Natural für Windows

Systemfunktionen

Version 6.3.8 für Windows

Februar 2010

Natural

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.

Inhaltsverzeichnis

1 Systemfunktionen

Diese Dokumentation beschreibt in Natural eingebaute Funktionen, die in bestimmten Statements benutzt werden können, siehe *Systemfunktionen* im *Leitfaden zur Programmierung*.

Anmerkung: Ab Natural Version 6.2 für Windows und UNIX, Version 6.3 für OpenVMS und Version 4.2 für Großrechner haben alle neuen Systemfunktionen einen Stern (*) als erstes Zeichen, um Namenskonflikte mit zum Beispiel Benutzervariablen in vorhandenen Anwendungen zu vermeiden.

Diese Dokumentation ist in die folgenden Abschnitte untergliedert:

Systemfunktionen für Verarbeitungsschleifen	Beschreibt Systemfunktionen, die im Zusammenhang mit einer Programmschleife verwendet werden können.
Mathematische Systemfunktionen	Beschreibt Systemfunktionen, die in arithmetischen Statements und in logischen Bedingungen (Logical Condition Criteria, LCC) unterstützt werden.
Verschiedene Systemfunktionen	Beschreibt Systemfunktionen zur Auswertung des Minimums/Maximums eines Feldes; Systemfunktion zur Feldidentifizierung; Systemfunktion zum Empfangen des Return Codes von einem Nicht-Natural-Programm; Systemfunktion zum Konvertieren von „nicht richtig sortierten“ Zeichen in andere Zeichen, die vom Sortierprogramm oder Datenbanksystem alphabetisch „richtig sortiert“ werden; Systemfunktion zur Umsetzung von Klein- auf Großbuchstaben; Systemfunktion zum Entfernen von führenden und/oder nachfolgenden Leerzeichen.
Natural Functions	Beschreibt Natural-Funktionen, welche die Software AG in der Library `SYSTEM` in der Systemdatei `FNAT` ausliefert. (Zurzeit nur in Englisch verfügbar.)

Siehe auch *Beispiel für Systemvariablen und Systemfunktionen* im *Leitfaden zur Programmierung*:

2 Systemfunktionen für Verarbeitungsschleifen

Dieses Kapitel erläutert die Natural-Systemfunktionen, die im Zusammenhang mit einer Programmschleife verwendet werden können.

Systemfunktionen für Verarbeitungsschleifen benutzen

Spezifikation/Auswertung

Natural-Systemfunktionen können angegeben werden in

- zuweisenden und arithmetischen Statements:
  - `MOVE`
  - `ASSIGN`
  - `COMPUTE`
  - `ADD`
  - `SUBTRACT`
  - `MULTIPLY`
  - `DIVIDE`
- in Eingabe-Ausgabe-Statements:
  - `DISPLAY`
  - `PRINT`
  - `WRITE`

die in einem der folgenden Statement-Blöcke stehen:

- `AT BREAK`
- `AT END OF DATA`
- `AT END OF PAGE`

d.h. für alle Verarbeitungsschleifen in den Statements `FIND`, `READ`, `HISTOGRAM`, `SORT` oder `READ WORK FILE`.

Wenn eine Systemfunktion in einem `AT END OF PAGE`-Statement verwendet wird, muss das betreffende `DISPLAY`-Statement eine `GIVE SYSTEM FUNCTIONS`-Klausel enthalten.

Datensätze, die aufgrund einer WHERE-Klausel zurückgewiesen werden, werden von einer Systemfunktion nicht ausgewertet.

Wenn mittels Systemfunktionen Datenbankfelder ausgewertet werden, die aus FIND-, READ-, HISTOGRAM- oder SORT-Schleifen stammen, die auf verschiedenen Ebenen liegen, werden die Feldwerte jeweils entsprechend ihrer Position in der Verarbeitungsschleifen-Hierarchie verarbeitet. Werte einer äußeren Schleife werden zum Beispiel nur dann verarbeitet, wenn für diese Schleife neue Datenwerte erhalten werden.

Wird eine Benutzervariable vor der ersten Verarbeitungsschleife definiert, wird sie für Systemfunktionen in der Schleife ausgewertet, in der das AT BREAK-, AT END OF DATA- oder AT END OF PAGE-Statement steht; wird eine Benutzervariable innerhalb einer Schleife definiert, wird sie in der gleichen Weise wie ein Datenbankfeld in der derselben Schleife behandelt.

Bei dem selektiven Einsatz von Systemfunktionen zur Auswertung von Benutzervariablen empfiehlt es sich, eine bestimmte Verarbeitungsschleife zu referenzieren (mittels Statement-Label oder Sourcecode-Zeilennummer), um genau festzulegen, in welcher Schleife der Wert der Benutzervariablen ausgewertet werden soll.

Systemfunktionen im SORT GIVE-Statement

Eine Systemfunktion kann auch referenziert werden, nachdem sie in der GIVE-Klausel eines SORT-Statements ausgewertet wurde.

In diesem Fall muss dem Namen der Systemfunktion bei der Referenzierung ein Stern (*) vorangestellt werden.

Arithmetischer Überlauf bei AVER, NAVER, SUM oder TOTAL

Ein Feld, das Sie mit den Systemfunktionen AVER, NAVER, SUM oder TOTAL verwenden, muss lang genug sein (standardmäßig oder vom Benutzer definiert), um die Summe der Feldwerte aufzunehmen. Falls der addierte Wert die Länge des Feldes überschreitet, erhalten Sie eine Fehlermeldung.

Normalerweise hat die Systemfunktion dieselbe Länge wie das angegebene Feld; ist diese Länge nicht ausreichend, dann verwenden Sie den Parameter NL des SORT GIVE-Statements, um die Ausgabelänge zu vergrößern:

```
SUM(field)(NL=nn)
```

Dadurch vergrößert sich nicht nur die Ausgabelänge, sondern auch die interne Länge des Feldes.

Statement-Referenzierung (r)

Statement-Referenzierung kann auch auf bei Systemfunktionen angewendet werden. Weitere Einzelheiten entnehmen Sie dem Unterabschnitt *Datenbankfelder mit der (r)-Notation referenzieren - Notation (r)* im Abschnitt *Benutzervariablen* im *Natural Leitfaden zur Programmierung*.

Durch Verwendung eines Statement-Labels oder Angabe der Sourcecode-Zeilennummer (*r*) können Sie bestimmen, in welcher Verarbeitungsschleife die jeweilige Systemfunktion für das betreffende Feld ausgewertet werden soll.

AVER(r)(field)

Format/Länge:	Wie Feld. Ausnahme: bei einem Feld mit Format N hat `AVER(field)` Format P (mit derselben Länge wie das Feld).

Diese Systemfunktion enthält den Durchschnittswert (Average) aller Werte des angegebenen Feldes. `AVER` wird aktualisiert, wenn die Bedingung, unter der `AVER` angefordert wurde, erfüllt ist.

COUNT(r)(field)

Format/Länge:	P7

Diese Systemfunktion zählt die Durchläufe einer Verarbeitungsschleife. Der Wert von `COUNT` erhöht sich jedesmal um 1, wenn die Verarbeitungsschleife, in der sich `COUNT` befindet, durchlaufen wird, und zwar unabhängig vom Wert des mit `COUNT` angegebenen Feldes.

MAX(r)(field)

Format/Länge:	Wie Feld.

Diese Systemfunktion enthält den größten Wert des angegebenen Feldes. `MAX` wird (falls erforderlich) jedesmal aktualisiert, wenn die Verarbeitungsschleife, in der sich `MAX` befindet, ausgeführt wird.

MIN(r)(field)

Format/Länge:	Wie Feld.

Diese Systemfunktion enthält den kleinsten Wert des angegebenen Feldes. `MIN` wird (falls erforderlich) jedesmal aktualisiert, wenn die Verarbeitungsschleife, in der sich `MIN` befindet, ausgeführt wird.

NAVER(r)(field)

Format/Länge:	Wie Feld. Ausnahme: bei einem Feld mit Format N hat `NAVER(field)` Format P (mit derselben Länge wie das Feld).

Diese Systemfunktion enthält den Durchschnittswert (Average) aller Werte des angegebenen Feldes, wobei Nullwerte nicht berücksichtigt werden. `NAVER` wird aktualisiert, wenn die Bedingung, unter der `NAVER` angefordert wurde, erfüllt ist.

NCOUNT(r)(field)

Format/Länge:	P7

Diese Systemfunktion zählt die Durchläufe einer Verarbeitungsschleife. Der Wert von `NCOUNT` erhöht sich jedesmal um 1, wenn die Verarbeitungsschleife, in der sich `NCOUNT` befindet, durchlaufen wird, wobei Durchläufe, bei denen der Wert des angegebenen Feldes Null ist, nicht mitgezählt werden.

Ob das Ergebnis von `NCOUNT` ein Array oder ein Skalarwert ist, hängt vom Argument (*field*) ab. Die Anzahl der resultierenden Ausprägungen ist dieselbe wie bei Feld.

NMIN(r)(field)

Format/Länge:	Wie Feld.

Diese Systemfunktion enthält den kleinsten Wert des angegebenen Feldes, wobei Nullwerte nicht berücksichtigt werden. `NMIN` wird (falls erforderlich) jedesmal aktualisiert, wenn die Verarbeitungsschleife, in der sich `NMIN` befindet, ausgeführt wird.

OLD(r)(field)

Format/Länge:	Wie Feld.

Diese Systemfunktion enthält den „alten“ Wert des angegebenen Feldes, d.h. den Wert, den das Feld vor einem in einer `AT BREAK`-Bedingung spezifizierten Gruppenwechsel (Wechsel des Feldwertes) bzw. vor einer Seitenende- oder Datenende-Bedingung (`END OF PAGE`, `END OF DATA`) hatte.

SUM(r)(field)

Format/Länge:	Wie Feld. Ausnahme: bei einem Feld mit Format N hat `SUM(field)` Format P (mit derselben Länge wie das Feld).

Diese Systemfunktion enthält die Summe aller Werte des angegebenen Feldes. `SUM` wird jedesmal aktualisiert, wenn die Verarbeitungsschleife, in der sich `SUM` befindet, ausgeführt wird. `SUM` wird nach jedem `AT BREAK`-Gruppenwechsel wieder auf Null gesetzt, addiert also nur Werte zwischen zwei Gruppenwechseln.

TOTAL(r)(field)

Format/Länge:	Wie Feld. Ausnahme: bei einem Feld mit Format N hat `TOTAL(field)` Format P (mit derselben Länge wie das Feld).

Diese Systemfunktion enthält die Gesamtsumme aller Werte des angegebenen Feldes in allen offenen Verarbeitungsschleifen, in denen `TOTAL` vorkommt.

