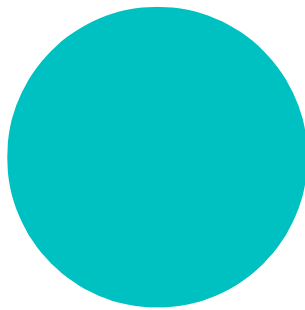


NAVISON® Attain SCHULUNG



Navision Attain Programming II

Die PC&C Vertriebs GmbH behält sich das Recht vor, Änderungen in dieser Publikation jederzeit fristlos vorzunehmen. Etwaige Änderungen sind nicht als Verpflichtung seitens PC&C Vertriebs GmbH zu betrachten.

Die PC&C Vertriebs GmbH übernimmt keine Haftung für etwaige Fehler und Mängel in dieser Publikation.

Die in dieser Publikation beschriebene Software wird unter Lizenz geliefert und darf ausschließlich laut den beiliegenden Lizenzbestimmungen verwendet und kopiert werden.

Fotografische, elektronische, mechanische oder sonstige Vervielfältigung und Weitergabe dieser Publikation, auch auszugsweise, sind nach geltendem deutschen Urheberrecht nicht zulässig.

Copyright: © 2001

Navision – PC&C Personal Computing and Communications Vertriebs GmbH -
Notkestraße 9-11; 22607 Hamburg

Tel: (0 40) 8 99 67 70

Autor: NAC

Navision Attain® ist ein eingetragenes Warenzeichen der Navision Software a/s.

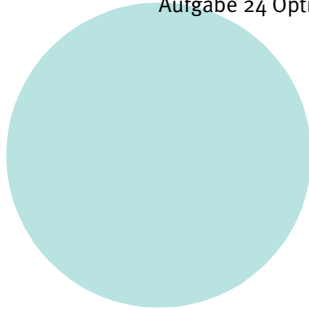
INHALTSVERZEICHNIS

Kapitel 1	Einleitung	7
	Über die Schulung	8
	Typographische Hervorhebungen	10
Kapitel 2	Grundlagen der Navision Attain	
	Entwicklungsumgebung C/SIDE	11
	Benutzen des Object Designers	12
	Table, Form, Report, Dataport, Codeunits	12
	Ein- bzw. Auslesen von Objekten	14
	Benutzen der Hilfe	17
	Hilfeaufbau.....	17
	Hilfesyntax	18
	Begriffserklärungen	20
	Trigger.....	20
	Property	21
	Feld	21
	Key	21
	Das C/AL Symbol Menu	23
Kapitel 3	Grundlagen allgemeiner Programmierung	25
	Variablen	26
	Anlegen und Definieren von Variablen.....	26
	Die verschiedenen Variablenarten.....	27
	Arrays	33
	Textkonstante.....	38
	Anlegen und Definieren von Textkonstanten	38
	Bedingungen	40
	IF-THEN Bedingung.....	40
	IF-THEN-ELSE-Bedingung	43
	BEGIN..END	44
	Der CASE-Befehl	46
	WITH-Befehl	47
	Schleifen.....	49
	FOR-Schleife	49
	REPEAT-UNTIL-Schleife.....	50
	WHILE-DO-Schleife	52
	Funktionen.....	55
	Funktionen ohne Parameter	55
	Funktionen mit Parameter	57

Funktionen mit Rückgabewert	58
Funktionen mit VAR-Parameter	60
Unterschied zwischen globalen und lokalen Variablen...	61
Kapitel 4 Grundlagen der Programmierung	
in Navision Attain.....	63
Variablentypen erweitert	64
Record	65
Form	65
Report	65
Dataport	66
Benutzung von Codeunits	66
Befehle der Kategorie Dialog	67
ERROR/MESSAGE.....	67
Eingabefenster	69
Befehle der Kategorie Record	71
GET	71
FIND	72
Unterschied zwischen GET und FIND	74
SETCURRENTKEY	75
SETRANGE	76
SETFILTER	77
Unterschied zwischen SETRANGE und SETFILTER	78
NEXT	79
RESET	80
MODIFY/MODIFYALL.....	82
DELETE/DELETEALL	84
INIT	86
INSERT	87
COMMIT	89
TESTFIELD/FIELDERROR	91
CALCFIELDS/CALCSUMS	93
Unterschied zwischen CALCFIELDS und CALCSUMS.....	95
Befehle der Kategorie System.....	97
String	97
Numeric.....	99
Date.....	100
Variable	103
Rec/xRec/FIELDNO	106
Rec	106
xRec.....	106
CurrFieldNo	106
Optionswertbestimmung mit Scope "::-"	107
Kapitel 5 Programmierung nach Style Guide.....	109
Variablenamen/Funktionsnamen.....	110

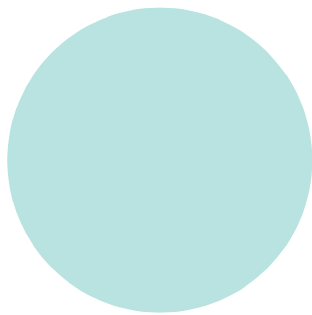
Einrückung IF-THEN-ELSE	111
Einrückung CASE	112
Einrückung für Schleifen	113
Dokumentation in Objekten	114
Mit //	114
Mit { }	114
Im Trigger "Dokumentation"	115
Kapitel 6 Lösungen.....	117
Aufgabe 1 Variablen	118
Aufgabe 1.1 Variablen erstellen	118
Aufgabe 1.2 Werte eingeben.....	118
Aufgabe 2 Array	119
Aufgabe 2.1 Anlegen und Anzeigen von Arrays	119
Aufgabe 2.2 Werte zuweisen	119
Aufgabe 4 Bedingungen	120
Aufgabe 4.1 Einfache IF-Bedingung	120
Aufgabe 4.2 Mehrfache IF-Bedingung.....	120
Aufgabe 4.3 NOT-Bedingung	120
Aufgabe 4.4 IF-THEN-ELSE	121
Aufgabe 4.5 BEGIN..END	121
Aufgabe 4.6 Verschachtelte IF-Bedingung.....	121
Aufgabe 4.7 CASE.....	122
Aufgabe 5 Schleifen	123
Aufgabe 5.1 FOR-Schleife.....	123
Aufgabe 5.2 REPEAT-UNTIL-Schleife	123
Aufgabe 5.3 WHILE-DO-Schleife	124
Aufgabe 5.4 Zusammenfassende Aufgabe.....	124
Aufgabe 6 Funktionen.....	126
Aufgabe 6.1 Ohne Parameter	126
Aufgabe 6.2 Mit Parameter.....	126
Aufgabe 6.3 Mit Rückgabewert.....	127
Aufgabe 6.4 Funktion mit VAR-Parameter.....	128
Aufgabe 6.5 Wiederholung Funktionsaufruf	129
Aufgabe 7 Unterschied lokale - globale Variable.....	130
Aufgabe 8 ERROR-MESSAGE	131
Aufgabe 9 Record Variable, GET und Dialogfenster	132
Aufgabe 10 SETCURRENTKEY, SETRANGE, FIND, NEXT	133
Aufgabe 10.1 FIND.....	133
Aufgabe 10.2 SETRANGE und FIND	133
Aufgabe 10.3 SETCURRENTKEY, SETRANGE, FIND.....	133
Aufgabe 10.4 FIND mit Abfrage	134
Aufgabe 10.5 FIND alle in Filter	134

Aufgabe 11 SETFILTER	135
Aufgabe 12 Zusammenfassung Filer.....	138
Aufgabe 13 MODIFY	139
Aufgabe 14 MODIFYALL.....	140
Aufgabe 15 DELETE	141
Aufgabe 16 DELETEALL.....	142
Aufgabe 17 INIT.....	143
Aufgabe 18 INSERT	144
Aufgabe 19 COMMIT	145
Aufgabe 20 TESTFIELD, FIELDERROR	146
Aufgabe 21 CALCFIELDS, CALCSUMS.....	147
Aufgabe 22 Befehle Kategorie STRING.....	148
Aufgabe 23 Befehle Kategorie Datum/STRING	149
Aufgabe 24 Optionswerte	150



AUFGABENVERZEICHNIS

1.1 Variablen erstellen.....	31
1.2 Werte eingeben	32
1.3 Werte zuweisen	32
2.1 Anlegen und Anzeigen von Arrays	35
2.2 Werte zuweisen	35
3 Datentypen in einer Tabelle (Beispiel Fuhrpark)	36
4.1 Einfache IF-Bedingung.....	42
4.2 Mehrfache IF-Bedingung.....	42
4.3 NOT-Bedingung	42
4.4 IF-THEN-ELSE	43
4.5 Begin..End	45
4.6 Verschachtelte IF-Bedingung	45
4.7 CASE	47
5.1 FOR-Schleife	50
5.2. REPEAT-UNTIL-Schleife	51
5.3 WHILE-DO-Schleife	53
5.4 Zusammenfassende Aufgabe.....	54
6.1 Ohne Parameter	56
6.2 Mit Parameter	58
6.3 Mit Rückgabewert.....	59
6.4 Mit VAR-Parameter	61
6.5 Wiederholung Funktionsaufruf	61
7 Unterschied lokale und globale Variable.....	62
8 ERROR-MESSAGE	68
9 Record Variable, GET und Dialogfenster	72
10.1 FIND	74
10.2 SETRANGE und FIND	77
10.3 SETCURRENTKEY, SETRANGE, FIND	80
10.4. FIND mit Abfrage	80
10.5 FIND alle in Filter	80
11 SETFILTER	81
12 Zusammenfassung Filter	81
13 MODIFY	82
14 MODIFYALL.....	84
15 DELETE	84
16 DELETEALL	86
17 INIT.....	87
18 INSERT	88
19 COMMIT	90
20 TESTFIELD, FIELDERROR	93
21 CALCFIELDS, CALCSUMS.....	95
22 BEFEHLE KATEGORIE STRING	98
23 Befehle Kategorie Datum/STRING	102
24 Optionswerte	107
25 Style Guide	115



Kapitel 1

Einleitung

In dem Kapitel Einleitung finden Sie grundlegende Informationen und einen Überblick über die Schulungsunterlagen und deren Inhalt.

Dieses Kapitel besteht aus den Abschnitten:

- Über die Schulung
- Typographische Hervorhebungen

1.1 ÜBER DIE SCHULUNG

Die Schulung Navision Attain Programming II soll Programmieranfängern als Einstieg in die Programmierung dienen. Sie werden sowohl allgemeine Grundlagen der Programmierung als auch die ersten Schritte der Programmierung in Navision Attain lernen. Umsteigern aus anderen Programmiersprachen können sich mit der in Navision Attain benutzten Syntax vertraut machen.

Voraussetzungen für dieses Seminar sind Erfahrungen im Umgang mit dem PC und der Bedienung von Microsoft Windows. Darüberhinaus sollten die Teilnehmer die Schulung Navision Attain Essentials besucht haben. Am Ende des Seminars sollen die Teilnehmer die Voraussetzungen für die Teilnahme am Seminar Navision Attain Solution Developer I erfüllen.

Nachfolgend wird in kurzer Form der Inhalt der einzelnen Kapitel dargelegt.

Grundlagen der Navision Attain Entwicklungsumgebung C/SIDE

In diesem Kapitel werden die Grundlagen der Navision Attain Entwicklungsumgebung C/SIDE vermittelt. Der Object Designer und seine Funktion in Navision Attain wird näher erläutert. Die Hilfefunktion wird inklusive der Syntax für die Befehle erklärt. Im Symbol Menü sind alle Befehle abrufbar. Auf den Aufbau dieses Symbol Menüs wird in diesem Kapitel eingegangen.

Grundlagen allgemeiner Programmierung

Dieses Kapitel beschäftigt sich mit den Grundlagen der allgemeinen Programmierung. In diesem Kapitel soll Programmieranfängern und Umsteigern aus anderen Programmiersprachen die Möglichkeit gegeben werden, allgemeine, wichtige Grundsätze für eine erfolgreiche Programmierung in Navision Attain zu erlernen. Es werden die Grundlagen aufgezeigt für die Definition von Variablen, die Erstellung von Bedingungen, der Bildung von Schleifen und dem Anlegen von Funktionen.

Grundlagen der Programmierung in Navision Attain

In diesem Kapitel werden die Grundlagen für die Programmierung von Navision Attain vermittelt. Alle wichtigen Befehle werden anhand von

Beispielen veranschaulicht. Zusätzliche Variablentypen und verschiedene Begriffe werden eingeführt.

Programmierung nach Style Guide

Style Guide ist ein Regelwerk für die Programmierung unter Navision Attain. Unter dem Gesichtspunkt einer einfachen Lesbarkeit von Programmen, müssen bestimmte Regeln während der Programmierung eingehalten werden. Diese Regeln werden im Folgenden Kapitel beschrieben.

1.2 TYPOGRAPHISCHE HERVORHEBUNGEN

Bevor Sie sich weiter in die Schulungsunterlagen vertiefen, ist es wichtig, dass Sie mit den im Text verwendeten Symbolen und typographischen Konventionen vertraut werden. Die Folgende Tabelle zeigt eine Übersicht, wie die verschiedenen Programmteile unterschiedlich hervorgehoben werden.

Hervorhebung	Element
ALT F1	Tasten der Tastatur. Sie werden in kleinen Kapitälchen geschrieben.
<u>Design</u>	Menüpunkte und Schaltflächen in Fenstern werden viertelfett dargestellt. Die Zugriffstaste wird zusätzlich unterstrichen.
Adresse	Feldnamen und Programmsymbole. Sie sind in halbfett hervorgehoben und beginnen mit einem Großbuchstaben.
<i>Debitorenposten</i>	Namen von Tabellen, Fenstern, Dialogfeldern und Registern. Sie sind halbfett, kursiv geschrieben und haben einen großen Anfangsbuchstaben.
<i>Hansen</i>	Benutzereingaben wie z B. Debitorennamen oder "...geben Sie <i>Ja</i> in dieses Feld ein". Sie sind in Kursivschrift hervorgehoben.
fin.flf	Dateinamen. Sie werden in Kleinbuchstaben, in der Schriftart Courier geschrieben.
↑ ↓ ▾ ➡ ...	Spezielle Symbole, die in den Fenstern erscheinen.

Achtung, Tip, Warnung

.....
In dieser Dokumentation gibt es Abschnitte, die wie dieser hier durch eine punktierte Linie hervorgehoben werden. Sie enthalten Hinweise oder Warnungen, die von besonderer Wichtigkeit sind.
.....

Kapitel 2

Grundlagen der Navision Attain Entwicklungsumgebung C/SIDE

In diesem Kapitel werden die Grundlagen der Navision Attain Entwicklungsumgebung C/SIDE vermittelt. Der Object Designer und seine Funktion in Navision Attain wird näher erläutert. Die Hilfefunktion wird inklusive der Syntax für die Befehle erklärt. Im C/AL Symbol Menu sind alle Befehle abrufbar. Auf den Aufbau dieses Symbol Menu wird in diesem Kapitel eingegangen.

Dieses Kapitel besteht aus den Abschnitten:

- Benutzen des Object Designers
- Benutzen der Hilfe
- Begriffserklärungen
- Das C/AL Symbol Menu

2.1 BENUTZEN DES OBJECT DESIGNERS

Der Object Designer ist die Entwicklungsumgebung (C/SIDE) für die Programmierung von Navision Attain. Dort kann man Änderungen an existierenden Anwendungen vornehmen, vollständig neue Module erstellen und alle Objekte verwalten.

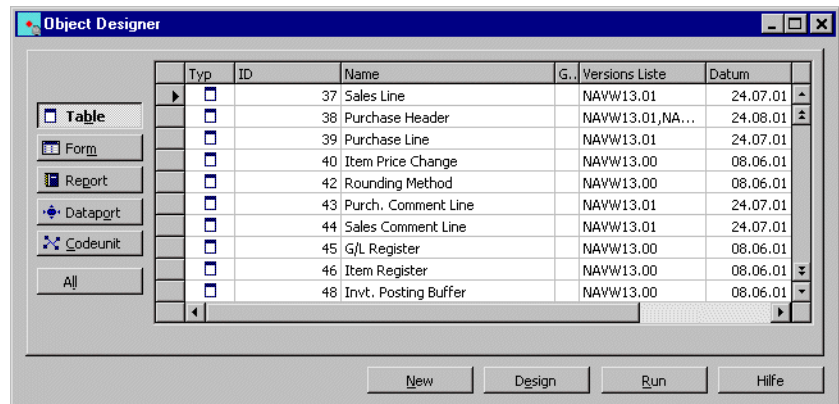
Nahezu alle Programmobjekte von Navision Attain sind hier aufgelistet und können je nach Zugriffsberechtigung bearbeitet werden.

Der Object Designer wird mit der Tastenkombination **SHIFT+F12** gestartet oder über den Menüpunkt **Extras** aufgerufen.

Table, Form, Report, Dataport, Codeunits

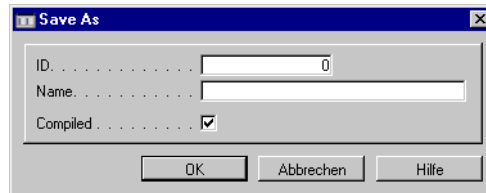
Der Object Designer verwaltet verschiedene Arten von Objekten.

Es gibt folgende Unterteilung: Table (Tabelle), Form (Fenster), Report (Bericht), Dataport (Ein- bzw. Auslesung) und Codeunit (Funktionssammlung).



Folgende Funktionsschaltflächen stehen zur Verfügung:

- **New**
Je nachdem, welche Objektart ausgewählt worden ist, kann ein neues Objekt erstellt werden. Soll das Objekt abgespeichert werden, erscheint folgendes Fenster:



In dem Feld **ID** wird die Objektnummer eingegeben.

In dem Feld **Name** wird ein Name eingegeben, der bis zu 30 Zeichen lang sein kann. Empfehlenswert ist ein aussagekräftiger Name.

Hinweis

Die Codebase in Navision Attain ist englisch. Das bedeutet, dass Variablen und Objekte englisch sind. Um eine konsistente Terminologie auch bei Zusatzprogrammierungen und Anpassungen zu gewährleisten, sollten Sie sich auch für Ihre Objekte und Variablen englische Bezeichnungen wählen.

In dem Feld **Compiled** wird ausgewählt, ob ein Objekt kompiliert gespeichert werden soll. Wenn ein Programm kompiliert wird, ist es lauffähig in die aktuelle Datenbank eingebunden. Falls ein Fehler bei der Kompilierung auftritt, muss dieser zuerst korrigiert werden, bevor das Objekt kompiliert gespeichert werden kann. Wird **Compiled** nicht gesetzt, so kann ein Objekt auch dann abgespeichert werden, wenn es noch Fehler enthält.

- **Design**
Das momentan markierte Objekt wird geöffnet. Es kann bearbeitet werden. Nach der Bearbeitung besteht die Möglichkeit, ein Objekt zu kompilieren.
- **Run**
Das ausgewählte Objekt wird ausgeführt. Mit dieser Schaltfläche kann ein Objekt auf seine Funktion getestet werden.
- **Hilfe**
Aufruf der Hilfefunktion.

Ein- bzw. Auslesen von Objekten

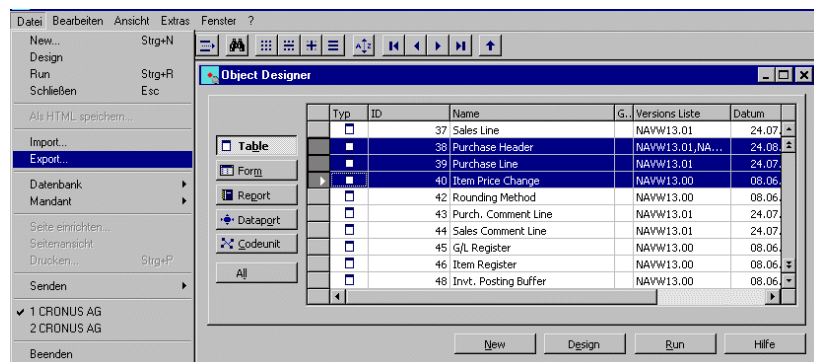
Damit Objekte von einer Navision Datenbank in eine andere Navision Datenbank überspielt werden können, gibt es die Möglichkeit, Objekte in einer Datei abzuspeichern. Diese Datei kann in eine Navision Datenbank ein- bzw. ausgelesen werden.

Mit dem Menüpunkt Datei, I_mport können Objekte eingelesen werden, die zuvor von einer anderen Datenbank ausgelesen wurden. Der Menüpunkt Datei, E_xport_ handelt das Auslesen von Objekten ab.

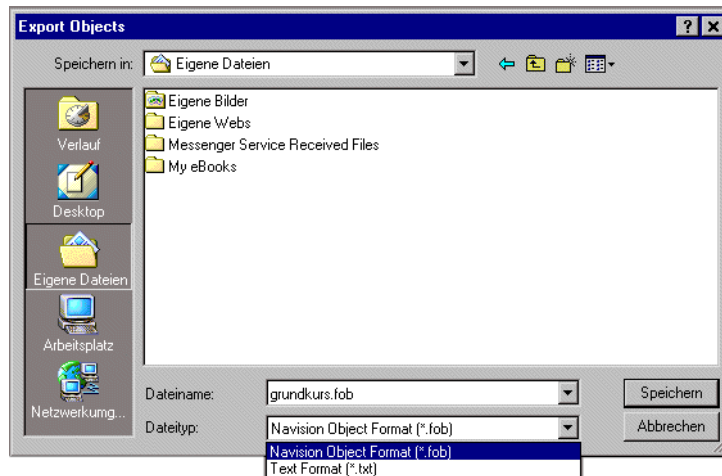
Es existieren zwei Formate für den Ex-/Import von Objekten. Das Standardformat Navision Attain speichert die Objekte als Binärcode ab. Das Format Text speichert die Objekte im Klartext ab, so dass man sie auch mit einem Editor betrachten oder mit anderen Tools weiter verarbeiten kann.

Auslesen von Objekten

Objekte, die exportiert werden sollen, müssen blau markiert sein. Nach Auswahl von Export kann ein Object File abgespeichert werden.

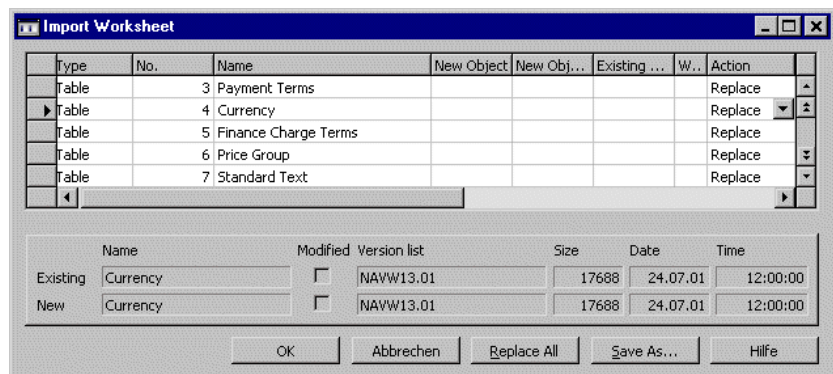


Die Datei mit den Objekten erhält standardmäßig die Endung .fob. Speichert man die Objekte als Text File, erhält die Datei die Endung .txt.



Einlesen von Objekten

Über den Menüpunkt **Import** werden Objekte eingelesen. Nachdem ein Objekt File zur Einlesung ausgewählt worden ist, wird geprüft, ob die einzulesenden Objekte andere, bereits in der Datenbank vorhandene, Objekte überschreiben könnten. Trifft dies zu, wird folgender Dialog geöffnet:



In dem Dialog wird geprüft, ob und wie ein Objekt eingespielt wird.

Wichtig ist das Feld **Action**. Mit diesem Feld wird ausgewählt, wie ein Objekt eingespielt wird. Bei der Überprüfung der einzelnen Objekte wird dem Feld eine bestimmte Option zugewiesen. Ist das neu einzuspielende Objekt noch

nicht in der Zieldatenbank vorhanden, erhält das Feld **Action** den Wert *Create*. Ist das Objekt schon vorhanden, enthält es den Wert *Replace*. Das Feld **Action** hat folgende Bedeutungen:

Action	Erklärung
Create	Dieses Objekt ist in der aktuellen Datenbank nicht vorhanden. Es kann deshalb problemlos eingespielt werden.
Replace	Beim Einspielen wird das bereits vorhandene Objekt durch das neue Objekt ersetzt.
Delete	Das bereits vorhandene Objekt in der Datenbank wird gelöscht. Das neue Objekt wird nicht eingespielt.
Skip	Das neue Objekt wird beim Importieren ignoriert.
Merge Existing < -New	Gilt nur für Tabellen. Die Felder in der Zieldatenbank werden beibehalten. Neue Felder aus der Objektdatei werden hinzugefügt.
Merge New < -Existing	Gilt nur für Tabellen. Alle Felder von der neuen Tabelle werden importiert. Alle zusätzlichen Felder aus der bereits existierenden Tabelle werden hinzugefügt.

Danach können die neuen Objekte durch Betätigen der Schaltfläche OK eingelesen werden.

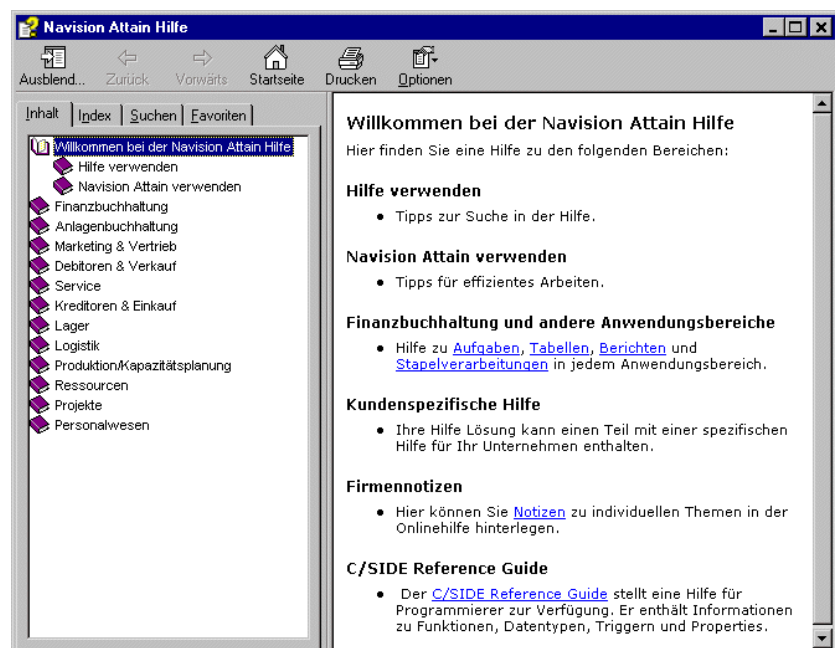
Wurden Indizes oder Indexfelder von Tabellen geändert, werden die Indizes dieser Tabelle neu aufgebaut.

