INTERFACE

```
INTERFACE  interface-name
 [ EXTERNAL]
 [ID interface-GUID]
 [property-definition]
 [method-definition]
END-INTERFACE
```

Dieses Kapitel behandelt folgende Themen:

- Funktion
- Syntax-Beschreibung

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

Verwandte Statements: `CREATE OBJECT` | `DEFINE CLASS` | `INTERFACE` | `METHOD` | `PROPERTY` | `SEND METHOD`

Gehört zur Funktionsgruppe: *Komponentenbasierte Programmierung*

Funktion

Bei der komponentenbasierten Programmierung ist ein Interface (eine Schnittstelle) eine Sammlung von Methoden und Eigenschaften, die semantisch zusammengehören und eine bestimmte Funktion einer Klasse darstellen.

Sie können ein oder mehrere Interfaces für eine Klasse definieren. Durch die Definition mehrerer Interfaces wird es Ihnen ermöglicht, Methoden nach ihrer Funktionsweise zu strukturieren/gruppieren, z.B. stellen Sie alle Methoden, die mit Dauerhaftigkeit (Laden, Speichern, Aktualisieren) zu tun haben, in ein Interface und die anderen Methoden in andere Interfaces.

Das `INTERFACE`-Statement dient zur Definition eines Interface. Es darf nur innerhalb eines Natural-Klassenmoduls verwendet werden und kann wie folgt definiert werden:

- innerhalb eines `DEFINE CLASS`-Statements. Diese Form wird verwendet, wenn das Interface nur in einer Klasse implementiert werden soll.
- in innerhalb der `INTERFACE USING`-Klausel des `DEFINE CLASS`-Statements enthaltenem Copycode. Diese Form wird verwendet, wenn das Interface in mehr als einer Klasse implementiert werden soll.

Die mit dem Interface verbundenen Eigenschaften und Methoden werden in den *Property-Definitionen* bzw. *Method-Definitionen* festgelegt.

Syntax-Beschreibung

interface-name	**Interface-Name:** Dies ist der dem Interface zuzuweisende Name. Der Interface-Name kann maximal 32 Zeichen lang sein und muss den Natural-Namenskonventionen für Benutzervariablen entsprechen (weitere Informationen finden Sie im Abschnitt *Namenskonventionen für Benutzervariablen*. Er muss pro Klasse eindeutig und nicht mit dem Klassen-Namen identisch sein. Wenn das Interface von Clients verwendet werden soll, die in anderen Programmiersprachen geschrieben sind, sollte der Interface-Name so gewählt sein, dass er nicht gegen die für diese Sprachen geltenden Namenskonventionen verstößt.
EXTERNAL	**EXTERNAL-Klausel:** Mit der `EXTERNAL`-Klausel wird angegeben, dass dieses Interface von der Klasse implementiert wird, es ursprünglich aber in einer anderen Klasse definiert ist. Diese Klausel ist nur relevant, wenn die Klasse mit DCOM registriert werden soll. Interfaces mit der `EXTERNAL`-Klausel werden ignoriert, wenn die Klasse mit DCOM registriert wird. Es wird angenommen, dass das Interface von der Klasse registriert ist, die es ursprünglich definiert hat.
ID *interface-GUID*	**ID-Klausel:** Mit der `ID`-Klausel wird dem Interface eine global eindeutige ID (globally unique ID) zugewiesen. Die Interface-GUID ist der Name einer GUID, die in einer Data Area definiert ist, die von der `LOCAL`-Klausel einbezogen wird. Die Interface-GUID ist eine (benannte) Alpha-Konstante. Eine GUID muss einem Interface zugewiesen werden, wenn die Klasse mit DCOM registriert werden soll.
property-definition	**Property-Definition für das Interface:** Die *property-definition* dient zur Definition von Eigenschaften für das Interface. Siehe *Property-Definition* weiter unten.
method-definition	**Method-Definition für das Interface:** Die *method-definition* dient zur Definition einer Methode für das Interface. Siehe *Method-Definition* weiter unten.
END-INTERFACE	Das für Natural reservierte Wort `END–INTERFACE` muss zum Beenden des `INTERFACE`-Statements benutzt werden.

Property-Definition

Die Property-Definition dient zur Definition von Eigenschaften für das Interface.

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

Properties sind Attribute eines Objekts, das von Clients aufgerufen werden kann. Ein Objekt, das einen Angestellten darstellt, kann zum Beispiel eine Property mit Namen `Name` und eine andere mit Namen `Department` haben. Das Einlesen oder Ändern des Namens oder der Abteilung des Angestellten durch Aufruf der Property für dessen Namen oder dessen Abteilung ist viel einfacher für einen Client als eine Methode aufzurufen, die den Wert zurückgibt, und eine andere Methode aufzurufen, die den Wert ändert.

