Dialog-Gestaltung

Dieses Kapitel beschreibt, wie Benutzeroberflächen erstellt werden können, die einen einfachen und flexiblen Benutzerdialog ermöglichen.

Folgende Punkte werden behandelt:

- Feldabhängige Verarbeitung
- Einfachere Programmierung
- Zeilenabhängige Verarbeitung
- Spaltenabhängige Verarbeitung
- Verarbeitung aufgrund von Funktionstasten
- Verarbeitung aufgrund der Namen von Funktionstasten
- Verarbeitung von Daten außerhalb des aktiven Fensters
- Daten vom Bildschirm kopieren
- Statements REINPUT/REINPUT FULL
- Objektorientierte Datenverarbeitung — der Natural-Kommando-Prozessor

---

Feldabhängige Verarbeitung

Feldabhängige Verarbeitung — *CURS-FIELD und POS(*fieldname*)

Mit der Systemvariablen `*CURS-FIELD` zusammen mit der Systemfunktion `POS(`*`field-name`*`)` können Sie eine Verarbeitung davon abhängig machen, in welchem Feld der Cursor sich gerade befindet, wenn der Benutzer EINGABE drückt.

`*CURS-FIELD` enthält die interne Identifikation des Feldes, in dem der Cursor sich gerade befindet; diese Systemvariable kann nicht allein, sondern nur zusammen mit `POS(`*`field-name`*`)` benutzt werden.

Mit `*CURS-FIELD` und `POS(`*`field-name`*`)` können Sie z.B. dem Benutzer die Möglichkeit geben, eine Funktion auszuwählen, indem er einfach den Cursor auf ein bestimmtes Feld platziert und EINGABE drückt.

Das Beispiel unten demonstriert eine solche Anwendung:

```
DEFINE DATA LOCAL
1 #EMP (A1)
1 #CAR (A1)
1 #CODE (N1)
END-DEFINE
*
INPUT USING MAP 'CURS'
*
DECIDE FOR FIRST CONDITION
```

```
  WHEN *CURS-FIELD = POS(#EMP) OR #EMP = 'X' OR #CODE = 1
    FETCH 'LISTEMP'
  WHEN *CURS-FIELD = POS(#CAR) OR #CAR = 'X' OR #CODE = 2
    FETCH 'LISTCAR'
   WHEN NONE
    REINPUT 'PLEASE MAKE A VALID SELECTION'
END-DECIDE
END
```

Und das Ergebnis:

```
                                SAMPLE MAP

                    Please select a function


                    1.) Employee information  _ <== Cursor auf Feld
                    2.) Vehicle information   _


                        Enter code:  _

  To select a function, do one of the following:

  - place the cursor on the input field next to desired function and press ENTER
  - mark the input field next to desired function with an X and press ENTER
  - enter the desired function code (1 or 2) in the 'Enter code' field and press
    ENTER
```

Wenn der Benutzer den Cursor auf das Eingabefeld (#EMP) neben `Employee information` platziert und EINGABE drückt, gibt das Programm `LISTEMP` eine Liste der Mitarbeiter aus:

```
  Page      1                                                2001-01-22  09:39:32

          NAME
  --------------------

  ABELLAN
  ACHIESON
  ADAM
  ADKINSON
  ADKINSON
  ADKINSON
  ADKINSON
  ADKINSON
  ADKINSON
  ADKINSON
  ADKINSON
  AECKERLE
  AFANASSIEV
  AFANASSIEV
  AHL
  AKROYD
```

Anmerkungen:

1. In Natural for Ajax-Anwendungen dient *CURS-FIELD zur Identifikation des Operanden, welcher den Wert des Control darstellt, welches den Eingabefokus hat. Sie können *CURS-FIELD in Verbindung mit der POS-Funktion benutzen, um eine Prüfung auf das Control, das den Eingabefokus hat, zu veranlassen und die Verarbeitung in Abhängigkeit von diesem Zustand durchzuführen.
2. Die Werte von *CURS-FIELD und POS(*field-name*) dienen nur der internen Identifikation der Felder. Sie können für arithmetische Operationen nicht verwendet werden.