Werden Text Files eingelesen, sind diese noch nicht kompiliert, im Gegensatz zu fob-Files, die bereits im kompilierten Zustand ausgelesen wurden. Text Files müssen noch manuell kompiliert werden.

2.2 BENUTZEN DER HILFE

Die Navision Attain Onlinehilfe ist aufgebaut wie jede Standardhilfe unter Windows. Durch Betätigen der F1-Taste wird die Hilfe aktiviert. Je nach Cursorposition wird entweder die Navision Attain Anwenderhilfe oder die Programmierhilfe aktiviert.

In der Anwenderhilfe gibt es die Möglichkeit, über die Auswahl C/SIDE Reference Guide in die Programmierhilfe zu wechseln.



Hilfeaufbau

Die Programmierhilfe ist in verschiedene Bereiche gegliedert.

Bereich	Erklärung
C/AL Reference by Category	Alle Befehle werden nach Gruppenzugehörigkeit sortiert aufgelistet. D. h., alle Befehle z. B. zum Bearbeiten eines Formulars werden nacheinander alphabetisch aufgelistet.
Data Types	Alle Variablenarten werden alphabetisch aufgelistet.

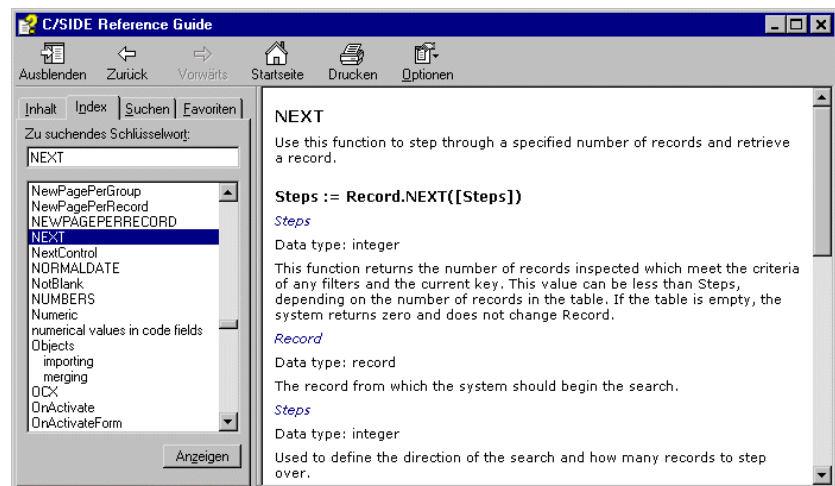
Bereich	Erklärung
Properties	Alle Eigenschaften (Properties) von Objekten /Steuerelementen werden alphabetisch aufgelistet.
Triggers	Alle Ereignissteuerungen (Trigger) werden alphabetisch aufgelistet.
Controls	Die wichtigsten Steuerelemente (Controls) werden alphabetisch aufgelistet.
Information	Allgemeine Information.

Hilfesyntax

Befehle unter Navision haben folgende einheitliche Schreibweise:

Syntax	Erklärung
[]	Ein Ausdruck oder Parameter in eckigen Klammern ist optional. D. h., dieser Ausdruck oder Parameter muss nicht angegeben werden, zum Beispiel: [OK :=] Record.FIND([Which])
()	Innerhalb der Klammer werden Parameter erwartet.
,	Trennzeichen der einzelnen Parameter
...	Es können noch weitere Parameter folgen, zum Beispiel: [OK :=] Record.GET([Value], ...)
Record.	Record ist eine Datensatzvariable. Falls ein Befehl mit diesem Wort auf eine Source-Tabelle eines Objektes angewendet wird, kann die Datensatzvariable weggelassen werden.

Alle anderen Bezeichner werden jeweils mit Angabe des erwarteten Datentyps nach der Syntaxbeschreibung in der Hilfe erklärt. Normalerweise sind zusätzlich zu der Syntaxerklärung des Befehles noch ein oder mehrere Beispiele aufgeführt.



2.3 BEGRIFFSERKLÄRUNGEN

Im Folgenden werden die Begriffe Trigger, Property, Feld und Key erläutert.

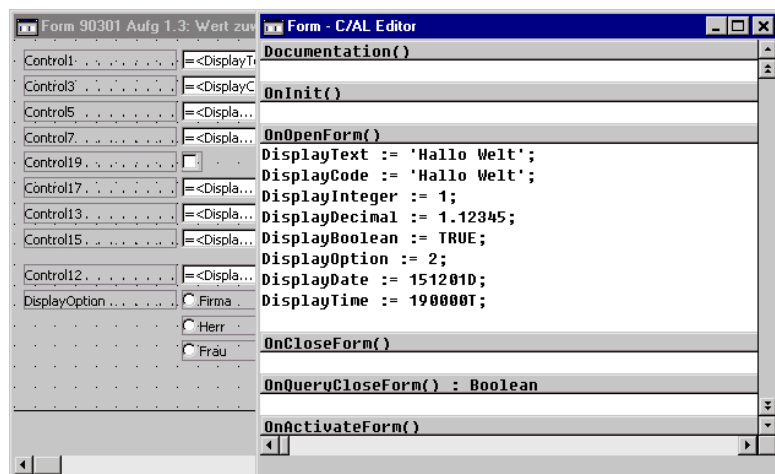
Trigger

Windows-Programme allgemein sind ereignisgesteuert. Jedes Programm wartet auf Aktionen des Benutzers oder auf Aktionen anderer Programme.

Ein Trigger ist ein Ereignis, wie z. B. ein Formular wird geöffnet, ein neuer Datensatz wird angelegt, ein Report wird gedruckt oder ein Feld wird verlassen... Hier beschreibt es die Reaktion des Systems auf ein Ereignis. Tritt ein Ereignis ein, so wird eine Art Nachverarbeitung aktiviert und alle Befehle, die in diesem Trigger stehen, werden abgearbeitet. Anschließend wird der Trigger wieder verlassen. Sichtbar werden die Trigger, wenn ein Objekt im Designmodus geöffnet wird und der C/AL Code aufgerufen wurde.



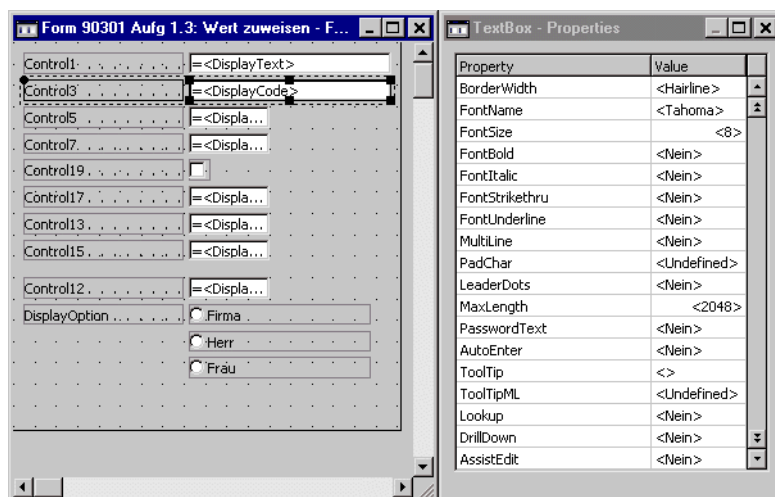
Dieser C/AL Code wird entweder unter dem Menüpunkt Ansicht, C/AL Code oder mit F9 oder mit dem Symbol in der Symbolleiste aufgerufen. Je nachdem, welches Steuerelement markiert ist, werden andere Trigger im C/AL-Code-Fenster sichtbar. Sämtliche Trigger sind in der Hilfefunktion von Navision Attain beschrieben.



Property

Ein Property ist eine von meist mehreren Eigenschaften (Properties) eines Applikationsobjektes. Beispielsweise hat eine Text Box das Property `FontSize`, mit der die Schriftgröße in der Text Box bestimmt werden kann.

Für die Änderung der Eigenschaften eines Steuerelements bietet Attain ein Extra Fenster, das über den Menüpunkt Ansicht, Properties ausgewählt wird, oder über das Symbol in der Symbolleiste. Die einzelnen Properties sind in der Hilfe beschrieben.



Feld

Ein Feld ist die kleinste Einheit in einem Datensatz. Jeder Datensatz besteht aus mehreren Feldern. Felder werden ähnlich wie Variablen definiert. Für jedes Feld gibt es Properties und Trigger.


Key

Ein Key (Schlüssel) einer Tabelle definiert die Suchreihenfolge, in welcher Daten in einer Tabelle sortiert sind. Keys können aus mehreren Feldern bestehen. Der Zugriff auf eine Tabelle wird erheblich beschleunigt, wenn über Felder in einer Tabelle gesucht wird, die in Schlüsseln vorhanden sind.

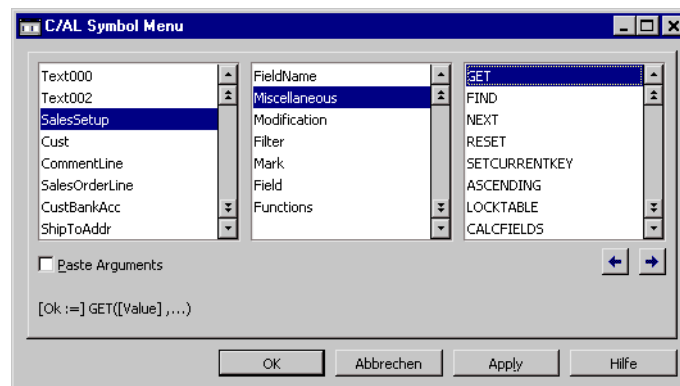
Der erste Schlüssel einer Tabelle wird Primärschlüssel genannt. Es können mehrere Schlüssel pro Tabelle angelegt werden. Primärschlüssel müssen im Unterschied zu Sekundärschlüsseln eindeutig sein.

2.4 DAS C/AL SYMBOL MENU

Mit dem Symbol Menu können alle Variablen und Befehle abgerufen werden.

 Mit F5 oder dem Symbol in der Symbolleiste wird das Menü gestartet.

Mit dem Cursor kann ein Befehl ausgewählt werden, die Syntax wird dabei unten angezeigt. Wird der Befehl mit OK übernommen, wird das Fenster geschlossen. Mit **Apply** wird ein Befehl übernommen, das Fenster bleibt aber offen. Wird in die Checkbox **Paste Arguments** ein Häkchen gesetzt, dann wird die gesamte Syntax in den Code übernommen, ansonsten nur der Befehl.



Kapitel 3

Grundlagen allgemeiner Programmierung

Dieses Kapitel beschäftigt sich mit den Grundlagen der allgemeinen Programmierung. In diesem Kapitel soll Programmieranfängern und Umsteigern aus anderen Programmiersprachen die Möglichkeit gegeben werden, allgemeine wichtige Grundsätze für eine erfolgreiche Programmierung in Navision Attain zu erlernen. Es werden Grundlagen für die Definition von Variablen, für die Erstellung von Bedingungen, für die Bildung von Schleifen und für das Anlegen von Funktionen aufgezeigt.

Dieses Kapitel besteht aus den Abschnitten:

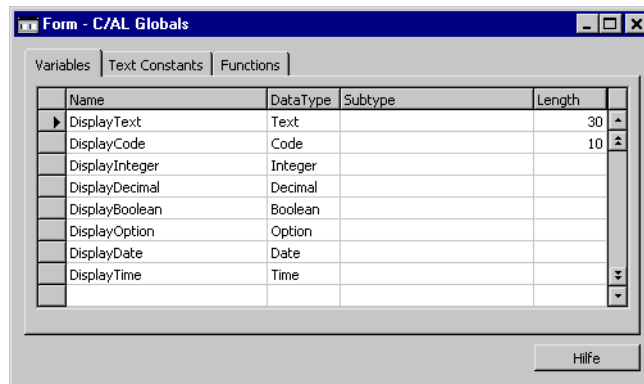
- Variablen
- Textkonstanten
- Bedingungen
- Schleifen
- Funktionen
- Unterschied zwischen globalen und lokalen Variablen

3.1 VARIABLEN

Variablen werden dazu benutzt, veränderbare (variable) oder statische Werte während des Programmablaufes zu verarbeiten.

Anlegen und Definieren von Variablen

Variablen werden während des Designvorgangs eines Objektes entweder als lokale oder globale Variable definiert. Die Variablendefinition wird unter der Menüpunkt Ansicht, C/AL Globals bzw. Ansicht, C/AL Locals ermöglicht.



Variablen müssen vor der Verwendung deklariert (angelegt) werden. Zur Deklaration einer Variablen sind zwei Dinge notwendig:

- 1 Benennen von Variablen (Feld **Name**).
- 2 Angeben des Typs einer Variablen (Feld **Data Type**).

Benennen von Variablen

Wenn eine Variable deklariert wird, muss ein Name für die Variable ausgewählt werden. Nach Möglichkeit sollte ein Name gewählt werden, der den Zweck der Variablen erkennen lässt, damit der Programmcode leichter lesbar ist. Zum Beispiel sind Tag, Name und Summe leichter zu merken als X, Y und Z.

Abgesehen von der Wahl möglichst aussagekräftiger Namen für die Variablen, sollten man einige Regeln beachten, die für Variablen gelten:

Variablennamen können bis zu 30 Zeichen lang sein.

Variablennamen müssen entweder mit einem Buchstaben oder einem Unterstrich (_) beginnen.

Die nachfolgenden Zeichen können Buchstaben, Unterstriche oder die Ziffern 0-9 sein. Es können auch Sonder- und Leerzeichen sein. Sobald Sonder- und Leerzeichen benutzt werden, muss die Variable in Anführungszeichen stehen.

Es sollte kein reserviertes Wort von Navision Attain für eine Variable verwendet werden.

Reservierte Wörter haben für den Compiler eine besondere Bedeutung. Diesen Wörtern ist von Navision Attain schon eine bestimmte Bedeutung zugewiesen worden. Dies kann entweder ein intern benutzter Variablenname sein (z. B. Workdate, UserID ...), oder ein Datentyp (Dialog, Integer ...). Falls reservierte Wörter als Variablen benutzt werden, kann es zu einer Fehlermeldung während des Kompilervorganges oder bei der Programmausführung kommen.

BEISPIEL FÜR VARIABLENNAMEN:

Customer, Stockgroup1, "@Vendor", "Purchase/Sales".

Die Variablenart

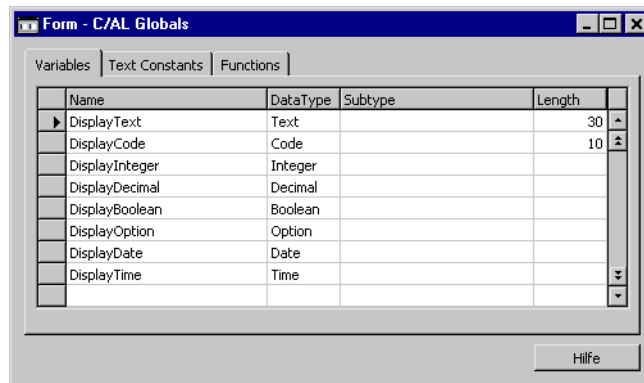
Die Variablenart gibt den Typ der Variable an (z. B. Text, Code oder Boolean). Die Variablenarten werden in den einzelnen Unterpunkten erklärt.

Bei einigen Datentypen (Text, Code) ist zusätzlich noch die Angabe einer Feldlänge notwendig.

Die verschiedenen Variablenarten



Die folgende Abbildung zeigt die verschiedenen Variablenarten in der Anwendung am Beispiel.



Text

Mit Textvariablen lassen sich Texte oder Zahlen speichern. Zahlen werden in einer Textvariablen nicht als Zahlen sondern als Text behandelt. D. h. man kann mit Zahlenwerten innerhalb einer Textvariablen nicht rechnen. Die Textwerte, die den Textvariablen zugewiesen werden, müssen in Hochkommas geschrieben werden. Bei dieser Variablenart ist bei der Deklaration die Angabe einer Feldlänge notwendig, d. h., wie viele Zeichen in ein Feld geschrieben werden können.

BEISPIEL ZUWEISUNG:

Richtig	Falsch
DisplayText := 'Hallo Welt'	DisplayText := 123,45;
DisplayText := '123,45';	

Code

Codevariablen sind den Textvariablen ähnlich. Der wesentliche Unterschied besteht darin, dass sämtliche Buchstaben automatisch in Großbuchstaben übersetzt werden. Bei dieser Variablenart ist bei der Deklaration die Angabe einer Feldlänge notwendig. Dieser Variablentyp wird eingesetzt für Suchfelder, z. B. Suchbegriff im Artikel.

BEISPIEL ZUWEISUNG:

DisplayCode := 'Hallo Welt'; (Variablenwert = "HALLO WELT")

Integer

Integer ist eine Ganzzahlvariable, d. h. nur ganze Zahlen können dieser Variablenart zugewiesen werden

Der Wertebereich dieser Variablen liegt zwischen $+ - 2.147.483.647$.

BEISPIEL ZUWEISUNG:

Richtig	Falsch
DisplayInteger := 1;	DisplayInteger := 'Hallo';
	DisplayInteger := 1.24;

Decimal

Decimal ist eine Variable für Dezimalzahlen. Dieser Variablenart können sowohl Integer als auch Dezimalzahlen zugewiesen werden. Der Wertebereich dieser Variablen liegt zwischen $+ - 10^{63}$

BEISPIEL ZUWEISUNG:

Richtig	Falsch
DisplayDecimal := 1;	DisplayDecimal := 'Haus';
DisplayDecimal := 1.25	

Boolean

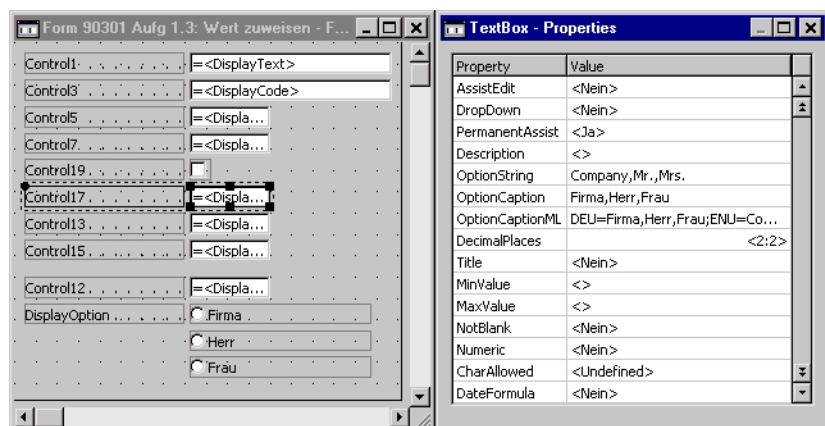
Eine Booleanvariable kann nur die Werte TRUE (wahr bzw. Ja) oder FALSE (falsch bzw. Nein) annehmen.

BEISPIEL ZUWEISUNG:

Richtig	Falsch
DisplayBoolean := TRUE;	DisplayBoolean := ja;
	DisplayBoolean := 'TRUE';
	DisplayBoolean := 1;

Option

Eine Optionsvariable lässt eine Auswahl aus mehreren, fest vorgegebenen Möglichkeiten zu (z. B. die Auswahl Firma/Company, Herr/Mr. oder Frau/Mrs.). Die einzelnen Optionen werden bei der Variablendefinition in dem Control dieser Variablen definiert. Intern werden die Optionswerte als Integerzahlen gespeichert. Dabei geben die Zahlen die Reihenfolge der Optionen wieder (Erste Option = 0, zweite Option = 1, ...). Im Property OptionString der Variablen müssen die Optionen immer in Englisch angegeben werden. Für die Anzeige in anderen Sprachen müssen zusätzlich die Properties OptionCaption und OptionCaptionML der Textbox, die die Optionsvariable anzeigt, gefüllt sein.



BEISPIEL ZUWEISUNG:

Eine Optionsvariable (DisplayOption) hat folgende Optionen: Firma/Company, Herr/Mr., Frau/Mrs. Dies entspricht den Optionswerten 0,1,2 (0 = Firma/Company, 1 = Herr/Mr., 2 = Frau/Mrs.)

Zuweisung	Erklärung
DisplayOption := 2;	(Die Variable enthält den Wert 2. Dies entspricht dem Optionstext Frau)
DisplayOption = DisplayOption :: "Mrs.";	(Die Variable enthält den Wert der Option Frau. Dieser Wert ist 2)

Date

Dies ist die Variablenart für Datumswerte. Die Zuweisung eines Wertes muss mit dem Buchstaben "D" enden.

BEISPIEL ZUWEISUNG:

Zuweisung	Erklärung
DisplayDate := 151201D;	(Der Variablenwert ist der 15. Dezember 2001)
DisplayDate := oD;	(Das Datumsfeld ist leer)

Time

Dies ist die Variablenart für Zeitwerte. Die Zuweisung eines Wertes muss mit dem Buchstaben "T" enden

BEISPIEL ZUWEISUNG:

Zuweisung	Erklärung
DisplayTime := 190000T;	(Der Variablenwert ist 19:00 Uhr)

AUFGABE 1.1 VARIABLEN ERSTELLEN

Erstellen Sie eine leere Form und definieren Sie folgende Variablen, die Sie auf der Form darstellen. Speichern Sie die Form unter 90300.

Name	Datentyp	Properties	Controltyp in der Form
DisplayText	Text		Text Box
DisplayCode	Code		Text Box
DisplayInteger	Integer		Text Box
DisplayDecimal	Decimal		Text Box
DisplayBoolean	Boolean		Check Box
DisplayOption	Option	Firma/Company,Herr /Mr.,Frau/Mrs.	Text Box
DisplayDate	Date		Text Box
DisplayTime	Time		Text Box

AUFGABE 1.2 WERTE EINGEBEN

Geben Sie anschließend in allen Feldern einen Wert ein.

- 1 Wenn Sie im Feld **DisplayText** und **DisplayCode** den gleichen Wert eingeben, worin besteht der Unterschied?
- 2 Verändern Sie die voreingestellte Länge der Variable **DisplayCode** von 10 auf 20 Stellen.
- 3 Versuchen Sie in **DisplayInteger** eine Dezimalzahl einzugeben.
- 4 Wie können Sie in der Variable **DisplayDecimal** eine Dezimalzahl mit 5 Nachkommastellen darstellen?
- 5 Geben Sie die Variable **DisplayBoolean** als Text Box aus. Geben Sie in das Feld die Zahl 0 und 1 ein. Für was steht die 0 und für was die 1?
- 6 Geben Sie die Variable **DisplayOption** als Option Button aus. Welche **OptionValues** müssen für Firma/Company, Herr/Mr. und Frau/Mrs. angegeben werden?
- 7 Geben Sie in der Textbox **DisplayOption** die Werte 0, 1 und 2 ein. Welcher **OptionButton** wird jeweils aktiviert?
- 8 Geben Sie in das Feld **DisplayDate** h ein und in das Feld **DisplayTime** geben Sie z ein. Welche Werte werden angezeigt?

AUFGABE 1.3 WERTE ZUWEISEN

- 1 Als nächstes wird jeder Variable ein Wert zugewiesen. Dies führen Sie im Trigger **OnOpenForm** aus. Speichern Sie die neue Form unter der Nummer 90301.

```
DisplayText := 'Hallo Welt';
```

```
DisplayCode := 'Hallo Welt';
```

```
DisplayInteger := 1;
```

```
DisplayDecimal := 1.12345;
```

```
DisplayBoolean := TRUE;
```

```
DisplayOption := 2;
```

```
DisplayDate := 151201D;
```

```
DisplayTime := 190000T;
```

- 2 Ändern Sie die Werte folgendermaßen ab:

```
DisplayText := Hallo;
```

```
DisplayCode := 123,22;
```

DisplayInteger := 10,12345;

DisplayDecimal := 'Hallo';

DisplayBoolean := JA;

DisplayOption := Herr;

DisplayDate := 311200;

DisplayTime := 120000;

- 3 Kompilieren Sie die Form mit F11. Korrigieren Sie die Werte bis die Form kompiliert werden kann.

Arrays

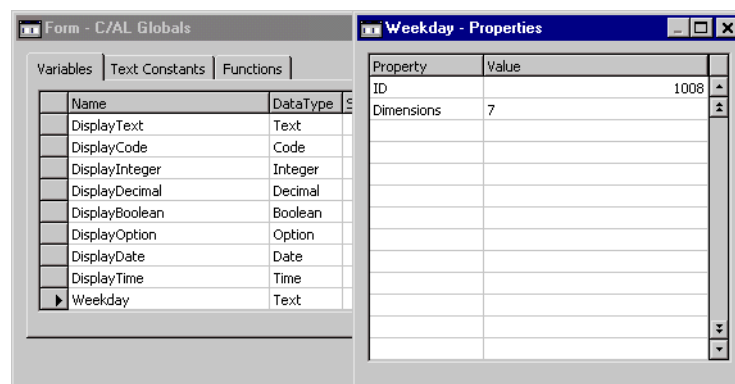
Ein Array ist eine Gruppe Variablen gleichen Datentyps, zusammengefasst in einer einzigen Variablen. Bis zu 10 Dimensionen (Spalten) kann eine Array-Variablen besitzen. Die maximale Anzahl an Variablen (bis 1.000.000) je Dimension wird in dem dazugehörigen Property der Arrayvariablen definiert. Mehrere Dimensionen werden durch Semikolon getrennt.

Der Zugriff auf die einzelne Variable innerhalb des Arrays erfolgt über einen Index, gesetzt in eckige Klammern (Arrayname[2,5]). Arrays sind in Navision Attain beginnend mit 1 indiziert.

BEISPIEL

Die Variable Weekday soll als Text die Namen jedes Wochentages enthalten.