Jede Property benötigt eine Variable in der Object Data Area der Klasse, um dessen Wert zu speichern – dies wird als Objektdaten-Variable bezeichnet. Die Property-Definition dient dazu, diese Variable für den Zugriff von Clients freizugeben. Die Property-Definition legt den Namen und das Format der Property fest und verbindet sie mit der Objektdaten-Variable. Im einfachsten Fall übernimmt die Property den Namen und das Format der Objektdaten-Variable selbst. Es ist auch möglich, den Namen und das Format innerhalb bestimmter Grenzen zu überschreiben.

property-name	**Property-Name:** Dies ist der der Property zuzuweisende Name. Der Property-Name kann maximal bis zu 32 Zeichen enthalten und muss den Natural-Namenskonventionen für Benutzervariablen entsprechen (weitere Informationen finden Sie unter *Namenskonventionen für Benutzervariablen*). Wenn die Property von Clients verwendet werden soll, die in anderen Programmiersprachen geschrieben sind, sollte der Property-Name so gewählt sein, dass er nicht gegen die für diese Sprachen geltenden Namenskonventionen verstößt.

format-length/array-definition	**Property-Format:** Damit wird das Format der Property definiert, wie es von den Clients erkannt wird. Wenn *format-length/array-definition* weggelassen wird, wird *format-length* und *array-definition* aus der in der `IS`-Klausel zugewiesenen Objektdaten-Variable genommen. Wenn *format-length/array-definition* angegeben wird, muss diese Angabe unbedingt datenübertragungskompatibel sein, und zwar zu dem und von dem in *operand* in der `IS`-Klausel angegebenen Format der Objektdaten-Variable. Im Falle einer `READONLY`-Property braucht die Datenübertragungs-Kompatibilität ausschließlich in einer Richtung zu gelten: mit der Objektdaten-Variable als Ausgangsoperand und der Property als Zieloperand. Wenn eine Array-Definition angegeben wird, muss sie in den Dimensionen, Ausprägungen pro Dimension, Untergrenzen und Obergrenzen mit der Array-Definition der entsprechenden Objektdaten-Variable übereinstimmen. Dies kommt durch Spezifikation eines Sterns (*) für jede Dimension zum Ausdruck.
ID *dispatch-ID*	**ID-Klausel:** Mit der `ID`-Klausel wird einem Property ein bestimmter numerischer Bezeichner (Identifier) zugewiesen. Dieser Bezeichner (die sogenannnte Dispatch-ID) ist nur relevant, wenn die Klasse mit DCOM registriert werden soll. Normalerweise weist Natural einem Property automatisch eine Dispatch-ID zu. Es ist nur dann erforderlich, für ein Property eine bestimmte Dispatch-ID explizit zu definieren, wenn das Property zu einem Interface mit einer `EXTERNAL`-Klausel gehört. (Dies ist ein Interface, das in dieser Klasse implementiert werden soll, das jedoch ursprünglich in einer anderen Klasse definiert wurde.) In diesem Fall ist die zu benutzende Dispatch-ID in der Regel durch die ursprüngliche Implementierung des Interfaces vorgegeben. Die Dispatch-ID ist eine positive Konstante, die nicht Null ist, im Format I4.

READONLY	**Schreibschutz für Property-Wert:** Wenn dieses Schlüsselwort angegeben wird, kann der Wert der Property nur gelesen und nicht gesetzt werden. Das in *operand* in der IS-Klausel angegebene Format der Objektdaten-Variable muss datenübertragungskompatibel mit dem in *format-length/array-definition* angegebenen Format sein. Es muss nicht datenübertragungskompatibel in der umgekehrten Richtung sein. Wenn das Schlüsselwort READONLY weggelassen wird, kann der Property-Wert sowohl gelesen als auch gesetzt werden.
IS *operand*	**IS-Klausel:** Der *operand* in der IS-Klausel weist eine Objektdaten-Variable als Adresse zu, um den Property-Wert zu speichern. Die zugewiesene Objektdaten-Variable muss nicht unbedingt eine Gruppe sein. Die Variable wird in normaler Operanden-Syntax referenziert. Dies bedeutet, dass wenn die Objektdaten-Variable ein Array ist, sie mit Index-Notation referenziert werden muss. Nur die vollständige Indexbereichs-Notation und Sternchen- Notation ist zulässig. Die IS-Klausel sollte nicht verwendet werden, wenn das INTERFACE-Statement aus einem Copycode-Member übernommen und in mehreren Klassen wiederverwendet wird. Wenn Sie das INTERFACE-Statement nochmals verwenden möchten, müssen Sie die Objektdaten-Variable in einem PROPERTY-Statement außerhalb des INTERFACE-Statements zuweisen. Wenn die IS-Klausel weggelassen wird, wird die Property mit der Objektdaten-Variable mit demselben Namen wie die Property verknüpft. Wenn eine Variable mit diesem Namen nicht definiert ist, oder wenn es sich um eine Gruppe handelt, führt dies zu einem Syntax-Fehler.
END-PROPERTY	Das für Natural reservierte Wort END–PROPERTY muss zum Beenden des Interfaces PROPERTY-Definition benutzt werden.