Einfachere Programmierung

Systemfunktion POS

Die Natural-Systemfunktion POS(*field-name*) enthält die interne Identifikation des Feldes, dessen Name mit der Systemfunktion angegeben wird.

POS(*field-name*) identifiziert ein bestimmtes Feld, unabhängig von seiner Position in einer Map. Auch wenn sich die Reihenfolge und Anzahl der Felder in einer Map ändert, identifiziert POS(*field-name*) nach wie vor eindeutig dasselbe Feld. Damit genügt zum Beispiel ein einziges REINPUT-Statement, um es von der Programmlogik abhängig zu machen, welches Feld MARKiert werden soll.

Anmerkung:
Der Wert von POS(*field-name*) dient nur zur internen Identifikation der Felder. Er kann nicht für arithmetische Operationen benutzt werden.

Beispiel:

```
...
DECIDE ON FIRST VALUE OF ...
   VALUE ...
      COMPUTE #FIELDX = POS(FIELD1)
   VALUE ...
      COMPUTE #FIELDX = POS(FIELD2)
   ...
END-DECIDE
...
REINPUT ... MARK #FIELDX
...
```

Weitere Einzelheiten zu *CURS-FIELD und POS(*field-name*) siehe *Systemvariablen* und *Systemfunktionen*.

Zeilenabhängige Verarbeitung

Systemvariable *CURS-LINE

Mit der Systemvariablen *CURS-LINE können Sie eine Verarbeitung davon abhängig machen, in welcher Zeile der Cursor gerade steht, wenn der Benutzer EINGABE drückt.

Mit dieser Variablen können Sie z.B. benutzerfreundliche Menüs gestalten. Bei entsprechender Programmierung braucht der Benutzer lediglich den Cursor in die Zeile der gewünschten Menü-Funktion zu platzieren und EINGABE zu drücken, um die Funktion auszuführen.

Die Cursor-Position bezieht sich auf das aktive Fenster, unabhängig von der Position auf dem physischen Bildschirm.

Anmerkung:
Die Meldungszeile, Funktionstastenleiste und Statistikzeile/Infoline zählen nicht als Datenzeilen auf dem Bildschirm.

Das Beispiel unten veranschaulicht die Möglichkeiten einer zeilenabhängigen Verarbeitung, die die Systemvariable `*CURS-LINE` bietet. Wenn der Benutzer in der Map EINGABE drückt, prüft das Programm, ob der Cursor in Zeile 8 des Bildschirms steht. Diese Zeile enthält die Option `Employee information`. Trifft dies zu, wird das Programm `LISTEMP` ausgeführt, das eine Liste der Mitarbeiter ausgibt.

```
DEFINE DATA LOCAL
1 #EMP (A1)
1 #CAR (A1)
1 #CODE (N1)
END-DEFINE
*
INPUT USING MAP 'CURS'
*
DECIDE FOR FIRST CONDITION
  WHEN *CURS-LINE = 8
     FETCH 'LISTEMP'
   WHEN NONE
     REINPUT 'PLACE CURSOR ON LINE OF OPTION YOU WISH TO SELECT'
END-DECIDE
END
```

Ausgabe:

```
                              Company Information


                   Please select a function


                [] 1.) Employee information

                   2.) Vehicle information









    Place the cursor on the line of the option you wish to select and press
    ENTER
```

Der Benutzer platziert den durch **[ ]** dargestellten Cursor in die Zeile der gewünschten Option und drückt EINGABE: das entsprechende Programm wird ausgeführt.

Spaltenabhängige Verarbeitung

Systemvariable *CURS-COL

Die Systemvariable `*CURS-COL` wird analog zu der Systemvariablen `*CURS-LINE` (wie oben beschrieben) benutzt. Mit `*CURS-COL` können Sie eine Verarbeitung davon abhängig machen, in welcher Spalte der Cursor steht.

Verarbeitung aufgrund von Funktionstasten

Systemvariable *PF-KEY

Oft ist eine funktionstastenabhängige Verarbeitung erwünscht.