Beispiele

- Beispiel 1 – AT BREAK-Statement mit Natural-Systemfunktionen OLD, MIN, AVER, MAX, SUM, COUNT
- Beispiel 2 – AT BREAK-Statement mit Natural-Systemfunktion AVER
- Beispiel 3 – AT END OF DATA-Statement mit Systemfunktionen MAX, MIN, AVER
- Beispiel 4 – AT END OF PAGE-Statement mit Systemfunktion AVER

Beispiel 1 – AT BREAK-Statement mit Natural-Systemfunktionen OLD, MIN, AVER, MAX, SUM, COUNT

```
** Example 'ATBEX3': AT BREAK  (with Natural system functions)
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 CITY
  2 SALARY    (1)
  2 CURR-CODE (1)
END-DEFINE
*
LIMIT 3
READ EMPLOY-VIEW LOGICAL BY CITY = 'SALT LAKE CITY'
  DISPLAY NOTITLE CITY NAME 'SALARY' SALARY(1) 'CURRENCY' CURR-CODE(1)
  /*
  AT BREAK OF CITY
    WRITE /  OLD(CITY) (EM=X^X^X^X^X^X^X^X^X^X^X^X^X^X^X^X^X^X)
      31T '   MINIMUM:' MIN(SALARY(1))  CURR-CODE(1) /
      31T '   AVERAGE:' AVER(SALARY(1)) CURR-CODE(1) /
      31T '   MAXIMUM:' MAX(SALARY(1))  CURR-CODE(1) /
      31T '       SUM:' SUM(SALARY(1))  CURR-CODE(1) /
      35T COUNT(SALARY(1)) 'RECORDS FOUND' /
  END-BREAK
  /*
  AT END OF DATA
```

```
    WRITE 22T 'TOTAL (ALL RECORDS):'
              T*SALARY TOTAL(SALARY(1))  CURR-CODE(1)
  END-ENDDATA
END-READ
*
END
```

Ausgabe des Programms `ATBEX3`:

```
CITY                 NAME           SALARY   CURRENCY
-------------------- -------------------- ---------- --------

SALT LAKE CITY       ANDERSON                50000 USD
SALT LAKE CITY       SAMUELSON               24000 USD

S A L T   L A K E   C I T Y      MINIMUM:      24000 USD
                                 AVERAGE:      37000 USD
                                 MAXIMUM:      50000 USD
                                     SUM:      74000 USD
                                         2 RECORDS FOUND

SAN DIEGO            GEE                     60000 USD

S A N   D I E G O                MINIMUM:      60000 USD
                                 AVERAGE:      60000 USD
                                 MAXIMUM:      60000 USD
                                     SUM:      60000 USD
                                         1 RECORDS FOUND

                     TOTAL (ALL RECORDS):     134000 USD
```

Beispiel 2 – AT BREAK-Statement mit Natural-Systemfunktion AVER

```
** Example 'ATBEX4': AT BREAK  (with Natural system functions)
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 CITY
  2 SALARY    (2)
*
1 #INC-SALARY (P11)
END-DEFINE
*
LIMIT 4
EMPL. READ EMPLOY-VIEW BY CITY STARTING FROM 'ALBU'
  COMPUTE #INC-SALARY = SALARY (1) + SALARY (2)
  DISPLAY NAME CITY SALARY (1:2) 'CUMULATIVE' #INC-SALARY
  SKIP 1
  /*
  AT BREAK CITY
```

```
    WRITE NOTITLE
      'AVERAGE:'            T*SALARY (1)  AVER(SALARY(1)) /
      'AVERAGE CUMULATIVE:' T*#INC-SALARY AVER(EMPL.) (#INC-SALARY)
  END-BREAK
END-READ
*
END
```

Ausgabe des Programms `ATBEX4`:

```
NAME                 CITY           ANNUAL    CUMULATIVE
                                                SALARY
-------------------- -------------------- ---------- ------------

HAMMOND              ALBUQUERQUE             22000        42200
                                             20200

ROLLING              ALBUQUERQUE             34000        65200
                                             31200

FREEMAN              ALBUQUERQUE             34000        65200
                                             31200

LINCOLN              ALBUQUERQUE             41000        78700
                                             37700

AVERAGE:                                     32750
AVERAGE CUMULATIVE:                                       62825
```

Beispiel 3 – AT END OF DATA-Statement mit Systemfunktionen MAX, MIN, AVER

```
** Example 'AEDEX1S': AT END OF DATA
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
  2 FIRST-NAME
  2 SALARY    (1)
  2 CURR-CODE (1)
END-DEFINE
*
LIMIT 5
EMP. FIND EMPLOY-VIEW WITH CITY = 'STUTTGART'
  IF NO RECORDS FOUND
    ENTER
  END-NOREC
  DISPLAY PERSONNEL-ID NAME FIRST-NAME
          SALARY (1) CURR-CODE (1)
  /*
  AT END OF DATA
```

```
    IF *COUNTER (EMP.) = 0
      WRITE 'NO RECORDS FOUND'
      ESCAPE BOTTOM
    END-IF
    WRITE NOTITLE / 'SALARY STATISTICS:'
                  / 7X 'MAXIMUM:' MAX(SALARY(1))  CURR-CODE (1)
                  / 7X 'MINIMUM:' MIN(SALARY(1))  CURR-CODE (1)
                  / 7X 'AVERAGE:' AVER(SALARY(1)) CURR-CODE (1)
  END-ENDDATA
  /*
END-FIND
*
END
```

Ausgabe des Programms `AEDEX1S`:

```
PERSONNEL         NAME              FIRST-NAME        ANNUAL   CURRENCY
   ID                                                 SALARY     CODE
--------- -------------------- -------------------- ---------- --------

11100328  BERGHAUS             ROSE                      70800 DM
11100329  BARTHEL              PETER                     42000 DM
11300313  AECKERLE             SUSANNE                   55200 DM
11300316  KANTE                GABRIELE                  61200 DM
11500304  KLUGE                ELKE                      49200 DM

SALARY STATISTICS:
       MAXIMUM:      70800 DM
       MINIMUM:      42000 DM
       AVERAGE:      55680 DM
```

Beispiel 4 – AT END OF PAGE-Statement mit Systemfunktion AVER

```
** Example 'AEPEX1S': AT END OF PAGE (structured mode)
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
  2 JOB-TITLE
  2 SALARY    (1)
  2 CURR-CODE (1)
END-DEFINE
*
FORMAT PS=10
LIMIT 10
READ EMPLOY-VIEW BY PERSONNEL-ID FROM '20017000'
  DISPLAY NOTITLE GIVE SYSTEM FUNCTIONS
          NAME JOB-TITLE 'SALARY' SALARY(1) CURR-CODE (1)
  /*
  AT END OF PAGE
```

```
    WRITE / 28T 'AVERAGE SALARY: ...' AVER(SALARY(1)) CURR-CODE (1)
  END-ENDPAGE
END-READ
*
END
```

Ausgabe des Programms AEPEX1S:

```
NAME                  CURRENT            SALARY   CURRENCY
                             POSITION                      CODE
-------------------- ------------------------ ---------- --------

CREMER               ANALYST                    34000 USD
MARKUSH              TRAINEE                    22000 USD
GEE                  MANAGER                    39500 USD
KUNEY                DBA                        40200 USD
NEEDHAM              PROGRAMMER                 32500 USD
JACKSON              PROGRAMMER                 33000 USD

                           AVERAGE SALARY: ...  33533 USD
```

3 Mathematische Systemfunktionen

In logischen Bedingungen und den Arithmetik-Statements `ADD`, `COMPUTE`, `DIVIDE`, `MULTIPLY` und `SUBTRACT` können Sie die folgenden mathematischen Funktionen verwenden:

Funktion	Format/Länge	Ausgegebener Wert
ABS*(field)*	wie Feld (*field*)	Absoluter Wert eines Feldes.
ATN*(field)*	F8 (*)	Arcustangens eines Feldes.
COS*(field)*	F8 (*)	Kosinus eines Feldes.
EXP*(field)*	F8 (*)	Potenz eines Feldes (*e field*) mit der Basis *e*, wobei *e* die Basis natürlicher Logarithmen ist.
FRAC*(field)*	wie Feld (*field*)	Bruchteil (hinter dem Komma) eines Feldes.
INT*(field)*	wie Feld (*field*)	Ganzzahliger Teil eines Feldes (Integer).
LOG*(field)*	F8 (*)	Natürlicher Logarithmus eines Feldes.
SGN*(field)*	wie Feld (*field*)	Vorzeichen (Sign) eines Feldes (−1, 0, +1).
SIN*(field)*	F8 (*)	Sinus eines Feldes.
SQRT*(field)*	F8 (*)	Quadratwurzel eines Feldes (Square Root). Ein negativer Feldwert wird wie ein positiver behandelt.
TAN*(field)*	F8 (*)	Tangens eines Feldes.
VAL*(field)*	wie Zielfeld (*field*)	Numerischer Wert eines alphanumerischen Feldes. Der Wert des Feldes muss ein in alphanumerischer Form (Codepage oder Unicode) dargestellter numerischer Wert sein. Leerstellen vor und nach dem Komma im Feld werden ignoriert. Komma (Dezimalpunkt) und Vorzeichen werden mitverarbeitet. Wenn das Zielfeld nicht lang genug ist, werden Nachkommastellen abgeschnitten (vgl. *Abschneiden und Runden von Feldwerten* im Abschnitt *Regeln für arithmetische Operationen* im *Leitfaden zur Programmierung*).

*Diese Funktionen werden wie folgt ausgewertet:

- Das Argument wird in Format/Länge F8 umgesetzt und dann an das Betriebssystem zur Berechnung übergeben.
- Das vom Betriebssystem zurückgelieferte Ergebnis hat Format/Länge F8 und wird in das Zielformat zurückgewandelt.

Ein mit einer mathematischen Funktion (außer `VAL`) verwendetes Feld kann eine Konstante oder ein Skalar sein und muss numerisches, gepackt numerisches, Ganzzahl- oder Gleitkomma-Format (N, P, I oder F) haben.

Ein mit der Funktion `VAL` verwendetes Feld kann eine Konstante, ein Skalar oder ein Array sein und muss alphanumerisches Format haben.

Beispiel für mathematische Funktionen:

```
** Example 'MATHEX': Mathematical functions
************************************************************************
DEFINE DATA LOCAL
1 #A     (N2.1) INIT <10>
1 #B     (N2.1) INIT <-6.3>
1 #C     (N2.1) INIT <0>
1 #LOGA  (N2.6)
1 #SQRTA (N2.6)
1 #TANA  (N2.6)
1 #ABS   (N2.1)
1 #FRAC  (N2.1)
1 #INT   (N2.1)
1 #SGN   (N1)
END-DEFINE
*
COMPUTE #LOGA = LOG(#A)
WRITE NOTITLE '=' #A 5X 'LOG'         40T #LOGA
*
COMPUTE #SQRTA = SQRT(#A)
WRITE         '=' #A 5X 'SQUARE ROOT' 40T #SQRTA
*
COMPUTE #TANA  = TAN(#A)
WRITE         '=' #A 5X 'TANGENT'     40T #TANA
*
COMPUTE #ABS   = ABS(#B)
WRITE     //  '=' #B 5X 'ABSOLUTE'    40T #ABS
*
COMPUTE #FRAC  = FRAC(#B)
WRITE         '=' #B 5X 'FRACTIONAL'  40T #FRAC
*
COMPUTE #INT   = INT(#B)
WRITE         '=' #B 5X 'INTEGER'     40T #INT
*
COMPUTE #SGN   = SGN(#A)
```

```
WRITE      // '=' #A 5X 'SIGN'        40T #SGN
*
COMPUTE #SGN   = SGN(#B)
WRITE         '=' #B 5X 'SIGN'        40T #SGN
*
COMPUTE #SGN   = SGN(#C)
WRITE         '=' #C 5X 'SIGN'        40T #SGN
*
END
```

Ausgabe des Programms `MATHEX`:

```
#A:  10.0     LOG                       2.302585
#A:  10.0     SQUARE ROOT               3.162277
#A:  10.0     TANGENT                   0.648360


#B:  -6.3     ABSOLUTE                  6.3
#B:  -6.3     FRACTIONAL               -0.3
#B:  -6.3     INTEGER                  -6.0


#A:  10.0     SIGN                      1
#B:  -6.3     SIGN                     -1
#C:   0.0     SIGN                      0
```

4 Verschiedene Funktionen

Folgende Themen werden behandelt:

- ***MINVAL/*MAXVAL - Minimum/Maximum auswerten**
- **POS - Feldidentifikationsfunktion**
- **RET - Returncode-Funktion**
- **SORTKEY - Sortierschlüssel-Funktion**
- ***TRANSLATE - Zeichen in Groß-/Kleinschreibung umsetzen**
- ***TRIM - Führende und/oder nachfolgende Leerstellen entfernen**

5 *MINVAL/*MAXVAL - Minimal-/Maximalwertes eines Feldes

```
{ *MINVAL } ([(IR=result-format/length)] operand,...)
{ *MAXVAL }
```

format/length: Format und Länge können entweder explizit mit der IR-Klausel angegeben oder automatisch mit Hilfe den unten aufgeführten Tabellen errechnet werden, siehe ***Tabelle für die Konvertierung der resulting-format/length-Werte***.

