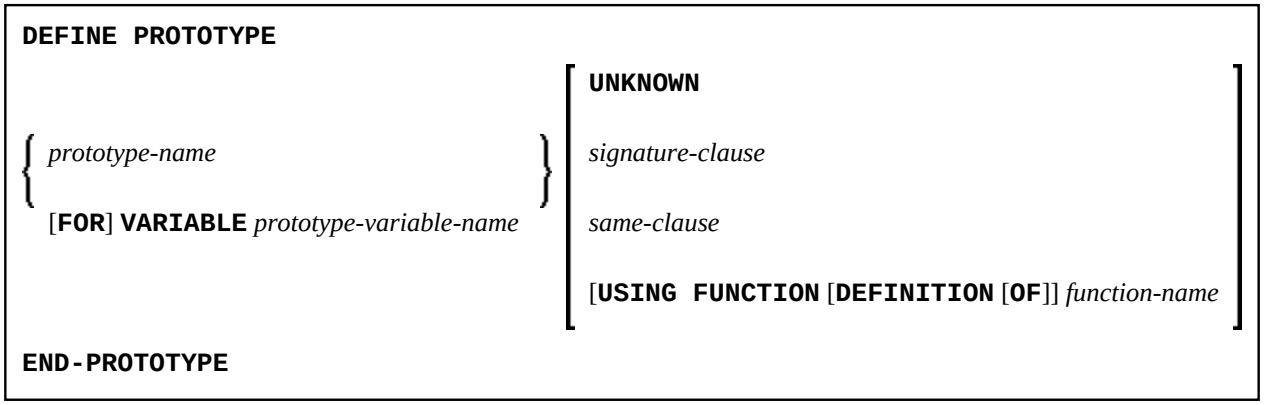
DEFINE PROTOTYPE



Dieses Kapitel behandelt folgende Themen:

- Funktion
- Syntax-Beschreibung
- Beispiel

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt *Syntax-Symbole*.

Verwandtes Statement: `DEFINE FUNCTION`

Funktion

Die Prototypdefinition kann dazu benutzt werden, eine Signatur gemäß einem bestimmen Function Call anzugeben. Für jeden Function Call müssen der Rückgabetyp und die Art des Function Calls (`VARIABLE`) bekannt sein. Deshalb müssen diese Daten bei jedem Function Call zur Verfügung stehen. Wenn diese Daten fehlen, muss das Prototypschlüsselwort in der Function Call-Referenz benutzt werden. Wenn es eine Parameterdefinition im Prototyp gibt, werden die Parameterwerte des Function Call mit den Parametern der Prototypdefinition verglichen. Wenn die Parameter nicht geprüft werden sollen, benutzen Sie das Schlüsselwort `UNKNOWN` im `DEFINE DATA PARAMETER`-Statement der Prototypdefinition.

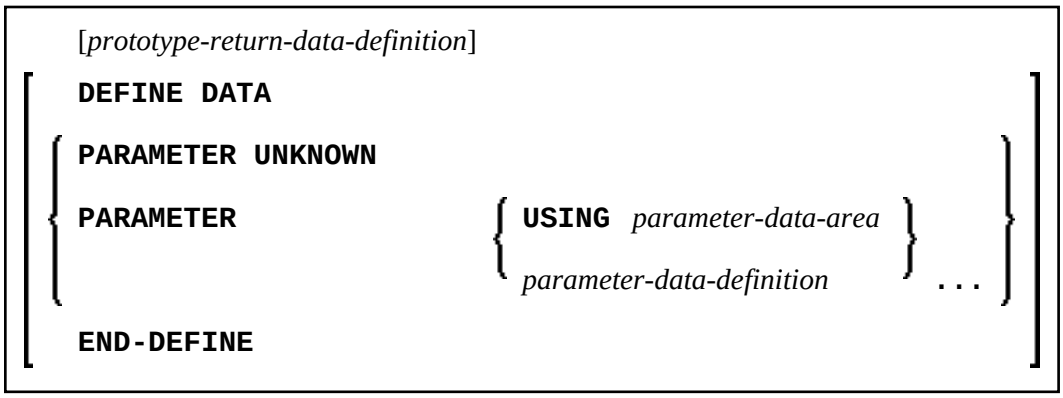
Weitere Informationen finden Sie in den folgenden Abschnitten im *Leitfaden zur Programmierung*:

- Natural-Objecttyp Function
- *Function Call*
- *Benutzerdefinierte Funktionen*

Syntax-Beschreibung

prototype-name	Für den *prototype-name* gelten dieselben Regeln wie für Benutzervariablen - mit einer Ausnahme: Prototypnamen dürfen Punkte (.) enthalten. Der *prototype-name* ist frei wählbar. Es ist nicht erforderlich, dass er denselben Namen hat wie die entsprechende Funktionsdefinition. Die maximale Länge des kompletten *prototype-name* beträgt 32 Zeichen.
VARIABLE *prototype-variable-name*	Mit dem *prototype-variable-name* können Sie Functions mit variablen Funktionsnamen aufrufen. Das ist ähnlich wie bei den `CALLNAT`-Funktionsaufrufen. Der *prototype-variable-name* ist der Name einer alphanumerischen Variable, die den richtigen Namen der Function enthält, die in der Funktionsreferenz aufgerufen werden soll.
UNKNOWN	Wenn die Parameter nicht geprüft werden sollen, benutzen Sie das Schlüsselwort `UNKNOWN` im `DEFINE DATA PARAMETER`-Statement der Prototypdefinition.
signature-clause	Siehe *Signature-Klausel* unten.
prototype-return-data-definition	Siehe *Prototyp-Rückgabewert-Definition* unten.
same-clause	Siehe *SAME AS-Klausel* unten.
USING FUNCTION [DEFINITION [OF]] *function-name*	Siehe *USING FUNCTION-Klausel* unten.
END-PROTOTYPE	Das für Natural reservierte Schlüsselwort `END-PROTOTYPE` muss zum Beenden des `DEFINE PROTOTYPE`-Statements benutzt werden.