LÖSUNG

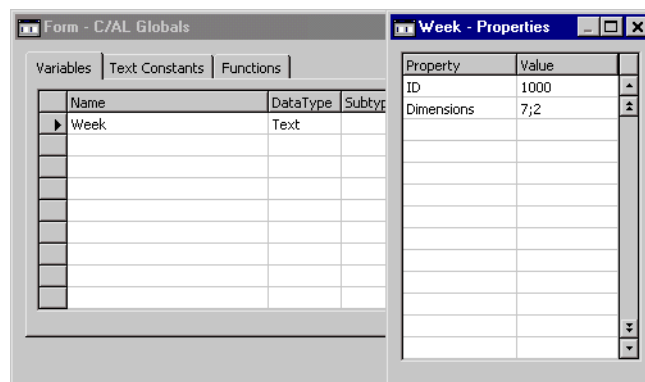


```
Weekday[1] := 'Montag';  
Weekday[2] := 'Dienstag';  
Weekday[3] := 'Mittwoch';  
Weekday[4] := 'Donnerstag';  
Weekday[5] := 'Freitag';  
Weekday[6] := 'Samstag';  
Weekday[7] := 'Sonntag';
```

BEISPIEL

Einem Array *Week*, das zweidimensional ist, sollen die Wochentagsbezeichnungen als Text zugewiesen werden. Außerdem sollen die Kurzbezeichnungen der Wochentage zugewiesen werden.

LÖSUNG



```
Week[1][1] := 'Montag';  
Week[2][1] := 'Dienstag';  
...  
Week[1][2] := 'Mo';  
Week[2][2] := 'Di';  
...
```

AUFGABE 2.1 ANLEGEN UND ANZEIGEN VON ARRAYS

Auf der Form sollen alle Monate dargestellt werden. Jeder Monat wäre eine Textvariable mit Länge 30. Um nicht 12 Variablen anlegen zu müssen, legen Sie die Variable Month mit Dimension 12 an und platzieren den Array auf der Form. Speichern Sie die Form unter der Nummer 90302.

AUFGABE 2.2 WERTE ZUWEISEN

- 1 Weisen Sie im Trigger OnOpenForm folgende Werte zu und speichern Sie die Form unter der Nummer 90303:

```
Month[1] := 'Januar';
```

```
Month[2] := 'Februar';
```

```
Month[3] := 'März';
```

```
Month[4] := 'April';
```

```
Month[5] := 'Mai';
```

```
Month[6] := 'Juni';
```

```
Month[7] := 'Juli';
```

```
Month[8] := 'August';
```

```
Month[9] := 'September';
```

```
Month[10] := 'Oktober';
```

```
Month[11] := 'November';
```

```
Month[12] := 'Dezember';
```

- 2 Versuchen Sie folgende Zuweisung zu programmieren:

```
Month[13] := 'Karneval'
```

Wann und was für eine Fehlermeldung erscheint?

AUFGABE 3 DATENTYPEN IN EINER TABELLE (BEISPIEL FUHRPARK)

Die Felder einer Tabelle haben die gleichen Eigenschaften wie die o. g. Variablen. In dieser Übung legen Sie eine Tabelle an, die in den folgenden Übungen benötigt wird. Es sollen alle Geschäftswagen einer Firma in der Tabelle **Vehicle** gespeichert werden. Speichern Sie die Tabelle unter der Nummer 90300.

Field No.	Field Name	Data Type	Length	Description
1	Kennzeichen/ License No.	Code	10	
2	Fahrzeugbezeichnung/ Cardescription	Text	30	
3	Anzahl Reparaturen/ Number of repairs	Integer		
4	Anschaffungspreis/ Purchaseprice	Decimal		
5	Automarke/ Brand	Option		,BMW,Mercedes,VW,Audi
6	Anschaffungsdatum/ Purchasedate	Date		
7	Anschaffungszeit/ Purchasetime	Time		
8	Für jeden Mitarbeiter/ For every employee	Boolean		
9	Kostenaufwand/ Cost	Decimal		

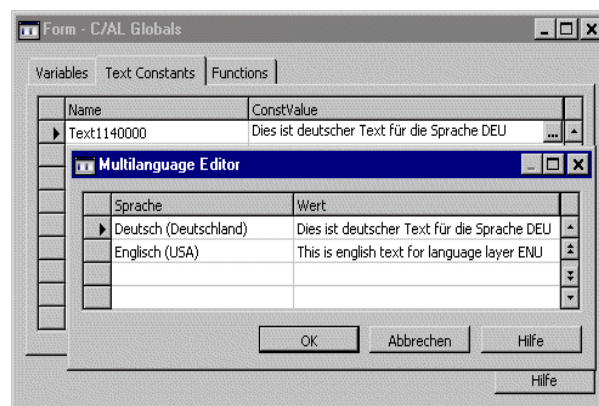
Erstellen Sie ein Kartenformular Fahrzeugkarte/Vehicle Card (Nr. 90304) und ein Tabellenformular Fahrzeugübersicht/Vehicle List (Nr. 90305). Definieren Sie die Übersicht als Lookup. Achten Sie darauf, dass von der Übersicht die Karte und von der Karte die Übersicht aufgerufen werden kann. Erfassen Sie mehrere Fahrzeuge und geben Sie unterschiedliche Werte in die Felder ein.

3.2 TEXTKONSTANTE

Textkonstanten werden für die Darstellung von Texten in einer Multilanguage-Umgebung verwendet. Überall dort wo dem Benutzer Texte angezeigt werden sollen, werden diese Texte nicht "hart" in den Programmtext geschrieben, sondern statt dem eigentlichen Text wird eine Textkonstante verwendet.

Anlegen und Definieren von Textkonstanten

Textkonstanten werden während des Designvorgangs eines Objektes entweder als lokale oder globale Textkonstanten definiert. Unter dem Menüpunkt Ansicht, C/AL Globals bzw. Ansicht, C/AL Locals können Textkonstanten deklariert werden.



- 1 Benennen von Textkonstanten.
- 2 Angeben des Textes für die verschiedenen Sprachlayer.

Benennen von Textkonstanten

Textkonstanten werden mit Text und einer fortlaufenden Nummer bezeichnet.

BEISPIEL FÜR TEXTKONSTANTEN:

Text001, Text002, Text1140000, Text1149999, usw.

Angeben des Textes für Sprachlayer

Der eigentliche Text den die Textkonstanten beinhalten sollen, wird im Multilanguage Editor eingegeben. Hier kann für jeden Sprachlayer die jeweilige Übersetzung hinterlegt werden.

AUFGABE

Legen Sie eine Textkonstante an mit Texten für die Sprachlayer Deutsch (DEU) und Englisch (ENU). Zeigen Sie die Textkonstante in einer Textbox an. Führen Sie anschließend die Form aus und wechseln Sie die Sprache, um die Auswirkungen zu sehen.

3.3 BEDINGUNGEN

Bedingungen werden angewendet, wenn bestimmte Befehlsfolgen nur nach Eintreffen eines bestimmten Ereignisses abgearbeitet werden sollen.

IF-THEN Bedingung

Eine IF-THEN Bedingung ermöglicht die bedingte Verarbeitung eines Befehls in einem Programm.

Syntax :

IF Bedingung THEN Anweisung

Ist die Bedingung erfüllt, wird der Programmcode, welcher nach dem Wort THEN folgt, ausgeführt. Ist die Bedingung nicht erfüllt, wird dieser Code nicht ausgeführt.

BEISPIEL

In Abhängigkeit einer Variablen NumberValue soll eine Nachricht ausgegeben werden, wenn die Variable NumberValue größer 10 ist.

LÖSUNG

```
IF NumberValue > 10 THEN
```

```
    MESSAGE('Die Variable NumberValue ist größer als 10');
```

Hinweis

.....

Damit dieser Code auch multilanguage-fähig ist, müssen Sie den Text des MESSAGE Befehls in einer Textkonstanten ablegen. In den folgenden Beispielen wird nicht mehr speziell auf die Verwendung von Textkonstanten hingewiesen, um die Beispiele verständlicher zu machen. Der multilanguage-fähige Code mit Verwendung einer Textkonstanten sieht folgendermaßen aus:

```
IF NumberValue > 10
```

```
    MESSAGE(TEXT1140000);
```

.....

Wichtig ist, dass 2 Zeilen Code im Beispiel als eine Zeile Code interpretiert werden. Deshalb folgt erst nach der zweiten Zeile ein Semikolon als Zeilenende.

Die Anweisung, die der Bedingung folgt, sollte um 2 Stellen, zwecks besserer Lesbarkeit, eingerückt sein.

Eine Bedingung muss immer als ganzes einen Wert "Wahr" oder "Falsch" als Ergebnis haben. In dem Beispiel wird geprüft, ob die Variable NumberValue größer als 10 ist. Ist die Antwort dieser Prüfung ja, dann wird die Nachricht auf dem Bildschirm angezeigt. Fällt die Prüfung negativ aus, wird nichts angezeigt.

BEISPIEL

Wenn die Variable NumberValue gleich 10 ist und die Variable LineText den Text "Hallo" enthält, soll eine Nachricht ausgegeben werden.

LÖSUNG

```
IF (NumberValue = 10) AND (LineText = 'Hallo') THEN
```

```
    MESSAGE('NumberValue enthält den Wert 10 und LineText enthält den Text Hallo');
```

Werden mehrere Bedingungen miteinander verknüpft, muss jede einzelne dieser Bedingungen in einer Klammer stehen und mit einem Operator verbunden werden.

Dieser Operator heißt entweder AND (und) oder OR (oder).

BEISPIEL

Wenn die Variable NumberValue ungleich 10 ist soll eine Nachricht ausgegeben werden.

LÖSUNG 1

```
IF NumberValue < > 10 THEN
```

```
    MESSAGE('Die Variable NumberValue ist ungleich 10');
```

LÖSUNG 2

```
IF NOT (NumberValue = 10) THEN
```

```
    MESSAGE('Die Variable NumberValue ist ungleich 10');
```

Lösung 2 benutzt den Befehl NOT. Dieser Befehl negiert das Ergebnis einer Bedingung. (NumberValue ist *nicht* gleich 10)

AUFGABE 4.1 EINFACHE IF-BEDINGUNG

In der Tabelle 90300 Vehicle sollen einige Bedingungen programmiert werden.

Wenn das Anschaffungsdatum größer als das heutige Datum ist, dann soll das Anschaffungsdatum auf das heutige Datum gesetzt werden. Programmieren Sie dies im Trigger OnValidate des Feldes **Anschaffungsdatum**/Purchasedate.

In einer Programmiersprache würde dies heißen:

WENN Anschaffungsdatum größer Heute DANN

Anschaffungsdatum ist gleich Heute

AUFGABE 4.2 MEHRFACHE IF-BEDINGUNG

Wenn das Anschaffungsdatum leer ist, dann soll die Anschaffungszeit auch auf leer gesetzt werden, vorausgesetzt in dem Feld **Anschaffungszeit** war ein Eintrag vorhanden.

Programmieren Sie dies im Trigger OnValidate des Feldes **Anschaffungsdatum**.

WENN Anschaffungsdatum gleich leer UND Anschaffungszeit ungleich leer DANN

Anschaffungszeit ist gleich leer

AUFGABE 4.3 NOT-BEDINGUNG

- 1 Wenn eine Eingabe in das Feld Anschaffungszeit gemacht wird und das Anschaffungsdatum ist leer, dann muss eine Fehlermeldung erscheinen.

Programmieren Sie dies im OnValidate Trigger des Feldes Anschaffungszeit.

WENN NICHT Anschaffungsdatum ungleich leer DANN

ERROR('Es muss ein Anschaffungsdatum eingegeben werden.');

- 2 Ein Anschaffungspreis soll nur angegeben werden können, wenn der Wagen nicht für alle Mitarbeiter ist.

Programmieren Sie dies im OnValidate Trigger des Feldes Anschaffungspreis.

WENN NICHT Für alle Mitarbeiter DANN

ERROR('Es darf nur ein Anschaffungspreis angegeben werden, wenn der Wagen für jeden Mitarbeiter ist.');

IF-THEN-ELSE-Bedingung

Im Unterschied zu der IF-THEN-Bedingung gibt es hier eine zusätzliche Entscheidungsmöglichkeit. Ist eine Bedingung nicht erfüllt, wird die Anweisung ausgeführt, die nach der ELSE-Anweisung steht.

Syntax :

IF Bedingung THEN Anweisung ELSE Anweisung

BEISPIEL

Wenn die Variable NumberValue gleich 10 ist, soll eine Nachricht ausgegeben werden, andernfalls soll der Text "Fehler" ausgegeben werden.

LÖSUNG 1

```
IF NumberValue = 10 THEN
```

```
    MESSAGE('Die Variable NumberValue ist gleich 10.')
```

```
ELSE
```

```
    MESSAGE('Fehler.');
```

Zu beachten ist, dass ein Semikolon nur am Ende nach dem ELSE-Zweig gesetzt wird. Eine IF-THEN-ELSE-Anweisung wird als ein einziger Befehl angesehen. Aus diesem Grund könnte man diese IF-THEN-ELSE-Anweisung auch folgendermaßen schreiben:

LÖSUNG 2

```
IF NumberValue = 10 THEN MESSAGE('Die Variable NumberValue ist gleich 10.') ELSE  
MESSAGE('Fehler.');
```

Um eine leichtere Lesbarkeit des Programmcodes zu gewährleisten, ist Lösung 1 anzuwenden.

AUFGABE 4.4 IF-THEN-ELSE

Wenn die Anzahl der Reparaturen größer als 10 ist, soll eine Meldung (MESSAGE) erscheinen, dass der Wagen verkauft werden muss. Ansonsten soll eine Meldung erfolgen, dass die maximale Anzahl an Reparaturen noch nicht erreicht ist. Programmieren Sie dies im Trigger OnValidate des Feldes **Anzahl Reparaturen**.

WENN Anzahl Reparaturen größer 10 DANN

MELDUNG: Auto verkaufen

SONST

MELDUNG: Max. Anzahl noch nicht erreicht

BEGIN..END

Damit mehrere Anweisungen, z. B. in einer IF-Anweisung, als Verbund abgearbeitet werden können, werden die reservierten Wörter BEGIN und END als Verbundbegrenzung benutzt.

BEISPIEL

Wenn die Variable NumberValue gleich 10 ist, soll eine Nachricht ausgegeben werden und die Variable NumberValue soll den Wert 0 erhalten, andernfalls soll der Text "Fehler" ausgegeben werden.

LÖSUNG

```
IF NumberValue = 10 THEN BEGIN
```

```
    MESSAGE('Die Variable NumberValue ist gleich 10.');
```

```
    NumberValue := 0;
```

```
END ELSE
```

```
    MESSAGE('Fehler.');
```

Der Programmcode zwischen BEGIN und END sollte zwecks besserer Lesbarkeit um 2 Stellen nach rechts eingerückt werden

BEISPIEL

Wenn die Variable NumberValue gleich 10 ist, soll eine Nachricht ausgegeben werden und die Variable NumberValue soll den Wert 0 enthalten; andernfalls soll der Text "Fehler" ausgegeben werden und die Variable NumberValue soll den Wert 100 enthalten.

LÖSUNG

```
IF NumberValue = 10 THEN BEGIN
```

```
    MESSAGE('NumberValue ist gleich 10.');
```

```
    NumberValue := 0;
```

```
END ELSE BEGIN
```

```
MESSAGE('Fehler.');
```

```
NumberValue := 100;
```

```
END;
```

AUFGABE 4.5 BEGIN..END

Wenn die Automarke auf leer gesetzt wird, dann sollen auch die Autobezeichnung und die Anzahl Reparaturen zurückgesetzt werden. Programmieren Sie dies im Trigger OnValidate des Feldes **Automarke/Brand**.

AUFGABE 4.6 VERSCHACHTELTE IF-BEDINGUNG

Wenn ein Datensatz geändert worden ist, dann wird der OnModify-Trigger durchlaufen. Hier soll nun programmiert werden, dass folgende Meldungen erscheinen:

Wenn die Autobezeichnung gefüllt ist, und die Anzahl der Reparaturen ist größer o, dann erfolgt die Meldung: 'Eine Autobezeichnung ist ausgewählt und die Anzahl der Reparaturen ist größer o.'

Ist die Autobezeichnung gefüllt, und die Anzahl der Reparaturen ist nicht größer o, dann erfolgt folgende Meldung: 'Eine Autobezeichnung ist ausgewählt und die Anzahl der Reparaturen ist nicht größer o.'

Ist Autobezeichnung nicht gefüllt, dann folgt die Meldung: 'Eine Autobezeichnung ist nicht ausgewählt worden.'

WENN Autobezeichnung ungleich leer DANN

WENN Anzahl Reparaturen größer o DANN

MELDUNG: 'Eine Autobezeichnung ist ausgewählt und die Anzahl Reparaturen ist größer o.'

SONST

MELDUNG: 'Eine Autobezeichnung ist ausgewählt und die Anzahl Reparaturen ist nicht größer o.'

SONST

MELDUNG: 'Eine Autobezeichnung ist nicht ausgewählt worden.'

Der CASE-Befehl

Eine CASE-Anweisung hat eine ähnliche Funktion, wie eine IF-Anweisung. Aufgrund eines Ausdrucks werden je nach Wert des Ausdrucks Anweisungen abgearbeitet. Trifft keine der Bedingungen zu, kann eine ELSE-Anweisung abgearbeitet werden. Der Ausdruck muss ordinal sein (z. B. vom Variablentyp Integer, Boolean oder Option)

Syntax

CASE Ausdruck OF

Konstante1 : Anweisung;

Konstante2 : Anweisung;

.....

ELSE

Anweisung;

END;

BEISPIEL

Eine Variable CurrentMonth wird als Integervariable definiert. Je nach Wert der Variablen CurrentMonth soll eine Nachricht ausgegeben werden, z. B. mit dem Inhalt 'Der aktuelle Monat ist Januar.'

LÖSUNG

CASE CurrentMonth OF

1 : MESSAGE('Der aktuelle Monat ist Januar.');

2 : MESSAGE('Der aktuelle Monat ist Februar.');

3 : MESSAGE('Der aktuelle Monat ist März.');

...

12 : MESSAGE('Der aktuelle Monat ist Dezember.');

ELSE

MESSAGE('Die Variable "Current Month" hat einen falschen Wert.');

END;

AUFGABE 4.7 CASE

Bei Änderung der Automarke soll eine Meldung erscheinen, die die ausgewählte Automarke am Bildschirm anzeigt:

Wenn BMW ausgewählt wird (die Option also 1 ist), erfolgt die Meldung: 'Sie haben BMW gewählt.'.

Wenn Mercedes ausgewählt wird, erfolgt die Meldung: 'Sie haben Mercedes ausgewählt.' etc.

Ist keine gültige Automarke ausgewählt worden, dann folgt die Meldung: 'Keine Automarke ausgewählt.'.

Damit Sie nicht für jede Möglichkeit eine IF-Bedingung programmieren müssen, nutzen Sie die CASE-Bedingung.

IM FALLE DASS Automarke IST

1: MELDUNG: 'Sie haben BMW gewählt.'

...

SONST

MELDUNG: 'Keine Automarke ausgewählt'

ENDE

WITH-Befehl

Normalerweise wird jedes Feld folgendermaßen geschrieben: Tabelle.Feld. Hat man mehrere Feldzuweisungen müsste bei jedem Feld die Tabelle dazugeschrieben werden:

Tabelle.Feld1 :=

Tabelle.Feld2 :=

Tabelle.Feld3 :=

etc.

Wird der WITH-Befehl zur Hilfe genommen, kann man sich die Tabellenzuweisung sparen.

Syntax

WITH Tabelle DO (BEGIN)

Anweisung;

(Anweisung);

(END)

Alle Felder innerhalb der WITH-Anweisung werden der Tabelle zugeordnet, die in dem WITH-Befehl angegeben wurde:

WITH Tabelle DO

Feld1 :=

Feld2 :=

Feld3 :=

etc.

END

BEISPIEL

WITH Item DO BEGIN

INIT;

"No." := '4711';

Description := 'Schreibtisch (natur)';

END;

3.4 SCHLEIFEN

Immer wenn bestimmte Programmteile mehrfach wiederholt werden müssen, kommen Schleifen zum Einsatz. Mit ihnen ist es möglich, bestimmte Programmabschnitte bedingt mehrfach zu wiederholen.

FOR-Schleife

Bei einer FOR-Anweisung wird eine Anweisung wiederholt ausgeführt, während einer Variablen eine fortlaufende Folge von Werten zugewiesen wird. Diese Variable wird auch als Laufvariable bezeichnet. Laufvariablen werden sehr oft nur mit einem Buchstaben definiert (z. B. X, Y). Die ausgeführte(n) Anweisung(en) kann in einer Verbundbegrenzung (BEGIN..END) stehen.

Syntax:

FOR Variable := Anfangswert TO Endwert DO (BEGIN)

Anweisung

(Anweisung)

(END)

BEISPIEL

Es soll 5 mal hintereinander die Meldung "Hallo Welt" und der Variableninhalt der Laufvariablen X auf dem Bildschirm erscheinen.

Lösung:

FOR X := 1 TO 5 DO

MESSAGE('Hallo Welt %1',X);

BEISPIEL

In einer Variablen "Day" steht der Wert eines Wochentages als Integer. In einer Array-Variablen "Weekday" steht der jeweilige Wochentag als Text. Es soll in einer Nachricht der Name des Wochentages angezeigt werden, wenn der dem Wert der Variablen "Day" entspricht.

LÖSUNG

FOR X := 1 TO 7 DO

IF X = Day THEN

MESSAGE('Der aktuelle Wochentag ist %1', Weekday[X]);

Ein weiterer Lösungsweg wäre:

MESSAGE('Der aktuelle Wochentag ist %1', Weekday[Day]);

AUFGABE 5.1 FOR-SCHLEIFE

- 1 Erstellen Sie auf der Fahrzeugkarte ein MenüItem. Im OnPush Trigger soll eine Meldung programmiert werden, dass die Schaltfläche betätigt wurde. Diese Meldung soll 3 mal erfolgen, wobei jedes Mal angezeigt wird, das wievielte mal die Meldung nun erschien.

VON Variable GLEICH 1 bis 3 FÜHRE DURCH

MELDUNG: 'Sie haben den Button gedrückt: %1', Variable

- 2 Die Meldung soll so oft erscheinen wie Reparaturen durchgeführt wurden, außerdem soll eine zweite Variable hochgezählt und angezeigt werden:

VON Variable1 GLEICH 1 bis Anzahl Reparaturen FÜHRE DURCH BEGINN

MELDUNG: 'Sie haben den Button gedrückt: %1. Variable 2: %2', Variable1, Variable2

Neue Variable2 ist gleich alte Variable2 plus 1

ENDE

REPEAT-UNTIL-Schleife

Die Anweisung REPEAT wiederholt eine Anweisung oder eine Folge von Anweisungen, bis eine Bedingung TRUE ergibt. Die Anweisung beginnt mit dem reservierten Wort REPEAT und endet mit dem reservierten Wort UNTIL, das von der auszuwertenden Bedingung gefolgt wird. Die Bedingung ist immer ein boolescher Ausdruck (wahr/falsch).

Da der Ausdruck erst am Ende der Schleife ausgewertet wird, wird die Anweisungsfolge mindestens einmal durchlaufen. Bei REPEAT-UNTIL Schleifen benötigt man keinen BEGIN- .. END-Verbund.

Syntax:

REPEAT

 Anweisung

 (Anweisung)

UNTIL Bedingung

BEISPIEL

Eine Variable TotalAmount soll den Anfangswert 10 bekommen. Eine Schleife soll solange durchlaufen werden, bis die Variable TotalAmount einen größeren Wert als 100 erreicht hat. Die Variable TotalAmount soll in dieser Schleife immer wieder mit 2 multipliziert werden

LÖSUNG

TotalAmount:= 10;

REPEAT

TotalAmount := TotalAmount * 2;

UNTIL TotalAmount > 100;

AUFGABE 5.2. REPEAT-UNTIL-SCHLEIFE

Erstellen Sie auf dem MenuButton der Fahrzeugkarte ein MenuItem. Im OnPush Trigger soll der Kostenaufwand ermittelt werden. Wenn man davon ausgeht, dass jede Reparatur EUR 500,- kostet, dann soll für jede Reparatur der Kostenaufwand um 500 EUR erhöht werden. Zählen Sie dabei eine Variable solange hoch, bis die Anzahl der Reparaturen erreicht ist.

Variable auf 1 setzen

WENN Anzahl Reparaturen nicht 0 DANN BEGINNE

 WIEDERHOLE

 Neuer Kostenaufwand ist gleich alter Kostenaufwand plus 500

 Variable um 1 hoch zählen

 BIS Variable gleich Anzahl Reparaturen ist

ENDE

Hinweis:

.....
 Nach Beendigung des Codes muss der Befehl MODIFY gesetzt werden, damit der Datensatz geändert wird. Dies wird später noch einmal erläutert.

WHILE-DO-Schleife

Während die REPEAT-Anweisung einen Ausdruck am Ende der Schleife auswertet, nimmt die Anweisung WHILE diese Prüfung am Anfang der Schleife vor. Sie beginnt mit dem reservierten Wort WHILE und der zu überprüfenden Bedingung, die immer ein boolescher Ausdruck sein muss. Wird die Bedingung erfüllt (der Ausdruck ergibt TRUE), wird der Programmcode in der WHILE-Anweisung ausgeführt, bis das Ende der Anweisung erreicht und die Bedingung erneut überprüft wird. Sobald die Bedingung den Wert FALSE annimmt, wird die Ausführung der WHILE-Anweisung abgebrochen und die Programmausführung nach der WHILE-Schleife fortgesetzt.

Syntax:

WHILE Bedingung DO (BEGIN)

Anweisung;

(Anweisung);

(END)

BEISPIEL

Eine Variable Test soll den Anfangswert 5 erhalten. Die Variable Test soll solange um den Wert 1 erhöht werden, bis diese den Wert 15 enthält. Diese Anforderung soll mit einer WHILE-Schleife realisiert werden.

LÖSUNG

Test := 5;

WHILE Test < 15 DO

Test := Test + 1;

BEISPIEL

Ein Geldbetrag von EUR 500,-- wird 10 Jahre bei einer Bank angelegt. In den ersten 5 Jahren soll dieser Betrag mit 4 % verzinst werden, in den nächsten 5 Jahren mit 5 %. Zur Berechnung der Summe wird die WHILE-Schleife benutzt.