Funktion

Die Natural-Systemfunktion *MINVAL bzw. *MAXVAL wertet die Minimal-/Maximalwerte aller gegebenen Operanden aus. Das Ergebnis ist immer ein Skalarwert. Wird ein Array als Operand angegeben, dann wird der Minimal- bzw. Maximalwert aller Array-Felder ausgewertet.

Wenn bei Verwendung von alphanumerischen oder binären Daten dieselben Daten als Argument angegeben sind, (z.B. *MINVAL('AB','AB')), dann ist das Ergebnis das Argument mit dem kleinsten/größten Längenwert.

Einschränkungen

Für die Verwendung der Systemfunktion *MINVAL/*MAXVAL gelten folgende Einschränkungen:

- *MINVAL/*MAXVAL darf nicht an Stellen verwendet werden, an denen eine Zielvariable erwartet wird.
- *MINVAL/*MAXVAL darf nicht in einer anderen Systemfunktion verschachtelt werden

Syntax-Beschreibung

Operanden-Definitionstabelle:

Operand	Possible Structure					Possible Formats													Referencing Permitted	Dynamic Definition
operand	C	S	A	G		A	U	N	P	I	F	B	D	T					yes	no

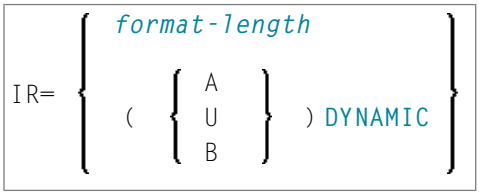
Syntax-Elementbeschreibung:

Schlüsselwort	Beschreibung
*MINVAL	Berechnet den Mininalwert aller gegebenen Operandenwerte.
*MAXVAL	Berechnet den Mininalwert aller gegebenen Operandenwerte.
operand	Der Operand bzw. die Operanden, deren Minimal-/Maximalwert(e) von der Systemfunktion *MINVAL bzw. *MAXVAL errechnet werden soll.
result-format-length	IR-Klausel zur **expliziten Angabe**der resultierenden Format/Länge-Werte. Siehe unten.

IR-Klausel

Diese Klausel kann zur expliziten Angabe der *result-format/length*-Werte für die gesamte Systemfunktion *MINVAL bzw. *MAXVAL verwendet werden.

```
IR=result-format/length
```



Eine Aufstellung der gültigen *result-format/length*-Werte ist im Abschnitt **Tabelle für die Konvertierung der *resulting-format/length*-Werte** enthalten.

Syntax-Elementbeschreibung:

Schlüsselwort	Beschreibung
format-length	Der Compiler versucht, die resultierenden Format-/Längenwerte der Gesamtfunktion zu bestimmen. Wenn dies nicht ohne Einbußen an Genauigkeit möglich ist, muss der Programmierer die *format-length*-Werte mit Hilfe der IR-Operandenerweiterung angeben.
A, U oder B	Format: Alphanumerisch oder binär für dynamische Varible.
DYNAMIC	Anstelle von festen Format-/Längenwerten kann auch ein alphanumerisches, Unicode- oder binäres Format mit dynamischer Länge angegeben werden.

Beispiel:

```
DEFINE DATA LOCAL
1 #RESULTI       (I4)
1 #RESULTA       (A20)
1 #RESULTADYN    (A) DYNAMIC
1 #A(I4)         CONST <1234>
1 #B(A20)        CONST <H'30313233'> /* '0123' stored
1 #C(I2/1:3)     CONST <2000, 2100, 2200>
END-DEFINE
*
#RESULTA     := *MAXVAL((IR=A20)         #A, #B)     /*no error, I4->A20 is allowed!
#RESULTADYN  := *MAXVAL((IR=(A)DYNAMIC)  #A, #B)     /*result is (A) dynamic
/* #RESULTI  := *MAXVAL((IR=I4)         #A, #B)     /*compiler error, because conv.
A20->I4 is not allowed!
#RESULTI     := *MAXVAL((IR=I4)          #A, #C(*))  /*maximum of the array is
evaluated
DISPLAY #RESULTA #RESULTADYN (AL=10) #RESULTI
END
```

Tabelle für die Konvertierung der resulting-format/length-Werte

Es gibt zwei Arten, die resultierenden Format-/Längenwerte für die gesamte Systemfunktion `*MINVAL/*MAXVAL` anzugeben.

- Explizite Angabe der resultierenden Format-/Längenwerte
- Implizite Angabe der resultierenden Format-/Längenwerte

Explizite Angabe der resultierenden Format-/Längenwerte

Die resultierenden Format-/Längenwerte für die gesamte Systemfunktion `*MINVAL` bzw. `*MAXVAL` können mit der `IR`-Klausel angegeben werden. Alle angegebenen Operanden werden in die mit dieser Klausel festgelegten resultierenden Format-/Längenwerte umgewandelt, wenn dies ohne Einbußen bezüglich der Genauigkeit möglich ist. Anschließend erfolgt die Berechnung der Minimal- bzw. Maximalwerte aller umgewandelten Operanden, und es wird ein einziger Skalarwert mit dem errechneten Format-/Längenwert als Ergebnis für die gesamte Systemfunktion festgelegt.

Implizite Angabe der resultierenden Format-/Längenwerte

Wenn keine IR-Klausel in der Systemfunktion *MINVAL bzw. *MAXVAL verwendet wird, erfolgt die Berechnung der resultierenden Format-/Längenwerte anhand der Formate/Längen aller in der Systemfunktion als Argument angegebenen Operanden. Dabei wird der *format/length*-Wert jedes einzelnen Operanden genommen und mit dem *format/length*-Wert des darauf folgenden Operanden in der Argument-Liste zusammengefasst. Der resultierenden Format-/Längenwert von zwei einzelnen Operanden wird dann gemäß der unten aufgeführten Tabellen errechnet.

Die Tabelle für die Konvertierung der *resulting-format/length*-Werte ist in zwei Untertabellen aufgeteilt. Kombinationen, die nicht in diesen beiden Tabellen aufgeführt sind, sind ungültig und dürfen in der Argument-Liste der Systemfunktion *MINVAL bzw. *MAXVAL nicht verwendet werden. Das Schlüsselwort FLF gibt an, wann zur Vermeidung von Ungenauigkeiten die IR-Klausel zur Festlegung des resultierenden Format-/Längenwertes benutzt werden muss.

Tabelle 1

Enthält alle nummerischen Kombinationsmöglichkeiten für zwei verschiedene Operanden.

		Zweiter Operand				
	Format/Länge	**I1**	**I2**	**I4**	**P*a*.*b*, N*a*.*b***	**F4, F8**
Erster Operand	**I1**	I1	I2	I4	Pmax(3,*a*).*b*	F8
	I2	I2	I2	I4	Pmax(5,*a*).*b*	F8
	I4	I4	I4	I4	Pmax(10,*a*).*b*	F8
	P*x*.*y*, N*x*.*y*	Pmax(3,*x*).*y*	Pmax(5,*x*).*y*	Pmax(10,*x*).*y*	wenn max(*x*,*a*) + max(*y*,*b*) <= 29 Pmax(*x*,*a*).max(*y*,*b*) sonst FLF	wenn *y*=0 und *x* <=15; F8 sonst FLF
	F4, F8	F8	F8	F8	wenn *b*=0 und *a* <=15 F8 sonst FLF	F8

Legende:

FLF	Format-/Längenangabe mittels IR-Klausel zwingend erforderlich.
I*x*	Format/Länge ist Integer (Ganzzahl). *x* gibt die Anzahl der Bytes an, die zum Speichern des Integer-Wertes benutzt werden.
F*x*	Format/Länge ist Float (Gleitkomma). *x* gibt die Anzahl der Bytes an, die zum Speichern des Float-Wertes benutzt werden.
P*x*.*y* **P***a*,*b*	Format ist Packed (gepackt) mit entsprechender Anzahl an Stellen vor dem Dezimalpunkt (*x*,*a*) und der Genauigkeit (*y*,*b*).

Nx.y **Na,b**	Format ist Numeric (nummerisch) mit entsprechender Anzahl an Stellen vor dem Dezimalpunkt (*x*, *a*) und der Genauigkeit (*y*, *b*).
Pmax(*c*, *d***).***e*	Das resultierende Format ist Packed (gepackt). Die Berechnung der Länge erfolgt anhand der nachfolgenden Informationen. Die Anzahl an Stellen vor dem Dezimalpunkt ist der Maximalwert von *c* und *d*. Der Genauigkeitswert ist *e*.
Pmax(*c*, *d***).max(***e*, *f***)**	Das resultierende Format ist Packed (gepackt). Die Berechnung der Länge erfolgt anhand der nachfolgenden Informationen. Die Anzahl an Stellen vor dem Dezimalpunkt ist der Maximalwert von *c* und *d*. Der Genauigkeitswert ist der Maximalwert von *e* und *f*.

Tabelle 2

Enthält alle Formate und Längen, die für Operanden in den Systemfunktionen `*MINVAL/*MAXVAL` verwendet werden können.

		Zweiter Operand				
	Format-length	**D**	**T**	**A***a*, **A dynamic**	**B***a*, **B dynamic**	**U***a*, **U dynamic**
Erster Operand	**D**	D	T	NA	NA	NA
	T	T	T	NA	NA	NA
	A*x*, **A dynamic**	NA	NA	A dynamic	A dynamic	U dynamic
	B*x*, **B dynamic**	NA	NA	A dynamic	B dynamic	U dynamic
	U*x*, **U dynamic**	NA	NA	U dynamic	U dynamic	U dynamic

Legende:

NA	Unzulässige Kombination.
D	Datenformat.
T	Zeitformat.
B*x*, **B***a*	Binärformat mit Länge *x*, *a*.
A*x*, **A***a*	Alphanumerisches Format mit Länge *x*, *a*.
U*x*, **U***a*	Unicode-Format mit Länge *x*, *a*.
B dynamic	Binärformat mit dynamischer Länge.
A dynamic	Alphanumerisches Format mit dynamischer Länge.
U dynamic	Unicode-Format mit dynamischer Länge.

Berechnung der result-format-length

Anhand der oben aufgeführten Regeln ist der Compiler in der Lage, die Source-Operanden unter Berücksichtigung von Operandenpaaren zu verarbeiten und die Zwischenergebnisse für jedes Paar zu berechnen. Das erste Paar besteht aus dem ersten und dem zweiten Operand, das zweite Paar aus dem Zwischenergebnis und dem dritten Operand usw. Nach Verarbeitung aller Operanden stellt das letzte Ergebnis den Vergleich zwischen Format und Länge dar, die zum Vergleich mit allen Operanden zwecks Berechnung des Minimal- bzw. Maximalwertes verwendet werden. Bei Verwendung dieser Format-/Längenberechnungsmethode können die Operanden in beliebiger Reihenfolge auftreten.