Diese wird mittels des `SET KEY`-Statements und der Systemvariablen `*PF-KEY` und einer Änderung der Standard-Einstellungen der Map (`Standard Keys` = `Y`) erreicht.

Das `SET KEY`-Statement weist während der Ausführung des Programms Funktionstasten Funktionen zu. Die Systemvariable `*PF-KEY` enthält die Identifikation der vom Benutzer zuletzt gedrückten Funktionstaste.

Das nachstehende Beispiel veranschaulicht die Anwendung von `SET KEY` mit `*PF-KEY`.

```
...
SET KEY PF1
*
INPUT USING MAP 'DEMO&'
IF *PF-KEY = 'PF1'
  WRITE 'Help is currently not active'
END-IF
...
```

Das `SET KEY`-Statement aktiviert PF1 als Funktionstaste.

Das `IF`-Statement bestimmt, welche Aktionen erfolgen sollen, wenn der Benutzer PF1 drückt.

Beim Programmablauf wird der aktuelle Inhalt der Systemvariablen `*PF-KEY` geprüft; wenn sie PF1 enthält, wird die entsprechende Aktion ausgeführt.

Weitere Einzelheiten zum Statement `SET KEY` und der Systemvariable `*PF-KEY` finden Sie in der *Statements*- bzw. *Systemvariablen*-Dokumentation.

Verarbeitung aufgrund der Namen von Funktionstasten

Systemvariable *PF-NAME

Oft wird eine bestimmte Verarbeitung durch Drücken einer Funktionstaste ausgelöst. Noch mehr Komfort wird durch die Systemvariable `*PF-NAME` geboten. Mit ihr können Sie eine Verarbeitung von dem Namen einer Funktion abhängig machen, anstatt von einer Funktion.

Die Systemvariable *PF-NAME enthält den Namen der zuletzt gedrückten Funktionstaste (d.h. den Namen, der der Funktionstaste mit der NAMED-Klausel des SET KEY-Statements zugewiesen wurde).

Wünschen Sie z.B., dass der Benutzer die Hilfe-Funktion durch Drücken von wahlweise PF3 oder PF12 aufrufen kann, weisen Sie beiden Tasten den gleichen Namen (im folgenden Beispiel: INFO) zu. Drückt der Benutzer eine dieser beiden Funktionstasten, wird die im IF-Statement definierte Verarbeitung ausgelöst.

```
...
SET KEY PF3  NAMED 'INFO'
        PF12 NAMED 'INFO'
INPUT USING MAP 'DEMO&'
IF *PF-NAME = 'INFO'
  WRITE 'Help is currently not active'
END-IF
...
```

Die mit NAMED definierten Funktionsnamen erscheinen in der Funktionstastenleiste:

```
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
                 INFO                                                    INFO
```

Verarbeitung von Daten außerhalb des aktiven Fensters

Die folgenden Themen werden behandelt:

- Systemvariable *COM
- Beispiel für die Benutzung von *COM
- Positionierung des Cursors auf *COM — Terminalkommando %T*

Systemvariable *COM

Wie im Abschnitt *Bildschirm-Gestaltung — Fenster* weiter oben beschrieben, ist jeweils nur *ein* Fenster aktiv. In der Regel bedeutet dies, dass Eingaben nur innerhalb dieses Fensters möglich sind.

Mit der Systemvariablen *COM, die als Kommunikationsbereich betrachtet werden kann, ist es möglich, Eingaben auch außerhalb des aktiven Fensters zu machen.

Die Voraussetzung hierfür ist, dass die Map *COM als modifizierbares Feld enthält. Der Benutzer kann dann in dieses Feld Daten eingeben, auch wenn ein Fenster aktiv ist. Eine weitere Verarbeitung kann vom Inhalt der *COM-Variablen abhängig gemacht werden.

Auf diese Weise können Sie Benutzeroberflächen einrichten, wie sie z.B. bei Con-nect, dem Büro-Kommunikationssystem der Software AG implementiert sind: hier hat der Benutzer immer die Möglichkeit, Daten in die Kommandozeile einzugeben, auch wenn ein Fenster mit eigenen Eingabefeldern aktiv ist.

