

Natural für Windows

Erste Schritte

Version 6.3.8 für Windows

Februar 2010

Dieses Dokument gilt für Natural ab Version 6.3.8 für Windows.

Hierin enthaltene Beschreibungen unterliegen Änderungen und Ergänzungen, die in nachfolgenden Release Notes oder Neuausgaben bekanntgegeben werden.

Copyright © 1992-2010 Software AG, Darmstadt, Deutschland und/oder Software AG USA, Inc., Reston, VA, Vereinigte Staaten von Amerika, und/oder ihre Lizenzgeber..

Der Name Software AG, webMethods und alle Software AG Produktnamen sind entweder Warenzeichen oder eingetragene Warenzeichen der Software AG und/oder der Software AG USA, Inc und/oder ihrer Lizenzgeber. Andere hier erwähnte Unternehmens- und Produktnamen können Warenzeichen ihrer jeweiligen Eigentümer sein.

Die Nutzung dieser Software unterliegt den Lizenzbedingungen der Software AG. Diese Bedingungen sind Bestandteil der Produktdokumentation und befinden sich unter <http://documentation.softwareag.com/legal/> und/oder im Wurzelverzeichnis des lizenzierten Produkts.

Diese Software kann Teile von Drittanbieterprodukten enthalten. Die Hinweise zu den Urheberrechten und Lizenzbedingungen der Drittanbieter entnehmen Sie bitte den "License Texts, Copyright Notices and Disclaimers of Third Party Products". Dieses Dokument

ist Bestandteil der Produktdokumentation und befindet sich unter <http://documentation.softwareag.com/legal/> und/oder im Wurzelverzeichnis des lizenzierten Produkts.

Inhaltsverzeichnis

1 Erste Schritte	1
2 Über dieses Tutorial	3
Voraussetzungen	4
Über die Beispielanwendung	4
3 Grundlagen der Benutzung	7
Natural Studio aufrufen	8
Library-Workspace	9
Kommandos eingeben	10
Benutzer-Library erstellen	10
Programmiermodi	10
4 Hello World!	13
Programm erstellen	14
Programm mit RUN ausführen	15
Programmfehler korrigieren	16
Programm mit STOW speichern	18
Workspace-Optionen angeben	19
5 Datenbankzugriff	21
Demodatenbank starten	22
Programm unter einem neuen Namen speichern	23
Benötigte Daten mit einem View definieren	24
Daten aus einer Datenbank lesen	27
Bestimmte Daten aus einer Datenbank lesen	28
6 Benutzereingaben	31
Benutzereingaben ermöglichen	32
Map für die Benutzereingabe gestalten	35
Map aus dem Programm aufrufen	46
Immer einen Endnamen benutzen	48
7 Verarbeitungsschleifen und Labels	51
Wiederholte Benutzung erlauben	52
Meldung für nicht gefundene Informationen anzeigen	54
8 Interne Subroutinen	57
Interne Subroutine definieren	58
Interne Subroutine ausführen	59
9 Verarbeitungsregeln und Helproutinen	63
Verarbeitungsregel definieren	64
Helproutine definieren	67
10 Local Data Areas	69
Local Data Area erstellen	70
Datenfelder definieren	71
Datenfelder aus einem DDM importieren	74
Local Data Area aus dem Programm aufrufen	75
11 Global Data Areas	79
Global Data Area mit Hilfe einer bestehenden Local Data Area erstellen	80








Local Data Area anpassen	82
Global Data Area aus dem Programm aufrufen	83
12 Externe Subroutinen	85
Externe Subroutine erstellen	86
Externe Subroutine aus dem Programm aufrufen	87
13 Subprogramme	91
Local Data Area ändern	92
Parameter Data Area mit Hilfe einer bestehenden Local Data Area erstellen	94
Eine zweite Local Data Area mit einem anderen View erstellen	95
Subprogramm erstellen	96
Subprogramm aus dem Programm aufrufen	97






1 Erste Schritte

Dieses Tutorial bietet eine einfache und kurze Einleitung in Natural Studio (welches die Entwicklungsumgebung für Natural ist) und in die Programmierung mit Natural.



Wichtig: Es ist wichtig, dass Sie die folgenden Themen in der unten angegebenen Reihenfolge lesen und dass Sie alle darin enthaltenen Übungen in derselben Reihenfolge wie in diesem Tutorial abarbeiten. Wenn Sie eine Übung überspringen, könnte dies zu Problemen führen.

	Über dieses Tutorial	Voraussetzungen und was Sie im Laufe dieses Tutorials lernen.
	Grundlagen der Benutzung	Wie Sie Natural Studio aufrufen. Informationen zum Library-Workspace und den verschiedenen Möglichkeiten der Befehlseingabe. Wie Sie die Library erstellen, mit der Sie in diesem Tutorial arbeiten werden. Informationen zu den Programmiermodi von Natural und über den Modus, der für dieses Tutorial erforderlich ist.
	Hello World!	Wie Sie Ihr erstes kurzes Programm erstellen, ausführen und speichern. Wie Sie den Inhalt der aktuellen Library anzeigen. Informationen über einige Optionen, von denen Ihr Workspace beeinflusst wird.
	Datenbankzugriff	Informationen zur Demodatenbank. Wie Sie bestimmte Informationen aus einer Datenbank lesen und ausgeben.
	Benutzereingaben	Wie Sie den Benutzer zur Eingabe von Informationen auffordern und wie Sie eine Map für die Benutzereingabe entwerfen. Wie Sie gewährleisten, dass immer derselbe Wert benutzt wird (hier: ein Endname), auch wenn er nicht vom Benutzer eingegeben wurde.
	Verarbeitungsschleifen und Labels	Wie Sie wiederholte Verarbeitungsschleifen und Labels für verschiedene Schleifen definieren. Wie eine Meldung angezeigt wird, wenn eine bestimmte Information (hier: der vom Benutzer eingegebene Startname) nicht gefunden wird.
	Interne Subroutinen	Wie Sie eine interne Subroutine (das heißt: eine Subroutine, die direkt ins Programm geschrieben wird) definieren und aufrufen.

 Verarbeitungsregeln und Helproutinen	Wie Sie eine Verarbeitungsregel (hier: eine Meldung, die erscheint, wenn der Benutzer keinen Startnamen angibt) und eine Helproutine (hier: ein Hilfetext für das Feld, in dem der Benutzer einen Startnamen angeben muss) definieren.
 Local Data Areas	Wie Sie die Felddefinitionen aus dem Programm in eine Local Data Area außerhalb des Programms verlagern.
 Global Data Areas	Wie Sie eine Global Data Area definieren, die von mehreren Programmen oder Routinen benutzt werden kann.
 Externe Subroutinen	Wie Sie eine externe Subroutine definieren und aufrufen (das heißt: eine Subroutine, die als separates Objekt außerhalb des Programms gespeichert ist).
 Subprogramme	Wie Sie eine Parameter Data Area für ein Subprogramm definieren. Wie Sie ein Subprogramm definieren und aufrufen.

2 Über dieses Tutorial

- Voraussetzungen 4
- Über die Beispielanwendung 4

Wenn Sie Natural zum ersten Mal benutzen, sollten Sie dieses Tutorial durcharbeiten, um Grundkenntnisse über bestimmte Merkmale der Natural-Programmierungsumgebung zu erhalten.

Es wird vorausgesetzt, dass Sie über Grundkenntnisse von Microsoft Windows verfügen.

Das Layout der in diesem Tutorial beschriebenen Beispielbildschirme und das hier beschriebene Verhalten von Natural kann von Ihren Resultaten abweichen. So kann zum Beispiel die Kommandozeile oder Meldungszeile an einer anderen Position im Bildschirm erscheinen oder die Ausführung eines Natural-Kommandos kann durch Sicherheitseinstellungen geschützt sein. Die Standardeinstellungen in Ihrer Umgebung sind abhängig von den Systemparametern, die von Ihrem Natural-Administrator gesetzt wurden.

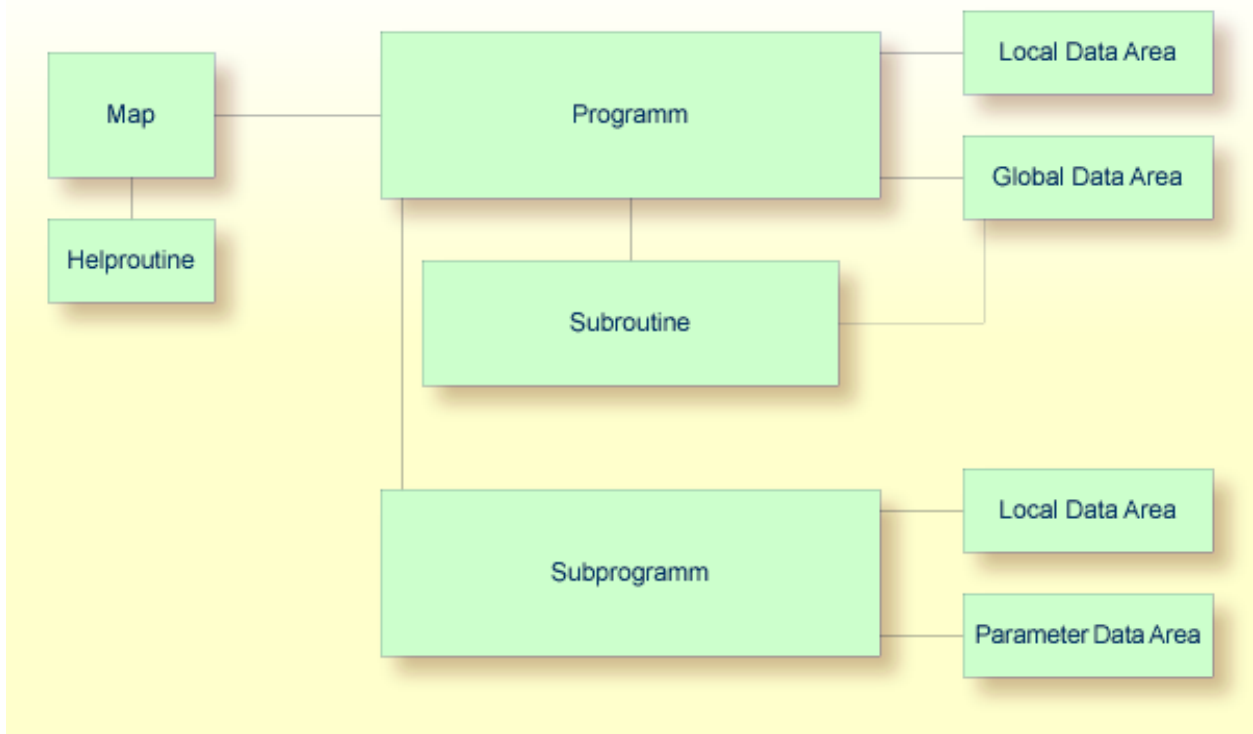
Voraussetzungen


Damit alle Schritte dieses Tutorials ausgeführt werden können, muss die Demodatenbank SAG-DEMO-DB installiert sein (entweder lokal unter Windows oder unter UNIX) und sie muss aktiv sein. Diese Datenbank wird mit Adabas installiert; sie wird nicht automatisch mit Natural installiert.


Über die Beispielanwendung

Dieses Tutorial zeigt, wie Sie eine Anwendung als eine Gruppe von Modulen strukturieren können. Es ist nicht dazu gedacht, Ihnen zu zeigen, wie eine Anwendung aufgebaut werden sollte.

Zuerst werden Sie Ihr erstes kurzes "Hello World"-Programm schreiben. Danach schreiben Sie ein Programm, mit dem Mitarbeiter- (Employees-) Informationen aus einer Datenbank gelesen und angezeigt werden. Der Benutzer wird aufgefordert einen Start- und einen Endnamen für die Ausgabe anzugeben. Sie werden Ihr Programm Schritt für Schritt erweitern und dabei bestimmte Programmteile in externe Module verschieben. Wenn Sie mit allen Schritten dieses Tutorials fertig sind, dann hat Ihre Anwendung die folgende Struktur:



 **Anmerkung:** Dieses Tutorial beschreibt, wie Sie eine Map erstellen. Dieser Objekttyp wird normalerweise nur in einer zeichenorientierten Umgebung (zum Beispiel bei einem Großrechner) benutzt. Bei einer grafischen Benutzeroberfläche würden Sie normalerweise einen Dialog erstellen - dies ist jedoch nicht Bestandteil dieses Tutorials.

 **Tipp:** Wenn Sie den Programmcode nicht selbst eintippen möchten, können Sie die Codebeispiele aus diesem Tutorial in die Natural-Editoren kopieren, zum Beispiel mit den Standard-Windows-Tastenkombinationen STRG+C (kopieren) und STRG+V (einfügen).

Sie können nun mit der ersten Übung beginnen: *Grundlagen der Benutzung*.

3 Grundlagen der Benutzung

- Natural Studio aufrufen 8
- Library-Workspace 9
- Kommandos eingeben 10
- Benutzer-Library erstellen 10
- Programmiermodi 10

Natural Studio aufrufen

Nachdem Natural installiert wurde, erscheint der entsprechende Ordner automatisch im **Programme**-Ordner des **Start**-Menüs. Er enthält alle Verknüpfungen für Natural, einschließlich der Verknüpfung für Natural Studio, welches die Entwicklungsumgebung für Natural ist. Wenn es bei der Installation angegeben wurde, finden Sie auch einige Verknüpfungen auch auf Ihrem Windows-Desktop.

▶ Natural Studio aufrufen

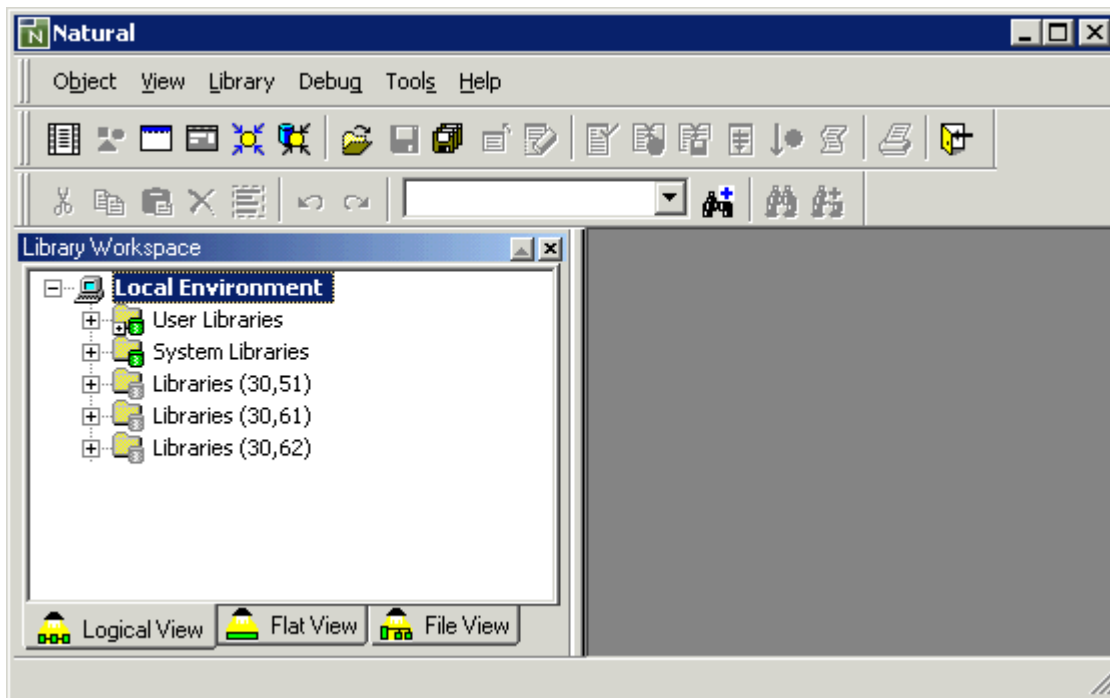
- Wählen Sie im Windows **Start**-Menü den Eintrag **Programme > Software AG Natural n.n > Natural**.

Oder:

Benutzen Sie die folgende Verknüpfung auf Ihrem Windows-Desktop (steht nur zur Verfügung, wenn dies bei der Installation angegeben wurde).



Das Fenster von Natural Studio erscheint.



Wenn Sie Natural Studio zum ersten Mal starten, wird nur die lokale Umgebung mit dem Library-Workspace angezeigt.

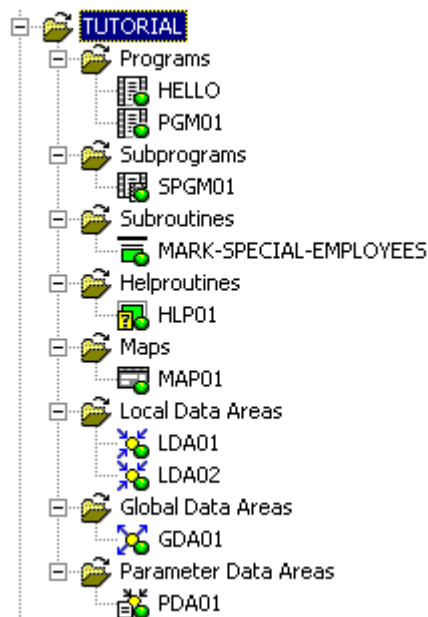
Library-Workspace

Alle Natural-Objekte, die zum Erstellen einer Anwendung erforderlich sind, werden in Natural-Libraries und Natural-Systemdateien gespeichert. Es gibt eine Systemdatei für Systemprogramme (FNAT) und eine Systemdatei für Benutzerprogramme (FUSER).

Natural unterscheidet zwischen System-Libraries und Benutzer-Libraries. Die System-Libraries, die mit den Buchstaben "SYS" beginnen, sind ausschließlich Zwecken der Software AG vorbehalten. Eine Benutzer-Library enthält alle Benutzerobjekte (beispielsweise Programme und Maps), aus denen eine Anwendung besteht. Der Name einer Benutzer-Library darf nicht mit den Buchstaben "SYS" beginnen.

Im Library-Workspace gibt es verschiedene Views. Die Übungen in diesem Tutorial gehen davon aus, dass Sie im Logical-View arbeiten. Im Logical-View werden unterschiedliche Knoten für die Libraries angeboten. Die Objekte in einer Library sind in verschiedene Knoten unterteilt, und zwar nach ihren Natural-Objekttypen.

Wenn Sie mit allen Übungen in diesem Tutorial fertig sind, dann enthält der Knoten für Ihre Benutzer-Library **TUTORIAL** die folgenden Ordner und Objekte:



Kommandos eingeben

Wie auch bei anderen Windows-Anwendungen können Sie bei Natural Studio die meisten Natural-Kommandos auf mehrere Arten eingeben: Sie können sie aus der Menüleiste oder einem Kontextmenü auswählen oder Sie können sie über eine Symbolschaltfläche oder Tastenkombination ausführen. In diesem Tutorial werden nicht alle Alternativen zur Auswahl desselben Kommandos aufgeführt, sondern nur die am häufigsten benutzten Methoden (in den meisten Fällen sind dies Kontextmenüs und Symbolschaltflächen).

Um ein Kontextmenü aufzurufen (zum Beispiel für ein Objekt im Library-Workspace), müssen Sie das Objekt markieren und dann mit der rechten Maustaste klicken oder **UMSCHALT+F10** drücken.

Einige Menüs oder Symbolleisten werden nur in einem bestimmten Kontext angezeigt. So wird zum Beispiel das Menü **Program** nur dann in der Menüleiste angezeigt, wenn Sie mit dem Programmmeditor arbeiten und das Programmmeditorfenster aktiv ist.

Benutzer-Library erstellen

Sie werden jetzt eine Benutzer-Library mit dem Namen `TUTORIAL` erstellen. Diese Library wird alle Natural-Objekte enthalten, die Sie im Laufe dieses Tutorials erstellen werden.

▶ Benutzer-Library erstellen

- 1 Markieren Sie im Logical-View den Knoten **User Libraries**. Er befindet sich direkt unter dem obersten Knoten **Local Environment**.
- 2 Wählen Sie aus dem Kontextmenü den Befehl **New**.

Eine neue Library mit dem Standardnamen **USRNEW** wird jetzt im Baum angezeigt.

- 3 Geben Sie "TUTORIAL" als Name für diese Library ein und drücken Sie **EINGABE**.

Programmiermodi

Bei Natural gibt es zwei Programmiermodi:

■ Structured Mode

Der Structured Mode ist für komplexe Anwendungen gedacht, bei denen es auf eine klare und sinnvoll gegliederte Programmstruktur ankommt. Grundsätzlich empfiehlt es sich, ausschließlich im Structured Mode zu arbeiten.

■ Reporting Mode

Der Reporting Mode eignet sich nur für die Erstellung einfacher Reports und Programme, die keine komplexe Daten- und Programmstruktur erfordern.



Wichtig: Dieses Tutorial setzt voraus, dass der Structured Mode aktiv ist. Falls Sie versuchen Ihr Programm im Reporting Mode auszuführen, dann werden Fehler durch END-IF, END-READ und END-REPEAT verursacht.