Beispiel:

```
DEFINE DATA LOCAL
1 A (I2)      INIT <34>
1 B (P4.2)    INIT <1234.56>
1 C (N4.4)    INIT <12.6789>
1 D (I1)      INIT <100>
1 E (I4/1:3)  INIT <32, 6745, 456>
1 #RES-MIN (P10.7)
1 #RES-MAX (P10.7)
END-DEFINE
*
MOVE *MINVAL(A, B, C, D, E(*)) TO #RES-MIN
MOVE *MAXVAL(A, B, C, D, E(*)) TO #RES-MAX
DISPLAY #RES-MIN #RES-MAX
END
```

Ausgabe:

```
        #RES-MIN                     #RES-MAX
---------------------------  ---------------------------

                12.6789000                6745.0000000
```

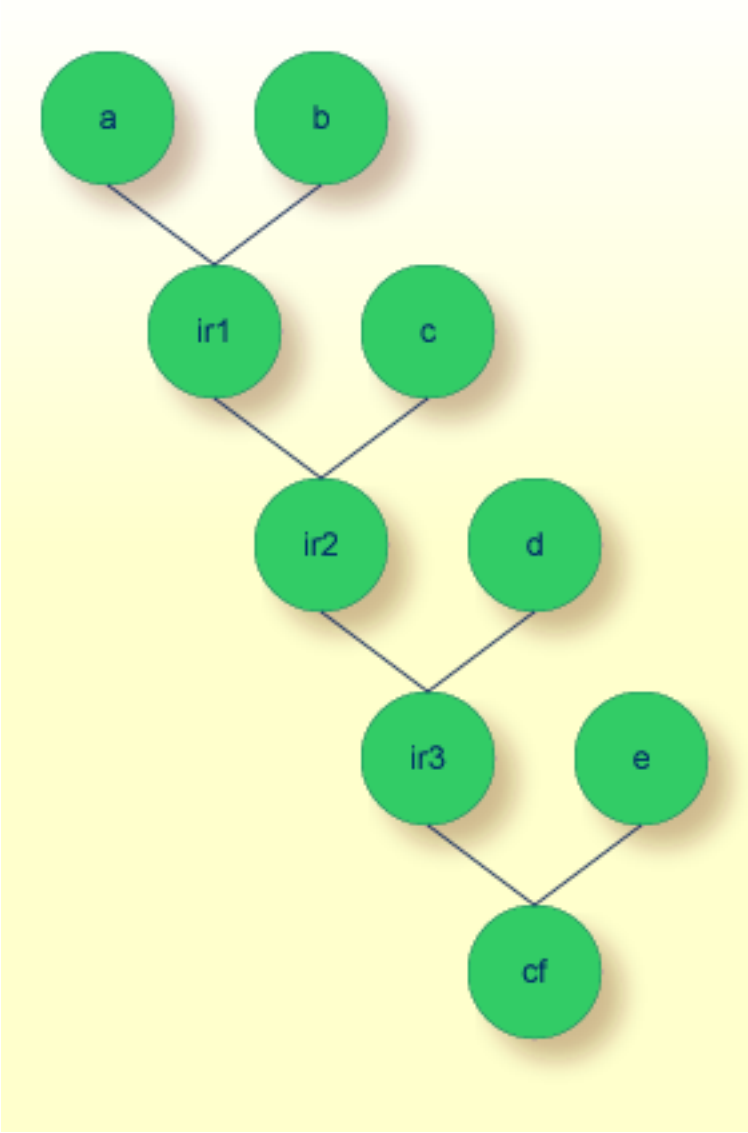
Die folgende Tabelle zeigt die einzelnen Schritte zur automatischen Format-/Längenberechnung im obigen Beispiel. Sie enthält die Zwischenergebnisse (ir) aller Schritte und den Format-/Längenwert des Vergleichs (cf), der als *result-format/length*-Wert verwendet wird.

Berechnungs-reihenfolge	Name des ersten Operanden	Format/Länge des ersten Operanden bzw. Zwischenergebnis (ir)	Name des zweiten Operanden	Format/Länge des zweiten Operanden bzw. Zwischenergebnis (ir)	Format/Länge des Zwischenergebnisses(ir)
1.	A	I2	B	P4.2	ir1 = P5.2
2.	ir1	P5.2	C	N4.4	ir2 = P5.4
3.	ir2	P5.4	D	I1	ir3 = P5.4
4.	ir3	P5.4	E	I4	**cf = P10.4**

Zur Laufzeit werden alle Operanden in die cf-Format/Länge umgewandelt. Danach werden alle umgewandelten Werte miteinander verglichen und der entsprechende Minimal- bzw. Maximalwert wird errechnet.

Reihenfolge der Berechnung von Format und Länge

Die folgende Grafik zeigt die Reihenfolge, in der die Berechnung von Format und Länge erfolgt:



Legende:

ir1, ir2, ir3	Zwischenergebnis (intermediate result) 1, 2, 3.
cf	Vergleich von Format und Länge (comparison).

6 POS - Feldidentifikationsfunktion

Format/Länge:	I4

Die 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 Maske (Map). Auch wenn sich die Reihenfolge und Anzahl der Felder in einer Maske 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 `MARK`iert werden soll.

Beispiel:

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

Wenn das mit `POS` angegebene Feld ein Array ist, muss eine bestimmte Ausprägung angegeben werden; zum Beispiel `POS(FIELDX(5))`. Auf einen Array-Bereich kann `POS` nicht angewendet werden.

POS und *CURS-FIELD

`POS(field-name)` kann in Verbindung mit der Natural-Systemvariablen `*CURS-FIELD` dazu verwendet werden, die Ausführung bestimmter Funktionen davon abhängig zu machen, in welchem Feld der Cursor zur Zeit steht.

`*CURS-FIELD` enthält die interne Identifikation des Feldes, in dem sich der Cursor zur Zeit befindet; *CURS–FIELD kann nicht alleine, sondern nur zusammen mit `POS(field-name)` verwendet werden. Sie können beide zusammen dazu benutzen, um zu prüfen, ob sich der Cursor gerade in einem bestimmten Feld befindet, und die weitere Verarbeitung von dieser Bedingung abhängig machen.

Beispiel:

```
IF *CURS-FIELD = POS(FIELDX)
   MOVE *CURS-FIELD TO #FIELDY
END-IF
...
REINPUT ... MARK #FIELDY
```

Anmerkungen:

1. Die Werte von `*CURS-FIELD` und `POS(field-name)` dienen nur zur internen Identifikation der Felder und können nicht für arithmetische Operationen verwendet werden.
2. Der von `POS(field-name)` zurückgegebene Wert für eine Ausprägung eines X-Arrays (ein Array, für das wenigstens eine Dimension als erweiterbar angegeben ist) kann sich ändern, nachdem die Anzahl der Ausprägungen für eine Dimension des Arrays mittels der Statements `EXPAND`, `RESIZE` oder `REDUCE` geändert wurde.
3. Natural RPC: Wenn `*CURS-FIELD` und `POS(field-name)` sich auf eine Kontextvariable beziehen, können die daraus resultierenden Informationen nur innerhalb derselben Konversation verwendet werden.
4. 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, welches den Eingabefokus hat, zu veranlassen und die Verarbeitung in Abhängigkeit von diesem Zustand durchzuführen.

Siehe auch *Dialog-Gestaltung*, *Feld-sensitive Verarbeitung* und *Einfachere Programmierung* im *Leitfaden zur Programmierung*.

7 RET - Returncode-Funktion

Format/Länge:	I4

Die Systemfunktion `RET(program-name)` kann dazu verwendet werden, den Returncode eines nicht in Natural geschriebenen Programms, das über ein `CALL`-Statement aufgerufen wurde, zu erhalten.

`RET(program-name)` kann in einem `IF`-Statement sowie in den Arithmetik-Statements `ADD`, `COMPUTE`, `DIVIDE`, `MULTIPLY` und `SUBTRACT` verwendet werden.

Beispiel:

```
DEFINE DATA LOCAL
1 #RETURN (I4)
...
END-DEFINE
...
...
CALL 'PROG1'
IF RET('PROG1') > #RETURN
   WRITE 'ERROR OCCURRED IN PROGRAM 1'
END-IF
...
```

8 SORTKEY - Sort-Key Function

SORTKEY (*character-string*)

Diese Systemfunktion dient zum Konvertieren von „nicht richtig sortierten“ Zeichen (oder Kombinationen von Zeichen) in andere Zeichen (oder Kombinationen von Zeichen), die vom Sortierprogramm oder Datenbanksystem alphabetisch „richtig sortiert“ werden.

Format/Länge:	A253

Es gibt in vielen Landessprachen Zeichen (oder Zeichenkombinationen), die von einem Sortierprogramm oder Datenbanksystem nicht in der richtigen alphabetischen Reihenfolge sortiert werden, da die Reihenfolge der Zeichen im vom Computer verwendeten Zeichensatz nicht immer der alphabetischen Reihenfolge der Zeichen entspricht.

Zum Beispiel wird der spanische Buchstabe CH in der Regel von einem Sortierprogramm bzw. Datenbanksystem wie zwei Buchstaben behandelt und zwischen CG und CI einsortiert — gehört aber eigentlich als eigener Buchstabe im spanischen Alphabet zwischen "C" und "D".

Oder es kann sein, dass Kleinbuchstaben und Großbuchstaben entgegen Ihren Wünschen bei der Sortierreihenfolge nicht gleich behandelt werden, dass Ziffern vor Buchstaben sortiert werden (Sie aber wünschen, dass Buchstaben vor Ziffern sortiert werden) oder dass Sonderzeichen (z.B. Bindestriche in Doppelnamen) zu einer unerwünschten Sortierreihenfolge führen.

In solchen Fällen können Sie die Systemfunktion SORTKEY(*character-string*) benutzen. Die von SORTKEY berechneten Werte werden nur als Sortierkriterium benutzt, während die ursprünglichen Werte für die Interaktion mit dem Endbenutzer verwendet werden.

Sie können die SORTKEY-Funktion in einem COMPUTE sowie in einer logischen Bedingung als arithmetischen Operanden verwenden.

Als *character-string* können Sie eine alphanumerische Konstante oder Variable oder eine einzelne Ausprägung eines alphanumerischen Arrays angeben.

Wenn Sie die SORTKEY-Funktion in einem Natural-Programm angeben, wird der User-Exit NATUSK*nn* aufgerufen — wobei *nn* der aktuelle Sprachcode (d.h. der aktuelle Wert der Systemvariablen *LANGUAGE) ist.

Sie können diesen User-Exit in jeder Programmiersprache, die über eine Standard-CALL-Schnittstelle verfügt, schreiben. Der mit SORTKEY angegebene *character-string* wird an den User-Exit übergeben. Der User-Exit muss so programmiert werden, dass er „falsch sortierte“ Zeichen in dieser Zeichenkette in entsprechende „richtig sortierte“ Zeichen umsetzt. Die umgesetzte Zeichenkette wird dann vom Natural-Programm zur weiteren Verarbeitung verwendet.

Allgemeine Aufruf-Konventionen für externe Programme sind in der Dokumentation zum CALL-Statement erläutert.

Nähere Informationen zu den Aufruf-Konventionen für SORTKEY User-Exits finden Sie beim CALL-Statement in *User Exits* in der Natural *Statements*-Dokumentation.

Beispiel:

```
DEFINE DATA LOCAL
1 CUST VIEW OF CUSTOMERFILE
  2 NAME
  2 SORTNAME
END-DEFINE
...
*LANGUAGE := 4
...
REPEAT
  INPUT NAME
  SORTNAME := SORTKEY(NAME)
  STORE CUST
  END TRANSACTION
  ...
END-REPEAT
...
READ CUST BY SORTNAME
  DISPLAY NAME
END-READ
...
```

Angenommen, im obigen Beispiel würden bei mehrmaliger Ausführung des INPUT-Statements nacheinander die Werte "Sanchez", "Sandino" und "Sancinto" eingegeben.

Bei der Zuweisung von SORTKEY(NAME) zu SORTNAME würde der User-Exit NATUSK04 aufgerufen. Dieser müsste so programmiert werden, dass er zunächst alle Kleinbuchstaben in Großbuchstaben umsetzt und dann die Zeichenfolge "CH" in "C*x*" umsetzt — wobei *x* dem letzten Zeichen im verwendeten Zeichensatz entspricht, also hexadezimal H'FF' (vorausgesetzt dieses letzte Zeichen ist kein druckbares Zeichen).

Es werden sowohl die „eigentlichen“ Namen (NAME) als auch die für die gewünschte Sortierung umgesetzten Namen (SORTNAME) gespeichert. Zum Lesen der Datei wird SORTNAME verwendet. Dann würden bei Ausführung des DISPLAY-Statements die Namen in der richtigen spanischen alphabetischen Reihenfolge ausgegeben:

```
Sancinto
Sanchez
Sandino
```

9 *TRANSLATE - Umsetzung in Groß-/Kleinschreibung

```
*TRANSLATE (operand [ ,{ LOWER } ] )
                       { UPPER }
```

Format/Länge: wie bei *operand*.

