

INTERFACE

```
INTERFACE      interface - name

               [EXTERNAL]
               [ID interface - GUID]
               [property - definition]
               [method - definition]

END - INTERFACE
```

Function

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.

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 variables (please refer to the section Statement Usage Related Topics for further information). 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. Bolero for example uses the Java naming convention. So, an interface that is planned to be used in a Bolero client should also respect the Java naming conventions.

EXTERNAL

Note:

This clause applies only on Windows platforms. It is only available with Natural version 5.1.1 and above.

The EXTERNAL clause is used to indicate that this interface is implemented by the class, but which is originally defined in a different class. The clause is only relevant if the class is to be registered with DCOM. Interfaces with the EXTERNAL clause are ignored when the class is registered with DCOM. It is assumed that the interface is registered by the class that originally defines it.

ID Clause

This clause applies only on Windows platforms.

The ID clause is used to assign a globally unique ID to the interface. The interface GUID is the name of a GUID defined in a data area that is included by the LOCAL clause. The interface GUID is a (named) alpha constant. A GUID must be assigned to an interface if the class is to be registered with DCOM.

property-definition

```

PROPERTY      property - name

               [format - length/array - definition]
               [ID dispatch - ID]
               [READONLY]
               [IS operand]

END - PROPERTY

```

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

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 (refer to the section Statement Usage Related Topics for further information).

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. Bolero for example uses the Java naming convention. So, a property that is planned to be used in a Bolero client should also respect the Java naming conventions.

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.

ID Clause

Note:

This clause applies only on Windows platforms. It is only available with version 5.1.1 and above.

The ID clause is used to assign a specific numeric identifier to a property. This identifier (the so-called dispatch ID) is only relevant if the class is to be registered with DCOM.

Normally, Natural automatically assigns a dispatch ID to a property. It is only necessary to explicitly define a specific dispatch ID for a property if the property belongs to an interface with the EXTERNAL clause. (This is an interface that shall be implemented in this class, but which is originally defined in a different class.) In this case the dispatch IDs to be used are usually dictated by the original implementation of the interface.

The dispatch ID is a positive, non-zero constant of format I4.

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 Clause

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.

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
```

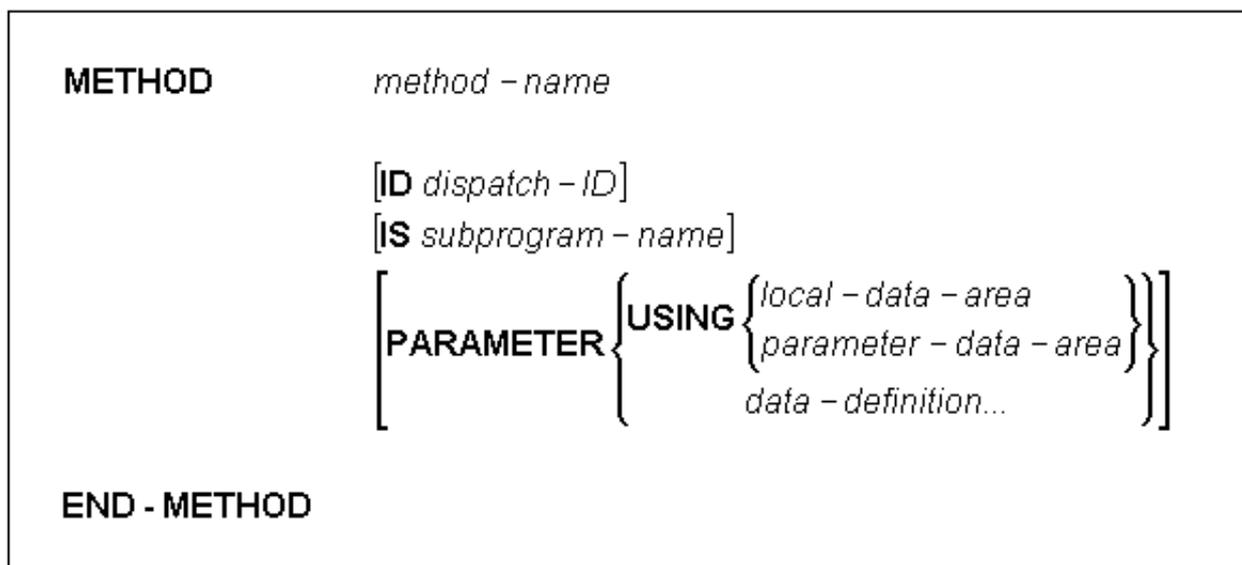
The following property definitions are not allowed:

```

/* 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 for user variables (please refer to the section Statement Usage Related Topics for further information). 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. Bolero for example uses the Java naming convention. So, a method that is planned to be used in a Bolero client should also respect the Java naming conventions.

ID Clause

Note:

This clause applies only on Windows platforms. It is only available with version 5.1.1 and above.

The ID clause is used to assign a specific numeric identifier to a method. This identifier (the so-called dispatch ID) is only relevant if the class is to be registered with DCOM.

Normally, Natural automatically assigns a dispatch ID to a method. It is only necessary to explicitly define a specific dispatch ID for a property if the property belongs to an interface with the EXTERNAL clause. (This is an interface that shall be implemented in this class, but which is originally defined in a different class.) In this case, the dispatch IDs to be used are usually dictated by the original implementation of the interface.

The dispatch ID is a positive, non-zero constant of format I4.

IS Clause

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 Clause

This specifies the parameters of the method, and has the same syntax as the PARAMETER clause of the DEFINE DATA statement. For further information on the DEFINE DATA statement, see the Natural Statements documentation.

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.

Parameters that are marked 'OPTIONAL' are available with Version 4.1.2 and all subsequent releases. Optional parameters need not to be specified when the method is called. They can be left unspecified by using the nX notation in the SEND METHOD statement.