▶ Programmiermodus überprüfen

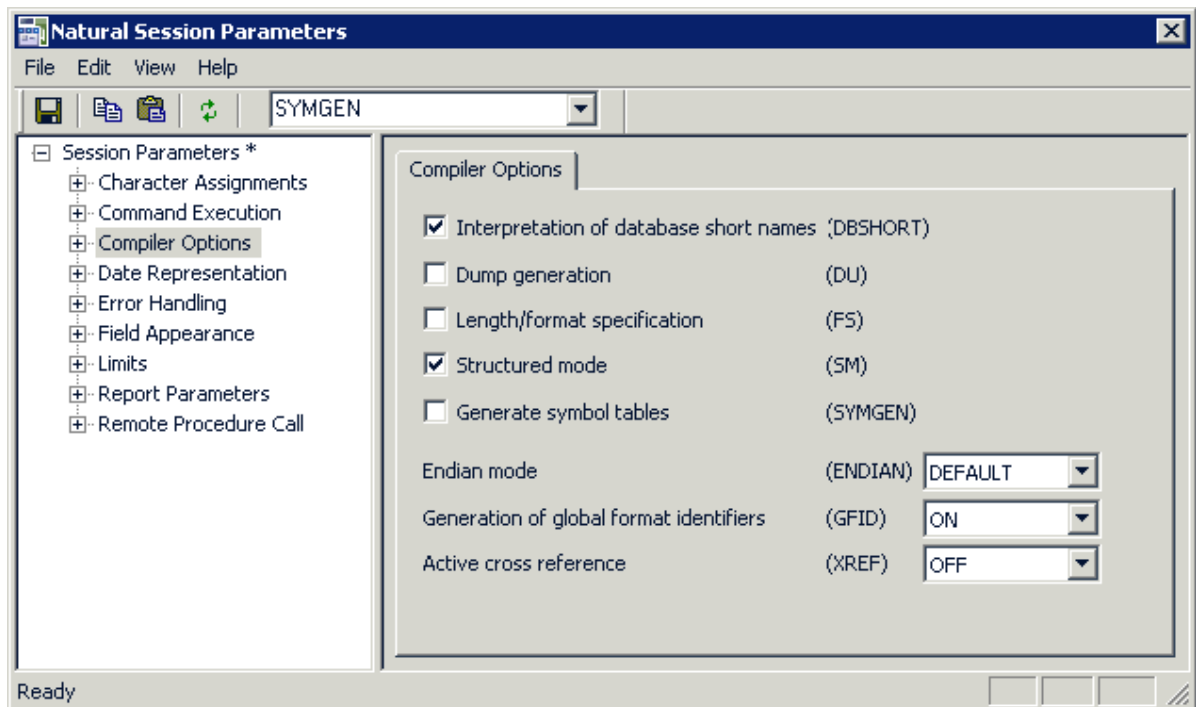
- 1 Wählen Sie aus dem Menü **Tools** den Befehl **Session Parameters**.

Das Dialogfeld **Natural Session Parameters** erscheint.

Bei der Installation von Natural definiert der Natural-Administrator Standardwerte für die Parameter, die dann für alle Natural-Benutzer gelten. Wenn Sie diese Werte ändern, dann gelten sie nur für die aktuelle Session.

- 2 Markieren Sie links im Baum den Eintrag **Compiler Options**.

Die Compiler-Optionen werden nun auf der rechten Seite angezeigt.



Wenn das Kontrollkästchen **Structured mode** bereits markiert, sind keine weiteren Schritte erforderlich und Sie können das Dialogfeld schließen. Wenn es nicht markiert ist, fahren Sie fort wie unten beschrieben.

- 3 Aktivieren Sie das Kontrollkästchen **Structured mode**.
- 4 Wählen Sie aus dem Menü **File** den Befehl **Save**.
- 5 Schließen Sie das Dialogfeld.

Sie können nun mit Ihrem ersten Programm beginnen: *Hello World!*

4 Hello World!

▪ Programm erstellen	14
▪ Programm mit RUN ausführen	15
▪ Programmfehler korrigieren	16
▪ Programm mit STOW speichern	18
▪ Workspace-Optionen angeben	19

Programm erstellen

Sie werden nun Ihr erstes kurzes Programm schreiben, das "Hello World!" anzeigt. Es wird in der Library gespeichert, die Sie zuvor erstellt haben.

▶ Neues Programm erstellen

- 1 Markieren Sie im Library-Workspace die Library `TUTORIAL`.
- 2 Wählen Sie aus dem Kontextmenü den Befehl **New Source > Program**.

Oder:

Wählen Sie die folgende Schaltfläche in der Symbolleiste:



Der Programmmeditor erscheint. Er ist zurzeit leer.

- 3 Geben Sie den folgenden Code im Programmmeditor ein:

```
* The "Hello world!" example in Natural.  
*  
DISPLAY "Hello world!"  
END /* End of program
```

Eine Kommentarzeile beginnt mit einem Stern (*), dem mindestens ein Leerzeichen oder ein zweiter Stern folgt. Wenn Sie das Leerzeichen oder den zweiten Stern vergessen, geht Natural davon aus, dass Sie eine Systemvariable angegeben haben; dies verursacht einen Fehler.

Wenn Sie Leerzeilen in Ihr Programm einfügen möchten, dann sollten Sie sie als Kommentarzeilen definieren. Dies ist hilfreich, wenn Sie Ihr Programm auf einer anderen Plattform (Windows, Großrechner, UNIX oder OpenVMS) weiterbearbeiten wollen. Mit der Großrechnerversion von Natural werden beispielsweise alle leeren Zeilen automatisch gelöscht, sobald Sie `EINGABE` drücken (Standardeinstellung).

Sie können auch einen Kommentar am Ende einer Statement-Zeile einfügen. In diesem Fall muss der Kommentar mit einem Schrägstrich beginnen, dem ein Stern folgt (/).

Der Text, der in der Ausgabe erscheinen soll, wird mit dem Statement `DISPLAY` angegeben. Er steht in Anführungszeichen.

Das Statement `END` markiert das physische Ende des Natural-Programms. Jedes Programm muss mit `END` abgeschlossen werden.

Programm mit RUN ausführen

Das Systemkommando `RUN` ruft automatisch das Systemkommando `CHECK` auf, welches das Programm auf Fehler überprüft. Wenn kein Fehler gefunden wird, wird das Programm im Flug kompiliert und ausgeführt.



Anmerkungen:

1. Systemkommandos werden auch bei der Großrechnerversion von Natural benutzt. Unter Windows werden sie mit den entsprechenden Befehlen im Menü **Object** ausgeführt.
2. `CHECK` steht ebenfalls als separates Systemkommando im Menü **Object** zur Verfügung.
3. Es gibt bei Natural auch das Systemkommando `EXECUTE`, welches die mit dem Systemkommando `STOW` gespeicherte Version Ihres Programms benutzt (dieses Kommando wird später in diesem Tutorial erklärt). Im Gegensatz zu `EXECUTE` werden bei `RUN` immer die letzten Programmänderungen berücksichtigt.

▶ Programm mit RUN ausführen

- 1 Wählen Sie aus dem Menü **Object** den Befehl **Run**.

Oder:

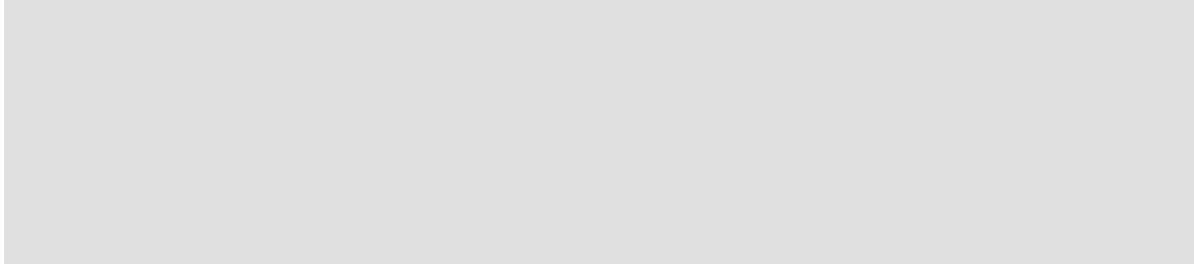
Wählen Sie die folgende Schaltfläche in der Symbolleiste:



Wenn Ihr Code syntaktisch korrekt ist, wird der von Ihnen definierte Text ausgegeben.

```
Page      1                               09-06-30  12:07:25
```

```
Hello world!
```



- 2 Drücken Sie `EINGABE` um zum Programmeditor zurückzukehren.

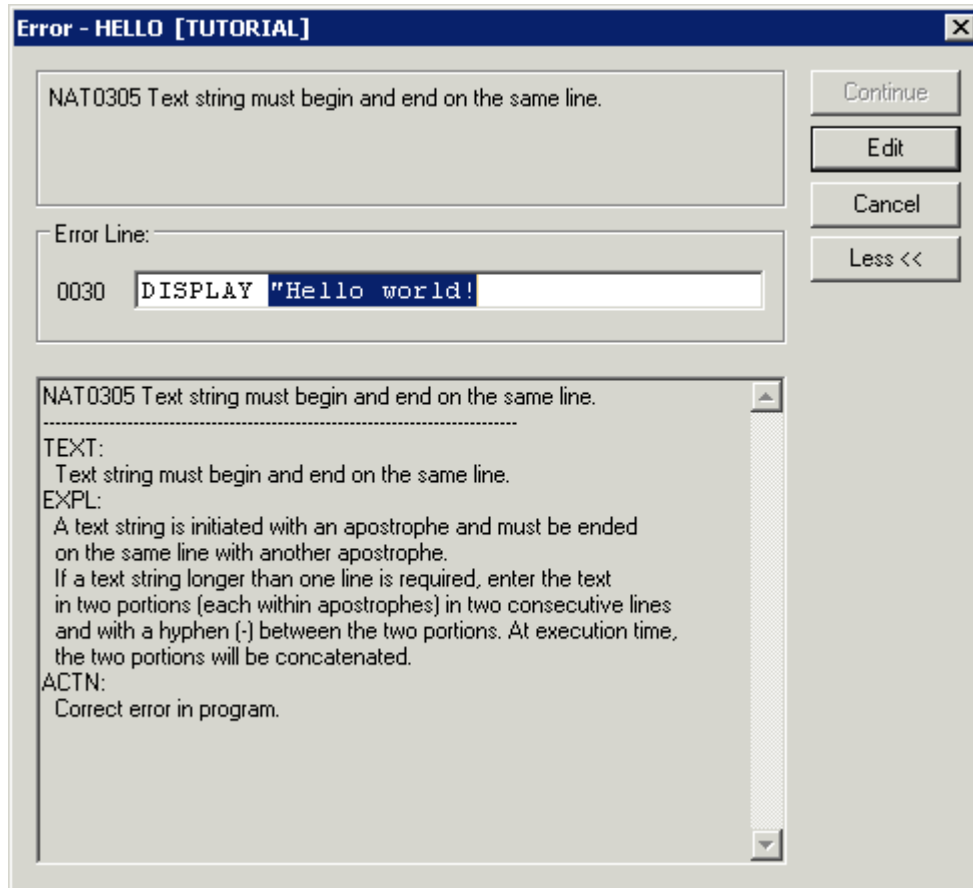
Programmfehler korrigieren

Sie werden jetzt einen Fehler in Ihr Programm einbauen und das Programm anschließend noch einmal mit `RUN` ausführen.

▶ Fehler korrigieren

- 1 Löschen Sie das zweite Anführungszeichen in der Zeile, die das `DISPLAY`-Statement enthält.
- 2 Führen Sie das Programm noch einmal wie oben beschrieben aus.

Wenn ein Fehler gefunden wird, erscheint ein Dialogfeld mit Hinweisen zu diesem Fehler.



- 3 Korrigieren Sie den Fehler im Dialogfeld. Das heißt: fügen Sie das fehlende Anführungszeichen wieder am Ende der Zeile ein.
- 4 Wählen Sie die Befehlsschaltfläche **Continue**, um den nächsten Fehler zu finden.

In diesem Fall werden keine weiteren Fehler gefunden und die Ausgabe wird angezeigt.

- 5 Drücken Sie EINGABE, um zum Programmeditor zurückzukehren.




Anmerkung: Statt der Befehlsschaltfläche **Continue** können Sie auch die Befehlsschaltfläche **Edit** wählen. Das Dialogfeld wird in diesem Fall geschlossen und Sie können den Fehler direkt im Programmeditor korrigieren.

Programm mit STOW speichern

Wenn Sie ein Programm mit STOW speichern, wird es kompiliert und der Sourcecode sowie das generierte Programm werden in der Systemdatei abgelegt.

Wie beim Systemkommando RUN wird auch beim Systemkommando STOW automatisch das Systemkommando CHECK aufgerufen. Ein Programm kann nur dann mit STOW gespeichert werden, wenn es syntaktisch korrekt ist.

 **Anmerkung:** Wenn Sie Ihre Programmänderungen speichern wollen, obwohl das Programm syntaktische Fehler enthält (wenn Sie Ihre Arbeit zum Beispiel bis zum nächsten Arbeitstag unterbrechen wollen), dann können Sie das Systemkommando SAVE benutzen. Dieses Systemkommando wird mit dem entsprechenden Befehl im Menü **Object** ausgeführt.

▶ Programm mit STOW speichern

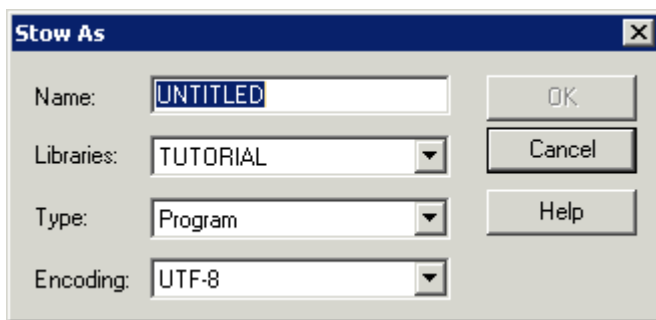
- 1 Wählen Sie aus dem Menü **Object** den Befehl **Stow**.

Oder:

Wählen Sie die folgende Schaltfläche in der Symbolleiste:



Da Ihr Programm noch nicht gespeichert wurde, erscheint das Dialogfeld **Stow As**.



Der Name der aktuellen Library wird automatisch im entsprechenden Dropdown-Listenfeld angezeigt.

- 2 Geben Sie den Namen "HELLO" im Textfeld **Name** ein.
- 3 Wählen Sie die Befehlsschaltfläche **OK**.

Es erscheint die Nachricht, dass das Speichern erfolgreich war. Diese Nachricht wird entweder in der Statuszeile oder in einem Dialogfeld angezeigt. Dies ist abhängig von der Einstellung einer bestimmten Workspace-Option (siehe unten).

Im Library-Workspace wird nun ein neuer Knoten mit dem Namen **Programs** angezeigt, und zwar als Unterknoten von **TUTORIAL**. Dieser Unterknoten enthält das soeben gespeicherte Programm.



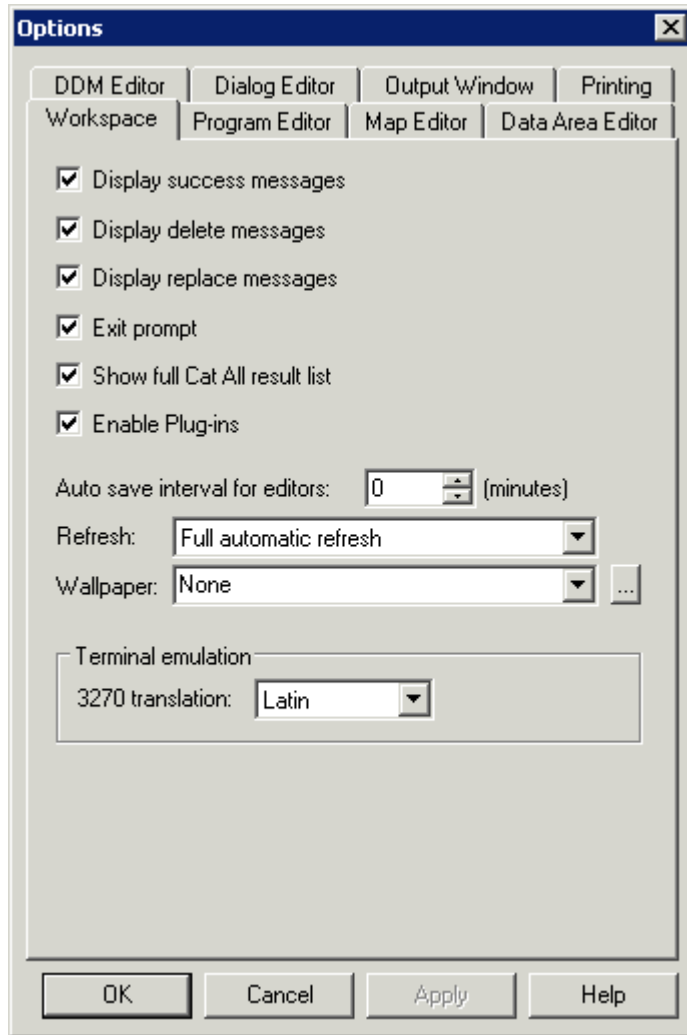
Der grüne Ball im Programm-Symbol bedeutet, dass es für dieses Objekt sowohl Sourcecode als auch ein generiertes Programm gibt.

Workspace-Optionen angeben

Sie werden jetzt die Einstellungen Ihrer Workspace-Optionen überprüfen.

▶ Workspace-Optionen prüfen

- 1 Wählen Sie aus dem Menü **Tools** den Befehl **Options**.
- 2 Zeigen Sie im daraufhin erscheinenden **Options**-Dialogfeld die **Workspace**-Seite an.



- 3 Wenn Erfolgsmeldungen in einem Dialogfeld angezeigt werden sollen, dann muss das Kontrollkästchen **Display success messages** markiert sein.



Anmerkung: Die Anzeige der Zeilennummern im Programmeditor wird durch eine Option auf der Seite **Program Editor** gesteuert.

- 4 Wählen Sie die Befehlsschaltfläche **OK**, um die Änderungen zu speichern und das Dialogfeld zu schließen.

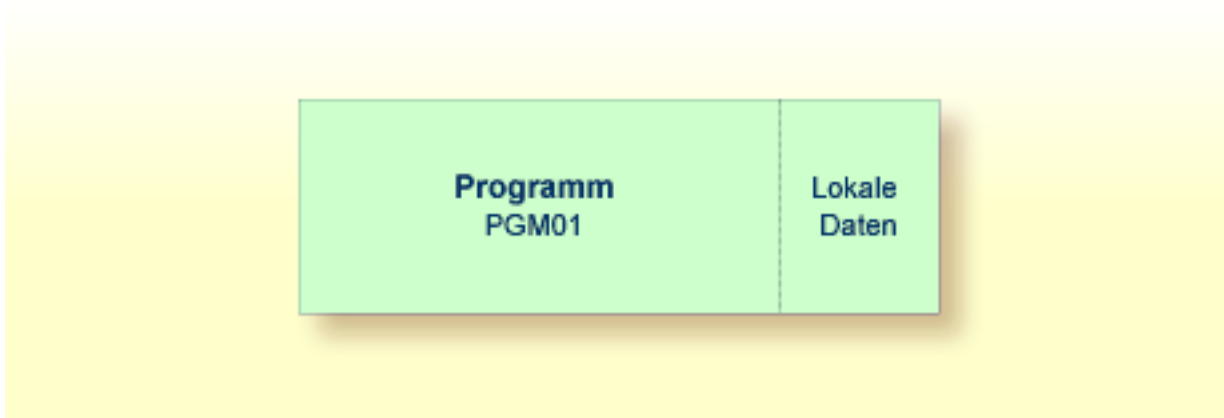
Sie können nun mit den nächsten Übungen fortfahren: [Datenbankzugriff](#).

5 Datenbankzugriff

- Demodatenbank starten 22
- Programm unter einem neuen Namen speichern 23
- Benötigte Daten mit einem View definieren 24
- Daten aus einer Datenbank lesen 27
- Bestimmte Daten aus einer Datenbank lesen 28

Sie werden nun ein kurzes Programm schreiben, mit dem bestimmte Daten aus einer Datenbankdatei gelesen und angezeigt werden.

Wenn Sie mit den Übungen in diesem Kapitel fertig sind, wird Ihre Beispielanwendung aus einem einzigen Modul bestehen (die Datenfelder, die vom Programm benutzt werden, sind direkt in diesem Programm definiert).



Demodatenbank starten

Die Demodatenbank `SAG-DEMO-DB` wird nicht automatisch gestartet. Bevor Sie mit den Übungen in diesem Kapitel fortfahren können, müssen Sie sich davon überzeugen, dass die Datenbank gestartet wurde. Andernfalls funktionieren die Beispiele nicht.

Die folgende Beschreibung gilt wenn Adabas lokal unter Windows installiert wurde. Wenn Sie mit der UNIX-Version arbeiten möchten und die Datenbank nicht aktiv ist, müssen Sie Ihren Administrator bitten, sie zu starten.

▶ Demodatenbank starten

- 1 Wählen Sie aus dem Menü **Start** den Befehl **Programme > Adabas n.n > DBA Workbench**.

Der Status der Demodatenbank wird in der daraufhin erscheinenden Datenbankliste angezeigt. Wenn der Status "Active" angezeigt wird, sind keine weiteren Schritte erforderlich und Sie können das Anwendungsfenster der DBA Workbench schließen.

Wenn der Status "Active" nicht angezeigt wird, müssen Sie wie unten beschrieben fortfahren.

- 2 Markieren Sie **SAG-DEMO-DB** in der Datenbankliste.
- 3 Wählen Sie aus dem Menü **Database** den Befehl **Start**.

Ein Dialogfeld erscheint mit dem Hinweis, dass die Datenbank gestartet wurde.

- 4 Wählen Sie die Befehlsschaltfläche **OK**, um das Dialogfeld zu schließen.
- 5 Schließen Sie das Anwendungsfenster der DBA Workbench.

Programm unter einem neuen Namen speichern

Sie werden jetzt ein neues Programm erstellen, das im weiteren Verlauf dieses Tutorials benutzt wird. Es wird erstellt, indem Sie Ihr "Hello World"-Programm unter einem neuen Namen speichern.

▶ Programm unter einem neuen Namen speichern

- 1 Wählen Sie aus dem Menü **Object** den Befehl **Save As**.



Tip: Achten Sie darauf, dass der Programmeditor aktiv ist. Andernfalls steht der oben genannte Befehl nicht zur Verfügung.

Das Dialogfeld **Save As** erscheint.

- 2 Geben Sie "PGM01" als neuen Namen für das Programm ein.
- 3 Wählen Sie die Befehlsschaltfläche **OK**.

Der neue Name wird jetzt in der Titelleiste des Programmeditors angezeigt.