Beachten Sie, dass der Inhalt von *COM nur dann gelöscht wird, wenn die Natural-Session beendet wird.

Beispiel für die Benutzung von *COM

Im folgenden Beispiel führt das Programm ADD eine einfache Addition der Daten, die in der Map eingegeben werden, durch. In dieser Map wurde *COM als modifizierbares Feld definiert (am unteren Rand der Map), und zwar mit der im AL-Feld des Extended Field Editing angegebenen Länge. Das Ergebnis der Berechnung wird in einem Fenster ausgegeben. Obwohl dieses Fenster keine Eingabefelder hat, kann der Benutzer dennoch Eingaben machen, und zwar in das *COM-Feld außerhalb des Fensters.

Programm ADD:

```
DEFINE DATA LOCAL
1 #VALUE1 (N4)
1 #VALUE2 (N4)
1 #SUM3 (N8)
END-DEFINE
*
DEFINE WINDOW EMP
  SIZE 8*17
  BASE 10/2
  TITLE 'Total of Add'
  CONTROL SCREEN
  FRAMED POSITION SYMBOL BOT LEFT
*
INPUT USING MAP 'WINDOW'
*
COMPUTE #SUM3 = #VALUE1 + #VALUE2
*
SET WINDOW 'EMP'
INPUT (AD=O) / 'Value 1 +' /
               'Value 2 =' //
               ' ' #SUM3
*
IF *COM = 'M'
  FETCH 'MULTIPLY' #VALUE1 #VALUE2
END-IF
END
```

Ausgabe des Programms ADD:

```
                Map to Demonstrate Windows with *COM

                                CALCULATOR

                     Enter values you wish to calculate


                     Value 1:  12__
                     Value 2:  12__
   +-Total of Add-+
   !              !
   ! Value 1 +    !
   ! Value 2 =    !
   !              !
   !          24  !
   !              !
   +--------------+


    Next line is input field (*COM) for input outside the window:
```

Durch Eingabe von `M` wird die Funktion Multiplikation angestoßen; die beiden Werte aus der Eingabemaske werden miteinander multipliziert und das Ergebnis in einem zweiten Fenster ausgegeben:

```
                Map to Demonstrate Windows with *COM

                                CALCULATOR

                     Enter values you wish to calculate


                     Value 1:  12__
                     Value 2:  12__
   +-Total of Add-+                              +--------------+
   !              !                              !              !
   ! Value 1 +    !                              ! Value 1 x    !
   ! Value 2 =    !                              ! Value 2 =    !
   !              !                              !              !
   !          24  !                              !       144    !
   !              !                              !              !
   +--------------+                              +--------------+


    Next line is input field (*COM) for input outside the window:
               M
```

Positionierung des Cursors auf *COM — Terminalkommando %T*

Wenn ein Fenster aktiv ist und keine Eingabefelder (`AD=M` oder `AD=A`) enthält, wird der Cursor standardmäßig in die obere linke Ecke des Fensters positioniert.

Mit dem Terminalkommando `%T*` können Sie den Cursor auf die Systemvariable `*COM` außerhalb des Fensters positionieren, wenn das aktive Fenster keine Eingabefelder enthält.

Bei erneuter Eingabe von `%T*` wird die standardmäßige Positionierung des Cursors wieder aktiv.

Beispiel:

```
...
INPUT USING MAP 'WINDOW'
*
COMPUTE #SUM3 = #VALUE1 + #VALUE2
*
SET CONTROL 'T*'
SET WINDOW 'EMP'
INPUT (AD=O) / 'Value 1 +' /
               'Value 2 =' //
               ' ' #SUM3
...
```

Daten vom Bildschirm kopieren

Folgende Themen werden behandelt:

- Terminalkommandos %CS und %CC
- Eine Ausgabezeile eines Reports zur weiteren Verarbeitung auswählen

Terminalkommandos %CS und %CC

Mit diesen Terminalkommandos können sie Teile eines Bildschirms in den Natural-Stack (`%CS`) bzw. in die Systemvariable `*COM` (`%CC`) kopieren. Die geschützten Daten einer bestimmten Bildschirmzeile werden Feld für Feld kopiert.