Funktion

Die Systemfunktion *TRANSLATE dient zum Umsetzen von Zeichen eines Operanden mit Alphanumerischem oder binärem Format in Groß- oder Kleinbuchstaben. Der Inhalt des Operanden bleibt dabei unverändert.

*TRANSLATE kann als *operand* an jeder Stelle eines Statements angegeben werden, an der ein Operand mit Format A oder B zulässig ist.

Einschränkungen

Für die Verwendung der Systemfunktion *TRANSLATE gelten folgende Einschränkungen:

- *TRANSLATE darf nicht an Stellen verwendet werden, an denen eine Zielvariable erwartet wird.
- *TRANSLATE darf nicht in einer anderen Systemfunktion verschachtelt werden.

Syntax-Beschreibung

Operanden-Definitionstabelle:

Operand	Mögliche Struktur					Mögliche Formate										Referenzierung erlaubt	Dynam. Definition
operand	C	S	A			A	U	B								ja	nein

Syntax-Elementbeschreibung:

*TRANSLATE (*operand*,LOWER)	**Umsetzung in Kleinbuchstaben** Bei Angabe des Schlüsselworts LOWER als zweites Argument angegeben wird die Zeichenkette in *operand* in Kleinbuchstaben umgesetzt.
*TRANSLATE (*operand*,UPPER)	**Umsetzung in Großbuchstaben** Bei Angabe des Schlüsselworts UPPER als zweites Argument angegeben wird die Zeichenkette in *operand* in Großbuchstaben umgesetzt.

Beispiel

```
DEFINE DATA LOCAL
1 #SRC  (A)DYNAMIC INIT <'aBcDeFg !§$%&/()=?'>
1 #DEST (A)DYNAMIC
END-DEFINE
*
PRINT 'Source string to be translated:.....' #SRC
*
MOVE *TRANSLATE(#SRC, UPPER) TO #DEST
PRINT 'Source string translated into upper case:' #DEST
*
MOVE *TRANSLATE(#SRC, LOWER) TO #DEST
PRINT 'Source string translated into lower case:' #DEST
END
```

Ausgabe:

```
Source string to be translated:..... aBcDeFg !§$%&/()=?

Source string translated into upper case: ABCDEFG !§$%&/()=?

Source string translated into lower case: abcdefg !§$%&/()=?
```

10 *TRIM - Entfernen von führenden und/oder nachfolgenden Leerstellen

```
*TRIM (operand [ ,{ LEADING  } ] )
                  { TRAILING }
```

Format/Länge: wie bei *operand* (A oder B)/DYNAMIC.

Funktion

Die Systemfunktion *TRIM dient zum Entfernen von führenden und/oder nachfolgenden Leerstellen aus einer aphanumerischen oder binären Zeichenkette. Der Inhalt von *operand* bleibt dabei unverändert. Bei Verwendung einer dynamischen Variablen als *operand*, wird die Länge dieser Variablen dem Ergebnis entsprechend angepasst.

Die Systemfunktion *TRIM kann als *operand* an jeder Stelle eines Statements angegeben werden, an der ein Operand mit Format A oder B zulässig ist.

Einschränkungen

Für die Verwendung der Systemfunktion *TRIM gelten folgende Einschränkungen:

- *TRIM darf nicht an Stellen verwendet werden, an denen eine Zielvariable erwartet wird.
- *TRIM darf nicht in einer anderen Systemfunktion verschachtelt werden.
- Wenn der Operand eine statische Variable ist, kann man mit *TRIM keine führenden Leerstellen entfernen, weil bei statischen Variablen die verbleibenden nachfolgenden Stellen des Variablenspeichers mit Leerzeichen aufgefüllt werden.

Syntax-Beschreibung

Operanden-Definitionstabelle:

Operand	Mögliche Struktur					Mögliche Formate											Referenzierung erlaubt	Dynam. Definition
operand	C	S	A			A	U	B									ja	nein

Syntax-Elementbeschreibung:

*TRIM(*operand*, LEADING)	**Führende Leerstellen entfernen** Bei Angabe des Schlüsselworts LEADING als zweites Argument werden alle führenden Leerstellen aus dem in *operand* enthaltenen String entfernt.
*TRIM(*operand*, TRAILING)	**Nachfolgende Leerstellen entfernen** Bei Angabe des Schlüsselworts TRAILING als zweites Argument werden alle nachfolgenden Leerstellen aus dem in *operand* enthaltenen String entfernt.
*TRIM(*operand*)	**Führende und Nachfolgende Leerstellen entfernen** Wenn kein zweites Schlüsselwort als Argument angegeben wird, bewirktTRIM, dass alle führenden und nachfolgenden Leerstellen aus dem in *operand* enthaltenen String entfernt werden.

Beispiele

- Beispiel 1 - Verwendung eines alphanumerischen Arguments
- Beispiel 2 - Verwendung eines binären Arguments

Beispiel 1 - Verwendung eines alphanumerischen Arguments

```
DEFINE DATA LOCAL
/*******************************
/* STATIC VARIABLE DEFINITIONS
/*******************************
1 #SRC  (A15) INIT <'  ab  CD  '>
1 #DEST (A15)

/* FOR PRINT OUT WITH DELIMITERS
1 #SRC-PRN  (A20)
1 #DEST-PRN (A20)

/*******************************
/* DYNAMIC VARIABLE DEFINITIONS
/*******************************
1 #DYN-SRC (A)DYNAMIC INIT <'  ab  CD  '>
1 #DYN-DEST (A)DYNAMIC

/* FOR PRINT OUT WITH DELIMITERS
1 #DYN-SRC-PRN  (A)DYNAMIC
1 #DYN-DEST-PRN (A)DYNAMIC

END-DEFINE

PRINT 'static variable definition:'
```

```
PRINT '---------------------------'
COMPRESS FULL ':' #SRC ':' TO #SRC-PRN LEAVING NO SPACE
PRINT ' '
PRINT ' 123456789012345      123456789012345'

MOVE *TRIM(#SRC, LEADING) TO #DEST
COMPRESS FULL ':' #DEST ':' TO #DEST-PRN LEAVING NO SPACE
DISPLAY #SRC-PRN #DEST-PRN '*TRIM(#SRC, LEADING)'

MOVE *TRIM(#SRC, TRAILING) TO #DEST
COMPRESS FULL ':' #DEST ':' TO #DEST-PRN LEAVING NO SPACE
DISPLAY #SRC-PRN #DEST-PRN '*TRIM(#SRC, TRAILING)'

MOVE *TRIM(#SRC) TO #DEST
COMPRESS FULL ':' #DEST ':' TO #DEST-PRN LEAVING NO SPACE
DISPLAY #SRC-PRN #DEST-PRN '*TRIM(#SRC)'

PRINT ' '
PRINT 'dynamic variable definition:'
PRINT '----------------------------'
COMPRESS FULL ':' #DYN-SRC ':' TO #DYN-SRC-PRN LEAVING NO SPACE
PRINT ' '
PRINT ' 1234567890           12345678'

MOVE *TRIM(#DYN-SRC, LEADING) TO #DYN-DEST
COMPRESS FULL ':' #DYN-DEST ':' TO #DYN-DEST-PRN LEAVING NO SPACE
DISPLAY (AL=20) #DYN-SRC-PRN #DYN-DEST-PRN '*TRIM(#SRC, LEADING)'

MOVE *TRIM(#DYN-SRC, TRAILING) TO #DYN-DEST
COMPRESS FULL ':' #DYN-DEST ':' TO #DYN-DEST-PRN LEAVING NO SPACE
DISPLAY (AL=20) #DYN-SRC-PRN #DYN-DEST-PRN '*TRIM(#SRC, TRAILING)'

MOVE *TRIM(#DYN-SRC) TO #DYN-DEST
COMPRESS FULL ':' #DYN-DEST ':' TO #DYN-DEST-PRN LEAVING NO SPACE
DISPLAY (AL=20) #DYN-SRC-PRN #DYN-DEST-PRN '*TRIM(#SRC)'

PRINT ' '
PRINT '":" := delimiter character to show the start and ending of a string!'
END
```

Ausgabe:

```
      #SRC-PRN             #DEST-PRN
-------------------- ----------------------

static variable definition:
---------------------------

 123456789012345         123456789012345
:  ab  CD       :       :ab  CD         :     *TRIM(#SRC, LEADING)
:  ab  CD       :       :  ab  CD       :     *TRIM(#SRC, TRAILING)
:  ab  CD       :       :ab  CD         :     *TRIM(#SRC)
```

```
dynamic variable definition:
----------------------------

 1234567890              12345678
:  ab  CD  :            :ab  CD  :            *TRIM(#SRC, LEADING)
:  ab  CD  :            :  ab  CD:            *TRIM(#SRC, TRAILING)
:  ab  CD  :            :ab  CD:              *TRIM(#SRC)

':' := delimiter character to show the start and ending of a string!
```

Beispiel 2 - Verwendung eines binären Arguments

```
DEFINE DATA LOCAL
/*******************************
/* STATIC VARIABLE DEFINITIONS
/*******************************
1 #SRC  (B10) INIT <H'2020FFFF2020FFFF2020'>
1 #DEST (B10)

/*******************************
/* DYNAMIC VARIABLE DEFINITIONS
/*******************************
1 #DYN-SRC (B)DYNAMIC INIT <H'2020FFFF2020FFFF2020'>
1 #DYN-DEST (B)DYNAMIC
END-DEFINE

FORMAT LS=100

PRINT 'static variable definition:
PRINT '----------------------------'
MOVE *TRIM(#SRC, LEADING) TO #DEST
PRINT #SRC #DEST '*TRIM(#SRC, LEADING)'

MOVE *TRIM(#SRC, TRAILING) TO #DEST
PRINT #SRC #DEST '*TRIM(#SRC, TRAILING)'

MOVE *TRIM(#SRC) TO #DEST
PRINT #SRC #DEST '*TRIM(#SRC)'

PRINT ' '
PRINT 'dynamic variable definition:'
PRINT '----------------------------'

MOVE *TRIM(#DYN-SRC, LEADING) TO #DYN-DEST
PRINT #DYN-SRC #DYN-DEST '    *TRIM(#SRC, LEADING)'

MOVE *TRIM(#DYN-SRC, TRAILING) TO #DYN-DEST
PRINT #DYN-SRC #DYN-DEST '    *TRIM(#SRC, TRAILING)'

MOVE *TRIM(#DYN-SRC) TO #DYN-DEST
```

```
PRINT #DYN-SRC #DYN-DEST '        *TRIM(#SRC)'

PRINT ' '

PRINT 'hex."20" := space character'
END
```

Ausgabe:

```
static variable definition:
---------------------------

2020FFFF2020FFFF2020 0000FFFF2020FFFF2020     *TRIM(#src, leading)
2020FFFF2020FFFF2020 00002020FFFF2020FFFF     *TRIM(#src, trailing)
2020FFFF2020FFFF2020 00000000FFFF2020FFFF     *TRIM(#src)

dynamic variable definition:
---------------------------

2020FFFF2020FFFF2020 FFFF2020FFFF2020         *TRIM(#src, leading)
2020FFFF2020FFFF2020 2020FFFF2020FFFF         *TRIM(#src, trailing)
2020FFFF2020FFFF2020 FFFF2020FFFF             *TRIM(#src)

hex.'20' := space character
```

11 Natural Functions

This document describes various Software AG Natural functions whose names start with SAG and which are based on the principle of a user defined function; see *User-Defined Functions* in the *Programming Guide*.

These Natural functions are delivered in the library SYSTEM on system file FNAT. Sample function calls are provided in the library SYSEXPG.