LÖSUNG

```
Amount := 500;

Years := 1;

WHILE Years <= 10 DO BEGIN

  IF Years <= 5 THEN

    Amount := Amount + (Amount / 100 * 4)

  ELSE

    Amount := Amount + (Amount / 100 * 5);

  Years := Years + 1;

END;

MESSAGE('Die Gesamtsumme ist %1 EUR.',Amount);
```

AUFGABE 5.3 WHILE-DO-SCHLEIFE

In dieser Aufgabe haben wir die gleiche Aufgabenstellung wie in 5.2. Nur soll diesmal das Aufaddieren durchgeführt werden, solange die Variable kleiner gleich der Anzahl Reparaturen ist.

Variable auf 1 setzen

SOLANGE Variable kleiner gleich Anzahl Reparaturen FÜHRE DURCH

Neuer Kostenaufwand ist gleich alter Kostenaufwand plus 500

Variable um 1 hoch zählen

ENDE

Hinweis:

.....
Nach Beendigung des Codes muss der Befehl MODIFY gesetzt werde, damit der Datensatz geändert wird. Dies wird später noch einmal erläutert.
.....

- 1 Was ist der Unterschied zwischen Aufgabe 5.2. und 5.3.? Haben beide das gleiche Ergebnis ? Was passiert, wenn Sie die Anzahl Reparaturen in beiden Aufgaben gleich 1 setzen ?
- 2 Korrigieren Sie den Code so, dass keine Probleme mehr entstehen können.

AUFGABE 5.4 ZUSAMMENFASSENDE AUFGABE

- 1 Erstellen Sie auf der Fahrzeugkarte ein Menüitem.

Im OnPush Trigger soll errechnet werden, wie viel Kapital man in 10 Jahren erhält, wenn anstatt des Autokaufes das Geld für 5 Jahre zu 3 % und weitere 5 Jahre für 4 % angelegt wird. Das Ergebnis soll in einer Meldung ausgegeben werden.

Variable Jahre auf 1 setzen

Geldbetrag ist gleich Kaufpreis

SOLANGE Jahre kleiner gleich 10 FÜHRE DURCH

WENN Jahre kleiner gleich 5 DANN

Neuer Geldbetrag ist gleich alter Geldbetrag plus Zinsen (3 %)

SONST

Neuer Geldbetrag ist gleich alter Geldbetrag plus Zinsen (4 %)

Variable Jahre um eins erhöhen

ENDE

Meldung: Der Geldbetrag ist ..

- 2 Der Ansatz aus 1 wurde mit der WHILE-DO-Schleife erstellt. Programmieren Sie das gleiche mit der REPEAT-UNTIL-Schleife.

Wie muss die UNTIL-Anweisung lauten, damit das gleiche Ergebnis wie in 5.4.1 erscheint?

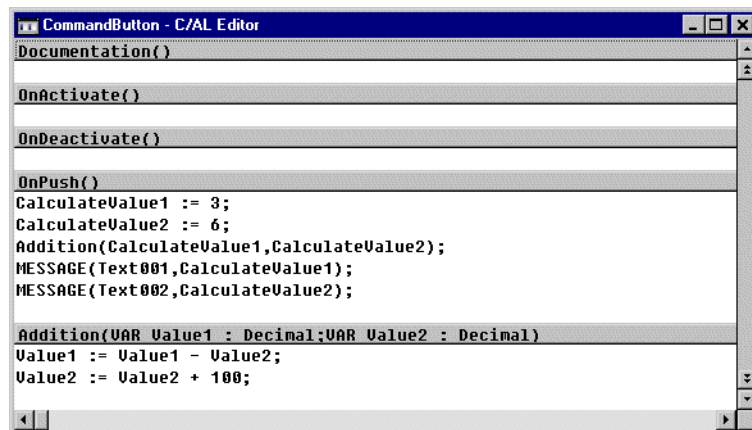
3.5 FUNKTIONEN

Werden bestimmte Programmabschnitte mehrmals in einem Programm benötigt, wie z. B. die Berechnung eines Prozentsatzes, kann dieser Programmblock als Funktion deklariert werden.

Funktionen ohne Parameter

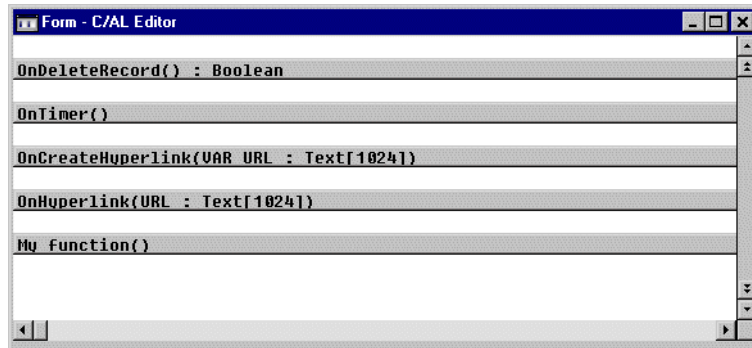
Eine Funktion wird ähnlich wie eine Variable erstellt. Sie wird unter dem Menüpunkt Ansicht, C/AL Globals wie eine Variable definiert. Das Fenster zur Variablendeklaration enthält neben einem Register *Variables* ein Register *Funktion*.

Ein Funktionsname sollte aussagekräftig sein (z. B. "CheckDate" oder "DeleteRecord").

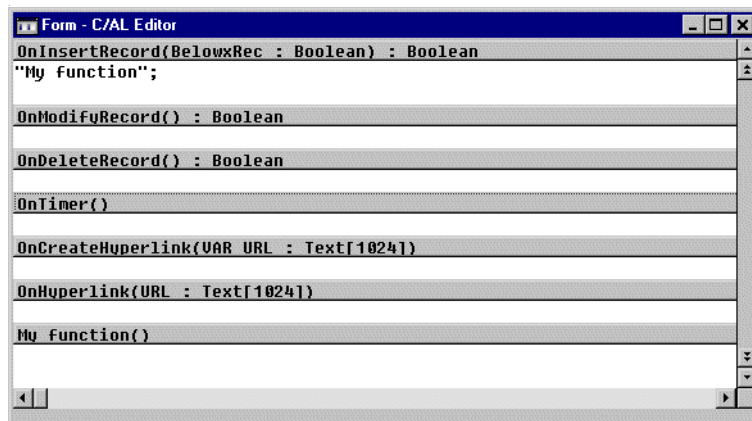


```
Documentation()  
  
OnActivate()  
  
OnDeactivate()  
  
OnPush()  
CalculateValue1 := 3;  
CalculateValue2 := 6;  
Addition(CalculateValue1, CalculateValue2);  
MESSAGE(Text001, CalculateValue1);  
MESSAGE(Text002, CalculateValue2);  
  
Addition(VAR Value1 : Decimal; VAR Value2 : Decimal)  
Value1 := Value1 - Value2;  
Value2 := Value2 + 100;
```

Wird eine Funktion definiert, steht sie im Programmcode zur Verfügung.



Eine Funktion wird aufgerufen durch Angabe des Funktionsnamens.



AUFGABE 6.1 OHNE PARAMETER

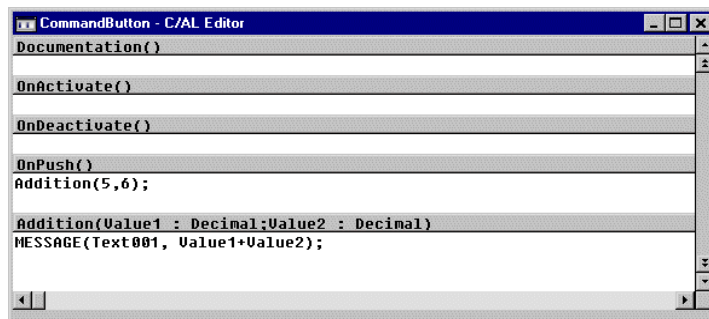
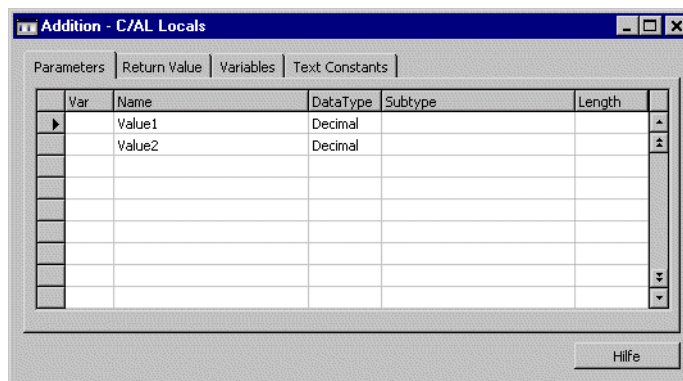
Die Aufgabe 5.4.1 wird an mehreren Orten in der Form aufgerufen. Damit nicht jedes Mal der gleiche Code programmiert werden muss, entscheiden Sie sich, den Code in einer Funktion abzulegen. Definieren Sie hierzu die Funktion *InvestmentCalculation*, in der Sie den Code aus 5.4.1 kopieren. Anschließend erstellen Sie auf der Fahrzeugkarte ein Menütem.

Im OnPush Trigger soll die Funktion *InvestmentCalculation* aufgerufen werden.

Funktionen mit Parameter

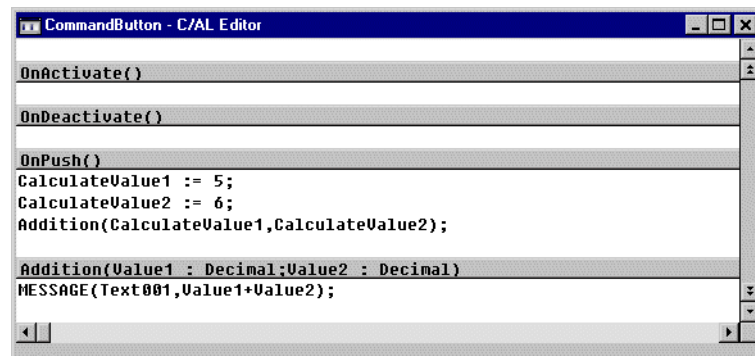
Damit eine Funktion variabel aufgerufen werden kann, benötigt man Parameter (z. B. eine Additionsfunktion soll variable Werte addieren können). Parameter werden wie Variablen behandelt und deklariert. Angelegt werden Parameter unter der Schaltfläche "Locals" in der Funktionsdeklaration.

Funktionsaufrufe für Funktionen mit Parametern müssen in Klammern für jeden Parameter einen Wert oder eine Variable entsprechend des Parametertyps der Funktion enthalten.



Aufruf einer Funktion, die Parameter enthält. Im Beispiel unten werden nicht die Werte 5 und 6 direkt übergeben, sondern zwei Variablen, die diese Werte enthalten. Die Namen der übergebenen Variablen ist dabei unabhängig von

den Parameternamen der Funktion. Entscheidend ist nur die Reihenfolge, d. h., der erste Wert in der Klammer wird an den ersten Parameter der Funktion übergeben.



AUFGABE 6.2 MIT PARAMETER

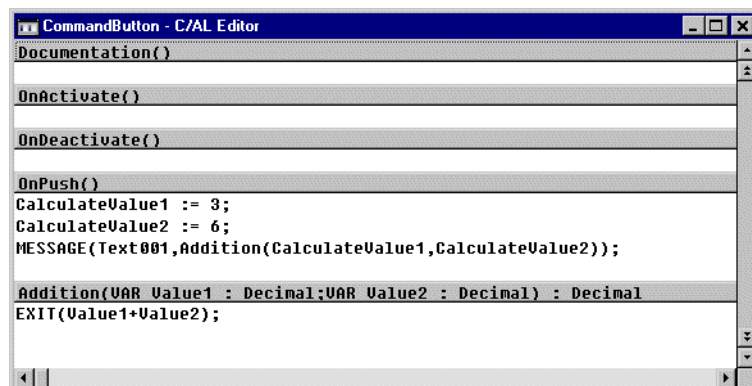
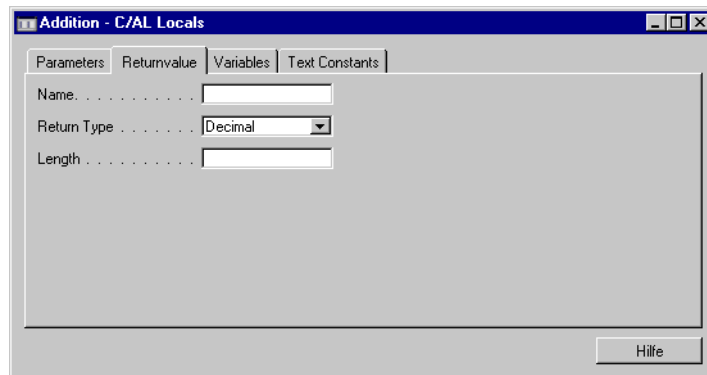
Je nachdem von wo die Funktion aus Aufgabe 6.1 aufgerufen wird, kann es erwünscht sein, dass die Zinssätze und der Betrag unterschiedlich sind und nicht "hart" in die Funktion codiert werden sollen. Aus diesem Grund soll die Funktion so geändert werden, dass die Zinssätze und der Anschaffungspreis übergeben werden.

Definieren Sie die Funktion "*InvCalc With Parameters*" und legen InterestRate1, InterestRate2 und Amount als Parameter an. Kopieren Sie den Code aus der Funktion InvestmentCalculation, ändern dann aber den Code so, dass mit den Parametern gerechnet wird.

Anschließend erstellen Sie auf der Fahrzeugkarte einen MenuButton mit einem Menueitem. Im OnPush Trigger soll die Funktion "*InvCalc With Parameters*" aufgerufen werden.

Funktionen mit Rückgabewert

Soll eine Funktion Werte zurückliefern, muss ein Rückgabewert definiert werden. Der Rückgabewert wird wie die Parameter unter der Schaltfläche Locals definiert. Das Register **Returnvalue** erlaubt die Definition eines Rückgabewertes. Soll mit dem Rückgabewert wie mit einer Variablen weitergearbeitet werden, wird dem Rückgabewert ein Name gegeben. Andernfalls reicht die Zuordnung eines Datentyps.



AUFGABE 6.3 MIT RÜCKGABEWERT

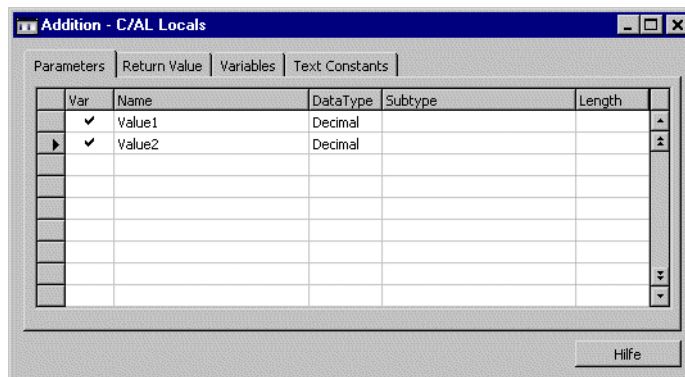
Die Meldung soll nicht in der Funktion erscheinen, sondern immer in dem aufrufenden Trigger. Dadurch kann, je nachdem von wo die Funktion gestartet wird, die Meldung unterschiedlich gestaltet werden. Die Funktion wird so geändert, dass das Ergebnis zurückgegeben wird und in der individuellen Meldung eingebaut werden kann.

Legen Sie die Funktion "InvCalc With Returnvalue" an (Kopie von Funktion aus 6.2). Definieren Sie TotalAmount als Rückgabewert (Decimal) und ändern Sie den Code, so dass das Ergebnis nicht gleich ausgegeben wird, sondern an den aufrufenden Trigger zurückgegeben wird. Anschließend erstellen Sie auf der Fahrzeugkarte ein Menüitem.

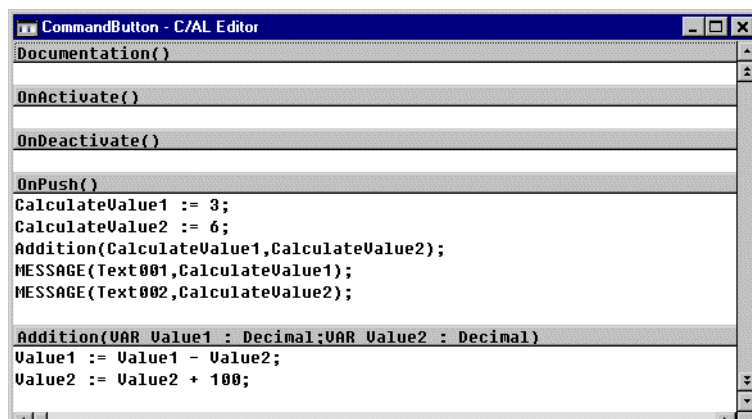
Im OnPush Trigger soll die Funktion "InvCalc With Returnvalue" aufgerufen werden und danach die Meldung erfolgen.

Funktionen mit VAR-Parameter

Soll eine Funktion mehrere Werte zurückliefern, oder eine übergebene Variable direkt verändert werden, so muss dem Parameter bei der Deklaration ein VAR vorangestellt sein. Diese Art der Parameterübergabe wird auch "Call by Reference" genannt, während die normale Parameterübergabe "Call by Value" genannt wird.



Beispiel einer Funktion mit VAR-Parametern:



Hier werden die Variablen CalculateValue1 und CalculateValue2 an die Parameter Value1 und Value2 übergeben. Beim "Call by reference" wird jede Veränderung dieser Parameter(Value1, Value2) an die

Variablen(CalculateValue1, CalculateValue2) zurückgegeben.
CalculateValue1 hat somit nach Verarbeitung der Funktion den Wert -3 und CalculateValue2 den Wert 106.

AUFGABE 6.4 MIT VAR-PARAMETER

In Aufgabe 6.3. wird der Betrag übergeben, neu berechnet und als Ergebnis wieder zurückgegeben. Es ist nur möglich, ein Ergebnis zurückzugeben. Es ist aber auch möglich, den Parameter Betrag als VAR-Parameter zu deklarieren. Dadurch gibt die Funktion automatisch den neuen Betrag zurück. Ein Rückgabewert ist somit nicht notwendig und kann anderweitig benutzt werden.

Legen Sie die neue Funktion "InvCalc With VAR" an (Kopie von Funktion aus 6.3.). Deklarieren Sie den Parameter Amount als VAR-Variable und löschen im Code den EXIT-Befehl. Anschließend erstellen Sie auf der Fahrzeugkarte ein Menüitem.

Im OnPush-Trigger soll die Funktion "InvCalc With VAR" aufgerufen werden und danach die Meldung erfolgen.

AUFGABE 6.5 WIEDERHOLUNG FUNKTIONSAUFRUF

Erstellen Sie die Codeunit 90.329 ,in der Sie wie in Übung 6.3. den Betrag ermitteln, der sich ergeben würde, wenn Sie 5.000,-- EUR 5 Jahre zu 10 % und 5 Jahre zu 11 % anlegen würden. Rufen Sie hierfür die Funktion auf, die Sie in der Fahrzeugkarte implementiert hatten (siehe 6.3.). Das Ergebnis soll auf 2 Dezimalstellen gerundet werden.

Schreiben Sie hierfür nur eine Zeile Code!

Unterschied zwischen globalen und lokalen Variablen

Variablen können sowohl lokal als auch global definiert werden. Werden Variablen global definiert, stehen sie in dem gesamten Objekt zur Verfügung. Werden Variablen lokal definiert, stehen diese Variablen nur in der gerade bearbeiteten Funktion/Trigger zur Verfügung.

Der Vorteil lokaler Variablen liegt darin, dass sie beim Kopieren von Funktionen mit kopiert werden und nicht neu angelegt werden müssen. Bei Verlassen der Funktion wird die lokale Variable geleert. Eine globale Variable steht weiterhin zu Verfügung, sie wird erst zurückgesetzt, wenn das Objekt ‚gecleart‘ wird (siehe Buchungsfunktion). Beim Aufruf des Fensters Navigate werden die Vorteile der globalen Variable zum Beispiel genutzt.

AUFGABE 7 UNTERSCHIED LOKALE UND GLOBALE VARIABLE

In dieser Übung soll der Unterschied von lokalen und globalen Variablen erarbeitet werden. Kopieren Sie die Funktion aus 6.4 in die Funktion "InvCalc With Locals". Definieren Sie in dieser Funktion (unter Locals) die Variable Years (Integer). Es existiert nun die Variable Years als globale und als lokale Variable. Anschließend erstellen Sie auf der Fahrzeugkarte ein Menüitem. Im OnPush Trigger soll die Variable Years auf 100 gesetzt werden. Handelt es sich hierbei um die globale oder lokale Variable? Danach wird die Funktion "InvestmentCalculation With Locals" aufgerufen. In der Meldung soll die Variable Years mit aufgenommen werden:

```
MESSAGE('Der Geldbetrag ist %1 in %2 Jahren.', Amount, Years);
```

Was erhalten Sie für ein Ergebnis?

Starten Sie den Trigger mit dem Debugger und schauen Sie sich die globale und lokale Variable Years an.

Was müsste getan werden, wenn die Funktion "InvCalc With Locals" nur in der Fahrzeugkarte aufgerufen werden soll, nicht aber aus anderen Objekten?

Kapitel 4

Grundlagen der Programmierung in Navision Attain

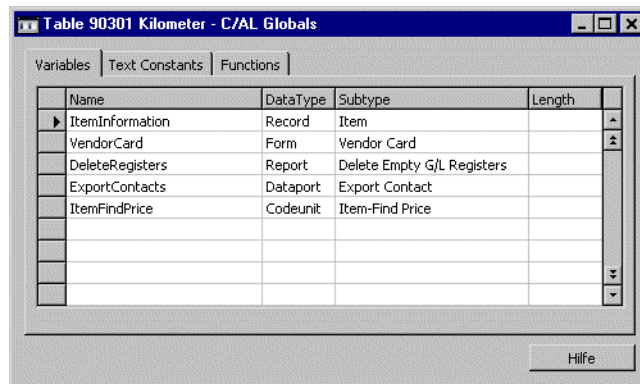
In diesem Kapitel werden die Grundlagen für die Programmierung von Navision Attain vermittelt. Alle wichtigen Befehle werden anhand von Beispielen veranschaulicht. Zusätzliche Variablentypen und verschiedene Begriffe werden eingeführt.


Dieses Kapitel besteht aus den Abschnitten:

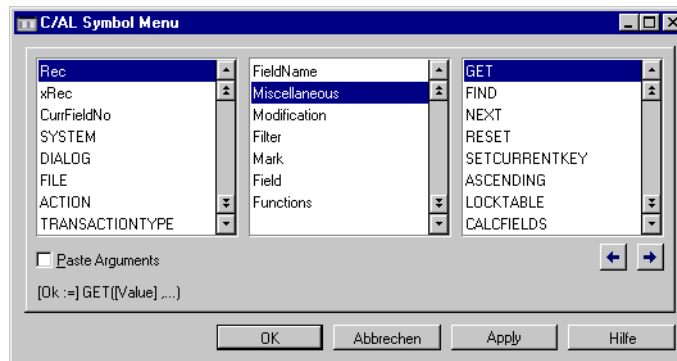
- Variablentypen erweitert
- Befehle der Kategorie Dialog
- Befehle der Kategorie Record
- Befehle der Kategorie System
- Rec/xRec/FIELDNO
- Optionswertbestimmung mit Scope "::"

4.1 VARIABLENTYPEN ERWEITERT

Um Objekte wie Tabellen, Fenster, Reports usw. ausführen zu können, müssen sie als Variablen definiert sein. Mit diesen Variablen können Funktionen, Befehle, Tabellenfelder oder Steuerelemente und die Unterfunktionen der Objekte angesprochen werden. Die folgende Abbildung zeigt die Deklaration von verschiedenen Variablentypen.



Welche Funktionen für die einzelnen Objekte zur Verfügung stehen, wird aus dem Symbol Menu ersichtlich. Das Symbol Menu wird aufgerufen über Ansicht, C/AL Symbol Menu oder mit F5 oder durch Anklicken des Symbols  in der Symbolleiste.



Record

Record ist der Variablentyp für den Zugriff auf einzelne Datensätze einer Tabelle. Alle Tabellenfelder eines Datensatzes können über die Recordvariable angesprochen werden. Zusätzlich können sowohl alle Standardfunktionen als auch die in der Tabelle selbst definierten Funktionen über diese Variable angesprochen werden.

Ein Record wird wie eine normale Variable deklariert. Als Subtype (Zusatzinformation) muss der Tabellename angegeben werden.

BEISPIEL ZUWEISUNG:

```
ItemInformation."No." := '10000';
```

Form

Form ist der Variablentyp für ein Bildschirmfenster. Als Subtype wird für die Deklaration der Name des gewählten Form benötigt.

BEISPIEL

Ein Form Kreditorenkarte soll im Programmcode gestartet werden.

```
VendorCard.Run;
```

Die Variable VendorCard muss vorher als globale oder lokale Variable deklariert werden.

Report

Reportvariablen gestatten den Zugriff auf Reportobjekte (Bericht, Stapel). Als Subtype wird der Name eines Reports benötigt.

BEISPIEL

Ein Bericht soll gestartet werden.

```
DeleteRegisters.Run;
```

Dataport

Das Ein- bzw. Auslesen von Tabellendaten erfolgt am einfachsten über Dataports. Damit diese auch von einem anderen Objekt gestartet werden können, gibt es den Variablentyp Dataport. Deklariert werden Dataports wie die anderen erweiterten Datentypen.

BEISPIEL

Interessenten sollen ausgelesen werden.

ExportContacts.Run;

Benutzung von Codeunits

Codeunits werden eingesetzt, wenn Funktionen oder ganze Programmabläufe in verschiedenen Objekten innerhalb von Navision Attain eingesetzt werden sollen. Um Codeunits in ein Objekt einzubinden, werden diese wie eine Variable definiert.

Es gibt 2 Möglichkeiten, mit Codeunits zu arbeiten.

- Die Codeunit wird mit dem Befehl Run gestartet. Damit wird der Trigger On-Run in der Codeunit aktiviert. Alle Befehle und Funktionsaufrufe, die in diesem Trigger stehen, werden abgearbeitet.
- Der Aufruf einzelner Funktionen. Jede in der Codeunit definierte Funktion kann angesprochen werden, solange deren Property Local = *Nein* lautet.