Eine vollständige Beschreibung dieser Terminalkommandos finden Sie in der *Terminalkommandos*-Dokumentation.

Befinden sich die Daten im Stack oder in `*COM`, stehen sie zur weiteren Verarbeitung zur Verfügung. Mit diesen Kommandos können sie benutzerfreundliche Anwendungen wie im nachfolgenden Beispiel erstellen.

Eine Ausgabezeile eines Reports zur weiteren Verarbeitung auswählen

Im folgenden Beispiel gibt das Programm `COM1` eine Liste der Mitarbeiter von `Abellan` bis `Alestia` aus.

Programm COM1:

```
DEFINE DATA LOCAL
1 EMP VIEW OF EMPLOYEES
  2 NAME(A20)
  2 MIDDLE-NAME (A20)
  2 PERSONNEL-ID (A8)
END-DEFINE
*
READ EMP BY NAME STARTING FROM 'ABELLAN' THRU 'ALESTIA'
  DISPLAY NAME
END-READ
FETCH 'COM2'
END
```

Ausgabe des Programms COM1:

```
  Page      1                                                2006-08-12  09:41:21

          NAME
  --------------------

  ABELLAN
  ACHIESON
  ADAM
  ADKINSON
  ADKINSON
  ADKINSON
  ADKINSON
  ADKINSON
  ADKINSON
  ADKINSON
  ADKINSON
  AECKERLE
  AFANASSIEV
  AFANASSIEV
  AHL
  AKROYD
  ALEMAN
  ALESTIA
  MORE
```

Die Kontrolle wird nun an das Programm COM2 übergeben.

Programm COM2:

```
DEFINE DATA LOCAL
1 EMP VIEW OF EMPLOYEES
  2 NAME(A20)
  2 MIDDLE-NAME (A20)
  2 PERSONNEL-ID (A8)
1 SELECTNAME (A20)
END-DEFINE
*
SET KEY PF5 = '%CCC'
*
INPUT NO ERASE 'SELECT FIELD WITH CURSOR AND PRESS PF5'
*
MOVE *COM TO SELECTNAME
FIND EMP WITH NAME = SELECTNAME
  DISPLAY NAME PERSONNEL-ID
END-FIND
END
```

In diesem Programm ist das Terminalkommando %CCC der Funktionstaste PF5 zugeordnet. Das Terminalkommando kopiert alle geschützten Daten von der Zeile, in der sich der Cursor befindet, in die Systemvariable *COM. Diese Daten stehen dann zur weiteren Verarbeitung zur Verfügung. Die Art der Verarbeitung wird in den fett hervorgehobenen Zeilen des Beispielprogramms festgelegt.

Jetzt platziert der Benutzer den Cursor auf den Namen, der ihn interessiert, drückt PF5, und weitere Daten dieses Mitarbeiters werden ausgegeben.

```
SELECT FIELD WITH CURSOR AND PRESS PF5                    2006-08-12  09:44:25

         NAME
  --------------------

  ABELLAN
  ACHIESON
  ADAM  <==  Cursor positioned on name for which more information is required
  ADKINSON
  ADKINSON
  ADKINSON
  ADKINSON
  ADKINSON
  ADKINSON
  ADKINSON
  ADKINSON
  AECKERLE
  AFANASSIEV
  AFANASSIEV
  AHL
  AKROYD
  ALEMAN
  ALESTIA
```

In diesem Fall wird die Personalnummer (`PERSONNEL ID`) des ausgewählten Mitarbeiters angezeigt:

```
  Page      1                                             2006-08-12  09:44:52

         NAME          PERSONNEL
                          ID
  -------------------- ---------

  ADAM                 50005800
```

Statements REINPUT/REINPUT FULL

Das Statement `REINPUT` dient dazu, zu einem `INPUT`-Statement zurückzukehren und dieses erneut auszuführen. In der Regel wird es dazu benutzt, eine Fehlermeldung auszugeben, die dem Benutzer sagt, dass auf das `INPUT`-Statement hin ungültige Daten eingegeben wurden.