Im Library-Workspace wird das neue Programm im Knoten **Programs** angezeigt. Da das Programm noch nicht mit `STOW` gespeichert wurde, enthält das Programmsymbol keinen grünen Ball.



- 4 Löschen Sie den gesamten Code im Programmeditor (zum Beispiel indem Sie den gesamten Text mit `STRG+A` markieren und dann mit der `ENTF`-Taste löschen - dies ist Windows-Standardfunktionalität).

Benötigte Daten mit einem View definieren

Die Datenbankdatei und die Felder, die in Ihrem Programm benutzt werden sollen, müssen am Anfang des Programms zwischen `DEFINE DATA` und `END-DEFINE` angegeben werden.

Damit Natural auf den Inhalt einer Datenbankdatei zugreifen kann, wird eine logische Definition der physischen Datenbankdatei benötigt. Eine solche logische Dateidefinition wird „Data Definition Module“ (DDM) genannt. Das DDM enthält Informationen über die einzelnen Felder der Datei. DDMs werden in der Regel vom Natural-Administrator definiert.

Damit die Datenbankfelder in einem Natural-Programm benutzt werden können, müssen Sie die Felder des DDMs in einem View angeben. Beispiel-DDMs stehen in der System-Library `SYSEXDDM` zur Verfügung. In diesem Tutorial wird das DDM für die Datenbankdatei `EMPLOYEES` benutzt.

Sie können die Felder, einschließlich Format- und Längenangaben, aus dem DDM in den Programmmeditor importieren.

▶ `DEFINE DATA`-Block angeben

- Geben Sie den folgenden Code im Programmmeditor ein:

```
DEFINE DATA  
LOCAL  
  
END-DEFINE  
*  
END
```



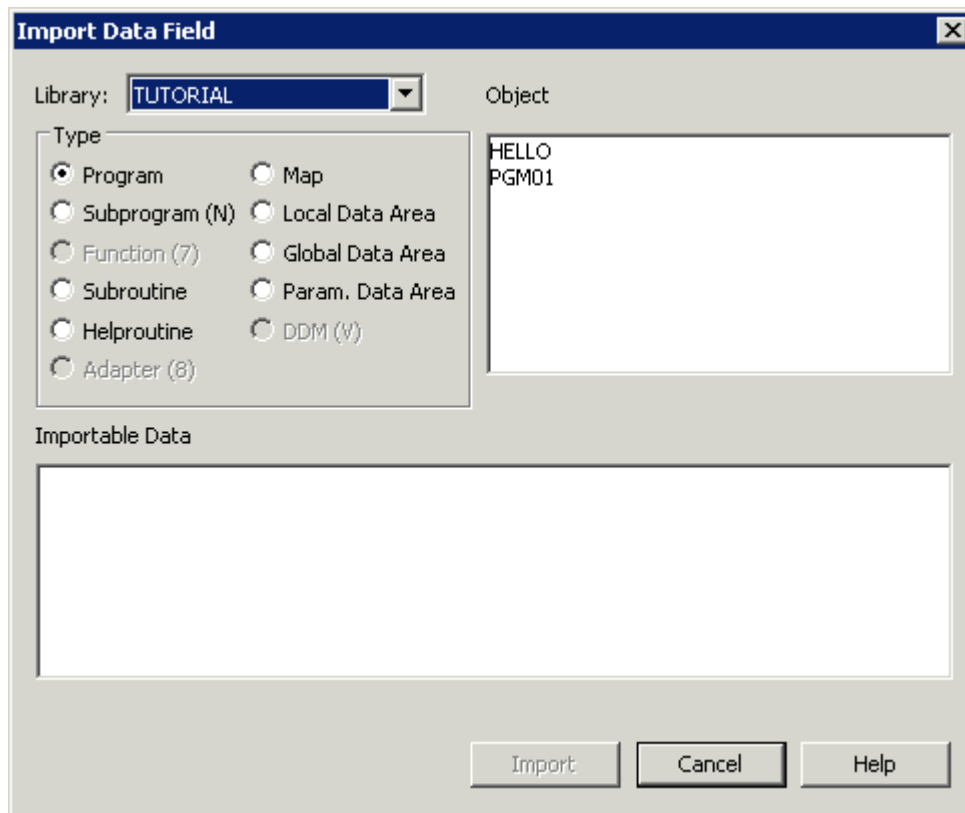
Tip: Die Windows-Version von Natural unterscheidet nicht zwischen Groß- und Kleinschreibung. Wenn Sie jedoch mit der Großrechner-Version von Natural arbeiten, dann werden Schlüsselwörter (Keywords) und Bezeichnungen (Identifiers) immer in Großbuchstaben angegeben; Textkonstanten können Kleinbuchstaben enthalten. Wenn Sie Ihr Programm also auch auf dem Großrechner editieren möchten, wird empfohlen, dass Sie den Programmcode so eingeben, wie Sie es auch auf einem Großrechner tun würden.

`LOCAL` bedeutet, dass die Variablen, die Sie im nächsten Schritt definieren werden, lokale Variablen sind, die nur in diesem Programm zur Verfügung stehen.

▶ Datenfelder aus einem DDM importieren

- 1 Stellen Sie den Cursor in die Zeile unter `LOCAL`.
- 2 Wählen Sie aus dem Menü **Program** den Befehl **Import**.

Das Dialogfeld **Import Data Field** erscheint.



- 3 Markieren Sie im Dropdown-Listenfeld **Library** den Eintrag **SYSEXDDM**.

Wenn das Optionsfeld **DDM** markiert ist, werden alle definierten DDMs im Listenfeld **Object** angezeigt.

- 4 Markieren Sie das Beispiel-DDM mit dem Namen `EMPLOYEES`.

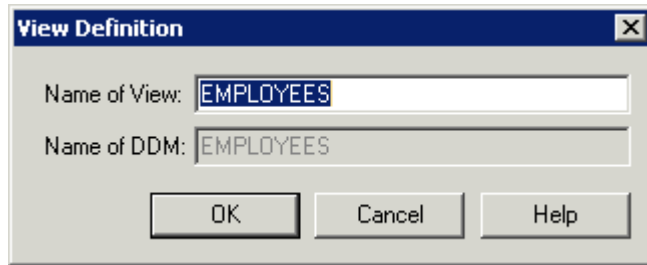
Die importierbaren Daten werden jetzt unten im Dialogfeld angezeigt.

- 5 Drücken Sie `STRG` und markieren Sie die folgenden Felder:

FULL - NAME
 NAME
 DEPT
 LEAVE - DATA
 LEAVE - DUE

- 6 Wählen Sie die Befehlsschaltfläche **Import**.

Das Dialogfeld **View Definition** erscheint.



Standardmäßig wird der Name des DDMs als View-Name vorgeschlagen. Sie können aber auch einen beliebigen anderen Namen angeben.

- 7 Geben Sie "EMPLOYEES-VIEW" als View-Name ein.
- 8 Wählen Sie die Befehlsschaltfläche **OK**.

Die Befehlsschaltfläche **Cancel** im Dialogfeld **Import Data Field** trägt nun den Namen **Quit**.

- 9 Wählen Sie die Befehlsschaltfläche **Quit**, um das Dialogfeld **Import Data Field** zu schließen.

Der folgende Code wurde im Programmeditor eingefügt:

```

1 EMPLOYEES-VIEW VIEW OF EMPLOYEES
2 FULL-NAME
3 NAME (A20)
2 DEPT (A6)
2 LEAVE-DATA
3 LEAVE-DUE (N2)

```

Die erste Zeile enthält den Namen des Views und den Namen der Datenbankdatei, aus der die Felder genommen werden. Dies wird immer auf der ersten Ebene (Level 1) definiert. Der Level wird am Anfang der Zeile definiert. Die Namen der Datenbankfelder aus dem DDM sind auf den Levels 2 und 3 definiert.

Levels werden im Zusammenhang mit Feldgruppierungen benutzt. Felder mit einem Level von 2 oder höher werden als Teil der direkt davor stehenden Gruppe angesehen, die auf einem niedrigeren Level steht. Die Definition einer Gruppe ermöglicht das Referenzieren von mehreren Feldern (dies kann auch nur ein einziges Feld sein), indem man den Gruppennamen benutzt. Dies ist eine komfortable und effiziente Methode zum Referenzieren von mehreren aufeinander folgenden Feldern.

Format und Länge jedes Feldes sind in Klammern angegeben. "A" steht für alphanumerisch und "N" steht für numerisch.

Daten aus einer Datenbank lesen

Jetzt, nachdem Sie alle erforderlichen Daten definiert haben, werden Sie eine `READ`-Schleife in Ihr Programm einfügen. Hiermit werden die Daten mit Hilfe des definierten Views aus der Datenbankdatei gelesen. Mit jeder Schleife wird ein Mitarbeiter aus der Datenbankdatei gelesen. Name, Abteilung (Department) und die restlichen Urlaubstage (Leave Due) für diesen Mitarbeiter werden angezeigt. Die Daten werden so lange gelesen, bis alle Mitarbeiter angezeigt wurden.



Anmerkung: Es kann passieren, dass eine Fehlermeldung erscheint, die besagt, dass die Transaktion abgebrochen wurde (transaction aborted). Dies passiert in der Regel dann, wenn das Adabas-Zeitlimit für Nichtaktivität überschritten wurde. Wenn ein solcher Fehler auftritt, sollten Sie Ihre letzte Aktion einfach wiederholen (geben Sie zum Beispiel das Kommando `RUN` noch einmal ein).

▶ Daten aus einer Datenbank lesen

- 1 Geben Sie folgendes unter `END-DEFINE` ein:

```
READ EMPLOYEES-VIEW BY NAME
*
  DISPLAY NAME 3X DEPT 3X LEAVE-DUE
*
END-READ
```

`BY NAME` bedeutet, dass die aus der Datenbank gelesenen Daten alphabetisch nach den Namen sortiert werden sollen.

Mit dem `DISPLAY`-Statement wird die Ausgabe im Spaltenformat angeordnet. Für jedes angegebene Feld wird eine Spalte erzeugt und jede Spalte enthält eine Überschrift. `3X` bedeutet, dass 3 Leerzeichen zwischen den Spalten eingefügt werden sollen.

- 2 Führen Sie das Programm mit `RUN` aus.

Die folgende Ausgabe wird angezeigt.

```
Page          1                                05-05-18  16:06:49
      NAME                DEPARTMENT          LEAVE
                        CODE                DUE
-----
ABELLAN                PROD04                20
ACHIESON                COMPO2                25
ADAM                    VENT59                19
ADKINSON                TECH10                38
ADKINSON                TECH10                18
```

ADKINSON	TECH05	17
ADKINSON	MGMT10	28
ADKINSON	TECH10	26
ADKINSON	SALE30	36
ADKINSON	SALE20	37
ADKINSON	SALE20	30
AECKERLE	SALE47	31
AFANASSIEV	MGMT30	26
AFANASSIEV	TECH10	35
AHL	MARK09	30
AKROYD	COMP03	20
ALEMAN	FINA03	20

Das `DISPLAY`-Statement sorgt dafür, dass die Spaltenüberschriften (die aus dem DDM genommen werden) unterstrichen sind und dass sich zwischen der Unterstreichung und den Daten eine Leerzeile befindet. Jede Spalte hat die Breite, die im `DEFINE DATA`-Block definiert wurde (das heißt: die Breite, die im View definiert ist).

Der Titel oben auf jeder Seite (mit Seitennummer, Datum und Uhrzeit) wird ebenfalls vom `DISPLAY`-Statement erzeugt.

- 3 Drücken Sie wiederholt `EINGABE`, um alle Seiten nacheinander anzuzeigen.

Sie kehren zum Programmreditor zurück, nachdem alle Mitarbeiter angezeigt wurden.



Tipp: Wenn Sie zum Programmreditor zurückkehren möchten, bevor alle Mitarbeiter angezeigt wurden, drücken Sie `ESC`.

Bestimmte Daten aus einer Datenbank lesen

Da die Ausgabe im Moment sehr lang ist, werden Sie sie nun eingrenzen. Es sollen nur noch die Daten für den Namenbereich angezeigt werden, der mit "Adkinson" beginnt und mit "Bennett" endet. Diese Namen sind in der Demodatenbank definiert.

► Ausgabe auf einen Namensbereich eingrenzen

- 1 Bevor Sie neue Variablen benutzen können, müssen Sie sie definieren. Geben Sie deshalb Folgendes unter LOCAL ein:

```
1 #NAME-START      (A20) INIT <"ADKINSON">
1 #NAME-END        (A20) INIT <"BENNETT">
```

Dies sind Benutzervariablen; sie sind nicht in der Demodatenbank definiert. Das Rautenzeichen (#) am Anfang des Namens wird dazu benutzt, die Benutzervariablen von den Feldern in der Demodatenbank zu unterscheiden; es ist jedoch nicht zwingend notwendig, dieses Zeichen zu verwenden.

INIT definiert den Vorgabewert für ein Feld. Der Vorgabewert muss in spitzen Klammern und Anführungszeichen angegeben werden.

- 2 Geben Sie Folgendes unter dem READ-Statement ein:

```
STARTING FROM #NAME-START
ENDING AT #NAME-END
```

Ihr Programm sollte nun folgendermaßen aussehen:

```
DEFINE DATA
LOCAL
  1 #NAME-START      (A20) INIT <"ADKINSON">
  1 #NAME-END        (A20) INIT <"BENNETT">
  1 EMPLOYEES-VIEW VIEW OF EMPLOYEES
    2 FULL-NAME
      3 NAME (A20)
    2 DEPT (A6)
    2 LEAVE-DATA
      3 LEAVE-DUE (N2)
END-DEFINE
*
READ EMPLOYEES-VIEW BY NAME
  STARTING FROM #NAME-START
  ENDING AT #NAME-END
*
  DISPLAY NAME 3X DEPT 3X LEAVE-DUE
*
END-READ
*
END
```

- 3 Führen Sie das Programm mit RUN aus.

Die Ausgabe wird angezeigt. Wenn Sie wiederholt EINGABE drücken, werden Sie bemerken, dass Sie nach wenigen Seiten zum Programmeditor zurückkehren (d.h. nachdem die Daten für den letzten Mitarbeiter mit dem Namen Bennett angezeigt wurden).

- 4 Speichern Sie das Programm mit STOW.

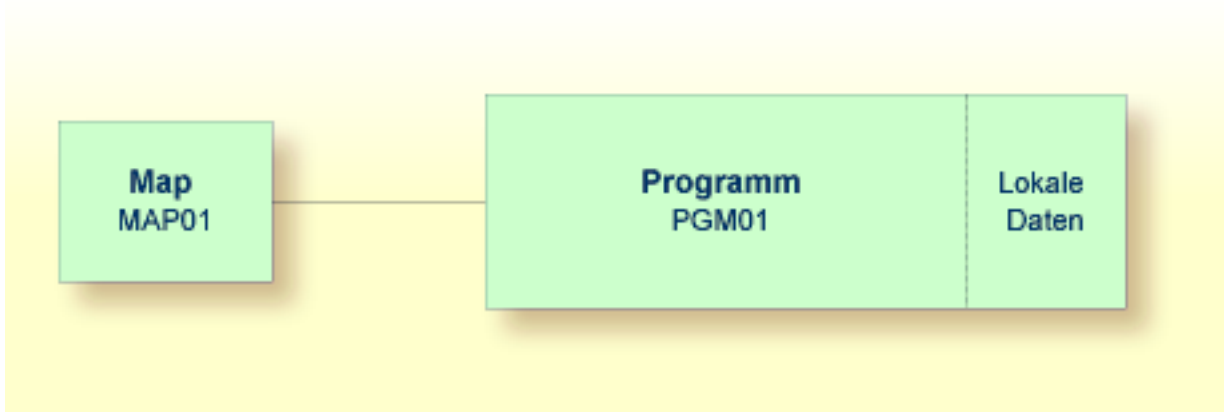
Sie können nun mit den nächsten Übungen fortfahren: *Benutzereingaben*.

6 Benutzereingaben

- Benutzereingaben ermöglichen 32
- Map für die Benutzereingabe gestalten 35
- Map aus dem Programm aufrufen 46
- Immer einen Endnamen benutzen 48

Sie lernen nun, wie Sie einen Benutzer zur Eingabe von Daten auffordern (d.h. der Benutzer soll einen Start- und Endnamen für die Ausgabe eingeben).

Wenn Sie mit den Übungen in diesem Kapitel fertig sind, wird Ihre Beispielanwendung aus den folgenden Modulen bestehen:



Benutzereingaben ermöglichen

Sie werden Ihr Programm jetzt so verändern, dass Eingabefelder für die Start- und Endnamen in der Ausgabe angezeigt werden. Hierzu benutzen Sie das `INPUT`-Statement.

► Eingabefelder definieren

- 1 Geben Sie Folgendes unter `END-DEFINE` ein:

```
INPUT (AD=MT)
  "Start:" #NAME-START /
  "End:  " #NAME-END
```

Der Session-Parameter `AD` steht für „Attribute Definition“ (Attributdefinition). Sein Wert `"M"` steht für „Modifiable output field“ (modifizierbares Ausgabefeld) und der Wert `"T"` steht für „translate lowercase to uppercase“ (Übersetzen von Kleinbuchstaben in Großbuchstaben).

Der Wert `"M"` in `AD=MT` bedeutet, dass die mit `INIT` definierten Vorgabewerte (d.h.: `"ADKINSON"` und `"BENNETT"`) in den Eingabefeldern angezeigt werden. Der Benutzer kann andere Werte eingeben. Wenn Sie den Wert `"M"` weglassen, bleiben die Eingabefelder leer, obwohl Vorgabewerte definiert wurden.

Der Wert `"T"` in `AD=MT` bedeutet, dass jede Eingabe, die in Kleinbuchstaben gemacht wurde, vor der weiteren Verarbeitung in Großbuchstaben übersetzt wird. Dies ist wichtig, weil die Namen in der Demodatenbank komplett in Großbuchstaben definiert wurden. Wenn Sie den

Wert "T" weglassen, müssen Sie alle Namen komplett in Großbuchstaben angeben. Andernfalls wird der angegebene Name nicht gefunden.

"Start:" und "End:" sind Textfelder (Bezeichnungen). Sie werden in Anführungszeichen eingegeben.

#NAME-START und #NAME-END sind Datenfelder (Eingabefelder), in denen der Benutzer den gewünschten Start- und Endnamen eingeben kann.

Der Schrägstrich (/) bedeutet, dass die nachfolgenden Felder in einer neuen Zeile angezeigt werden sollen.

Ihr Programm sollte nun folgendermaßen aussehen:

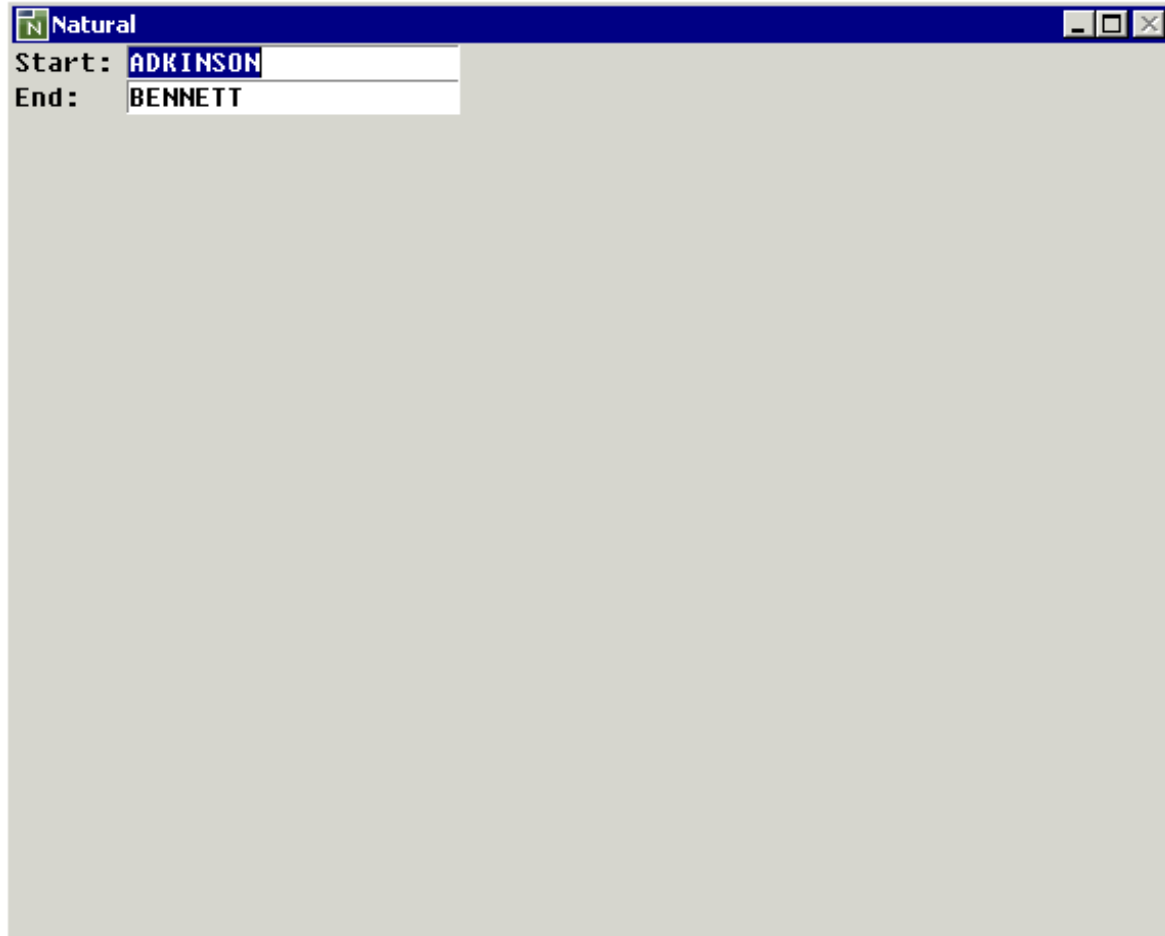
```

DEFINE DATA
LOCAL
  1 #NAME-START      (A20) INIT <"ADKINSON">
  1 #NAME-END        (A20) INIT <"BENNETT">
  1 EMPLOYEES-VIEW VIEW OF EMPLOYEES
    2 FULL-NAME
      3 NAME (A20)
    2 DEPT (A6)
    2 LEAVE-DATA
      3 LEAVE-DUE (N2)
END-DEFINE
*
INPUT (AD=MT)
  "Start:" #NAME-START /
  "End:  " #NAME-END
*
READ EMPLOYEES-VIEW BY NAME
  STARTING FROM #NAME-START
  ENDING AT #NAME-END
*
  DISPLAY NAME 3X DEPT 3X LEAVE-DUE
*
END-READ
*
END

```

- 2 Führen Sie das Programm mit RUN aus.