Aufruf	Erklärung
Codeunit.Run;	//Eine Codeunit wird gestartet. Der OnRun-Trigger wird //ausgeführt.
Codunit.Function;	//Eine Funktion innerhalb der Codeunit wird aufgerufen.

Hinweis:

.....
Funktionen können auch in anderen Objekten wie z. B. Tables, Reports, Forms definiert werden. Auch hier gilt wie bei den Codeunits, solange das Property Local auf NEIN steht, können diese Funktionen von überall in Attain aufgerufen werden.
.....

4.2 BEFEHLE DER KATEGORIE DIALOG

Zu den Befehlen der Kategorie Dialog gehören Error und Message sowie Eingabefenster.

ERROR/MESSAGE

Im Folgenden werden die Befehle Error und Message betrachtet und gegeneinander abgegrenzt.

ERROR

Bricht die Ausführung von Programmcode mit einer Fehlermeldung ab.

Syntax:

ERROR(Text [, Wert1, ...])

Die Fehlermeldung kann frei gestaltet werden und mit Hilfe von Stellvertreterzeichen (%1,%2, ...) können Variableninhalte angezeigt werden. Durch das Einfügen von “\” wird ein Zeilenumbruch erzeugt.

Code	Erklärung
SalesLine.SETCURRENTKEY(Type,"No.");	// Passenden Schlüssel wählen.
SalesLine.SETFILTER("Document Type",'%1 %2', SalesLine."Document Type"::Order, SalesLine."Document Type"::"Return Order");	//Auftragszeilen auf die Belegart //Auftrag/Reklamation und
SalesLine.SETRANGE(Type,SalesLine.Type::Item); SalesLine.SETRANGE("No.,"No.");	//den aktuellen Artikel abgrenzen.
IF SalesLine.FIND('.') THEN ERROR(Text001,TABLECAPTION,"No.", SalesLine."Document Type");	//Falls noch eine Auftragszeile gefunden //wird, die der Abgrenzung entspricht, dann //bricht die Fehlermeldung den //Löschvorgang ab.

MESSAGE

Zeigt eine Meldung auf dem Bildschirm an, die sich genau wie beim Befehl ERROR frei gestalten lässt. Im Gegensatz zu ERROR erfolgt aber kein Abbruch der Ausführung.

Syntax:

MESSAGE(Text[, Wert1, ...])

BEISPIEL

MESSAGE('Dies ist eine Meldung mit\Zeilenumbruch.');

bzw.

MESSAGE(Texto01);



AUFGABE 8 ERROR-MESSAGE

Wenn in Tabelle Fahrzeug/Vehicle im Feld Kostenaufwand/Cost ein Betrag eingegeben wird, so muss überprüft werden, ob dieser kleiner oder größer als der Anschaffungspreis/Purchaseprice ist. Ist der Kostenaufwand kleiner als der Anschaffungspreis so erfolgt eine Fehlermeldung. Andernfalls erfolgt eine Meldung, dass der Kostenaufwand größer oder gleich dem Anschaffungspreis ist. Schauen Sie sich hierzu in der Hilfe die Befehle ERROR und MESSAGE an.

WENN Kostenaufwand kleiner Anschaffungspreis DANN

Fehler: Kostenaufwand darf nicht kleiner als der Anschaffungspreis sein in der Tabelle Fuhrpark/Vehicle

SONST

MELDUNG(Kostenaufwand ist größer/gleich als der Anschaffungspreis in der Tabelle Fuhrpark/Vehicle.

Wenn die Tabelle Fuhrpark/Vehicle oder die Felder **Kostenaufwand/Cost** und **Anschaffungspreis/Purchaseprice** umbenannt werden, dann stimmt die Fehlermeldung nicht mehr mit den Feldnamen überein. Nutzen Sie deshalb die Befehle TABLECAPTION und FIELDCAPTION, damit immer die richtigen Namen für die jeweils gewählte Sprache ausgegeben werden.

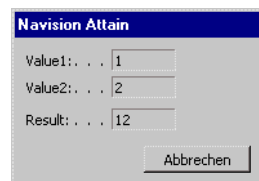
Eingabefenster

Es besteht die Möglichkeit, ein Dialogfenster über C/SIDE Befehle zu programmieren. Hierzu muss eine Variable vom Typ Dialog angelegt werden. Anschließend sind folgende Befehle im Symbol Menu verfügbar:

Syntax	Erklärung
OPEN(String [, Variable1], ...)	Öffnet ein Dialogfenster.
NewControlID := INPUT([ControlID] [,Variable])	Eingabemöglichkeit im Fenster. NewControlID gibt an, auf welchem Platzhalter der Cursor nun steht.
UPDATE([Number][,Value])	Aktualisiert das Fenster.
CLOSE	Schließt das Fenster.

BEISPIEL

Im Folgendem wurde die Variable Fenster als Dialog definiert. Außerdem wurden die Variablen Value1, Value2 und Result jeweils als String benötigt.



Die Tabelle zeigt den für die Lösung einzugebenden Code.

Code	Erklärung
Window.OPEN('Value1: #1####\''+ 'Value2: #2####\''+ 'Result: #3####');	//Öffnet die Variable Window, der Text wird als //String definiert. Das Zeichen \ führt einen //Zeilenumbruch durch. Zur besseren //Übersicht kann der String mit + getrennt //werden. # ist ein Platzhalter. Jeder //Platzhalter erhält eine eindeutige Nummer.
ML-fähig: Window.OPEN (Textoo1 + Textoo2+ Textoo3);	

Code	Erklärung
Window.INPUT(1,Value1);	//Input erwartet eine Eingabe im Platzhalter 1. //Die Eingabe wird in der Variablen Value1 //gespeichert.
Window.INPUT(2,Value2);	
Result := Value1 + Value2;	//Die Strings Value1 und Value2 werden //addiert.
Window.UPDATE(3,Result);	//Ergebnis wird im Platzhalter 3 aktualisiert.
Window.CLOSE;	//Eingabefenster wird geschlossen.

4.3 BEFEHLE DER KATEGORIE RECORD

Zu den Befehlen der Kategorie Record gehören u.a. GET, FIND, SETCURRENTKEY, MODIFY, MODIFYALL, DELETE, DELETEALL, INIT, INSERT, COMMIT, TESTFIELD, FIELDERROR, CALCFIELDS und CALCSUMS. Diese Befehle sind auf Recordvariablen anwendbar.

GET

Die Funktion GET übergibt an die Variable Record einen Datensatz auf Basis der Primärschlüsselfelder einer Tabelle.

Syntax:

[OK :=] Record.GET([Primärschlüsselfeld1,[Primärschlüsselfeld2,...]])

Als Rückgabewert an das Programm liefert die Funktion (TRUE = Satz gefunden, FALSE = Satz nicht gefunden) zurück.

Wird kein Datensatz gefunden und der Rückgabewert nicht abgefragt, so wird eine Fehlermeldung ausgegeben.

BEISPIEL

```
ItemNumber := '4711';
```

```
IF NOT Item.GET('ItemNumber') THEN
```

```
    MESSAGE('Der Artikel mit der Nummer %1 ist nicht vorhanden.',ItemNumber)
```

```
ELSE
```

```
    MESSAGE('Der Artikel wurde gefunden.');
```

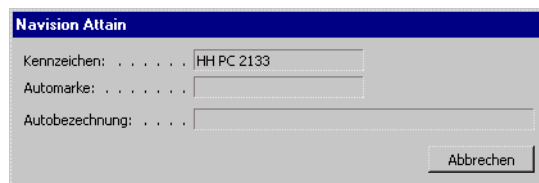
In diesem Beispiel wird ein Datensatz mit der Nummer 4711 gesucht. Der Variablen ItemNumber wird der Wert 4711 übergeben. Danach wird in der IF-Anweisung der Befehl GET ausgeführt. Der Befehl GET übergibt der IF-Anweisung einen Rückgabewert. Ist dieser TRUE (= Satz gefunden) wird eine Meldung ausgegeben, die bestätigt, dass dieser Datensatz gefunden wurde. Automatisch werden die Daten an die Variable Artikel übergeben.

Wurde der Datensatz nicht gefunden, wird eine Fehlermeldung ausgegeben.

AUFGABE 9 RECORD VARIABLE, GET UND DIALOGFENSTER

Erstellen Sie die Codeunit 90300, in der in einem Dialogfenster das Autokennzeichen eingegeben werden muss. Aufgrund des Kennzeichens (Primärschlüssel!) soll der entsprechende Datensatz aus der Tabelle Fuhrpark/Vehicle geholt und die Automarke und die Autobezeichnung im Dialogfenster angezeigt werden. Definieren Sie die Variable Window (Dialog), die Variable ID (Code 10) und die Variable Vehicle (Record Vehicle). Schauen Sie sich die Befehle OPEN, INPUT, UPDATE, CLOSE, GET in der C/SIDE Hilfe an.

Öffnen Sie das Fenster mit folgendem Design (Befehl OPEN)



Im Platzhalter 1 soll die Variable ID eingetragen werden (Befehl INPUT). Aufgrund der ID wird der Datensatz aus der Variable Vehicle geholt (Befehl GET). In Platzhalter 2 wird die Automarke/Cardescription und in Platzhalter 3 die Autobezeichnung/Brand ausgegeben (Befehl UPDATE). Das Fenster wird wieder geschlossen (Befehl CLOSE).

Wenn Sie das Kennzeichen eingegeben haben, wird die Automarke und die Bezeichnung sehr schnell angezeigt und das Fenster sofort geschlossen, so dass Sie das Ergebnis nicht sehen können. Führen Sie deshalb den Befehl INPUT noch einmal aus. Dadurch hält das Programm an und Sie können das Ergebnis in der Dialogbox ansehen.

Was passiert, wenn Sie ein falsches Kennzeichen angeben?

Kann ein Abbruch abgefangen werden?

FIND

Der Befehl FIND hat Ähnlichkeit mit dem Befehl GET. Mit diesem Befehl werden Datensätze gesucht.

Syntax:

[OK :=] Record.FIND([Suchkriterium])

Zwei Arten zum Einsatz des Befehls FIND sind möglich.

- 1 Nachdem die Schlüsselfelder des aktuellen Schlüssels einer Record-Variablen gefüllt sind, kann mit dem Parameter Suchkriterium ein Datensatz gesucht werden.
- 2 Der erste bzw. letzte Datensatz einer Tabelle kann gesucht werden. Wurde zuvor die Tabelle mittels der Befehle SETRANGE oder SETFILTER auf bestimmte Bereiche abgegrenzt, so wirkt sich das generell auf das Ergebnis des Befehls FIND aus.

Bei Abfrage des Rückgabewertes, wird bei erfolgreichem Suchen ein TRUE zurückgegeben. Wird kein Datensatz gefunden, so wird ein FALSE zurückgegeben.

Folgende Suchkriterien sind zulässig:

Suchkriterium	Bedeutung
<	Der nächste Datensatz, der kleinere Werte in den Schlüsselfeldern enthält, als momentan in den Schlüsselfeldern enthalten sind, wird gesucht.
>	Der nächste Datensatz, der größere Werte in den Schlüsselfeldern enthält, als momentan in den Schlüsselfeldern enthalten sind, wird gesucht.
=	Der Datensatz, der dem Inhalt der Schlüsselfelder einer Record-Variablen entspricht wird gesucht.
+	Der letzte Datensatz einer Tabelle wird gesucht. Falls Eingrenzungen in der Tabelle vorhanden sind, wird der letzte Datensatz gesucht, der innerhalb dieser Eingrenzungen vorhanden ist.
-	Der erste Datensatz einer Tabelle wird gesucht. Falls Eingrenzungen in der Tabelle vorhanden sind, wird der erste Datensatz gesucht, der innerhalb dieser Eingrenzungen vorhanden ist.

Wird kein Suchkriterium angegeben, so hat dies die gleiche Wirkung wie "=".

BEISPIEL 1

```
Item."No." := '4711';
```

```
IF NOT Item.FIND('=') THEN
```

```
MESSAGE('Der Artikel mit der Nummer %1 ist nicht vorhanden.', Item."No.");
```

Der Artikel mit der Artikelnummer 4711 wird in der Tabelle Item gesucht. Wenn dieser Artikel in der Tabelle nicht gefunden wurde, wird eine Fehlermeldung ausgegeben.

BEISPIEL 2

Code	Erklärung
ItemLedgEntries.SETRANGE("Item No.", '1992-W');	//Tabelle <i>Item Ledger Entry</i> wird auf //Artikel 1992-W abgegrenzt.
ItemLedgEntries.SETRANGE("Source No.", '30000');	//Tabelle <i>Item Ledger Entry</i> wird //zusätzlich auf die Herkunftsnr. //30000 abgegrenzt.
IF ItemLedgEntries.FIND('-') THEN "Current Number" := ItemLedgEntries."Entry No.;"	//Gibt es dazu einen oder mehrere //Artikelposten, wird von dem ersten //Datensatz, der gefunden wurde, das Feld // Current Number mit dem Feld Entry No. //der Tabelle <i>Item Ledger Entry</i> gefüllt.

AUFGABE 10.1 FIND

Es soll der erste Datensatz aus der Tabelle Vehicle angezeigt werden. Erstellen Sie die Codeunit 90301 und definieren Sie Vehicle als Variable vom Typ Record Vehicle. Suchen Sie den ersten Datensatz. Geben Sie eine Meldung aus, die nur das Kennzeichen des Autos angibt. Ändern Sie den Code und suchen den letzten Datensatz.

Unterschied zwischen GET und FIND

Der Befehl GET wird eingesetzt, wenn ein ganz bestimmter Datensatz gefunden werden soll. Zum Beispiel soll Debitor 10000 gefunden werden. Gibt es keinen Debitor 10000, soll kein anderer Datensatz gefunden werden. Der Befehl GET kann nur eingesetzt werden, wenn über die Schlüsselfelder des Primärschlüssels gesucht wird. Außerdem kann der Befehl GET nur verwendet werden, wenn der Inhalt aller Felder, die zum Primärschlüssel gehören, genau bekannt ist, also der Datensatz genau identifiziert werden kann.

Der Befehl FIND dagegen wird eingesetzt, wenn ein Datensatz eines bestimmten Bereiches gefunden werden soll. Beispielsweise die erste Auftragsposition des Auftrages 12345. Der Befehl gibt hier immer den ersten Datensatz der gewählten Sortierung zurück, egal ob die Zeilennummer der Standardnummer 10000 entspricht oder z. B. 22500 lautet.

Als Regel dient: Wenn aufgrund des Primärschlüssel ein Datensatz gesucht werden kann, dann nimmt man den Befehl GET. Wird aufgrund des Sekundärschlüssel gesucht, dann nutzt man den Befehl FIND.

Hinweis:

.....
 Während der FIND Befehl Datensätze nur innerhalb des gefilterten Bereiches sucht, arbeitet der GET Befehl immer auf allen Datensätzen in der Tabelle, egal welcher Filter gesetzt ist.

SETCURRENTKEY

Der Befehl SETCURRENTKEY wählt einen bereits definierten Schlüssel (Sortierreihenfolge) einer Tabelle aus, nach welchem die Variable Record in der Folge sortiert wird.

Syntax:

[OK :=] Record.SETCURRENTKEY(Schlüsselfeld1[,Schlüsselfeld2,
 Schlüsselfeld3,.....])

Wenn der Rückgabewert abgefragt wird, so gibt der Befehl bei erfolgreicher Auswahl der Sortierreihenfolge ein TRUE zurück. Existiert der Schlüssel so wie angegeben nicht, oder ist der Schlüssel nicht aktiviert, dann wird ein FALSE zurückgegeben.

BEISPIEL

Item.SETCURRENTKEY("Shelf/Bin No.");

In diesem Beispiel wird die Tabelle Artikel nach der Regalnummer sortiert.

SETRANGE

Mit dem Befehl SETRANGE wird ein bestimmter Bereich innerhalb einer Tabelle abgegrenzt.

Syntax:

Record.SETRANGE(Feld[,Von Wert].Bis Wert)

Nach Einsatz dieses Befehls verhält sich eine Tabelle so, als würde sie nur noch aus dem abgegrenzten Bereich bestehen.

Eine Abgrenzung kann aufgehoben werden durch den Befehl RESET oder den Befehl SETRANGE ohne untere und obere Grenze.

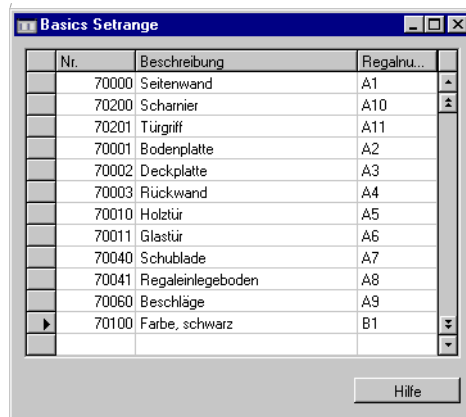
Wird nur eine Grenze angegeben, wird genau auf diesen Bereich abgegrenzt.

Soll auf mehrere Felder abgegrenzt werden, so muss für jedes Feld der Befehl SETRANGE einzeln benutzt werden.

BEISPIEL

Im Folgenden wird auf die Regalnummer A1 bis B1 abgegrenzt.

Code	Erklärung
Item.SETCURRENTKEY("Shelf/Bin No.");	//Wählt die passende Sortierreihenfolge //nach Regalnummer
Item.SETRANGE("Shelf/Bin No.", 'A1', 'B1');	// Grenzt auf die Regalnummern 'A1' bis // 'B1' ab.
Item.SETRANGE("Shelf/Bin No.")	// Setzt die Abgrenzung auf das Feld //Regalnummer zurück



AUFGABE 10.2 SETRANGE UND FIND

Kopieren Sie Codeunit in Codeunit 90302. In dieser Aufgabe soll nur der erste Datensatz ausgegeben werden, der zur Automarke BMW gehört. Grenzen Sie also auf Automarke/Brand gleich 1 ab.

SETFILTER

Der Befehl SETFILTER wendet einen bestimmten Filterausdruck auf ein Tabellenfeld eines Datensatzes an.

Syntax:

Record.SETFILTER(Feld, Filterausdruck[,Filterwert,...])

Nach Einsatz dieses Befehls verhält sich eine Tabelle, als wenn diese Tabelle nur noch aus dem gefilterten Bereich bestehen würde. Ein Filter kann aufgehoben werden durch den Befehl RESET.

Innerhalb des C/AL-Codes sind auch Platzhalter/Filterwerte erlaubt. Diese Filterwerte müssen einen identischen Datentyp wie das gefilterte Feld haben.

verwendbare Filterausdrücke sind :

Filterausdruck	Bedeutung
..	Bereich von bis ("100..200")

Filterausdruck	Bedeutung
&	Und Verknüpfung ("100..200 & 300..400")
	Oder Verknüpfung ("100 200 300")
<	Kleiner als (" < 300")
<=	Kleiner oder gleich als (" <= 200")
>	Größer als (" > 300")
>=	Größer oder gleich als (" >= 200")
< >	Ungleich (" < > 300 & < > 300..500")
*	Platzhalter für beliebige Zeichenfolge

BEISPIELE

Code	Erklärung
Item.SETFILTER("Shelf/Bin No.", '< > A1');	//Alle Artikel mit einer Regalnummer //ungleich "A1" werden angezeigt
Item.SETFILTER("Shelf/Bin No.", 'A1 B1');	//Alle Artikel mit der Regalnummer // "A1" oder einer Regalnummer "B1" //werden angezeigt.
NumberFrom := '70000';	//Variable NumberFrom erhält den //Wert 70000
NumberTo := '71000';	//Variable NumberTo erhält den Wert //71000
Item.SETFILTER("No.", "%1..%2", NumberFrom, NumberTo);	// Alle Artikel, die eine Artikelnummer // zwischen 70000 und 71000 haben // werden angezeigt.

Unterschied zwischen SETRANGE und SETFILTER

■ Mit SETFILTER sind im Unterschied zu SETRANGE wesentlich komplexere Filter möglich.

Der Befehl SETRANGE erlaubt nur, auf einen einzelnen Wert bzw. auf einen Bereich zwischen einem Maximum und einem Minimum abzugrenzen, während SETFILTER mehr Varianten bietet.

Normalerweise wird, sofern möglich, immer der Befehl SETRANGE benutzt.

NEXT

Die Funktion bewegt sich abhängig vom Schlüssel und den gewählten Filtern schrittweise durch eine Gruppe von Datensätze einer Tabelle, und liefert bei Erfolg den nächsten Datensatz.

Syntax:

AnzahlSchritte := Record.NEXT([Schritte])

Als Rückgabewert gibt die Funktion die Anzahl der Schritte zurück, die sie weitergehen konnte. Das bedeutet, wird eine 0 zurückgegeben folgt kein weiterer Datensatz.

Wird der Parameter Schritte nicht angegeben, wird, wenn möglich der nächste Datensatz geholt (Schritt ist also automatisch 1).

Wert des Parameters Schritte	Bedeutung
< 0	Anzahl Schritte rückwärts in der Tabelle.
> 0	Anzahl Schritte vorwärts in der Tabelle.
= 0	Keine Auswirkung auf die Tabelle.

BEISPIEL

Code	Erklärung
Item.SETCURRENTKEY("Shelf/Bin No.");	//Passenden Schlüssel wählen
Item.SETRANGE("Shelf/Bin No.", 'A1');	//Eingrenzung auf Artikel mit der //Regalnummer A1
IF Item.FIND('-') THEN	//Wenn Artikel in der Abgrenzung //gefunden wird
REPEAT	//REPEAT-Schleife Anfang
Item."Unit Price" := Item."Unit Price" * 1.05;	//Erhöht den VK-Preis um 5 %
Item.MODIFY(TRUE);	//Ändert den aktuellen Datensatz in der //Tabelle

Code	Erklärung
UNTIL Item.NEXT = 0;	//Die REPEAT-Schleife wird solange //durchlaufen, bis kein nächster //Datensatz gefunden wird.

RESET

Löscht alle Abgrenzungen auf eine Tabelle und wechselt auf den Primärschlüssel.

Syntax:
Record.RESET

BEISPIEL

ItemLedgerEntries.RESET

AUFGABE 10.3 SETCURRENTKEY, SETRANGE, FIND

Bei vielen Datensätzen hätte der Code aus 10.2 eine schlechte Performance. Da die Datensätze nach Kennzeichen sortiert sind. Das System geht also die Datensätze sequentiell durch und überprüft die Automarke. Dies kann bei vielen Datensätzen sehr lange dauern. Die Datensätze müssen vorher also erst nach Automarke sortiert werden. Definieren Sie in der Tabelle Fuhrpark/Vehicle den Sekundärschlüssel Marke/Brand. Kopieren Sie die Codeunit 90302 in Codeunit 90303. Wählen Sie vor der Abgrenzung den Schlüssel aus.

AUFGABE 10.4. FIND MIT ABFRAGE

Wenn kein Datensatz innerhalb der Abgrenzung gefunden wird, bricht das Programm ab. Dies soll abgefangen werden. Fragen Sie das Ergebnis der Suchfunktion ab. Nur wenn diese erfolgreich war, soll eine Meldung erscheinen. Erstellen Sie hierfür die Codeunit 90304.

AUFGABE 10.5 FIND ALLE IN FILTER

- 1 Es sollen nicht nur der erste, sondern alle Datensätze innerhalb der Abgrenzung ausgegeben werden. (Codeunit 90305)

Hierzu muss der erste Datensatz innerhalb der Abgrenzung gesucht werden. Wird dieser gefunden, wird eine Schleife begonnen. In dieser Schleife wird nach der Ausgabe auf den nächsten Datensatz gesprungen bis kein Datensatz mehr vorhanden ist.

Setze Schlüssel auf Automarke/Brand.

Grenze auf Automarke gleich 1 ab.

WENN der erste Datensatz gefunden wird DANN

WIEDERHOLE

MELDUNG Fuhrpark.Kennzeichen

BIS Schritt zu nächstem Datensatz gleich 0

- 2 Ändern Sie den Filter so ab, dass Autos der Automarken BMW und Mercedes angezeigt werden (Codeunit 90306).

AUFGABE 11 SETFILTER

In Übung 10 wurde auf eine Automarke oder auf einen Bereich abgegrenzt. Nun sollen weitere Abgrenzungen getätigt werden. Diese sind mit dem Befehl SETRANGE teilweise nicht möglich. Legen Sie für jede Teilübung eine Codeunit an.

A	Abgrenzen auf Automarke BMW (1) und VW (3)	Codeunit 90307
B	Abgrenzen auf Automarke ungleich leer	Codeunit 90308
C	Abgrenzen auf Automarke von BMW bis VW	Codeunit 90309
D	Abgrenzen auf Automarke ab Mercedes	Codeunit 90310
E	Abgrenzen auf Automarke ab Mercedes und ungleich Audi	Codeunit 90311
F	Abgrenzen auf Autobezeichnung ungleich leer	Codeunit 90312
G	Abgrenzen auf Kaufpreis > 4000	Codeunit 90313
H	Abgrenzen Kaufpreis > 20000 und < 60000	Codeunit 90314
I	Abgrenzen auf Kaufpreis < 40000 oder > 60000	Codeunit 90315

AUFGABE 12 ZUSAMMENFASSUNG FILTER

In Aufgabe 11 wurde immer nur auf ein Feld ein Filter gesetzt. Nun soll eine Abgrenzung auf mehrere Felder erfolgen. Erstellen Sie die Codeunit 90316, in der alle Autos der Automarken BMW und alle Autos mit Kaufpreis größer als 50000 angezeigt werden.

Gibt es einen Schlüssel, den man zur Performancesteigerung auswählen kann?

Setzen Sie anschließend die Sortierung wieder auf den Primärschlüssel zurück, und löschen Sie alle gesetzten Filter und Abgrenzungen.

MODIFY/MODIFYALL

Im Folgenden werden die Befehle MODIFY und MODIFYALL beschrieben.

MODIFY

Verändert den aktuellen Datensatz einer Tabelle in der Datenbank.

Syntax:

[OK :=] Record.MODIFY([RunTrigger])

Bei Abfrage des Rückgabewertes, wird bei erfolgreichem Ändern des Datensatzes ein TRUE zurückgegeben. Wenn der Rückgabewert nicht abgefragt wurde, kommt es zu einer Fehlermeldung, wenn der Datensatz in der Tabelle nicht gefunden werden konnte. Unabhängig von jeder Abgrenzung identifiziert das System den zu ändernden Datensatz anhand des Inhalts der Primärschlüsselfelder.

