If the value to test against is the empty string, the field has to have a value other than blank or 0 (zero).

The following statements:

```
Code := 'DK'
TESTFIELD(Code, 'ZX');
```

would give this error message:



**VALIDATE** The **VALIDATE** function is used to call the **OnValidate** trigger of a field, as in this example, where it will call the **OnValidate** trigger of the **Total Amount** field:

```
VALIDATE("Total Amount");
```

The function is useful for centralizing processing – thus making your application easier to maintain. Suppose that the **OnValidate** trigger of the **Total Amount** field performs a calculation with values from three other fields as operands.

If the contents of any of these fields changes, the calculation must be performed. You should avoid entering the calculation formula in the **OnValidate** triggers of each field – there will be all sorts of possibilities for errors if the calculation formula later has to be changed.

Instead, you should perform the calculation in the **OnValidate** trigger in 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 available for displaying messages and gathering input – but generally, you should use forms whenever it is possible. When you use forms, your application will have a much more consistent user interface.

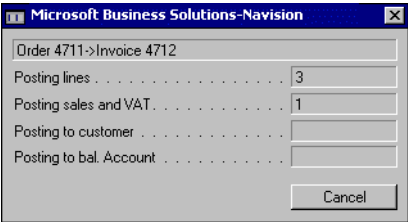
There are, however, situations where it is reasonable to use the dialog functions. The two most important uses are to display a window that indicates the progress of some processing that may take a long time and to halt execution (in order to display an error message or get the user to confirm a choice before the program continues execution). You will also find the `STRMENU` function useful for creating forms to 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

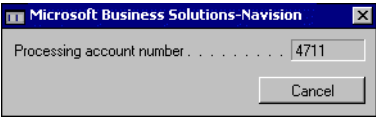
When you have written an application where some processing may – for perfectly good reasons – take a long time, you should consider displaying a window that informs the user of the progress that is being made. The information itself may be superfluous, but it is a good idea to indicate to the user that something is actually going on and the program is still running.

Using a dialog window will also give the user an opportunity to stop the processing – a Cancel button is automatically part of a dialog window.

In some applications, you can create an indicator control to do this. How to do it is described in the section Using an Indicator to Display Values on page 137. In other applications, you can create a window like this instead:



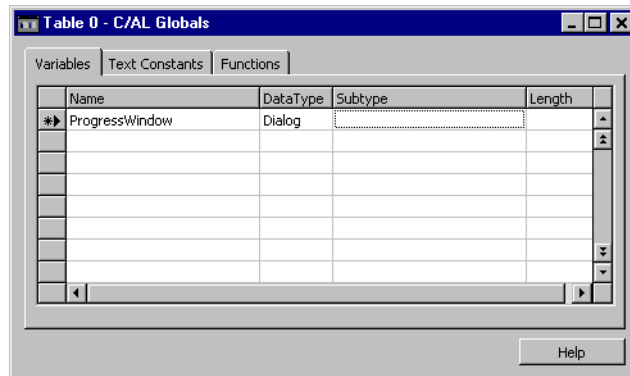
The idea is that each field is updated while the program is running. In the example here, the fields are used to count the number of postings being made. In another situation you could display, for example, the number of the account that is currently being processed, like this:





To create a window like this:

- 1 Declare a variable of type Dialog:



- 2 Open the dialog window, and define the string that will be displayed:

```
ProgressWindow.OPEN('Processing account number #1#####');
```

The part of the string that contains pound signs (#) and a number defines a field that will be displayed in the window, and the number ("1" in this example) can be used to refer to the field.

- 3 You can display the value of any variable in the field. In the example below, the number of each account will be displayed as it is processed:

```
REPEAT
ProgressWindow.UPDATE(1,ChartOfAcc."No.");
// process the account...
UNTIL ChartOfAcc.NEXT = 0;
```

- 4 Finally, close the window when you are finished using it:

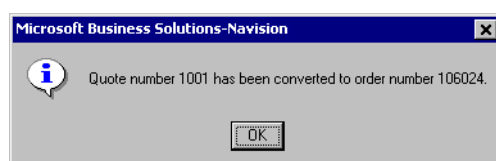
```
ProgressWindow.CLOSE;
```

## Other User Messages

There are a number of other dialog functions available for displaying short user messages. A common trait of these dialogs – except `MESSAGE` – is that execution stops 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. Note that `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:



The window was created by this statement:

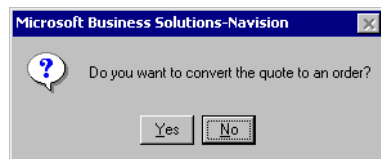
```
MESSAGE(
    'Quote %1 has been changed to order %2.',
    "No.",SalesOrderHeader."No.");
```

#### Note

.....  
Unlike in the example of the progress indication windows, the MESSAGE function was used without first declaring a variable of type Dialog, since there will be no need to refer to this window again.  
.....

**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 272.

**CONFIRM** The CONFIRM function is used to display a message, just like MESSAGE: but unlike MESSAGE, this function returns a value that can (and must) be used, depending on whether the user chooses Yes or No. Its obvious use is for asking a question like this:



The window was created by this statement:

```
IF CONFIRM('Do you want to convert the quote to an order?',FALSE)
THEN
    .... // do the conversion
ELSE
    EXIT;
```

The FALSE parameter means that the negative answer (No) will be 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 user selection to the program.

STRMENU has this syntax:

```
OptionNumber := STRMENU(OptionString [, DefaultNumber]);
```

where OptionNumber is the number of the option 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.)

The statement

```
Selection := STRMENU('Save,Close,Cancel',3);
```

will create this:



Notice that the Cancel option is the default – as the DefaultNumber parameter was set to 3. It is a good idea to let the default option be a "harmless" action, like Cancel, as this option can be chosen by pressing ENTER. If the user inadvertently presses ENTER, no catastrophes will happen, which they might, if, for example, one of the options was "Delete all".



## Chapter 15

### Debugging C/AL Code

This chapter describes the nature of program errors, bugs, and how to use the Microsoft Business Solutions–Navision Debugger to track down errors.

- What Are Bugs?
- Syntax Errors
- Runtime Errors
- Program Logic Errors
- The Microsoft Business Solutions–Navision Debugger

## 15.1 WHAT ARE BUGS?

There are three categories of errors you can meet when you develop applications that use C/AL code

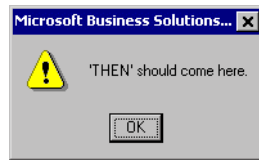
- Syntax errors
- Runtime errors
- Program logic errors

Traditionally, errors in computer programs are called *bugs*, and the process of finding and correcting errors is, correspondingly, called *debugging*.

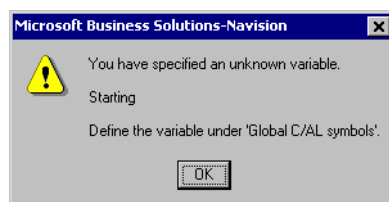
This chapter describes how you can find and eliminate bugs and errors, and it shows how you use the Navision Debugger to find runtime and program logic errors.

## 15.2 SYNTAX ERRORS

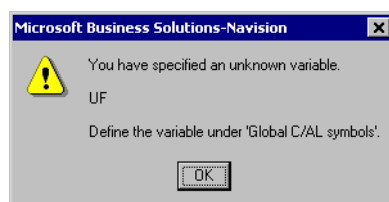
These errors are detected by the C/AL compiler when you try to compile C/AL code, be it in a codeunit or as code in another object (table, form, report, dataport or codeunit). The compiler will notify you of the error with a message like this:



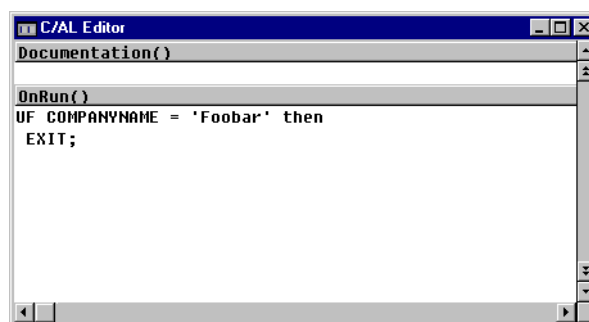
or this:



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.

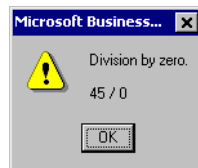
The compiler will not compile code that contains any syntax errors, like a missing `THEN` in an `IF` statement, or code that uses undeclared variables.

## 15.3 RUNTIME ERRORS

Runtime 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:

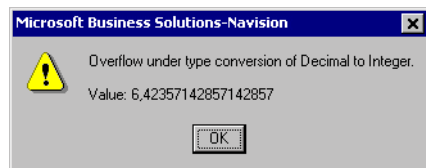
```
Ratio := First_number / Second_number;
```

There is nothing wrong with the syntax, but the statement may cause the following error:



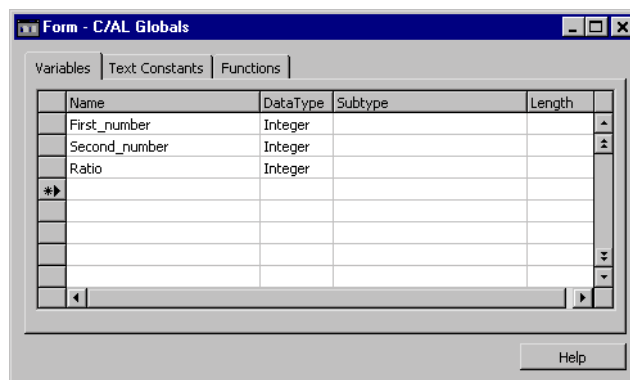
This error occurs because the `Second_number` variable has been assigned a value of 0 (zero), thereby causing a division by zero.

If all three variables are of type integer, the following error could occur:



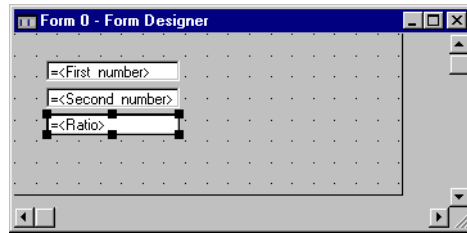
This error occurs because the result of the division cannot be contained in an integer. Therefore, the result is converted to decimal, but then the conversion back to integer (to fit the result into the `Ratio` variable) fails. The common trait of these errors is that the code can work perfectly in many situations, and then fail in some. 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 runtime error in your code, the default messages shown above will appear. If, as in the example, the division by zero was attempted using three variables that were assigned values in a simple form, like this:





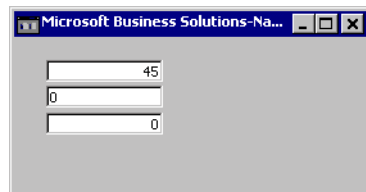
and the form was designed like this:



and, finally, the expression

```
Ratio := First_number / Second_number;
```

was entered in the OnValidate trigger of the Second\_number text box. Then, after the user has acknowledged the runtime error by clicking OK, the form will look like this:



At this point, the user cannot move out of the text box where Second\_number is to be entered, or close the form, without changing the value to something other than 0 (zero).

### About Runtime Errors and Data Consistency

.....

You may now be wondering if runtime errors could compromise the integrity of the database, for example, if some fields are updated in a trigger and a runtime error occurs while some other fields have not been updated. Chapter 22, C/SIDE in Multiuser Environments, explains how data integrity is always maintained, under all circumstances. When a trigger is entered, a write transaction is begun. If a runtime error occurs inside the trigger, the write transaction will be rolled back and the execution of the trigger terminated.

.....

### How to Avoid Runtime Errors

Basically, runtime errors should never occur, and they do not have to, provided that you exercise care when programming. The description below gives some guidelines on how to avoid runtime errors, but they are only guidelines, as the conditions under which runtime errors occur are highly dependent on the context of your application. If, for example, you use the GET function to locate a record, you will have to handle the possibility that a runtime error will 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 runtime 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 runtime errors: 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 most important thing to do to avoid this category of runtime errors is to use correct data types. Errors like the type conversion error on page 282, and overflow errors, can be avoided by using the correct data types. In the context of the example, integer was obviously not a good choice for the Ratio variable. See *Introducing the C/AL Data Types* on page 236. for a description of the data types, and Chapter 19 for a description of how type conversion takes place in C/SIDE.

The division by zero error on page 282 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 never should 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, a value of 0 (zero) could never be entered in the field, no matter how many different text boxes are used to enter data in the field.

### Other Runtime Errors

Any function that can fail to accomplish what it is intended to do can cause a runtime error. A good example is the GET function, used to locate a record in a table according to specified criteria. Consult the online *C/SIDE Reference Guide* for the GET function, and observe the syntax of the command

```
[Ok :=] Record.GET([Value1], [Value2 ],...)
```

The return value of the function is Ok, a boolean. If a record is found, Ok will be TRUE, otherwise it will be 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 runtime error will occur and a system-generated error message will be displayed. If,

on the other hand, you test the return value, a runtime error will not occur, as it is then assumed that you handle the condition yourself.

The online *C/SIDE Reference Guide* always describes whether a function handles errors in a way similar to `GET`. 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 online *C/SIDE Reference Guide* 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 sorting order of a table, and in this case it will return `TRUE` if the sorting order is ascending, and `FALSE` if it is descending.

#### EXAMPLE

By using 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 runtime 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 a record cannot be found and the trigger from where the `GET` function was called is exited.

You should only take the examples above as general guidelines. You will have to consider how to handle situations like these in the context of your own application.

## Finding and Correcting Runtime Errors

As you can see from the runtime error messages reproduced on page 282, 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 runtime error, you will 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 297.

## 15.4 PROGRAM LOGIC ERRORS

The third major category of errors is the 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 perfectly well be compiled, that can be run without causing runtime errors, but that fails to function as was intended.

It can be argued that many, if not most, runtime errors are also program logic errors. However, the "true" program logic error will not make itself noticed in a similarly spectacular way but will quietly generate erroneous data that may not always be detected straight away. The following example illustrates what a program logic error is.

### EXAMPLE

In an application, sales orders are entered on a main form/subform: the general information is entered in the sales header table from the main form, and specific information about items that are ordered is entered in the sales line table from the subform. During data entry, a form that shows statistics about the current order can be displayed. When the form is called, a series of calculations take place and the resulting information is shown like this:

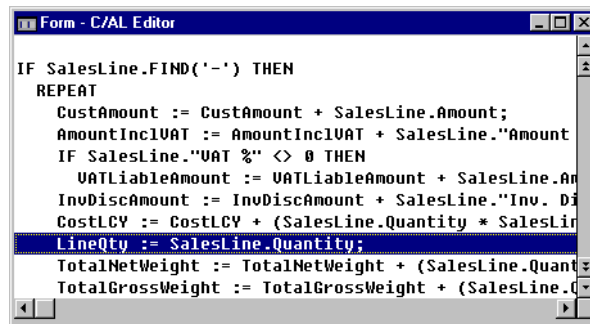
Amount . . . . .	8,145.45	Quantity . . . . .	2.00
Inv. Discount Amount . .	0.00	Net Weight . . . . .	271.20
Total Excl. VAT . . . . .	8,145.45	Gross Weight . . . . .	312.10
VAT Amount . . . . .	0.00	Parcels . . . . .	18.00
Total Incl. VAT . . . . .	8,145.45	Unit Volume . . . . .	13.34
VAT Base . . . . .	0.00	Credit Limit . . . . .	0.00
Total Cost (LCY) . . . . .	21,334.00	Balance . . . . .	0.00
Profit (LCY) . . . . .	7,305.47		
Profit % . . . . .	25.5		

When the form is run with sample data, something appears to be wrong. When compared with the sales order entry forms, the Quantity on the **Sales Statistics** form seems to be incorrectly calculated.

	Typ	No.	Description	Quantity	Unit of Measure	Unit Price	Amount	Line Disc
▶	Iter	1952-W	OSLO Storage Unit/Shelf	2	pcs	385.94928	656.12	
▶	Iter	1928-W	ST.MORITZ Storage Unit/Drawer	2	pcs	833.0475	1,416.19	

The sum of the quantities for those two sales lines that are visible in the subform alone is 4 (and there are several lines below those two lines).

The numbers that are shown in the **Sales Statistics** form are calculated in the OnAfterGetRecord trigger of the form, like this:

The image shows a screenshot of the 'Form - C/AL Editor' window. The code is as follows:

```
IF SalesLine.FIND('-') THEN
    REPEAT
        CustAmount := CustAmount + SalesLine.Amount;
        AmountInclVAT := AmountInclVAT + SalesLine."Amount
        IF SalesLine."VAT %" <> 0 THEN
            VATLiableAmount := VATLiableAmount + SalesLine.Am
            InvDiscAmount := InvDiscAmount + SalesLine."Inv. Di
            CostLCY := CostLCY + (SalesLine.Quantity * SalesLin
            LineQty := SalesLine.Quantity;
            TotalNetWeight := TotalNetWeight + (SalesLine.Quant
            TotalGrossWeight := TotalGrossWeight + (SalesLine.Q
```

The line `LineQty := SalesLine.Quantity;` is highlighted in blue. This line is the erroneous statement mentioned in the text.

The erroneous statement is highlighted. Instead of adding the quantity from each sales line in the REPEAT loop to the variable LineQty (the source expression of the Quantity text box), the variable is assigned the quantity on the current sales line in each iteration. The value that is finally displayed is simply the quantity on the last of the sales lines.

In this example, the error was easy enough to find, just by looking at the C/AL code. The Code Coverage Tool on page 297 shows how the debugger can be used to find the error. In a more complex application, this will be, if not the only way, then the fastest.

## 15.5 THE MICROSOFT BUSINESS SOLUTIONS–NAVISION DEBUGGER

### Overall Description

Navision 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. 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 runtime 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 297, 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 Navision and from Navision Application Server:

#### From Navision

To activate the debugger from Navision, click Tools, Debugger, Active (SHIFT+CTRL+F11). You can also start Navision with the debugger active from the command line by using the `debug` parameter:

#### EXAMPLE

```
fin.exe debug
```

### From Navision Application Server

To activate the debugger from Navision Application Server, you include the `debug` parameter at start-up:

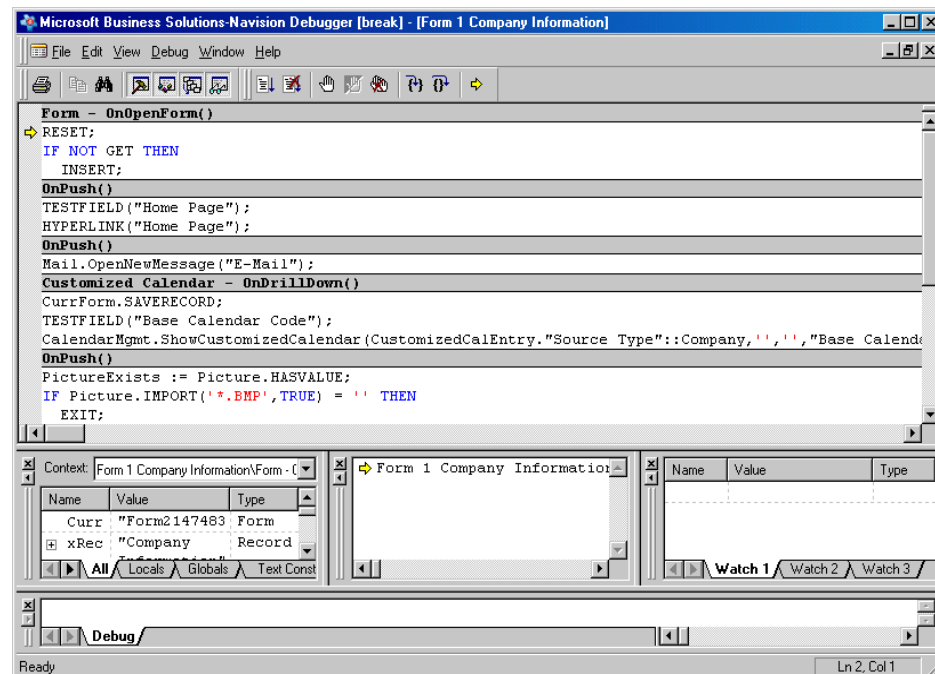
#### EXAMPLE

```
nas debug,startupparameter="test",servername=PC0123
```

If you deactivate the debugger, you cannot activate it again unless you terminate Navision Application Server and then start it up with the `debug` parameter.

### The Debugger Interface

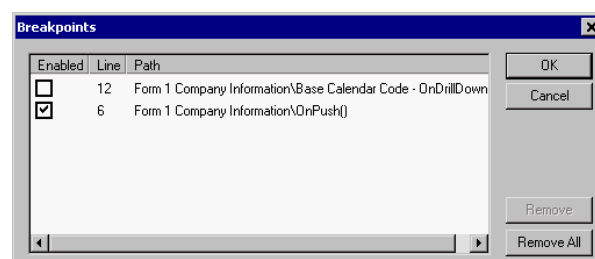
The debugger interface provides special menus, windows and a dialog box. These are described in the following.



### 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.

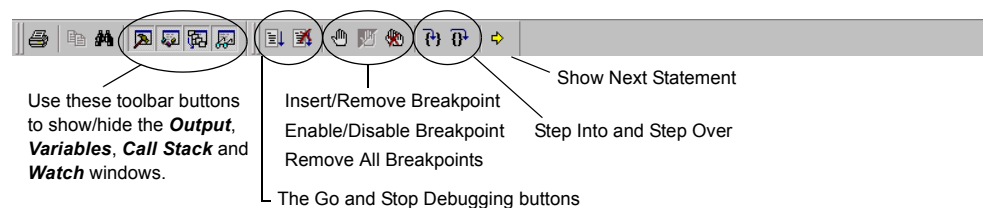
*Step Into* executes statements one at a time, and you can decide how to continue after each statement. The execution will step into any function that is called, which means that the debugger will single-step through the statements in the function.

*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.

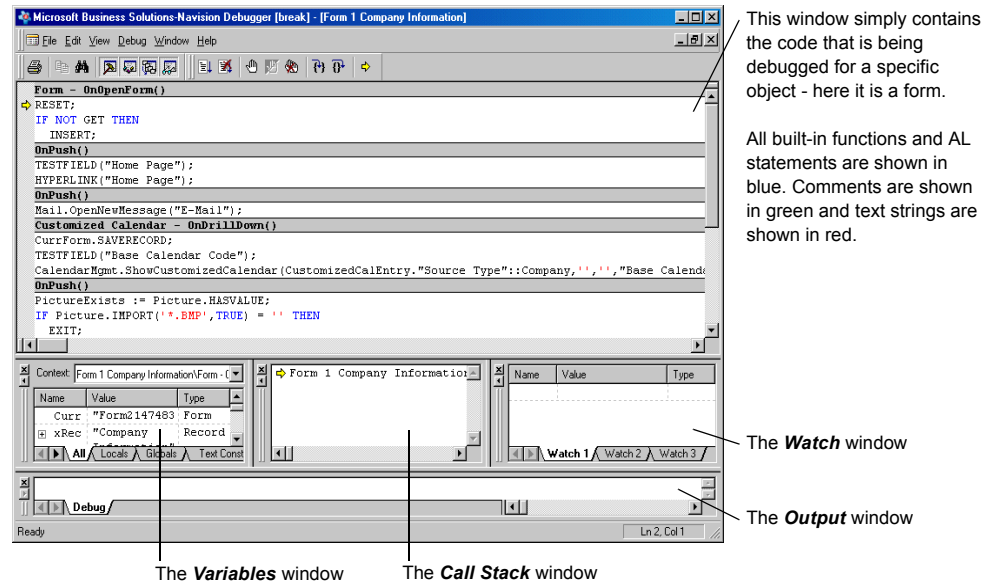
### The Debugger Toolbar



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.



**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 runtime.
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. Here is an example of how your form could look:

[illegible]

Information about breakpoints is saved when you close an object or when you save a new object – compilation is unnecessary. Breakpoints are therefore not stored for objects that you do not save.

## Storage of Breakpoints in an XML File

Breakpoints that are stored in the **Breakpoints** virtual table are automatically stored in a `NaviBP.xml` file. The file is located by default in the same folder as the `fin.zup` file. On a Windows 2000 or Windows XP computer, the path is: `C:\Documents and Settings\user\Application Data`.

Here is an example of an XML file that contains breakpoint information for two objects:

```
<?xml version="1.0" ?>
- <BreakpointList>
+ <Object Type="Codeunit" ID="2" Name="MyCodeunit1">
+ <Object Type="Codeunit" ID="3" Name="MyCodeunit2">
</BreakpointList>
```

This file contains breakpoints for MyCodeunit 1 and MyCodeunit 2. The objects are shown as XML elements called "Object". The object element has three attributes: Type, ID and Name.

If we expand the first object, MyCodeunit 1, we can see one "Breakpoint" element. This shows that the object contains one breakpoint:

```
<?xml version="1.0" ?>
- <BreakpointList>
- <Object Type="Codeunit" ID="2" Name="MyCodeunit1">
- <Breakpoint>
  <TriggerName>OnRun</TriggerName>
  <CodeNo>6</CodeNo>
  <TriggerLine>12</TriggerLine>
  <Enabled>No</Enabled>
</Breakpoint>
</Object>
+ <Object Type="Codeunit" ID="3" Name="MyCodeunit2">
+ <BreakpointList>
```

When a breakpoint element is expanded, we 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 runtime.
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 Navision or Navision Application Server Using Another Breakpoint File

You can start both Navision and Navision 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  
Chapter 25 Performance.  
.....

## 15.6 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 Navision 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.

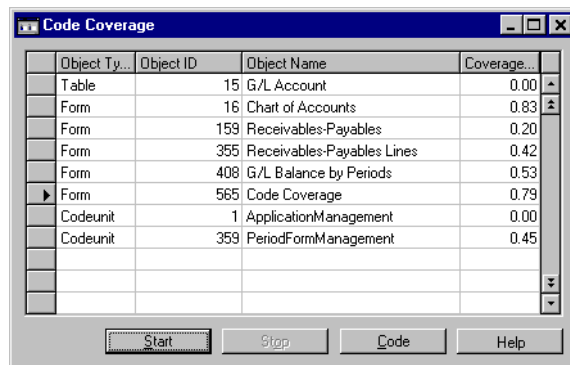
The **Code Coverage** window displays the objects (tables, forms, reports, dataports and/or codeunits) for which code has been executed and logged during one or more transactions. The **Code Overview** window displays the code that has been logged for a selected object. You can read about the **Code Coverage** and **Code Overview** windows in the following section.

### Using the Code Coverage Tool

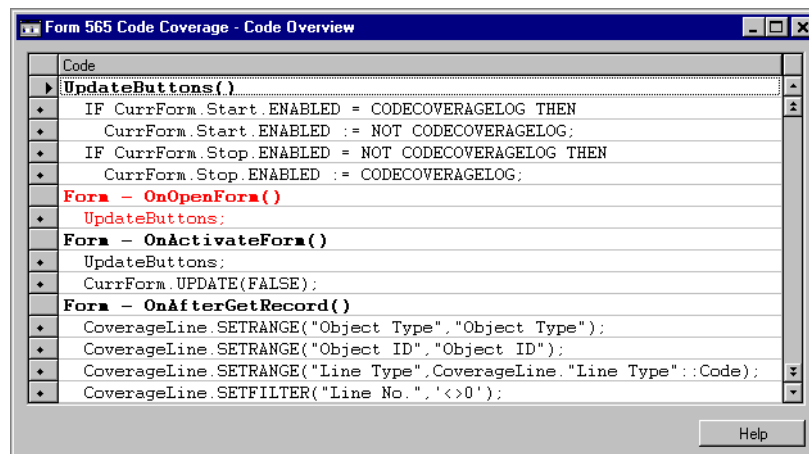
As stated earlier the Code Coverage tool is useful for giving you an overview of the objects that are called when you perform any tasks and the code that is used during these transactions.

To log code:

- 1 Click Tools, Debugger, Code Coverage. The **Code Coverage** window appears:



- 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 16

### 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

## 16.1 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 (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:

This book gives a broad overview of the subject without going into too much detail:

- David Chappell. *Understanding ActiveX and OLE*. Microsoft Press (1996).

This book provides a more technical description:

- Dale Rogerson. *Inside COM*. Microsoft Press (1997).

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):

- Kraig Brockschmidt. *Inside OLE*, 2nd edition. Microsoft Press (1995).

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 table below 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.)
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.

## 16.2 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 332.

### 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)	
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)	
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	
float (VT_R4)	Decimal	
double (VT_R8)	Decimal	

COM Data Type	C/AL Data Type	Comment
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	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 C/AL Date type. S
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 318).

## 16.3 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 type Automation.
- 2 Declare all the other interfaces (classes) as variables of 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 will 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, we will:

*Implement functionality that writes a letter in Microsoft Word by clicking a menu item on the customer card. The letter should 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 we need to transfer to Microsoft Word is in the **Customer** table. Here we find the information about the customer that we will use in the letterhead, such as the name and the address of the customer and the name of the contact to whom we will address the letter.

The **Customer** table also contains a FlowField called **Sales (LCY)**. This field contains the financial information that we need, namely the aggregated sales for the customer. For the sake of this example (where the emphasis is on using automation), we 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.

We will also need to retrieve information from the **Company Information** and the **User** tables to be used in the letterhead and in the greeting of the letter.



We will put all the code in a separate code unit that is called from a menu item on the customer card for the following reasons:

#### Where to Place Automation Code

There are two major concerns when deciding where to place code that uses automation. The first is the fact that an object that uses automation can be compiled only if the automation server is installed on the machine where the compilation takes place. This means that if an object is to be recompiled and modified on a machine where the automation server is not installed, you have to modify the code drastically in order to compile it again. Therefore, it is recommended that you isolate code that uses automation in separate code units.

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 repetitively, it will give better performance if you arrange your code so that the server is created only once (as opposed to 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, we have chosen not to put the automation code on the customer card, but to isolate it in a separate code unit. The performance in a situation where the user wants to create letters to a series of customers in one session could have been improved if we had kept the code on the customer card, thus avoiding having to create and destroy Microsoft Word for each letter.

There is, however, a simple trick that more or less circumvents this problem: if Microsoft Word is already open when it is created from C/AL, the running instance can be reused. This means that the user could either open Microsoft Word "manually" or just not close it after creating the first letter.

#### Background Information about Using Microsoft Word for This Example

What we are aiming for here is a way to transfer data about one customer at a time to Microsoft Word, and the ability to initiate this transfer and the subsequent processing in Microsoft Word from the customer card.

This approach to mail merge is different from the mail merge you can obtain by using C/ODBC, which is better suited for bulk processing (creating a large number of letters).

The chosen approach does, however, force us to use Microsoft Word in a slightly unorthodox way. We want to have a template with a form letter and to put in information at predefined places. In short, we need some placeholders. And here is the catch: without using the regular mail merging facilities of Microsoft Word, there is no straightforward way to do this.