Die Ausgabe enthält die Felder, die Sie gerade definiert haben.



- 3 Behalten Sie die Vorgabewerte bei und drücken Sie EINGABE.

Die Liste der Mitarbeiter wird nun angezeigt.

- 4 Drücken Sie wiederholt EINGABE bis Sie wieder im Programmeditor sind, oder drücken Sie ESC.
- 5 Speichern Sie das Programm mit STOW.

Map für die Benutzereingabe gestalten

Sie lernen jetzt eine weitere Möglichkeit kennen, den Benutzer zur Eingabe aufzufordern. Sie werden mit dem Map-Editor eine Map erstellen, die dieselben Felder enthält, die Sie vorher in Ihrem Programm definiert haben. Eine Map ist ein separates Objekt; sie wird benutzt, um das Layout der Benutzeroberfläche von der Geschäftslogik der Anwendung zu trennen.

Die Map, die Sie jetzt erstellen werden, wird folgendermaßen aussehen:



The screenshot shows a map editor interface. At the top left, there is a header 'XXXXXXXXXX' and at the top right, 'TT:TT:TT'. The main area contains two data fields and two text fields. The first data field is labeled 'Start:' and the second is labeled 'End:'. Both data fields contain 'XXXXXXXXXXXXXXXXXXXXXXXXX'. The text fields are empty and are positioned to the right of the data fields.

Die erste Zeile der Map enthält Systemvariablen für das aktuelle Datum und die aktuelle Uhrzeit. Es gibt zwei Datenfelder (Eingabefelder), in denen der Benutzer einen Startnamen und einen Endnamen eingeben kann. Vor den Datenfeldern befinden sich Textfelder (Bezeichnungen).

Für die oben gezeigte Map sind die folgenden Schritte erforderlich:

- Map erstellen
- Textfelder definieren
- Bezeichnungen für die Textfelder definieren
- Datenfelder definieren
- Namen und Attribute für Datenfelder definieren
- Systemvariablen einfügen
- Map testen

- Map mit STOW speichern

Map erstellen

Sie werden jetzt den Map-Editor aufrufen, mit dem Sie das Layout Ihrer Map gestalten.

Lassen Sie den Programmmeditor im Hintergrund offen.

▶ Map erstellen

- 1 Markieren Sie im Library-Workspace die Library, die auch Ihr Programm enthält (d.h. markieren Sie den Knoten **TUTORIAL**).
- 2 Wählen Sie aus dem Kontextmenü den Befehl **New Source > Map**.

Oder:

Wählen Sie die folgende Schaltfläche in der Symbolleiste:



Ein leeres Map-Editor-Fenster erscheint.

Textfelder definieren

Sie werden jetzt zwei Textfelder (auch Konstanten oder Bezeichnungen genannt) in der Map definieren.

▶ Textfelder definieren

- 1 Wählen Sie aus dem Menü **Insert** den Befehl **Text Constant**.



Anmerkung: Wenn der Map-Editor aktiv ist, werden andere Menüs in der Menüleiste angezeigt. Jetzt werden die Menüs **Insert**, **Field** und **Map** angezeigt (statt des Menüs **Programs**, welches zu sehen war als der Programmmeditor aktiv war).

Oder:

Wählen Sie die folgende Schaltfläche in der Symbolleiste:



Anmerkung: Standardmäßig wird die Symbolleiste, in der diese Schaltfläche enthalten ist, in vertikaler Ausrichtung rechts neben dem Map-Editor-Fenster angezeigt. Als der Programmierer aktiv war, wurde dort eine andere Symbolleiste angezeigt.

- 2 Gehen Sie mit der Maus an die Stelle im Map-Editor-Fenster, an der die Textkonstante eingefügt werden soll.

Der Mauszeiger verändert sich. Er zeigt jetzt ein Kreuz und das Symbol für eine Textkonstante.

- 3 Drücken Sie die Maustaste und halten Sie sie gedrückt.
- 4 Ziehen Sie die Maus solange nach rechts bis das Feld die gewünschte Länge hat. Für unser Feld ist eine Länge von etwa 10 Zeichen ausreichend.

Während Sie die Maus ziehen, wird die aktuelle Länge in der Statusleiste des Anwendungsfensters angezeigt. Format und Position in der Map werden dort ebenfalls angezeigt. Ein Textfeld hat immer das Format A (alphanumerisch).

- 5 Lassen Sie die Maustaste los.

Jetzt wird ein Textfeld mit einer Standardbezeichnung angezeigt. Der Anfasser lässt erkennen, dass das Textfeld markiert ist. Der Mauszeiger zeigt noch immer ein Kreuz und das Symbol für eine Textkonstante; Sie können sofort ein weiteres Textfeld erstellen.



Anmerkung: Wenn Sie diesen Modus verlassen wollen, können Sie eine beliebige Position in der Map anklicken.

- 6 Erstellen Sie ein zweites Textfeld unter dem ersten Textfeld.

Falls das Feld in der falschen Spalte beginnt, können Sie es verschieben, indem Sie den Mauszeiger auf dieses Feld stellen, die Maustaste drücken und gedrückt halten, und die Maus dann an die gewünschte Position ziehen. Wenn Sie die Maustaste loslassen, wird wieder der normale Mauszeiger angezeigt.

Bezeichnungen für die Textfelder definieren

Die Textfelder, die Sie eben erstellt haben, haben noch nicht die richtigen Bezeichnungen. Diese werden Sie jetzt definieren.

▶ Bezeichnungen definieren

- 1 Markieren Sie das erste Textfeld und wählen Sie aus dem Kontextmenü den Befehl **Definition**.

Oder:

Klicken Sie das Textfeld doppelt an.

Der bestehende Text wird markiert.

- 2 Geben Sie "Start:" als Bezeichnung für das erste Textfeld ein und drücken Sie EINGABE.

Das Textfeld (für das Sie vorher eine Länge von 10 Zeichen definiert haben) wird automatisch auf die Länge dieser Zeichenkette gebracht.

- 3 Wiederholen Sie die Schritte oben und geben Sie "End:" als Bezeichnung für das zweite Textfeld ein.

Datenfelder definieren

Sie werden jetzt zwei Datenfelder in der Map definieren. Dies sind die Eingabefelder, in denen der Benutzer den Start- und Endnamen eingeben kann.

Sie können die Datenfelder auf zwei verschiedene Arten definieren: mit dem Befehl **Data Field**, wobei es in Ihrer Verantwortung liegt, das Format und die Länge des Datenfelds korrekt anzugeben, oder mit dem Befehl **Import > Data Field**, wobei Sie das Datenfeld einfach aus einer Liste auswählen und das Format und die Länge bereits korrekt definiert sind. Diese beiden Arten sind unten beschrieben.

▶ Datenfeld definieren und dabei die Länge selbst angeben

- 1 Wählen Sie aus dem Menü **Insert** den Befehl **Data Field**.

Oder:

Wählen Sie die folgende Schaltfläche in der Symbolleiste:



- 2 Erstellen Sie das Datenfeld rechts neben dem zuvor eingegebenen Textfeld für den Startnamen. Sie erstellen das Datenfeld auf dieselbe Weise wie ein Textfeld. Für das Datenfeld ist eine Länge von 20 Zeichen erforderlich (falls Ihr Feld zu kurz oder zu lang ist, erfahren Sie in einer späteren Übung, wie Sie dies ändern können). Das Datenfeld wird automatisch mit mehreren "X"-Zeichen gefüllt.

▶ Datenfeld importieren

- 1 Wählen Sie aus dem Menü **Insert** den Befehl **Import > Data Field**.

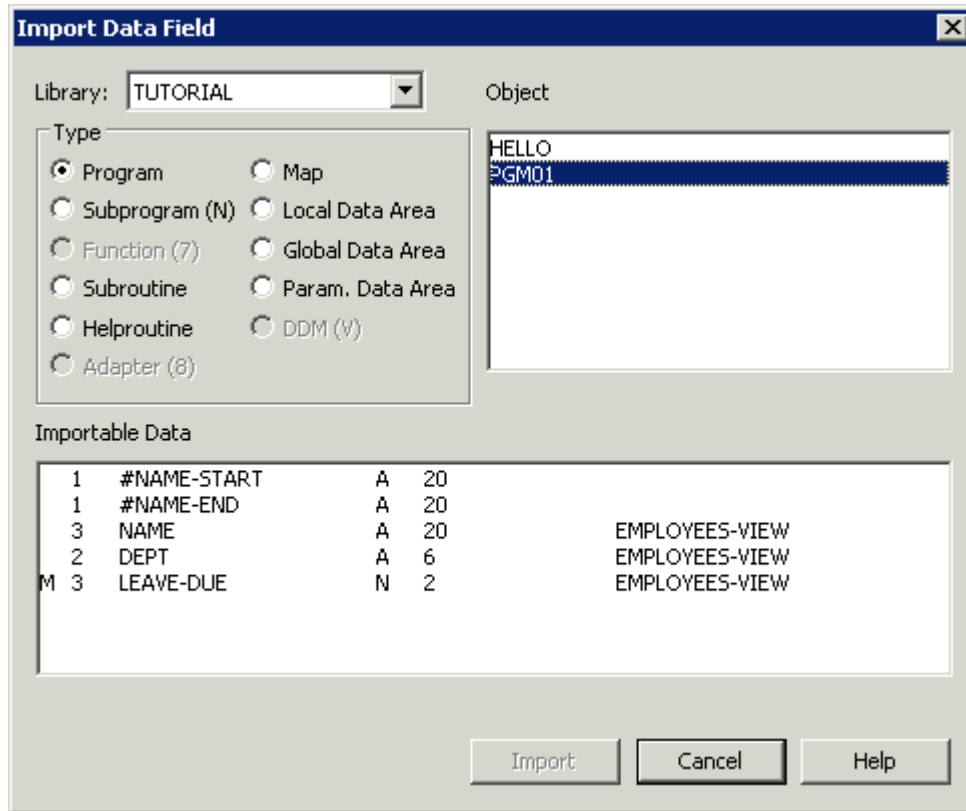
Das Dialogfeld **Import Data Field** erscheint.

- 2 Wählen Sie aus dem Dropdown-Listenfeld **Library** den Eintrag **TUTORIAL**.
- 3 Markieren Sie das Optionsfeld **Program**.

Alle Programme, die zurzeit in Ihrer Library definiert sind, werden jetzt im Listenfeld **Object** angezeigt.

- 4 Markieren Sie das Programm mit dem Namen PGM01.

Die importierbaren Felder werden jetzt unten im Dialogfeld angezeigt.



- 5 Markieren Sie das Feld `#NAME-END` und wählen Sie die Befehlsschaltfläche **Import**.

Der Name der Befehlsschaltfläche **Cancel** im Dialogfeld **Import Data Field** ändert sich in **Quit**.

- 6 Wählen Sie die Befehlsschaltfläche **Quit**, um das Dialogfeld **Import Data Field** zu schließen.

Das Datenfeld wird jetzt oben links in der Map angezeigt. Es ist mit "X"-Zeichen gefüllt. Anfasser weisen darauf hin, dass das Datenfeld markiert ist.

- 7 Verschieben Sie das Datenfeld, so dass es rechts neben dem zuvor eingegebenen Textfeld für den Endnamen erscheint. Hierzu stellen Sie den Mauszeiger auf das Feld, drücken die Maustaste und halten sie gedrückt, ziehen die Maus an die gewünschte Position und lassen die Maustaste wieder los.

Namen und Attribute für Datenfelder definieren

Folgendes gilt nur für das Datenfeld für den Startnamen, das Sie manuell definiert haben. Es gilt nicht für das Datenfeld für den Endnamen, das Sie importiert haben: Wenn Sie ein neues Datenfeld für eine Benutzervariable erstellen, wird von Natural ein Name für dieses Feld vergeben. Dieser Feldname enthält eine Nummer. Sie müssen die Namen der neu erstellten Felder an die Namen anpassen, die in Ihrem Programm definiert wurden.

Sie werden jetzt dafür sorgen, dass dieselben Namen wie in Ihrem Programm benutzt werden: `#NAME-START` und `#NAME-END`. Die Ausgabe dieser Felder (d.h. die Benutzereingabe) wird an die entsprechenden Benutzervariablen in Ihrem Programm übergeben.

Sie werden auch dafür sorgen, dass für `#NAME-START` und `#NAME-END` dieselben Attribute definiert sind.

► Namen und Attribute für Datenfelder definieren

- 1 Markieren Sie das Datenfeld für den Startnamen und wählen Sie aus dem Kontextmenü den Befehl **Definition**.

Oder:

Klicken Sie das Datenfeld doppelt an.

Das Dialogfeld **Field Definition** erscheint.

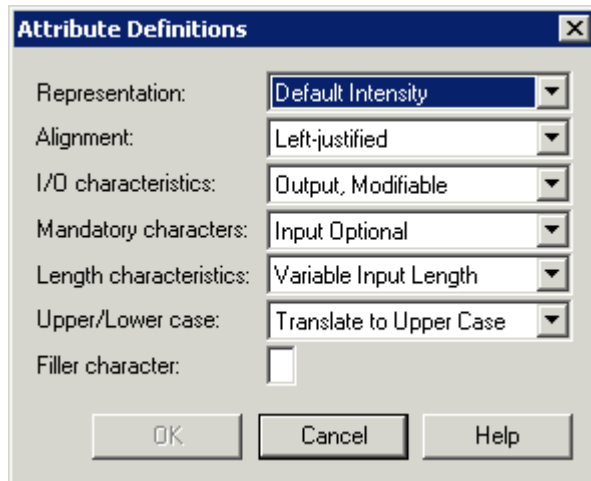
Das Textfeld **Field** enthält den Feldnamen, der von Natural vergeben wurde: `#FIELD_#1`.

- 2 Geben Sie "#NAME-START" im Textfeld **Field** ein.

Das Format muss **A** ein. Dies wird automatisch vorgegeben.

- 3 Geben Sie "20" im Textfeld **Length** ein (falls Ihr Feld eine andere Länge hat).
- 4 Wählen Sie die Befehlsschaltfläche **Attributes**.

Das Dialogfeld **Attribute Definitions** erscheint.



- 5 Achten Sie darauf, dass im Dropdown-Listenfeld **I/O characteristics** der Eintrag **Output, Modifiable** ausgewählt ist.

Hiermit wird das Feld als modifizierbares Ausgabefeld definiert.

- 6 Achten Sie darauf, dass im Dropdown-Listenfeld **Upper/Lower case** der Eintrag **Translate to Upper Case** ausgewählt ist.

Dies ermöglicht es dem Benutzer, den Namen in Kleinbuchstaben einzugeben. Bis jetzt konnten die Namen in der Demodatenbank nur gefunden werden, wenn die Namen komplett in Großbuchstaben eingegeben wurden.

- 7 Geben Sie im Textfeld **Filler character** einen Unterstrich (_) ein.

Standardmäßig ist ein Leerzeichen als Füllzeichen definiert. Deshalb müssen Sie das Leerzeichen zuerst löschen, bevor Sie den Unterstrich eingeben können.

Füllzeichen werden in der Map benutzt, um leere Positionen in den Eingabefeldern zu füllen, damit der Benutzer bei der Dateneingabe die exakte Position und Länge eines Feldes erkennen kann.

- 8 Wählen Sie die Befehlsschaltfläche **OK**, um das Dialogfeld **Attribute Definitions** zu schließen.
- 9 Wählen Sie die Befehlsschaltfläche **OK**, um das Dialogfeld **Field Definition** zu schließen.

- 10 Wiederholen Sie die oben aufgeführten Schritte für den Endnamen. Achten Sie darauf, dass der richtige Feldname (#NAME - END), das richtige Format (A) und die richtige Länge (20) definiert sind. Achten Sie darauf, dass dieselben Attributdefinitionen wie bei #NAME - START benutzt werden.

Systemvariablen einfügen

Natural-Systemvariablen enthalten Information über die aktuelle Natural-Session, wie zum Beispiel die aktuelle Library, Benutzer, oder Datum und Uhrzeit. Diese Informationen können an jedem beliebigen Punkt in einem Natural-Programm referenziert werden. Alle Systemvariablen beginnen mit einem Stern (*).

Sie werden jetzt Systemvariablen für Datum und Uhrzeit in der Map einfügen. Wenn das Programm mit RUN ausgeführt wird, werden das aktuelle Datum und die aktuelle Uhrzeit in der Map angezeigt.

▶ Systemvariablen einfügen

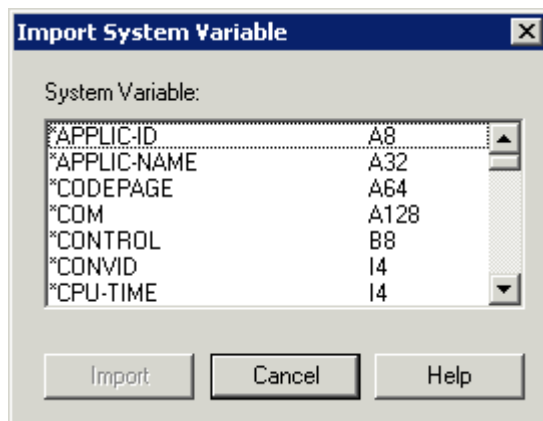
- 1 Wählen Sie aus dem Menü **Insert** den Befehl **Import > System Variable**.

Oder:

Wählen Sie die folgende Schaltfläche in der Symbolleiste:



Das Dialogfeld **Import System Variable** erscheint.



- 2 Rollen Sie die Liste bis zu *DAT4I und markieren Sie diesen Eintrag.
- 3 Rollen Sie die Liste bis zu *TIMX, drücken Sie STRG und markieren Sie diesen Eintrag.
- 4 Wählen Sie die Befehlsschaltfläche **Import**, um die markierten Variablen zu importieren.

Die Befehlsschaltfläche **Cancel** im Dialogfeld **Import System Variable** trägt nun den Namen **Quit**.

- 5 Wählen Sie die Befehlsschaltfläche **Quit**, um das Dialogfeld zu schließen.

Beide Systemvariablen sind nun oben links in der Map zu sehen.

- 6 Markieren Sie **TT:TT:TT** (d.h. die Systemvariable für die Uhrzeit) und verschieben Sie dies an das Ende der ersten Zeile.



Anmerkung: Es ist unter Umständen erforderlich, das Map-Editor-Fenster zu vergrößern, damit Sie das Ende der Zeile sehen können.

Map testen

Um zu überprüfen, ob Ihre Map wie beabsichtigt funktioniert, werden Sie sie jetzt testen.

▶ Map testen

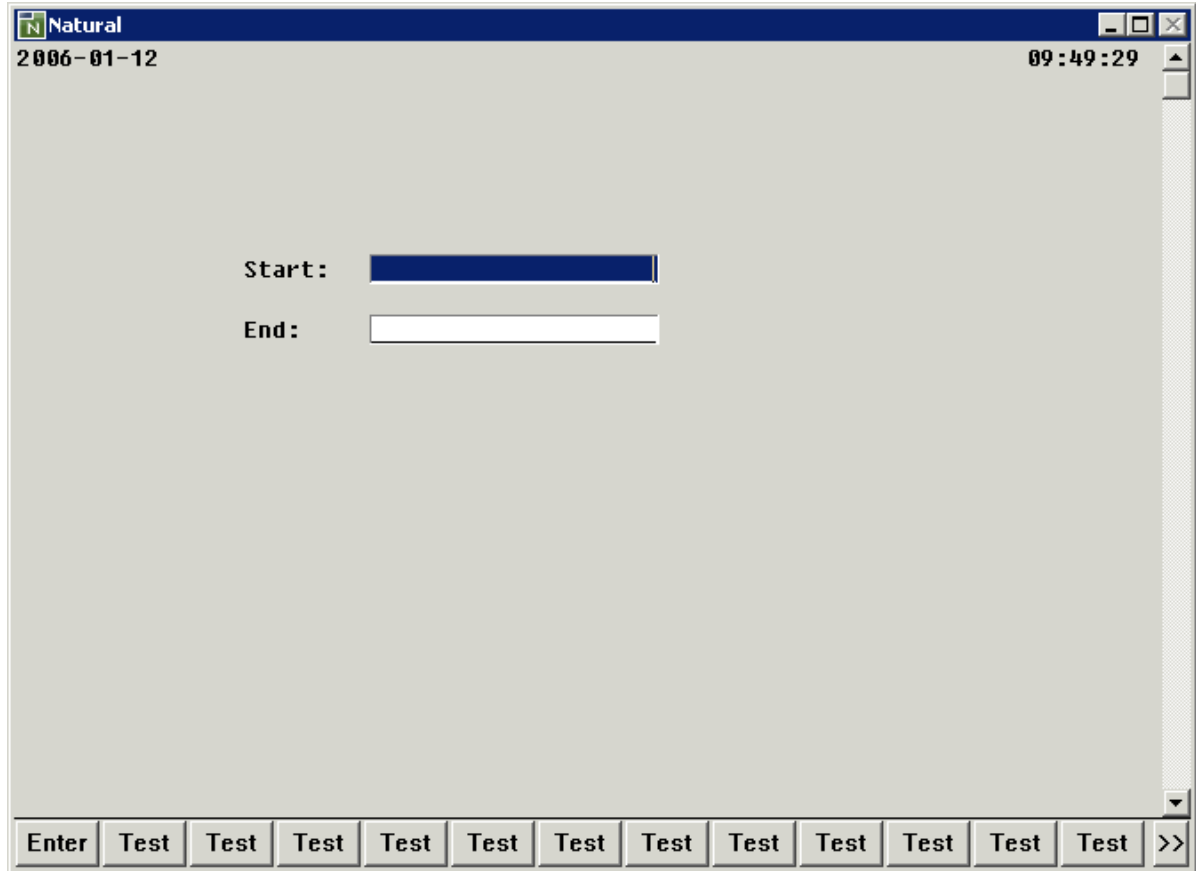
- 1 Wählen Sie aus dem Menü **Object** den Befehl **Test**.