Der Parameter RunTrigger ist ein Boolean-Wert. Wird dieser auf TRUE gesetzt, wird nach der Änderung des Datensatzes der OnModify Trigger der Tabelle ausgeführt. Standardmäßig wird dieser Trigger nicht ausgeführt.

BEISPIEL

Code	Erklärung
IF Item.GET('70000') THEN BEGIN Item.Description := 'Schreibtischlampe'; Item."Shelf/Bin No." := '42';	//Wenn Artikel 70000 gefunden wurde, //wird das Feld Beschreibung und das //Feld Regalnummer geändert.
Item.MODIFY(TRUE); END;	//Artikel wird geändert, und der //OnModify Trigger der Tabelle Artikel //wird ausgelöst.

AUFGABE 13 MODIFY

Alle Autos der Marke Audi sollen für alle Mitarbeiter zur Verfügung stehen. Erstellen Sie eine Codeunit (90317), in der in einer Schleife das Feld **Für jeden Mitarbeiter** mit einem Haken versehen wird, sofern es noch nicht gesetzt worden ist.

Setze Schlüssel auf Automarke.

Grenze auf Automarke Audi (4) ab.

WENN der erste Datensatz gefunden wird DANN

WIEDERHOLE

WENN Feld "Für jeden Mitarbeiter" gleich FALSCH DANN BEGINN

"Für jeden Mitarbeiter" ist GLEICH WAHR

Modifiziere Datensatz

ENDE

..BIS Schritt zu nächstem Datensatz gleich o

MODIFYALL

Die Funktion weist einem Tabellenfeld einen neuen Wert zu, wobei alle Datensätze verändert werden, die sich innerhalb der aktuellen, vorher definierten Abgrenzung befinden.

Syntax

[OK :=] Record.MODIFYALL(Feld, NeuerWert [,RunTrigger])

Bei Abfrage des Rückgabewertes, wird bei erfolgreichem Ändern aller Datensätze ein TRUE zurückgegeben.

Der Parameter RunTrigger ist ein Boolean-Wert. Wird dieser auf TRUE gesetzt, wird vor der eigentlichen Änderung der Datensätze der OnModify Trigger der Tabelle für jeden geänderten Datensatz ausgeführt. Standardmäßig wird dieser Trigger nicht ausgeführt, da das Ändern der Datensätze somit schneller abläuft.

BEISPIEL

Code	Erklärung
Item.SETCURRENTKEY("Shelf/Bin No.");	//Schlüssel Regalnummer wird gesetzt.
Item.SETRANGE("Shelf/Bin No.", 'A1');	//Tabelle Artikel wird auf Regalnummer //A1 eingegrenzt.
Item.MODIFYALL("Statistics Group", 1, TRUE);	//In Artikeln, die sich innerhalb der //Abgrenzung befinden, wird das Feld //Statistikgruppe mit dem Wert 1 gefüllt. //Danach wird der OnModify-Trigger der //Tabelle Artikel ausgeführt.

AUFGABE 14 MODIFYALL

Bei allen BMW Autos soll das Feld Für jeden Mitarbeiter auf NEIN gesetzt werden. Dieses mal soll nicht abgefragt werden, ob das Kennzeichen evtl. schon gesetzt ist. Ändern Sie das Feld für alle Datensätze innerhalb der Abgrenzung. Erstellen Sie Codeunit 90318

Setze Schlüssel auf Automarke.

Grenze auf Automarke BMW (1) ab.

Modifiziere jeden Datensatz mit dem neuen Wert.

DELETE/DELETEALL



Im Folgenden werden die Befehle DELETE und DELETEALL erläutert.

DELETE

Die Funktion löscht den aktuellen Datensatz einer Tabelle.

Syntax:

[OK :=] Record.DELETE([RunTrigger])

Bei Abfrage des Rückgabewertes, wird bei erfolgreichem Löschen des Datensatzes ein TRUE zurückgegeben. Wenn der Rückgabewert nicht abgefragt wird, kommt es zu einer Fehlermeldung, wenn der Datensatz in der Tabelle nicht gefunden wurde. Unabhängig von jeder Abgrenzung identifiziert das System den zu löschenden Datensatz anhand des Inhalts der Primärschlüsselfelder.

Der Parameter RunTrigger ist ein Boolean-Wert. Wird dieser auf TRUE gesetzt, wird vor der eigentlichen Löschung des Datensatzes der On-DELETE Trigger der Tabelle ausgeführt. Standardmäßig wird dieser Trigger nicht ausgeführt.

AUFGABE 15 DELETE

In dieser Codeunit (90319) soll auf alle Autos abgegrenzt werden, bei denen kein Kaufdatum erfaßt wurde. In einer Schleife wird überprüft, ob ein Kaufpreis vorhanden ist. Wenn nicht, wird der Datensatz gelöscht. Am Ende der Codeunit soll die Anzahl der gelöschten Datensätze ausgegeben werden.

Grenze auf Kaufdatum leer ab.

WENN der erste Datensatz gefunden wird DANN

WIEDERHOLE

WENN Feld Kaufpreis gleich o DANN BEGINN

Zähle einen Zähler hoch

Lösche Datensatz

ENDE

..BIS Schritt zu nächstem Datensatz gleich o

MELDUNG: Anzahl gelöschter Datensätze:

DELETEALL

Die Funktion löscht alle Datensätze, welche sich innerhalb der aktuellen, vorher definierten Abgrenzung befinden.

Syntax:

[OK :=] Record.DELETEALL([RunTrigger])

Bei Abfrage des Rückgabewertes, wird bei erfolgreichem Löschen der Datensätze ein TRUE zurückgegeben. Wenn der Rückgabewert nicht abgefragt wird, kommt es zu einer Fehlermeldung, wenn die Datensätze in der Tabelle nicht gefunden wurden.

Der Parameter RunTrigger ist ein Boolean-Wert. Wird dieser auf TRUE gesetzt, wird vor der eigentlichen Löschung der Datensätze der OnDelete Trigger der Tabelle ausgeführt. Standardmäßig wird dieser Trigger nicht ausgeführt.

BEISPIEL

Code	Erklärung
IF Item.GET('4711') THEN BEGIN Item.DELETE(TRUE);	//Wird der Artikel 4711 gefunden, //wird er gelöscht und dabei der //OnDELETE-Trigger der Tabelle //ausgeführt.
ItemVendor.SETRANGE("Itemno.",Item."No.");	//Die Tabelle mit den Artikel- //Lieferantenpreisen wird auf den //gelöschten Artikel abgegrenzt.

Code	Erklärung
ItemVendor.DELETEALL;	//Alle Datensätze der Tabelle //ArtikelLieferant innerhalb der //Abgrenzung werden gelöscht.
END;	

AUFGABE 16 DELETEALL

Es sollen alle Autos gelöscht werden, bei denen keine Automarke vorhanden ist. Da wir auf eine Anzeige der gelöschten Datensätze verzichten, sollen alle Datensätze innerhalb der Abgrenzung gelöscht werden (Codeunit 90320).

INIT

Der Befehl initialisiert eine Recordvariable.

Syntax:
Record.INIT

Das heißt, alle Felder einer Recordvariablen werden zurückgesetzt, und, falls vorhanden, mit denen in der Tabelle vorgegebenen Standardwerten initialisiert (Property Initvalue).

Der Inhalt aller Primärschlüsselfelder bleibt erhalten!

Normalerweise wird dieser Befehl vor dem eigentlichen Einfügen eines Datensatzes in eine Tabelle aufgerufen.

BEISPIEL

```
IF NOT Item.GET('70000') THEN
```

```
    Item.INIT;
```

Nach Aufruf dieses Befehls werden alle Felder der Recordvariablen Artikel entweder zurückgesetzt, oder der Initialisierungswert des Feldes wird eingesetzt. Ausnahme das Feld **No**. Da es das Primärschlüsselfeld der Tabelle **Item** ist, steht hier noch der Wert 70000.

Auszug aus den Feldinhalten nach dem Beispiel:

Feld	Wert
Item. "No."	= 70000
Item.Description	= ''
Item. "Price Unit Conversion"	= 0
Item. "Unit Price"	= 0,00
Item. "Costing Method"	= FIFO

AUFGABE 17 INIT

Bei allen Autos, bei denen das Kaufdatum leer ist, sollen die Felder auf Ihre Anfangswerte gesetzt werden. (Codeunit 90321)

Grenze auf Kaufdatum leer ab.

WENN der erste Datensatz gefunden wird DANN

WIEDERHOLE

Initialisiere Datensatz

Modifiziere Datensatz

..BIS Schritt zu nächstem Datensatz gleich 0

INSERT

Fügt einen Datensatz in eine Tabelle ein und gibt über den Rückgabewert Auskunft über den Erfolg des Befehls.

Syntax:

[OK :=] Record.INSERT([RunTrigger])

Wenn der Rückgabewert nicht abgefragt wurde, kommt es zu einer Fehlermeldung, wenn das System den Datensatz nicht einfügen konnte, weil er z. B. in der Tabelle bereits existiert. Unabhängig von jeder Abgrenzung identifiziert das System den einzufügenden Datensatz anhand des Inhalts der Primärschlüsselfelder.

Mit dem Parameter RunTrigger kann festgelegt werden, ob der Code des OnInsert-Tabellentriggers ausgeführt wird (TRUE/FALSE).

BEISPIEL

Code	Erklärung
Item.INIT;	//Alle Feldinhalte werden initialisiert.
Item."No." := '4711';	//Primärschlüssel der Tabelle Artikel //wird gefüllt.
Item.Description := 'Schreibtisch (natur)';	//Weiteres Feld wird gefüllt.
IF NOT Item.INSERT(TRUE) THEN	//Wenn Artikel "4711" nicht eingefügt //werden kann
MESSAGE('Der Artikel %1 existiert bereits.', Item."No.");	//wird eine Fehlermeldung angezeigt.

AUFGABE 18 INSERT

- 1 Erstellen Sie in der Codeunit 90322 einen neuen Datensatz. Erfassen Sie erst alle Felder und schreiben danach den neuen Datensatz in die Datenbank.

Füllen Sie die Felder mit folgenden Werten:

Feld	Wert
Kennzeichen/License No.	HH XY 123
Autobezeichnung/Cardescription	BMW 318i
Kaufpreis/Purchaseprice	45.000
Automarke/Brand	BMW
Kaufdatum/Purchaseprice	01.01.01
Kaufzeit/Purchasetime	12:00:00

- 2 Ändern Sie den Code ab, indem Sie die WITH-DO-Anweisung nutzen (Codeunit 90323).

COMMIT

Der Befehl beendet die aktuelle Schreibtransaktion und bestätigt sie.

Syntax:

COMMIT

Normalerweise werden nach dem Verlassen eines Objektes (Trigger, Codeunit, Report etc.) sämtliche Daten in die Datenbank geschrieben, da erst dann die Transaktion zu Ende ist. Das heißt, solange ein Trigger oder eine Codeunit nicht verlassen wird, werden die Daten nicht geschrieben. Die Daten werden zwischengespeichert.

Wird eine Fehlermeldung erzeugt, werden sämtliche Schreibtransaktionen nicht durchgeführt. Durch den Befehl COMMIT werden die Daten auch ohne Verlassen der Codeunit oder des Triggers in die Datenbank geschrieben.

BEISPIEL

Code	Erklärung
Item.SETCURRENTKEY("Shelf/Bin No.");	//Passenden Schlüssel wählen.
Item.SETRANGE("Shelf/Bin No.", 'A1');	//Alle Artikel mit der Regalnummer A1 //ausgewählt.
Item.MODIFYALL("Statistics Group", 1);	//In Artikeln, die sich innerhalb der //Abgrenzung befinden, wird das Feld //Statistikgruppe mit dem Wert 1 gefüllt.
COMMIT;	//Schreibtransaktion durchführen. Ab hier //beginnt eine neue Transaktion!
Item."Price Unit Conversion" := 2;	
Item.MODIFY;	//Diese Schreibtransaktion wird erst //ausgeführt, wenn der Trigger beendet //wird, weil kein COMMIT folgt.

Warnung

Bei diesem Befehl ist viel Vorsicht ratsam. Ein unkontrolliertes Setzen dieses Befehls kann die Datenbank bei einem Fehler in einen inkonsistenten Zustand versetzen. Wenn z. B. bei einer Buchung nach der Habenbuchung ein Commit gesetzt wird, die Sollbuchung anschließend abbricht, wäre in der Datenbank nur die Habenbuchung.

AUFGABE 19 COMMIT

Setzen Sie in der Tabelle *Fuhrpark/Vehicle* bei allen Autos das Feld **Für jeden Mitarbeiter/For every employee** auf *NEIN* (Sie können dies mit einer Funktion oder auch manuell in der Fuhrparkkarte durchführen). Im dritten Datensatz setzen Sie das Feld auf */A*.

Anschließend erstellen Sie eine Codeunit (90324), die durch alle Datensätze geht und das Feld **Für jeden Mitarbeiter/For every employee** auf */A* setzt. Nach jedem geänderten Datensatz soll ein Commit erfolgen (das heißt, der Datensatz wird in die Datenbank geschrieben). Wenn das Feld **Für jeden Mitarbeiter/For every employee** bereits auf */A* steht, soll ein Abbruch mit einer Fehlermeldung erfolgen.

Hole Datensatz

WIEDERHOLE

WENN Für jeden Mitarbeiter gleich WAHR DANN

FEHLER: Das Feld steht bereits auf Wahr

Setze Für jeden Mitarbeiter auf WAHR

Modifiziere Datensatz

Schreibe Änderung in Datenbank

BIS Schritt zu nächstem Datensatz gleich 0

Welche Datensätze wurden geändert?

Was passiert, wenn der Befehl COMMIT entfernt wird?

TESTFIELD/FIELDERROR

Bei diesen Fehlermeldungen werden die Werte des Primärschlüssels mit angezeigt. Dies erleichtert die Suche nach dem Datensatz mit dem entsprechenden Fehler.

TESTFIELD

Mit dieser Funktion kann geprüft werden, ob der Inhalt eines Feldes in einer Tabelle einem Vergleichswert entspricht.

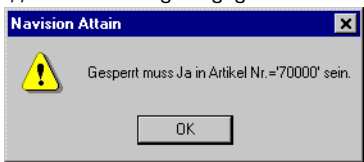
Syntax

Record.TESTFIELD(Feld, [Wert])

Der Vergleichswert muss den gleichen Datentyp haben wie das zu überprüfende Feld. Wenn der Vergleichswert nicht dem Feldinhalt entspricht, wird eine Fehlermeldung angezeigt. Die Fehlermeldung bezieht sich auf das Feld, den Datensatz und den Vorgabewert.

Falls eine Fehlermeldung ausgegeben wird, wird der aktuelle Trigger oder die Codeunit nach Anzeige der Fehlermeldung ohne weitere Abarbeitung von Befehlen verlassen.

BEISPIEL

Code	Erklärung
Item.GET('70000');	//Holt den Artikel mit der Nummer // '70000'
Item.TESTFIELD(Blocked,TRUE);	//Prüft, ob der Artikel gesperrt ist. Falls //der Artikel nicht gesperrt ist, wird eine //Fehlermeldung ausgegeben.
	
Item.TESTFIELD("Inventory Posting Group");	//Prüft, ob das Feld gefüllt ist. Wenn //nein erfolgt eine Fehlermeldung
Item.TESTFIELD("Inventory Posting Group", '');	//Prüft, ob der Inhalt des Feldes leer ist. //Wenn nicht, erfolgt eine //Fehlermeldung.

FIELDERROR

Bricht die Ausführung von Programmcode mit einer Fehlermeldung ab, die sich auf das Feld und die Tabelle bezieht.

Syntax:

Record.FIELDERROR(Feld, [Text])

Optional kann ein zusätzlicher Fehlertext der Funktion übergeben werden.

BEISPIEL 1

```
Item."No." := '';
```

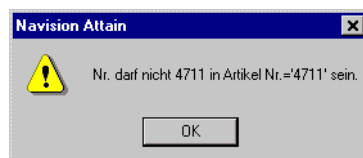
```
Item.FIELDERROR("No.");
```



BEISPIEL 2

```
Item."No." := '4711';
```

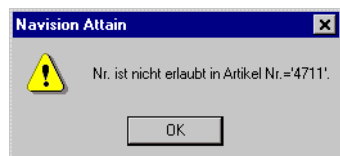
```
Item.FIELDERROR("No.");
```



BEISPIEL 3

```
Item."No." := '4711';
```

```
Item.FIELDERROR("No.", 'ist nicht erlaubt');
```



AUFGABE 20 TESTFIELD, FIELDERROR

Erstellen Sie in der Fuhrparkkarte einen Menüaufruf, der den Inhalt einiger Felder auf Richtigkeit überprüft:

"Für jeden Mitarbeiter" muss auf JA stehen


Autobezeichnung muss leer sein

"Anzahl Reparaturen" muss 0 sein

Kaufpreis muss gefüllt sein

Wenn Kaufdatum leer ist, soll Fehlermeldung erscheinen, dass das Feld Kaufdatum falsch ist, der Wert des Primärschlüsselfeldes soll mit angezeigt werden.

CALCFIELDS/CALCSUMS

 Im Folgenden werden die Befehle CALCFIELDS und CALCSUMS erläutert.

CALCFIELDS

Die Funktion aktualisiert den Inhalt der als Parameter übergebenen kalkulierten Felder des aktuellen Datensatzes.

Syntax:

[OK :=] Record.CALCFIELDS(Feld1, [Feld2],...)

Jedesmal wenn sich der Datensatz ändert, werden alle kalkulierten Felder auf 0 oder Leer gesetzt. Als Rückgabewert gibt der Befehl Auskunft über den Erfolg des Befehls (TRUE oder FALSE).

BEISPIEL

Code	Erklärung
Item.GET('70000');	//Holt den Artikel 70000 und setzt alle //kalkulierten Felder auf 0.
Item.SETRANGE(Location Filter,'BLAU');	//Begrenzt die Berechnung auf das Lager //BLAU.

Code	Erklärung
Item.CALCFIELDS(Inventory);	//Aktualisiert den Inhalt des kalkulierten //Feldes Lagerbestand unter //Berücksichtigung der vorher gesetzten //Abgrenzungen.

CALCSUMS

Die Funktion berechnet die angegebenen Summenfelder einer Tabelle unter Berücksichtigung vorher gesetzter Abgrenzungen und gibt als Rückgabewert (TRUE/FALSE) über den Erfolg des Befehls zurück.

Syntax:

[OK :=] Record.CALCSUMS (Feld1, [Feld2],...)

Wird der Rückgabewert nicht abgefragt und es kommt zu einem Fehler, bricht das System mit einer Meldung ab.

Mögliche Fehlerursachen:

- 1 Angabe eines Feldes, das kein Summenfeld ist.
- 2 Es gibt eine Abgrenzung auf ein Feld, das in keinem Schlüssel enthalten ist, dem das Summenfeld zugeordnet ist.

Code	Erklärung
ItemLedgerEntry.SETCURRENTKEY("Item No.", "Variant Code", "Drop Shipment", "Location Code", "Bin Code", "Posting Date");	//Auswahl des passenden //Schlüssels.
ItemLedgerEntry.SETRANGE("Item No.", '70000');	//Abgrenzung auf Artikel 70000.
ItemLedgerEntry.SETRANGE("Location Code", 'BLAU');	//Abgrenzung auf Lagerort BLAU.
ItemLedgerEntry.SETRANGE("Posting Date", CALCDATE('<-CY>', WORKDATE), CALCDATE('<+CY>', WORKDATE));	//Abgrenzung auf Anfang bis Ende //des aktuellen Jahres.

Code	Erklärung
ItemLedgerEntry.CALCSUMS(Quantity);	//Berechnung des Summenfeldes //auf Basis der obigen //Abgrenzungen.

Unterschied zwischen CALCFIELDS und CALCSUMS

Der Befehl CALCFIELDS wird bei Tabellen genutzt, bei denen ein FlowField definiert ist (z. B. Item.Inventory). Der Befehl ruft die Kalkulation des Feldes auf.

Der Befehl CALCSUMS wird bei Tabellen genutzt, bei denen im Schlüssel ein SumIndexField definiert ist (z. B. "Item Ledger Entry".Quantity). Der Befehl führt die Summierung des SumIndexFields durch.

AUFGABE 21 CALCFIELDS, CALCSUMS

Erstellen Sie eine neue Tabelle Kilometer (90301)

Field No.	Field Name	Data Type	Length	Description
1	Kennzeichen/License No.	Code	10	TableRelation auf Tabelle Vehicle
2	Zeilennr./Line No.	Integer		
3	Kilometer/Kilometer	Decimal		

Primärschlüssel ist Kennzeichen, Zeilennr. (SumIndexField Kilometer).

Legen Sie ein Übersichtsform an, in dem nur das Feld **Kilometer** angezeigt wird. Setzen Sie das Property AutoSplitKey auf TRUE. Diese Form wird von der Fuhrparkkarte aufgerufen. Geben Sie für die Fahrzeuge einige Daten ein.

Erstellen Sie in der Tabelle **Fuhrpark/Vehicle** ein neues Feld **Anzahl Kilometer/Number Kilometer**. Dieses Feld ist ein kalkuliertes Feld auf das Feld **Kilometer** der Tabelle **Kilometer**. Zeigen Sie dieses Feld in der Fuhrparkkarte an.

- Das Ergebnis soll nun auch in einer Codeunit errechnet werden:

Erstellen Sie die Codeunit 90325, in der Sie auf den ersten Datensatz springen.

Kalkulieren Sie das Feld Anzahl Kilometer.

Geben Sie die Meldung aus. Achtung, da in dem Befehl MESSAGE nur Texte ausgegeben werden können, muss das Ergebnis mit dem Befehl FORMAT von Zahl zu Text umgewandelt werden.

- 2 Erstellen Sie eine weitere Codeunit 90326, in der nicht die Kalkulation des Feldes **Anzahl Kilometer** ausgedruckt wird, sondern die Summierung in der Tabelle **Kilometer** durchgeführt wird.

Suchen Sie den ersten Datensatz der Tabelle **Fuhrpark/Vehicle**.

Grenzen Sie die Tabelle **Kilometer** nach dem Kennzeichen des Datensatzes aus der Tabelle **Fuhrpark/Vehicle** ab

Summieren Sie die Kilometer.

Geben Sie die Meldung aus. Achtung, da in dem Befehl MESSAGE nur Texte ausgegeben werden können, muss das Ergebnis mit dem Befehl FORMAT von Zahl zu Text umgewandelt werden.

Ist das Ergebnis bei beiden Codeunits gleich?

4.4 BEFEHLE DER KATEGORIE SYSTEM

In diesem Abschnitt werden die wichtigsten Befehle der Kategorie System erläutert. Es gibt noch mehr Befehle in dieser Kategorie, die aber im Standard sehr wenig genutzt werden. Die Befehle sind aufgeteilt in die unterschiedlichen Variablentypen.

String

Im Folgenden werden Befehle behandelt, die auf Text angewendet werden.

STRPOS

Der Befehl STRPOS gibt die Position eines Teilstrings (SubString) innerhalb eines Strings zurück.

Syntax:

Position := STRPOS(String, SubString)

BEISPIEL

Pos := STRPOS('Test', 'e');

Ergebnis:

Pos = 2

STRLEN

Mit STRLEN kann die Länge eines Strings ermittelt werden.

Syntax:

Length := STRLEN(String)

BEISPIEL

Length := STRLEN('Test');

Ergebnis:

Length = 4

INCSTR

Ist in einem String eine Zahl enthalten, wird diese mit diesem Befehl um eins hochgezählt.

Syntax:

NewString := INCSTR(String)

Dieser Befehl wird vor allem dann genutzt, wenn ein Nummernkreis (z.B. Debitoren) hochgezählt werden muss. Dabei ist String der alte String und NewString der um eins hochgezählte String.

BEISPIEL

CustomerNo := 'DEB0000122';

CustomerNo := INCSTR(CustomerNo);

Ergebnis:

CustomerNo = 'DEB0000123'

COPYSTR

Mit COPYSTR wird aus einem String ein Teilstring kopiert und in NewString gespeichert.

Syntax:

NewString := COPYSTR(String, Position [, Length])

Position gibt an, ab welcher Stelle im String kopiert wird und Length gibt die Länge des zu kopierenden Strings an. Ist Length nicht angegeben, wird der String ab Position bis zum Ende kopiert.

BEISPIEL

NewString := COPYSTR('Wörterbuch', 7);

Ergebnis:

NewString = 'buch'

AUFGABE 22 BEFEHLE KATEGORIE STRING

Der Text 'Dies ist die 1. Übung für die Schulung Navision Attain Programming II' soll geändert werden. Schreiben Sie die Codeunit 90327, die die Änderungen durchführt und das Ergebnis als Meldung ausgibt.

Erhöhen Sie die Zahl im Text um 1.

Kopieren Sie 25 Zeichen aus dem Text ab der Stelle 14.

An welcher Position steht das Wort 'für'.

Ermitteln Sie die Länge des neuen Textes.

Numeric

Im Folgenden werden die Befehle behandelt, die auf Zahlenwerte angewendet werden.

ROUND

ROUND rundet eine Zahl (Number) und gibt die neue Zahl (NewNumber) zurück.

Syntax:

NewNumber := ROUND(Number [, Precision][, Direction])

Precision gibt an, mit welcher Präzision gerundet werden wird. Soll auf 2 Dezimalstellen gerundet werden, wird 0,01 angegeben. Direction gibt an, wie gerundet werden soll:

Syntax	Erklärung
'='	Kaufmännisch (Standard)
'>'	Aufwärts
'<'	Abwärts

BEISPIEL

Number	Precision	Direction	NewNumber
1234.56789	100	=	1200
	0,1	=	1234,6
	0,001	=	1234,568
	0,001	<	1234,567
	0,001	>	1234,568

-1234.56789	100	=	-1200
	0,1	=	-1234,6
	0.001	=	-1234,568
	0,001	<	-1234,567
	0,001	>	-1234,568

ABS

Gibt den Absolutwert zurück, d. h. aus negativen Zahlen werden positive Zahlen.

