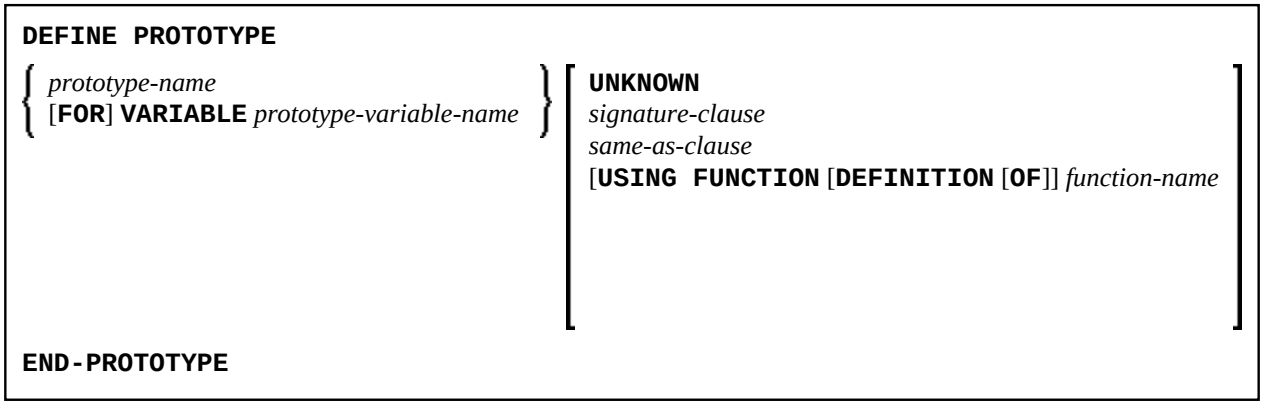
DEFINE PROTOTYPE



This chapter covers the following topics:

- Function
- Syntax Description
- Example

For an explanation of the symbols used in the syntax diagram, see *Syntax Symbols*.

Related Statement: `DEFINE FUNCTION`

Function

The prototype definition may be used to specify a signature according to a certain function call. For each function call, the return type must be known, as well as the kind of the function call (`VARIABLE`). Therefore, this data must be available for each function call. If any of this data is missing, the prototype keyword must be used inside the function call reference. If there is a parameter definition inside the prototype, the parameter values of the function call are compared with the parameters of the prototype definition. If the parameters should not be checked, use the `UNKNOWN` keyword inside the `DEFINE DATA PARAMETER` statement of the prototype definition.

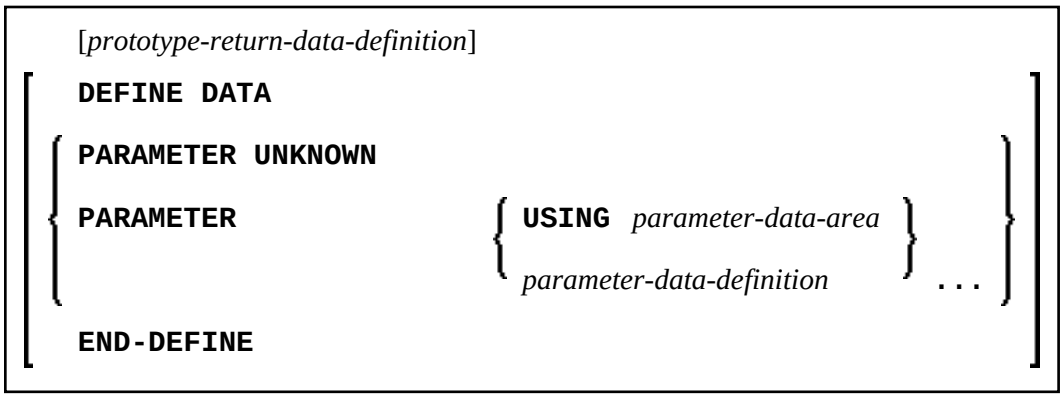
For further information, see the following sections in the *Programming Guide*:

- Natural object type Function
- *Function Call*
- *User-Defined Functions*

Syntax Description

Syntax Element	Description
prototype-name	The *prototype-name* must follow the same rules as those used for defining user-defined variables - with one exception: prototype names may contain periods (.). The *prototype-name* is completely arbitrary. It is not necessary for it to have the same name as a corresponding function definition. The maximum length of the whole *prototype-name* is 32 characters.
VARIABLE *prototype-variable-name*	The *prototype-variable-name* allows you to call functions using variable function names. This is similar to CALLNAT function calls. The *prototype-variable-name* is the name of an alphanumeric variable containing the real name of the function, which is to be called in the function reference.
UNKNOWN	If the parameters should not be checked, use the UNKNOWN keyword inside the DEFINE DATA PARAMETER statement of the prototype definition.
signature-clause	See *Signature Clause* below.
prototype-return-data-definition	See *Prototype Return Data Definition* below.
same-clause	See *SAME AS Clause* below.
USING FUNCTION [DEFINITION [OF]] *function-name*	See *USING FUNCTION Clause* below.
END-PROTOTYPE	The Natural reserved word END-PROTOTYPE must be used to terminate the DEFINE PROTOTYPE statement.

Signature Clause

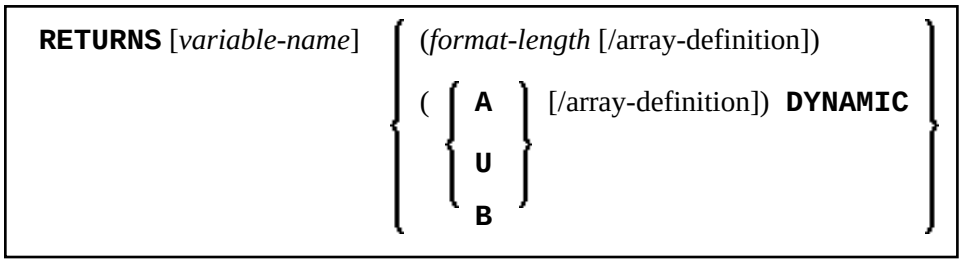


This clause looks like a certain function call. Normally, the prototype agrees with the function definition. But it does not need to be exactly the same. So it is possible to omit the parameter data and to set the keyword `UNKNOWN` instead. In this case, there will be no parameter checking at compilation time.

The type of the return value must be set in every case. If no return value is defined, there is no assignment allowed from the function call to a variable.

If no signature is specified inside a prototype definition (signature is `UNKNOWN`), the corresponding signature of a function call must be specified using the keyword `PT`. For further information about `PT`, see *Function Call*, section *prototype-cast* in the *Programming Guide*.

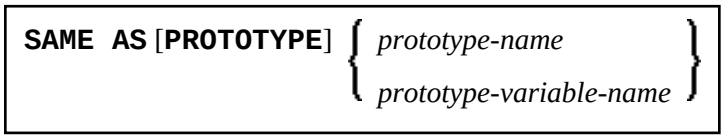
Prototype Return Data Definition



This clause defines the *format-length* of the return value which must be known at compilation time.

The optional variable name is ignored. It was introduced to have a syntax structure similar to the `RETURNS` clause of the `DEFINE FUNCTION` statement.

SAME AS Clause



This clause may be used in order to use signatures previously defined to define a new prototype.

USING FUNCTION Clause

This explicit clause offers you the possibility to analyze a generated object for function parameter definitions which will then be taken to create an indirect `DEFINE PROTOTYPE` statement under the logical name of that function. *`function-name`* is the logical name, but not the object name of the function object. The logical function name is defined in the function body of the corresponding function object: `DEFINE FUNCTION` *`function-name`* `... END-FUNCTION`.

Example

- Example 1 - DEFINE PROTOTYPE
- Example 2 - DEFINE PROTOTYPE

Example 1 - DEFINE PROTOTYPE

This is a prototype definition of a function named `GET-FIRST-BYTE`. Using the following prototype, the function `GET-FIRST-BYTE` can be called as a symbolic function call:

```
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
```

Example 2 - DEFINE PROTOTYPE

The following Natural code contains the prototype definition of a function, in this case `GET-FIRST-BYTE`. In order to be able to call the function dynamically, the name of the function must be stored inside the alphanumeric variable `#A`. The variable `#A` must be defined as an alphanumeric variable within the `DEFINE DATA` statement definition before it may be used.

```
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
```