INTERFACE

```
INTERFACE  interface-name
  [property-definition]
  [method-definition]
END-INTERFACE
```

This chapter covers the following topics:

- Function
- Syntax Description

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

Related Statements: CREATE OBJECT | DEFINE CLASS | INTERFACE | METHOD | PROPERTY | SEND METHOD

Belongs to Function Group: *Component Based Programming*

Function

In component-based programming, an interface is a collection of methods and properties that belong together semantically and represent a certain feature of a class.

You can define one or several interfaces for a class. Defining several interfaces allows you to structure/group methods according to what they do, e.g., you put all methods that deal with persistency (load, store, update) in one interface and put other methods in other interfaces.

The `INTERFACE` statement is used to define an interface. It may only be used in a Natural class module and can be defined as follows:

- within a `DEFINE CLASS` statement. This form is used when the interface is only to be implemented in one class, or
- in a copycode which is included by the `INTERFACE USING` clause of the `DEFINE CLASS` statement. This form is used when the interface is to be implemented in more than one class.

The properties and methods that are associated with the interface are defined by the property and method definitions.

Syntax Description

interface-name	This is the name to be assigned to the interface. The interface name can be up to a maximum of 32 characters long and must conform to the Natural naming conventions for user-defined variables; see *Naming Conventions for User-Defined Variables* in the *Using Natural* documentation. It must be unique per class and different from the class name. If the interface is planned to be used by clients written in different programming languages, the interface name should be chosen in a way that it does not conflict with the naming conventions that apply in these languages.
property-definition	The property definition is used to define a property of the interface. See *Property Definition* below.
method-definition	The method definition is used to define a method for the interface. See *Method Definition* below.
END-INTERFACE	The Natural reserved word `END-INTERFACE` must be used to end the `INTERFACE` statement.

Property Definition

The property definition is used to define a property of the interface.

```
PROPERTY property-name
  [(format-length/array-definition)]
  [READONLY]
  [IS operand]
END-PROPERTY
```

Properties are attributes of an object that can be accessed by clients. An object that represents an employee might for example have a `Name` property and a `Department` property. Retrieving or changing the name or department of the employee by accessing her `Name` or `Department` property is much simpler for a client than calling one method that returns the value and another method that changes the value.

Each property needs a variable in the object data area of the class to store its value - this is referred to as the object data variable. The property definition is used to make this variable accessible to clients. The property definition defines the name and format of the property and connects it to the object data variable. In the simplest case, the property takes the name and format of the object data variable itself. It is also possible to override the name and format within certain limits.

property-name	This is the name to be assigned to the property. The property name can contain up to a maximum of 32 characters and must conform to the Natural naming conventions for user variables; see *Naming Conventions for User-Defined Variables* in the *Using Natural* documentation. If the property is planned to be used by clients written in different programming languages, the property name should be chosen in a way that it does not conflict with the naming conventions that apply in these languages.
format-length/array-definition	This defines the format of the property as it will be seen by clients. If *format-length/array-definition* is omitted, the format-length and array-definition will be taken from the object data variable assigned in the `IS` clause. If *format-length/array-definition* is specified, it must be data transfer-compatible both to and from the format of the object data variable specified in *operand* in the `IS` clause. In the case of a `READONLY` property, the data transfer-compatibility needs to hold only in one direction: with the object data variable as source operand and the property as destination operand. If an array-definition is specified, it must be equal in dimensions, occurrences per dimension, lower bounds and upper bounds to the array definition of the corresponding object data variable. This is expressed by specifying an asterisk for each dimension.
READONLY	If this keyword is specified, the value of the property can only be read and not set. The format of the object data variable specified in *operand* in the `IS` clause must be data transfer-compatible to the format specified in *format-length/array-definition*. It does not have to be data transfer-compatible in the inverse direction. If the keyword `READONLY` is omitted, the property value can be both read and set.
IS *operand*	The *operand* in the `IS` clause assigns an object data variable as the place to store the property value. The assigned object data variable may not be a group. The variable is referenced in normal operand syntax. This means that if the object data variable is an array, it must be referenced with index notation. Only the full index range notation and asterisk notation is allowed. The `IS` clause should not be used if the `INTERFACE` statement will be included from a copycode member and reused in several classes. If you want to reuse the `INTERFACE` statement, you must assign the object data variable in a `PROPERTY` statement outside the `INTERFACE` statement. If the `IS` clause is omitted, the property is connected to the object data variable with the same name as the property. If a variable with this name is not defined or if it is a group, a syntax error results.