Syntax:

NewNumber := ABS(Number)

BEISPIEL

NewNumber := ABS(-123,22);

Ergebnis:

NewNumber = 123,22

Date

Im Folgenden werden Befehle behandelt, die auf Datumswerte angewendet werden.

TODAY

Gibt den heutigen Tag zurück.

Syntax:

Date := TODAY

WORKDATE

Mit Workdate wird das Arbeitsdatum im C/SIDE gesetzt.

Syntax:

[WorkDate]:= WORKDATE([NewDate])

Das gleiche kann manuell über Extras – Arbeitsdatum durchgeführt werden. Wird NewDate nicht gesetzt, dann gibt dieser Befehl das aktuelle Arbeitsdatum zurück.

BEISPIEL

"The Workingdate" := WORKDATE(010101D);

CALCDATE

CALCDATE kalkuliert ausgehend von einem Datum ein neues Datum.

Syntax:

NewDate := CALCDATE(DateExpression [, Date])

DateExpression ist dabei ein String, der besagt, wie das neue Datum errechnet werden soll. Ist Date nicht angegeben, rechnet das System vom heutigen Datum aus. Der String für die DateExpression wird folgendermaßen beschrieben:

<Term> =	<Number> <Unit> <Unit> <Number> <Prefix> <Unit>
<Number>	Positive Ganzzahl
<Unit>	D WD W M Q Y (D=Tag/Day, WD=Wochentag/Weekday, W=Woche/Week, M=Monat/Month, Q=Quartal/Quarter, Y=Jahr/Year)
Prefix	C (=Laufend/Current)

Damit eine DateExpression auch in einer Multilanguageumgebung problemlos interpretiert werden kann, muss die Dateexpression in spitzen Klammern eingeschlossen werden. Z.Bsp. '<-CY>', '<+CY - 3M + 4D>'

Dies bedeutet, dass eine Kalkulation folgende Terminologie haben kann.

Inhalt	Beispiel	Bedeutung
Zahl und Einheit <Number> <Unit>	<5D>	plus fünf Tage
Einheit und Zahl <Unit> <Number>	<D5>	5. Tag

Inhalt	Beispiel	Bedeutung
Prefix und Unit	<CM>	laufender Monat

Ein Ausdruck kann auch aus mehreren DataExpressions bestehen.

BEISPIEL

Date := 010100D	
Date1 := CALCDATE(' <CQ+1M-10D>',Date);	//Laufendes Quartal + 1 Monat - 10 Tage
Date2 := CALCDATE(' <+WD2>',Date);	//Letzter Wochentag Nr.2(letzter Dienstag)
Date3 := CALCDATE(' <-CM+20D>',Date);	//1. Tag des laufender Monats + 20 Tage

Ergebnis:

Date1 = 20.04.000

Date2 = 28.12.1999

Date3 = 21.01.2000

AUFGABE 23 BEFEHLE KATEGORIE DATUM/STRING

Die Codeunit 90328 soll folgende Aufgabe lösen: Speichern Sie in der Variable Datum das Arbeitsdatum. Nun soll das Ende des Monats ermittelt werden und 12 Tage abgezogen werden. Geben Sie das neue Datum als Meldung im Format TT.MM.JJ aus.

TIME

Gibt die aktuelle Uhrzeit zurück.

Syntax:

Time := TIME

Variable

Im Folgenden werden Befehle behandelt, die auf Variablenwerte angewendet werden.

CLEAR

Dieser Befehl setzt eine Variable zurück

Syntax:

CLEAR(Variable)

CLEARALL

CLEARALL setzt jede Variable zurück.

Syntax:

CLEARALL

Beide Befehle setzen die Variablen folgendermaßen zurück:

Variablentyp	Wert
Zahlenvariable	o (Null)
Stringvariable	Leerer String
Datumsvariable	oD (Undefiniertes Datum)
Zeitvariable	oT (Undefinierte Zeit)
Boolean Variable	FALSE

BEISPIEL

Number := 10;

Text := 'test';

Date := 010100D;

Time := 120000T;

YesNo := TRUE

CLEARALL;

Ergebnis:

```
Number:= 0;  
Text := '';  
Date := 0D;  
Time:= 0T;  
YesNo := FALSE
```

EVALUATE

Mit dem Befehl Evaluate kann ein String in eine Zahl (Variable) umgewandelt werden.

Syntax:
[Ok :=] EVALUATE(Variable, String)

Wenn der Vorgang erfolgreich war, wird TRUE, bei Nichterfolg wird FALSE zurückgegeben.

BEISPIEL

```
EVALUATE(Number, '1234');  
Ergebnis:  
Number = 1234
```

```
EVALUATE(Number, 'Test');
```

Ergebnis:
Abbruch mit einer Fehlermeldung, da Test nicht in Zahl umgewandelt werden kann.
Diese Fehlermeldung kann jedoch abgefangen werden, da EVALUATE einen Rückgabewert liefert.

FORMAT

Formatiert einen Wert in einen neuen String.

Syntax:
String := FORMAT(Value [, Length][, FormatNumber | FormatString])

Value kann dabei eine Option, Integer, Decimal, Char, Text, Code, Date, Time, Boolean oder Binary Variable sein. Soll der neue String eine bestimmte Länge haben, so muss Length angegeben werden. Ist dabei Length ungleich der Stringlänge, wird der String mit Leerzeichen aufgefüllt oder

abgeschnitten. FormatNumber und FormatString gibt an, wie die Formatierung durchgeführt wird. Hierfür stehen in der Navision Attain Hilfe unter Format einige Beispiele.

BEISPIEL

Formatierung Dezimalzahl:

<Sign> <Integer Thousand> <Decimals>

Formatierung Datum:

<Closing> <Day,2>-<Month,2>-<Year>

Formatierung Zeit

<Hours24>.<Minutes,2>.<Seconds,2> <Second dec.>

4.5 REC/xREC/FIELDNO

Im Folgenden wird die Verwendung von REC, xREC und FIELDNO erklärt.

Rec

Rec ist die Standardvariable, für die einem Objekt (Form, Tabelle) zugeordnete Tabelle. Während der Programmausführung enthält "Rec" den aktuellen Datensatz. Bei der Verwendung von Feldnamen bzw. Funktionen von Rec kann die Bezeichnung Rec weggelassen werden.

xRec

xRec ist wie Rec eine weitere Standardvariable für die einem Objekt zugeordnete Tabelle. Sie unterscheidet sich dadurch, dass während der Programmausführung xRec den Inhalt des aktuellen Datensatzes vor einer Feldeingabe enthält.

CurrFieldNo

CurrFieldNo ist eine Standardvariable der Tabelle, welche während der Ausführung die Feldnummer des aktuellen Felds enthält.

4.6 OPTIONSWERTBESTIMMUNG MIT SCOPE "::"

Wegen der besseren Lesbarkeit sollte beim Zugriff auf eine Optionsvariable immer die Textform verwendet werden. Gekennzeichnet wird diese durch zwei Doppelpunkte zwischen dem Namen der Optionsvariablen und dem Optionstext.

BEISPIEL

```
OptionVar := OptionVar::Herr;
```

```
IF OptionVar = OptionVar::" " THEN
```

```
    MESSAGE('Bitte Anrede auswählen.');
```

Diese IF-Abfrage wird niemals aufgerufen, weil im Befehl davor die Variable OptionVar mit dem Wert Herr gefüllt wird.

AUFGABE 24 OPTIONSWERTE

Schreiben Sie den Trigger aus Aufgabe 4.8 des Feldes **Automark/Brand** in der Tabelle **Fuhrpar/Vehicle** so um, dass die Optionswerte nicht mit Zahlen sondern mit ':' angesprochen werden.

Kapitel 5

Programmierung nach Style Guide

Style Guide ist ein Regelwerk für die Programmierung unter Navision Attain. Unter dem Gesichtspunkt einer einfachen Lesbarkeit von Programmen, müssen bestimmte Regeln während der Programmierung eingehalten werden. Diese Regeln werden im folgenden Kapitel beschrieben.

Dieses Kapitel besteht aus den Abschnitten:

- Variablennamen/Funktionsnamen
- Einrückung IF-THEN-ELSE
- Einrückung CASE
- Einrückung für Schleifen
- Dokumentation in Objekten
- Im Trigger "Dokumentation"

5.1 VARIABLENNAMEN/FUNKTIONSNAMEN

Für die Variablennamen gilt das unter Kapitel 3 Grundlagen der Programmierung im Abschnitt Anlegen und Definieren von Variablen beschriebene Vorgehen, das im Folgenden noch einmal wiederholt wird.

Wenn eine Variable deklariert wird, muss ein Name für die Variable ausgewählt werden. Nach Möglichkeit sollte ein Name gewählt werden, der den Zweck der Variablen erkennen lässt, damit der Programmcode leichter lesbar ist. Zum Beispiel sind Tag, Name und Summe leichter zu merken als X, Y und Z.

Abgesehen von der Wahl möglichst aussagekräftiger Namen für die Variablen, sollten man einige Regeln beachten, die für Variablen gelten:

Variablennamen können bis zu 30 Zeichen lang sein.

Variablennamen müssen entweder mit einem Buchstaben oder einem Unterstrich (`_`) beginnen.

Die nachfolgenden Zeichen können Buchstaben, Unterstriche oder die Ziffern 0-9 sein. Es können auch Sonder- und Leerzeichen sein. Sobald Sonder- und Leerzeichen benutzt werden, muss die Variable in Anführungszeichen stehen.

Es sollte kein reserviertes Wort von Navision Attain für eine Variable verwendet werden.

Reservierte Wörter haben für den Compiler eine besondere Bedeutung. Diesen Wörtern ist von Navision Attain schon eine bestimmte Bedeutung zugewiesen worden. Dies kann entweder ein intern benutzter Variablennamen sein (z. B. Workdate, UserID ...), oder ein Datentyp (Dialog, Integer ...). Falls reservierte Wörter als Variablen benutzt werden, kann es zu einer Fehlermeldung während des Kompilervorganges oder bei der Programmausführung kommen.

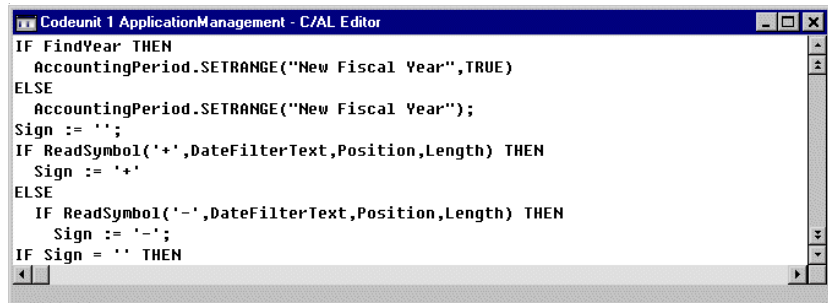
BEISPIEL FÜR VARIABLENNAMEN:

```
DisplayCode := 'Hallo Welt'; (Variablenwert = "HALLO WELT")
```

Für Funktionsnamen gelten die gleichen Regeln wie für Variablennamen.

5.2 EINRÜCKUNG IF-THEN-ELSE

Nach der IF-Anweisung wird in der nächsten Zeile um 2 Zeichen eingerückt. Das Wort ELSE wird zur besseren Lesbarkeit des Programmcodes in eine Zeile geschrieben. Danach beginnt der ELSE-Zweig in der nächsten Zeile wieder um 2 Zeichen versetzt

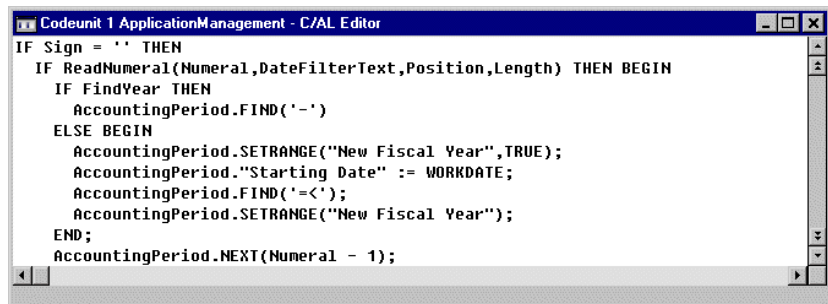


```

Codeunit 1 ApplicationManagement - C/AL Editor
IF FindYear THEN
    AccountingPeriod.SETRANGE("New Fiscal Year",TRUE)
ELSE
    AccountingPeriod.SETRANGE("New Fiscal Year");
Sign := '';
IF ReadSymbol('+',DateFilterText,Position,Length) THEN
    Sign := '+'
ELSE
    IF ReadSymbol('-',DateFilterText,Position,Length) THEN
        Sign := '-';
IF Sign = '' THEN

```

Werden mehrere IF-Anweisungen ineinander verschachtelt, so wird jede IF-Anweisung um 2 Zeichen weiter eingerückt.



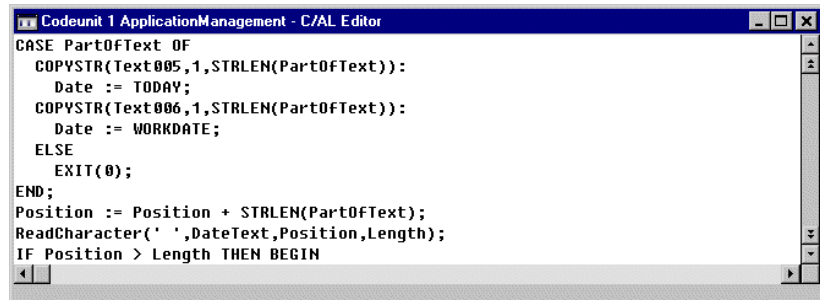
```

Codeunit 1 ApplicationManagement - C/AL Editor
IF Sign = '' THEN
    IF ReadNumeral(Numeral,DateFilterText,Position,Length) THEN BEGIN
        IF FindYear THEN
            AccountingPeriod.FIND('-')
        ELSE BEGIN
            AccountingPeriod.SETRANGE("New Fiscal Year",TRUE);
            AccountingPeriod."Starting Date" := WORKDATE;
            AccountingPeriod.FIND('<');
            AccountingPeriod.SETRANGE("New Fiscal Year");
        END;
        AccountingPeriod.NEXT(Numeral - 1);
    END;

```

5.3 EINRÜCKUNG CASE

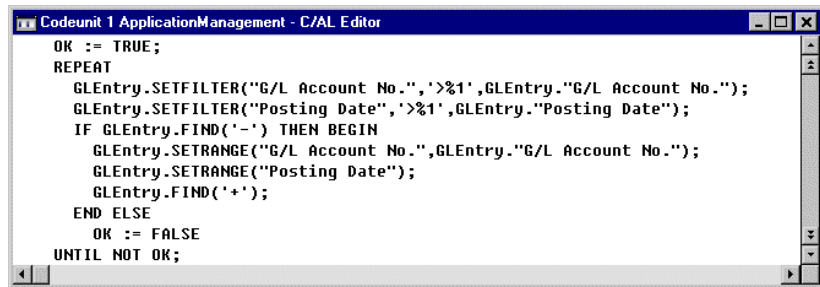
Nach der CASE-Anweisung werden die einzelnen Optionszeilen um 2 Zeichen eingerückt. Hat eine Option mehrere Befehlszeilen, werden diese weiteren Zeilen wiederum um 2 Zeichen eingerückt. Nach der ELSE-Anweisung, die gleich wie eine Option eingerückt wird, wird die folgende Zeile um 2 Zeichen eingerückt. Der Befehl End wird wieder in der gleichen Spalte wie die CASE-Anweisung begonnen.



```
Codeunit 1 ApplicationManagement - C/AL Editor
CASE PartOfText OF
    COPYSTR(Text005,1,STRLEN(PartOfText)):
        Date := TODAY;
    COPYSTR(Text006,1,STRLEN(PartOfText)):
        Date := WORKDATE;
    ELSE
        EXIT(0);
END;
Position := Position + STRLEN(PartOfText);
ReadCharacter(' ',DateText,Position,Length);
IF Position > Length THEN BEGIN
```

5.4 EINRÜCKUNG FÜR SCHLEIFEN

Schleifen werden wie IF-Anweisungen eingerückt.



```
Codeunit 1 ApplicationManagement - C/AL Editor
OK := TRUE;
REPEAT
    GLEntry.SETFILTER("G/L Account No.", '>%1', GLEntry."G/L Account No.");
    GLEntry.SETFILTER("Posting Date", '>%1', GLEntry."Posting Date");
    IF GLEntry.FIND('-') THEN BEGIN
        GLEntry.SETRANGE("G/L Account No.", GLEntry."G/L Account No.");
        GLEntry.SETRANGE("Posting Date");
        GLEntry.FIND('+');
    END ELSE
    OK := FALSE
UNTIL NOT OK;
```

5.5 DOKUMENTATION IN OBJEKTEN

Damit mehrere Programmierer an einem Projekt zusammenarbeiten können, ist es wichtig, eine ausführliche Dokumentation in den einzelnen Objekten zur Verfügung zu stellen.

Mit //

Mit den Zeichen // kann eine Zeile, oder ein Teil einer Zeile als nicht ausführbarer Code deklariert werden. D. h. Zeilen, die mit den Zeichen // gekennzeichnet sind, werden nicht kompiliert.

Mit { }

Sollen mehrere Zeilen innerhalb des Programmcodes als Dokumentation dienen, werden die Zeichen {} verwendet. Begonnen wird mit dem Zeichen {. Danach können die Dokumentationszeilen geschrieben werden. Beendet wird die Dokumentation mit dem Zeichen }. Mit dieser Methode kann auch Programmcode ausgeklammert werden, der nicht abgearbeitet werden soll.

Diese Dokumentationsmöglichkeit entspricht allerdings nicht dem Style Guide. Es kann zu Problemen führen, wenn das Objekt im Textformat ausgelesen werden soll. Im Textformat werden Trigger, Properties etc. mit { und } eingeklammert.

5.6 IM TRIGGER "DOKUMENTATION"

In jedem Objekt gibt es einen Trigger "Documentation". Dieser Trigger dient der Dokumentation des Objektes. Hier kann ausführlich das gesamte Objekt und die einzelnen Änderungen im Objekt dokumentiert werden.

AUFGABE 25 STYLE GUIDE

Überprüfen Sie in den Aufgaben 11, 12, 5.4, 6.1, 6.2., 6.3. und 6.4, ob der Style Guide eingehalten worden ist.

Kapitel 6

Lösungen

In diesem Kapitel finden Sie die Lösungen zu allen in den Schulungsunterlagen gestellten Aufgaben.

AUFGABE 1 VARIABLEN

Im Folgenden finden Sie die Lösungen für die Aufgaben 1.1. und 1.2.

Aufgabe 1.1 Variablen erstellen

Name	DataType	Subtype
DisplayText	Text	30
DisplayCode	Code	10
DisplayInteger	Integer	
DisplayDecimal	Decimal	
DisplayBoolean	Boolean	
DisplayOption	Option	
DisplayDate	Date	
DisplayTime	Time	

Aufgabe 1.2 Werte eingeben

```
DisplayText := 'Hallo Welt';  
  
DisplayCode := 'Hallo Welt';  
  
DisplayInteger := 1;  
  
DisplayDecimal := 1.12345;  
  
DisplayBoolean := TRUE;  
  
DisplayOption := 2;  
  
DisplayDate := 151201D;  
  
DisplayTime := 190000T;
```


AUFGABE 2 ARRAY

Im folgenden finden Sie die Lösungen für die Aufgaben 2.1. und 2.2.

Aufgabe 2.1 Anlegen und Anzeigen von Arrays



Name	DataType	Subtype
Month	Text	30

Property Dimension auf 12 setzen!

Aufgabe 2.2 Werte zuweisen



```
Month[1] := 'Januar';
Month[2] := 'Februar';
Month[3] := 'März';
Month[4] := 'April';
Month[5] := 'Mai';
Month[6] := 'Juni';
Month[7] := 'Juli';
Month[8] := 'August';
Month[9] := 'September';
Month[10] := 'Oktober';
Month[11] := 'November';
Month[12] := 'Dezember';
```

AUFGABE 4 BEDINGUNGEN

Im Folgenden finden Sie die Lösungen der Aufgaben 4.1 bis 4.7.

Aufgabe 4.1 Einfache IF-Bedingung

```
Purchasedate - OnValidate()  
  
//Aufgabe 4.1 Einfache IF-Bedingung  
  
IF Purchasedate > TODAY THEN  
  
Purchasedate:= TODAY;
```

Aufgabe 4.2 Mehrfache IF-Bedingung

```
Purchasedate - OnValidate()  
  
//Aufgabe 4.2. Mehrfache IF-Bedingung  
  
IF (Purchasedate = oD) AND (Purchasetime < > oT) THEN  
  
Purchasetime := oT;
```

Aufgabe 4.3 NOT-Bedingung

```
Purchasetime - OnValidate()  
  
//Aufgabe 4.3.1 NOT-Bedingung  
  
IF NOT (Purchasedate < > oD) THEN  
  
ERROR('Es muss ein Kaufdatum angegeben werden');  
  
Purchaseprice - OnValidate()  
  
//Aufgabe 4.3.2 NOT-Bedingung  
  
IF NOT "For every employee" THEN  
  
ERROR('Es darf nur ein Kaufpreis angegeben werden, wenn der Wagen für  
jeden Mitarbeiter ist');
```

Aufgabe 4.4 IF-THEN-ELSE

```

Number of repairs - OnValidate()

//Aufgabe 4.4. IF-THEN-ELSE

IF "Number of repairs" > 10 THEN

    MESSAGE('Es wurden bereits 10 Reparaturen durchgeführt. Verkaufen Sie
den Wagen')

ELSE

    MESSAGE('Sie haben die max. Anzahl an Reparaturen noch nicht erreicht');
```

Aufgabe 4.5 BEGIN..END

```

Brand - OnValidate()

//Aufgabe 4.5 BEGIN..END

IF Brand = 0 THEN BEGIN

    Cardescription := '';

    "Number of repairs" := 0;

END;
```

Aufgabe 4.6 Verschachtelte IF-Bedingung

```

OnModify

//Aufgabe 4.7. Verschachtelte IF-Bedingung

IF Cardescription <> '' THEN

    IF "Number of repairs" > 0 THEN

        MESSAGE('Eine Autobezeichnung ist ausgewählt und die Anzahl der Repa-
raturen ist größer 0')

    ELSE
```

```
MESSAGE('Eine Autobezeichnung ist ausgewählt und die Anzahl der Repa-  
raturen ist nicht größer o')
```

```
ELSE
```

```
MESSAGE('Eine Autobezeichnung ist nicht ausgewählt worden')
```

Aufgabe 4.7 CASE

```
Brand - OnValidate()
```

```
//Aufgabe 4.8 CASE-Bedingung
```

```
CASE Brand OF
```

```
1: MESSAGE('Sie haben BMW ausgewählt');
```

```
2: MESSAGE('Sie haben Mercedes ausgewählt');
```

```
3: MESSAGE('Sie haben VW ausgewählt');
```

```
4: MESSAGE('Sie haben Audi ausgewählt');
```

```
ELSE
```

```
MESSAGE('Keine Automarke ausgewählt');
```

```
END;
```

AUFGABE 5 SCHLEIFEN

Im Folgenden finden Sie die Lösungen der Aufgaben 5.1 bis 5.4.

Aufgabe 5.1 FOR-Schleife

```

OnPush()

//Aufgabe 5.1.1 FOR-NEXT-Schleife

FOR i := 1 TO 3 DO

    MESSAGE('Sie haben den Button gedrückt: %1', i);

OnPush()

//Aufgabe 5.1.2 FOR-NEXT-Schleife

FOR i := 1 TO "Number of repairs" DO BEGIN

    MESSAGE('Sie haben den Button gedrückt: %1. Zweite Variable: %2', i, y);

    y := y + 1;

END;
```

Aufgabe 5.2 REPEAT-UNTIL-Schleife

```

OnPush()

//Aufgabe 5.2 REPEAT-UNTIL-Schleife

i := 1;

IF "Number of repairs" < > 0 THEN BEGIN

    REPEAT

        Cost := Cost + 1000;

        i := i + 1;

    UNTIL i = "Number of repairs";
```

MODIFY;

END;

Aufgabe 5.3 WHILE-DO-Schleife

```
OnPush()  
  
//Aufgabe 5.3 WHILE-DO-Schleife  
  
i := 1;  
  
WHILE i <= "Number of repairs" DO BEGIN  
    Cost := Cost + 1000;  
    i := i + 1;  
END;  
  
MODIFY;
```

Aufgabe 5.4 Zusammenfassende Aufgabe

```
OnPush()  
  
//Aufgabe 5.4.1 Zusammenfassende Aufgabe  
  
Years := 1;  
  
Amount := Purchaseprice;  
  
WHILE Years <= 10 DO BEGIN  
    IF Years <= 5 THEN  
        Amount := Amount + (Amount / 100 * 3)  
    ELSE  
        Amount := Amount + (Amount / 100 * 4);  
    Years:= Years + 1;
```

```
END;

MESSAGE('Der Amount ist %1', Amount);

OnPush()

//Aufgabe 5.4.2 Zusammenfassende Aufgabe

Years := 1;

Amount := Purchaseprice;

REPEAT

    IF Years <= 5 THEN

        Amount := Amount + (Amount / 100 * 3)

    ELSE

        Amount := Amount + (Amount / 100 * 4);

    Years := Years + 1;

UNTIL Years > 10;

MESSAGE('Der Betrag ist %1', Amount);
```

AUFGABE 6 FUNKTIONEN

Im folgenden finden Sie die Lösungen der Aufgaben 6.1 bis 6.4.