Instead, we will "abuse" Microsoft Word a little. In the template (the form letter), we will put in a number of fields (using Insert, Field... in Microsoft Word). Then we edit these


fields to contain some convenient mnemonic names that correspond to the names of the C/SIDE record fields we are going to use.

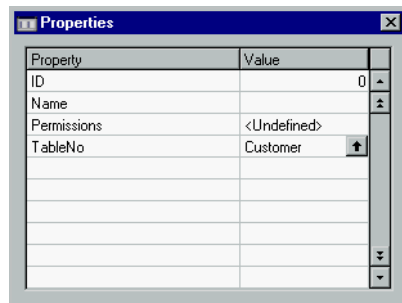
To make this work, we have to use two extra calls to Microsoft Word from our C/AL code. Before starting to use the fields, we will call `ActiveDocument.Fields.Update`. After we have transferred all our information we will call `ActiveDocument.Fields.Unlink`. In this way we can successfully use the Microsoft Word fields for placeholders.

And one more thing, while we can give the fields names like Customer or Address, we will have to reference them by indexing into the Fields collection of the document. This makes the C/AL code somewhat harder to understand.

### Creating the Code Unit and Declaring Variables

The first step is to create the code unit that calls Microsoft Word. Later, we will add the functionality that calls this code unit from the customer card.

- 1 Open the Object Designer, and click Codeunit.
- 2 Click New to create a new code unit.
- 3 On the menu bar, click View, Properties. The Properties form appears.
- 4 In the **TableNo** field, click the AssistButton  to open the **Table List** form. Select the **Customer** table and click OK:

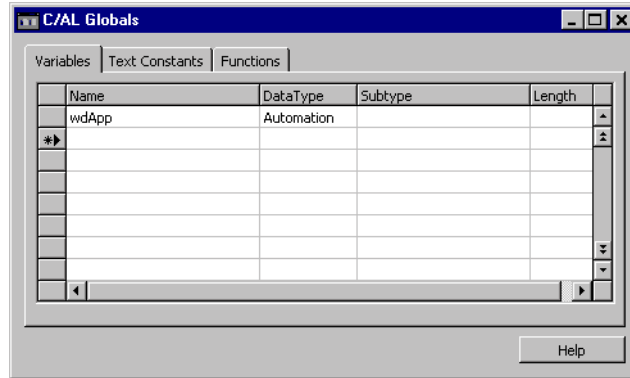


By setting the TableNo property to the **Customer** table, we get a very neat connection between the customer card and this code unit. When we later add the menu item that calls the code unit to the customer card, the code unit will be called with the currently selected record of the customer card as its current record. We do not have to do anything special to coordinate the two.

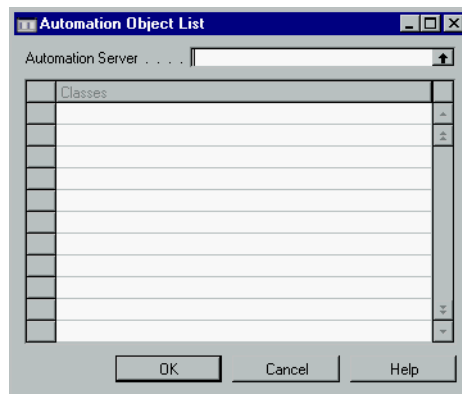
Declaring the variables

- 5 Now we will declare the variables we need. Close the **Properties** form and click View, C/AL Globals on the menu bar.
- 6 First, we will declare the top-level (creatable) class of Microsoft Word. The name of this class is Application. (You can find information about it in the Microsoft Word Objects entry of the online Help for Microsoft Word.) We will name this variable *wdApp*.

Enter *wdApp* as the name of a new variable, and give it the Automation data type. When you move into the **Subtype** field, you will see that there is an AssistButton ... in the field:

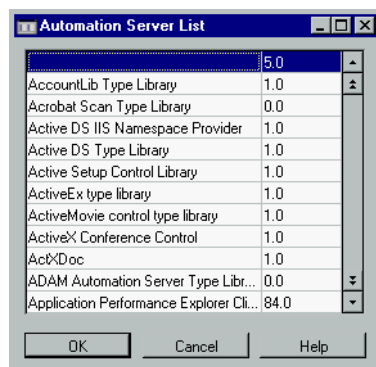


7 In the **Subtype** field, click the AssistButton ... and the **Automation Object List** form appears:

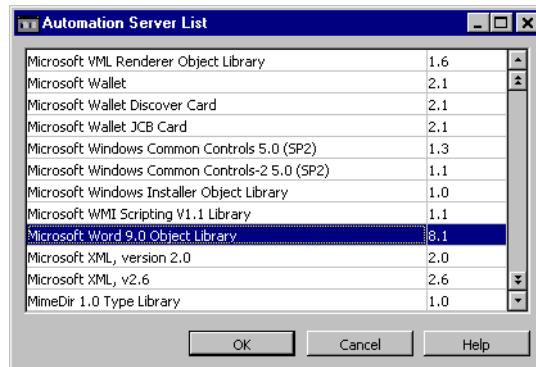


This is the form where you will select the class of the automation server that this variable is referring to, but first, you must select an automation server.

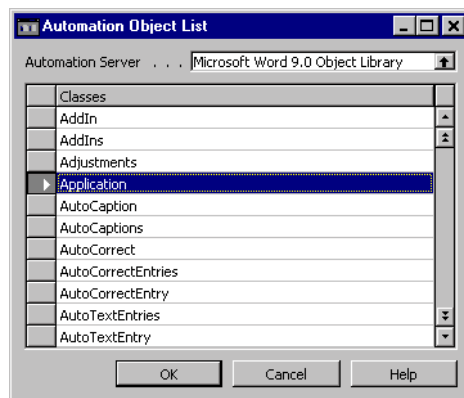
8 In the **Automation Server** field, click the AssistButton ↑. The **Automation Server List** form appears:



This is a list of the automation servers that are installed on the machine. Scroll down to Microsoft Word, and select it:

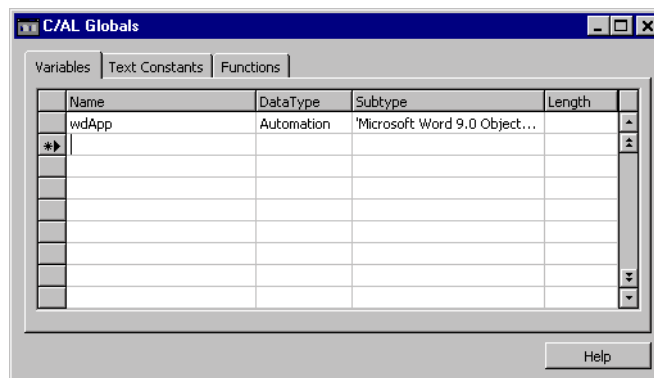


- 9 Click OK. When the **Automation Server List** form has closed, you will see that the **Automation Object List** form has been filled in with a list of all the classes in the Microsoft Word 9.0 Object Library:



- 10 Select Application, and click OK.

Now we have defined the creatable (top-level) class of Microsoft Word as a variable:



11 We will need two more classes from Microsoft Word for this example: Document and Range. Go ahead and declare the variables in the same way as we did for Application. The **C/AL Globals** form should look like this when you are done:

Name	Data Type	Subtype	Length
wdApp	Automation	'Microsoft Word 9.0 Object...	
wdDoc	Automation	'Microsoft Word 9.0 Object...	
wdRange	Automation	'Microsoft Word 9.0 Object...	

12 We also need a few other variables: two records that point to the **Company Information** and **User** tables respectively, and a text variable to hold the name of the template for the Microsoft Word letter we are writing. Setting up these variables is straightforward. The **C/AL Globals** form should look like this when you are done:

Name	Data Type	Subtype	Length
wdApp	Automation	'Microsoft Word 9.0 Object...	
wdDoc	Automation	'Microsoft Word 9.0 Object...	
wdRange	Automation	'Microsoft Word 9.0 Object...	
CompanyInfo	Record	Company Information	
UserInfo	Record	User	
TemplateName	Text		250

Note that the length of the TemplateName text variable has been increased to 250 from the default value of 30.

### Writing the C/AL Code

Before we start writing the part of the C/AL code that uses automation, we have to do some initial processing:

```
Initial processing  CALCFIELDS( "Sales (LCY)" );
                   IF ( "Sales (LCY)" < 2500 ) THEN
                       EXIT;

                   CompanyInfo.FIND;
                   UserInfo.GET( USERID );
```

We start by calculating the **Sales (LCY)** FlowField. Then we check if the customer qualifies for a discount. Finally, we retrieve the information we will need to fill in some fields in the letter from the **Company Info** and **User** tables.

Creating the automation server

Before we can use Microsoft Word, we have to create it, that is, we have to create an instance of Microsoft Word. The C/AL function CREATE does exactly this. We call CREATE like this:

```
CREATE ( wdApp ) ;
```

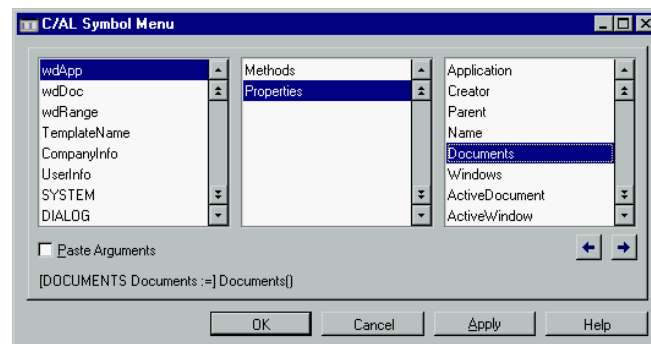
Note that CREATE has an optional argument, NewServer, which by default is FALSE. This means that an already running instance of the automation will be reused. If we had set NewServer to TRUE, as in CREATE(wdApp, TRUE), we would have requested a new instance of Microsoft Word. Note that ultimately the automation server itself can control whether 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 we will add a new document to Microsoft Word using a predesigned template:

```
TemplateName := 'C:\My Documents\Discount.dot';
wdDoc := wdApp.Documents.Add(TemplateName);
wdApp.ActiveDocument.Fields.Update;
```

Because the Add method of the Documents collection requires the path of a template to be passed by reference, we have to set up the TemplateName variable to hold this information. We will get a compile-time error if we 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 press F1 while the Documents property is highlighted, you will see the online Help of Microsoft Word Visual Basic for the property. By browsing through this Help, we learn that the Documents property returns a Documents collection representing all open documents. We also learn that the Documents collection object has an Add method, and that the Add method has this syntax:

```
expression.Add(Template, NewTemplate)
```

where *expression* is a required argument, and it has to be an expression that returns a Documents object. Template and NewTemplate are optional arguments. We will use Template to open a new document based on our form letter 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 USERDEF type. It means that the property returns a Documents class (or IDispatch interface). This information helps the compiler perform a better compile-time type check.

It also means that the statement:

```
wdDoc := wdApp.Documents.Add(TemplateName);
```

succeeds and can pass both compile-time and runtime type checks.

Finally, the Add method returns a Document class. While we did not have to declare a C/AL variable for the "interim" Documents class, we have declared a variable for this return value, wdDoc.

The third line (`wdApp.ActiveDocument.Fields.Update;`) contains a call that is necessary to make the template work as intended (see Background Information about Using Microsoft Word for This Example on page 307 for details.)

Transferring data to  
Microsoft Word

Now we are ready to transfer the actual data from the Customer record to the placeholder fields in the Microsoft Word document.

If we set up the third field in the template for the address of the customer, we can transfer the address like this:

```
wdRange := wdApp.ActiveDocument.Fields.Item(3).Result;
wdRange.Text := Address;
wdRange.Bold := 0;
```

Again, we are really tweaking Microsoft Word here. We cannot use the fields directly as variables (and do an assignment such as `...Fields.Item(3) := Address`). Instead, we use the Result property of the field. This property returns the result of the field as a range. We place this range in the third automation variable declared, wdRange.

Then we can set the Text property of the range to the desired value, in this case, the Address of the customer. Finally, we turn off the bold formatting that the text would otherwise have by default.

### Using Default Members

.....  
You will notice that the documentation for Microsoft Word Visual Basic uses this syntax:

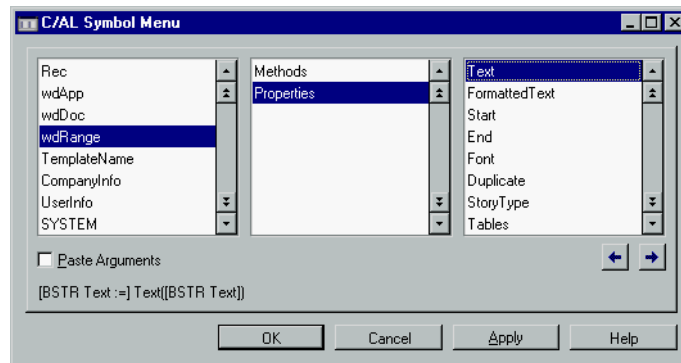
```
wdApp.ActiveDocument.Fields(3).Result
instead of
```

```
wdApp.ActiveDocument.Fields.Item(3).Result
```

in the examples. This is because the Item method is the default member for the Fields collection. Visual Basic will use this method if the programmer does not provide a method to be used. C/SIDE, however, does not have this facility, so you must use the Item method explicitly.

.....

One thing to remember is that whatever data you are transferring, it has to be in text format. If it is not, you will get a compile-time 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.



Therefore, any data that is not `Text` or `Code` must be converted before it is passed on to Microsoft Word. For example, we need to transfer the **Sales (LCY)** field, which is a `Decimal` field. Thus, we have to use `FORMAT` to convert it to `Text`:

```
FORMAT("Sales (LCY)", 0, '<Sign><Integer><Decimals>3>');
```

We can transfer data from tables other than the **Customer** table. These two statements use some of the information we retrieved from the **Company Info** and **User** tables:

```
wdRange := wdApp.ActiveDocument.Fields.Item(11).Result;
wdRange.Text := CompanyInfo.Name;
wdRange := wdApp.ActiveDocument.Fields.Item(12).Result;
wdRange.Text := UserInfo.Name;
```

Finishing the code

After transferring the data we need to Microsoft Word, we need two more statements to finish the processing:

```
wdApp.Visible := TRUE;
wdApp.ActiveDocument.Fields.Unlink;
```

The first statement makes Microsoft Word visible (it was not visible before). The second statement is part of the Microsoft Word tweaking to make fields work as placeholders.

Save and compile

Finally, save and compile the code unit and give it a number and a name. In this example, we have used the name `DiscountLetter`.

To-do list

Although the code described above will work, you will have to add a few things to make it 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 could have different templates for different kinds of letters to the customer.

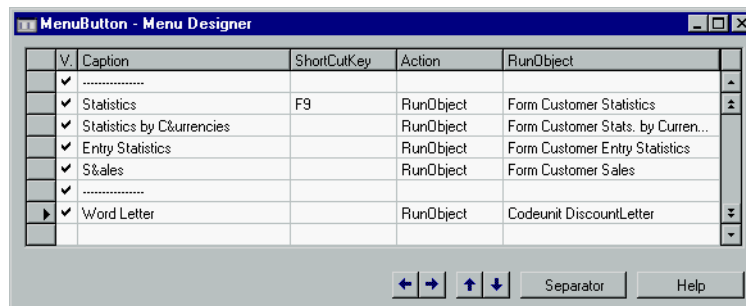


- You should add some error-handling code. For example, the CREATE call fails if the user does not have Microsoft Word installed or if the installation has been corrupted. You should check the return value of CREATE and give an appropriate message if it fails.
- The user should get a message if the customer does not qualify for the discount. In the example, the code unit closes without further ado.

### Calling the Code Unit from the Customer Card

The final task is to make it possible to call the code unit from the customer card. We will add a menu item to the Customer menu button.

- 1 In the Object Designer, click Form.
- 2 Scroll down to the Customer Card form (Form 21), and select it.
- 3 Click Design.
- 4 Right-click the Customer menu button. The context-sensitive menu appears.
- 5 Click Menu Items in the context-sensitive menu. The context-sensitive will close.
- 6 Scroll down to the bottom of the list of menu items.
- 7 Click Separator to insert a separator line.
- 8 Fill in the **Caption** field with the text you want to appear in the menu (here, we have used *Word Letter*).
- 9 In the **Action** field, click the AssistButton ▾, and select RunObject.
- 10 In the **RunObject** field, click the AssistButton ▲ and select the code unit you have created:



- 11 Save and compile the **Customer Card**.

### Graphing With Microsoft Excel

In this example, we will transfer data from the **G/L Entry** table to Microsoft Excel and create a graph. The main point of the example is to show how to handle enumerations.

### Background Information about This Example

We will create a graph in Microsoft Excel that shows the distribution of personnel expenses by departments. In the chart of accounts, we can see that **Total Personnel Expenses** is the total of accounts 8700 to 8790. In the **Departments** table, we can see that there are three departments: ADM, PROD and SALES.

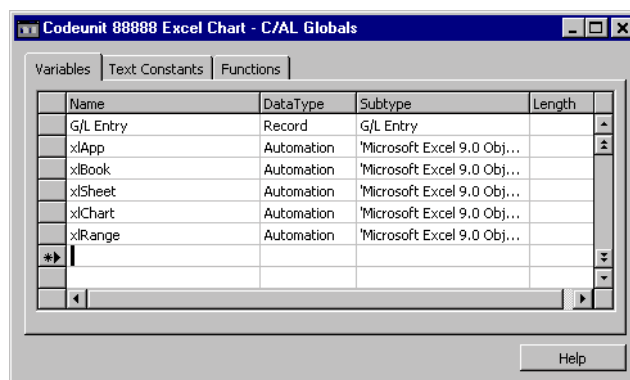
We will create a code unit that retrieves the data from the **G/L Entry** table, transfers it to Microsoft Excel and creates a graph. We will run the code unit directly from the Object Designer, but in a real application you would call it from an appropriate place, for example, from a menu in the **Chart of Accounts** window.

### Creating the Code Unit: Declaring Variables

The first series of steps involve defining the necessary variables:

- 1 Open the Object Designer, and click Codeunit.
- 2 Click New to create a new code unit.
- 3 On the menu bar, click View, C/AL Globals.
- 4 Define a Record variable that has the **G/L Entry** table as Subtype. Here, we have called it G/L Entry.
- 5 Define an Automation variable with Microsoft Excel 9.0 Object Library as Automation Server and Application as Automation Object. Call it xlApp.
- 6 Define an Automation variable with Microsoft Excel 9.0 Object Library as Automation Server and Workbook as Automation Object. Call it xlBook
- 7 Define an Automation variable with Microsoft Excel 9.0 Object Library as Automation Server and Worksheet as Automation Object. Call it xlSheet.
- 8 Define an Automation variable with Microsoft Excel 9.0 Object Library as Automation Server and Chart as Automation Object. Call it xlChart.
- 9 Define an Automation variable with Microsoft Excel 9.0 Object Library as Automation Server and Range as Automation Object. Call it xlRange.

After these steps, the **C/AL Globals** form should look like this:



## Creating the Code Unit: Initial Steps

The code itself is quite simple. First, we will set the key we need for the **G/L Entry** table and then use `SETFILTER` to select the accounts we are interested in:

```
"G/L Entry".SETCURRENTKEY("G/L Account No.", "Business Unit
Code", "Department Code", "Project Code", "Posting Date");

"G/L Entry".SETFILTER("G/L Account No.", '8700..8790');
```

Then, we proceed to create Microsoft Excel:

```
CREATE(xlApp);
```

Next, we add a new workbook to Microsoft Excel:

```
xlBook := xlApp.Workbooks.Add(-4167);
xlSheet:= xlApp.ActiveSheet;
xlSheet.Name := 'Personnel Expenses';
```

In the first line, we use the `Add` method of the `Workbooks` collection to return a new workbook. Then we use the `ActiveSheet` property of the `Application` class to make sure that what we do next will affect the active sheet of the new workbook. In the third line we give the sheet a name.

Now you are probably wondering what the argument, `-4167`, to `Add` is? If we look in the Microsoft Excel Visual Basic online Help, we can see that the `Add` method 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`.

We want to create a workbook with a single sheet. Judging from the description, we should give an `XIWBATemplate` constant with the value `xlWBATWorkSheet` as the `Template` argument.

Nevertheless, we are passing the number `-4167`. The following paragraphs explain why.

### Enumerations

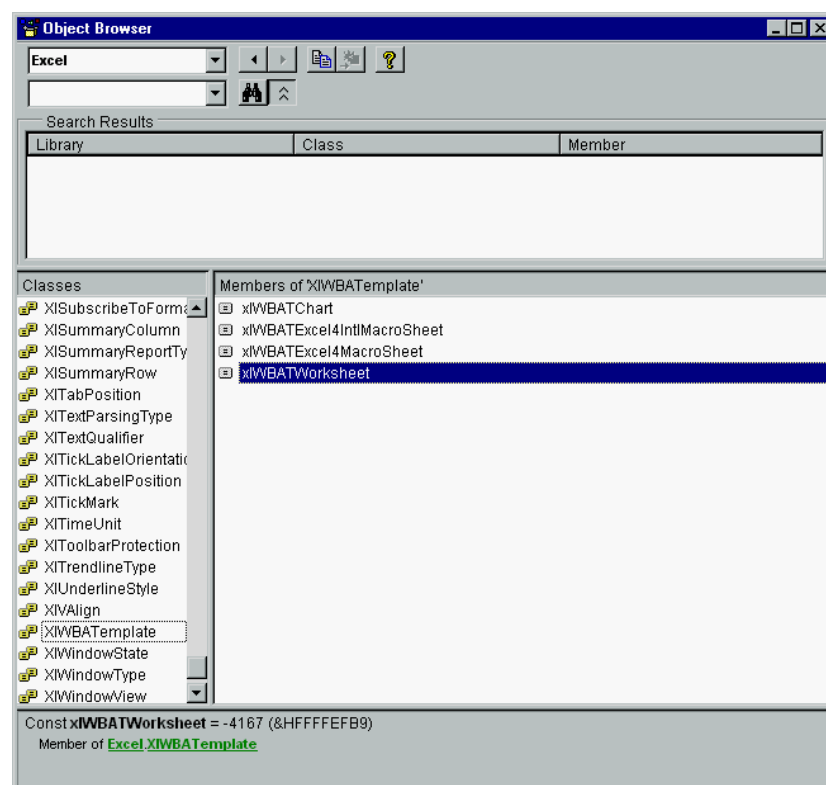
As described on page 304, this particular `VARIANT` is an enumeration.

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 of the form `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`). But how do you find this value?

Finding an  
enumerator value

- 1 Open Microsoft Excel.
- 2 On the menu bar, click Tools, Macro, Visual Basic Editor.
- 3 On the menu bar, 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 shorter method:

A shortcut to the  
enumerator

Create a macro in Microsoft 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 Code Unit: Transferring Data

To transfer the data, we need to do two things: calculate the data and transfer the results of the calculation. To calculate the data, we do the following:

```
"G/L Entry".SETRANGE("Department Code", 'ADM');
"G/L Entry".CALCSUMS(Amount);
```

We use **SETRANGE** to filter the entries in the **G/L Entry** table on the **Department Code** field. The first department is ADM (Administration). Then, we use **CALCSUMS(Amount)** to get the sum for the ADM department.

Now we can transfer the data to Microsoft Excel:

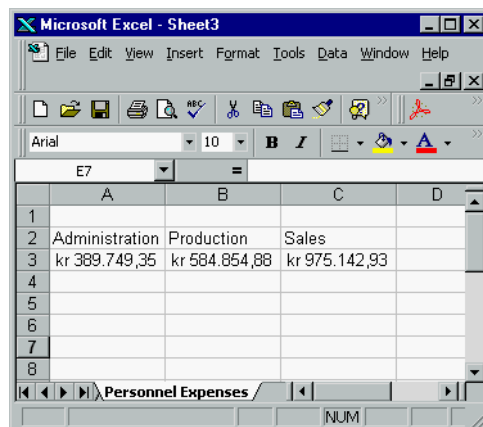
```
xlSheet.Range('A2').Value := 'Administration';
xlSheet.Range('A3').Value := "G/L Entry".Amount;
```

We repeat this for the other two departments, PROD and SALES:

```
"G/L Entry".SETRANGE("Department Code", 'PROD');
"G/L Entry".CALCSUMS(Amount);
xlSheet.Range('B2').Value := 'Production';
xlSheet.Range('B3').Value := "G/L Entry".Amount;

"G/L Entry".SETRANGE("Department Code", 'SALES');
"G/L Entry".CALCSUMS(Amount);
xlSheet.Range('C2').Value := 'Sales';
xlSheet.Range('C3').Value := "G/L Entry".Amount;
```

This is how the data looks once it is transferred to Microsoft Excel:



	A	B	C	D
1				
2	Administration	Production	Sales	
3	kr 389.749,35	kr 584.854,88	kr 975.142,93	
4				
5				
6				
7				
8				

### Creating the Code Unit: Making the Graph

The final step is to create the graph. We 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, we must define a range for the data for the graph:

```
xlRange := xlSheet.Range('A2:C3');
```

Then, we add a new chart sheet and give it a name:

```
xlChart := xlBook.Charts.Add;
xlChart.Name := 'Personnel Expenses - Graph';
```

Finally, this call creates the graph for us:

```
xlChart.ChartWizard(xlRange, -4102, 7, 1, 1, 0, 0, 'Personnel Expenses');
```

We 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').
Gallery	The chart type	-4102 – the enumerator for the xl3DPie XlChartType enumeration. See Finding an enumerator value on page 318.
Format	The option number for the built-in autoformats	We found this one by trial and error, because the Microsoft 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 – we 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 – we do not have series labels in our data.
HasLegend	TRUE to include a legend	2 – for some reason this value works well. There is a possible error in either Microsoft Excel or the documentation here.
Title	VARIANT with the title of the chart	We pass a string, 'Personnel Expenses - Graph.' This works well and the runtime conversion will succeed.

And, finally, we make Microsoft Excel visible:

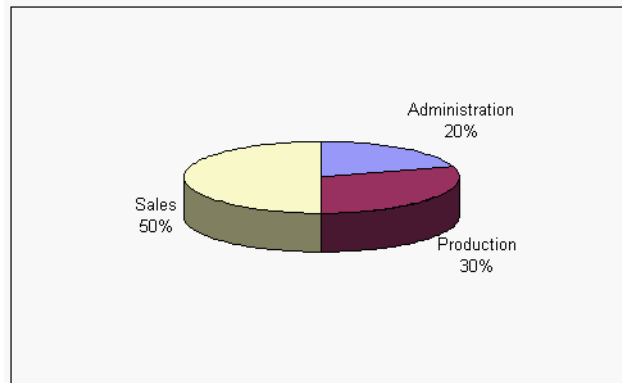
```
xlApp.Visible := TRUE;
```

Microsoft Excel produces a General Protection Fault error when you close a new Excel worksheet created while Microsoft Excel is invisible. To solve the problem, you can make Microsoft Excel visible immediately after you create a new worksheet. Alternatively, you can make Microsoft 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:



## 16.4 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 the 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 "::<Event name>." 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 324.

In the following example, "Receiving Notification of Inbound XML Documents," we enable C/SIDE to receive events from the Navision Communication Component.

For more information about the Navision Communication Component and the way in which it handles the exchange of data streams between Navision Application Server and a bus adapter, see the online Help *Development Guide for Communication Components*.

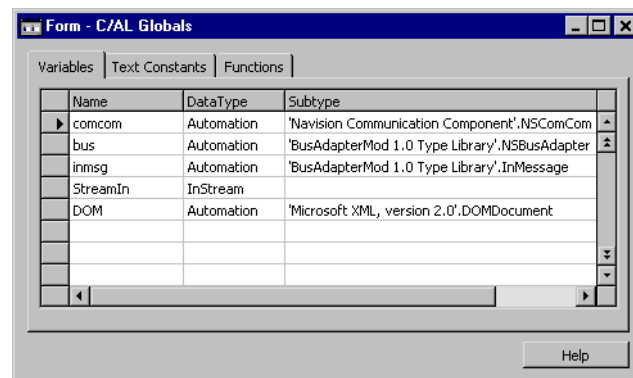
### Receiving Notification of Inbound XML Documents

Before reading on, it will be helpful for you to read the example, Writing a Letter In Microsoft Word on page 306.

In this example, we enable C/SIDE to receive an event – notification of an inbound XML document. There is also an example of the code that can be executed when such an event occurs.

### Declaring Variables for the External Components

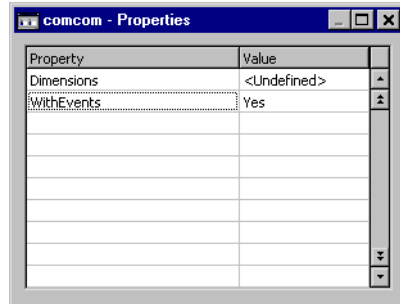
To receive XML documents from a message bus, C/SIDE depends on the existence of two external components, the Navision Communication Component and a bus adapter. The first step is therefore to declare variables of the data type Automation for both components. For each variable, we must also select an automation server and the class of the automation server that the variable refers to.





## Setting the WithEvents Property

Now that we have declared variables for the Navision Communication Component and a bus adapter, we must set the WithEvents property for the Navision Communication Component to Yes. By doing so, we subscribe to events from this component. This means that C/SIDE can receive notification of inbound XML documents.



Setting the WithEvents property to Yes automatically creates a trigger for each event that the selected subtype of the global variable provides. In the case of the Navision Communication Component, it creates the following trigger:

```
comcom::MessageReceived( VAR InMessage : Automation)
```

## Creating the Automation Servers

Before we can use the Navision Communication Component and a bus adapter, we have to create instances of them. To do this, we use the C/AL function CREATE. We call CREATE in the following way:

```
OnInit()
IF ISCLEAR(comcom) THEN
    CREATE(comcom);
IF ISCLEAR(bus) THEN
    CREATE(bus);
comcom.Init(bus);
```

ComCom.Init(Bus) initializes the Navision Communication Component and checks whether the bus adapter has the correct interface.

## Writing Code in the Trigger

By writing code within the MessageReceived trigger, we determine what will happen when the event occurs. In the following example, the code loads the XML document into an XML DOM.

```
comcom::MessageReceived( VAR InMessage : Automation)
inmsg:=InMessage();
inmsg.GetStream(StreamIn);
IF ISCLEAR(DOM) THEN
    CREATE(DOM);
xmldoc.load(StreamIn);
inmsg.commit();
//Data in DOM can now be processed as required.
CLEAR(xmldoc);
```

### Important

.....  
 If you delete a global variable to which event triggers are associated, the event triggers and their code will also be deleted. Furthermore, if you change the DataType or Subtype for a global variable, all the event triggers and their code will be deleted.  
 .....

## Event Triggers

The following information is useful when you have enabled C/SIDE to receive events from a component that it controls, for example, an automation server. 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 subject to certain restrictions.