Oder:

Wählen Sie die folgende Schaltfläche in der Symbolleiste:




Die folgende Ausgabe wird angezeigt.



 **Anmerkung:** Es ist eventuell erforderlich, das Ausgabefenster zu vergrößern, damit Sie das Datum oben rechts sehen können.

Das Eingabefeld für den Startnamen ist automatisch markiert, da es das erste Eingabefeld in der Map ist. Beide Eingabefelder enthalten Füllzeichen.

 **Anmerkung:** Wenn der Einfügemodus aktiv ist, muss der Benutzer die Füllzeichen löschen, damit die Eingabe von Text möglich ist. Im Überschreibemodus ist dies nicht erforderlich.

- 2 Drücken Sie EINGABE, um zum Map-Editor zurückzukehren.

Map mit STOW speichern

Wenn die Map erfolgreich getestet wurde, müssen Sie sie mit STOW speichern, damit sie von Ihrem Programm gefunden werden kann.

▶ Map mit STOW speichern

- 1 Speichern Sie die Map auf dieselbe Weise mit STOW, wie Sie ein Programm speichern.
- 2 Wenn Sie nach einem Namen für die Map gefragt werden, geben Sie "MAP01" ein.

Im Library-Workspace erscheint ein neuer Knoten mit dem Namen **Maps** als Unterknoten von **TUTORIAL**. Dieser Unterknoten enthält die Map, die Sie eben gespeichert haben.

Lassen Sie den Map-Editor für spätere Änderungen offen.

Map aus dem Programm aufrufen

Sobald eine Map mit STOW gespeichert wurde, kann Sie mit einem WRITE- oder INPUT-Statement aus einem Natural-Programm heraus aufgerufen werden.

▶ Map aus dem Programm aufrufen

- 1 Kehren Sie zum Programmeditor zurück.

Falls Sie den Programmeditor nicht sehen können (weil Sie vorher zum Beispiel das Fenster für den Map Editor vergrößert haben), können Sie zu einem offenen Programmeditorfenster zurückkehren, indem Sie den entsprechenden Befehl für PGM01 aus dem Menü **Window** wählen.

Sie können auch PGM01 im Library-Workspace doppelt anklicken (oder wenn Sie mit der Tastatur arbeiten, können Sie es markieren und EINGABE drücken). Wenn das Programm zuvor geschlossen wurde, wird es dadurch wieder geöffnet. Wenn es im Hintergrund noch immer geöffnet ist, wird das Editorfenster dadurch wieder in den Vordergrund geholt.

- 2 Ersetzen Sie die vorher definierten INPUT-Zeilen durch die folgende Zeile:

```
INPUT USING MAP 'MAP01'
```

Hiermit wird die von Ihnen erstellte Map aufgerufen.

Der Name der Map muss in einfache Anführungszeichen gesetzt werden, um die Map von einer Benutzervariablen zu unterscheiden.

Ihr Programm sollte nun folgendermaßen aussehen:

```
DEFINE DATA
LOCAL
  1 #NAME-START      (A20) INIT <"ADKINSON">
  1 #NAME-END        (A20) INIT <"BENNETT">
  1 EMPLOYEES-VIEW  VIEW OF EMPLOYEES
    2 FULL-NAME
      3 NAME (A20)
    2 DEPT (A6)
    2 LEAVE-DATA
      3 LEAVE-DUE (N2)
END-DEFINE
*
INPUT USING MAP 'MAP01'
*
READ EMPLOYEES-VIEW BY NAME
  STARTING FROM #NAME-START
  ENDING AT #NAME-END
*
  DISPLAY NAME 3X DEPT 3X LEAVE-DUE
*
END-READ
*
END
```

- 3 Führen Sie das Programm mit RUN aus.
Ihre Map wird nun angezeigt.
- 4 Drücken Sie wiederholt EINGABE bis Sie zum Programmeditor zurückkehren, oder drücken Sie ESC.
- 5 Speichern Sie das Programm mit STOW.

Immer einen Endnamen benutzen

So wie Ihr Programm jetzt kodiert ist, werden keine Daten gefunden, wenn kein Endname angegeben wird.

Sie werden jetzt die Anfangswerte für Start- und Endname entfernen; danach müssen diese Namen immer vom Benutzer angegeben werden. Um zu gewährleisten, dass ein Endname immer benutzt wird, auch wenn er vom Benutzer nicht eingegeben wurde, werden Sie Ihrem Programm nun ein entsprechendes Statement hinzufügen.

► Den Endnamen benutzen

- 1 Gehen Sie zum `DEFINE DATA`-Block und entfernen Sie die Anfangswerte (`INIT`) für die Felder `#NAME-START` und `#NAME-END`, so dass die entsprechenden Zeilen folgendermaßen aussehen:

```
1 #NAME-START      (A20)
1 #NAME-END        (A20)
```

- 2 Geben Sie Folgendes unter `INPUT USING MAP 'MAP01'` ein:

```
IF #NAME-END = ' ' THEN
    MOVE #NAME-START TO #NAME-END
END-IF
```

Wenn das Feld `#NAME-END` leer ist (d.h. wenn der Benutzer keinen Endnamen angibt), wird der Startname automatisch als Endname benutzt.



Anmerkung: Statt des Statements `MOVE #NAME-START TO #NAME-END` können Sie auch die folgende Variante des `ASSIGN`- oder `COMPUTE`-Statements benutzen: `#NAME-END := #NAME-START`.

Ihr Programm sollte nun folgendermaßen aussehen:

```
DEFINE DATA
LOCAL
1 #NAME-START      (A20)
1 #NAME-END        (A20)
1 EMPLOYEES-VIEW VIEW OF EMPLOYEES
2 FULL-NAME
3 NAME (A20)
2 DEPT (A6)
2 LEAVE-DATA
3 LEAVE-DUE (N2)
END-DEFINE
*
```

```
INPUT USING MAP 'MAP01'  
*  
IF #NAME-END = ' ' THEN  
  MOVE #NAME-START TO #NAME-END  
END-IF  
*  
READ EMPLOYEES-VIEW BY NAME  
  STARTING FROM #NAME-START  
  ENDING AT #NAME-END  
*  
  DISPLAY NAME 3X DEPT 3X LEAVE-DUE  
*  
END-READ  
*  
END
```

- 3 Führen Sie das Programm mit RUN aus.
- 4 Geben Sie in der daraufhin erscheinenden Map "JONES" in dem Feld für den Startnamen ein und drücken Sie EINGABE.



Anmerkung: Da **Translate to Upper Case** für dieses Feld definiert wurde, können Sie den Namen jetzt auch in Kleinbuchstaben angeben.

In der daraufhin erscheinenden Liste werden jetzt nur die Mitarbeiter mit dem Namen "Jones" angezeigt.

- 5 Drücken Sie EINGABE, um zum Programmeditor zurückzukehren.
- 6 Speichern Sie das Programm mit STOW.

Sie können nun mit den nächsten Übungen fortfahren: [Verarbeitungsschleifen und Labels](#).

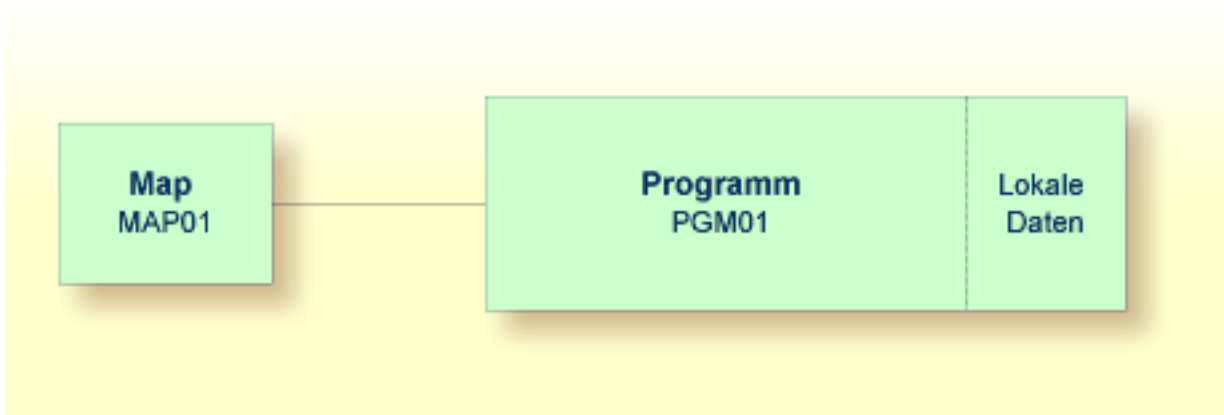
7

Verarbeitungsschleifen und Labels

- Wiederholte Benutzung erlauben 52
- Meldung für nicht gefundene Informationen anzeigen 54

Sie werden Ihr Programm jetzt erweitern, indem Sie Verarbeitungsschleifen und Labels hinzufügen.

Wenn Sie mit den Übungen in diesem Kapitel fertig sind, wird Ihre Beispielanwendung immer noch aus denselben Modulen wie im vorherigen Kapitel bestehen:



Wiederholte Benutzung erlauben

So wie Ihr Programm jetzt aufgebaut ist, wird es sofort nach dem Anzeigen der Map und dem Anzeigen der Mitarbeiterliste beendet. Damit der Benutzer sofort wieder eine neue Mitarbeiterliste anzeigen kann, ohne das Programm neu zu starten, werden Sie den entsprechenden Programmcode nun in eine REPEAT-Schleife setzen.

▶ Verarbeitungsschleife definieren

- 1 Geben Sie Folgendes unter END-DEFINE ein:

```
RP1 . REPEAT
```

REPEAT definiert den Beginn der Verarbeitungsschleife. RP1 . ist ein Label, das beim Verlassen der Verarbeitungsschleife benutzt wird (dies wird weiter unten definiert).

- 2 Definieren Sie das Ende der Verarbeitungsschleife, indem Sie Folgendes vor dem END-Statement eingeben:

```
END-REPEAT
```

- 3 Geben Sie Folgendes unter INPUT USING MAP 'MAP01' ein:

```
IF #NAME-START = '.' THEN
  ESCAPE BOTTOM (RP1.)
END-IF
```

Das IF-Statement, das mit END-IF beendet werden muss, prüft den Inhalt des Feldes #NAME-START. Wenn ein Punkt (.) in diesem Feld eingegeben wird, wird das Statement ESCAPE BOTTOM zum Verlassen der Schleife benutzt. Die Verarbeitung wird mit dem ersten Statement nach der Schleife fortgesetzt (in diesem Fall ist das END).

Indem Sie einer Schleife ein Label zuordnen (hier ist es RP1.), können Sie sich mit dem Statement ESCAPE BOTTOM auf diese Schleife beziehen. Da Schleifen geschachtelt werden können, sollten Sie angeben, welche Schleife Sie verlassen möchten. Andernfalls wird nur die innerste aktive Schleife verlassen.

Ihr Programm sollte nun folgendermaßen aussehen:

```
DEFINE DATA
LOCAL
  1 #NAME-START      (A20)
  1 #NAME-END        (A20)
  1 EMPLOYEES-VIEW  VIEW OF EMPLOYEES
    2 FULL-NAME
      3 NAME (A20)
    2 DEPT (A6)
    2 LEAVE-DATA
      3 LEAVE-DUE (N2)
END-DEFINE
*
RP1. REPEAT
*
  INPUT USING MAP 'MAP01'
*
  IF #NAME-START = '.' THEN
    ESCAPE BOTTOM (RP1.)
  END-IF
*
  IF #NAME-END = ' ' THEN
    MOVE #NAME-START TO #NAME-END
  END-IF
*
  READ EMPLOYEES-VIEW BY NAME
  STARTING FROM #NAME-START
```

```
ENDING AT #NAME-END
*
DISPLAY NAME 3X DEPT 3X LEAVE-DUE
*
END-READ
*
END-REPEAT
*
END
```



Anmerkung: Zur besseren Lesbarkeit wurde der Inhalt der REPEAT-Schleife eingerückt.

- 4 Führen Sie das Programm mit `RUN` aus.
- 5 Geben Sie in der daraufhin erscheinenden Map "JONES" in dem Feld für den Startnamen ein und drücken Sie `EINGABE`.

In der daraufhin erscheinenden Liste werden nur die Mitarbeiter mit dem Namen "Jones" angezeigt. Drücken Sie `EINGABE`. Wegen der `REPEAT`-Schleife wird die Map wieder angezeigt. Sie können nun sehen, dass "JONES" als Endname eingetragen wurde.

- 6 Geben Sie einen Punkt (.) in dem Feld für den Startnamen ein und drücken Sie `EINGABE`, um die Map zu verlassen. Vergessen Sie nicht, die restlichen Zeichen des Namens zu löschen, der noch in diesem Feld angezeigt wird.
- 7 Speichern Sie das Programm mit `STOW`.

Meldung für nicht gefundene Informationen anzeigen

Sie werden jetzt die Meldung definieren, die angezeigt werden soll, wenn der Benutzer einen Startnamen angibt, der in der Datenbank nicht gefunden werden kann.

► Meldung für nicht gefundene Mitarbeiter definieren

- 1 Geben Sie das Label `RD1.` am Anfang der Zeile ein, die das `READ`-Statement enthält:

```
RD1. READ EMPLOYEES-VIEW BY NAME
```

2 Geben Sie Folgendes unter END-READ ein:

```
IF *COUNTER (RD1.) = 0 THEN
  REINPUT 'No employees meet your criteria.'
END-IF
```

Mit der Systemvariablen *COUNTER wird die Anzahl der Datensätze geprüft, die in der READ-Schleife gefunden werden. Wenn der Inhalt gleich 0 ist (d.h. wenn ein Mitarbeiter mit dem angegebenen Namen nicht gefunden wurde), wird die Meldung, die mit dem REINPUT-Statement definiert wurde, unten in der Map angezeigt.

Um die READ-Schleife zu bezeichnen, geben Sie ihr ein Label (in diesem Fall ist es RD1.). Da ein komplexes Datenbankzugriffsprogramm viele Schleifen enthalten kann, müssen Sie angeben, auf welche Schleife Sie sich beziehen.

Ihr Programm sollte nun folgendermaßen aussehen:

```
DEFINE DATA
LOCAL
  1 #NAME-START      (A20)
  1 #NAME-END        (A20)
  1 EMPLOYEES-VIEW  VIEW OF EMPLOYEES
    2 FULL-NAME
      3 NAME (A20)
    2 DEPT (A6)
    2 LEAVE-DATA
      3 LEAVE-DUE (N2)
END-DEFINE
*
RP1. REPEAT
*
  INPUT USING MAP 'MAP01'
*
  IF #NAME-START = '.' THEN
    ESCAPE BOTTOM (RP1.)
  END-IF
*
  IF #NAME-END = ' ' THEN
    MOVE #NAME-START TO #NAME-END
  END-IF
*
  RD1. READ EMPLOYEES-VIEW BY NAME
    STARTING FROM #NAME-START
    ENDING AT #NAME-END
*
  DISPLAY NAME 3X DEPT 3X LEAVE-DUE
*
END-READ
*
IF *COUNTER (RD1.) = 0 THEN
```

```
    REINPUT 'No employees meet your criteria.'  
  END-IF  
*  
END-REPEAT  
*  
END
```

- 3 Führen Sie das Programm mit `RUN` aus.
- 4 Geben Sie in der daraufhin erscheinenden Map einen Startnamen ein, der nicht in der Demodatenbank definiert ist (zum Beispiel "XYZ") und drücken Sie `EINGABE`.

Ihre Meldung sollte jetzt in der Map angezeigt werden.

- 5 Geben Sie einen Punkt (.) in dem Feld für den Startnamen ein und drücken Sie `EINGABE`, um die Map zu verlassen. Vergessen Sie nicht, die restlichen Zeichen des Namens zu löschen, der noch in diesem Feld angezeigt wird.

Oder:

Drücken Sie `ESC`.

- 6 Speichern Sie das Programm mit `STOW`.

Sie können nun mit den nächsten Übungen fortfahren: [Interne Subroutinen](#).

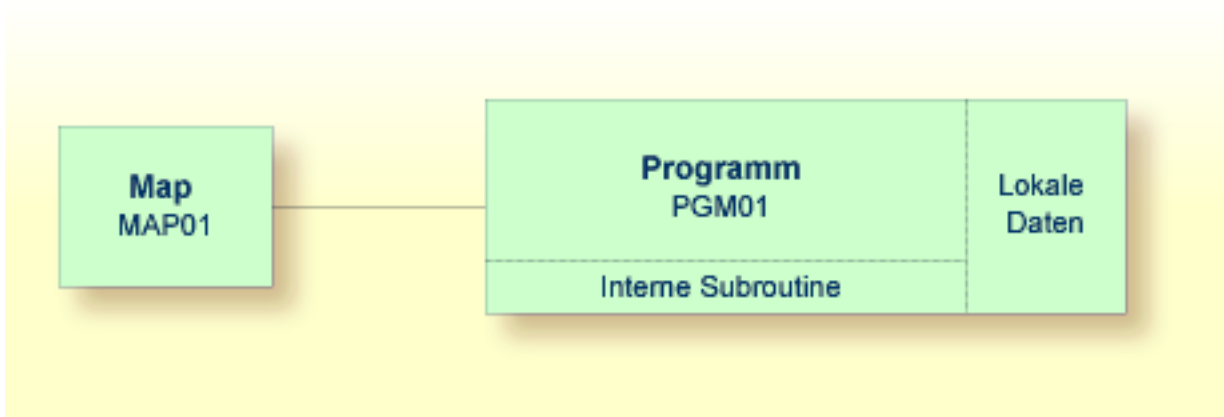
8 Interne Subroutinen

- Interne Subroutine definieren 58
- Interne Subroutine ausführen 59

Natural unterscheidet zwei Arten von Subroutinen: interne Subroutinen, die direkt im Programm definiert werden, und externe Subroutinen, die als separate Objekte außerhalb des Programms gespeichert werden (dies wird später in diesem Tutorial beschrieben).

Sie werden jetzt eine interne Subroutine in Ihrem Programm definieren, mit der ein Stern (*) in die neue Benutzervariable #MARK geschrieben wird. Diese Subroutine wird aufgerufen, wenn ein Mitarbeiter 20 oder mehr Urlaubstage hat.

Wenn Sie mit den Übungen in diesem Kapitel fertig sind, wird Ihre Beispielanwendung folgendermaßen strukturiert sein:



Interne Subroutine definieren

Sie werden jetzt die interne Subroutine in Ihr Programm einfügen.

► Interne Subroutine definieren

- 1 Geben Sie Folgendes unter der Benutzervariablen #NAME - END ein:

```
1 #MARK (A1)
```

Diese Variable wird von der Subroutine benutzt. Daher muss Sie zuerst definiert werden.

- 2 Um die Subroutine zu definieren, geben Sie Folgendes vor dem END-Statement ein:

```
DEFINE SUBROUTINE MARK-SPECIAL-EMPLOYEES
  MOVE '*' TO #MARK
END-SUBROUTINE
```

Wenn diese Subroutine ausgeführt wird, schreibt sie einen Stern (*) in die Benutzervariable #MARK.



Anmerkung: Statt des Statements `MOVE '*' TO #MARK` können Sie auch die folgende Variante des ASSIGN- oder COMPUTE-Statements benutzen: `#MARK := '*'`.

- 3 Ändern Sie das DISPLAY-Statement wie folgt:

```
DISPLAY NAME 3X DEPT 3X LEAVE-DUE 3X '>=20' #MARK
```

Hiermit wird eine neue Spalte in der Ausgabe angezeigt. Die Überschrift dieser Spalte ist ">=20". Die Spalte enthält einen Stern (*), wenn der betreffende Mitarbeiter 20 oder mehr Urlaubstage hat.

Interne Subroutine ausführen

Jetzt, nachdem Sie die interne Subroutine definiert haben, können Sie den entsprechenden Code zum Aufruf dieser Subroutine eingeben.

▶ Interne Subroutine ausführen

- 1 Geben Sie Folgendes vor dem DISPLAY-Statement ein:

```
IF LEAVE-DUE >= 20 THEN
  PERFORM MARK-SPECIAL-EMPLOYEES
ELSE
  RESET #MARK
END-IF
```

Wenn ein Mitarbeiter gefunden wird, der 20 oder mehr Urlaubstage (LEAVE-DUE) hat, wird die neue Subroutine mit dem Namen MARK-SPECIAL-EMPLOYEES ausgeführt. Wenn ein Mitarbeiter weniger als 20 Urlaubstage hat, wird der Inhalt von #MARK zurückgesetzt.