Signature-Klausel

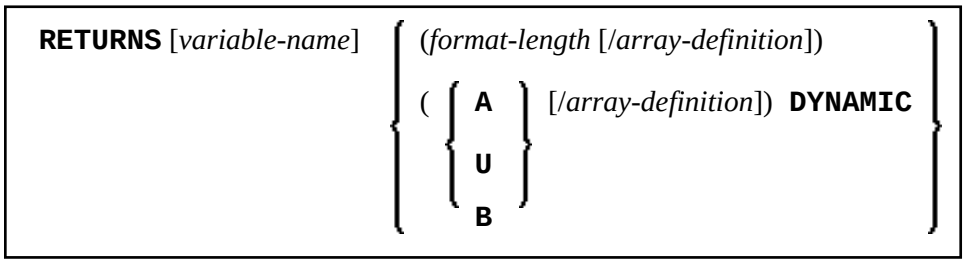


Diese Klausel sieht aus wie ein bestimmter Function Call. Normalerweise stimmt der Prototyp mit der Funktionsdefinition überein. Er muss jedoch nicht exakt derselbe sein. So ist es möglich, die Parameterdaten wegzulassen und statt dessen das Schlüsselwort `UNKNOWN` zu benutzen. In diesem Fall werden die Parameter zum Kompilierzeitpunkt nicht geprüft.

Der Typ des Rückgabewerts muss in jedem Fall gesetzt werden. Wenn kein Rückgabewert definiert ist, dann ist die Zuweisung vom Function Call an eine Variable nicht erlaubt.

Wenn in einer Prototypdefinition keine Signatur angegeben ist (Signatur ist `UNKNOWN`), muss die entsprechende Signatur eines Function Call mit dem Schlüsselwort `PT` angegeben werden. Weitere Informationen zu `PT` finden Sie unter *Function Call*, Abschnitt *prototype-cast*, im *Leitfaden zur Programmierung*.

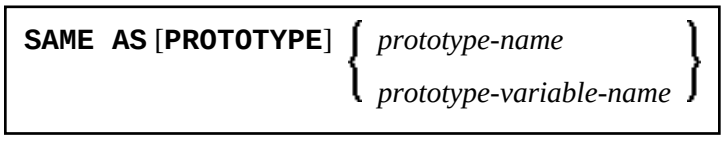
Prototyp-Rückgabewert-Definition



Diese Klausel definiert Format und Länge (*format-length*) des Rückgabewerts, der zum Kompilierzeitpunkt bekannt sein muss.

Der optionale Variablenname (*variable-name*) wird ignoriert. Er wurde eingeführt, damit die Syntaxstruktur ähnlich ist wie bei der `RETURNS`-Klausel des `DEFINE FUNCTION`-Statements.

SAME AS -Klausel



Diese Klausel kann dafür benutzt werden, Signaturen zu benutzen, die vorher definiert wurden, um einen neuen Prototyp zu definieren.

USING FUNCTION-Klausel

[**USING FUNCTION** [**DEFINITION** [**OF**]] *function-name*]

Diese explizite Klausel bietet Ihnen die Möglichkeit, ein generiertes Objekt auf Funktionsparameterdefinitionen zu analysieren, die dann dazu verwendet werden, ein indirektes `DEFINE PROTOTYPE`-Statement unter dem logischen Namen dieser Funktion zu erstellen. *function-name* ist der logische Name; er ist nicht der Objektname des Function-Objekts. Der logische Function-Name ist im Funktionsrumpf (Body) des entsprechenden Function-Objekts definiert: `DEFINE FUNCTION` *`function-name`* `... END-FUNCTION`.

Beispiel

- Beispiel 1 - DEFINE PROTOTYPE
- Beispiel 2 - DEFINE PROTOTYPE

Beispiel 1 - DEFINE PROTOTYPE

Dies ist eine Prototypdefinition einer Function mit dem Namen `GET-FIRST-BYTE`. Mit dem folgenden Prototyp kann die Function `GET-FIRST-BYTE` als symbolischer Function Call aufgerufen werden.

```
GET-FIRST-BYTE(<#A>)

DEFINE DATA LOCAL
1 #A(A10) INIT <'abcdefghij'>
END-DEFINE
DEFINE PROTOTYPE GET-FIRST-BYTE
  RETURNS (A1)
  DEFINE DATA PARAMETER
  1 PARM1(A10)
  END-DEFINE
END-PROTOTYPE
WRITE GET-FIRST-BYTE(<#A>)
END
```

Beispiel 2 - DEFINE PROTOTYPE

Der folgende Natural-Code enthält die Prototypdefinition einer Function, in diesem Fall `GET-FIRST-BYTE`. Damit die Function dynamisch aufgerufen werden kann, muss der Name der Function in der alphanumerischen Variablen `#A` gespeichert sein. Bevor die Variable `#A` benutzt werden kann, muss sie als alphanumerische Variable im `DEFINE DATA`-Statement definiert werden.

```
DEFINE DATA LOCAL
1 FUNCTION-NAME(A32) INIT<’GET-FIRST-BYTE’>
1 #A(A10) INIT <’abcdefghij’>
END-DEFINE
DEFINE PROTOTYPE VARIABLE FUNCTION-NAME
  RETURNS (A1)
  DEFINE DATA PARAMETER
  1 PARM1(A10)
  END-DEFINE
END-PROTOTYPE
WRITE FUNCTION-NAME(<#A>)
END
```