### Limitations

- C/SIDE only supports the default outgoing, that is, source interfaces, which are defined by the automation variable. If more than one outgoing interface is defined by the automation server for a coclass, only event triggers for the default outgoing interface are generated in the AL code.
- There can be a maximum of 39 parameters in function calls.
- There can be a maximum of 1024 characters in prototype text strings for functions.
- The connectable object strategy in COM is used to connect Navision and the automation server. The Sink object defined in this strategy and implemented in Navision only supports the IDispatch interface (and IUnknown). It is therefore expected that the automation server calls on IDispatch when executing events.
- Parameter names will be 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 will be 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 AL data types.

The data remains in this internal data type until it is used inside the trigger. When data is used, it is converted to the necessary AL data type. Note, however, that 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 is not possible because there is an invalid data type, or because data is outside range, the event trigger causes an error message to pop up and terminates execution. Note that if data is never used in the event trigger, no checks for valid data, data type and data range are performed.

If parameter is a VAR parameter (that is, called ByRef) and data is used inside the event trigger, there will be an implicit conversion just before the event trigger returns. A check is made of whether conversion is possible. If this is not the case, an error message is shown and the event trigger terminates.

## 16.5 USING CUSTOM CONTROLS FROM C/SIDE

As mentioned in Terminology and History on page 301, *Custom Controls* are (or were) also known as OLE Controls and ActiveX Controls. Because they often have the file name extension .ocx, they have also been called OCXs.

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 (an OCX) in C/SIDE, we will have a look at the C/OCX Samples product. If you have access to the product, you can see the code for yourself (both the C++ code for the OCX and the full code for the C/SIDE sample that uses it). The sample comes with a Help file that contains help for both the methods and properties for the control and for the sample application.

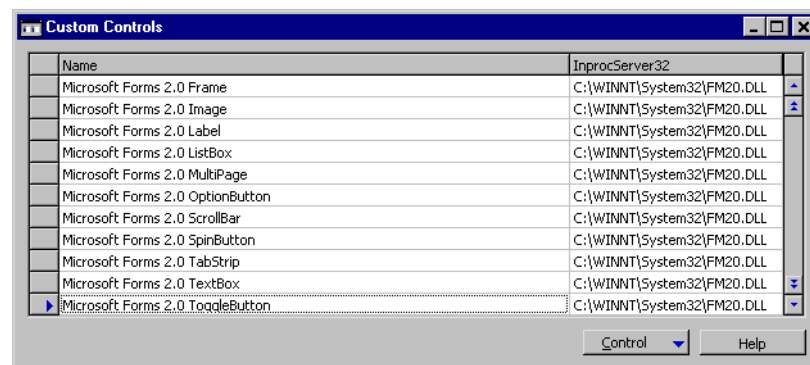
The sample OCX was made in Microsoft Visual C++ 4.2 (but will compile in Microsoft Visual C++ 5.0), using the OLE ControlWizard and the ClassWizard. It has a number of methods for annuity calculations.

### Installing and Registering the Control

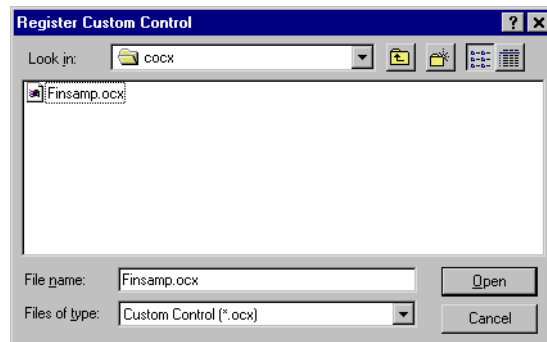
The first requirement for using a control is that it is physically installed on the target machine. But a control also has to be registered with the operating system in order to be used.

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:

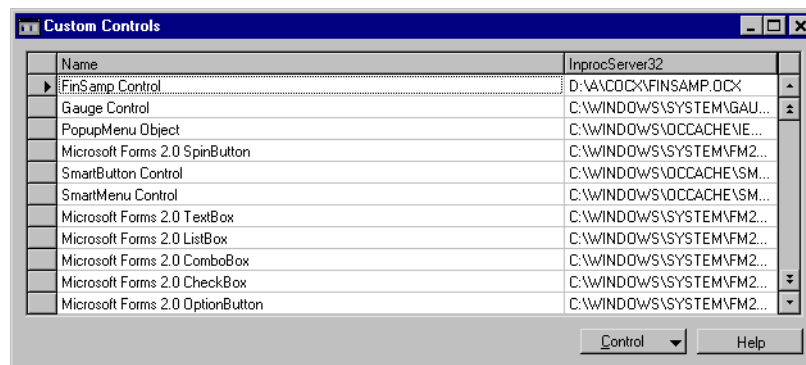
- 1 Copy the control from the distribution media to the hard disk. If you have the C/OCX Samples, there is a readme.txt file with more information.
- 2 On the menu bar, click Tools, Custom Controls... to open this window:



3 Click Control, Browse to open the following window:



4 When you have located the control, select it and click Open. This will register the control with the system. You will receive a confirmation message once the registration is complete. Click OK to return to the **Custom Controls** form. You can verify that the control was added by closing this form and clicking Tools, Custom Controls... again. The new control will appear in the list (see the top line in the window below):



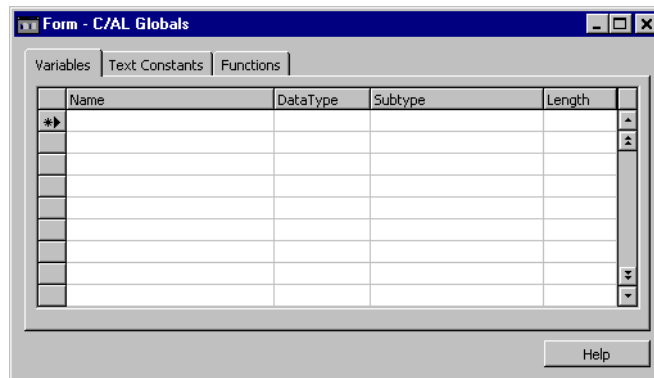
### Using the Control in C/AL


The control must be declared as a (global or 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).

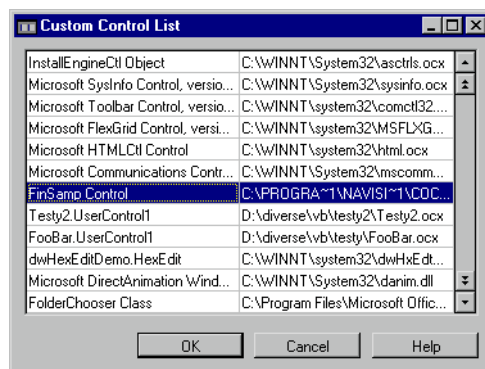
## Declaring the Control as a Variable

Follow these steps to declare the control as a variable:

- 1 On the menu bar, click View, C/AL Globals to open the following window:



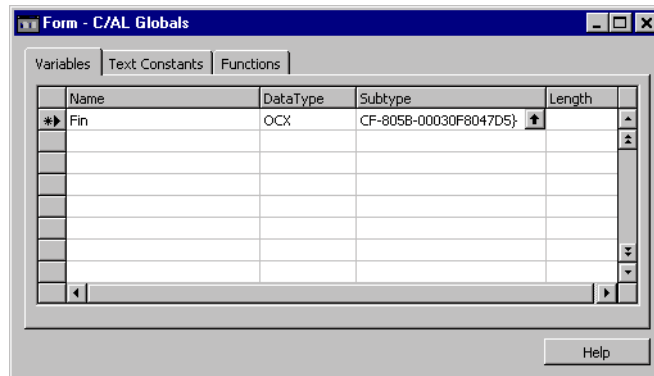
- 2 Give the variable a name (this example uses *Fin*), and select OCX as the type. In the **Subtype** field, click the AssistButton  to open the following window:



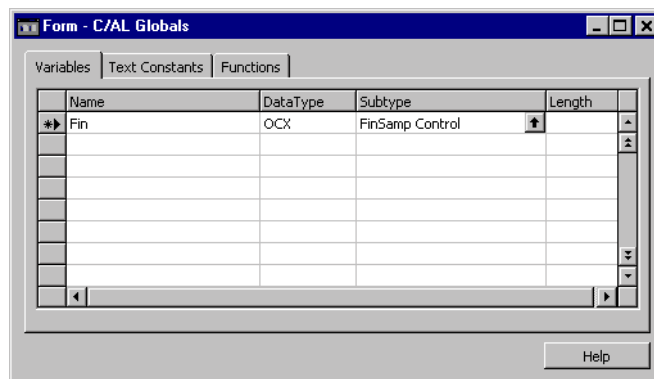
- 3 Find the control on the list and select it. Then click OK.

If the control you want to use is not on the list, read the section Installing and Registering the Control on page 326 for a description of how to install and register a control.

- 4 Now the **C/AL Globals** form should look like the following picture. The CLSID of the FinSamp control has been entered in the **Subtype** field:



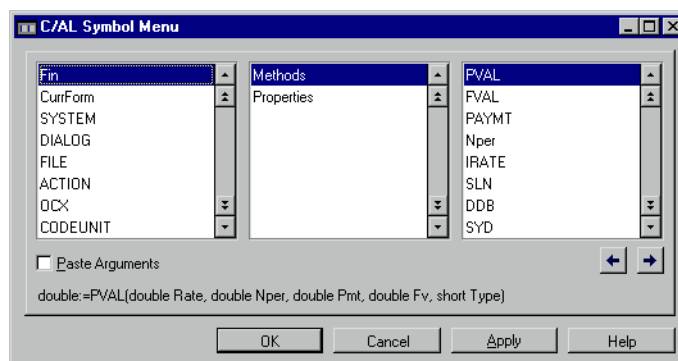
- 5 When you move the cursor out of the **Subtype** field the name of the control replaces the CLSID (which is rather cryptic):



### Accessing the Control

Now you can access the methods and the properties of the control from C/AL. You can explore these methods and properties in the C/AL Symbol Menu:

- 1 Click View, C/AL Symbol Menu to open this window:



- 2 Whenever a method or a property is selected (they are in the column at the right), you can see a short syntax description at the bottom of the window. If there is a

Help file for the control, you can access it by pressing F1 while the method or property you are interested in is selected. The following picture shows the Help for the PVAL method:

**PVAL**  
Method

---

Present Value

**PV(Rate, Nper, Pmt, Fv, Type)**

The present value of an investment. The present value is the total amount that a series of future payments is worth now. For example, if you borrow money, the loan amount is the present value to the lender.

$$PVAL = -\frac{Pmt(1 + Rate \cdot Type)\left(\frac{(1 + Rate)^{Nper} - 1}{Rate}\right) + Fv}{(1 + Rate)^{Nper}}$$

Note: Make sure that Rate and Nper refer to periods of the same length.. If you make monthly payments on a four-year loan at 12% annual interest, use 12%/12 for Rate and 4\*12 for Nper. If you make annual payments on the same loan, use 12% for Rate and 4 For Nper.

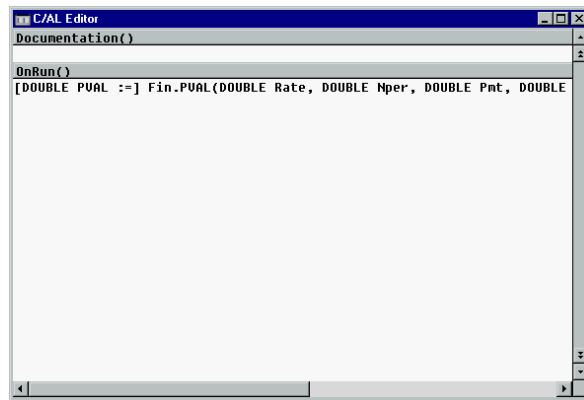
**Example**

Suppose that you are thinking of buying an insurance annuity that pays \$500 at the end of every month for the next 20 years. The cost of the annuity is \$60,000, and the money could otherwise earn 8% p.a.. You want to determine whether this would be a good investment. Using the PVAL function you find that the present value of the annuity is:

`PVAL(0.08/12, 12*20, 500, 0, 0) = -$59,777.15.`

The result is negative because it represents money that you would pay (an outgoing cash flow). The present value of the annuity (\$59,777.15) is less than what you are asked to pay (\$60,000). Therefore you determine that this would not be a good investment.

- When you click OK or Apply in the C/AL Symbol Menu, a "template" statement with the method or property is pasted into the C/AL Editor (provided that the editor is open) at the current insertion point:



### Using a Method

Once the control has been installed and registered and you have declared the variable in the manner described in the preceding sections, using the methods and properties the control exposes is no different from using a normal C/AL function. The statement below uses the PVAL method of the FinSamp control:

```
Amount := Fin.PVAL(Rate,NoOfPeriods,-Payment,0,0);
```



The interesting details are:

*Fin* is the name of the OCX variable and it points to the FinSamp control from the C/OCX Samples.

You can see in the C/AL Symbol Menu that the PVAL method has this syntax:

```
[DOUBLE PVAL]:= PVAL(DOUBLE Rate, DOUBLE Nper, DOUBLE Pmt, DOUBLE Fv,
SHORT Type)
```

The two last arguments (Fv and Type) are both constants with a value of 0 (zero) and will be passed on to the control with the appropriate types (double and short). Rate, Nper and Pmt are *Rate*, *NoOfPeriods* and *Payment*, respectively, in C/AL. They are all of type decimal in C/AL (see Parameters, Return Values and Data Types on page 302). The return value is stored in Amount, a C/AL decimal.

### Using Properties

The properties of a control are read and set just as other properties. For example, the FinSamp control has a boolean property called Error (see the section about exceptions later in this chapter for more information about the purpose of the property). The value can be read like this:

```
Result := Fin.Error
```

where Result is a Boolean and Fin is the FinSamp control.

The Error property is meant to be read from C/AL, and cannot be set. FinSamp has another called property called RateGuess that is used for the initial interest rate when using the iterative IRATE method. RateGuess is set like this:

```
Fin.RateGuess := 0.07
```

or

```
DefaultGuess := 0.07
Fin.RateGuess := DefaultGuess
```

where DefaultGuess has been declared as a Decimal.

#### Parameterized properties

Properties can be *parameterized* (also known as *property arrays*). The term property array describes exactly what this is: a group of related properties can be grouped together as one property. Each value is then accessible as an element of the array through subscripting.

Suppose that we are using a control that has been declared as an OCX variable with the name *MyControl*. This control has a parameterized property called *COLOR* of type short. COLOR has two parameters, both shorts.

If Result has been declared as a C/AL Integer, the value of the property COLOR[2][3] can be retrieved in C/AL by:

```
Result := MyControl.COLOR(2,3);
```

If `NewValue` has been declared as a C/AL Integer, the value of the property `COLOR[2][3]` can be set by:

```
NewValue := 4;  
MyControl.COLOR(2,3,NewValue);
```

### Handling Exceptions

As mentioned earlier (see page 302), you cannot use the exception-handling mechanisms that are described in, for example, *Inside OLE*. The samples in the C/OCX Samples show how you can handle exceptions in another, just as easy, way (but only for controls you create yourself).

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, `FALSE` otherwise. 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`) `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
```

## 16.6 ACQUIRING CONTROLS

**Buy** It is possible to buy a third-party control and use it in C/SIDE, provided that the control fits within the restrictions imposed. These restrictions are described on page 302. In short, they are:

- Only non-visual controls are supported.
- Events are not supported.

**Develop** To develop a control yourself, you will need an appropriate tool for doing so. Currently, the recommended tools are:

- Microsoft Visual C++ 4.0, or later
- Microsoft Visual Basic 5.0

There are other tools on the market capable of creating controls, but the tools mentioned above have been tested with C/SIDE. Furthermore, they both have highly efficient wizards that make the process of 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 thus created are considerably larger than controls created more directly, and they will require additional runtime libraries (for example, controls created with the wizards in Microsoft Visual C++ require the Microsoft Foundation Classes runtime library). For controls that are meant to be loaded over the Internet, this is a consideration of real importance. For controls distributed as extensions to C/SIDE, this consideration is probably less important.

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.







## Chapter 17

### 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

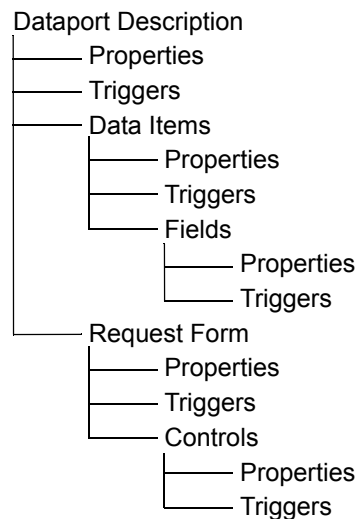
## 17.1 WHAT ARE DATAPORTS?

Dataports are objects that are used for importing data from and exporting data to external text files. During importing and exporting there is a wide range of options for defining the format and layout of the external file.

When importing, you can control what happens if a record in the import file 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 to run the OnValidate trigger for each field.

Dataports can be *dynamic*, that is, on execution the dataport determines whether the process is an import or an export and the name of the file to read from or to be written to. This is achieved either with 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 how a dataport is composed of a number of different components. The following is a short description of each component.

**Dataport Description** This is the complete description of the dataport: 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. To retrieve information from a table, you define a data item and add dataport fields to it.

**Field** A dataport field can be a field in a data item, that is a field in 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 delimited by certain characters that you define.

**Request Form** A request form is a form that is run before the actual dataport begins execution. It is used to gather requests and options from the user of the dataport, for example, the name of the external file. Note that dataports can be run without user



interaction and that you cannot run dataports with request forms on Navision Application Server.

**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 for a field should be executed (when inserting imported data in a table). Properties are set on the Property Sheet of an object.

**Trigger** Certain predefined events that happen to a dataport cause the system to execute a user-definable 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 exporting data, each data item is iterated for all records in the underlying table, and you can set up sorting order, keys and table views to use. For each record you can decide whether or not it should be written to the external file.
Import	When importing data, records read from the external file can be inserted into tables that correspond to data items. You can examine the records before inserting them, and you can specify whether records should be inserted automatically and whether records already in the database should be overwritten or updated when a record with the same primary key is read from the external file (or, of course, whether this record should be inserted at all).

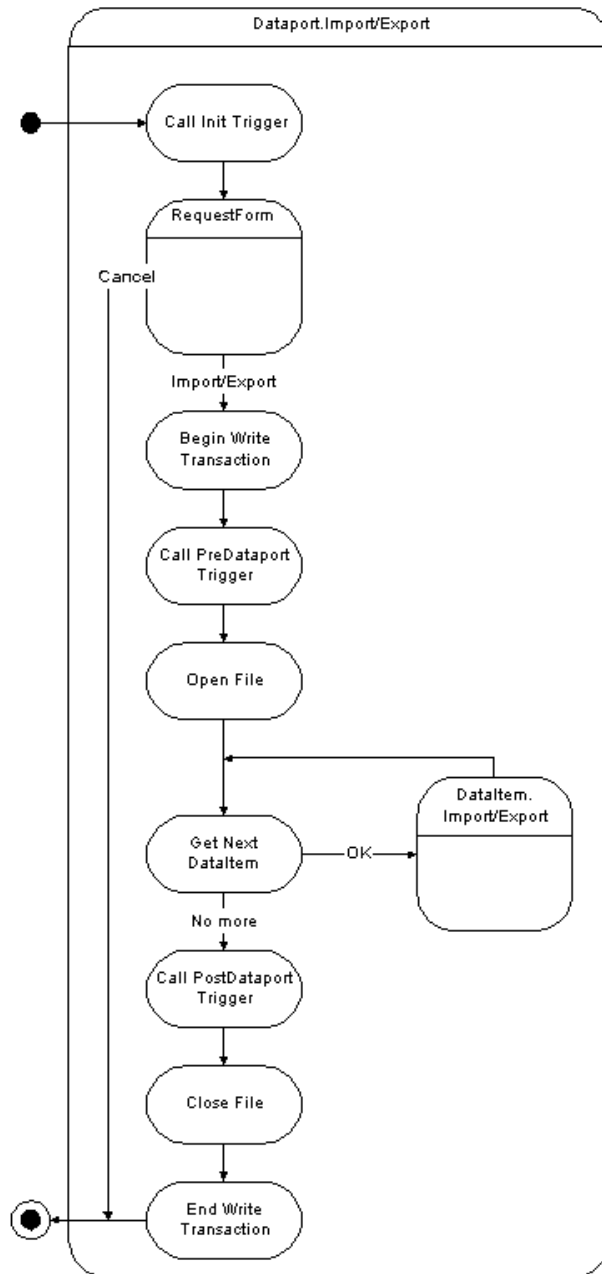
### External file

The layout of the external file is defined by means of a set of dataport properties. During importing, these properties describe how the input stream should be broken up into records and fields. During exporting, these properties describe how the fields and records should be written to the file.

Depending on the file format of the external field, you can set different properties.

## 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 493.



- 1 When the user initiates 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. Here, the user can choose to cancel the dataport run.

- 3 If the user chooses to continue, the dataport enters a transaction (a Begin Write Transaction (BWT) is issued) and then the OnPreDataport trigger is called. At this point, no data has yet been processed.
- 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 (if any) data item 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 at any time before the final EWT, there will be no trace left of the interrupted run afterwards 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 thus cannot be compiled. You can also test-compile a dataport without closing or saving it.

### Saving and Closing a Dataport

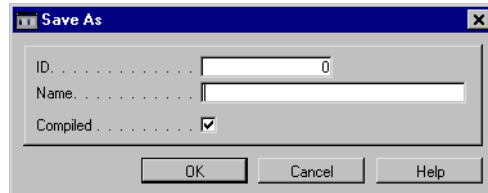
A dataport is closed when the **Dataport Designer** window is closed. You can close this window in the same ways that you can close any other window.

#### 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 without closing it, follow this procedure:

- 1 Click File, Save.
- 2 Assign a name and a unique identifier (ID) to the dataport. The ID must be unique and follow the rules for numbering objects. For more information about object ID numbers, contact your NTR.



- 3 If your dataport is not yet ready to be compiled, click the **Compiled** field to remove the check mark.
- 4 Click OK to save the dataport.

In Step 1, if you click File, Save As, you can rename an existing dataport or you can copy the dataport by assigning it a new ID number.

### Compiling a Dataport

Dataports, like other objects in C/SIDE, must be compiled before they can be run. Dataports can be compiled in two ways:

- When saving the dataport, as described in the section called "Saving and Closing a Dataport".
- Without saving the dataport.

The second option is useful when you are designing a dataport, because you can test-compile the object to find possible errors. To test-compile a dataport during design, click Tools, Compile, or press F11.

### Running a Dataport

In a finished application, your dataports will be incorporated into menus, or they will be called, for example, from a command button on a form. However, while you are designing dataports, you will often want to run them before they have been integrated into an application.

#### Test-running dataports

While designing a dataport, you can test-run it by clicking Tools, Run, or by pressing CTRL+R. The dataport is then compiled and run in its current stage of development. It will not be saved, which means that you can use this functionality to verify that the changes you are making work as intended before you save the object.

**Note**

.....  
 If the dataport is an Import, no records will actually be saved in the database table during a test-run.  
 .....

Running dataports  
 from the Object  
 Designer

You can run a dataport from the list of dataports in the Object Designer by selecting it and clicking Run.

## 17.2 DESIGNING DATAPORTS

This section provides an overview of the elements of designing dataports. The tables of properties and triggers summarizes what each property or trigger is used for. Full (and most up-to-date) explanations are found in the online C/SIDE Reference Guide.

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.

Note that the *Import* and *FileName* properties can be set and reset dynamically. For example, you can create a dataport where the user can select whether to import or export, or 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	ID of the dataport – must be unique among dataports.
Name	Name of the dataport.
Caption	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	List of all translations of the object's caption. For more information, see Chapter 18 "Multilanguage Functionality".
Import	This property determines whether 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	Determines the format of the external file: <i>Variable</i> , <i>Fixed</i> or <i>UPXML</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	This property is used to define the string that separates records on input or output.
DataItemSeparator	This property is used to define the string that separates data items on input or output.
UseReqForm	This property determines whether the request form should be run before the dataport itself is run.

ShowStatus	This property determines whether a status window will be shown while the dataport is running. This window also has a Cancel button that makes it possible to interrupt the dataport run – which is only possible otherwise if you create a dialog yourself.
XMLIncludeTextConst	This property is specific to User Portal and it is only visible when the file format is set to UPXML. You use this property to include the texts constants in the exported XML data. The property accepts the values Yes and No. If you set the XMLIncludeTextConst property to Yes, all text constants will be appended to the XML output when data is exported.
TransactionType	There are four basic transaction type options: Browse, Snapshot, UpdateNoLocks and Update. Each transaction type defines the behavior of a transaction in Navision 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, thereby enabling the user to use dataports that retrieve information from tables that he or she cannot normally access.)

The *FieldStartDelimiter* and the *FieldEndDelimiter* properties are used to place the field contents in quotation marks in situations where the data of a field contains the character that is defined as a separator (*FieldSeparator*, *RecordSeparator* or *DataltemSeparator*). 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 have the delimiters optionally; it will make no difference during importing. During exporting, all fields are written to the external file with all delimiters and separators.

## File Format

The format of the external file is determined by the *FileFormat* property, which defines how a record is read from or written to the file in conjunction with two other properties. 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 *DataltemSeparator* 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, and if you set the file format to UPXML, the *XMLIncludeTextConst* property becomes 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 they are described on page 348.

- 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.
- FileFormat: UPXML** When the file format of the external file is set to UPXML, the dataport can be used by User Portal Application Server to import data from User Portal and export data to User Portal in XML format. When the file format is set to UPXML, a new property is added to the Dataport Designer: *XMLIncludeTextConst*.

### 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 10, page 173). Four properties are special for a dataport: *XMLDataItemName*, *AutoSave*, *AutoUpdate* and *AutoReplace*.

Property	Meaning
DataItemTable	The name of the underlying table. Can be set in the designer when creating data items.
DataItemVarName	The name of the data item as a variable. The default value is the value of DataItemTable.
DataItemTableView	The key, sort order and filters to apply.
ReqFilterHeading	Tab caption for this item on the request form. The default value is the value of DataItemTable.
ReqFilterFields	Names of the fields that initially will be included in the ReqFilter form.
CalcFields	Names of the fields that will be calculated after a record has been retrieved.
AutoSave	This property determines whether records that are imported will be automatically inserted in a C/SIDE table.
AutoUpdate	This property determines whether records that are imported will be initialized with values from an existing record with the same primary key.
AutoReplace	This property determines whether records that are imported will automatically replace existing records with the same primary key.
XMLDataItemName	This property to set the name for a data item that will be used in the XML data generated by a User Portal dataport. An underscore character ( _ ) replaces special characters not supported by XML.

**User Portal Dataports** The XMLDataItemName property is one of four properties that are activated when the FileFormat property for the dataport is set to UPXML. The other three are DataItemIndent, DataItemLinkReference and DataItemLink, which are also used for



reports as described on page 176. These four properties are only used for User Portal dataports.

#### Note

.....  
 You can only indent data items in dataports where the FileFormat property is set to UPXML. All other dataports cannot be compiled with indented data items.  
 .....

### AutoUpdate, AutoReplace, AutoSave

The three *Auto\** properties determine how records that are read from the external file are handled. They are also used to resolve the conflict that arises when a record that has been read from the external file during importing 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:

Auto Save	Auto Update	Auto Replace	Record exists in database and in import file	Record exists only in import file
No*	---	---	The record in the database is not automatically updated or replaced	The import record is not automatically inserted in the database
Yes	No	No	A runtime error will be provoked, and the import is terminated	The import record is automatically inserted in the database
Yes	No	Yes	The import record will replace the existing record	The import record is automatically inserted in the database
Yes	Yes	---	The import record will update the existing record in the database	The import 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 by a brief 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 records that are already 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 information other than the item numbers and the prices, for example names of the items, stock level, and so forth).

AutoUpdate solves this dilemma. When a record is imported, it actually replaces the existing record, but fields that are not present in the imported record are initialized with the data from the already existing record instead of being left empty. 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 to be broken into fields. During exporting, these properties determine how data from the database is written to the external file.

Property	Meaning
Enabled	Determines whether 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.
SourceExpr	The source expression of the field. For an import, this could be the name of the database table field where the value that is read from the external should be stored, but it can be any valid C/AL variable. For an export, 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.
XMLFieldName	Contains the name of a field that will be used in the XML data generated by a User Portal dataport. An underscore character ( _ ) replaces special characters not supported by XML. The default value of this property is the value of the SourceExpr property.
Caption	Caption in the currently selected language. The value is taken from the CaptionML 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	List of all translations of the field's caption. For more information, see Chapter 18 "Multilanguage Functionality".
StartPos	If FileFormat is Fixed, this is the starting position of this field. Positions are numbered from 1 upwards.
Width	If FileFormat is Fixed, this field contains the width of the field.
CallFieldValidate	Determines whether the OnValidate trigger will be executed when the field is imported.
Format	Describes how the field will be formatted during exporting. For example, you can determine the number of decimal places, and so forth.

When fields are exported, the data is converted to text before the export. If the format is Fixed and the Width property is set to a value smaller than the actual width of the data after conversion, the contents will be 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 will 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 will make it impossible to compile (and thus to execute) the dataport.

When you are importing data, a value that is too large for the data type or defined width of the database table field where it is to be inserted will cause an error and execution will stop. As the whole dataport is inside a transaction, no traces will be left of the aborted run in the database.

## Dataport Triggers

The following is a list of all triggers that are executed when a dataport is run. The descriptions provide an overview only. The online C/SIDE Reference Guide contains the full and most recent descriptions.

Dataport Trigger Name	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 data items have been processed.
OnPreDataItem	Before the data item is processed – but after the associated variable has been initialized and 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.

## 17.3 EXPORTING DATA

This section tells you how to create dataports for exporting data using each of the two basic formats for the external file: fixed and variable format. For examples of using the UPXML file format, see the training manual *User Portal Developer's Guide*. In the next section, 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 sample dataport will export records to a file in fixed format. The records in the file will be separated by newlines, and within each record (line, in effect), a field will have the same width in all records, no matter how wide the actual data of the field is.

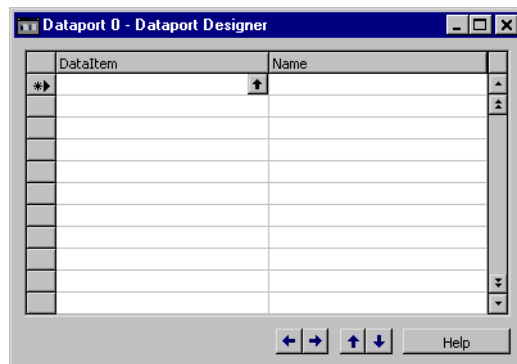
The database table that will be used as an example is the **G/L Account** table (the chart of accounts). This table contains several FlowFields, which have to be calculated when exporting.


### Simple Version

The first version is very basic. It will be extended to a more refined object in the next subsection.

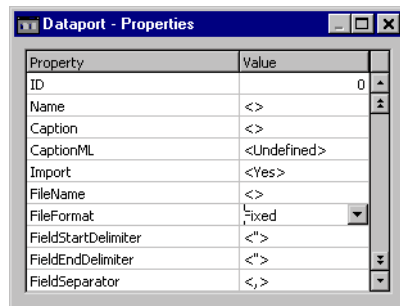
To create a dataport, follow this procedure:

- 1 Click Tools, Object Designer.
- 2 In the Object Designer, click Dataport.
- 3 Click New. The **Dataport Designer** window appears:



- 4 In the **Dataport Designer** window, in the first **DataItem** field, click the AssistButton  and select the **G/L Account** table from the **Table List** form that appears. The program automatically fills in the **Name** field with the name of the table. In this dataport, this default value is acceptable.
- 5 Click an empty line in the Dataport Designer to select the object and press SHIFT+F4 to open the Property Sheet for the dataport (not for the data item). To select the object rather than a data item, click an empty line or click Edit, Select Object.

You can also open the Property Sheet by clicking View, Properties.



6 Leave the default settings except for the following two properties:

**Import** – set it to No to create a dataport that exports data.

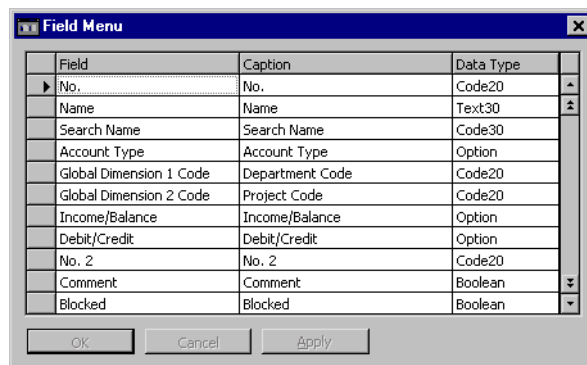
**FileFormat** – set it to Fixed.

We have 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

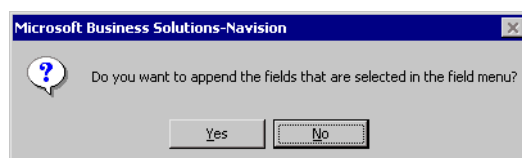
To add dataport fields to a dataport, follow this procedure:

- 1 In the Dataport Designer, click the G/L Account data item to select it.
- 2 Click View, Dataport Fields.
- 3 When the Field Designer is open, click View, Field Menu.

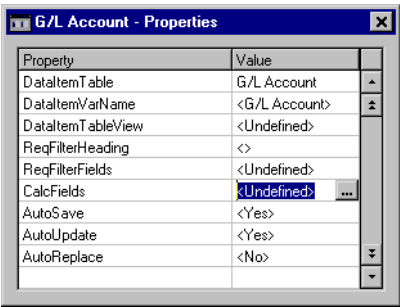


4 Now, select the fields to be exported, for example the **No.**, **Name**, **Balance at Date** and **Net Change** fields. You may have to scroll the Field Menu to select these fields, depending on how the **Field Menu** window is currently sized.

5 Click the Field Designer. You will be asked if you want to append the selected fields. Click Yes.

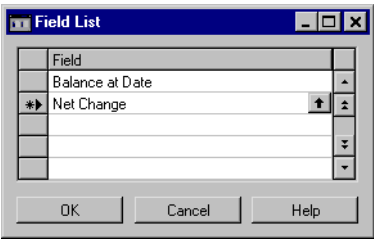


6 Go to the Property Sheet of the G/L Account data item.



7 In the CalcFields property, click the AssistButton to open the Field List.

8 In the **Field List** window, select those fields from the data item that must be calculated when they are exported, that is, the FlowFields. In this case, it is the **Balance at Date** and **Net Change** fields. Click OK to close the **Field List** window.

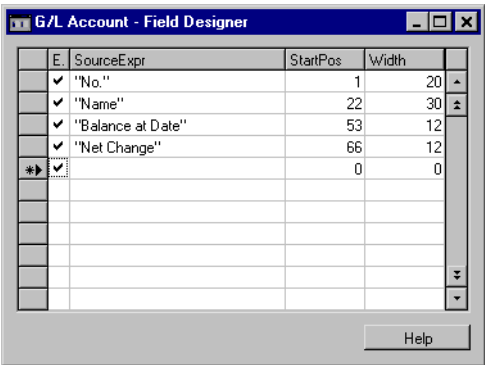


You can now run the dataport. You can run it before saving it by clicking File, Run. If you want to save the dataport first, you will be prompted for a name and a number as with other objects. For more information about saving dataports, see page 341.

StartPos and Width Properties

When the dataport fields have been inserted from the Field Menu, you can check the settings of the StartPos and Width properties of all the fields in the **Field Designer** window.

To open the **Field Designer** window, click View, Dataport Fields. In this example, the following window appears:



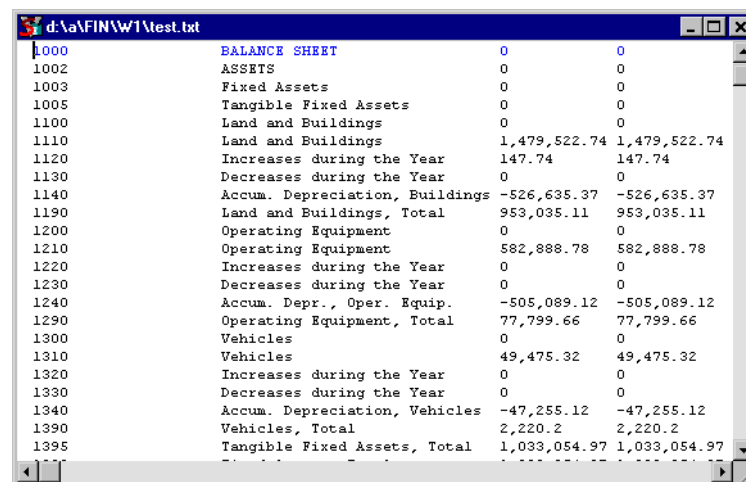
As you can see, both the StartPos and the Width properties have been filled in for you. They are assigned values according to these rules:

Datatype 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 will see the default request form. The first tab, G/L Account, is of little interest, but the second one, Options, is important. Here you can state the name of the external file to write to.

When you have stated the name of the file, click OK to run the dataport. The default status window will show you the progress. When the run is over, you can look at the file in a text editor:



1000	BALANCE SHEET	0	0
1002	ASSETS	0	0
1003	Fixed Assets	0	0
1005	Tangible Fixed Assets	0	0
1100	Land and Buildings	0	0
1110	Land and Buildings	1,479,522.74	1,479,522.74
1120	Increases during the Year	147.74	147.74
1130	Decreases during the Year	0	0
1140	Accum. Depreciation, Buildings	-526,635.37	-526,635.37
1190	Land and Buildings, Total	953,035.11	953,035.11
1200	Operating Equipment	0	0
1210	Operating Equipment	582,888.78	582,888.78
1220	Increases during the Year	0	0
1230	Decreases during the Year	0	0
1240	Accum. Depr., Oper. Equip.	-505,089.12	-505,089.12
1290	Operating Equipment, Total	77,799.66	77,799.66
1300	Vehicles	0	0
1310	Vehicles	49,475.32	49,475.32
1320	Increases during the Year	0	0
1330	Decreases during the Year	0	0
1340	Accum. Depreciation, Vehicles	-47,255.12	-47,255.12
1390	Vehicles, Total	2,220.2	2,220.2
1395	Tangible Fixed Assets, Total	1,033,054.97	1,033,054.97

### Refined Version

The simple dataport is based on default values, but depending on what the dataport is to be used for, several things can be changed to make it easier to use. The following are examples of what you can do:

- The first tab in the request form should not be shown, as we do not want the user to set filters and keys.
- Only accounts where the **Account Type** field is set to Posting or End-Total should be exported.

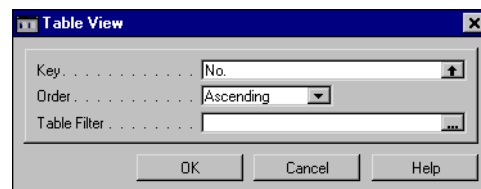
- The numbers must be formatted as thousands. There must be no thousand separators, no decimals, and the sign should be prefixed.

#### Changing the Request Form

You must not set the UseReqForm property to No in order to remove the request form. Doing so will make it impossible for the user to set the name of the file to write the data to. In fact, since no filename has been set, the dataport will provoke a runtime error.

Instead, you can keep the Options tab and remove the data item tab by setting a DataltemTableView for 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.

- 1 In the Dataport Designer, select the data item G/L Account and open the Property Sheet.
- 2 In the DataltemTableView property, click the AssistButton to open the **Table View** window:

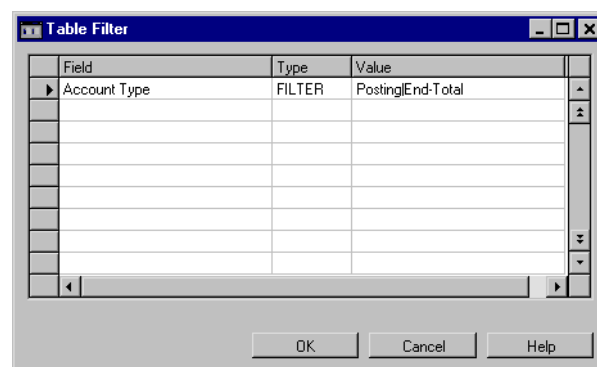


- 3 Fill in at least one of the fields. In the picture above, the **Key** field and the **Order** field have been modified.

#### Selecting Specific Account Types

To select only some account types, you must set a table filter. You can do this in the DataltemTableView property. Continue from step 3 above:

- 4 In the **Table Filter** field, click the AssistButton to open the following window:



- 5 Set the values of Field, Type and Value as shown in the preceding picture. This creates a table filter that will select records where the account type is Posting or End-Total (the character between the two values is a | (pipe) which means OR.)
- 6 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:

**7** Go to the Property Sheet of both the **Balance at Date** and **Net Change** fields.

**8** Enter this expression as the SourceExpr property of the first field:

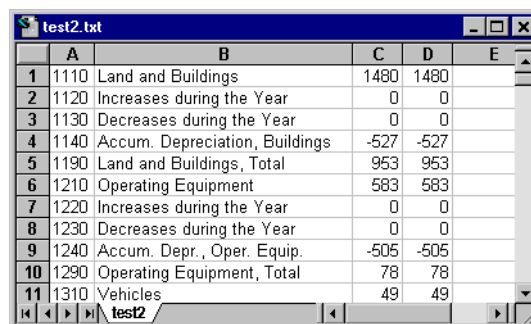
```
FORMAT(ROUND("Balance at Date"/1000,1,'='),0,1)
```

In the C/AL statement, the value of the **Balance at Date** field is first divided by 1000. The result is rounded by the ROUND function, and then FORMAT is used to render the return value from ROUND in format 1 (which for a decimal value means <Sign><Integer><Decimals>).

**9** Enter this expression as the SourceExpr of the second field:

```
FORMAT(ROUND("Net Change"/1000,1,'='),0,1)
```

The file that is created by this dataport could look as follows when it is imported into a spreadsheet:

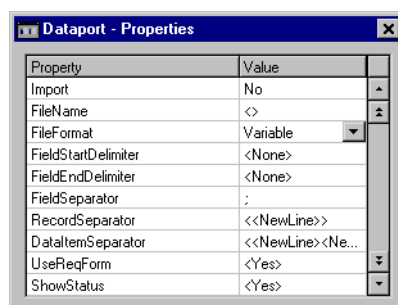


	A	B	C	D	E
1	1110	Land and Buildings	1480	1480	
2	1120	Increases during the Year	0	0	
3	1130	Decreases during the Year	0	0	
4	1140	Accum. Depreciation, Buildings	-527	-527	
5	1190	Land and Buildings, Total	953	953	
6	1210	Operating Equipment	583	583	
7	1220	Increases during the Year	0	0	
8	1230	Decreases during the Year	0	0	
9	1240	Accum. Depr., Oper. Equip.	-505	-505	
10	1290	Operating Equipment, Total	78	78	
11	1310	Vehicles	49	49	

### Exporting - Variable Format

This sample dataport will export the same records as in 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.

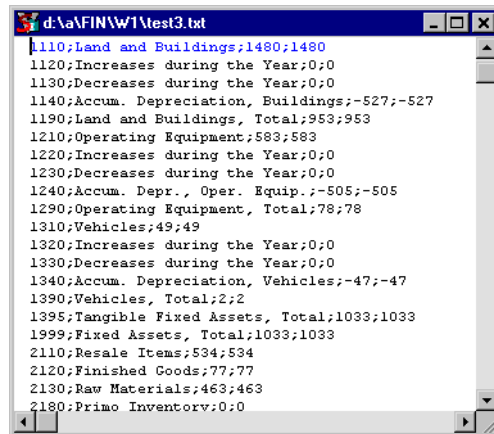
To create the dataport with variable file format, follow the procedure described in the section called "Exporting - Fixed Format" on page 350 with the following exceptions in the dataport properties:



Property	Value
Import	No
FileName	<>
FileFormat	Variable
FieldStartDelimiter	<None>
FieldEndDelimiter	<None>
FieldSeparator	:
RecordSeparator	<<NewLine>>
DataItemSeparator	<<NewLine><Ne...
UseReqForm	<Yes>
ShowStatus	<Yes>

The FileFormat property is set to Variable, and the FieldSeparator property is set to ; (semicolon). The FieldStartDelimiter and FieldEndDelimiter properties are both set to <None>. The RecordSeparator and the DataItemSeparator properties have been left at their default settings, which means that records will be separated by newlines, and data items by two newlines (which is not of real interest here, because we have only one data item in this dataport.)

The exported file could look as follows in a text editor:

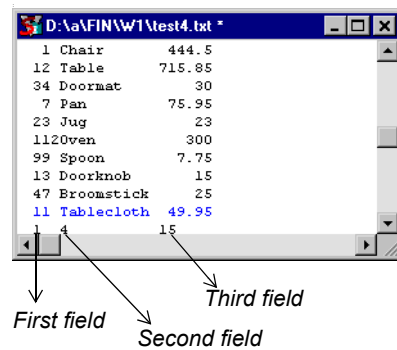


```
d:\a\FIN\W1\test3.txt
1110;Land and Buildings;1480;1480
1120;Increases during the Year;0;0
1130;Decreases during the Year;0;0
1140;Accum. Depreciation, Buildings;-527;-527
1190;Land and Buildings, Total;953;953
1210;Operating Equipment;583;583
1220;Increases during the Year;0;0
1230;Decreases during the Year;0;0
1240;Accum. Depr., Oper. Equip.;-505;-505
1290;Operating Equipment, Total;78;78
1310;Vehicles;49;49
1320;Increases during the Year;0;0
1330;Decreases during the Year;0;0
1340;Accum. Depreciation, Vehicles;-47;-47
1390;Vehicles, Total;2;2
1395;Tangible Fixed Assets, Total;1033;1033
1999;Fixed Assets, Total;1033;1033
2110;Resale Items;534;534
2120;Finished Goods;77;77
2130;Raw Materials;463;463
2180;Primo Inventory;0;0
```

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.



fixed format, and by carefully looking at the layout of the lines, these field starting positions displayed in the bottom line can be deduced:



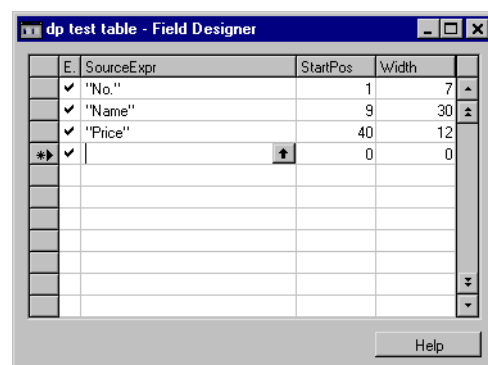
This tells us that the fields must have the following properties:

Field	StartPos	Width
No.	1	3
Name	4	11
Price	15	6

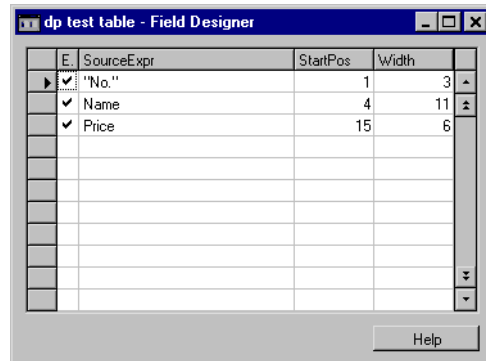
Once it has been established how the lines in the import file are organized, creating the dataport is straightforward:

- 1 Create a dataport as described earlier.
- 2 Set the Import property to Yes and the FileFormat property to *Fixed*.
- 3 Create a data item based on the table shown on page 357.
- 4 Add all the fields from the table to the dataport by using the Field Designer and the Field Menu as described earlier.

At this point, the fields will be set up with sizes (and corresponding starting positions), as deduced from the table design. It will look like this:



- 5 Change the setting of the **StartPos** and **Width** fields to the 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 (as described earlier):



- 6 Run the dataport.

Remember that if you run it 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 354).

After running the dataport so that records actually are imported into the database, the table will look like this:

No.	Name	Price
1	Chair	444.50
7	Pan	75.95
11	Tablecloth	49.95
12	Table	715.85
13	Doorknob	15.00
23	Jug	23.00
34	Doormat	30.00
47	Broomstick	25.00
99	Spoon	7.75
112	Oven	300.00

Note that because **Field No.** is the primary key of the table, the records are displayed in an order determined by this field, which, incidentally, is not the same order as they appeared in the import file.

#### Possible Errors

It is easy to make errors when deciding how to "cut up" the lines in the import file. In some cases, this will give a runtime error when the dataport is run. Consider, for example, if we made an error in setting up the **Field No.** field so that it had been assigned 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

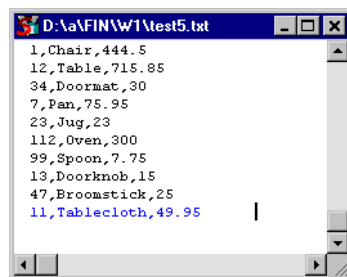
.....

In some cases, the order of the fields you want in your table does not correspond to the order of the fields in the import file. If this is the case, you can just design the data item to reflect the order that you need, by adding the fields individually. You will have to set the **StartPos** and **Width** manually.

.....

### Importing - Variable Format

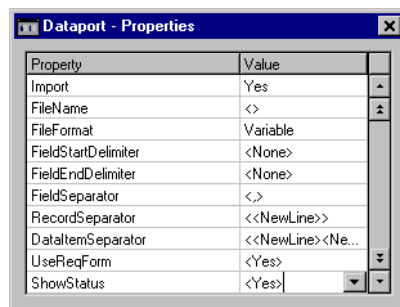
Changing the importing dataport to deal with an external file in variable format is not very difficult. Suppose the file looks like this:



The data is exactly the same as in the file with fixed format that we imported from above, but here the fields are separated by commas. The only differences in creating the dataport are:

- 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 will look like this when you have finished:

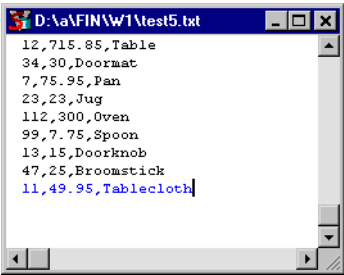


These are the only differences between the fixed format dataport and variable format dataport. The result of running this dataport will be the same as running the dataport with a fixed format.

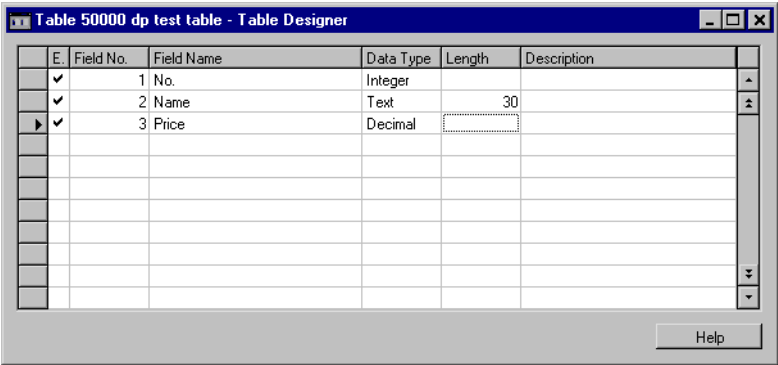
Hint

If the order of the fields in the import file is different from the order of the fields in the data item, you cannot use the same method to move the fields around as you would if the file had a fixed format. Instead, you can change the order of the fields in the Field Designer.

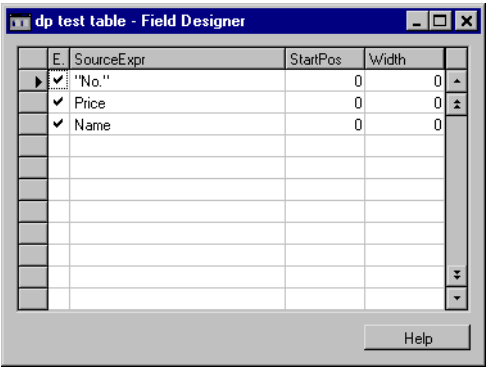
Suppose the import file had the following format instead:



Suppose the table has the following layout (same as before):



In this case, you can add the fields in the Field Designer in the order in which they appear in the import file:



Note that because the fields have been added one-by-one from the Field Menu (instead of all-at-once), **StartPos** and **Width** have not been calculated by the

designer. It does not matter, however, because these properties are not used in a dataport with a variable format.

### Importing or Exporting: A Dynamic Dataport

The final example is more advanced than the previous ones. We will create a dataport that can be used to update prices in the *Item* table in an external program, Microsoft Excel.

The dataport works as follows:

- 1 The user will be able to select whether to import or export from the request form, and to set a filename.
- 2 When exporting, only records where the **Gen. Prod. Posting Group** is RETAIL will be exported.
- 3 Only a subset of the fields will be exported – **No.**, **Description** and **Unit Price**.
- 4 In addition, the text field (**Description** field) that corresponds to the **Tariff No.** field will be retrieved from the **Tariff Number** table for each record in the *Item* data item. The text will be written as a fourth field when each record is exported.
- 5 The user is supposed to change the **Unit Price** field in the external program – or is at least able to do it. Therefore, we already know that this field in the external file may be different from that in the database. This means that we want the **Unit Price** fields from the external file to replace the values in the database when we import the file again. That is, we want the records in the database to be *updated* when we import the file.

### Creating the Export Part

We will start by creating the export part of the dataport. The first task is to determine what the format of the external file should be? We are going to edit the file in Microsoft Excel, so we will select a format that suits that program well. A comma-delimited format appears to be the best choice.

With this knowledge, we can start creating the dataport:

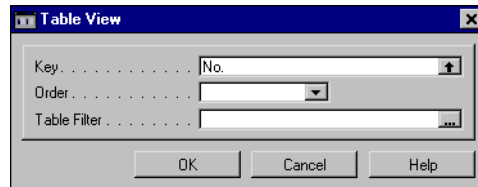
- 1 Create a new dataport as described above.
- 2 Set the properties of the dataport as follows:

Property	Value
FileFormat	Variable
FieldStartDelimiter	" (quote)
FieldEndDelimiter	" (quote)
FieldSeparator	, (comma)

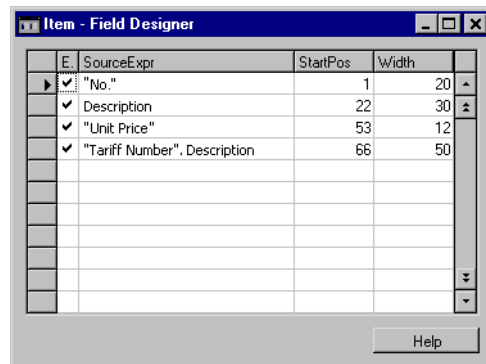
Leave the other properties at their default settings.



- 3 Create a data item based on the *Item* table.
- 4 Set the **DatalItemTableView** property of the data item as follows. Do not use Table Filter to select records even if we are going to select a subset of the records in the *Item* table. The reason will become evident later.



- 5 Create a global variable of type Record, with the *Tariff Number* table as the subtype.
- 6 Now, set up the dataport fields as follows:



- 7 To select those records where **Gen. Prod. Posting Group** is RETAIL, open the C/AL editor, and enter these lines in the OnPreDatalItem trigger:

```
IF NOT CurrDataport.IMPORT THEN
    Item.SETRANGE("Gen. Prod. Posting Group", 'RETAIL');
```

#### About Filters

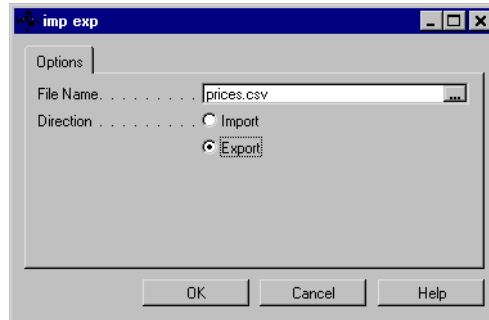
The filter is set only if the dataport is used to export (remember that the user can decide whether to import or export at runtime). The reason for using this construction instead of setting a TableFilter in **DatalItemTableView** is that we are putting the filter on a field that we are not exporting. If we had used TableFilter the filter would always be set, also during import. As the field is not in the import file, no records will be selected for import. If the field had been exported and imported, you could of course have used TableFilter instead.

- 8 To retrieve the text from the *Tariff Number* table and to export it, enter the following in the OnBeforeExportRecord trigger of the data item:

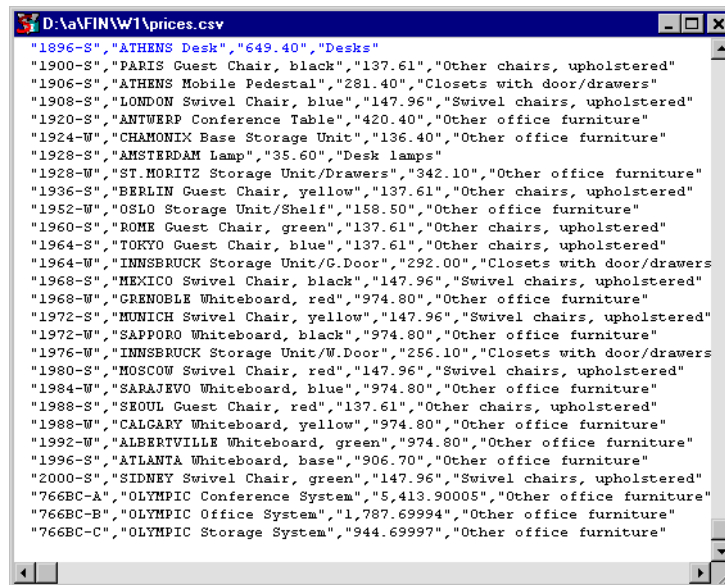
```
IF "Tariff No." <> '' THEN
BEGIN
    "Tariff Number"."No." := "Tariff No.";
    "Tariff Number".FIND;
END
```

```
ELSE
    "Tariff Number".Description := 'NO TARIFF NUMBER';
```

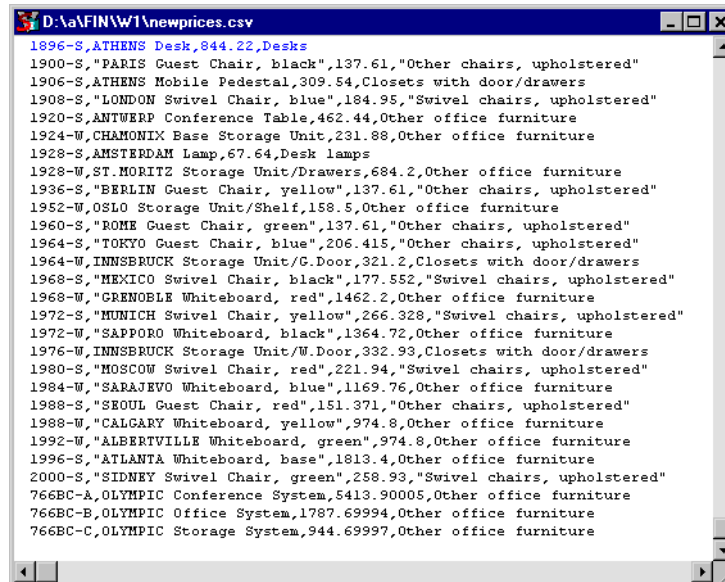
This is sufficient for creating the part of the dataport that exports the records. Let's try it out. When it is run, the user will see a request form, and can fill it out as follows:



When you click OK, the dataport will run and the records that match the criteria we have set up will be written to a file called `prices.csv`. It could look as follows in a text editor:



This file can then be opened and processed in (for example) Microsoft Excel. We have done that, and saved the result as a new comma separated file called `newprices.csv` that looks as follows:



```

D:\a\AFIN\W1\newprices.csv
1896-S,ATHENS Desk,844.22,Desks
1900-S,"PARIS Guest Chair, black",137.61,"Other chairs, upholstered"
1906-S,ATHENS Mobile Pedestal,309.54,Closets with door/drawers
1908-S,"LONDON Swivel Chair, blue",184.95,"Swivel chairs, upholstered"
1920-S,ANTWERP Conference Table,462.44,Other office furniture
1924-W,CHAMONIX Base Storage Unit,231.88,Other office furniture
1928-S,AMSTERDAM Lamp,67.64,Desk lamps
1928-W,ST.MORITZ Storage Unit/Drawers,684.2,Other office furniture
1936-S,"BERLIN Guest Chair, yellow",137.61,"Other chairs, upholstered"
1952-W,OSLO Storage Unit/Shelf,158.5,Other office furniture
1960-S,"ROME Guest Chair, green",137.61,"Other chairs, upholstered"
1964-S,"TOKYO Guest Chair, blue",206.415,"Other chairs, upholstered"
1964-W,INNSBRUCK Storage Unit/G.Door,321.2,Closets with door/drawers
1968-S,"MEXICO Swivel Chair, black",177.552,"Swivel chairs, upholstered"
1968-W,"GRENOBLE Whiteboard, red",1462.2,Other office furniture
1972-S,"MUNICH Swivel Chair, yellow",266.328,"Swivel chairs, upholstered"
1972-W,"SAPPORO Whiteboard, black",1364.72,Other office furniture
1976-W,INNSBRUCK Storage Unit/W.Door,332.93,Closets with door/drawers
1980-S,"MOSCOW Swivel Chair, red",221.94,"Swivel chairs, upholstered"
1984-W,"SARAJEVO Whiteboard, blue",1169.76,Other office furniture
1988-S,"SEOUL Guest Chair, red",151.371,"Other chairs, upholstered"
1988-W,"CALGARY Whiteboard, yellow",974.8,Other office furniture
1992-S,"ALBERTVILLE Whiteboard, green",974.8,Other office furniture
1996-S,"ATLANTA Whiteboard, base",1813.4,Other office furniture
2000-S,"SIDNEY Swivel Chair, green",258.93,"Swivel chairs, upholstered"
766BC-A,OLYMPIC Conference System,5413.90005,Other office furniture
766BC-B,OLYMPIC Office System,1787.69994,Other office furniture
766BC-C,OLYMPIC Storage System,944.69997,Other office furniture

```

As you can see, some prices have been changed and some have not. Now, we will create the part of the dataport that can import these new prices.

### Creating the Import Part

This part is very easy to create. The user decides when the dataport is run whether it should export or import, and can also select the file that it should write to or read from.

To specify that the records that are imported must update the existing records with the new prices, do the following:

- 1 Set the **AutoSave** property of the data item to Yes.
- 2 Set the **AutoUpdate** property of the data item to Yes.

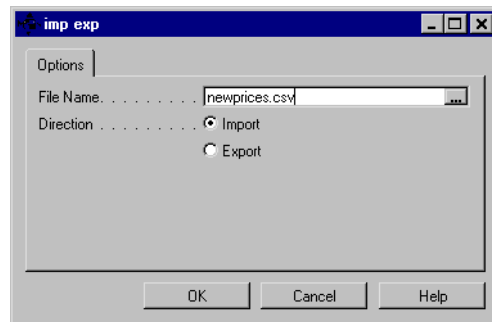
The net effect of these settings is to update the existing records with the data that is different in the imported records, in this case, the **Unit Price**.

Before the dataport is used to import the converted data, the records look this:



No.	Description	Unit Price	Gen. Pro...	Comment	Quantity on H...	Net Invo
1936-S	BERLIN Guest Chair, yellow	137.61	RETAIL		136	
1952-W	OSLO Storage Unit/Shelf	158.50	RETAIL		15	
1960-S	ROME Guest Chair, green	137.61	RETAIL		177	
1964-S	TOKYO Guest Chair, blue	137.61	RETAIL		113	
1964-W	INNSBRUCK Storage Unit/G.D...	292.00	RETAIL		63	
1968-S	MEXICO Swivel Chair, black	147.96	RETAIL		265	
1968-W	GRENOBLE Whiteboard, red	974.80	RETAIL		-22	
1972-S	MUNICH Swivel Chair, yellow	147.96	RETAIL		122	
1972-W	SAPPORO Whiteboard, black	974.80	RETAIL		11	

To use the dataport to import data, fill out the request form as follows:



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 look like this:

No.	Description	Unit Price	Gen. Pro...	Comment	Quantity on H...	Net Invo
1936-S	BERLIN Guest Chair, yellow	137.61	RETAIL		136	
1952-w	OSLO Storage Unit/Shelf	158.50	RETAIL		15	
1960-S	ROME Guest Chair, green	137.61	RETAIL		177	
1964-S	TOKYO Guest Chair, blue	206.415	RETAIL		113	
1964-w	INNSBRUCK Storage Unit/G.D...	321.20	RETAIL		63	
1968-S	MEXICO Swivel Chair, black	177.552	RETAIL		265	
1968-w	GRENOBLE Whiteboard, red	1,462.20	RETAIL		-22	
1972-S	MUNICH Swivel Chair, yellow	266.328	RETAIL		122	
1972-w	SAPPORO Whiteboard, black	1,364.72	RETAIL		11	

### Further Work

As it is now, this dataport has some drawbacks. For example, the **Unit Price** is used to calculate the **Profit %** field in the *Item* table when the **Unit Price** field is validated. We have not used the **CallFieldValidate** property to force 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, fields that are not part of this dataport. At the time of the validation, these fields do not have values because the updating takes place later. One way to solve this problem could be to export all fields, after all (thus making the field more difficult to handle in Excel). This problem shows that you have to give careful consideration to 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.





## Chapter 18

### Multilanguage Functionality

This chapter explains certain aspects of the multilanguage functionality of Navision.

The chapter contains information about the following subjects:

- Multilanguage Functionality
- Developing Multilanguage-Enabled Applications
- Learning the Code Base Language
- Number Ranges for Text Constants

## 18.1 MULTILANGUAGE FUNCTIONALITY

Everything to do with multilanguage functionality in C/SIDE in Navision runs automatically. Note that in order to avail of this multilanguage functionality, you must upgrade your application to the multilanguage-enabled Navision. For more information, see the manual *Upgrade Toolkit* on the Navision product CD.

### Defining the Current Application Language

C/SIDE executes the ApplicationLanguage function (trigger) 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 374.

For more information about language ID, see the section "The Windows Language Virtual Table" starting on page 372.

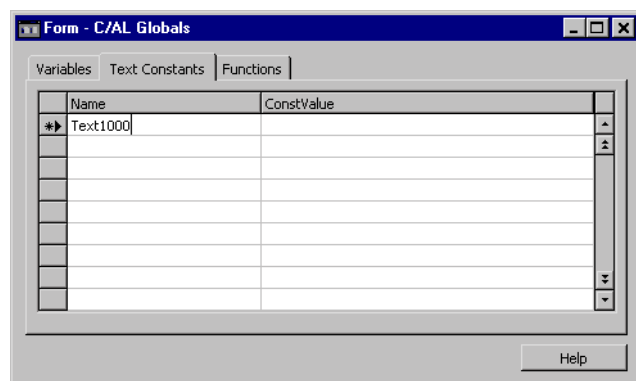
### Selecting a Language from the User Interface

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 hardcoded language dependent text strings.





For more information about the multilanguage use of text constants, see page 376, and for more information about creating text constants, see the section called "Defining Variables, Text Constants and Functions in Codeunits" on page 221.

## 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 372.

If you create a subdirectory for a language and then install the \*.stx, \*.etx, \*.chm and \*.hh files while Navision 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 using the Translate, Import item on the Tools menu. You can either export all text strings and translate them in a translation tool, such as the Navision Localization Workbench, or you can enter the translation of the text strings directly to the Multilanguage Editor.

You access the **Multilanguage Editor** window by opening an object from the Object Designer, opening the **Properties** window and clicking the AssistButton in the *CaptionML* property. In the **Multilanguage Editor** window, you can click the AssistButton in the Language column and choose your language from the list that is shown. You can also simply enter the three-letter code and move the cursor to the **Value** field. The system then replaces the abbreviation with the full language description. In the **Value** field, enter the correct term for this object in this language. To save your entry, you must click OK when leaving the window.

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 Navision 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 to a database, there is only one way to delete it again.

Click Tools, Language Module, Export. Select the language that will no longer be used in this database, 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
Language ID	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.

Field Name	Description
<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 Navision, 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 Navision.

If you select this option, external tools are able to use the 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 Navision Tables with External Tools" on page 36.

## C/ODBC

C/ODBC is multilanguage enabled. A C/ODBC user can retrieve the application data from Navision in different languages independent of the current Navision application language.

C/ODBC covers the following multilanguage features:

- Table name
- Field name
- OptionString value
- Date Formula

For more information, see the manual *C/ODBC 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 telling 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 will get the languages automatically.

If the user has documents that he wants to be printed in the language of the recipient rather than in his own working language, you may add a single line of code in the

document to handle this. This functionality is already enabled for most printing reports in the standard Navision database. Then 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 line as the first line in `OnAfterGetRecord()`:

```
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.

## 18.2 DEVELOPING MULTILANGUAGE-ENABLED APPLICATIONS

When you develop 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 Navision, the code base is English (United States). This means that the Name property of, for example, an object must always be 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 will automatically be 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 the example, `Text000` is an available name for a text constant in that object. The text constants must then be created as such in the **C/AL GLobals** window. For more information about creating text constants, see the section called "Defining Variables, Text Constants and Functions in Codeunits" on page 221.

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 18.4 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 382.

### 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.FIELDCAPTION("VAT Calculation Type")
```

where `VATPostingSetup` is the Name property in English (United States), and `FIELDCAPTION` makes sure that the Caption property in the relevant language is used rather than the Name property.

When you are programming, you should always have in mind 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 380.

### CaptionML Property

The *CaptionML* property is what 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
```

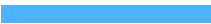
In the **CaptionML Value** field, you can either enter the value for English (United States) directly, or you can click the AssistButton to open the Multilanguage Editor and enter the value there.

Note that if you are creating a new field, you must enter the value for English (United States) in the **Name** field in order to get started.

Note also that you must click OK to save the information when you exit the **Multilanguage Editor** window.

**Note**  
.....  
In a form, if you have created a new field, 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 Navision.

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 will always be 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 options 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 data type `DateFormula` to the field. This new data type is non-language dependent and provides multilanguage capabilities to the `CALCDATE` function.

You achieve a similar result if you apply the data type `CODE` or `TEXT` to the field and then set the `DATEFORMULA` property to `Yes`. But 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);
```

# 18.3 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 variables so that you can compare the English name for the variable to the name in your local language.

You can use the Navision 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 Navision, 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 Navision Localization Workbench, see the manual *Navision 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), follow this procedure:

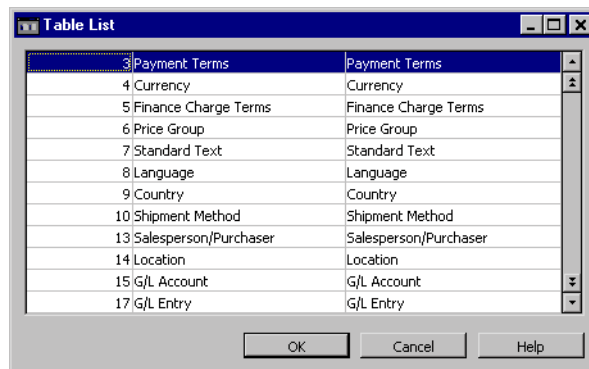
- 1 Open the object, for example, a purchase order, and place the cursor in the relevant field.
- 2 Click CTRL, F8 to zoom in.
- 3 In the **Zoom** window, right-click one of the column headers and select *Show Columns* from the list.

- 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.

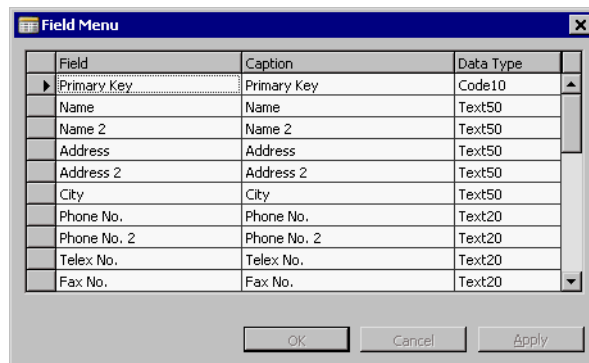
#### Table List, Form List, Field List, Object List and Field Menu

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 as shown in the following picture:



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 377.

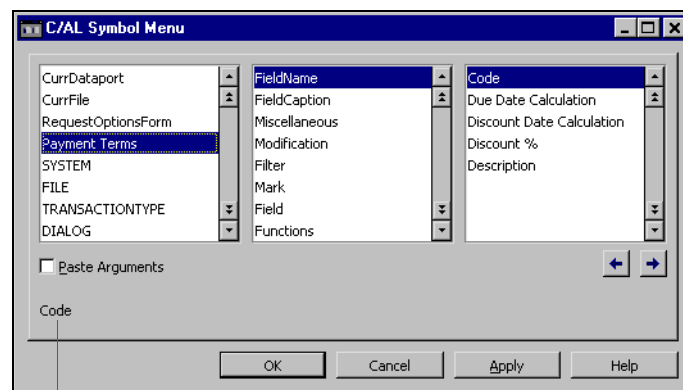
## C/AL Scanner

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 Navision 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 226.

## 18.4 NUMBER RANGES FOR TEXT CONSTANTS

C/SIDE assigns unique IDs to text constants according to the following table of number ranges:

Developer	From	To
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

<b>Developer</b>	<b>From</b>	<b>To</b>
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
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

Developer	From	To
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.









## Chapter 19

### 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

## 19.1 TYPE CONVERSION IN EXPRESSIONS

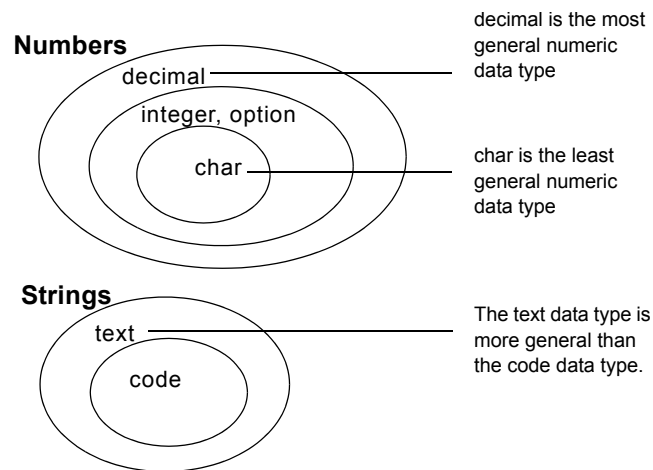
Consider the following statements:

```
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 246.)

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", as defined below.



- The most general data types include all 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 type. In order to add these, the system will 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  $\Rightarrow$  decimal`

**EXAMPLE 2**

Evaluation of a string expression:

`text + code`

This expression contains two sub-expressions to be concatenated. To do this, the system will 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  $\Rightarrow$  text`

19.2 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 table below defines the evaluation rules for relational operators. The rules assume that the data types of the expressions can be compared. Refer to Valid Uses of Relational Operators below 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 else 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 the above 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 395, which provide a full description of all possible uses of C/AL operators and the resulting data types.

**EXAMPLE**

The table below 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 above, a runtime error will occur if date is a closing date or if date is undefined (0D).

#### 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
+	○	●	●	●	●	●	●	●	●
-	○	●	●	●	●	●	●	○	○
*	○	●	●	●	●	○	○	○	○
/	○	●	●	●	●	○	○	○	○
DIV	○	●	●	●	●	○	○	○	○
MOD	○	●	●	●	●	○	○	○	○

● 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) (C)</sup>	date <sup>(A) (C)</sup>	date <sup>(A) (C)</sup>	date <sup>(A) (C) (D)</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>	-	-	-	-
<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 THE ABOVE 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 20

### SumIndexFields

This chapter describes SumIndexFields™, which are the basis for the FlowFields in a C/SIDE application. This chapter describes how SIFT™ works on Navision Database Server as well as some details of the way that SIFT is implemented in the SQL Server Option for Navision. This information will help programmers develop efficient applications that use SumIndexField Technology.

- SumIndexFields
- SIFT and the SQL Server Option for Navision

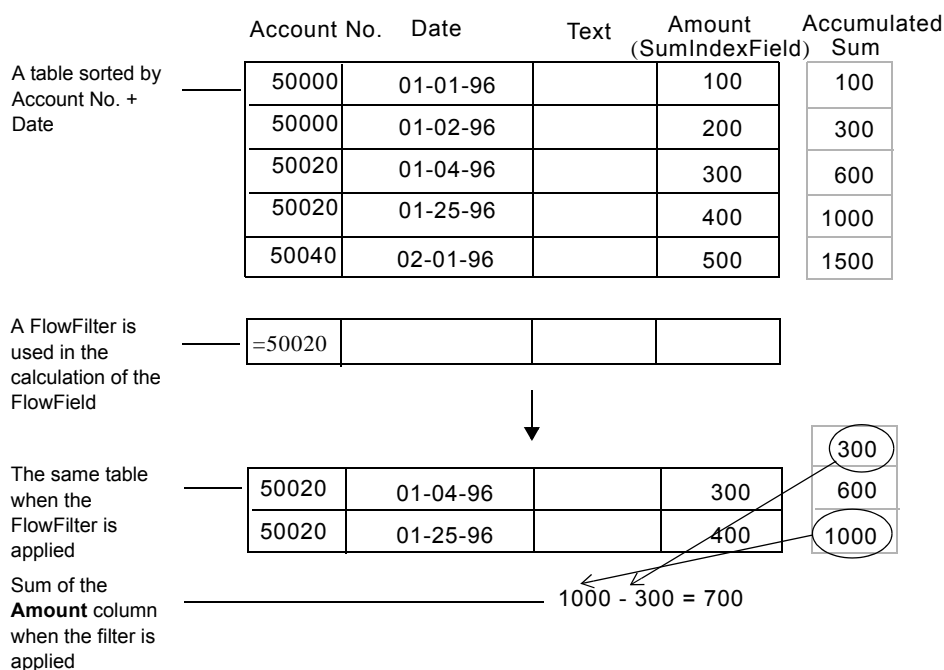
## 20.1 SUMINDEXFIELDS

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 Navision 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 Navision application and has also been implemented in the SQL Server Option.

### SIFT and Navision 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 figure below 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 above 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 above 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 Navision 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.

## 20.2 SIFT AND THE SQL SERVER OPTION FOR NAVISION

As mentioned earlier, SIFT has also been implemented in the SQL Server Option for Navision. 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 Navision 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. Navision 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 Navision 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 Navision. This is known as a SIFT trigger. A base table is a standard Navision table, as opposed to an extra SQL Server table that is created to support Navision 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 Navision 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 Navision.

## SIFT Triggers

The body of the SIFT trigger is generated by Navision 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 Navision 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

Enabled	Key	SumIndexFields	MaintainSIFTIndex
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.

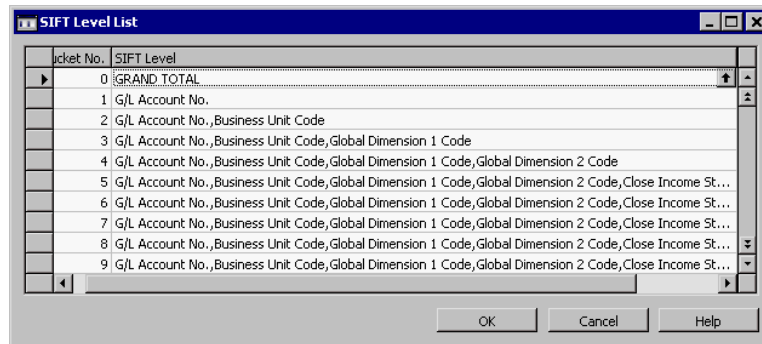
After the bucket column, comes a set of columns with names that start with the letter "f". These are also known as f- or key-columns. Each of these columns represents one field of the SIFT key. The name of these columns has the following format: *f<Field No.>*, where Field No. is the integer value of 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 example above).

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 Navision. 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 Navision, 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 418.

### 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 420.

.....

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 <b>f2</b> (Item No.). (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 example shown above 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, Navision 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 Navision 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.

Index	Clustered	Index_Keys
\$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 405. 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 Navision 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.

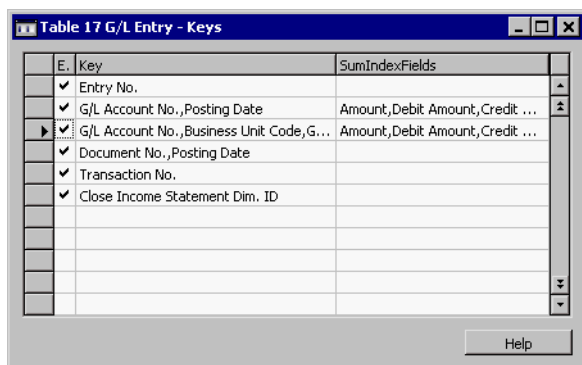
If you no longer want to maintain these SIFT structures, you must set the *MaintainSIFTIndex* property for this key to *No*. For more information about the factors that must be taken into consideration before deciding whether or not to maintain these structures, see page 420.

To change the SIFT levels that are maintained:

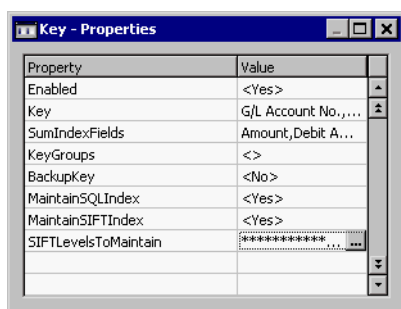
- 1 Open the Object Designer and click Table.
- 2 Select the table that contains the SIFT indexes that you want to modify and click Design. In this example we are using table 17, **G/L Entry**.



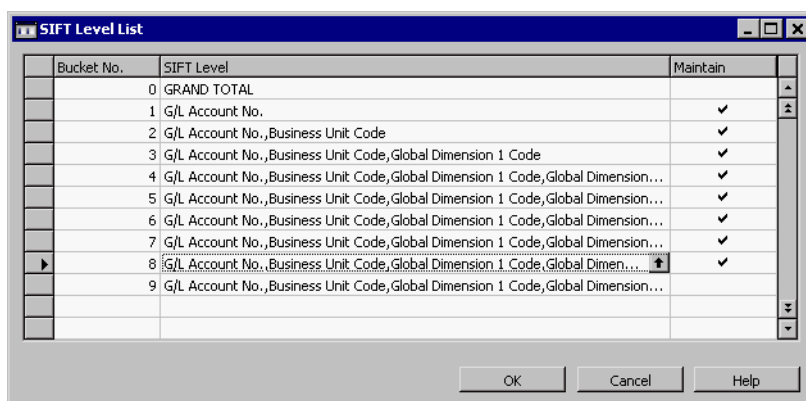
**3** Click View, Keys to open the **Keys** window for this table:



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.



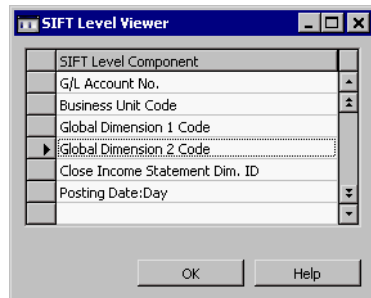
This window lists all the SIFT levels and their components that have been created to support the SumIndexFields associated with this key.

**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<sup>+</sup> 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.

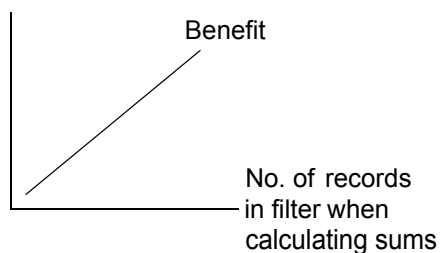
Maintaining the most detailed SIFT level is not recommended because you will not need totals that are this detailed often enough to justify the cost to performance.

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.

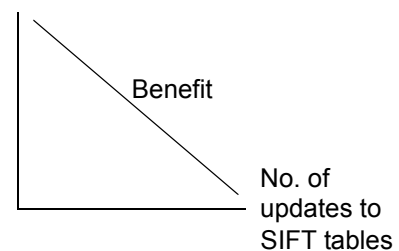
Consider the costs and the benefits of maintaining SIFT tables and SIFT levels:

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 Navision. 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 Navision 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: Microsoft Business Solutions–Navision SQL Server Option*.

## Chapter 21

### Numbering in Navision

This chapter explains how items, such as documents, are numbered in Navision. The information is helpful if you use or are planning to use the SQL Server Option for Navision. We also recommend that you read this chapter if you use Navision Database Server and want Navision to sort numbers correctly when you view data with external programs.

This chapter contains the following section:

- How Does Number Sorting Work?

21.1 HOW DOES NUMBER SORTING WORK?

Code fields in Navision can contain both numerical values and text strings. Navision ensures that numbers kept in code fields on Navision 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 Navision 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 Navision 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 Navision does not permit numbers that start with zero.

Furthermore, the SQL Server Option for Navision 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 Navision. 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 22

### C/SIDE in Multiuser Environments

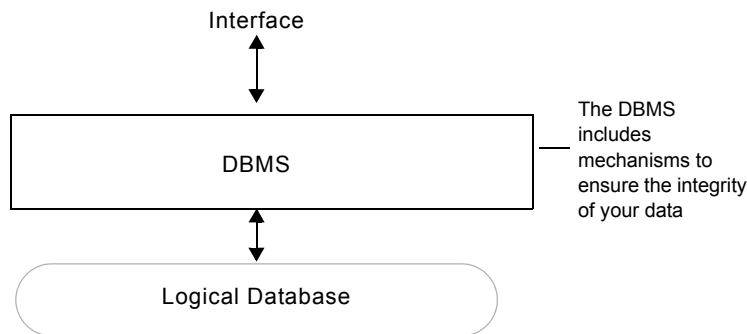
This chapter explains how the DBMS handles data integrity in a multiuser environment. It describes how the C/SIDE system handles situations where more than one user or process try to change the same object.

This chapter contains the following section:

- Ensuring Data Integrity in a Multiuser Environment
- Locking in Navision – a Comparison of the two Server Options

## 22.1 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 figure below:



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 means to obtain data integrity are centered around a few basic 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 completed either 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 transaction operations are applied to the database while 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.
- Missing information makes it impossible to complete the transaction.
- 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 tables are restored to the state they had before the transaction started.

A typical example of a write transaction is illustrated by a banking system where \$100 must be transferred 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. By bundling both operations into a single transaction, 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 in C/SIDE will be consistent 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 from failures.

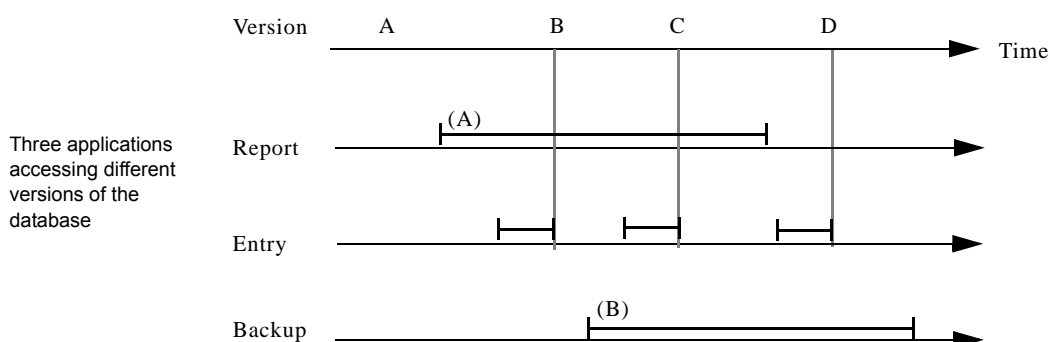
Assume that an application program aborts because of power failure or is aborted by the operator. The database is now in an inconsistent state, and all modifications already made to the database must be cancelled. In common database systems this is achieved by so-called *rollback recovery*, that is, 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 figure below 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, the generation of a 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 as 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 in 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. Only when you commit your updates do your modifications become public. Your newly-committed updates plus the part of the database which was not modified make up the newest version.

### What is a Data Version?

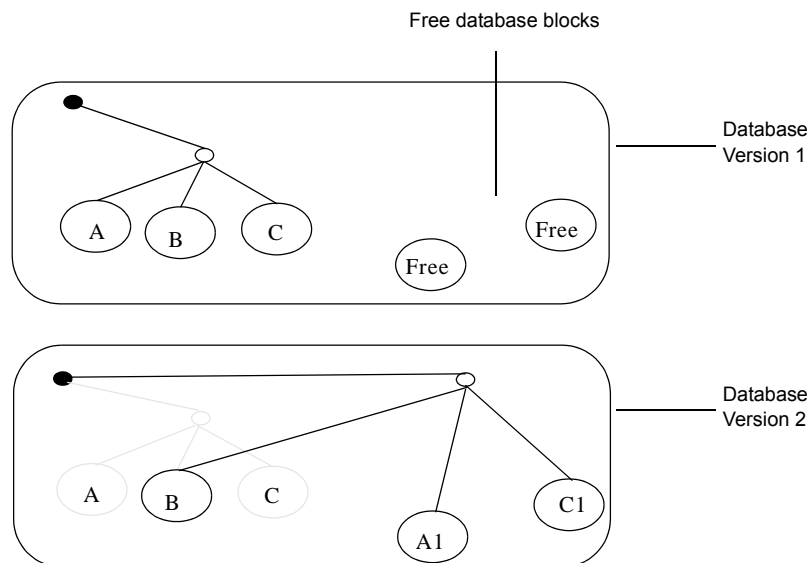
The data in the C/SIDE database is stored in a well known data structure that resembles a tree. A tree data structure is formed of nodes. Each node in the tree, except for a special node called the root, has one parent node and one or more child nodes. The root node has no parent. A node that has no 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. Furthermore 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 figure below, 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.

As order entries are added to the database, newer versions of the database are created. 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.

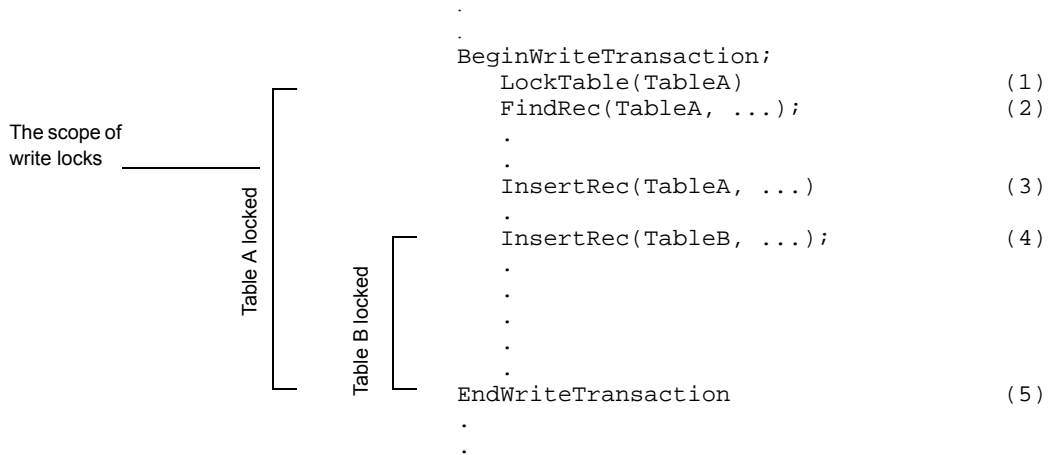
This conflict is solved by the DBMS by giving priority to the write transaction and ejecting application A. A runtime error message is sent to A on its next read operation—"Data version is no longer valid" – and it is forced to restart the process with the newest version. But as long as B continues and 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 certain a write operation will be performed. 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 Navision 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.