Ihr Programm sollte nun folgendermaßen aussehen:

```

DEFINE DATA
LOCAL
  1 #NAME-START      (A20)
  1 #NAME-END        (A20)
  1 #MARK            (A1)
  1 EMPLOYEES-VIEW  VIEW OF EMPLOYEES
    2 FULL-NAME
      3 NAME (A20)
    2 DEPT (A6)
    2 LEAVE-DATA
      3 LEAVE-DUE (N2)
END-DEFINE
*
RP1. REPEAT
*
  INPUT USING MAP 'MAP01'
*
  IF #NAME-START = '.' THEN
    ESCAPE BOTTOM (RP1.)
  END-IF
*
  IF #NAME-END = ' ' THEN
    MOVE #NAME-START TO #NAME-END
  END-IF
*
  RD1. READ EMPLOYEES-VIEW BY NAME
    STARTING FROM #NAME-START
    ENDING AT #NAME-END
*
    IF LEAVE-DUE >= 20 THEN
      PERFORM MARK-SPECIAL-EMPLOYEES
    ELSE
      RESET #MARK
    END-IF
*
    DISPLAY NAME 3X DEPT 3X LEAVE-DUE 3X '>=20' #MARK
*
  END-READ
*
  IF *COUNTER (RD1.) = 0 THEN
    REINPUT 'No employees meet your criteria.'
  END-IF
*
END-REPEAT
*
DEFINE SUBROUTINE MARK-SPECIAL-EMPLOYEES
  MOVE '*' TO #MARK
END-SUBROUTINE

```

```
*  
END
```

- 2 Führen Sie das Programm mit `RUN` aus.
- 3 Geben Sie in der daraufhin erscheinenden Map "JONES" ein und drücken Sie `EINGABE`.

Die Liste der Mitarbeiter sollte jetzt eine zusätzliche Spalte enthalten.

- 4 Drücken Sie `ESC`, um das Ausgabefenster zu schließen.
- 5 Speichern Sie das Programm mit `STOW`.

Sie können nun mit den nächsten Übungen fortfahren: *Verarbeitungsregeln und Helproutinen*.

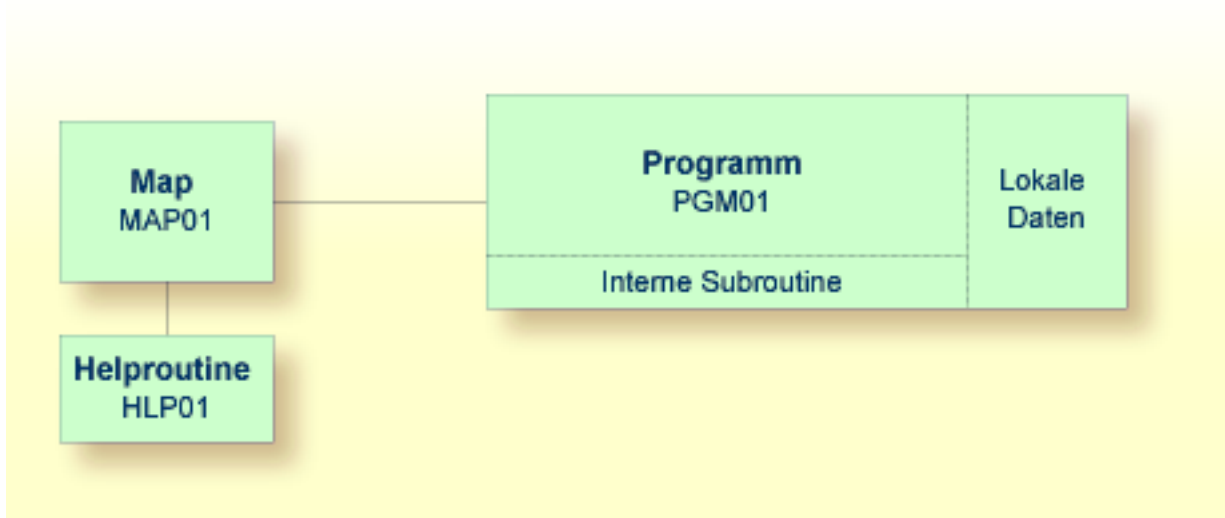
9

Verarbeitungsregeln und Helproutinen

- Verarbeitungsregel definieren 64
- Helproutine definieren 67

Verarbeitungsregeln und Helproutinen werden für die Felder in einer Map definiert.

Wenn Sie mit den Übungen in diesem Kapitel fertig sind, wird Ihre Beispielanwendung aus den folgenden Modulen bestehen (eine Verarbeitungsregel kann nicht als separates Modul definiert werden; sie ist immer Teil einer Map):



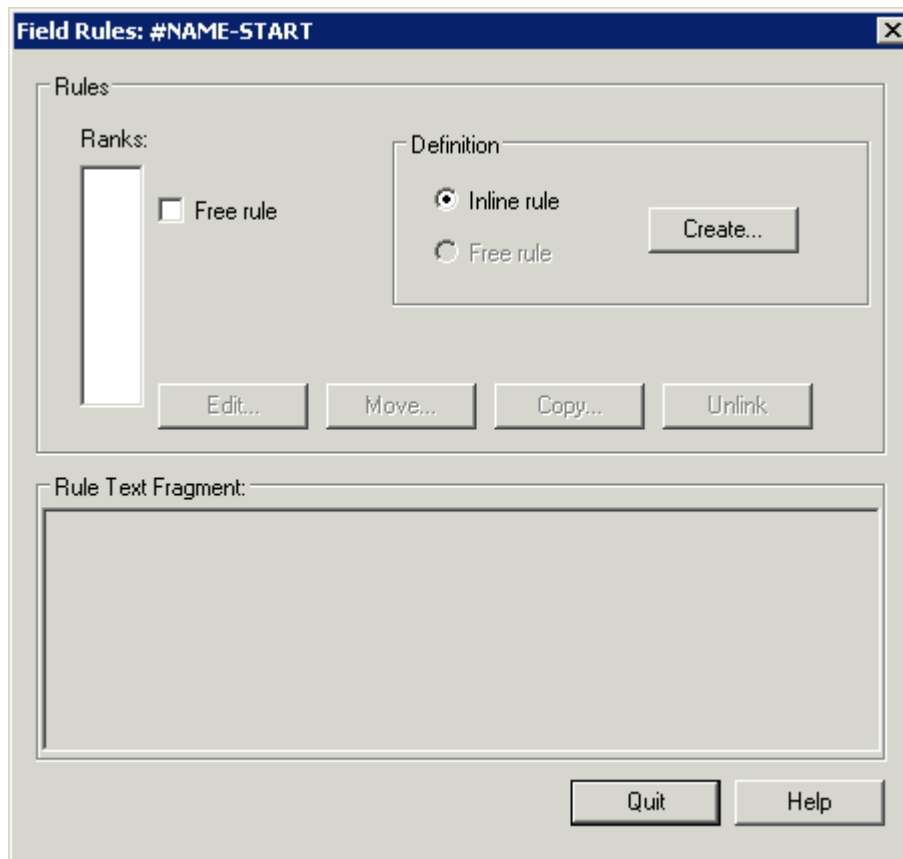
Verarbeitungsregel definieren

Sie werden jetzt die Meldung definieren, die angezeigt werden soll, wenn der Benutzer EINGABE drückt ohne vorher einen Startnamen anzugeben.

▶ Verarbeitungsregel definieren

- 1 Kehren Sie zum Map-Editor zurück.
- 2 Markieren Sie das Eingabefeld für den Startnamen.
- 3 Wählen Sie aus dem Kontextmenü den Befehl **Rules**.

Das Dialogfeld **Field Rules** erscheint für das Feld #NAME-START.



- 4 Wählen Sie die Befehlsschaltfläche **Create**.

Ein leeres Editorfenster erscheint.

- 5 Geben Sie die folgende Verarbeitungsregel ein:

```
IF & = ' ' THEN
  REINPUT 'Please enter a starting name.'
  MARK *&
END-IF
```

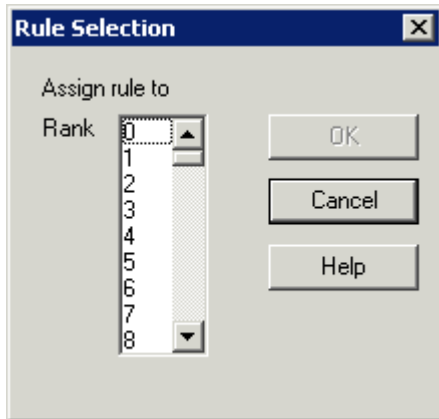
Das Kaufmanns-Und (&) in der Verarbeitungsregel wird bei der Ausführung des Programms dynamisch mit dem Namen des Feldes ersetzt. In diesem Fall wird es mit #NAME-START ersetzt. Wenn #NAME-START leer ist, wird die Meldung angezeigt, die mit dem REINPUT-Statement definiert wurde.

MARK ist eine Option des REINPUT-Statements. Die Syntax ist MARK **fieldname*.

MARK definiert das Feld, in das der Cursor gestellt werden soll, wenn das REINPUT-Statement ausgeführt wird. In diesem Fall wird der Cursor in das Feld #NAME-START gestellt.

- 6 Speichern Sie den Inhalt des Editorfensters.

Das Dialogfeld **Rule Selection** erscheint.



- 7 Markieren Sie im Listenfeld **Rank** (Rang) den Wert **1** und wählen Sie die Befehlsschaltfläche **OK**.

Mit dem Rang wird die Reihenfolge definiert, in der die Regeln für die verschiedenen Felder verarbeitet werden. Alle Regeln mit Rang 1 werden zuerst verarbeitet, gefolgt von denen mit Rang 2, usw.

- 8 Schließen Sie das Editorfenster, in dem Sie die Verarbeitungsregel eingegeben haben.
- 9 Testen Sie die Map.
- 10 Geben Sie in der daraufhin erscheinenden Ausgabe einen beliebigen Startnamen ein und drücken Sie EINGABE.

Das Ausgabefenster wird geschlossen.

- 11 Testen Sie die Map noch einmal. Geben Sie dieses Mal keinen Namen ein und drücken Sie EINGABE.

Die Meldung, die mit der Verarbeitungsregel definiert wurde, sollte jetzt in der Map erscheinen.

- 12 Geben Sie einen Punkt (.) in dem Feld für den Startnamen ein und drücken Sie EINGABE, um das Ausgabefenster zu verlassen.
- 13 Speichern Sie die Map mit STOW.

Helproutine definieren

Eine Helproutine wird ausgeführt, wenn der Benutzer die Hilfetaste drückt, während der Cursor in dem Eingabefeld für den Startnamen steht.

Sie werden zuerst die Helproutine definieren und sie dann mit einem bestimmten Feld verknüpfen.

▶ Helproutine definieren

- 1 Markieren Sie im Library-Workspace die Library, die auch Ihr Programm enthält (d.h. markieren Sie den Knoten **TUTORIAL**).
- 2 Wählen Sie aus dem Kontextmenü den Befehl **New Source > Helproutine**.

Ein leerer Editor erscheint.

- 3 Geben Sie Folgendes ein:

```
WRITE 'Type the name of an employee'
END
```

- 4 Speichern Sie die Helproutine mit **STOW**.
Das Dialogfeld **Stow As** erscheint.
- 5 Geben Sie "HLP01" als Name für die Helproutine ein.
- 6 Wählen Sie die Befehlsschaltfläche **OK**.

Im Library-Workspace erscheint ein neuer Knoten mit dem Namen **Helproutines** als Unterknoten von **TUTORIAL**. Dieser Unterknoten enthält die Helproutine, die Sie eben gespeichert haben.

- 7 Schließen Sie das Editorfenster, in dem Sie die Helproutine eingegeben haben.

▶ Helproutine mit einem Feld in der Map verknüpfen

- 1 Kehren Sie zurück zum Map-Editor.
- 2 Markieren Sie das Datenfeld für den Startnamen.
- 3 Wählen Sie aus dem Kontextmenü den Befehl **Definition**.

Oder:

Klicken Sie das Datenfeld doppelt an.

Das Dialogfeld **Field Definition** erscheint.

- 4 Geben Sie im Textfeld **Helproutine** den Namen "'HLP01'" ein (einschließlich der einfachen Anführungszeichen).

Dies ist der Name, unter dem Sie die Helproutine gespeichert haben.

- 5 Wählen Sie die Befehlsschaltfläche **OK**.
- 6 Testen Sie die Map.
- 7 Geben Sie in der daraufhin erscheinenden Ausgabe ein Fragezeichen (?) im Eingabefeld für den Startnamen ein.

Der von Ihnen definierte Hilfetext wird jetzt in einem separaten Fenster angezeigt.

- 8 Drücken Sie EINGABE, um das Fenster zu schließen.
- 9 Geben Sie einen Punkt (.) in dem Feld für den Startnamen ein und drücken Sie EINGABE, um die Map zu verlassen.
- 10 Speichern Sie die Map mit STOW.
- 11 Schließen Sie das Map-Editor-Fenster.

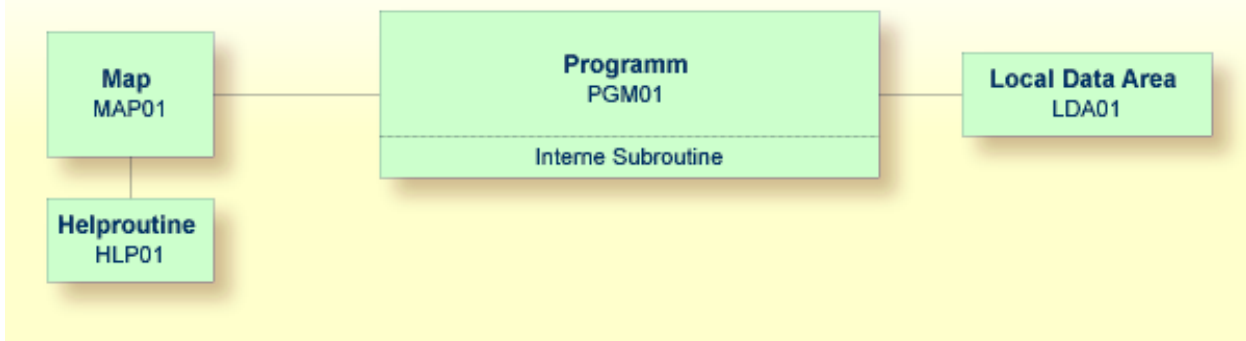
Sie können nun mit den nächsten Übungen fortfahren: *Local Data Areas*.

10 Local Data Areas

▪ Local Data Area erstellen	70
▪ Datenfelder definieren	71
▪ Datenfelder aus einem DDM importieren	74
▪ Local Data Area aus dem Programm aufrufen	75

Zurzeit werden die Felder, die in Ihrem Programm benutzt werden, mit `DEFINE DATA` im Programm selbst definiert. Es ist jedoch auch möglich, die Felddefinitionen in einer Local Data Area (LDA) außerhalb des Programms zu definieren und diese Local Data Area im `DEFINE DATA`-Statement des Programms lediglich namentlich zu referenzieren. Für die Wiederverwendbarkeit und zu Gunsten einer klaren Anwendungsstruktur ist es in der Regel besser, die Felder außerhalb eines Programms in einer Data Area zu definieren.

Sie werden jetzt die Informationen aus dem `DEFINE DATA`-Statement in eine Local Data Area verlagern. Wenn Sie mit den Übungen in diesem Kapitel fertig sind, wird Ihre Beispielanwendung aus den folgenden Modulen bestehen:



Local Data Area erstellen

Sie werden jetzt den Data-Area-Editor aufrufen, in dem Sie die erforderlichen Felder definieren.

▶ Local Data Area erstellen

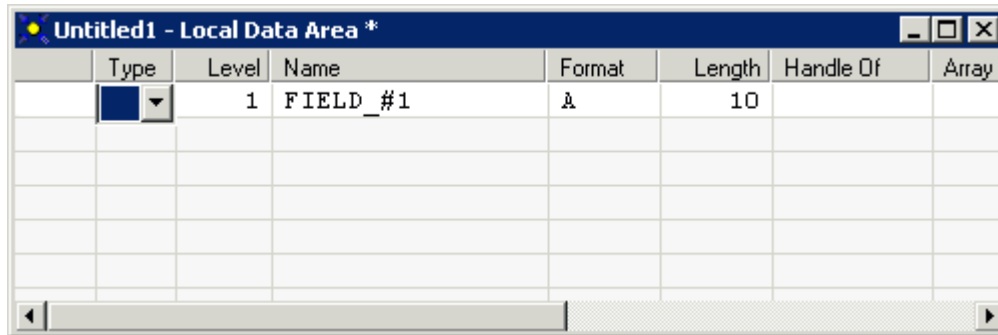
- 1 Markieren Sie im Library-Workspace die Library, die auch Ihr Programm enthält (d.h. markieren Sie den Knoten **TUTORIAL**).
- 2 Wählen Sie aus dem Kontextmenü den Befehl **New Source > Local Data Area**.

Oder:

Wählen Sie die folgende Schaltfläche in der Symbolleiste:



Ein Editorfenster erscheint.



Level, Name, Format und Länge (Length) sind in der ersten Zeile automatisch voreingestellt.

Datenfelder definieren

Sie werden jetzt die folgenden Felder definieren:

Level	Name	Format	Länge
1	#NAME-START	A	20
1	#NAME-END	A	20
1	#MARK	A	1

Dies sind die Benutzervariablen, die Sie zuvor im `DEFINE DATA`-Statement definiert haben.

Sie können die Datenfelder auf zwei verschiedene Arten definieren: entweder manuell im Editorfenster, wobei es in Ihrer Verantwortung liegt, das korrekte Format und die korrekte Länge des Datenfeldes zu definieren, oder mit dem Befehl **Import**, wobei Sie die Datenfelder einfach aus einer Liste auswählen und das korrekte Format und die korrekte Länge automatisch benutzt wird. Diese beiden Möglichkeiten sind unten beschrieben.

Der Zustand der folgenden Schaltfläche in der Insert-Symbolleiste des Data-Area-Editors weist auf die Einfügeposition hin. Wenn die Schaltfläche gedrückt aussieht, wird das neue Feld hinter dem markierten Feld eingefügt (dieser Zustand wird in diesem Tutorial vorausgesetzt); andernfalls wird das neue Feld vor dem markierten Feld eingefügt.



▶ **Datenfeld manuell im Editorfenster definieren**

- 1 Achten Sie darauf, dass die Spalte **Level** in der ersten Reihe den voreingestellten Wert "1" enthält.
- 2 Ändern Sie den voreingestellten Wert in der Spalte **Name** der ersten Reihe in "#NAME-START" ab.
- 3 Achten Sie darauf, dass die Spalte **Format** in der ersten Reihe den voreingestellten Wert "A" enthält.
- 4 Ändern Sie den voreingestellten Wert in der Spalte **Length** der ersten Reihe in "20" ab.

▶ **Datenfelder aus einem Programm importieren**

- 1 Markieren Sie im Editor die Zeile, die für #NAME-START erstellt wurde.



Anmerkung: Falls erforderlich, drücken Sie EINGABE oder ESC um vom Selektionsmodus für einzelne Zellen in den Selektionsmodus für komplette Reihen umzuschalten.

- 2 Wählen Sie aus dem Kontextmenü den Befehl **Import**.

Oder:

Wählen Sie die folgende Schaltfläche in der Symbolleiste:



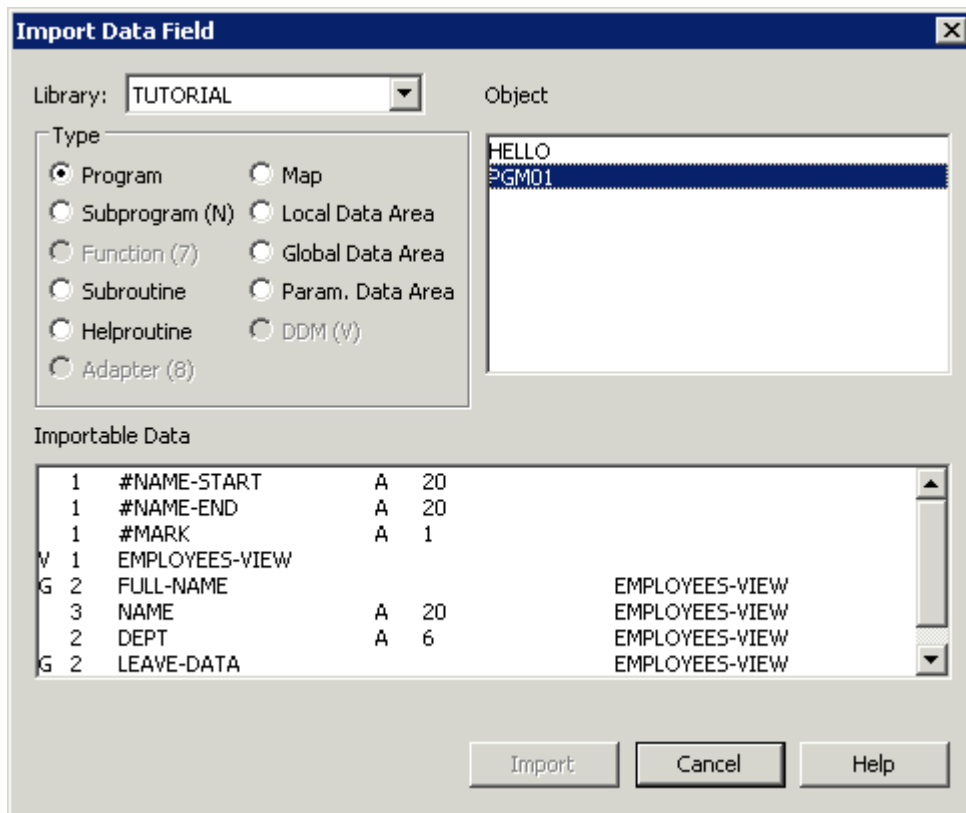
Das Dialogfeld **Import Data Field** erscheint.

- 3 Achten Sie darauf, dass **TUTORIAL** im Dropdown-Listenfeld **Library** markiert ist.
- 4 Markieren Sie das Optionsfeld **Program**.

Alle zurzeit in Ihrer Library definierten Programme werden jetzt im Listenfeld **Object** angezeigt.

- 5 Markieren Sie das Programm mit dem Namen PGM01.

Die importierbaren Felder werden jetzt unten im Dialogfeld angezeigt.



- 6 Drücken Sie STRG und markieren Sie die folgenden Felder:

#NAME - END

#MARK

- 7 Wählen Sie die Befehlsschaltfläche **Import**.