URL Encoding

Interfacing Natural applications with HTTP requests often requires that the URI (Uniform Resource Identifiers) is URL-encoded. The REQUEST DOCUMENT statement needs such a URL to access a document.

URL-Encoding (or Percent-Encoding) is a mechanism to replace some special characters in parts of a URL. Only characters of the US-ASCII character set can be used to form a URL. Some characters of the US-ASCII character set have a special meaning when used in a URL - they are classified as „reserved“ control characters, which structure the URL string into different semantic subcomponents. The quasi standard concerning the generic syntax of an URL is laid down in RFC3986, a document composed by the Internet community. It describes under which conditions the URL-Encoding is needed. This includes the representation of characters which are not inside the US-ASCII character set (for example, Euro sign), and it describes the use of reserved characters.

Reserved characters are:

?	=	&	#	!	$	%	'	(	)	*	+	,	/	:	;	@	[	]

Non reserved characters are:

A-Z	a-z	0-9	-	_	.	~

A URL may only consist of reserved and non-reserved characters, other characters are not permitted. If other byte values are needed (which do not correspond to any of the reserved and non-reserved characters) or if reserved characters are used as data (which should not have a special semantic meaning in the URL context), they need to be translated into the „%-encoding“ form - a percent sign, immediately followed by the two-digit hexadecimal representation of the code point, due to the Windows-1252 encoding scheme. This causes a plus sign (+) to appear as %2B, a percent sign (%) to appear as %25 and an at sign (@) to appear as %40 in the string.

The following encoding functions are operating the complete input string. You should take care not to encode a complete URL or parts of it if they contain control characters (reserved characters) which must not be translated into the percent-form. These functions should only be applied to characters not permitted for use in a URL, and to characters with a special meaning inside the URL context, which are supplied as a data item.

Simple Encoding

The single input parameter contains the character string to be encoded or decoded. All data inside is regarded as represented in code page Windows-1252, regardless which session code page is really active at this time. The execution of the `SAGENC/SAGDEC` functions does not require Unicode support. The following characters are replaced with the corresponding US-ASCII hexadecimal equivalents.

Character	<	(	+	|	&	!	$	*	)	;	/	,	%	>	?	`	:	#	@	'	=	"	^	[	]	{	}	\
is encoded to %*nn*	3C	28	2B	7C	26	21	24	2A	29	3B	2F	2C	25	3E	3F	60	3A	23	40	27	3D	22	5E	5B	5D	7B	7D	5C

In addition, a space is replaced with a plus sign (+). All other characters are not translated and remain as they are. The simple encoding function should be sufficient in most cases of URL encoding. The result field returned is of format A dynamic.

The following functions are available:

- `SAGENC` - Simple encoding (format A to format A)
- `SAGDEC` - Simple decoding (format A to format A)

SAGENC - Simple Encoding (Format A to Format A)

The function `SAGENC` encodes a character string into its percent-encoded form. According to standard RFC3986, reserved characters and characters below US-ASCII `x'7F'` (*which are not allowed in a URL*) will be percent-encoded, a space character is replaced with a plus sign (+). Unreserved characters according to RCF3986 and characters above US-ASCII `x'7F'`, such as German umlauts, are not encoded. If you want to encode such characters, use the extended encoding function `SAGENCE`.

SAGENC	This is the simple encoding function call.
SAGENCP	The copycode containing the prototype definition is used at compilation time only in order to determine the type of the return variable for function call reference and to check the parameters, if this is desired. SAGENCP is optional.
URLX01	**Example program** contained in library SYSEXPG. `#URL-ENC := SAGENC(<#URL-DEC>)`

SAGDEC - Simple Decoding (Format A to Format A)

The function SAGDEC decodes the percent-encodings as provided by the function SAGENC. Besides the decoding string, no other input parameters are necessary.

SAGDEC	This is the simple decoding function call.
SAGDECP	The copycode containing the prototype definition is used at compilation time only in order to determine the type of the return variable for function call reference and to check the parameters, if this is desired. SAGDECP is optional.
URLX01	**Example program** contained in library SYSEXPG. `#URL-DEC := SAGDEC(<#URL-ENC>)`

Extended Encoding

The extended function considers all issues which are specified or recommended in RFC 3986. The following parameters may be considered (default settings shown in bold):

1. *<dynamic U-string>* to be encoded/decoded
2. Return code: <> 0 (Natural error) if error in MOVE ENCODED statement.
3. Error character if return code <> 0
4. Space character: %20/+/don't encode (default: **+**)
5. Unreserved characters: encode/**don't encode**
6. Reserved characters: **encode**/don't encode
7. Other special characters (neither unreserved nor reserved): **encode**/don't encode
8. Character Percent-Encoding: ISO-8859-1/UTF-8/any other code page/if = ' ' then *CODEPAGE (default Natural code page, not default encoding code page!)

9. User-selected character in an X-array of format U, which shall not be percent-encoded according to the above parameters, for example, for the Euro sign character, which is not in the ISO-8859-1 code page, or to prevent a character from percent-encoding.
10. User defined percent-encoding in an X-array of format A, for a user-selected character in the same occurrence of the X-array.

The input parameter for a character string will be in Natural format U. This means the input string may contain all Unicode characters. The output string of the extended function is of format A in the Natural default code page (`*CODEPAGE`). The code page of the percent-encoding can be selected. The UTF-8, ISO-8859-1, percent-encoding of the Euro sign will be done by the `MOVE ENCODED` statement. If an input character does not exist in the target code page used for percent-encoding, the character will not be encoded. This means the character will be returned unchanged in the default Natural code page. If the character does not exist in the default Natural code page either, it will be replaced by that substitution character which is returned by the `MOVE ENCODED` statement. The substitution character will be percent-encoded. This may happen only if the percent-encoding code page is not UTF-8. The last `MOVE ENCODED` error will be returned.

The parameters are optional parameters. If the user does not specify a parameter, the default value will be assumed. If the user specifies an own character translation table, the characters in the table will be percent-encoded according to this table and not according to the other parameters. If the percent-encoding of a character in the user-defined translation table is equal to the character or blank, this character will not be encoded. Thus, single characters from the reserved or unreserved character set can be excluded.

The following functions are available:

- `SAGENCE` - Extended encoding (format U to format A, optional parameters)
- `SAGDECE` - Extended decoding (format A to format U, optional parameters)

SAGENCE - Extended Encoding (Format U to Format A, Optional Parameters)

The function `SAGENCE` percent-encodes a string, using the hexadecimal value of the selected code page (default UTF-8). According to standard RFC3986, reserved characters and characters below US-ASCII `x'7F'`, which are not allowed in a URL, will be percent-encoded. Also, the space and the percent sign (%) will be encoded.

In addition, unreserved characters according to RCF3986 and characters above US-ASCII `x'7F'`, such as German umlauts, will be encoded by this function.

SAGENCE needs Natural Unicode support.

SAGENCE	This is the extended encoding function call.
	Parameters:

```
P-DEC-STR-E    (U)
P-RET          (I4)     OPTIONAL  /* 0:    ok
                                  /* else: Natural error returned
                                  /*       by the GIVING clause of
                                  /*       MOVE ENCODED.
                                  /*       This is the error which
                                  /*       comes up when a character
                                  /*       cannot be converted into
                                  /*       the target code page.
/* Error strategy:
/* Step 1: If a character shall be %-encoded and is not available
/* in the code page for %-encoding, the character will not be
/* %-encoded. It will be copied.
/* Step 2: If a character will not be %-encoded but copied from the
/* input format U-variable to a format A-variable (in *CODEPAGE)
/* and the character is not available in *CODEPAGE, a substitution
/* character will be used instead. The substitution character will
/* be %-encoded.
/* The last error will be returned in P-RET.
P-ERR-CHAR     (U1)     OPTIONAL  /* Character causing the error
P-SPACE        (A1)     OPTIONAL  /* '%'  => %20
                                  /* ' '  => ' '
                                  /* else => '+' (default)
P-UNRES        (A1)     OPTIONAL  /* 'E'  => encode
                                  /* else => don't encode (default)
P-RES          (A1)     OPTIONAL  /* 'E'  => encode (default)
                                 /* else => don't encode
P-OTHER        (A1)     OPTIONAL  /* 'E'  => encode (default)
                                  /* else => don't encode
P-CP           (A64)    OPTIONAL  /* IANA name e.g. UTF-8 (default)
                                  /* or ISO-8859-1
/* On mainframe only code page names defined with the macro NTCPAGE
/* in the source module NATCONFG can be used. Other code page names
/* are rejected with a corresponding runtime error.
/*
P-CP-TABLE-CHAR(U1/1:*)  OPTIONAL /* user selected char to be
                                  /* %-encoded, e.g. 'ö' or '/'
P-CP-TABLE-ENC (A12/1:*) OPTIONAL /* user %-encoding
                                  /* e.g. character 'ö'
                                  /*      '%F6'    -> ISO-8859-1
                                  /*      '%C3%B6' -> UTF-8
                                  /* e.g. character '/'
                                  /*      '/'    -> '/' not encoded
                                  /*       although P-RES = 'E'
/* Characters in this table will be encoded according to the
/* specifed %-encoding. If the U12 encoding part is blank (space
```

	`/* according to *CODEPAGE) or the P-CP-TABLE-ENC value is equal to` `/* the character, then the character will not be encoded at all.` `/*`
SAGENCEP	The copycode containing the prototype definition is used at compilation time only in order to determine the type of the return variable for function call reference and to check the parameters, if this is desired. SAGENCEP is optional.
URLX01	**Example program** contained in library SYSEXPG.
	Sample Calls Default values will be taken: `#URL-ENC := SAGENCE(<#URL-DEC-U>)` All possible parameters are specified: `#URL-ENC := SAGENCE(<#URL-DEC-U,L-RET,L-ERR-CHAR,L-SPACE,L-UNRES,` `L-RES,L-OTHER,L-CP,L-CP-TAB-CHAR(*),L-CP-TAB-ENC(*) >)`

SAGDECE - Extended Decoding (Format A to Format U, Optional Parameters)

The function SAGDECE decodes the percent-encodings as provided by the function SAGENCE. If a space character and/or a code page is specified, the values must be the same as specified for encoding.

SAGDECE needs Natural Unicode support.