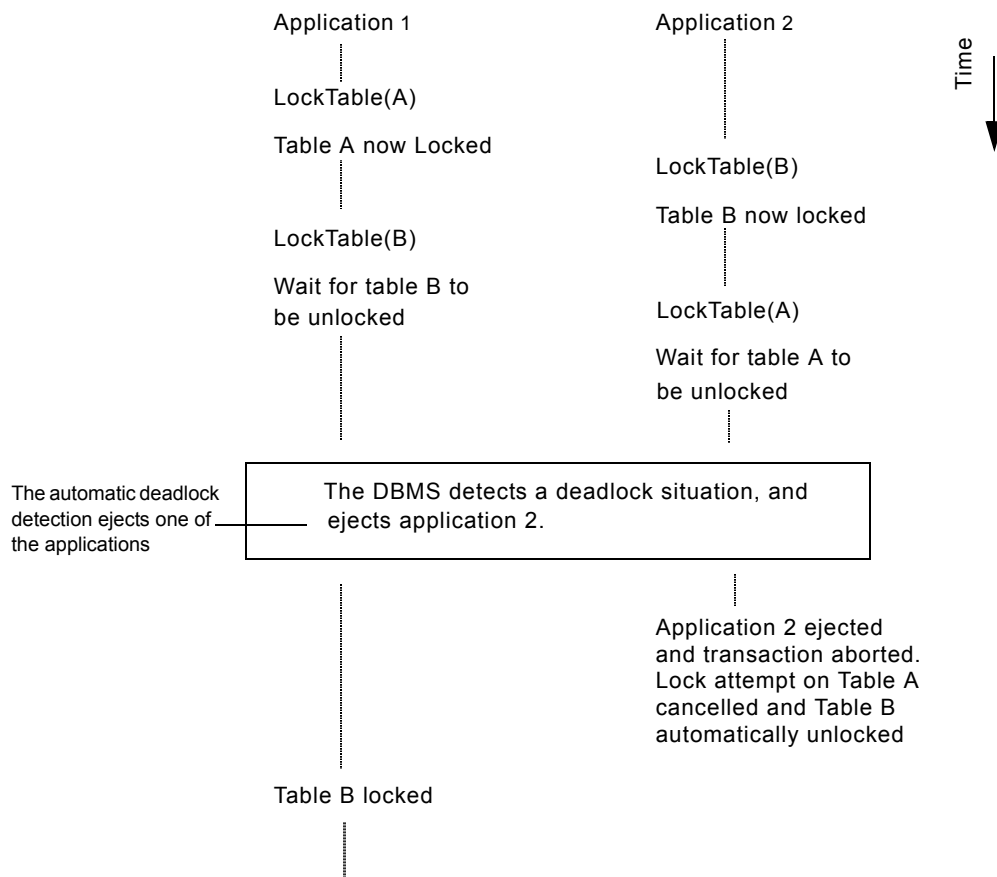


This illustrates both an *explicit lock* and an *automatic lock*. Line (1) in the write transaction explicitly locks table A. If this explicit lock were not set on table A, the DBMS would automatically lock this table when a record is inserted (3). Table B is not locked explicitly, but is locked automatically 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?

The correct functioning of a multiuser system will depend on the coordination of the activities. If transaction processes require write access to several tables at once, care must be taken to avoid the situation where one transaction process can obtain access to some of the necessary tables and another can obtain others of them, but neither of them can proceed without the other having finished. This causes each transaction process to wait for the other to finish. As a result both processes will have to wait forever. Such a situation is known as a *Deadlock* (or as *Deadly Embrace*).

In order to avoid deadlock situations, the DBMS has been provided with an automatic deadlock detection mechanism, which detects these situations and ejects one of the write transactions. The next figure illustrates how a deadlock situation can arise.



The DBMS will always eject the application which causes the deadlock to occur—as in the example above. This rule applies for any number of applications 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 makes it possible 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.

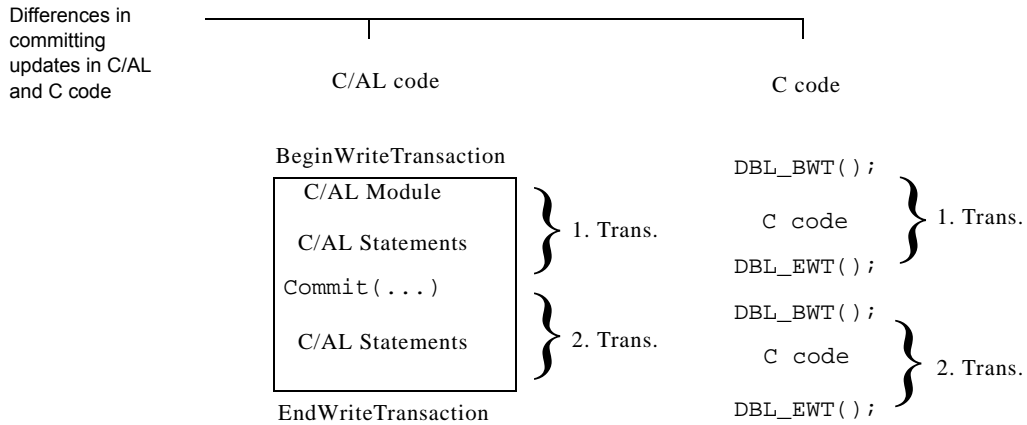
When you want to perform an update using C/FRONT, the first thing you must do is to tell the system explicitly that you want to perform a write transaction (use `DBL_BWT`, `BeginWriteTransaction`.) Likewise you must use `DBL_EWT` (`EndWriteTransaction`) to explicitly tell the system when your write transaction ends.

When you use C/AL code to perform updates to a C/SIDE database, these `BeginWriteTransaction` and `EndWriteTransaction` statements are handled implicitly by the system. That is, the system automatically executes these commands before the C/AL code is entered and after the C/AL code has been 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. As the execution of the C/AL code begins, write transactions are automatically enabled. By issuing the command `COMMIT()`, 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 has been completed, the system automatically ends the second write transaction. When you use C code to perform the same transactions, each transaction must explicitly be encapsulated by `DBL_BWT()` and `DBL_EWT()` commands.

## 22.2 LOCKING IN NAVISION – 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 Navision: Navision Database Server and the SQL Server Option.

### Important

.....  
The following information only covers the default transaction type `UpdateNoLocks` for the SQL Server Option for Navision. For information about the other transaction types, see the online C/SIDE Reference Guide.  
.....

### Both Server Options

Locking	In the beginning of a transaction, the data that you read will not be locked. This means that reading data will 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 will 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 of 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	<p>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.</p> <p>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.</p>

### 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 on 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 Navision Database Server

Data that is not locked will be read from the same snapshot (version) of the database.

If you call or a modifying function (for example, `INSERT`, `MODIFY` or `DELETE`), on a table, the whole table will be locked.

**Locking Record Sets** As mentioned previously, you will normally 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. With Navision 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 will receive an error message if another transaction has modified the records in the meantime.

You will 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 postpone the table lock that Navision Database Server puts on the table. This improves concurrency.

## Locking in SQL Server

When data is read without locking, you will get the latest (possibly uncommitted) data from the database.

If you call `Rec.LOCKTABLE`, nothing will happen right away. However, when data is read from the table after `LOCKTABLE` has been called, the data will be locked.

If you call `INSERT`, `MODIFY` or `DELETE`, the specified record will be locked immediately. This means that two transactions, which either insert, modify or delete separate records in the same table will not conflict. Furthermore, locks will also be 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 will 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 will be locked. This means that two transactions, which read specific, but separate, records in a table will 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) will be locked as you browse through it. However, the locking implementation used in SQL Server will also cause the record before and the record after this record set to be 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 will 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 Navision only supports the default values for the parameters of the `LOCKTABLE` function - `LOCKTABLE( TRUE , FALSE )`.

**Note**  
.....  
Navision 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 Navision 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 Navision table without causing conflicts.  
.....

Locking Differences in the Code

A typical use of `LOCKTABLE( TRUE , TRUE )` in Navision Database Server is shown in the first column of the table below. 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 Navision. This is currently the only server that supports record level locking.

Navision 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 23

### 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

## 23.1 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 upon by the system every time the field or control is to be 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 will be passed to this function when called upon:

- LANGUAGE
- CAPTIONEXPR

**LANGUAGE**      The LANGUAGE parameter is automatically mentioned by the system and is the Windows Language ID of the active language in Navision.

### EXAMPLE

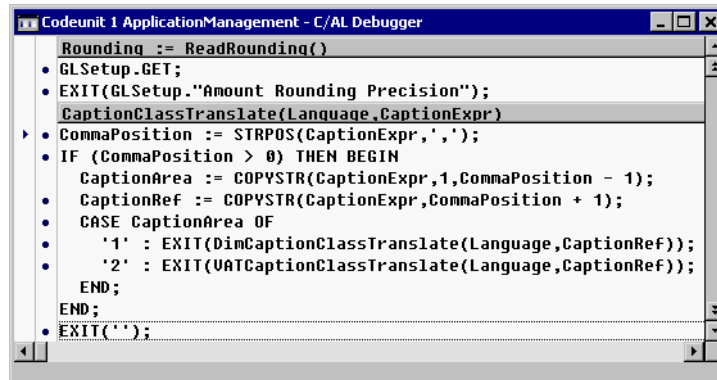
If the active language in Navision is English (United States), LANGUAGE will hold the value 1033.

**CAPTIONEXPR**      The CAPTIONEXPR parameter will hold the content of the CaptionClass property of the field or control.

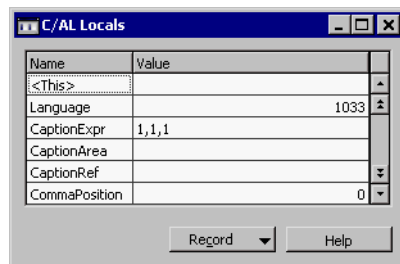


**EXAMPLE**

In Table 13 (*Salesperson/Purchaser*), the field **Global Dimension 1 Code (5050)** has the string '1,1,1' as its CaptionClass. If we open the *Salespeople/Purchasers* form, activate the debugger and type Ctrl + F8 (to open the **Zoom** window), the CaptionClassTranslate function trigger will pop up.



In the **C/AL Locals** window, we can see that the CaptionExpr parameter holds the string '1,1,1'.

**Note**

.....

In the Zoom window, we will not find the field **Global Dimension 1 Code**. Instead we see the field name **Department Code** – as a result of the CaptionClass property and the CaptionClassTranslate function trigger.

.....

**Function Code**

In a way we could say that the function trigger CaptionClassTranslate (ID 15) is a system trigger. Here a programmer can intervene every time the CaptionClass property – if defined – is evaluated by the system.

If, in a standard CRONUS database, we take a look at this trigger, we see that there already has been written code in it:

```

CaptionClassTranslate(Language : Integer;CaptionExpr : Text[80]) :
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 above, this parameter has the syntax:

```
CAPTIONEXPR := <CAPTIONAREA>, <CAPTIONREF>
```

Depending upon the value of the CAPTIONAREA, different procedures will be called upon. Look at the CASE ... OF statement, either:

```
DimCaptionClassTranslate(Language, CaptionRef)
```

or

```
VATCaptionClassTranslate(Language, CaptionRef)
```

In the following, there is a detailed description of these functions.

**CAPTIONAREA** So the first part of the CaptionExpr parameter, up to the first comma, is the CAPTIONAREA, with the 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** So the second part of the CaptionExpr parameter, after the first comma, is the CAPTIONREF, with the 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 Navision deals with the CaptionClass property. Every field or control with a defined CaptionClass has a string in this property with syntax as described above. 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 above, 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. Below 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 syntax below 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

```

## 23.2 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 will pass the Language ID and the CAPTIONREF part of the CaptionClass property.

This function can be split up 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);

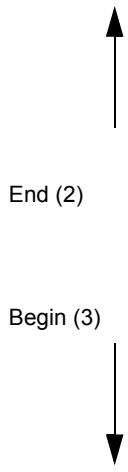
CommaPosition := STRPOS(DimOptionalParam1,',');

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;
            END;
        '2':
            BEGIN
                (same as case '1' for field "Global Dimension 2 Code")
            END;
        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
    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;
        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;

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 up 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 24

### Supporting Record Level Security

This chapter outlines some factors that must be taken into consideration when programming for the SQL Server Option for Navision.

- Record Level Security

## 24.1 RECORD LEVEL SECURITY

The SQL Server Option for Navision allows you to limit the access that users have to the information stored in the database by specifying that they can only see specific records. This is called record level security and this section describes how to support record level security in the code, that is, the specific "extras" you must have in the code in order to support record level security.

Navision automatically applies record level security filters in most situations once they have been set up. This means that the users will only receive 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 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 to 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.

### 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 using 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.

## Chapter 25

### 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.

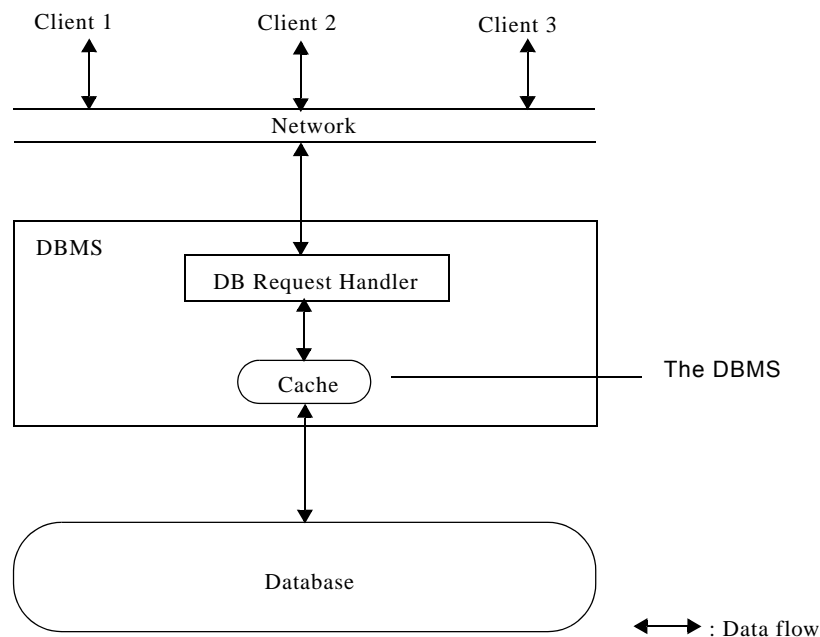
- The DBMS Cache
- The Commit Cache
- The Command Buffer
- Keys, Queries and Performance
- C/AL Database Functions and Performance on SQL Server
- Configuration Parameters
- Bulk Inserts

## 25.1 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 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 may 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 will be 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 will be. (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 from 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 Commit Cache on page 460 for more information.  
 .....

See C/SIDE Specifications on page 475 for information about the maximum cache size.

## 25.2 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.

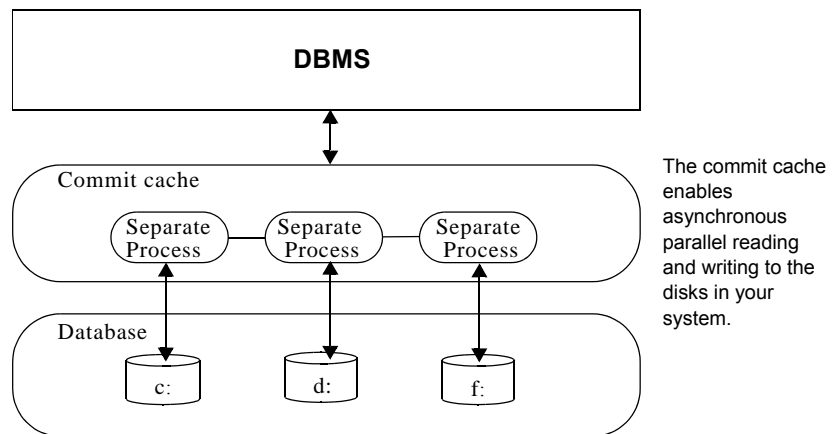
As described in the section The Physical and the Logical Database on page 8, the logical database can be stored in several distinct disk files (which can be stored on separate disks). When more than one disk is used to store the database, each of these disks is controlled by separate commit cache processes, 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 committed transactions 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 cause corruption of 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.



## 25.3 THE COMMAND BUFFER

The command buffer only applies to Navision Database Server, and is placed as a link between your application and the DBMS in the C/SIDE system. It is a temporary place of 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, commands that do not have to return a value for the execution of the C/AL code to continue, do not have to be sent immediately to the DBMS.

### 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 statements below, supposed to be the complete contents of a codeunit, illustrate the difference between running code with and without the debugger:

```
Customer."No." := '12';
Customer.DELETE();
First := 7;
Second := 0;
Ratio := First / Second;
```

Now, suppose that there is no Customer with the number 12. A runtime error will occur 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 fail. Further, the last statement is a division by zero.

When the debugger is *inactive*, the `DELETE` command is absorbed by 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 Customer record cannot be found.

## 25.4 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 Navision 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 Navision 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 Navision cannot locate and read the subset efficiently, performance will deteriorate. In the worst case scenario, Navision 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 Navision Database Server. However, Navision 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 Code** as the first field and **Customer No.** as the second field and if there are only a few different country 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 Code, Customer No.,Posting Date
```

The query could look like this:

```
SETCURRENTKEY("Country 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 Navision Database Server on the way that keys are defined in tables and on the way they are used in queries.

## 25.5 C/AL DATABASE FUNCTIONS AND PERFORMANCE ON SQL SERVER

The fastest SQL statement that Navision sends to SQL Server runs slower than most database functions on Navision 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 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 below, 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 Navision Database Server. Similarly, translating and renaming tables and fields are slower on SQL Server.



# 25.6 CONFIGURATION PARAMETERS

You can configure a Navision database by creating a SQL Server table configuration parameter table and entering parameters into the table that will determine some of the behavior of Navision when it is using this database.

In the database create a table, owned by dbo:

```
CREATE TABLE [$ndo$dbconfig] (config VARCHAR(512) NOT NULL)
```

(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 below, 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 Navision to use index hinting for such problematic queries. When you use index hinting, Navision 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 Navision 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 Navision 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`, Navision will override SQL Server and place `ROWLOCKS`.

## 25.7 BULK INSERTS

The SQL Server Option for Navision uses bulk data insertion to improve performance.

After 5 records are inserted into a table, bulk insert mode begins on the table and continues until the mode is cancelled. Inserts are buffered in memory on the client, and sent to the server in batches. As each batch is sent, it is flushed from the buffer. If the table contains SIFT keys, the batches are inserted into a temporary table and then copied to the target table with an `INSERT . . . SELECT` statement in order to optimize the execution of the SIFT trigger.

The requirements for beginning and continuing in bulk insert mode are:

- No other operations can be performed on the table that is receiving the inserts, but any operation can be performed on other tables. For example, when records are copied from table A to table B, the read operations being performed on table A will not disturb the bulk insert mode of table B.
- A `COMMIT` or intervening operation will flush any buffered inserts but bulk insert mode will continue on the table for a `COMMIT`.

The bulk insert mode is cancelled and disabled if you:

- Activate the C/AL debugger.
- Test the result of an `INSERT` with `IF`.
- Set either of the Dataport Dataltem properties *AutoUpdate* or *AutoReplace* to *Yes*.









## 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	16 x 2 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	1000 MB

## 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 (Navision 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 legend below, 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