Die Befehlsschaltfläche **Cancel** im Dialogfeld **Import Data Field** trägt nun den Namen **Quit**.

- 8 Wählen Sie die Befehlsschaltfläche **Quit**, um das Dialogfeld **Import Data Field** zu schließen.

Die Felder #NAME - END und #MARK werden jetzt unter dem Feld #NAME - START im Editorfenster angezeigt.

Datenfelder aus einem DDM importieren

Sie werden jetzt dieselben Datenfelder importieren, die Sie zuvor im `DEFINE DATA`-Statement des Programms definiert hatten. Die Felder werden direkt aus einem Natural-Daten-View in den Data-Area-Editor eingelesen. Ein Daten-View referenziert die Datenbankfelder, die in einem DDM (Data Definition Module) definiert sind.

Im Data-Area-Editor werden die importierten Datenfelder unter dem zurzeit markierten Datenfeld eingefügt.

► Datenfelder aus einem DDM importieren

- 1 Markieren Sie im Editor die Zeile, die `#MARK` enthält.
- 2 Wählen Sie aus dem Kontextmenü den Befehl **Import**.

Das Fenster **Import Data Field** erscheint.

- 3 Wählen Sie aus dem Dropdown-Listefeld **Library** den Eintrag **SYSEXDDM**.

Das Optionsfeld **DDM** ist markiert.

- 4 Markieren Sie im Listefeld **Object** das Beispiel-DDM mit dem Namen **EMPLOYEES**.
- 5 Drücken Sie `STRG` und markieren Sie die folgenden importierbaren Felder:

```
PERSONNEL - ID  
FULL - NAME  
NAME  
DEPT  
LEAVE - DATA  
LEAVE - DUE
```



Anmerkung: Das Feld `PERSONNEL - ID` wird später benutzt, wenn Sie das Subprogramm erstellen.

- 6 Wählen Sie die Befehlsschaltfläche **Import**.

Das Dialogfeld **View Definition** erscheint.

- 7 Geben Sie denselben Namen an, den Sie vorher für den View definiert haben (d.h. `EMPLOYEES - VIEW`).
- 8 Wählen Sie die Befehlsschaltfläche **OK**.
- 9 Wählen Sie die Befehlsschaltfläche **Quit**, um das Dialogfeld **Import Data Field** zu schließen.

Die Local Data Area sollte nun folgendermaßen aussehen:

Type	Level	Name	Format	Length	Handle
	1	#NAME-START	A	20	
	1	#NAME-END	A	20	
	1	#MARK	A	1	
V	1	EMPLOYEES-VIEW			
	2	PERSONNEL-ID	A	8	
G	2	FULL-NAME			
	3	NAME	A	20	
	2	DEPT	A	6	
G	2	LEAVE-DATA			
	3	LEAVE-DUE	N	20	

Die Spalte **Type** gibt den Variablentyp an. Der View ist mit "V" gekennzeichnet und jede Gruppe ist mit "G" gekennzeichnet.



Anmerkung: Wenn das Kontrollkästchen **Expand/Collapse** in den Optionen für den Data-Area-Editor markiert wurde, erscheinen in der ersten Spalte Umschaltssymbole zum Ein- und Ausblenden der darunter liegenden Levels (für den View und jede Gruppe).

- 10 Speichern Sie die Local Data Area mit `STOW`.
- 11 Wenn Sie dazu aufgefordert werden, einen Namen für die Local Data Area einzugeben, geben Sie "LDA01" ein.

Im Library-Workspace erscheint ein neuer Knoten mit dem Namen **Local Data Areas** als Unterknoten von **TUTORIAL**. Dieser Unterknoten enthält die Local Data Area, die Sie eben gespeichert haben.

Local Data Area aus dem Programm aufrufen

Sobald eine Local Data Area mit `STOW` gespeichert wurde, kann sie aus einem Natural-Programm heraus aufgerufen werden.

Sie werden jetzt das `DEFINE DATA`-Statement in Ihrem Programm so ändern, dass die eben von Ihnen definierte Local Data Area benutzt wird.

Lassen Sie den Data-Area-Editor im Hintergrund geöffnet.

▶ Local Data Area in Ihrem Programm benutzen

- 1 Kehren Sie zum Programmeditor zurück.
- 2 Löschen Sie im DEFINE DATA-Statement alle Variablen zwischen LOCAL und END-DEFINE.
- 3 Referenzieren Sie die Local Data Area, indem Sie die LOCAL-Zeile folgendermaßen ändern:

```
LOCAL USING LDA01
```

Ihr Programm sollte nun folgendermaßen aussehen:

```
DEFINE DATA
  LOCAL USING LDA01
END-DEFINE
*
RP1. REPEAT
*
  INPUT USING MAP 'MAP01'
*
  IF #NAME-START = '.' THEN
    ESCAPE BOTTOM (RP1.)
  END-IF
*
  IF #NAME-END = ' ' THEN
    MOVE #NAME-START TO #NAME-END
  END-IF
*
  RD1. READ EMPLOYEES-VIEW BY NAME
    STARTING FROM #NAME-START
    ENDING AT #NAME-END
*
    IF LEAVE-DUE >= 20 THEN
      PERFORM MARK-SPECIAL-EMPLOYEES
    ELSE
      RESET #MARK
    END-IF
*
    DISPLAY NAME 3X DEPT 3X LEAVE-DUE 3X '>=20' #MARK
*
  END-READ
*
  IF *COUNTER (RD1.) = 0 THEN
    REINPUT 'No employees meet your criteria.'
  END-IF
*
END-REPEAT
*
DEFINE SUBROUTINE MARK-SPECIAL-EMPLOYEES
  MOVE '**' TO #MARK
END-SUBROUTINE
```

```
*  
END
```

- 4 Führen Sie das Programm mit `RUN` aus.
- 5 Um zu überprüfen, ob das Ergebnis immer noch dasselbe ist wie vorher (als noch keine Local Data Area mit `DEFINE DATA` referenziert wurde), geben Sie "JONES" als Startname ein und drücken Sie `EINGABE`.
- 6 Drücken Sie `ESC`, um das Ausgabefenster zu schließen.
- 7 Speichern Sie das Programm mit `STOW`.

Sie können nun mit den nächsten Übungen fortfahren: *Global Data Areas*.

11 Global Data Areas

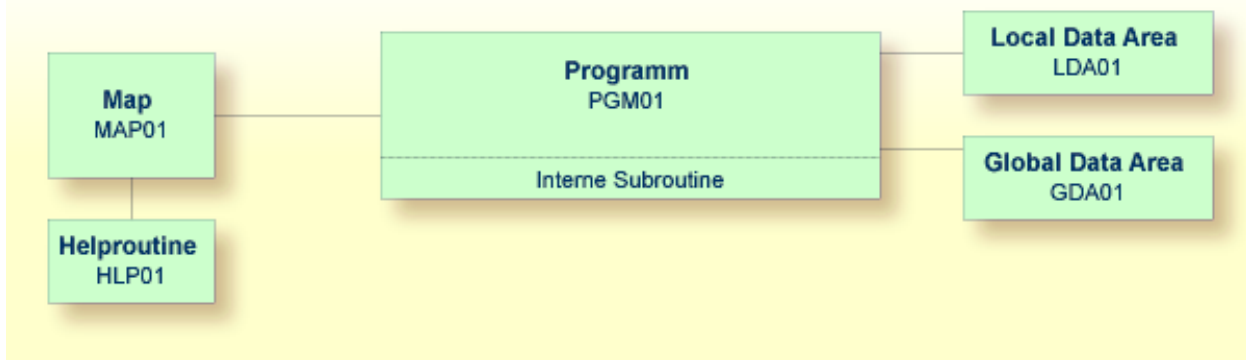
- Global Data Area mit Hilfe einer bestehenden Local Data Area erstellen 80
- Local Data Area anpassen 82
- Global Data Area aus dem Programm aufrufen 83

Die Daten in einer Global Data Area (GDA) können von mehreren Programmen, externen Subroutinen und Helproutinen benutzt werden.

Jede Änderung eines Datenelements in einer Global Data Area hat Auswirkungen auf alle Natural-Objekte, die diese Global Data Area referenzieren. Aus diesem Grund müssen Sie nach der Änderung einer Global Data Area alle zuvor erstellten Natural-Objekte, die diese Global Data Area referenzieren, noch einmal mit `STOW` speichern. Die Reihenfolge, in der die Objekte mit `STOW` gespeichert werden, ist wichtig. Sie müssen zuerst die Global Data Area mit `STOW` speichern und danach das Programm. Wenn Sie das Programm zuerst mit `STOW` speichern wollen und dann erst die Global Data Area, dann kann das Programm nicht gespeichert werden, weil die neuen Elemente in der Global Data Area nicht gefunden werden.

Sie werden jetzt eine Global Data Area erstellen, die von Ihrem Programm benutzt wird und auch von einer externen Subroutine, die Sie später erstellen werden. Als Basis für Ihre Global Data Area übernehmen Sie einige Informationen aus der Local Data Area, die Sie soeben erstellt haben.

Wenn Sie mit den Übungen in diesem Kapitel fertig sind, wird Ihre Beispielanwendung aus den folgenden Modulen bestehen:



Global Data Area mit Hilfe einer bestehenden Local Data Area erstellen

Sie können eine neue Data Area erstellen, indem Sie eine bestehende Data Area editieren und sie dann unter einem anderen Namen und einem anderen Typ abspeichern. Die ursprüngliche Data Area bleibt hierbei unverändert und die neue Data Area kann sofort editiert werden. Da die Felder `#NAME - START` und `#NAME - END` nicht in der Global Data Area benötigt werden, werden Sie sie entfernen.



Anmerkung: Es ist auch möglich, eine Global Data Area zu erstellen, indem Sie aus dem Menü **Object** den Befehl **New > Global Data Area** wählen.

► Global Data Area erstellen

- 1 Kehren Sie zur Local Data Area zurück.
- 2 Wählen Sie aus dem Menü **Object** den Befehl **Save As**.

Das Dialogfeld **Save As** erscheint.

- 3 Geben Sie "GDA01" als Name für die Global Data Area ein.
- 4 Achten Sie darauf, dass im Library-Workspace die Library markiert ist, die auch Ihr Programm enthält (d.h. der Knoten **TUTORIAL**).
- 5 Wählen Sie aus dem Dropdown-Listefeld **Type** den Eintrag **Global**.
- 6 Wählen Sie die Befehlsschaltfläche **OK**.

Der neue Name und Typ werden nun in der Titelleiste des Editorfensters angezeigt. Im Library-Workspace wird die neue Global Data Area im Knoten **Global Data Areas** angezeigt.

- 7 Drücken Sie STRG und markieren Sie die folgenden Felder:

#NAME - START

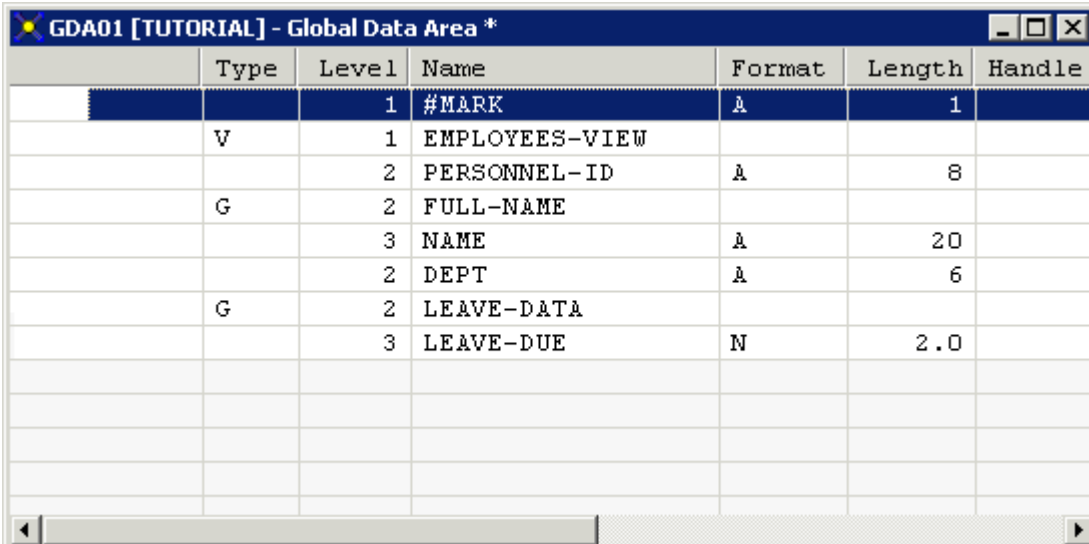
#NAME - END

- 8 Wählen Sie aus dem Kontextmenü den Befehl **Delete**.

Oder:

Drücken Sie ENTF.

Die Global Data Area sollte nun folgendermaßen aussehen:



	Type	Level	Name	Format	Length	Handle
		1	#MARK	A	1	
	V	1	EMPLOYEES-VIEW			
		2	PERSONNEL-ID	A	8	
	G	2	FULL-NAME			
		3	NAME	A	20	
		2	DEPT	A	6	
	G	2	LEAVE-DATA			
		3	LEAVE-DUE	N	2.0	

- 9 Speichern Sie die Global Data Area mit STOW.

Die Global Data Area kann jetzt von Ihrem Programm gefunden werden, und auch von der externen Subroutine, die Sie später anlegen werden.

- 10 Schließen Sie das Editorfenster für die Global Data Area.

Local Data Area anpassen

Die Felder, die jetzt in der Global Data Area enthalten sind, werden nicht mehr in der Local Data Area benötigt. Deshalb werden Sie jetzt alle Felder außer #NAME-START und #NAME-END aus der Local Data Area entfernen.

► Felder entfernen

- 1 Markieren Sie die Local Data Area LDA01 im Library-Workspace und wählen Sie aus dem Kontextmenü den Befehl **Open**.

Oder:

Klicken Sie die Local Data Area LDA01 im Library-Workspace doppelt an.

- 2 Markieren Sie im daraufhin erscheinenden Data-Area-Editor alle Felder außer #NAME-START und #NAME-END.

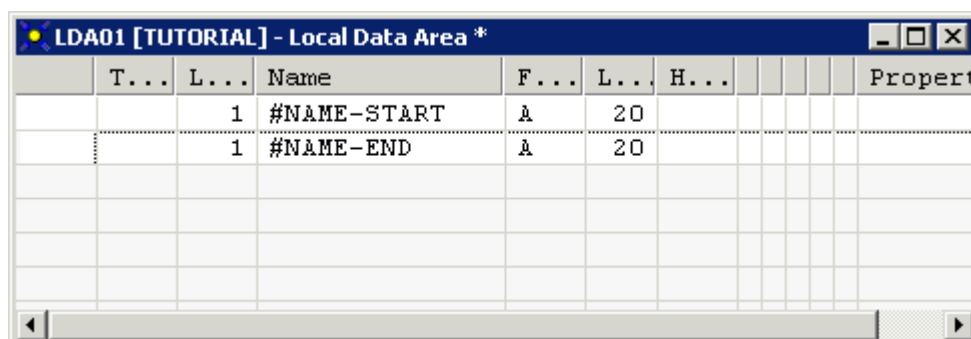
- 3 Wählen Sie aus dem Kontextmenü den Befehl **Delete**.

Oder:

Drücken Sie ENTF.

- 4 Speichern Sie die geänderte Local Data Area mit STOW.

Die Local Data Area sollte nun folgendermaßen aussehen:



T...	L...	Name	F...	L...	H...	Propert
	1	#NAME-START	A	20		
	1	#NAME-END	A	20		

Global Data Area aus dem Programm aufrufen

Sobald eine Global Data Area mit `STOW` gespeichert wurde, kann sie aus einem Natural-Programm heraus aufgerufen werden.

Sie werden jetzt das `DEFINE DATA`-Statement in Ihrem Programm so ändern, dass die eben von Ihnen definierte Global Data Area benutzt wird.

Lassen Sie den Data-Area-Editor im Hintergrund geöffnet.

▶ Global Data Area in Ihrem Programm benutzen

- 1 Kehren Sie zum Programmeditor zurück.
- 2 Geben Sie Folgendes in der Zeile über `LOCAL USING LDA01` ein:

```
GLOBAL USING GDA01
```

Eine Global Data Area muss immer vor einer Local Data Area angegeben werden. Andernfalls tritt ein Fehler auf.

Ihr Programm sollte nun folgendermaßen aussehen:

```
DEFINE DATA
  GLOBAL USING GDA01
  LOCAL USING LDA01
END-DEFINE
*
RP1. REPEAT
*
  INPUT USING MAP 'MAP01'
*
  IF #NAME-START = '.' THEN
    ESCAPE BOTTOM (RP1.)
  END-IF
*
  IF #NAME-END = ' ' THEN
    MOVE #NAME-START TO #NAME-END
  END-IF
*
  RD1. READ EMPLOYEES-VIEW BY NAME
    STARTING FROM #NAME-START
    ENDING AT #NAME-END
*
  IF LEAVE-DUE >= 20 THEN
    PERFORM MARK-SPECIAL-EMPLOYEES
  ELSE
```

```
        RESET #MARK
        END-IF
*
        DISPLAY NAME 3X DEPT 3X LEAVE-DUE 3X '>=20' #MARK
*
        END-READ
*
        IF *COUNTER (RD1.) = 0 THEN
            REINPUT 'No employees meet your criteria.'
        END-IF
*
        END-REPEAT
*
        DEFINE SUBROUTINE MARK-SPECIAL-EMPLOYEES
            MOVE '**' TO #MARK
        END-SUBROUTINE
*
        END
```

- 3 Führen Sie das Programm mit RUN aus.
- 4 Um zu überprüfen, ob das Ergebnis immer noch dasselbe ist wie vorher (als noch keine Global Data Area mit DEFINE DATA referenziert wurde), geben Sie "JONES" als Startname ein und drücken Sie EINGABE.
- 5 Drücken Sie ESC, um das Ausgabefenster zu schließen.
- 6 Speichern Sie das Programm mit STOW.

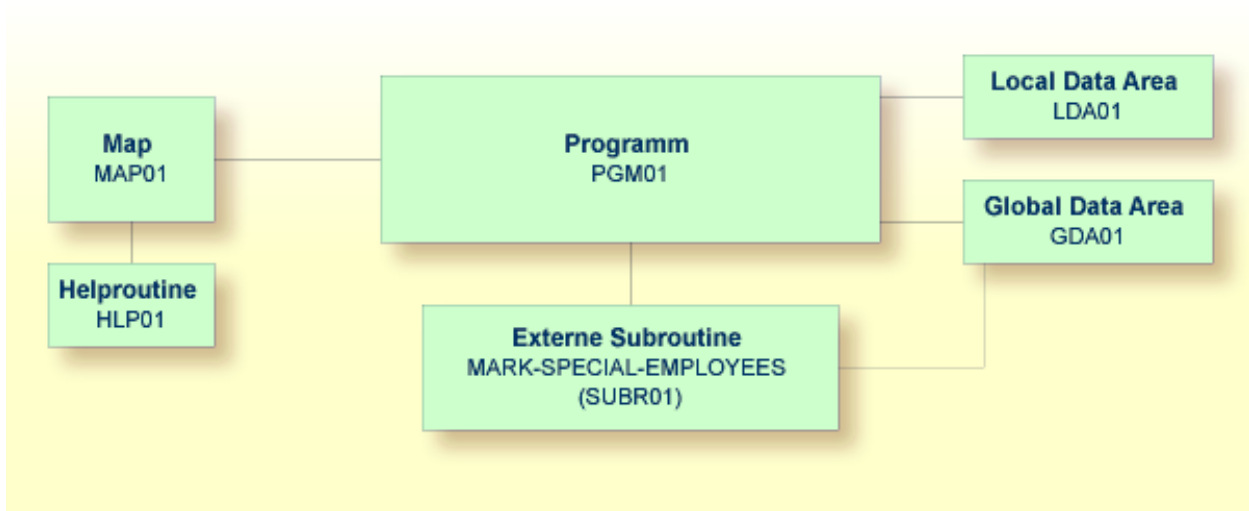
Sie können nun mit den nächsten Übungen fortfahren: [Externe Subroutinen](#).

12 Externe Subroutinen

- Externe Subroutine erstellen 86
- Externe Subroutine aus dem Programm aufrufen 87

Bis jetzt ist die Subroutine `MARK-SPECIAL-EMPLOYEES` mit einem `DEFINE SUBROUTINE`-Statement im Programm selbst definiert. Sie werden diese Subroutine jetzt als separates Objekt definieren, das sich außerhalb des Programms befindet.

Wenn Sie mit den Übungen in diesem Kapitel fertig sind, wird Ihre Beispielanwendung aus den folgenden Modulen bestehen:



Externe Subroutine erstellen

Sie werden jetzt einen Editor aufrufen, in dem Sie den Code für die externe Subroutine eingeben.

Das `DEFINE SUBROUTINE`-Statement wird in der externen Subroutine genauso kodiert wie in der internen Subroutine des Programms.

► Externe Subroutine erstellen

- 1 Markieren Sie im Library-Workspace die Library, die auch Ihr Programm enthält (d.h. markieren Sie den Knoten **TUTORIAL**).
- 2 Wählen Sie aus dem Kontextmenü den Befehl **New Source > Subroutine**.

Ein leeres Editorfenster erscheint.

3 Geben Sie Folgendes ein:

```

DEFINE DATA
  GLOBAL USING GDA01
  LOCAL USING LDA01
END-DEFINE
*
DEFINE SUBROUTINE MARK-SPECIAL-EMPLOYEES
  MOVE '*' TO #MARK
END-SUBROUTINE
*
END

```

4 Speichern Sie die Subroutine mit STOW.

Das Dialogfeld **Stow As** erscheint.

5 Geben Sie "SUBR01" als Name für die externe Subroutine ein und wählen Sie die Befehlsschaltfläche **OK**.

Im Library-Workspace wird die neue externe Subroutine im Knoten **Subroutines** angezeigt. Im Logical-View wird der Name der Subroutine so angezeigt wie im Code: MARK-SPECIAL-EMPLOYEES. In den anderen Views wird der Name SUBR01 angezeigt.