SAGDECE	This is the extended decoding function call.
	Parameters:
	`1 P-ENC-STR-E    (A)` `1 P-RET          (I4)     OPTIONAL  /* 0:    ok` `                                     /* else: Natural error returned` `                                     /*       by the GIVING clause of` `                                     /*       MOVE ENCODED.` `                                     /*       This error comes up` `                                     /*       when a %-encoded` `                                     /*       character cannot be` `                                     /*       converted into the` `                                     /*       target code page.` `  /* The last error will be returned in P-RET.` `1 P-ERR-CHAR     (A12)    OPTIONAL  /* Error character %-encoded` `1 P-SPACE        (A1)     OPTIONAL  /* ' '  => ' '` `                                     /* else => '+' (default)` `1 P-CP           (A64)    OPTIONAL  /* IANA name e.g. UTF-8 (default)`

	`/* or ISO-8859-1` `/* On mainframe only code page names defined with the macro NTCPAGE` `/* in the source module NATCONFG can be used. Other code page names` `/* are rejected with a corresponding runtime error.` `/*`
SAGDECEP	The copycode containing the prototype definition is used at compilation time only in order to determine the type of the return variable for function call reference and to check the parameters, if this is desired. SAGDECEP is optional.
URLX01	**Example program** contained in library SYSEXPG.
	Sample Calls Default values will be taken: `#URL-DEC-U := SAGDECE(<#URL-ENC>)` All possible parameters are specified: `#URL-DEC-U := SAGDECE(<#URL-ENC,L-RET,L-ERR-CHAR-DEC,L-SPACE,L-CP>)`

Example Program

Example program contained in library SYSEXPG:

```
** Example 'URLX01': ENCODED-STR := SAGENC(<DECODED-STR>)
************************************************************************
DEFINE DATA
LOCAL
1 SAMPLE-STRING (A72)
/*
1 #URL-DEC      (A) DYNAMIC
1 #URL-ENC      (A) DYNAMIC
/*
1 #URL-DEC-U    (U) DYNAMIC
/*
1 L-RET         (I4)   /* Return code
1 L-ERR-CHAR    (U1)   /* Error character
1 L-ERR-CHAR-DEC(A12)  /* Decoded error character
1 L-SPACE       (A1)   /* '%'  => %20, ' '  => ' ',
                       /* else => '+' (default)
1 L-UNRES       (A1)   /* 'E' => encode, else => don't encode (default)
1 L-RES         (A1)   /* 'E' => encode (default), else => don't encode
1 L-OTHER       (A1)   /* 'E' => encode (default), else => don't encode
1 L-CP          (A64)  /* default *CODEPAGE
1 L-CP-TAB-CHAR (U1/1:1)
1 L-CP-TAB-ENC  (A12/1:1)
```

```
1 L-MSG         (U72)
END-DEFINE
/*
/*
/*
WRITE 'Sample string to be processed:'
/* The string below shall be encoded and decoded again.
/* After decoding it should be unchanged.
SAMPLE-STRING := '"Decoded data!"'
WRITE SAMPLE-STRING (AL=72) /
/*
/* Assign the sample string to the input variable #URL-DEC of the
/* simple encoding function.
#URL-DEC      := SAMPLE-STRING
/*
/* Copycode SAGENCP containing the prototype definition is used at
/* compilation time only in order to determine the type of the return
/* variable for function call reference and to check the parameters,
/* if this is desired. SAGENCP is optional.
INCLUDE SAGENCP
/*
/* SAGENC(<#URL-DEC>) is the simple encoding function call.
/*
/* Function SAGENC %-encodes a string to code page ISO-8859-1.
/* According to standard RFC3986 reserved characters and characters
/* below US-ASCII x'7F' which are not allowed in a URL will be
/* %-encoded.
/* Also the space and the percent sign will be encoded.
/* Unreserved characters according to RCF3986 and characters above
/* US-ASCII x'7F' will not be encoded. If you want to encode such
/* characters, use the extended encoding function.
/*
/* ---- Space                        ' ' -> '+'
/* ---- Percent sign                 '%' -> '%25'
/*
/* Unreserved characters according to RFC3986 (will not be encoded!):
/* ---- Period (fullstop)            '.' -- '%2E'
/* ---- Tilde                        '~' -- '%7E'
/* ---- Hyphen                       '-' -- '%2D'
/* ---- Underscore character         '_' -- '%5F'
/* ---- digits, lower and upper case characters
/* ---- 0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ
/*
/* Reserved characters according to RFC3986:
/* ---- Exclamation mark             '!' -> '%21'
/* ---- Number sign                  '#' -> '%23'
/* ---- Dollar sign                  '$' -> '%24'
/* ---- Ampersand                    '&' -> '%26'
/* ---- Apostrophe                   ''' -> '%27'
/* ---- Left parenthesis             '(' -> '%28'
/* ---- Right parenthesis            ')' -> '%29'
/* ---- Asterisk                     '*' -> '%2A'
```

```
/* ---- Plus sign                    '+' -> '%2B'
/* ---- Comma                        ',' -> '%2C'
/* ---- Reverse solidus (backslash) '/' -> '%2F'
/* ---- Colon                        ':' -> '%3A'
/* ---- Semi-colon                   ';' -> '%3B'
/* ---- Equals sign                  '=' -> '%3D'
/* ---- Question mark                '?' -> '%3F'
/* ---- Commercial at                '@' -> '%40'
/* ---- Square bracket open          '[' -> '%5B'
/* ---- Square bracket close         ']' -> '%5D'
/*
/* Other characters below x'7F' (US-ASCII) but not allowed in URL
/* ---- Quotation mark               '"' -> '%22'
/* ---- Less than                    '<' -> '%3C'
/* ---- Greater than                 '>' -> '%3E'
/* ---- Reverse solidus (backslash) '\' -> '%5C'
/* ---- Accent, Circumflex           '^' -> '%5E'
/* ---- Accent, Grave                '`' -> '%60'
/* ---- Opening brace                '{' -> '%7B'
/* ---- Vertical bar                 '|' -> '%7C'
/* ---- Closing brace                '}' -> '%7D'
/*
#URL-ENC := SAGENC(<#URL-DEC>)
/*
/*
WRITE 'Simple function, encoded:'
WRITE #URL-ENC (AL=72)
/*
/* Copycode SAGDECP containing the prototype definition is used at
/* compilation time only in order to determine the type of the return
/* variable for function call reference and to check the parameters,
/* if this is desired. SAGDECP is optional.
INCLUDE SAGDECP
/*
/* SAGDEC(<#URL-ENC>) is the simple decoding function call.
/* It decodes the above described %-encodings.
/*
#URL-DEC := SAGDEC(<#URL-ENC>)
/*
/*
/* The result after encoding and decoding must be equal to the original
/* SAMPLE-STRING.
WRITE 'Simple function, decoded:'
WRITE #URL-DEC (AL=72)
/*
/*
/*
WRITE /
/*
/*
/*
/* Assign the sample string to the input variable #URL-DEC-U of the
```

```
/* enhanced encoding function.
#URL-DEC-U := SAMPLE-STRING
/*
/* Copycode SAGENCEP containing the prototype definition is used at
/* compilation time only in order to determine the type of the return
/* variable for function call reference and to check the parameters,
/* if this is desired. SAGENCEP is optional.
INCLUDE SAGENCEP
/*
/* This is the enhanced encoding function call.
/* The way, characters will be %-encoded dependes on the input
/* parameter of the function.
/* The parameters of the encoding and decoding function are preset
/* with the default values.
/* L-CP-TAB-CHAR(*) and L-CP-TAB-ENC(*) don't have default values.
/* L-CP-TAB-CHAR(1) = 'ä' and L-CP-TAB-ENC(1) = '%C3%A4' will not be
/* used for the sample string '"Decoded data!"'. The string does not
/* contain an 'ä'.
L-SPACE     := '+'           /* encoding and decoding
L-UNRES     := 'D'           /* encoding only
L-RES       := 'E'           /* encoding only
L-OTHER     := 'E'           /* encoding only
L-CP        := 'UTF-8'       /* encoding and decoding
                             /* e.g. ISO-8859-1, UTF-16BE, UTF-32BE
L-CP-TAB-CHAR(1) := 'ä'      /* encoding only
L-CP-TAB-ENC (1) := '%C3%A4' /* encoding only
/*
/* Note that all possible parameters are specified for this sample
/* call.
/* If the default values shall be used and no return code is wanted,
/* all parameters can be omitted, besides the string #URL-DEC-U.
/*
#URL-ENC := SAGENCE(<#URL-DEC-U,L-RET,L-ERR-CHAR,L-SPACE,L-UNRES,
  L-RES,L-OTHER,L-CP,L-CP-TAB-CHAR(*),L-CP-TAB-ENC(*) >)
WRITE 'Extended function, encoded:'
WRITE #URL-ENC (AL=72)
IF L-RET NE 0 THEN
  /* If L-RET = 0, the function worked ok. Else L-RET contains the
  /* Natural error returned by the GIVING clause of MOVE ENCODED.
  /* The error comes up when a character cannot be converted into
  /* the target codepage, e.g. because a character does not exist
  /* in the target codepage.
  COMPRESS 'Error' L-RET 'with MOVE ENCODED of' L-ERR-CHAR INTO L-MSG
  WRITE L-MSG
END-IF
/*
/* Copycode SAGDECEP containing the prototype definition is used at
/* compilation time only in order to determine the type of the return
/* variable for function call reference and to check the parameters,
/* if this is desired. SAGDECEP is optional.
INCLUDE SAGDECEP
/*
```

```
/* This is the 1st enhanced decoding function call with 5 parameters.
/* Note that all possible parameters are specified for this sample
/* call.
/* Since the parameters have the default values, the subsequent
/* function calls return the same result although parameters
/* have been omitted.
#URL-DEC-U := SAGDECE(<#URL-ENC,L-RET,L-ERR-CHAR-DEC,L-SPACE,L-CP>)
WRITE 'Extended function, decoded:'
WRITE #URL-DEC-U (AL=72)
IF L-RET NE 0 THEN
  /* If L-RET = 0, the function worked ok. Else L-RET contains the
  /* Natural error returned by the GIVING clause of MOVE ENCODED.
  /* The error comes up when a %-encoded character cannot be converted
  /* into the target codepage, e.g. because a character does not exist
  /* in the target codepage.
  COMPRESS 'Error' L-RET 'with MOVE ENCODED of' L-ERR-CHAR INTO L-MSG
  WRITE L-MSG
  RESET L-RET
END-IF
/*
/* This is the 2nd enhanced decoding function call with one parameter.
#URL-DEC-U := SAGDECE(<#URL-ENC>)
WRITE #URL-DEC-U (AL=72)
/* L-RET will not be returned
/*
/* This is the 3rd enhanced decoding function call with 3 parameters.
#URL-DEC-U := SAGDECE(<#URL-ENC,L-RET,2X,L-CP>)
WRITE #URL-DEC-U (AL=72)
IF L-RET NE 0 THEN
  COMPRESS 'Error' L-RET 'with MOVE ENCODED of' L-ERR-CHAR INTO L-MSG
  WRITE L-MSG
  RESET L-RET
END-IF
/*
END
```

Base64 Encoding

This section describes Natural functions which can be used to convert binary data into printable, network-compatible data or vice versa, using Base64 conversion.

Base64 conversion means conversion from format B to format A and back to format B, where 6 (binary) bits will be converted into 8 (alphanumerical) bits; for example, a B3 value will be converted into an A4 value.

Anmerkung: Every binary value will be converted into a non-ambiguous alphanumerical value. Re-converting this alphanumerical value again will result in the original binary value.

However, this is not the case for most of the format A to format B and back to format A conversions.

The conversion may be used to transfer a `.bmp` file via TCP/IP, or to transfer Natural binary or integer values via the utility protocol.

On Open Systems only: There are 3 modes available: `RFC3548`, `RFC2045` and `NATRPC` (default). `NATRPC` means the conversion is done according the `NATRPC` logic. This is 100% mainframe compatible. `RFC2045` is the default of the CMBASE64 call. `RFC3548` is like `NATRPC`, but alphanumerical bytes which are not needed are filled with an equals sign character (=).

The following functions are available:

- `SAG64BA` - Binary to Alphanumerical Conversion
- `SAG64AB` - Alphanumerical to Binary Conversion

These two functions together provide the same functionality as the Natural application programming interface `USR4210N`, which is delivered in library `SYSEXT`.

SAG64BA - Binary to Alphanumerical Conversion

The function `SAG64BA` converts binary data into printable, network-compatible data, using Base64 encoding.