Beispiele

Angenommen die Object Data Area enthält die folgenden Daten-Definitionen:

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

Dann sind die folgenden Property-Definitionen erlaubt:

```
property Salary
  end-property
  property Pay is Salary
  end-property
  property Pay(P7.2) is Salary
  end-property
  property Pay(N7.2) is Salary
```

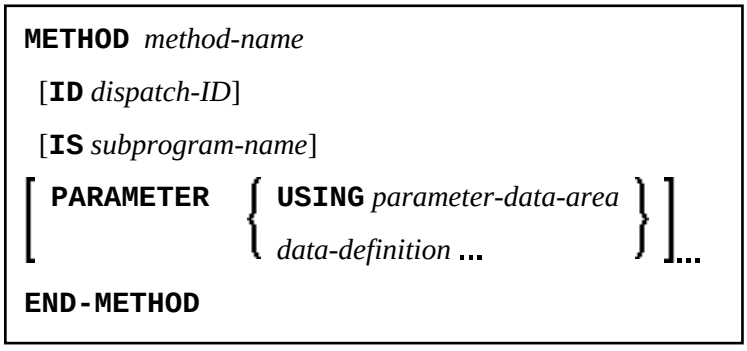
```
  end-property
  property SalaryHistory
  end-property
  property OldPay is SalaryHistory(*)
  end-property
  property OldPay is SalaryHistory(1:10)
  end-property
  property OldPay(P7.2/*) is SalaryHistory(1:10)
  end-property
  property OldPay(N7.2/*) is SalaryHistory(*)
  end-property
```

Die folgenden Property-Definitionen sind nicht zulässig:

```
/* 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

Die Method-Definition dient zur Definition einer Methode für das Interface.



Um das Interface in verschiedenen Klassen wiederverwenden zu können, übernehmen Sie die Interface-Definition aus einem Copycode und definieren Sie das Subprogramm hinter der Interface-Definition mit einem `METHOD`-Statement. Dann können Sie die Methode in verschiedenen Klassen anders implementieren.

method-name	**Method-Name:** Dies ist der der Methode zuzuweisende Name. Der Methoden-Name kann maximal bis zu 32 Zeichen enthalten und muss den Natural-Namenskonventionen für Benutzervariablen entsprechen (weitere Informationen entnehmen Sie dem Abschnitt *Namenskonventionen für Benutzervariablen*). Er muss pro Interface eindeutig sein. Wenn die Methode von Clients verwendet werden soll, die in anderen Programmiersprachen geschrieben sind, sollte der Methoden-Name so gewählt sein, dass er nicht gegen die für diese Sprachen geltenden Namenskonventionen verstößt.
ID *dispatch-ID*	**ID-Klausel:** Mit der `ID`-Klausel wird einer Methode ein bestimmter numerischer Bezeichner (Identifier) zugewiesen. Dieser Bezeichner (die sogenannnte Dispatch-ID) ist nur relevant, wenn die Klasse mit DCOM registriert werden soll. Normalerweise weist Natural einer Methode automatisch eine Dispatch-ID zu. Es ist nur dann erforderlich, für eine Methode eine bestimmte Dispatch-ID explizit zu definieren, wenn die Methode zu einem Interface mit einer `EXTERNAL`-Klausel gehört. (Dies ist ein Interface, das in dieser Klasse implementiert werden soll, das jedoch ursprünglich in einer anderen Klasse definiert wurde.) In diesem Fall ist die zu benutzende Dispatch-ID in der Regel durch die ursprüngliche Implementierung des Interfaces vorgegeben. Die Dispatch-ID ist eine positive Konstante, die nicht Null ist, im Format I4.
IS *subprogram-name*	**Name des Subprogramms:** Dies ist der Name des die Methode implementierenden Subprogramms. Der Name des Subprogramms besteht aus bis zu 8 Zeichen. Die Voreinstellung ist *method-name* (wenn die `IS`-Klausel nicht angegeben wird).

PARAMETER	**Parameter-Definition:** Mit dieser Klausel werden die Parameter der Methode angegeben; sie hat dieselbe Syntax wie die PARAMETER-Klausel vom DEFINE DATA-Statement. Die Parameter müssen mit den Parametern übereinstimmen, die später bei der Implementierung des Subprogramms verwendet werden. Dies wird am besten durch Verwendung einer Parameter Data Area (PDA) gewährleistet. In der Parameter Data Area als BY VALUE markierte Parameter sind Eingabe-Parameter der Methode. Nicht als BY VALUE markierte Parameter werden referenziert (By Reference) übergeben und sind Eingabe-/Ausgabe-Parameter. Dies ist die Voreinstellung. Der als BY VALUE RESULT markierte erste Parameter wird als Rückgabewert für die Methode zurückgegeben. Wenn mehr als ein Parameter so markiert ist, werden die anderen als Eingabe/Ausgabe-Parameter behandelt.
END-METHOD	Das für Natural reservierte Wort END–METHOD muss zum Beenden der METHOD-Definition für das Interface benutzt werden.