END-PROPERTY	The Natural reserved word END-PROPERTY must be used to end the interface PROPERTY definition.

Examples

Let the object data area contain the following data definitions:

```
1 Salary(p7.2)
1 SalaryHistory(p7.2/1:10)
```

Then the following property definitions are allowed:

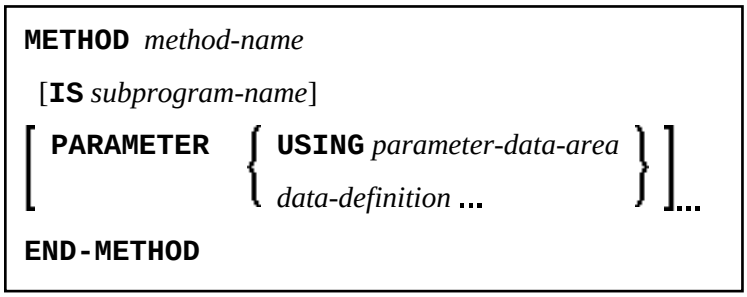
```
property Salary
  end-property
  property Pay is Salary
  end-property
  property Pay(P7.2) is Salary
  end-property
  property Pay(N7.2) is Salary
  end-property
  property SalaryHistory
  end-property
  property OldPay is SalaryHistory(*)
  end-property
  property OldPay is SalaryHistory(1:10)
  end-property
  property OldPay(P7.2/*) is SalaryHistory(1:10)
  end-property
  property OldPay(N7.2/*) is SalaryHistory(*)
  end-property
```

The following property definitions are not allowed:

```
/* Not data transfer-compatible. */
  property Pay(L) is Salary
  end-property
  /* Not data transfer-compatible. */
  property OldPay(L/*) is SalaryHistory(*)
  end-property
  /* Not data transfer-compatible. */
  property OldPay(L/1:10) is SalaryHistory(1:10)
  end-property
  /* Assigns an array to a scalar. */
  property OldPay(P7.2) is SalaryHistory(1:10)
  end-property
  /* Takes only a sub-array. */
  property OldPay(P7.2/3:5) is SalaryHistory(*)
  end-property
  /* Index specification omitted in ODA variable SalaryHistory. */
  property OldPay is SalaryHistory
  end-property
  /* Only asterisk notation allowed in property format specification. */
  property OldPay(P7.2/1:10) is SalaryHistory(*)
  end-property
```

Method Definition

The method definition is used to define a method for the interface.



To make the interface reusable in different classes, include the interface definition from a copycode and define the subprogram after the interface definition with a `METHOD` statement. Then you can implement the method differently in different classes.

method-name	This is the name to be assigned to the method. The method name can contain a maximum of up to 32 characters and must conform to the Natural naming conventions; see *Naming Conventions for User-Defined Variables* in the *Using Natural* documentation. It must be unique per interface. If the method is planned to be used by clients written in different programming languages, the method name should be chosen in a way that it does not conflict with the naming conventions that apply in these languages.
IS ***subprogram-name***	This is the name of the subprogram that implements the method. The name of the subprogram consists of up to 8 characters. The default is method-name (if the `IS` clause is not specified).
PARAMETER	This specifies the parameters of the method, and has the same syntax as the `PARAMETER` clause of the `DEFINE DATA` statement. The parameters must match the parameters which are later used in the implementation of the subprogram. This is ensured best by using a parameter data area. Parameters that are marked `BY VALUE` in the parameter data area are input parameters of the method. Parameters which are not marked `BY VALUE` are passed "by reference" and are input/output parameters. This is the default. The first parameter that is marked `BY VALUE RESULT` is returned as the return value for the method. If more than one parameter is marked in this way, the others will be treated as input/output parameters. `OPTIONAL` parameters need not be specified when the method is called. They can be left unspecified by using the *n*`X` notation in the `SEND METHOD` statement.
END-METHOD	The Natural reserved word `END-METHOD` must be used to end the `METHOD` definition for the interface.