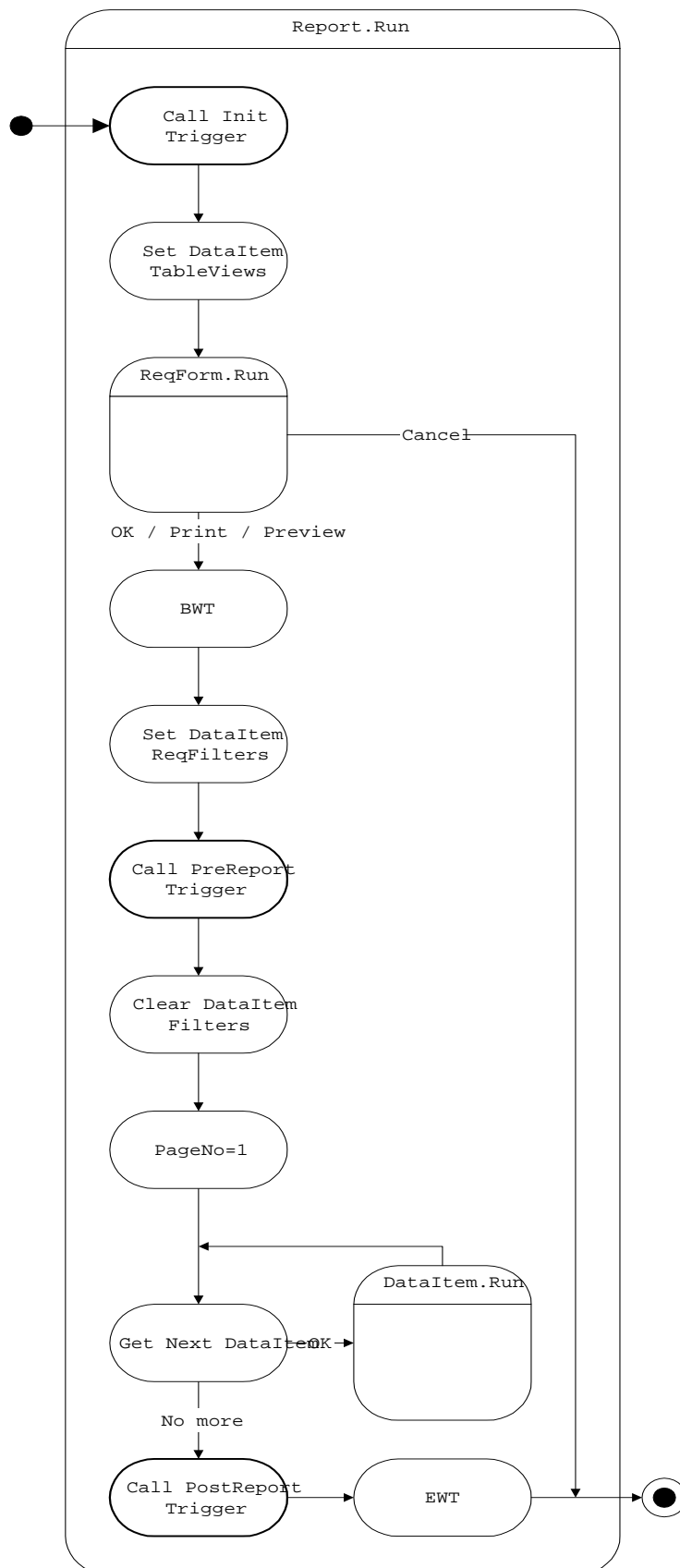
BWT

Begin Write Transacti

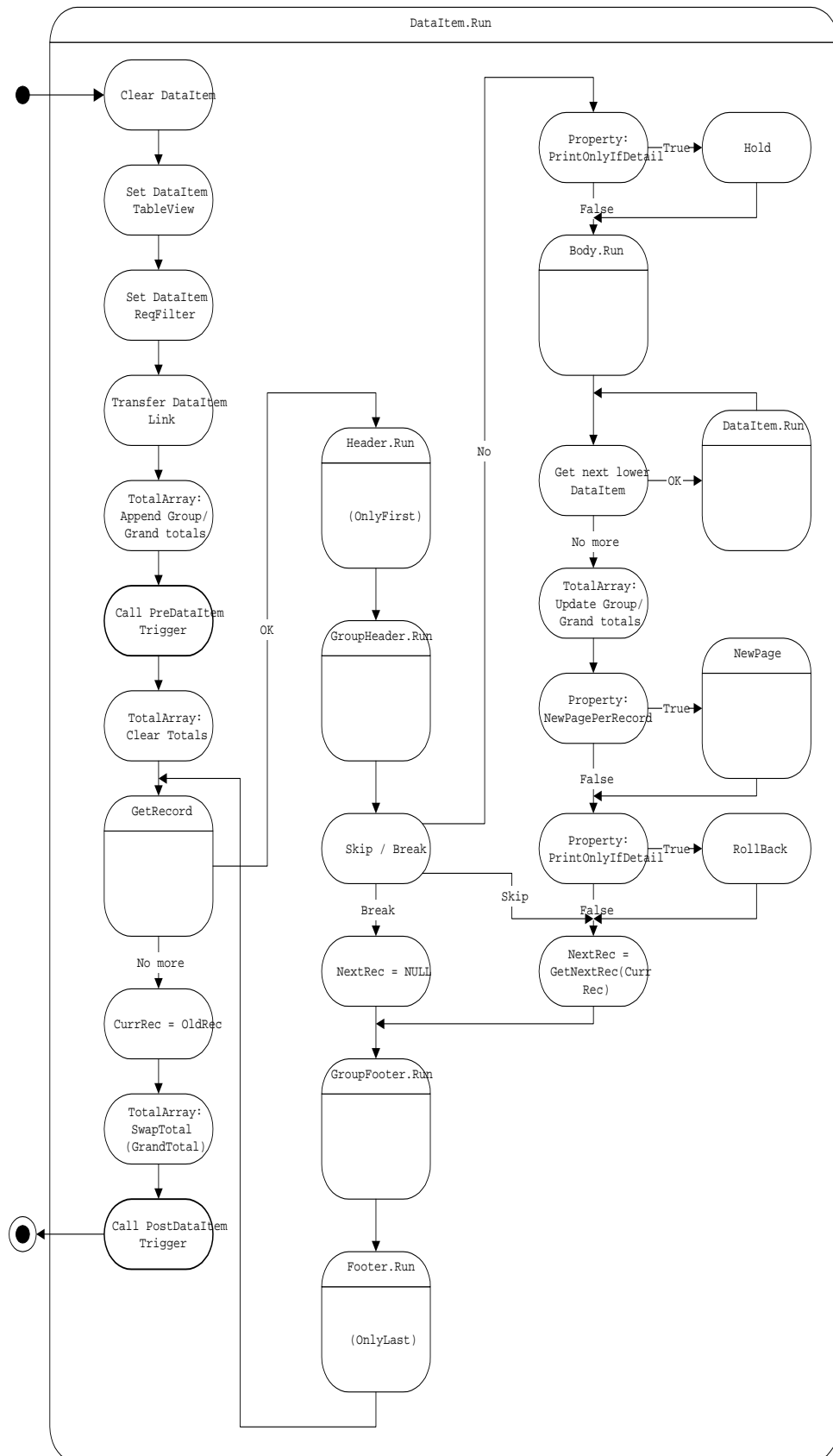
EWT

End Write Transactor

## B.2 REPORT.RUN

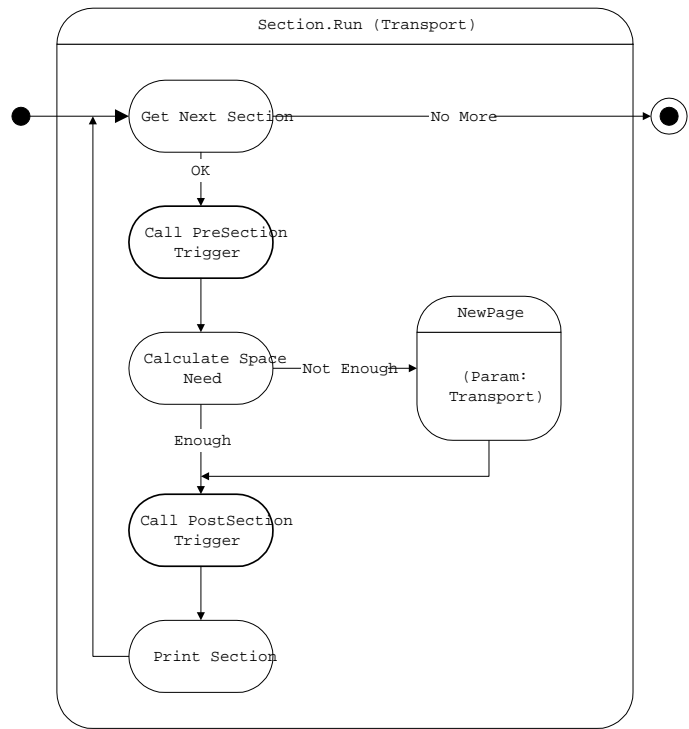


## B.3 DATAITEM.RUN

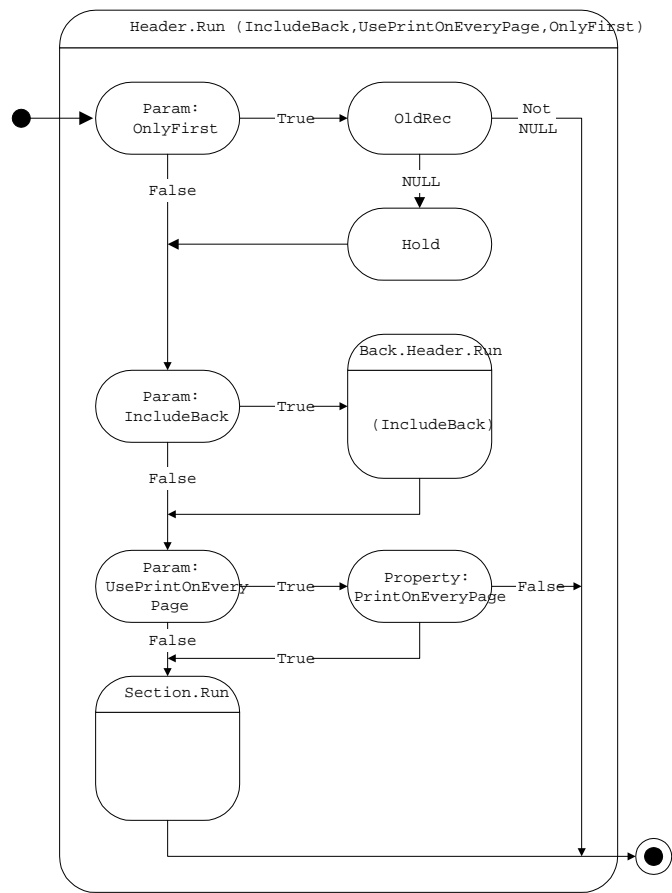




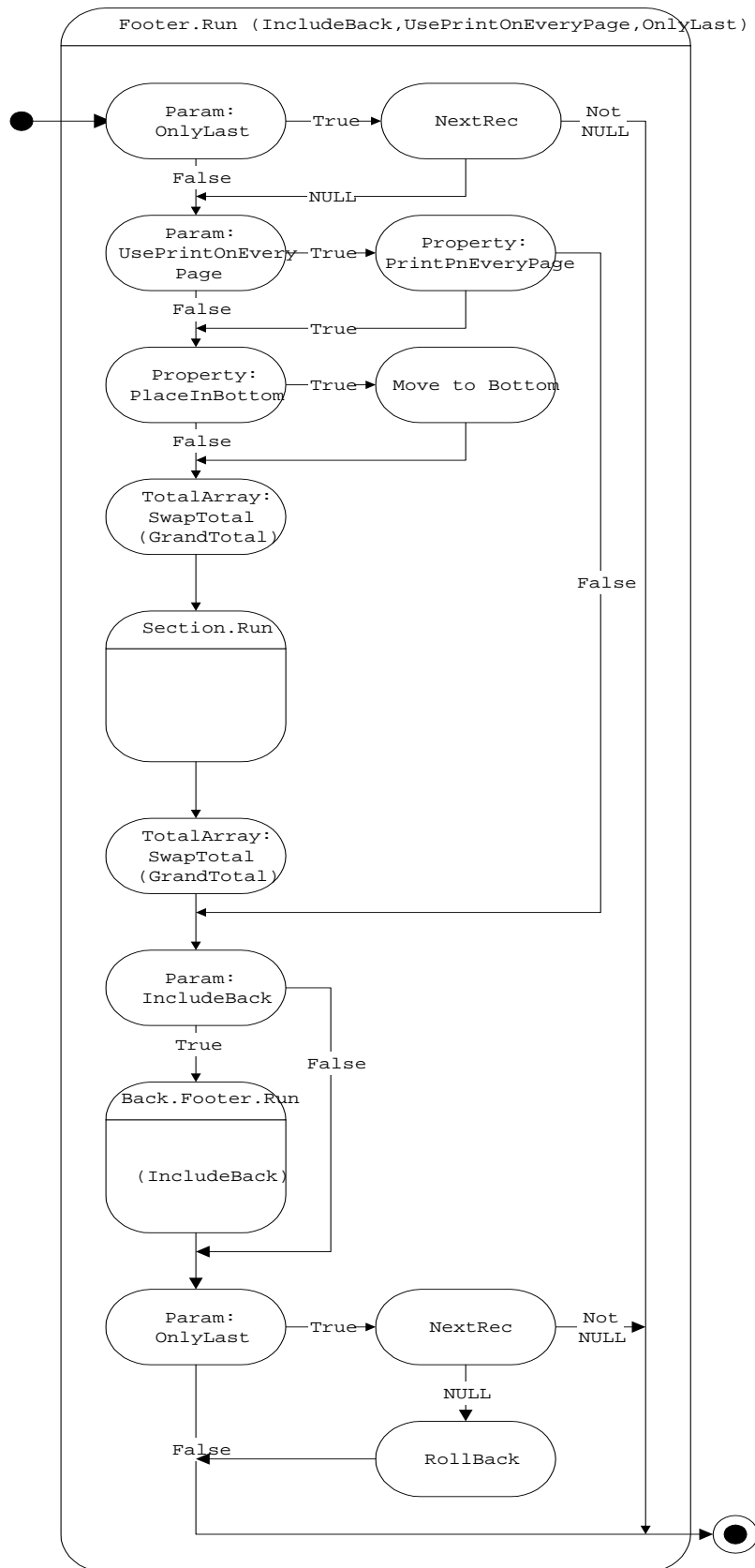
B.4 SECTION.RUN



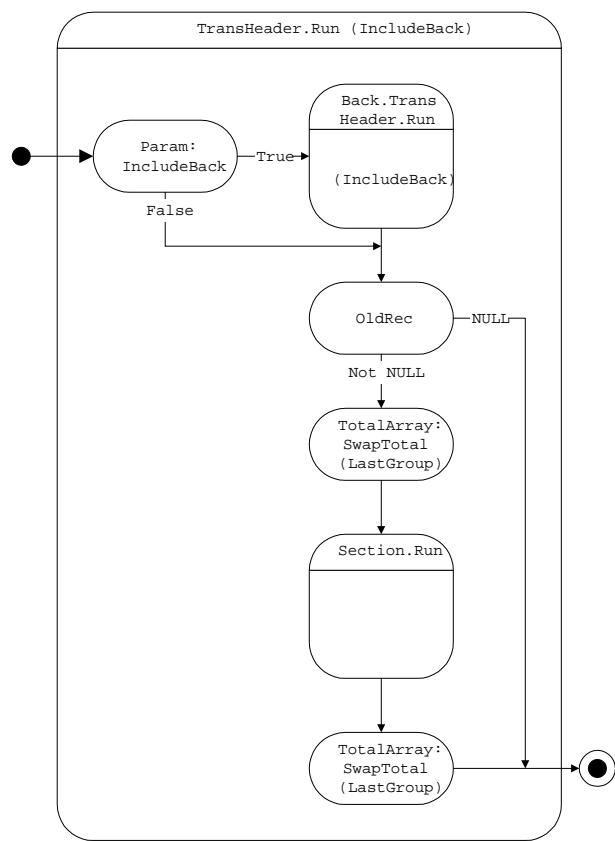
B.5 HEADER.RUN



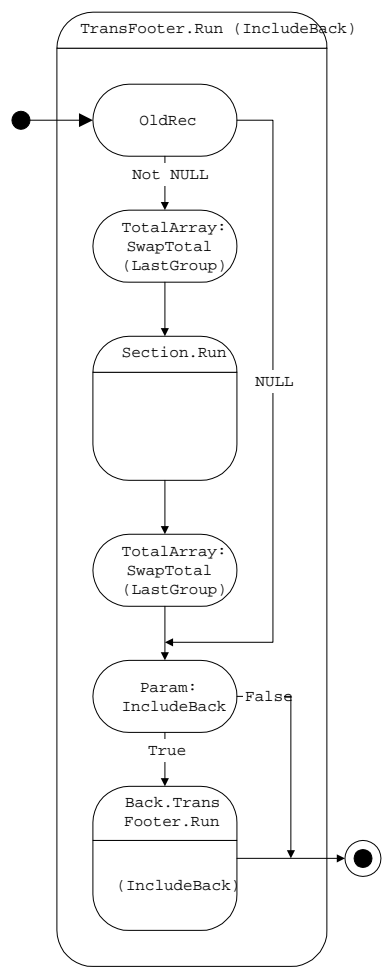
## B.6 FOOTER.RUN



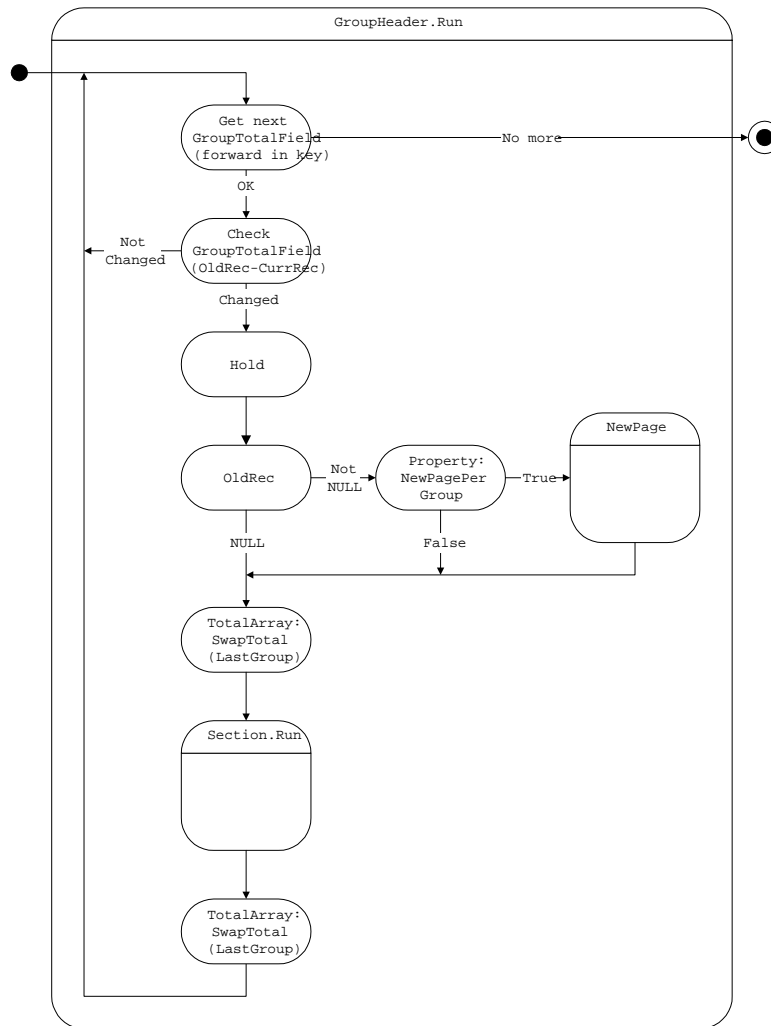
B.7 TRANSHEADER.RUN



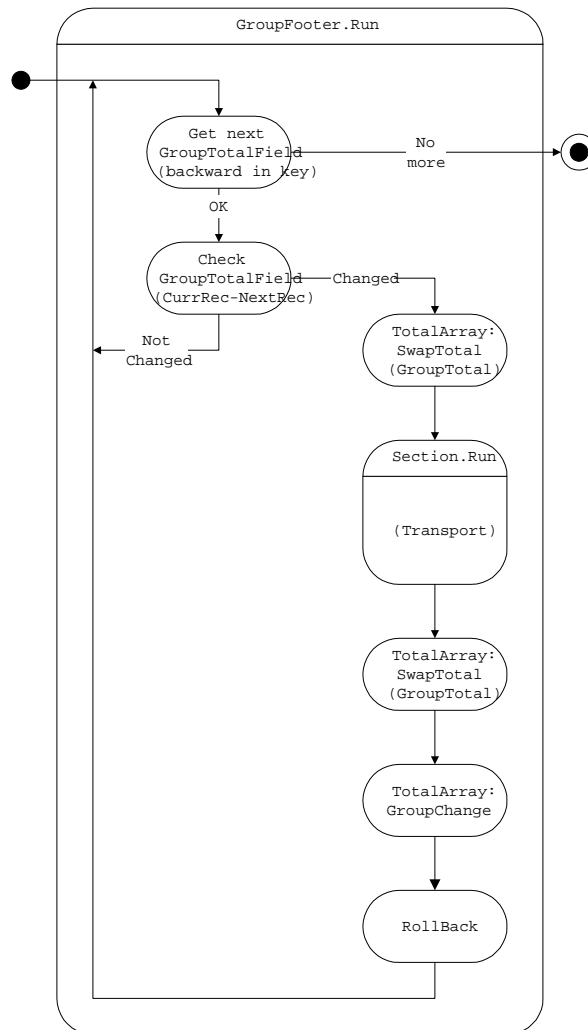
B.8 TRANSFOOTER.RUN



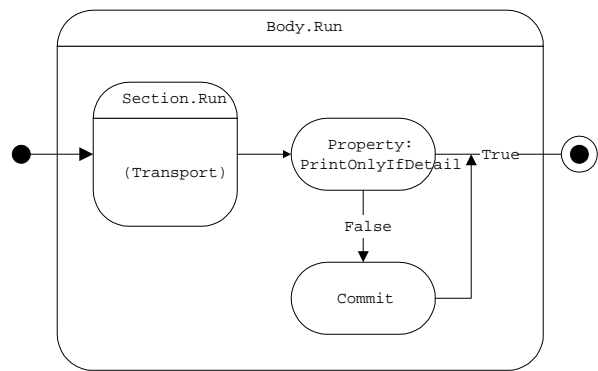
## B.9 GROUPHEADER.RUN



## B.10 GROUPFOOTER.RUN

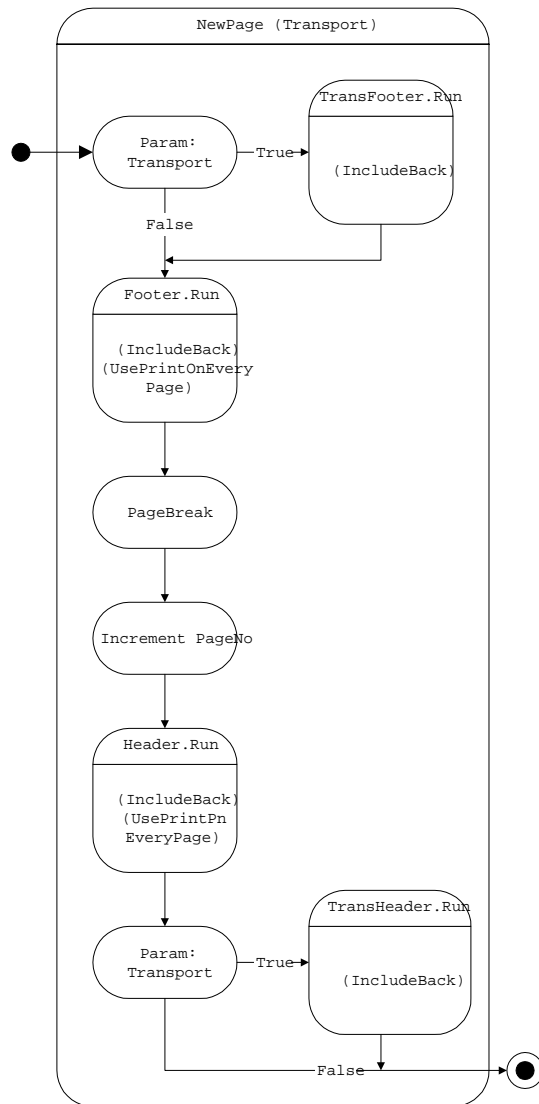


B.11
BODY.RUN

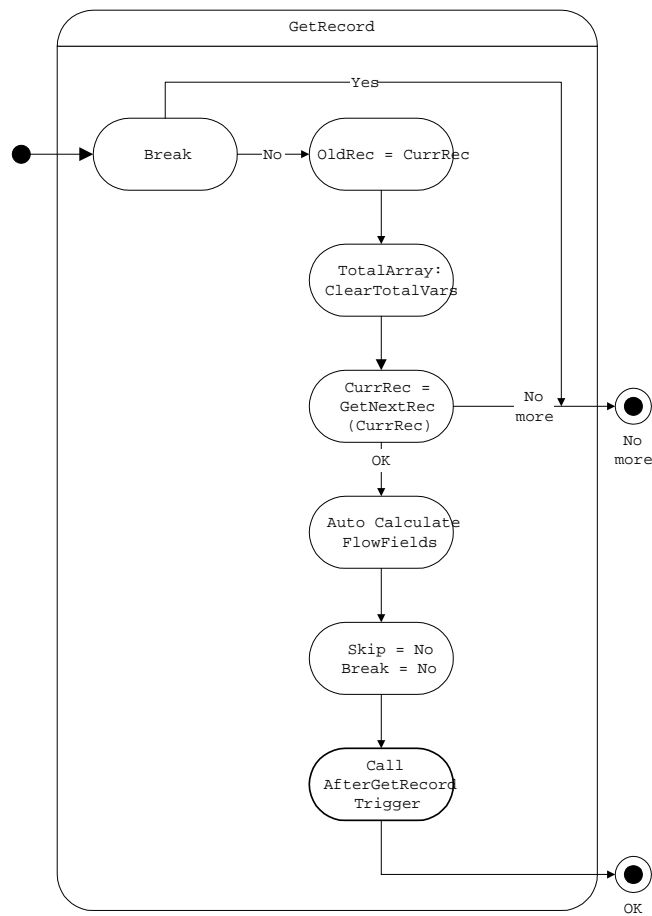




## B.12 NEWPAGE



B.13 GETRECORD



## Appendix C

### Dataport Flow Charts

This appendix illustrates the flow of control for dataports in C/SIDE.

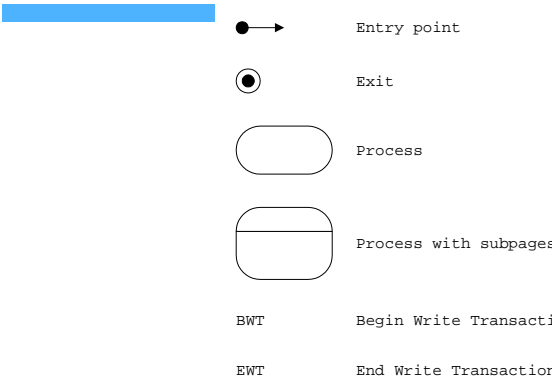
- Dataport Flow charts
- Dataport.Import/Export
- DataItem.Export
- VariableRecord.Export
- FixedRecord.Export
- DataItem.Import
- VariableRecord.Import
- FixedRecord.Import

C.1 DATAPORT FLOW CHARTS

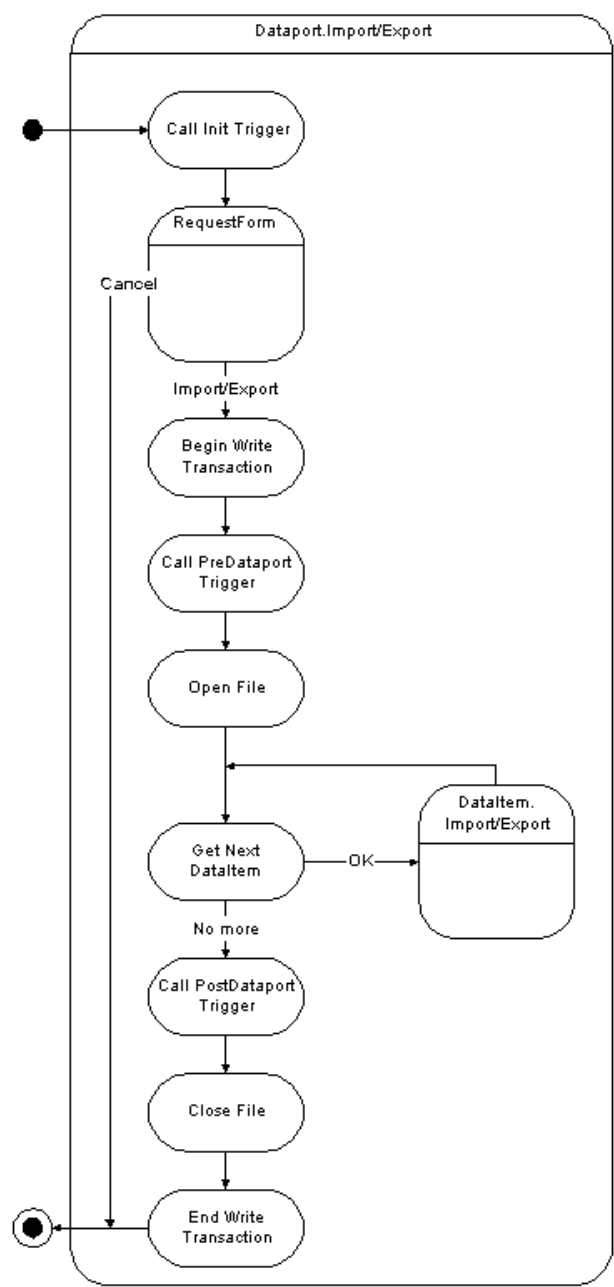
The following sections contain flow charts that show the flow of control for dataports in C/SIDE.

As indicated by the legend below, some processes in one flow chart are "exploded" in the following pages in order to show more details.

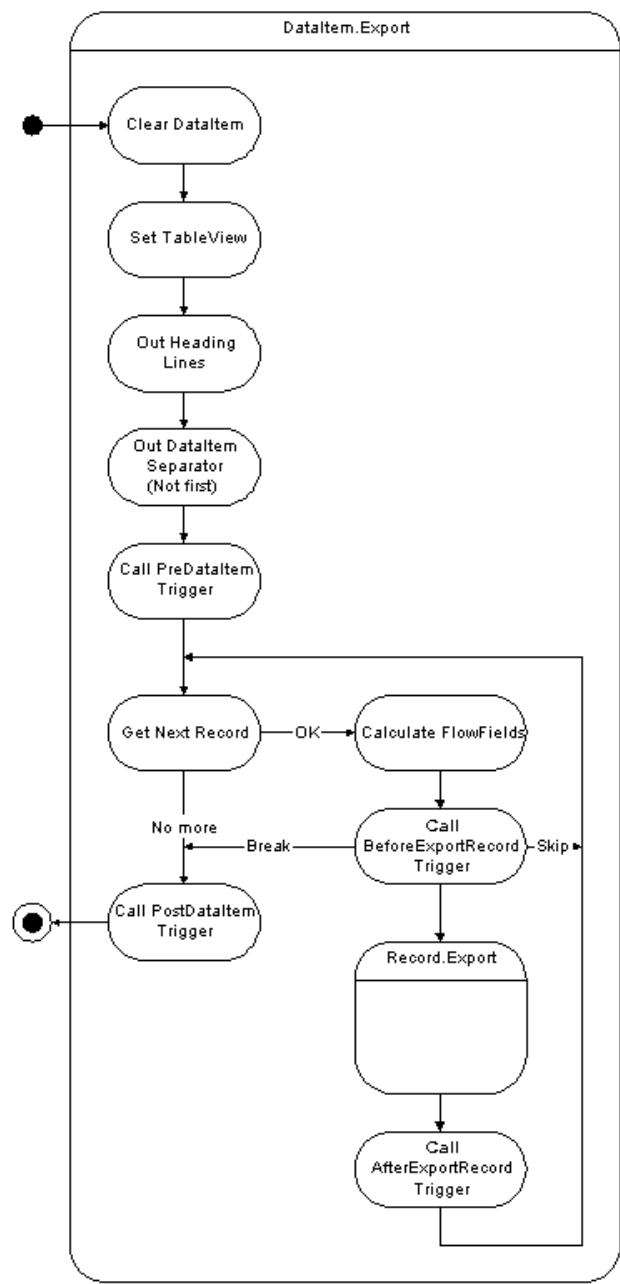
Legend

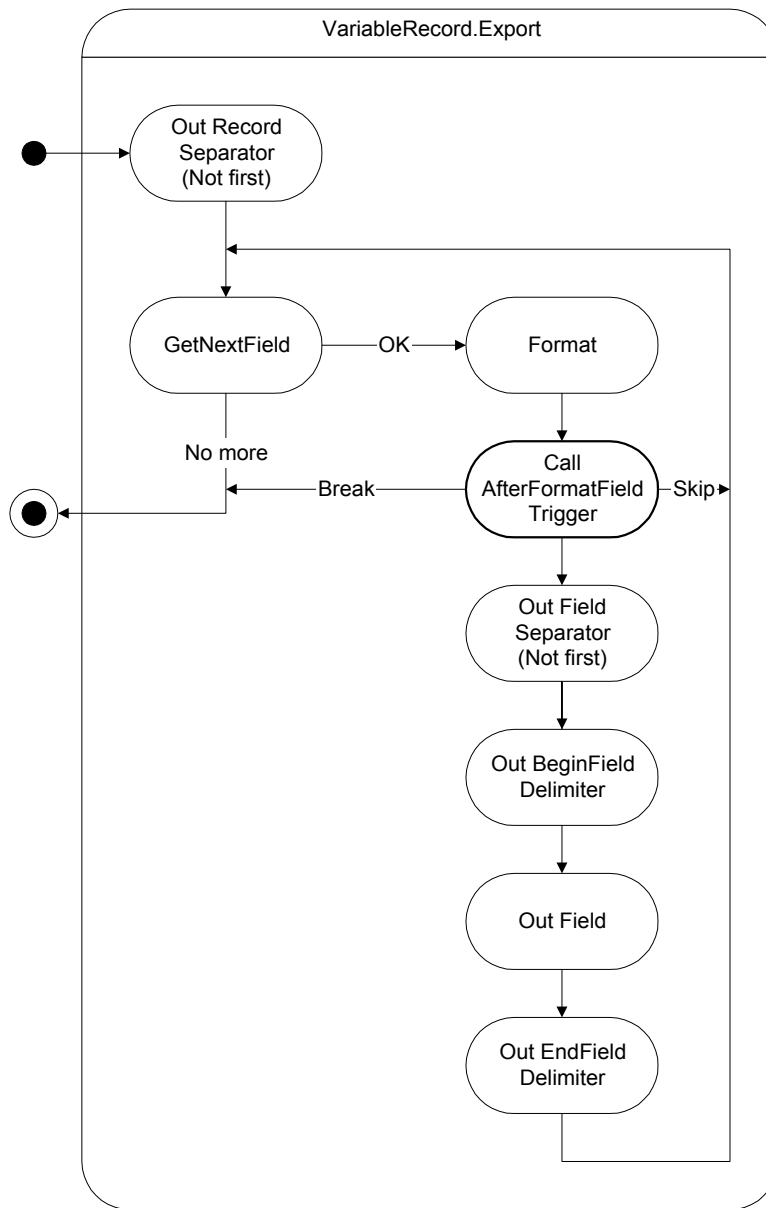


C.2 DATAPORT.IMPORT/EXPORT

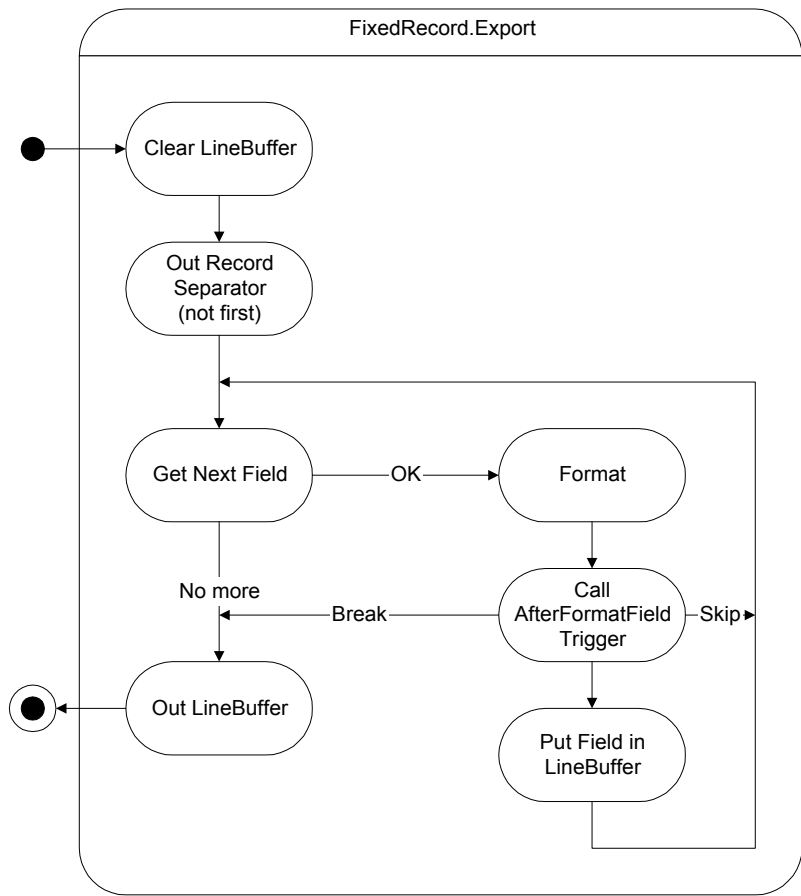


C.3 DATAITEM.EXPORT



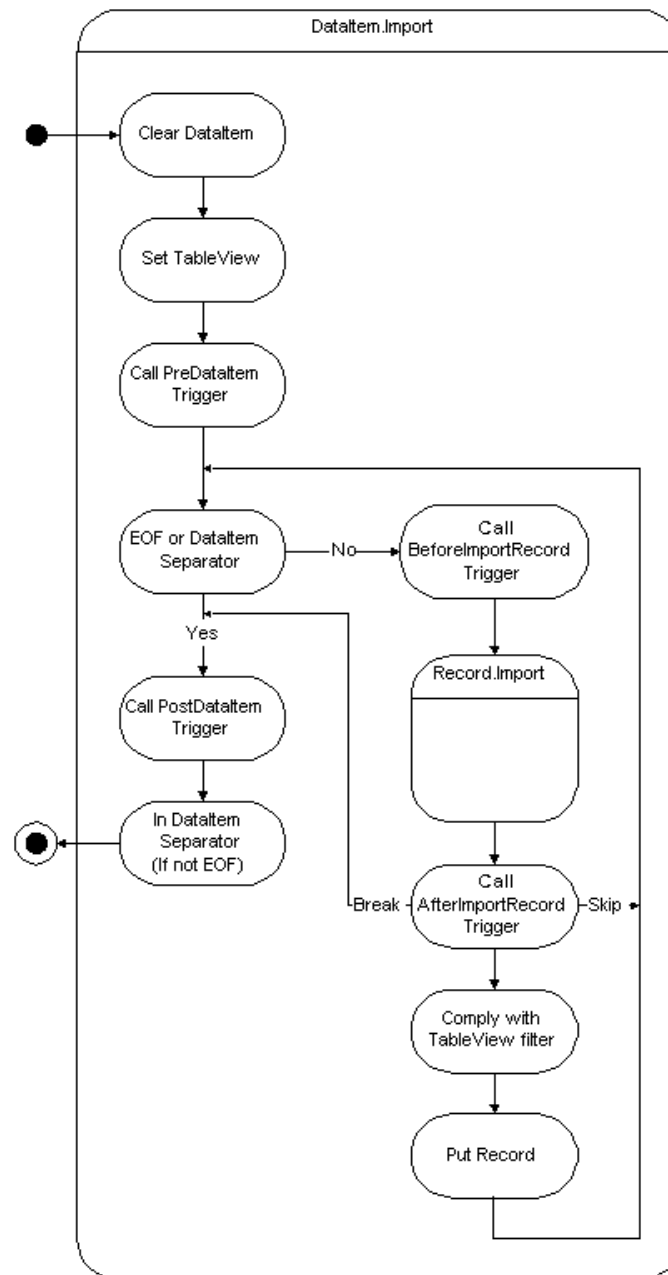
**C.4 VARIABLERECORD.EXPORT**

C.5 FIXEDRECORD.EXPORT

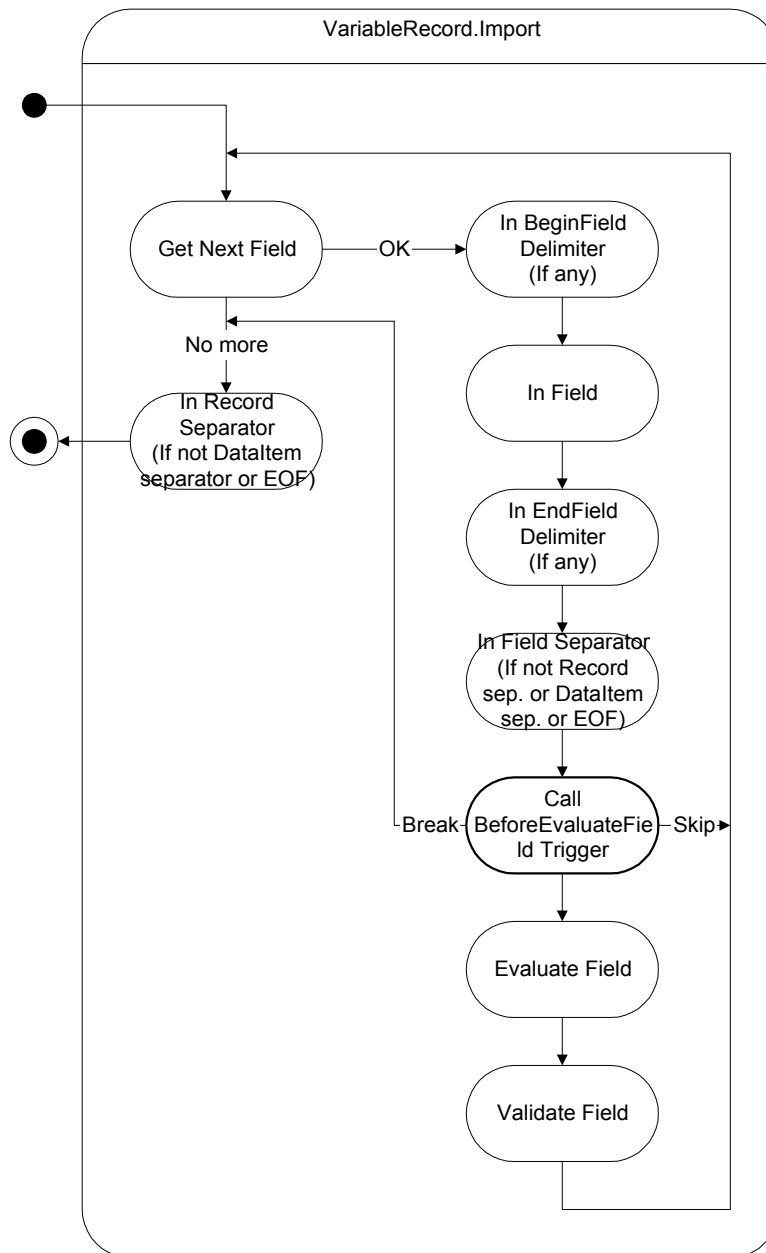




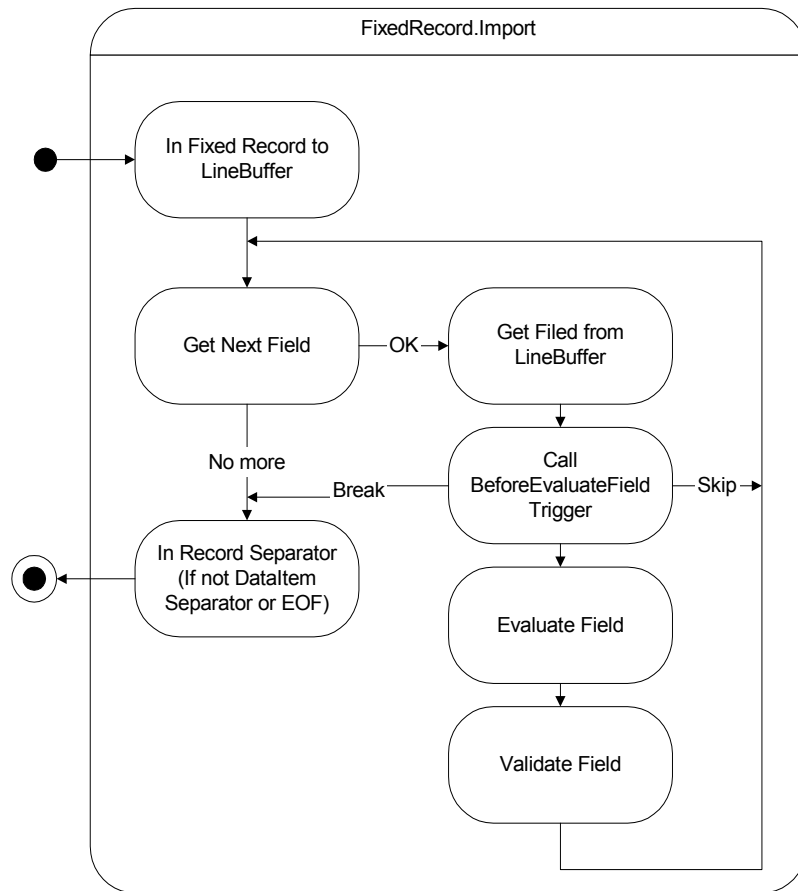
## C.6 DATAITEM.IMPORT



## 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 Navision has been implemented. Although it is not a guide for C/AL development, it can help you understand the way Navision 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 Navision 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 Navision 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 Navision, 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 Navision, 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 Navision, 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 Navision 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 Navision 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 Navision) 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 Navision (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.