`SAG64BA`	This is the binary to alphanumerical format conversion function.
	Parameters:

```
1 PARM-B         (B)      DYNAMIC BY VALUE
                /* Binary source input/target output
1 PARM-RC        (I4)     OPTIONAL
                /* 0:    ok
                /* Mainframe
                /* 1  Source is not numeric
                /* 2  Source is not packed
                /* 3  Source is not floating point
                /* 4  Overflow, source doesn't fit into target
                /* 5  Integer overflow
                /* 6  Source is not a valid date or time
                /* 7  Length error (hex input not even)
                /* 8  Target precision is less than source precision
                /* 9  Float underflow (result->0)
                /* 10 Alpha source contains non-hex characters
                /* 20 Invalid function code
                /* Open Systems
                /* 1  Invalid value for RFC parameter
                /* 2  Invalid function code
                /* 3  CMBASE64: Overflow, source doesn't fit into
                /*              target
```

	`` /* 4 CMBASE64: Non-base64 character found in encoded /* data /* 5 CMBASE64: Out of memory /* 6 CMBASE64: Invalid number of parameters /* 7 CMBASE64: Invalid parameter type /* 8 CMBASE64: Invalid parameter length /* 9 CMBASE64: Invalid function code /* 10 CMBASE64: Unkown return code 1 PARM-ERRTXT (A72) OPTIONAL /* blank, if ok no error /* else error text 1 PARM-RFC (B1) OPTIONAL /* OS only, not used for MF /* 0 - RFC3548; 3 - RFC2045; 4 - NATRPC; ```
SAG64BAP	The copycode containing the prototype definition is used at compilation time only in order to determine the type of the return variable for function call reference and to check the parameters, if this is desired. SAG64BAP is optional.
B64X01	**Example program** contained in library SYSEXPG. Default values will be taken: ``` PARM-A := SAG64BA(<PARM-B>) ``` All possible parameters are specified (PARM-RFC does not apply to mainframe): ``` PARM-A := SAG64BA(<PARM-B,PARM-RC,PARM-ERRTXT,PARM-RFC>) ```

SAG64AB - Alphanumerical to Binary Conversion

The function SAG64AB converts printable, network-compatible data into binary data, using Base64 encoding.

SAG64AB	This is the alphanumerical to binary format conversion function.
	Parameters:
	`` 1 PARM-A (A) /* Alpha source input/target output 1 PARM-RC (I4) OPTIONAL /* 0: ok /* Mainframe /* 1 Source is not numeric /* 2 Source is not packed /* 3 Source is not floating point /* 4 Overflow, source doesn't fit into target /* 5 Integer overflow ```

	`` /* 6 Source is not a valid date or time /* 7 Length error (hex input not even) /* 8 Target precision is less than source precision /* 9 Float underflow (result->0) /* 10 Alpha source contains non-hex characters /* 20 Invalid function code /* Open Systems /* 1 Invalid value for RFC parameter /* 2 Invalid function code /* 3 CMBASE64: Overflow, source doesn't fit into /* target /* 4 CMBASE64: Non-base64 character found in encoded /* data /* 5 CMBASE64: Out of memory /* 6 CMBASE64: Invalid number of parameters /* 7 CMBASE64: Invalid parameter type /* 8 CMBASE64: Invalid parameter length /* 9 CMBASE64: Invalid function code /* 10 CMBASE64: Unkown return code 1 PARM-ERRTXT (A72) OPTIONAL /* blank, if ok no error /* else error text 1 PARM-RFC (B1) OPTIONAL /* OS only, not used for MF /* 0 - RFC3548; 3 - RFC2045; 4 - NATRPC; ```
SAG64ABP	The copycode containing the prototype definition is used at compilation time only in order to determine the type of the return variable for function call reference and to check the parameters, if this is desired. SAG64ABP is optional.
B64X01	**Example program** contained in library SYSEXPG. Default values will be taken: `PARM-B := SAG64AB(<PARM-A>)` All possible parameters are specified (PARM-RFC does not apply to mainframe): `PARM-B := SAG64AB(<PARM-A,PARM-RC,PARM-ERRTXT,PARM-RFC>)`

Example Program

Example program `B64X01` contained in library `SYSEXPG`:

```
** Example 'B64X01': BASE64-A-STR := SAG64BA(<BASE64-B-STR>)
************************************************************************
* Function ......... Convert binary data into printable,
*                    network-compatible data or vice versa using
*                    Base64 encoding.
*
*                    Base64 encoding means (B) -> (A) -> (B),
*                    where 6 (binary) bits will be encoded into 8
*                    (alpha) bits, e.g a (B3) value will be encoded
*                    into a (A4) value.
*
*                    Note: Every binary value will be encoded into
*                    a non-ambiguous alpha value. Re-encoding this
*                    alpha value again will result in the original
*                    binary value. However, this is not the case with
*                    most of the (A) -> (B) -> (A) encodings.
*
*                    The encoding may be used to transfer a .bmp
*                    file via TCP/IP, or to transfer Natural binary or
*                    integer values via the utility protocol.
*
*                    Open Systems only:
*                    On Open Systems, there are 3 modes:
*                    RFC3548, RFC2045 and NATRPC (default).
*                    NATRPC means the encoding follows
*                    the NATRPC logic. This is 100% MF compatible.
*                    RFC2045 is the default of the CMBASE64 call.
*                    RFC3548 is like NATRPC, but alpha bytes not
*                    needed are filled with '='.
*
DEFINE DATA
LOCAL
1 FUNCTION        (A2)
                  /* 'AB' Alpha to binary encoding
                  /* 'BA' Binary to alpha encoding
1 PARM-RC         (I4)
                  /* 0:    ok
                  /* Mainframe
                  /* 1  Source is not numeric
                  /* 2  Source is not packed
                  /* 3  Source is not floating point
                  /* 4  Overflow, source doesn't fit into target
                  /* 5  Integer overflow
                  /* 6  Source is not a valid date or time
                  /* 7  Length error (hex input not even)
                  /* 8  Target precision is less than source precision
                  /* 9  Float underflow (result->0)
```

```
                  /* 10 Alpha source contains non-hex characters
                  /* 20 Invalid function code
                  /* Open Systems
                  /* 1  Invalid value for RFC parameter
                  /* 2  Invalid function code
                  /* 3  CMBASE64: Overflow, source doesn't fit into
                  /*              target
                  /* 4  CMBASE64: Non-base64 character found in encoded
                  /*              data
                  /* 5  CMBASE64: Out of memory
                  /* 6  CMBASE64: Inalid number of parameters
                  /* 7  CMBASE64: Invalid parameter type
                  /* 8  CMBASE64: Invalid parameter length
                  /* 9  CMBASE64: Invalid function code
                  /* 10 CMBASE64: Unkown return code
1 PARM-ERRTXT     (A72)
                  /* blank, if ok no error
                  /* else error text
1 PARM-A          (A)   DYNAMIC
                  /* Alpha source input/target output
1 PARM-B          (B)   DYNAMIC
*                 /* Binary source input/target output
1 PARM-RFC        (B1)
                  /* OS only, not used for MF
                  /* 0 - RFC3548; 3 - RFC2045; 4 - NATRPC;
/*
1 #BACKUP-A       (A) DYNAMIC
1 #BACKUP-B       (B) DYNAMIC
END-DEFINE
/*
/*
SET KEY ALL
/*
/* Copycode SAG64BAP and SAG64ABP containing the prototype definition
/* is used at compilation time only in order to determine the type of
/* the return variable for function call reference and to check the
/* parameters, if this is desired. SAG64BAP and SAG64ABP are optional.
INCLUDE SAG64BAP
INCLUDE SAG64ABP
/*
REPEAT
  RESET PARM-A PARM-B
  REDUCE DYNAMIC PARM-A TO 0
  REDUCE DYNAMIC PARM-B TO 0
  FUNCTION := 'BA'
  PARM-B := H'0123456789ABCDEF'
  INPUT (AD=MIL IP=OFF CD=NE) WITH TEXT PARM-ERRTXT
   // 10T 'Base64 Encoding:' (YEI)
    / 10T '-' (19) (YEI) /
    / 10T 'Function (BA,AB) ..' (TU) FUNCTION (AD=T)
    / 10T 'Alpha In/Output ...' (TU) PARM-A (AL=30)
    / 10T 'Binary In/Output ..' (TU) PARM-B (EM=HHHHHHHH)
```

```
  / 10T 'Response ..........' (TU) PARM-RC (AD=OD CD=TU)
  / PARM-ERRTXT (AD=OD CD=TU)
RESET PARM-ERRTXT
IF *PF-KEY NE 'ENTR'
  ESCAPE BOTTOM
END-IF
/*
RESET #BACKUP-A #BACKUP-B
REDUCE DYNAMIC #BACKUP-A TO 0
REDUCE DYNAMIC #BACKUP-B TO 0
#BACKUP-A := PARM-A
#BACKUP-B := PARM-B
/*
IF FUNCTION = 'BA'
  /* Parameter PARM-RC, PARM-ERRTXT and PARM-RFC are optional
  /* Parameter PARM-RFC does not apply to mainframe
  /* PARM-A := SAG64BA(<PARM-B,PARM-RC,PARM-ERRTXT,PARM-RFC>)
  PARM-A := SAG64BA(<PARM-B,PARM-RC,PARM-ERRTXT>)
  /* PARM-A := SAG64BA(<PARM-B,PARM-RC>)
  /* PARM-A := SAG64BA(<PARM-B>)
ELSE
  /* Parameter PARM-RC, PARM-ERRTXT and PARM-RFC are optional
  /* Parameter PARM-RFC does not apply to mainframe
  /* PARM-B := SAG64AB(<PARM-A,PARM-RC,PARM-ERRTXT,PARM-RFC>)
  PARM-B := SAG64AB(<PARM-A,PARM-RC,PARM-ERRTXT>)
  /* PARM-B := SAG64AB(<PARM-A,PARM-RC>)
  /* PARM-B := SAG64AB(<PARM-A>)
END-IF
/*
IF PARM-RC NE 0 THEN
  WRITE 'Encoding' FUNCTION
  WRITE NOTITLE PARM-ERRTXT
ELSE
  IF FUNCTION = 'BA' THEN
    WRITE 'Binary -> Alpha'
    WRITE '=' PARM-B (EM=HHHHHHHHHHHHHHHHHHHHHHHHHH)
      / '=' PARM-A (AL=50)
    RESET PARM-B
    REDUCE DYNAMIC PARM-B TO 0
    FUNCTION := 'AB'
  ELSE
    WRITE 'Alpha -> Binary'
    WRITE '=' PARM-A (AL=50) /
      '=' PARM-B (EM=HHHHHHHHHHHHHHHHHHHHHHHHHH)
    RESET PARM-A
    REDUCE DYNAMIC PARM-A TO 0
    FUNCTION := 'BA'
  END-IF
  /*
  IF FUNCTION = 'BA'
    /* Parameter PARM-RC, PARM-ERRTXT and PARM-RFC are optional
    /* Parameter PARM-RFC does not apply to mainframe
```

```
      /* PARM-A := SAG64BA(<PARM-B,PARM-RC,PARM-ERRTXT,PARM-RFC>)
      PARM-A := SAG64BA(<PARM-B,PARM-RC,PARM-ERRTXT>)
      /* PARM-A := SAG64BA(<PARM-B,PARM-RC>)
      /* PARM-A := SAG64BA(<PARM-B>)
    ELSE
      /* Parameter PARM-RC, PARM-ERRTXT and PARM-RFC are optional
      /* Parameter PARM-RFC does not apply to mainframe
      /* PARM-B := SAG64AB(<PARM-A,PARM-RC,PARM-ERRTXT,PARM-RFC>)
      PARM-B := SAG64AB(<PARM-A,PARM-RC,PARM-ERRTXT>)
      /* PARM-B := SAG64AB(<PARM-A,PARM-RC>)
      /* PARM-B := SAG64AB(<PARM-A>)
    END-IF
    IF PARM-RC NE 0 THEN
      WRITE 'Encoding' FUNCTION
      WRITE NOTITLE PARM-ERRTXT
    ELSE
      IF FUNCTION = 'BA' THEN
        WRITE 'Binary -> Alpha'
        WRITE '=' PARM-B (EM=HHHHHHHHHHHHHHHHHHHHHHHHHH)
          / '=' PARM-A (AL=50)
        IF PARM-A = #BACKUP-A THEN
          WRITE '******** Encoding successful ********'
        ELSE
          WRITE '******** Value changed by encoding ********'
        END-IF
      ELSE
        WRITE 'Alpha -> Binary'
        WRITE '=' PARM-A (AL=50) /
          '=' PARM-B (EM=HHHHHHHHHHHHHHHHHHHHHHHHHH)
        IF PARM-B = #BACKUP-B THEN
          WRITE '******** Encoding successful ********'
        ELSE
          WRITE '******** Value changed by encoding ********'
        END-IF
      END-IF
    END-IF
  END-IF
END-REPEAT
END
```