Wenn Sie die Option `FULL` in einem `REINPUT`-Statement angeben, wird das entsprechende `INPUT`-Statement vollständig neu ausgeführt:

- Bei einem normalen `REINPUT`-Statement (ohne `FULL`-Option) werden Inhalte von Variablen, die zwischen `INPUT`- und `REINPUT`-Statement geändert wurden, nicht angezeigt; d.h. alle Variablen auf dem Schirm zeigen den Inhalt, den Sie hatten, als das `INPUT`-Statement ursprünglich ausgeführt wurde.

- Bei einem `REINPUT FULL`-Statement werden alle nach der ersten Ausführung des `INPUT`-Statements gemachten Änderungen sichtbar, wenn das `INPUT`-Statement erneut ausgeführt wird; d.h. alle Variablen auf dem Schirm haben den Inhalt, den sie zum Zeitpunkt der Ausführung des `REINPUT`-Statements hatten.

- Wollen Sie zusätzlich den Cursor auf ein bestimmtes Feld positionieren, so können Sie die MARK-Option verwenden. Um auf eine bestimmte Position innerhalb eines Feldes zu positionieren, geben Sie die MARK POSITION-Option an.

Das Beispiel unten veranschaulicht die Anwendung von REINPUT FULL mit MARK POSITION.

```
DEFINE DATA LOCAL
1 #A (A10)
1 #B (N4)
1 #C (N4)
END-DEFINE
*
INPUT (AD=M) #A #B #C
IF #A = ' '
  COMPUTE #B = #B + #C
  RESET #C
  REINPUT FULL 'Enter a value' MARK POSITION 5 IN *#A
END-IF
END
```

Der Benutzer gibt 3 in das Feld #B und 3 in das Feld #C ein und drückt EINGABE.

```
  #A            #B     3 #C     3
```

Das Programm verlangt, dass das Feld #A einen Wert enthält. Das Statement REINPUT FULL mit MARK POSITION 5 IN *#A gibt den Eingabeschirm erneut aus, und zwar mit der geänderten Variable #B, die jetzt den Wert 6 enthält (Stand nach der COMPUTE-Berechnung). Der Cursor wird an die 5. Stelle in das für Neueingaben bereite Feld #A platziert.

```
Enter name of field
  #A    _       #B     6 #C     0



Enter a value
```

Das gleiche Statement ohne die FULL-Option würde folgenden Bildschirm ausgeben. Beachten Sie, dass die Variablen #B and #C auf den Stand zur Ausführungszeit des INPUT-Statements zurückgesetzt wurden (beide Felder enthalten den Wert 3).

```
   #A    _       #B     3 #C     3
```

Objektorientierte Datenverarbeitung — der Natural-Kommando-Prozessor

Der Natural-Kommando-Prozessor wird zur Definition und Steuerung der Navigation innerhalb einer Anwendung benutzt.

- Der *Entwicklungsteil* ist die Utility SYSNCP. Mit dieser Utility definieren Sie Kommandos, d.h. Kombinationen von Schlüsselwörtern, und die Aktionen, die als Reaktion auf die Ausführung dieser Kommandos ausgeführt werden sollen. Aus Ihren Definitionen erzeugt SYSNCP Entscheidungstabellen, die festlegen, was passiert, wenn ein Benutzer ein Kommando eingibt.

- Der *Laufzeitteil* ist das Statement `PROCESS COMMAND`. Dieses Statement wird benutzt, um den Kommando-Prozessor in einem Natural-Programm aufzurufen. Im Statement geben Sie den Namen der `SYSNCP`-Tabelle an, die benutzt wird, um die Dateneingabe eines Benutzers zu diesem Zeitpunkt zu verarbeiten.

Weitere Informationen zum Natural-Kommando-Prozessor siehe *SYSNCP Utility* in der *Utilities*-Dokumentation und das Statement `PROCESS COMMAND` in der *Statements*-Dokumentation.