## 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 Navision 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 82.

Executing even the simplest query in SQL Server to obtain a record is more expensive than retrieving a single record in Navision 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. Navision generally executes simple queries. There is no SQL optimizer in Navision 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
INSERT	INSERT	1 per table
DELETE	DELETE	per-connection limit
MODIFY	UPDATE	per-connection limit
DELETEALL	DELETE with filter	per-connection limit
MODIFYALL	UPDATE with filter	per-connection limit
	INSERT (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 Navision 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 (i.e. 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 Navision. This implementation involves creating a new table on SQL Server for every SumIndexFields that exists in a Navision 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 Navision.

For more information, see the section SIFT and the SQL Server Option for Navision on page 404.

## 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 Navision 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 `DELETEDALL` 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 code unit 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 actually 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. These criteria are described on page 471.



## INDEX

### A

ActiveX ..... 300  
 application  
   application object ..... 5  
   design ..... 12  
   design (reference to books) ..... 16  
   object ..... 3  
   the term ..... 5  
 Application Area Name (field) ..... 372  
 array ..... 242  
 ascending order ..... 39  
 automation ..... 300, 306  
   using Microsoft Word ..... 306  
   where to put code ..... 307  
 AutoReplace ..... 347  
 AutoSave ..... 347  
 AutoUpdate ..... 347

### B

bigint  
   SQL Server data type ..... 33  
 BigInteger  
   C/AL data type ..... 33  
   field data type ..... 25  
 binary  
   C/AL data type ..... 33, 34  
   SQL Server data type ..... 34  
 binary (field data type) ..... 24  
 bit  
   SQL Server data type ..... 34  
 BLOB ..... 24  
   C/AL data type ..... 33, 34, 240  
   field data type ..... 24  
 boolean  
   C/AL data type ..... 33, 34, 237  
   displaying ..... 132  
   field data type ..... 24  
 bound control ..... 101, 125  
 bound form ..... 101  
 breakpoints ..... 289  
   in the C/AL Editor ..... 293  
   storage in XML file ..... 294  
 Breakpoints virtual table ..... 293  
 bugs ..... 280  
 bulk inserts ..... 471

### C

C/AL  
   bugs ..... 280  
   comments ..... 258  
   constants ..... 243  
   control language ..... 252  
   data types ..... 236  
   debugging ..... 280  
   defined ..... 6

dialogs ..... 274  
 editor ..... 4, 218  
 essential functions ..... 266  
 expressions ..... 235  
 function calls ..... 250  
 Globals ..... 224  
 Locals ..... 225  
 operator hierarchy ..... 250  
 operators ..... 248  
 program logic errors ..... 287  
 repetitive statements ..... 255  
 reusing code ..... 263  
 run-time errors ..... 282  
 statements ..... 235  
 Symbol Menu ..... 226, 382  
 syntax errors ..... 281  
 where to place ..... 262

### C/AL functions

CALCDATE ..... 379  
 CALCFIELDS ..... 271  
 CALCSUMS ..... 271  
 CLEAR ..... 230  
 CONFIRM ..... 276  
 DELETE ..... 270  
 DELETEALL ..... 270  
 ERROR ..... 276  
 FIELDERROR ..... 272  
 FIELDNAME ..... 273  
 FIND ..... 266  
 GET ..... 266  
 GETRANGEMAX ..... 269  
 GETRANGEMIN ..... 268  
 INIT ..... 273  
 INSERT ..... 269  
 LOCKTABLE ..... 271  
 MESSAGE ..... 275  
 MODIFY ..... 269  
 MODIFYALL ..... 270  
 NEXT ..... 267  
 OPEN ..... 275  
 overview ..... 266  
 SETCURRENTKEY ..... 267  
 SETFILTER ..... 268  
 SETRANGE ..... 268  
 STRMENU ..... 276  
 TESTFIELD ..... 273  
 UPDATE ..... 275  
 VALIDATE ..... 273

### cache

commit cache ..... 460  
 DBMS cache ..... 458

- CalcFormula (property) . . . . . 44
- calculation formula . . . . . 44
- CAPTIONAREA . . . . . 444
- CaptionClassTranslate trigger . . . . . 443
- CAPTIONREF . . . . . 444
  - syntax . . . . . 445
- card form . . . . . 102
  - creating . . . . . 103
- CASE . . . . . 254
- char
  - SQL Server data type . . . . . 33
- char (C/AL data type) . . . . . 239
- check box to display booleans . . . . . 132
- CLEARALL . . . . . 230
- Client Monitor . . . . . 82
  - additional parameters for SQL Server 84
- code
  - C/AL data type . . . . . 33, 239
  - field data type . . . . . 24
- Code Coverage Tool . . . . . 297
- code examples
  - DimCaptionClassTranslate . . . . . 448
  - VATCaptionClassTranslate . . . . . 454
- code fields . . . . . 34
- codeunit
  - accessing functions . . . . . 229
  - assigning a . . . . . 230
  - compiling . . . . . 227
  - creating . . . . . 220
  - defined . . . . . 218
  - limitations . . . . . 231
  - running . . . . . 219
  - saving . . . . . 227
  - single instance . . . . . 230
  - specifications . . . . . 478
  - temporary tables . . . . . 218
- codeunit (C/AL data type) . . . . . 241
- Color tool . . . . . 124
- column . . . . . 20
- COM . . . . . 300
  - and C/SIDE . . . . . 302
  - automation . . . . . 306
  - CREATE . . . . . 312
  - custom controls . . . . . 326
  - default members . . . . . 313
  - enumerations . . . . . 304
  - exceptions . . . . . 332
  - Microsoft Excel . . . . . 315
  - OCX . . . . . 326
  - terminology . . . . . 301
  - USERDEF . . . . . 304
  - using Microsoft Word . . . . . 306
- commit
  - in C . . . . . 434
  - in C/AL . . . . . 434
- commit cache . . . . . 460
- COMMIT() . . . . . 435
- company . . . . . 41
- Company system table . . . . . 77
- complex data types . . . . . 240
- compound statement . . . . . 252
- concurrency . . . . . 430
- conditional statements . . . . . 252
- configuration parameters
  - SQL Server Option . . . . . 467
- consistency . . . . . 430
- constant . . . . . 243
  - text . . . . . 222, 243
- container control . . . . . 100, 119
  - frame . . . . . 133
- control
  - adding . . . . . 121
  - aligning . . . . . 111
  - bitmaps . . . . . 136
  - bound . . . . . 101, 125
  - branch . . . . . 100
  - changing caption . . . . . 125
  - changing name . . . . . 125
  - check box . . . . . 132
  - Color tool . . . . . 124
  - command button . . . . . 120, 132
  - container . . . . . 100, 119, 133
  - container control selection . . . . . 110
  - control branch selection . . . . . 110
  - data . . . . . 119
  - data types . . . . . 119
  - defined . . . . . 6
  - display properties . . . . . 126
  - displaying numbers . . . . . 126
  - Font tool . . . . . 124
  - formatting data . . . . . 126
  - frame . . . . . 119, 133
  - image . . . . . 119
  - in reports . . . . . 161
  - indicator . . . . . 137
  - input . . . . . 127
  - label . . . . . 119
  - menu button . . . . . 120, 150
  - menu item . . . . . 120
  - moving . . . . . 110
  - multiple selection . . . . . 108
  - option button group . . . . . 131
  - option drop-down . . . . . 130
  - picture box . . . . . 134, 136
  - properties of, in forms . . . . . 125
  - properties of, in reports . . . . . 177
  - shape . . . . . 119, 134, 135
  - sizing . . . . . 111
  - static . . . . . 119
  - subform . . . . . 120
  - tab control . . . . . 119, 138
  - table box . . . . . 120, 138
  - text box . . . . . 121, 123, 126, 129

- toolbox ..... 121
- tools ..... 124
- ToolTip ..... 128
- types of ..... 119
- unbound ..... 101, 125
- control types ..... 119
- CREATE
  - automation variable ..... 306, 312
- custom control ..... 300
  - developing ..... 333
  - using in C/SIDE ..... 326
- D**
- data ..... 23
- data container ..... 120
- data controls ..... 119
- data integrity ..... 428, 432
- data item ..... 160, 161
  - data model ..... 178, 186, 192
  - dataport ..... 338
  - defined ..... 7
  - properties ..... 176
- data model ..... 13
  - dataport ..... 339
- data type
  - BigInteger (field) ..... 25
  - binary (field) ..... 24
  - BLOB (C/AL) ..... 240
  - BLOB (field) ..... 24
  - boolean (C/AL) ..... 237
  - boolean (field) ..... 24
  - char (C/AL) ..... 239
  - code (C/AL) ..... 239
  - code (field) ..... 24
  - codeunit (C/AL) ..... 241
  - complex (C/AL) ..... 240
  - controls and data types ..... 119
  - date (C/AL) ..... 238
  - date (field) ..... 24
  - DateFormula ..... 379
  - dateformula (field) ..... 25
  - DateTime (field) ..... 25
  - datetime (SQL) ..... 24
  - decimal (C/AL) ..... 238
  - decimal (field) ..... 23
  - decimal (SQL) ..... 23
  - descriptive (C/AL) ..... 240
  - dialog (C/AL) ..... 241
  - Duration (field) ..... 25
  - field data type ..... 25
  - file (C/AL) ..... 241
  - form (C/AL) ..... 240
  - fundamental ..... 237
  - GUID (field) ..... 25
  - image (SQL) ..... 24
  - integer (C/AL) ..... 237
  - integer (field) ..... 23
  - integer (SQL) ..... 23
  - mixing ..... 390
  - option (C/AL) ..... 237
  - option (field) ..... 23
  - record (C/AL) ..... 240
  - report (C/AL) ..... 241
  - RowID (field) ..... 25
  - table fields ..... 23
  - text (C/AL) ..... 239
  - text (field) ..... 23
  - time (C/AL) ..... 238
  - time (field) ..... 24
  - tinyint (SQL) ..... 24
  - varbinary (SQL) ..... 24
  - varchar (SQL) ..... 23, 24
- data version
  - defined ..... 429
  - historical ..... 431
  - storage of ..... 431
- database
  - defined ..... 8
  - design ..... 12
  - design (reference to books) ..... 16
  - logical ..... 8
  - physical ..... 8
  - the term ..... 5
- Database File virtual table ..... 88
- Database Key Groups system table ... 78
- dataport
  - AutoReplace ..... 347
  - AutoSave ..... 347
  - AutoUpdate ..... 347
  - combining export and import ..... 362
  - data item ..... 338
  - data item properties ..... 346
  - data model ..... 339
  - description ..... 338
  - designing ..... 344
  - dynamic dataport example ..... 362
  - export examples ..... 350
  - export, fixed format ..... 350
  - export, variable format ..... 355
  - external file ..... 339
  - field ..... 338
  - field properties ..... 348
  - FileFormat property ..... 345
  - fixed format export ..... 350
  - fixed format import ..... 357
  - import examples ..... 357
  - import, fixed format ..... 357
  - import, variable format ..... 360
  - logical design ..... 339
  - property ..... 339, 344
  - request form ..... 338
  - running ..... 340, 342
  - trigger ..... 339, 349
  - variable format export ..... 355
  - variable format import ..... 360

- Dataport Designer ..... 4
- date
  - C/AL data type ..... 33, 34, 238
  - field data type ..... 24, 36
- Date virtual table ..... 80
- DateFormula
  - C/AL data type ..... 33
- dateformula
  - field data type ..... 25
- DateTime
  - C/AL data type ..... 33
  - field data type ..... 25
- datetime
  - SQL Server data type ..... 33
- DBL\_BWT ..... 434
- DBL\_EWT ..... 434
- DBMS ..... 428
  - cache ..... 458
  - specifications ..... 476
- deadlock detection ..... 433
- debugger
  - activating the ..... 289
  - breakpoints stored in XML file ..... 294
  - code coverage ..... 297
  - interface ..... 290
  - menus ..... 290
  - overview of shortcut keys ..... 295
  - running on Navision Application Server  
290
  - setting breakpoints in the C/AL Editor 293
  - storage of information in the fin.zup file  
295
  - the Breakpoints virtual table ..... 293
  - toolbar ..... 291
  - windows ..... 292
- decimal
  - C/AL data type ..... 33, 34, 238
  - field data type ..... 23
  - SQL Server data type ..... 33
- default members (COM) ..... 313
- delayed write back ..... 460
- deleting language ..... 372
- descending order ..... 39
- descriptive data types ..... 240
- Designer
  - Dataport ..... 4
  - Form ..... 4
  - Object ..... 3
  - Report ..... 4
  - Table ..... 4
- dialog ..... 274
  - C/AL data type ..... 241
- DimCaptionClassTranslate ..... 448
- DIMCAPTIONREF ..... 446
- DIMCAPTIONTYPE ..... 446
- dimension area ..... 446
- document ..... 208
- Documentation section ..... 219
- drill-down
  - form ..... 147
- Drive virtual table ..... 82
- Duration
  - C/AL data type ..... 33
  - field data type ..... 25
- E**
- editor
  - purpose ..... 4
  - using ..... 218
- Entity-Relationship (ER) model ..... 13
- enumerations ..... 304
- Events
  - receiving ..... 322
- EXIT ..... 257
- expression
  - basic elements of ..... 243
  - defined ..... 235
  - evaluation ..... 236, 390
- external tools
  - accessing table data with ..... 36
- F**
- field
  - dataport ..... 338
  - defined ..... 6
  - illustration ..... 20
  - property ..... 52
  - trigger ..... 58
- Field Menu
  - in reports ..... 167
- Field Virtual Table ..... 90
- Field virtual table ..... 90
- file (C/AL data type) ..... 241
- File virtual table ..... 81
- FileFormat property ..... 345
- filters
  - and number sorting ..... 426
- float
  - SQL Server data type ..... 34
- FlowField
  - calculation formula ..... 44
  - table filter ..... 46
- FlowFilter field ..... 42
- Font tool ..... 124
- FOR TO/DOWNT0 ..... 255
- Form ..... 373
- form
  - bound ..... 101
  - card form ..... 102, 103
  - closing ..... 113
  - compiling ..... 113



- creating ..... 102, 103, 105, 106
  - description ..... 100
  - design ..... 100, 141
  - drill-down ..... 147
  - Form Wizard ..... 102
  - lookup ..... 145
  - main form ..... 140
  - running ..... 114, 149
  - saving ..... 113
  - specifications ..... 478
  - subform ..... 140
  - tabular form ..... 102
  - test-compiling ..... 113
  - unbound ..... 101
  - form (C/AL data type) ..... 240
  - form design
    - Color tool ..... 124
    - Font tool ..... 124
    - toolbox ..... 121
    - tools ..... 124
  - Form Designer ..... 4, 100
  - Form Wizard ..... 102, 103
    - creating a card form ..... 103
    - creating a tabular form ..... 105
  - function
    - accessing in codeunit ..... 229
    - C/AL ..... 218
    - creating ..... 221
  - fundamental data types ..... 237
- G**
- global variable ..... 218
  - Globals ..... 224
  - graph
    - creating with Microsoft Excel ..... 315
  - groups in reports ..... 192
  - GUID
    - C/AL data type ..... 33
    - field data type ..... 25
- I**
- ID number ..... 3
  - IDE ..... 2
  - IF THEN ELSE ..... 253
  - image
    - SQL Server data type ..... 33
  - index hinting
    - SQL Server Option ..... 467
  - integer
    - C/AL data type ..... 33, 34, 237
    - field data type ..... 23
    - SQL Server data type ..... 33, 34
  - Integer virtual table ..... 81
  - integrity ..... 428
- K**
- key
    - defined ..... 6
    - discussed ..... 26
- groups ..... 78
  - in ER model ..... 14
  - list ..... 26, 29, 31
  - performance ..... 30
  - primary ..... 26
  - property ..... 55
  - secondary ..... 27
  - SumIndexFields ..... 402
- L**
- Language ..... 372
  - language
    - adding ..... 371
    - deleting ..... 372
    - multiple document ..... 374
  - language ID ..... 372
  - language layer ..... 371
  - lazy write ..... 460
  - limitations
    - event triggers ..... 324
  - linked objects
    - description ..... 66
    - requirements ..... 67
  - Locals ..... 225
  - lock granularity
    - SQL Server Option ..... 469
  - locking
    - a comparison of Navision Database Server and SQL Server ..... 436
    - in Navision Database Server ..... 437
    - in SQL Server ..... 437
  - locks ..... 432
  - log file ..... 429
  - logical database ..... 8
  - lookup ..... 143, 145
    - form ..... 145
    - table relation ..... 144
  - looping (C/AL) ..... 255
- M**
- main form ..... 140
    - design ..... 141
  - Member Of system table ..... 75
  - menu
    - design ..... 150, 151
    - menu button ..... 150
    - menu item ..... 150
    - menu line ..... 151
    - shortcut key ..... 152
  - Microsoft Enterprise Manager ..... 36
  - Microsoft Excel ..... 315
  - Microsoft Word ..... 306
  - money
    - SQL Server data type ..... 34
  - Monitor virtual table ..... 82
  - multilanguage ..... 369
    - C/ODBC ..... 374
    - date formulas ..... 379
    - multiple document languages ..... 374

- SQL views ..... 37
- text constants ..... 370, 376
- Multilanguage Editor ..... 371
- N**
- Navision Database Server
  - locking in ..... 437
  - number sorting ..... 423
  - SIFT ..... 402
- Navision debugger ..... 289
- nchar
  - SQL Server data type ..... 34
- nonprinting report ..... 212
- ntext
  - SQL Server data type ..... 34
- Number ..... 240
- number sorting
  - a definition of ..... 424
  - and filters ..... 426
  - differences between Navision Database Server and SQL Server ..... 424
  - principles ..... 425
  - recommendations ..... 424
- numbering principles ..... 425
- numeric
  - SQL Server data type ..... 34
- nvarchar
  - SQL Server data type ..... 34
- O**
- Object Designer ..... 3
- OCX ..... 300
  - developing ..... 333
  - registering ..... 326
  - using in C/SIDE ..... 326
- OLE ..... 300
- OnRun section ..... 219
- operators
  - arithmetic (type conversion) ..... 393
  - hierarchy ..... 250
  - logical (type conversion) ..... 393
  - relational (type conversion) ..... 392
  - using in C/AL ..... 248
- option
  - C/AL data type ..... 33, 34, 237
  - field data type ..... 23
- option (C/AL data type) ..... 33
- order
  - ascending ..... 39
  - descending ..... 39
- P**
- performance
  - C/AL functions and SQL Server 464, 466
  - command buffer ..... 462
  - commit cache ..... 460
  - DBMS cache ..... 458
  - keys ..... 30
  - keys and queries ..... 464
  - measuring ..... 82
- Permission system table ..... 76
- physical database ..... 8
- program logic errors ..... 287
- property
  - CalcFormula ..... 44
  - control ..... 101, 116, 125, 126, 127
  - control, general properties ..... 118
  - dataport ..... 339, 344
  - defined ..... 6
  - fields ..... 52
  - form ..... 101, 116
  - inheriting ..... 116
  - key ..... 55
  - list of, in forms ..... 117
  - list of, in reports ..... 161
  - list of, in tables ..... 50
  - parameterized ..... 331
  - Property Sheet ..... 101
  - report ..... 174, 175
  - TableRelation ..... 60
- Property Sheet ..... 101
- R**
- read consistency ..... 430
- real
  - SQL Server data type ..... 34
- record
  - C/AL data type ..... 240
  - defined ..... 20
- record level security
  - supporting on SQL Server option .. 456
- registering an OCX ..... 326
- relationship ..... 60
- REPEAT UNTIL ..... 256
- report
  - closing ..... 170
  - compiling ..... 170
  - control ..... 161
  - control properties ..... 177
  - controlling output ..... 205
  - data item ..... 160, 161, 178, 186, 192
  - data item properties ..... 176
  - data item triggers ..... 198
  - data model ..... 178, 186, 192
  - definition ..... 160
  - designing sections ..... 181, 188, 195
  - documents ..... 208
  - execution ..... 164
  - Field Menu ..... 167
  - flow chart ..... 164
  - grouping ..... 192
  - nonprinting ..... 212
  - properties ..... 161, 174
  - property, description of ..... 175
  - report description ..... 160
  - Report Designer ..... 167
  - report triggers ..... 198

- request form ..... 161
  - Request Options Form Designer .. 167, 168
  - running ..... 171
  - saving ..... 170
  - section ..... 161, 162, 181
  - Section Designer ..... 167
  - section properties ..... 176
  - section triggers ..... 198
  - specifications ..... 478
  - totaling ..... 192
  - totals ..... 193
  - triggers ..... 161, 198
  - virtual tables in reports ..... 200
  - report (C/AL data type) ..... 241
  - report description ..... 160
  - Report Designer ..... 4, 167
  - request form ..... 161
    - dataport ..... 338
  - Request Options Form Designer 167, 168
  - RequestForm ..... 161
    - defined ..... 7
  - row ..... 20
  - RowID
  - C/AL data type ..... 33
  - field data type ..... 25
  - run-time errors ..... 265, 282
- S**
- section ..... 161, 162
    - defined ..... 7
    - designing ..... 181, 188, 195
    - properties ..... 176
    - triggers ..... 198
  - Section Designer ..... 167
  - Session virtual table ..... 85
  - shortcut key ..... 152
  - shortcut keys
    - in the C/AL Editor ..... 221
    - in the debugger ..... 295
  - SID - Account ID virtual table ..... 95
  - SIFT
    - Navision Database Server ..... 402
    - SQL Server Option ..... 404–422
    - tables on SQL Server ..... 32
  - smalldatetime
    - SQL Server data type ..... 34
  - smallint
    - SQL Server data type ..... 34
  - smallmoney
    - SQL Server data type ..... 34
  - specifications
    - codeunit ..... 478
    - DBMS ..... 476
    - forms ..... 478
    - reports ..... 478
    - tables ..... 477
- SQL Server
- additional parameters in the Client
    - Monitor ..... 84
    - how code fields work in ..... 24
    - locking in ..... 437
    - SIFT tables ..... 32
- SQL Server Option
- configuration parameters ..... 467
  - index hinting ..... 467
  - linked objects ..... 66
  - lock granularity ..... 469
  - maintaining table relationships ..... 63
  - SIFT ..... 404–422
  - SIFT buckets ..... 406
  - SIFT table ..... 405
  - SIFT table, extended key ..... 413
  - SIFT table, indexes ..... 413
  - SIFT table, layout ..... 413
  - SIFT table, optimizing ..... 422
  - SIFT Trigger ..... 405
  - SIFT, costs and benefits ..... 421
  - SIFT, Date fields ..... 409
  - SIFT, DateTime fields ..... 412
  - SIFT, deleting records ..... 418
  - SIFT, updating the base table ..... 416
  - sorting numerical values in code fields . 423
  - supporting record level security ... 456
- statement
- compound ..... 252
  - conditional ..... 252
  - defined ..... 235
- String ..... 240
- subform ..... 120, 140
- design ..... 141
- SumIndexField
- and FlowFields ..... 42
  - and SQL Server ..... 438
  - defined ..... 402
- Symbol Menu ..... 226, 382
- syntax errors ..... 281
- system table ..... 74
- Company ..... 77
  - Database Key Groups ..... 78
  - Member Of ..... 75
  - Permission ..... 76
  - User ..... 75
  - User Role ..... 76
  - Windows Access Control ..... 77
  - Windows Login ..... 77
- system-defined variable ..... 264
- T**
- tab control ..... 119
- table
- accessing data with external tools .. 36
  - adding records ..... 39
  - defined ..... 20

- description ..... 20
  - modifying the design of ..... 65
  - property ..... 50
  - relationship ..... 60
  - saving ..... 38
  - specifications ..... 477
  - system ..... 74
  - temporary ..... 72
  - trigger ..... 58
  - validating relationship ..... 145
  - viewing data in ..... 38
  - virtual ..... 79
  - Table Designer ..... 4, 22
  - Table Information virtual table ..... 89, 90
  - table property
    - LinkedInTransaction ..... 66
    - LinkedObject ..... 66
    - list ..... 50
    - viewing and modifying ..... 50
  - table relation ..... 144
    - and assist edit ..... 62
    - example ..... 62
  - TableFilter
    - C/AL data type ..... 33
    - data type (field) ..... 25
  - TableRelation property ..... 60
  - tabular form ..... 102
    - creating ..... 105
  - template ..... 7
  - temporary table ..... 72
  - text
    - C/AL data type ..... 33, 34, 239
    - field data type ..... 23
    - SQL Server data type ..... 34
  - text box
    - adding ..... 121, 123
    - adding a label ..... 126
    - calculation ..... 129
    - multiple lines ..... 129
  - text constant ..... 222, 243
  - Text Constants ..... 370
  - time
    - C/AL data type ..... 33, 238
    - field data type ..... 24, 36
  - tinyint
    - SQL Server data type ..... 33, 34
  - toolbox ..... 121
  - ToolTip ..... 128
  - totals
    - and sections ..... 193
  - totals in reports ..... 192
  - transaction ..... 428
  - trigger
    - control ..... 101, 155
    - data item ..... 198
    - dataport ..... 339, 349
    - defined ..... 6
    - field ..... 58
    - form ..... 101, 154
    - overview of control triggers ..... 155
    - overview of form triggers ..... 154
    - overview of report triggers ..... 198
    - report ..... 161, 198
    - table ..... 58
  - triggers
    - CaptionClassTranslate ..... 443
  - type conversion ..... 390
- U**
- unbound control ..... 101
    - changing to bound ..... 125
  - unbound form ..... 101
  - uniqueidentifier
    - SQL Server data type ..... 33, 34
  - user interface ..... 2
  - User Role system table ..... 76
  - User SID virtual table ..... 95
  - User system table ..... 75
  - USERDEF ..... 304
  - user-defined variable ..... 244
- V**
- varbinary
    - SQL Server data type ..... 33
  - varchar
    - SQL Server data type ..... 33
    - SQLdata type ..... 24
  - variable
    - arrays ..... 242
    - assignment ..... 246
    - creating ..... 221
    - CurrForm ..... 264
    - CurrReport ..... 264
    - global ..... 218
    - initialization ..... 246
    - naming conventions ..... 244
    - Rec ..... 264
    - system-defined ..... 264
    - user-defined ..... 244
    - xRec ..... 264
  - VATCAPTIONREF ..... 447
  - VATCAPTIONTYPE ..... 447
  - virtual table ..... 79
    - Breakpoints ..... 293
    - Database File ..... 88
    - Date ..... 80
    - Drive ..... 82
    - Field ..... 90
    - File ..... 81
    - Integer ..... 81
    - Monitor ..... 82
    - Session ..... 85
    - SID - Account ID ..... 95
    - Table Information ..... 89, 90
    - User SID ..... 95
    - Windows Group Member ..... 94
    - Windows Object ..... 93

virtual tables . . . . . 200

## **W**

WHILE DO . . . . . 256

Windows Access Control system table . 77

Windows Group Member virtual table . 94

Windows Login system table . . . . . 77

Windows Object virtual table . . . . . 93

WITH . . . . . 257

write locks . . . . . 432

write transaction . . . . . 428