Aufgabe 6.1 Ohne Parameter

```
OnPush()

//Aufgabe 6.1 Funktion ohne Parameter

InvestmentCalculation;

InvestmentCalculation()

//Aufgabe 6.1 Funktion ohne Parameter

Years := 1;

Amount := Kaufpreis;

WHILE Years <= 10 DO BEGIN

    IF Years <= 5 THEN

        Amount := Amount + (Amount / 100 * 3)

    ELSE

        Amount := Amount + (Amount / 100 * 4);

    Years := Years + 1;

END;

MESSAGE('Der Betrag ist %1', Amount);
```

Aufgabe 6.2 Mit Parameter

```
OnPush()

//Aufgabe 6.2 Funktion mit Parameter

"InvCalc With Parameters"(3,4,Purchaseprice)
```



```
InvCalc With Parameters(InterestRate1 : Decimal;InterestRate2 :
Decimal;Amount : Decimal)
```

```
//Aufgabe 6.2 Funktion mit Parameter
```

```
Years := 1;
```

```
WHILE Years <= 10 DO BEGIN
```

```
  IF Years <= 5 THEN
```

```
    Amount:= Amount + (Amount / 100 * InterestRate1)
```

```
  ELSE
```

```
    Amount := Amount + (Amount / 100 * InterestRate2);
```

```
  Years := Years + 1;
```

```
END;
```

```
MESSAGE('Der Betrag ist %1', Betrag);
```

Aufgabe 6.3 Mit Rückgabewert

```
OnPush()
```

```
//Aufgabe 6.3 Funktion mit Rückgabewert
```

```
Amount := "InvCalc With Returnvalue"(3,4,Purchaseprice);
```

```
MESSAGE('Der Betrag ist %1', Amount);
```

```
//oder
```

```
//MESSAGE('Der Betrag ist %1', "InvCalc With Returnvalue"(3,4,Purchaseprice));
```

```
InvCalc With Returnvalue(InterestRate1 : Decimal;InterestRate2 :
Decimal;Amount : Decimal) TotalAmount : Decimal
```

```
//Aufgabe 6.3 Funktion mit Rückgabewert
```

```
Years := 1;

WHILE Years <= 10 DO BEGIN

  IF Years <= 5 THEN

    Amount := Amount + (Amount / 100 * InterestRate1)

  ELSE

    Amount := Amount + (Amount / 100 * InterestRate2);

  Years := Years + 1;

END;

EXIT(Amount);
```

Aufgabe 6.4 Funktion mit VAR-Parameter

```
OnPush()

//Aufgabe 6.4 Funktion mit VAR-Parameter

"InvCalc With VAR"(3,4,Purchaseprice);

MESSAGE('Der Betrag ist %1',Purchaseprice);

InvCalc With VAR(InterestRate1 : Decimal;InterestRate2 : Decimal;VAR
Amount: Decimal)

//Aufgabe 6.4 Funktion mit VAR-Parameter

Years := 1;

WHILE Years <= 10 DO BEGIN

  IF Years <= 5 THEN

    Amount:= Amount + (Amount / 100 * InterestRate1)

  ELSE
```

```
Amount := Amount + (Amount / 100 * InterestRate2);  
Years := Years + 1;  
END;
```

Aufgabe 6.5 Wiederholung Funktionsaufruf

```
OnRun()
```

```
//Aufgabe 6.5: Wiederholung Funktionsaufruf
```

```
MESSAGE(FORMAT(ROUND(Vehiclecard."InvCalc With Return-  
value"(10,11,10000))));
```

AUFGABE 7 UNTERSCHIED LOKALE - GLOBALE VARIABLE

Lokale Variable: Years (Typ: Integer)

OnPush()

//Aufgabe 7 Funktion mit Locals

Years := 100;

"InvCalc With Locals"(3,4,Kaufpreis);

MESSAGE('Der Betrag ist %1 in %2 Jahren', Purchaseprice, Years);

InvCalc With Locals(InterestRate1 : Decimal;InterestRate2 : Decimal;VAR
Amount: Decimal)

//Aufgabe 7 Funktion mit Locals

Years := 1;

WHILE Years <= 10 DO BEGIN

IF Years <= 5 THEN

Amount := Amount + (Amount / 100 * InterestRate1)

ELSE

Amount := Amount + (Amount / 100 * InterestRate2);

Years := Years + 1;

END;

AUFGABE 8 ERROR-MESSAGE

```
Cost - OnValidate()
```

```
//Aufgabe 8 ERROR - MESSAGE
```

```
IF Cost < Purchaseprice THEN
```

```
    ERROR('%1 darf nicht kleiner als der %2 sein in der Tabelle %3', FIELDCA-  
TION(Cost), FIELDCAPTION(Purchaseprice), Rec.TABLECAPTION)
```

```
ELSE
```

```
    MESSAGE('%1 ist größer/gleich als der %2 in der Tabelle %3', FIELDCA-  
TION(Cost), FIELDCAPTION(Purchaseprice), Rec.TABLECAPTION)
```

AUFGABE 9 RECORD VARIABLE, GET UND DIALOGFENSTER

```
OnRun()

//Aufgabe 9: Record Variable, GET und Dialogfenster

Window.OPEN('Kennzeichen:  #1#####\'+
'Automarke:      #2#####\'+
'Autobezeichnung: #3#####');

REPEAT

    Window.INPUT(1,KZ);

    IF Vehicle.GET(KZ) THEN BEGIN

        Window.UPDATE(2,Vehicle.Brand);

        Window.UPDATE(3,Vehicle.Cardescription);

    END;

UNTIL o = 1;

Window.CLOSE;
```

AUFGABE 10 SETCURRENTKEY, SETRANGE, FIND, NEXT

Im folgenden finden Sie die Lösungen der Aufgaben 10.1. bis 10.5

Aufgabe 10.1 FIND

```
OnRun()  
  
//Aufgabe 10.1: FIND  
  
Vehicle.FIND('-');  
  
MESSAGE(Vehicle."License No.");
```

Aufgabe 10.2 SETRANGE und FIND

```
OnRun()  
  
//Aufgabe 10.2: SETRANGE, FIND  
  
Vehicle.SETRANGE(Brand,1);  
  
Vehicle.FIND('-');  
  
MESSAGE(Vehicle."License No.");
```

Aufgabe 10.3 SETCURRENTKEY, SETRANGE, FIND

```
OnRun()  
  
//Aufgabe 10.3: SETCURRENTKEY, SETRANGE, FIND  
  
Vehicle.SETCURRENTKEY(Brand);  
  
Vehicle.SETRANGE(Brand,1);  
  
Vehicle.FIND('-');  
  
MESSAGE(Vehicle."License No.");
```

Aufgabe 10.4 FIND mit Abfrage

```
OnRun()  
  
//Aufgabe 10.4: FIND mit Abfrage  
  
Vehicle.SETCURRENTKEY(Brand);  
  
Vehicle.SETRANGE(Brand,4);  
  
IF Vehicle.FIND('-') THEN  
  
    MESSAGE(Vehicle."License No.");
```

Aufgabe 10.5 FIND alle in Filter

```
OnRun()  
  
//Aufgabe 10.5.1 : FIND alle in Filter  
  
Vehicle.SETCURRENTKEY(Brand);  
  
Vehicle.SETRANGE(Brand,1);  
  
IF Vehicle.FIND('-') THEN  
  
    REPEAT  
  
        MESSAGE(Vehicle."License No.");  
  
    UNTIL Vehicle.NEXT = 0;  
  
//Aufgabe 10.5.2 : FIND alle in Filter  
  
Vehicle.SETCURRENTKEY(Brand);  
  
Vehicle.SETRANGE(Brand,1,2);  
  
IF Vehicle.FIND('-') THEN  
  
    REPEAT  
  
        MESSAGE(Vehicle."License No.");  
  
    UNTIL Vehicle.NEXT = 0;
```


AUFGABE 11 SETFILTER

```
OnRun()
```

```
//Aufgabe 11A: Filter auf Automarke BMW (1) und VW (3)
```

```
Vehicle.SETFILTER(Brand,'%1|%2',1,3);
```

```
IF Vehicle.FIND('-') THEN
```

```
  REPEAT
```

```
    MESSAGE(Vehicle."License No.");
```

```
  UNTIL Vehicle.NEXT = 0;
```

```
OnRun()
```

```
//Aufgabe 11B: Abgrenzen auf Automarke ungleich leer
```

```
Vehicle.SETFILTER(Brand,'<>%1',0);
```

```
IF Vehicle.FIND('-') THEN
```

```
  REPEAT
```

```
    MESSAGE(Vehicle."License No.");
```

```
  UNTIL Vehicle.NEXT = 0;
```

```
OnRun()
```

```
//Aufgabe 11C: Abgrenzen auf Automarke von BMW bis VW
```

```
Vehicle.SETFILTER(Brand,'%1..%2',1,3);
```

```
IF Vehicle.FIND('-') THEN
```

```
  REPEAT
```

```
    MESSAGE(Vehicle."License No.");
```

```
  UNTIL Vehicle.NEXT = 0;
```

```
OnRun()
```

```
//Aufgabe 11D: Abgrenzen auf Automarke ab Mercedes
```

```
Vehicle.SETFILTER(Brand,'%1..',2);
```

```
IF Vehicle.FIND('-') THEN
```

```
  REPEAT
```

```
    MESSAGE(Vehicle."License No.");
```

```
  UNTIL Vehicle.NEXT = 0;
```

```
OnRun()
```

```
//Aufgabe 11E: Abgrenzen auf Automarke ab Mercedes und ungleich Audi
```

```
Vehicle.SETFILTER(Brand,'%1..&<>%2',2,4);
```

```
IF Vehicle.FIND('-') THEN
```

```
  REPEAT
```

```
    MESSAGE(Vehicle."License No.");
```

```
  UNTIL Vehicle.NEXT = 0;
```

```
OnRun()
```

```
//Aufgabe 11F: Abgrenzen auf Autobezeichnung ungleich leer
```

```
Vehicle.SETFILTER(Cardescription,'<>%1','');
```

```
IF Vehicle.FIND('-') THEN
```

```
  REPEAT
```

```
    MESSAGE(Vehicle."License No.");
```

```
  UNTIL Vehicle.NEXT = 0;
```

```
OnRun()
```

```
//Aufgabe 11G: Abgrenzen auf Kaufpreis > 40000
```

```
Vehicle.SETFILTER(Kaufpreis,'>40000');
```

```

IF Vehicle.FIND('-') THEN

    REPEAT

        MESSAGE(Vehicle."License No.");

    UNTIL Vehicle.NEXT = o;

OnRun()

//Aufgabe 11H: Abgrenzen Kaufpreis > 20000 und < 60000
Vehicle.SETFILTER(Purchaseprice,'>20000 & <60000');

//oder
//Vehicle.SETFILTER(Purchaseprice,'20000..60000');

IF Vehicle.FIND('-') THEN

    REPEAT

        MESSAGE(Vehicle."License No.");

    UNTIL Vehicle.NEXT = o;

OnRun()

//Aufgabe 11I: Abgrenzen auf Kaufpreis < 40000 oder > 60000
Vehicle.SETFILTER(Purchaseprice,'<40000 | >60000');

IF Vehicle.FIND('-') THEN

    REPEAT

        MESSAGE(Vehicle."License No.");

    UNTIL Vehicle.NEXT = o;

```

AUFGABE 12 ZUSAMMENFASSUNG FILER

```
OnRun()

//Aufgabe 12: Zusammenfassung: Filter auf
// alle Autos der Automarken BMW und
// alle Autos mit Kaufpreis größer als 50000
Vehicle.SETCURRENTKEY(Brand);
Vehicle.SETRANGE(Brand,1);
Vehicle.SETFILTER(Purchaseprice,'>50000');
IF Vehicle.FIND('-') THEN
    REPEAT
        MESSAGE(Vehicle."License No.");
    UNTIL Vehicle.NEXT = 0;
Vehicle.RESET;
```

AUFGABE 13 MODIFY

```
OnRun()

//Aufgabe 13: Modify

Vehicle.SETCURRENTKEY(Brand);

Vehicle.SETRANGE(Brand,4);

IF Vehicle.FIND('-') THEN

  REPEAT

    IF NOT Vehicle."For every employee" THEN BEGIN

      Vehicle."For every employee" := TRUE;

      Vehicle.MODIFY;

    END;

  UNTIL Vehicle.NEXT = 0;
```

AUFGABE 14 MODIFYALL

```
OnRun()
```

```
//Aufgabe 14: Modifyall
```

```
Vehicle.SETCURRENTKEY(Brand);
```

```
Vehicle.SETRANGE(Brand,1);
```

```
Vehicle.MODIFYALL("For every employee",FALSE);
```

AUFGABE 15 DELETE

```
OnRun()

//Aufgabe 15: Delete

Vehicle.SETRANGE(Purchasedate,oD);

IF Vehicle.FIND('-') THEN

  REPEAT

    IF Vehicle.Purchaseprice = o THEN BEGIN

      i := i + 1;

      Vehicle.DELETE;

    END;

  UNTIL Vehicle.NEXT = o;

MESSAGE('Anzahl gelöschter Datensätze: %1',i);
```

AUFGABE 16 DELETEALL

```
OnRun()  
  
//Aufgabe 16: Deleteall  
  
Vehicle.SETCURRENTKEY(Brand);  
  
Vehicle.SETRANGE(Brand,o);  
  
Vehicle.DELETEALL;
```


AUFGABE 17 INIT

```
OnRun()  
  
//Aufgabe 17: Init  
  
Vehicle.SETRANGE(Purchasedate,oD);  
  
IF Vehicle.FIND('-') THEN  
  
    REPEAT  
  
        Vehicle.INIT;  
  
    UNTIL Vehicle.NEXT = o;  
  
Vehicle.MODIFY;
```

AUFGABE 18 INSERT

```
OnRun()

//Aufgabe 18.1: Insert

Vehicle.INIT;

Vehicle."License No." := 'HH XY 123';

Vehicle.Cardescription := 'BMW 318i';

Vehicle.Purchaseprice := 45000;

Vehicle.Brand := 1;

Vehicle.Purchasedate := 010101D;

Vehicle.Purchasetime := 120000T;

Vehicle.INSERT;

OnRun()

//Aufgabe 18.2: Insert mit WITH-DO

WITH Vehicle DO BEGIN

    INIT;

    "License No." := 'HH XY 123';

    Cardescription := 'BMW 318i';

    Purchaseprice := 45000;

    Brand := 1;

    Purchasedate := 010101D;

    Purchasetime := 120000T;

    INSERT;

END;
```

AUFGABE 19 COMMIT

```
OnRun()

//Aufgabe 19: Commit

IF Vehicle.FIND('-') THEN;

REPEAT

  IF Vehicle."For every employee" THEN

    ERROR('Das Feld steht bereits auf TRUE');

  Vehicle."For every employee" := TRUE;

  Vehicle.MODIFY;

  COMMIT;

UNTIL Vehicle.NEXT = o;
```

AUFGABE 20 TESTFIELD, FIELDERROR

```
OnPush()

//Aufgabe 20: Testfield, Fielderror

TESTFIELD("For every employee",TRUE);

TESTFIELD(Cardescription,"");

TESTFIELD("Number of repairs",o);

TESTFIELD(Purchaseprice);

IF Purchasedate = oD THEN

    FIELDERROR(Purchasedate,'ist falsch');
```

AUFGABE 21 CALCFIELDS, CALCSUMS

```
OnRun()
```

```
//Aufgabe 21.1: Calcfields, Calcsums
```

```
Vehicle.FIND('-');
```

```
Vehicle.CALCFIELDS(Kilometers);
```

```
MESSAGE(FORMAT(Vehicle.Kilometers));
```

```
OnRun()
```

```
//Aufgabe 21.2: Calcfields / Calcsums
```

```
Vehicle.FIND('-');
```

```
Kilometer.SETRANGE("Licenseno.",Vehicle."License No.");
```

```
Kilometer.CALCSUMS(Kilometer);
```

```
MESSAGE(FORMAT(Kilometer.Kilometer));
```

AUFGABE 22 BEFEHLE KATEGORIE STRING

```
OnRun()

//Aufgabe 22: Befehle Kategorie String

Text := 'Dies ist die 1. Übung für die Schulung Programmierung Grundlagen';

Text := INCSTR(Text);

Text := COPYSTR(Text,14,25);

Pos := STRPOS(Text,'für');

Länge := STRLEN(Text);

MESSAGE('%1\'+
        'Länge: %2\'+
        'Position: %3',Text,Länge,Pos);
```

AUFGABE 23 BEFEHLE KATEGORIE DATUM/STRING

```
OnRun()
```

```
//Aufgabe 23: Befehle Kategorie Datum/STRING
```

```
Date := WORKDATE;
```

```
Date := CALCDATE('LM+12T',Date);
```

```
MESSAGE(FORMAT(Date,8,'<Day,2>.<Month,2>.<Year,2>'));
```

AUFGABE 24 OPTIONSWERTE

```
//Aufgabe 24 Optionswerte  
  
CASE Brand OF  
  
    Brand::BMW: MESSAGE('Sie haben BMW ausgewählt');  
  
    Brand::Mercedes: MESSAGE('Sie haben Mercedes ausgewählt');  
  
    Brand::VW: MESSAGE('Sie haben VW ausgewählt');  
  
    Brand::Audi: MESSAGE('Sie haben Audi ausgewählt');  
  
ELSE  
  
    MESSAGE('Keine Automarke ausgewählt');  
  
END;
```


Navision Attain Programming II



Allgemeines

Zeiten

(09.30) 09.00 - 17.00 Uhr
12.30 - 13.30 Uhr (Pause)

Voraussetzungen

- ▼ Navision Attain Essentials

Inhalt

- ▼ Grundlagen der Entwicklungsumgebung in Navision Attain
- ▼ Grundlagen allgemeiner Programmierung
- ▼ Grundlagen der Programmierung in Navision Attain

Ziel

- ▼ Vorbereitung auf den Solution Developer I



Begriffe

C/SIDE™	Client/Server Integrated Development Environment
C/AL	Die C/SIDE Programmiersprache
C/CAPS	Configuration and Pricing System
C/SHELL	Die Währungseinheit für C/CAPS
SIFT™	Sum-Indexed Flow Technology
FlowField	Zur Laufzeit berechnete Felder
FlowFilter	Filter für FlowFields
SumIndex	Basis für FlowField, Aufsummierung von Werten



Nummernkonvention

Standardapplikation	1 - 9.999	Base Application Design Area
Länderanpassungen	10.000 - 49.999	Country Design Area
Kundenanpassungen	50.000 - 99.999	Customer Design Area - siehe Lotus Notes
Zusatzmodule	100.000 - 999.999.999	NDP Design Area

NAVISTAR
The Way to Drive

Feldnummerierung

Standardapplikation	50.000 - 99.999	Table: 1 - 49.999
Kundenanpassungen	1 - 999.999.999	Table: 50.000 - 99.999
Zusatzmodule	50.000 - 99.999	Table: 100.000 - 999.999.999

NAVISTAR
The Way to Drive

Technische Spezifikation - Objekte

Größe der Tabelle	unbeschränkt
Anzahl Datensätze	unbeschränkt
Datensatzlänge	4000 Bytes
Felder pro Datensatz	500
Tabelle-/Feldname	30 Zeichen
Anzahl Schlüssel	40
Größe pro Schlüssel	250 Bytes
Felder pro Schlüssel	PK = 20 SK = 20 - PK

NAVISTAR
The Way to Drive

Feld/Variablentypen

Typ	Bemerkung
Text	alphanumerischer String, max. 250 Zeichen lang
Code	alphanumerischer String, max. 250 Zeichen lang
Integer	Ganzzahl von -2.147.483.647 bis 2.147.483.647
Dezimal	von -10^{63} bis 10^{63}
Boolean	TRUE/FALSE, formatiert Ja/Nein
Option	Integer (0,1,2,...)
Date	1.1.0000 - 31.12.9999
Time	00:00:00 - 23:59:59.999

NAVISTION
The Way to Drive

Funktionsaufruf - Call by Value

Variable deklarieren: Rechenfunktion (Typ = Codeunit)

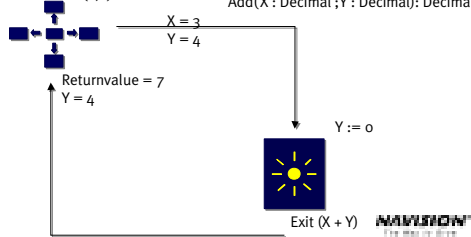
Aufruf der Funktion Add in der Codeunit Rechenfunktion:

X := 3;

Y := 4;

Z := Rechenfunktion.Add(X,Y)

Codeunit Rechenfunktion:
Add(X : Decimal ; Y : Decimal): Decimal



Funktionsaufruf - Call by Reference

Variable deklarieren: Rechenfunktion (Typ = Codeunit)

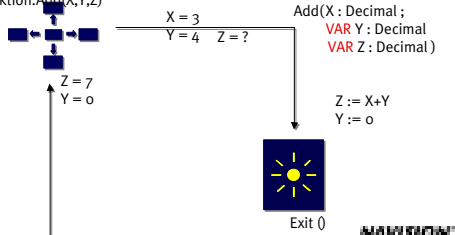
Aufruf der Funktion Add in der Codeunit Rechenfunktion:

X := 3;

Y := 4;

Z := Rechenfunktion.Add(X,Y,Z)

Codeunit Rechenfunktion:
Add(X : Decimal ;
VAR Y : Decimal
VAR Z : Decimal)



Record.Get

RECORD.GET(30.000)

Nr.	Name
10.000	Meier
20.000	Kiel
30.000	Müller
40.000	Buck
50.000	Theodor
60.000	Kalz

- Immer Primärschlüssel
- Keine Filter
- Sehr schnell

NAVISTOW®
The Master of Data

Suchen & Finden von Datensätzen

Record.SETCURRENTKEY(Name)

Nr.	Name	Nr.	Name
10.000	Müller	40.000	Buck
20.000	Hassler	20.000	Hassler
30.000	Müller	10.000	Müller
40.000	Buck	30.000	Müller
50.000	Müller	50.000	Müller
60.000	Osswald	60.000	Osswald

Record.SETRANGE(Name, 'Müller')

Nr.	Name
40.000	Buck
20.000	Hassler
10.000	Müller
30.000	Müller
50.000	Müller
60.000	Osswald

Record.FIND('-')

Record.NEXT

Record.NEXT

Record.NEXT = 0

Datensatzzeiger

Datensatzzeiger

Datensatzzeiger

Datensatzzeiger

NAVISTOW®
The Master of Data

Typischer Programmaufbau

Ziel : Alle Fahrzeuge der Marke BMW sollen für jeden Mitarbeiter = Ja erhalten.

Fuhrpark - Karteikarten sind abgelegt in drei verschiedenen Sortierungen



Um möglichst schnell die Karten der Marke BMW zu finden, wird man den Karteikasten benutzen, der nach Automarken sortiert ist:

NAVISTOW®
The Master of Data

Typischer Programmaufbau

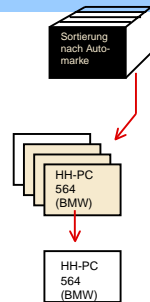
Vehicle.SETCURRENTKEY(Brand);

Jetzt werden alle Karten mit Automarke = BMW herausgenommen

Vehicle.SETRANGE(Brand,Brand::BMW);

Wenn die erste Karte gefunden wurde, dann

IF Vehicle.FIND('-') THEN



NAVISTION®
Für Ihren Erfolg

Typischer Programmaufbau

wird die Karte geändert

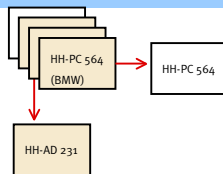
REPEAT

Vehicle."For every employee" := TRUE;

Vehicle.MODIFY;

und die nächste gesucht, bis es keine nächste Karte mehr gibt.

UNTIL Vehicle.NEXT = 0;



NAVISTION®
Für Ihren Erfolg

SETRANGE

Tabelle ohne Filter:

Personen	Anschaffung	Automarke	Anschaff.	Anschaff.
HH AD 676	40.000,00	VW	01.05.98	14.05.04
HH AD 798	65.299,00	BMW	15.05.98	9.00.00
HH PC 453	45.000,00	BMW	12.03.99	15.00.00
HH Z 122	40.500,00	Audi	01.01.00	11.00.00
HH VC 123	45.200,00	Mercedes	01.01.00	11.00.00
HH Z 122	40.500,00	Mercedes	01.01.00	11.00.00
HH CA 98	95.000,00	Audi	01.01.00	23.00.00
HH SE 100	11.000,00	VW	06.03.00	23.00.00

SETRANGE(Purchaseprice,40000,90000);

Personen	Anschaffung	Automarke	Anschaff.	Anschaff.
HH AD 676	40.000,00	VW	01.05.98	14.05.04
HH AD 798	65.299,00	BMW	15.05.98	9.00.00
HH PC 453	45.000,00	BMW	12.03.99	15.00.00
HH Z 122	40.500,00	Mercedes	01.01.00	11.00.00

SETRANGE(Brand,Brand::BMW);

Personen	Anschaffung	Automarke	Anschaff.	Anschaff.
HH AD 798	65.299,00	BMW	15.05.98	9.00.00
HH PC 453	45.000,00	BMW	12.03.99	15.00.00

NAVISTION®
Für Ihren Erfolg

SETFILTER

Tabelle ohne Filter:

Kennzeichen	Anschaffung	Automarke	Anschaff...	Anschaff...
HH KL 299	65.299,00	BMW	15.05.98	9:00:00
HH PC 453	45.000,00	BMW	12.03.99	15:00:00
HH TC 123	95.200,00	Mercedes	01.01.00	11:00:00
HH Z 122	40.500,00	Mercedes	01.01.00	11:00:00
HH PC 98	45.000,00	BMW	12.03.99	15:00:00
HH DE 100	12.500,00	VW	06.03.00	23:00:00

SETFILTER(Kaufpreis,'40000..90000|<13000&<->0');

Kennzeichen	Anschaffung	Automarke	Anschaff...	Anschaff...
HH KL 299	65.299,00	BMW	15.05.98	9:00:00
HH PC 453	45.000,00	BMW	12.03.99	15:00:00
HH Z 122	40.500,00	Mercedes	01.01.00	11:00:00
HH DE 100	12.500,00	VW	06.03.00	23:00:00

SETFILTER(Kennzeichen,'HH*');
SETFILTER(Automarke,'BMW|Mercedes');

Kennzeichen	Anschaffung	Automarke	Anschaff...	Anschaff...
HH KL 299	65.299,00	BMW	15.05.98	9:00:00
HH PC 453	45.000,00	BMW	12.03.99	15:00:00
HH Z 122	40.500,00	Mercedes	01.01.00	11:00:00

NAVISTON®
First choice for drivers

SETRANGE/SETFILTER

SETRANGE : Abgrenzung nur auf einen bestimmten Wert oder auf einen Wertebereich zwischen einer Ober- und einer Untergrenze möglich.

SETFILTER : Wesentlich komplexere Filterkombinationen möglich

NAVISTON®
First choice for drivers

Vielen Dank für Ihre Aufmerksamkeit!

NAVISTON®
First choice for drivers