6 Schließen Sie das Editorfenster, in dem Sie die externe Subroutine eingegeben haben.

Externe Subroutine aus dem Programm aufrufen

Mit dem `PERFORM`-Statement kann man sowohl interne als auch externe Subroutinen aufrufen. Wenn im Programm keine interne Subroutine gefunden wird, versucht Natural automatisch, eine externe Subroutine mit demselben Namen auszuführen. Natural sucht hierbei nach dem Namen, der im Subroutinencode definiert wurde (d.h. dem Subroutinennamen). Natural sucht nicht nach dem Namen, den Sie beim Speichern der Subroutine angegeben haben (d.h. dem Objektnamen).

Jetzt, nachdem Sie eine externe Subroutine erstellt haben, müssen Sie die interne Subroutine (die denselben Namen hat wie die externe Subroutine) aus Ihrem Programm entfernen.

▶ Externe Subroutine in Ihrem Programm benutzen

1 Kehren Sie zum Programmeditor zurück.

2 Entfernen Sie die folgenden Zeilen:

```
DEFINE SUBROUTINE MARK-SPECIAL-EMPLOYEES
  MOVE '*' TO #MARK
END-SUBROUTINE
```

Ihr Programm sollte nun folgendermaßen aussehen:

```
DEFINE DATA
  GLOBAL USING GDA01
  LOCAL USING LDA01
END-DEFINE
*
RP1. REPEAT
*
  INPUT USING MAP 'MAP01'
*
  IF #NAME-START = '.' THEN
    ESCAPE BOTTOM (RP1.)
  END-IF
*
  IF #NAME-END = ' ' THEN
    MOVE #NAME-START TO #NAME-END
  END-IF
*
  RD1. READ EMPLOYEES-VIEW BY NAME
    STARTING FROM #NAME-START
    ENDING AT #NAME-END
*
  IF LEAVE-DUE >= 20 THEN
    PERFORM MARK-SPECIAL-EMPLOYEES
  ELSE
    RESET #MARK
  END-IF
*
  DISPLAY NAME 3X DEPT 3X LEAVE-DUE 3X '>=20' #MARK

END-READ
*
IF *COUNTER (RD1.) = 0 THEN
  REINPUT 'No employees meet your criteria.'
END-IF
*
END-REPEAT
*
END
```

3 Führen Sie das Programm mit RUN aus.

4 Geben Sie "JONES" als Startname ein und drücken Sie EINGABE.

Die daraufhin erscheinende Liste sollte immer noch einen Stern bei jedem Mitarbeiter anzeigen, der 20 oder mehr Urlaubstage hat.

- 5 Drücken Sie ESC, um das Ausgabefenster zu schließen.
- 6 Speichern Sie das Programm mit STOW.

Sie können nun mit den nächsten Übungen fortfahren: *Subprogramme*.

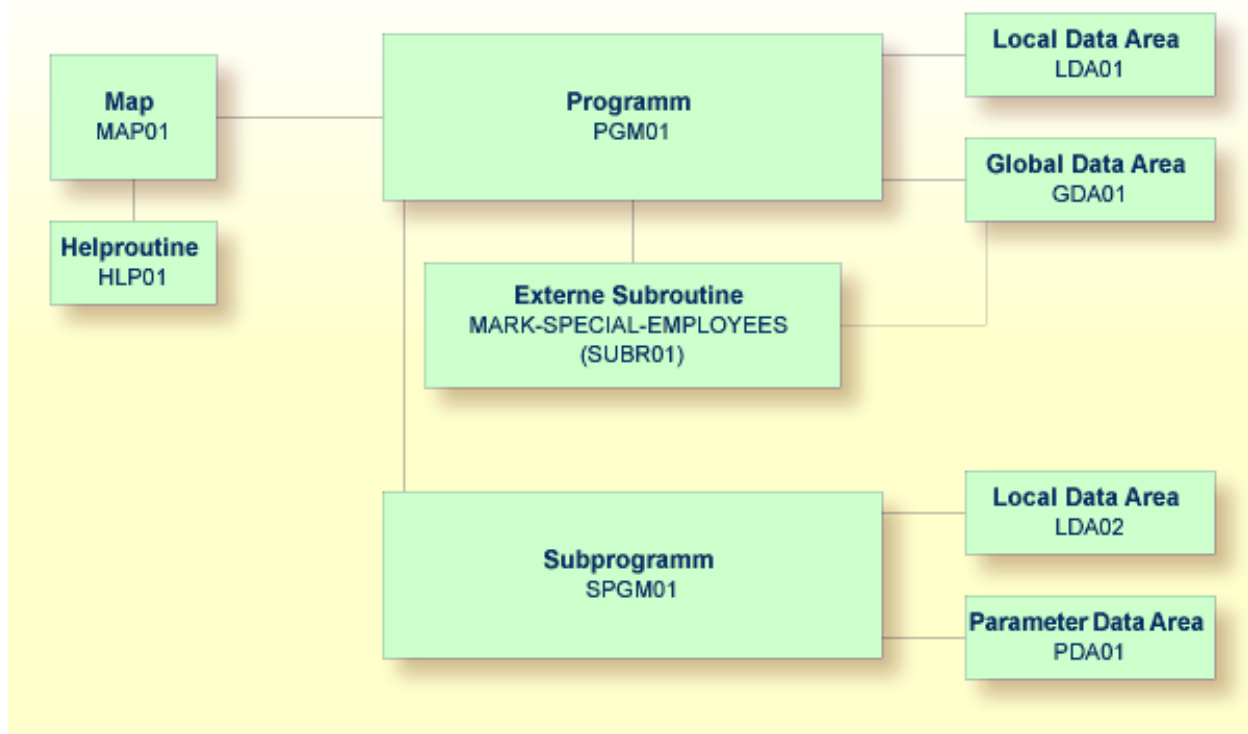
13 Subprogramme

- Local Data Area ändern 92
- Parameter Data Area mit Hilfe einer bestehenden Local Data Area erstellen 94
- Eine zweite Local Data Area mit einem anderen View erstellen 95
- Subprogramm erstellen 96
- Subprogramm aus dem Programm aufrufen 97

Sie werden Ihr Programm jetzt durch ein `CALLNAT`-Statement erweitern, mit dem ein Subprogramm aufgerufen wird. In dem Subprogramm bilden die Mitarbeiter, die vom Hauptprogramm gefunden wurden, die Grundlage für eine `FIND`-Anfrage in der `VEHICLES`-Datei. Diese Datei ist ebenfalls Bestandteil der Demodatenbank. In der Ausgabe werden dann Fahrzeuginformationen aus dem Subprogramm und Mitarbeiterinformationen aus dem Hauptprogramm zu sehen sein.

Das neue Subprogramm benötigt eine weitere Local Data Area und eine Parameter Data Area.

Wenn Sie mit den Übungen in diesem Kapitel fertig sind, wird Ihre Beispielanwendung aus den folgenden Modulen bestehen:



Local Data Area ändern

Sie werden jetzt weitere Felder in die Local Data Area einfügen, die Sie zuvor erstellt haben. Diese Felder werden von dem Subprogramm benutzt, das Sie später erstellen werden.

► Weitere Felder in die Local Data Area einfügen

- 1 Kehren Sie zur Local Data Area zurück.
- 2 Markieren Sie die Zeile, die `#NAME - END` enthält.

- 3 Wählen Sie aus dem Kontextmenü den Befehl **Insert > Data Field**.

Oder:

Wählen Sie die folgende Schaltfläche in der Symbolleiste:



Das Dialogfeld **Data Field Definition** erscheint.

- 4 Definieren Sie die folgenden Felder:

Level	Name	Format	Länge
1	#PERS-ID	A	8
1	#MAKE	A	20
1	#MODEL	A	20

Wählen Sie die Befehlsschaltfläche **Add**, nachdem Sie ein Feld definiert haben.

- 5 Nachdem Sie alle Felder definiert haben, wählen Sie die Befehlsschaltfläche **Quit**.

Die Local Data Area sollte nun folgendermaßen aussehen:

T...	L...	Name	F...	L...	H...	Propert
	1	#NAME-START	A	20		
	1	#NAME-END	A	20		
	1	#PERS-ID	A	8		
	1	#MAKE	A	20		
	1	#MODEL	A	20		

- 6 Speichern Sie die Local Data Area mit STOW.

Parameter Data Area mit Hilfe einer bestehenden Local Data Area erstellen

Eine Parameter Data Area (PDA) wird zur Angabe der Datenparameter benutzt, die zwischen Ihrem Natural-Programm und dem Subprogramm (das Sie später noch erstellen werden) ausgetauscht werden sollen. Die Parameter Data Area wird im Subprogramm referenziert.

Mit kleinen Änderungen kann Ihre Local Data Area zur Erstellung der Parameter Data Area benutzt werden: Sie werden zwei Datenfelder in der Local Data Area löschen und die so veränderte Local Data Area als Parameter Data Area speichern. Die ursprüngliche Local Data Area wird hierdurch nicht verändert.

 **Anmerkung:** Es ist auch möglich, eine Parameter Data Area zu erstellen, indem Sie aus dem Menü **Object** den Befehl **New > Parameter Data Area** wählen.

▶ Parameter Data Area erstellen

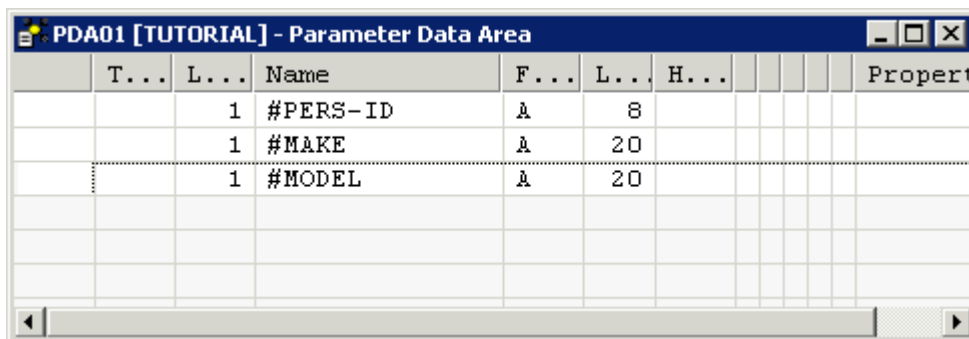
- 1 Löschen Sie die Felder **#NAME - START** und **#NAME - END** in der Local Data Area.
- 2 Wählen Sie aus dem Menü **Object** den Befehl **Save As**.

Das Dialogfeld **Save As** erscheint. Geben Sie "PDA01" als Name für die Parameter Data Area ein.

- 3 Achten Sie darauf, dass die Library markiert ist, die auch Ihr Programm enthält (d.h. der Knoten **TUTORIAL**).
- 4 Wählen Sie aus dem Dropdown-Listenfeld **Type** den Eintrag **Parameter**.
- 5 Wählen Sie die Befehlsschaltfläche **OK**.

Der neue Name und Typ werden nun in der Titelleiste des Editorfensters angezeigt. Im Library-Workspace wird die neue Parameter Data Area im Knoten **Parameter Data Areas** angezeigt.

Die Parameter Data Area sollte nun folgendermaßen aussehen:



T...	L...	Name	F...	L...	H...	Propert
	1	#PERS-ID	A	8		
	1	#MAKE	A	20		
	1	#MODEL	A	20		

- 6 Speichern Sie die Parameter Data Area mit `STOW`.
- 7 Schließen Sie das Editorfenster für die Parameter Data Area.

Eine zweite Local Data Area mit einem anderen View erstellen

Sie werden jetzt eine zweite Local Data Area erstellen und Felder aus dem DDM für die Datenbankdatei `VEHICLES` importieren. Dieses DDM steht auch in der System-Library `SYSEXDDM` zur Verfügung.

Diese Local Data Area wird dann in Ihrem Subprogramm referenziert.

▶ Local Data Area erstellen

- 1 Markieren Sie im Library-Workspace die Library, die auch Ihr Programm enthält (d.h. markieren Sie den Knoten **TUTORIAL**).
- 2 Wählen Sie aus dem Kontextmenü den Befehl **New Source > Local Data Area**.

Oder:

Wählen Sie die folgende Schaltfläche in der Symbolleiste:



Ein Editorfenster erscheint.

- 3 Wählen Sie aus dem Menü **Insert** den Befehl **Import**.

Das Dialogfeld **Import Data Field** erscheint.

- 4 Wählen Sie aus dem Dropdown-Listenfeld **Library** den Eintrag **SYSEXDDM**.

Markieren Sie das Optionsfeld **DDM**.

Markieren Sie im Listenfeld **Object** das Beispiel-DDM mit dem Namen **VEHICLES**.

- 5 Drücken Sie `STRG` und markieren Sie die folgenden importierbaren Felder:

```
PERSONNEL - ID
CAR-DETAILS
MAKE
MODEL
```

- 6 Wählen Sie die Befehlsschaltfläche **Import**.

Das Dialogfeld **View Definition** erscheint.

- 7 Geben Sie "VEHICLES-VIEW" als Name für den View an.
- 8 Wählen Sie die Befehlsschaltfläche **OK**.
- 9 Wählen Sie die Befehlsschaltfläche **Quit**, um das Dialogfeld **Import Data Field** zu schließen.
- 10 Speichern Sie die neue Local Data Area mit STOW.
- 11 Wenn Sie dazu aufgefordert werden, einen Namen für die Local Data Area einzugeben, geben Sie "LDA02" ein und wählen Sie die Befehlsschaltfläche **OK**.

Die Local Data Area sollte nun folgendermaßen aussehen:

Type	Level	Name	Format	Length	Handl
V	1	VEHICLES-VIEW			
	2	PERSONNEL-ID	A	8	
G	2	CAR-DETAILS			
	3	MAKE	A	20	
	3	MODEL	A	20	

- 12 Schließen Sie das Editorfenster für die Local Data Area.

Subprogramm erstellen

Sie werden jetzt ein Subprogramm erstellen, das mit Hilfe einer Parameter Data Area und einer Local Data Area Informationen aus der VEHICLES-Datei abrufen. Das Programm PGM01 übergibt das Personalkennzeichen (PERSONNEL-ID) an das Subprogramm. Das Subprogramm benutzt dieses Kennzeichen für die Suche in der VEHICLES-Datei.

▶ Subprogramm erstellen

- 1 Markieren Sie im Library-Workspace die Library, die auch Ihr Programm enthält (d.h. markieren Sie den Knoten **TUTORIAL**).
- 2 Wählen Sie aus dem Kontextmenü den Befehl **New Source > Subprogramm**.

Ein Editorfenster erscheint.

3 Geben Sie Folgendes ein:

```

DEFINE DATA
  PARAMETER USING PDA01
  LOCAL USING LDA02
END-DEFINE
*
FD1. FIND (1) VEHICLES-VIEW
  WITH PERSONNEL-ID = #PERS-ID
  MOVE MAKE (FD1.) TO #MAKE
  MOVE MODEL (FD1.) TO #MODEL
  ESCAPE BOTTOM
END-FIND
*
END

```

Dieses Subprogramm gibt die folgenden Informationen an ein bestimmtes Personalkennzeichen zurück: die Marke und das Modell des Firmenfahrzeuges eines Mitarbeiters.

Auf Grund des Suchkriteriums #PERS-ID wählt das FIND-Statement eine Reihe von Datensätzen (hier: einen Datensatz) aus der Datenbank aus.

Das Feld #PERS-ID des Subprogramms erhält den Wert von PERSONNEL-ID, der vom Programm PGM01 übergeben wurde. Das Subprogramm benutzt diesen Wert für die Suche in der VEHICLES-Datei.

- 4 Speichern Sie das Subprogramm mit STOW.
- 5 Wenn Sie dazu aufgefordert werden, einen Namen für das Subprogramm einzugeben, geben Sie "SPGM01" ein und wählen Sie die Befehlsschaltfläche **OK**.
- 6 Schließen Sie das Editorfenster für das Subprogramm.

Subprogramm aus dem Programm aufrufen

Ein Subprogramm wird mit einem CALLNAT-Statement aus dem Hauptprogramm aufgerufen. Ein Subprogramm kann nur mit einem CALLNAT-Statement aufgerufen werden; es kann selbst nicht ausgeführt werden. Ein Subprogramm kann nicht auf die Global Data Area zugreifen, die von dem aufrufenden Objekt benutzt wird.

Die Daten werden vom Hauptprogramm an das Subprogramm mit Hilfe von Parametern übergeben, die im Subprogramm mit einem DEFINE DATA PARAMETER-Statement referenziert werden.

Die Variablen, die in der Parameter Data Area des Subprogramms definiert sind, müssen nicht unbedingt dieselben Namen haben wie die Variablen im CALLNAT-Statement. Sie müssen nur in der Reihenfolge, im Format und in der Länge übereinstimmen.

Sie werden jetzt Ihr Hauptprogramm ändern, damit es das eben von Ihnen erstellte Subprogramm benutzen kann.

► Subprogramm in Ihrem Hauptprogramm benutzen

- 1 Kehren Sie zum Programmeditor zurück.
- 2 Geben Sie Folgendes direkt über dem `DISPLAY`-Statement ein:

```
RESET #MAKE #MODEL  
CALLNAT 'SPGM01' PERSONNEL-ID #MAKE #MODEL
```

Das `RESET`-Statement setzt die Werte für `#MAKE` und `#MODEL` auf Nullwerte.

- 3 Löschen Sie die Zeile, die das `DISPLAY`-Statement enthält, und ersetzen Sie sie mit Folgendem:

```
WRITE TITLE  
  / '*** PERSONS WITH 20 OR MORE DAYS LEAVE DUE ***'  
  / '*** ARE MARKED WITH AN ASTERISK ***'//  
*  
DISPLAY 1X '//NAME' NAME  
        1X '//DEPT' DEPT  
        1X '//LV/DUE' LEAVE-DUE  
        ' ' #MARK  
        1X '//MAKE' #MAKE  
        1X '//MODEL' #MODEL
```

Der mit dem `WRITE TITLE`-Statement definierte Text wird bei der Ausgabe oben auf jeder Seite angezeigt. Das `WRITE TITLE`-Statement setzt den Standardseitentitel außer Kraft: die Informationen, die bisher oben auf jeder Seite angezeigt wurden (Seitenzahl, Datum und Uhrzeit) werden nicht mehr angezeigt. Jeder Schrägstrich (/) sorgt dafür, dass die nachfolgenden Informationen in einer neuen Zeile angezeigt werden.

Da das Subprogramm jetzt zusätzliche Fahrzeuginformationen ausgibt, muss die Größe der Spalten in der Ausgabe angepasst werden. Die Spalten erhalten kürzere Überschriften. Die Spalte, in der der Stern angezeigt wird (`#MARK`), bekommt überhaupt keine Überschrift. Zwischen den einzelnen Spalten wird je ein Leerzeichen eingefügt (1X). Jeder Schrägstrich in einer Spaltenüberschrift lässt die nachfolgenden Informationen in derselben Spalte in einer neuen Zeile erscheinen.

Ihr Programm sollte nun folgendermaßen aussehen:

```

DEFINE DATA
  GLOBAL USING GDA01
  LOCAL USING LDA01
END-DEFINE
*
RP1. REPEAT
*
  INPUT USING MAP 'MAP01'
*
  IF #NAME-START = '.' THEN
    ESCAPE BOTTOM (RP1.)
  END-IF
*
  IF #NAME-END = ' ' THEN
    MOVE #NAME-START TO #NAME-END
  END-IF
*
  RD1. READ EMPLOYEES-VIEW BY NAME
    STARTING FROM #NAME-START
    ENDING AT #NAME-END
*
  IF LEAVE-DUE >= 20 THEN
    PERFORM MARK-SPECIAL-EMPLOYEES
  ELSE
    RESET #MARK
  END-IF
*
  RESET #MAKE #MODEL
  CALLNAT 'SPGM01' PERSONNEL-ID #MAKE #MODEL
*
  WRITE TITLE
  / '*** PERSONS WITH 20 OR MORE DAYS LEAVE DUE ***'
  / '*** ARE MARKED WITH AN ASTERISK ***'//
*
  DISPLAY 1X '//N A M E' NAME
          1X '//DEPT' DEPT
          1X '//LV/DUE' LEAVE-DUE
          ' ' #MARK
          1X '//MAKE' #MAKE
          1X '//MODEL' #MODEL
*
  END-READ
*
  IF *COUNTER (RD1.) = 0 THEN
    REINPUT 'No employees meet your criteria.'
  END-IF
*
END-REPEAT

```

```
*  
END
```

- 4 Führen Sie das Programm mit RUN aus.
- 5 Geben Sie "JONES" als Startname ein und drücken Sie EINGABE.

Die daraufhin erscheinende Liste sollte der folgenden Ausgabe ähneln:

```
*** PERSONS WITH 20 OR MORE DAYS LEAVE DUE ***  
*** ARE MARKED WITH AN ASTERISK ***  
  
      N A M E           DEPT  LV  DUE  MAKE           MODEL  
-----  
JONES           SALE30  25 * CHRYSLER           IMPERIAL  
JONES           MGMT10  34 * CHRYSLER           PLYMOUTH  
JONES           TECH10  11  GENERAL MOTORS     CHEVROLET  
JONES           MGMT10  18  FORD                ESCORT  
JONES           TECH10  21 * GENERAL MOTORS     BUICK  
JONES           SALE00  30 * GENERAL MOTORS     PONTIAC  
JONES           SALE20  14  GENERAL MOTORS     OLDSMOBILE  
JONES           COMP12  26 * DATSUN             SUNNY  
JONES           TECH02  25 * FORD                ESCORT 1.3
```

- 6 Drücken Sie ESC, um das Ausgabefenster zu schließen.
- 7 Speichern Sie das Programm mit STOW.

Sie haben das Tutorial jetzt erfolgreich abgeschlossen.